Click on any topic in the table below

Welcome to Panorama!

Tips for Using This Documentation	
Installation & Activation	
History of Panorama (what's new!)	
Table of Contents	Condensed
	Full
Step-by-Step Tutorials	Mailing List
	Checkbook
	Invoice
Panorama Reference	



Panorama 4.0 Tutorial & Handbook
Copyright © 2001, ProVUE Development,
All Rights Reserved

Table of Contents (Condensed)



– Click on any entry to jump to the page —

Table of Contents (Full) - 9

Welcome to Panorama! - 41

What is a Database? - 41, What is Panorama? - 43, Getting Started With Panorama For First Time Users - 64, Upgrading to Panorama V From an Earlier Version - 65, Deploying Applications Built With Panorama - 66

Tips for Using This Documentation - 71

Typographical Conventions - 72, Opening the Documentation - 72, Finding a Topic - 74, Cross Reference Links - 76, Using the Table of Contents - 77, Searching the Manual - 80, Display Options - 83

Installation & Activation - 89

Getting Organized - 89, Installing the Software - 89, Activating the Software - 94, Using Panorama's "Demo Mode" - 105

Step-by-Step Tutorials - 1

Lesson 1: Building Your First Mailing List Database - 2, Entering Data Into Your New Database - 4,
Making Corrections - 5, Editing a Multi-Line Cell - 6, Saving Your Work - 8, Importing Data Into the Mailing List - 10,
Adjusting Column Widths, Font and Size - 14, Sorting the Database - 17, Sorting By Two or More Fields - 18,
Finding a Person - 20, Finding Multiple People - 22, Selecting Instead Of Finding - 24, Using the Sounds Like Option - 25,
Making More Complex Selections - 26, Selecting All Records - 28, Adding A Database To Your List Of Favorites - 29,
Closing a Database - 31, Re-Opening a Favorite File - 32, Printing the Data Sheet - 39,
Creating a Form for Printing Mailing Labels - 42, Previewing the Labels - 47, Printing the Labels - 48,
Switching Between Forms and the Data Sheet - 48, Creating a Data Entry Form - 49, Creating Graphic Objects - 51, Creating Text Objects - 58, Creating Data Cell Objects - 62, Using Data Cells to Edit Data - 67,
Building the Data Entry Form - 69, Printing a Mail Merge Letter - 79, Tidying Up - 81,
Lesson 2: Building and Organizing a Checkbook - 82, Loading the Data - 85, Analyzing the Checkbook - 87, Selecting Data - 87, Calculating the Grand Total - 89, Calculating Subtotals - 90, Hiding and Showing Detail - 94,
Removing Summary Records - 101, Automating Tasks with Procedures - 104, Using the Action Menu - 106,
Editing a Procedure - 107, Analyzing Data with a Crosstab - 109, Auditing the Crosstab Table - 115,
Making a Bar Chart - 119, Data Entry Helpers - 130, Calculating the Checkbook Balance - 146,
Drawing a Check Form - 149, Making the Form Elastic - 164, **Lesson 3: Building an Invoice Database - 169**,
Line Item Fields - 171, Creating an Invoice Form - 172, Entering Data into the Invoice - 183,
Lesson 4: Building a Price List - 186, Loading the Data - 186, Linking the Price List to the Invoice - 189,
Using the Linked Invoice & Price List - 196, Creating a Procedure to Add an Item - 198,
Creating a Price List Form with Buttons - 199, Creating a Price List Form with Buttons - 199, Saving a File Set - 209

Guide to Wizards & Demo Files - 1

Wizard & Demo File Quick Reference - 2, Primary Wizards - 4, Calculation Wizards - 9, Developer Tool Wizards - 12, Documentation Wizards - 20, Form Tools Wizards - 25, Internet Wizards - 27, Mini Application Wizards - 42, Preferences Wizards - 54, Utilities Wizards - 70, Business Demo Files - 76

Chapter 1: Files and Memory - 107

Files, Icons and the Desktop - 107, Opening a Database - 108, The Favorite Databases Wizard - 109,
Advanced Database Opening Techniques - 118, Opening with Multiple Copies of Panorama on Your System - 119,
Creating a New Database - 121, Using the New Database Wizard - 122, Saving a Database - 131,
Saving Window Positions - 132, Revert to Saved - 133, Auto-Save - 133,
Viewing and Modifying Database Metadata - 135, Working with Multiple Databases - 137, File Sets - 138,
Appending One Database to Another - 141, Replacing Obsolete Data - 143,
Importing and Exporting Data - 144, Importing a Text File - 145, Importing HTML Tables - 150,
Using the Text Import Wizard - 157, Exporting a Text File - 165, Exporting with the Text Export Wizard - 169,
Exporting HTML Tables - 178, Monitoring Memory Usage - 186, Adjusting Panorama's Memory Allocation (Windows and OS X) - 189, Adjusting Panorama's Memory Allocation (Macintosh OS 9) - 190, Changing Scratch Memory Size (Macintosh OS 9) - 192

Chapter 2: Windows - 195

Window Components - 195, Tool Palette - 196, Close Box - 197, Drag Bar - 197, Title - 197, Zoom Box (Maximize) - 197, Grow Box - 198, Scroll Bars - 198, Splitting a Window - 198, Info Palette - 199, Bringing a Window to the Front - 200, Hiding Windows - 201, Zooming Into a Box - 201, Saving Window Positions - 202, Saving with No Windows - 202
Turning Window Components On and Off (Window Tweak Wizard) - 203, Measuring a Window (Window Size Wizard) - 205, Arranging All Open Windows at Once (Tiling and Stacking) - 209

Chapter 3: Views - 217

Types of Panorama Views - 217, Data Sheet and Form Views - 218, Other Views - 219, Switching Between Views - 222, Opening More Than One Window Per Database - 223, Window Options - 226, The View Wizard - 227,
Form Modes: Data Access vs. Graphic Design - 234, Form Operation: Individual Pages vs.View-As-List - 235,
Creating a New Form, Crosstab or Procedure - 236, Renaming a Form, Crosstab or Procedure - 237,
Deleting a Form, Crosstab or Procedure - 237, Changing the Order of Forms, Crosstabs or Procedures - 237,
The Privilege Dialog - 238

Chapter 4: Records - 241

Data Organization - 241, Tables vs. Individual Pages - 242, Special Records - 242, Summary Records - 243,
Invisible Records - 244

Chapter 5: Fields - 247

The Setup Menu - 248, Add Field - 249, Field Properties - 250, Delete Field - 250, Changing the Width of a Field - 251, The Design Sheet - 252, Database "Generations" - 252, Field Properties - 255, Adding New Fields Using the Design Sheet - 256, Removing Fields Using the Design Sheet - 257, Making a Copy of a Field - 257, Re-Arranging Fields - 259, Rules for Field Names - 260, Repeating Fields (Line Items) - 262, "Generic" Fields - 270, Adding Generic Fields to an Existing Database - 271, Using Generic Fields with the VCard Wizard - 276

Chapter 6: Data Types - 285

Data Types and Memory Usage - 286, Setting Up a Field's Data Type - 286, Numeric Data - 289,
Numeric Output Patterns - 290, Dates - 294, Date Output Patterns - 295, Choices - 298

Chapter 7: Data Entry & Editing - 303

Editing Records - 303, Moving From Record to Record - 303, Moving from Field to Field - 305,
Adding a New Record - 306, Inserting a New Record - 306, Deleting a Record - 307, Deleting Multiple Records - 308,
Delete All - 308, Duplicating a Record - 309, Moving a Record - 309, Editing Data Within a Cell - 310,
Expanding the Input Box - 311, Editing Cells Within a Form - 313, Tab Down - 315, Tab Order in Forms - 315,
Tabbing with the Space Bar - 317, Data Entry Accelerators - 318, Automatic Capitalization - 319,
Checking for Duplicate Data - 320, Clairvoyance® - 321, Clairrows - 327, Input Patterns - 327, Restricting Character Types - 330, Default Values - 333, Creating a Unique Record Number - 337, Automatic Time/Date Stamping - 338,
Automatic Calculations - 340, Automatically Triggering a Procedure - 350, The Choice Palette - 353,
Searching for Text Within the Input Box - 355, Replacing Words or Phrases Within a Cell - 356,
Using the Spelling Checker within a Cell - 357

Chapter 8: Sorting - 359

Basic Sorting - 359, Sorting By More Than One Field - 360, Sorting Numbers and Dates - 364, Sorting Within Groups - 365, Sorting Choices - 365

Chapter 9: Searching and Selecting - 367

Finding vs. Selecting - 367, The Find/Select Dialog - 369, Locating Dates by Month, Quarter, or Year - 373,
Find and Find Next - 374, Select - 375, Multiple Find/Select Criteria - 376, Select Reverse - 377, Undo Select - 378,
Permanently Removing Unselected Data - 378, The Live Clairvoyance™ Wizard - 379, Formula Find/Select - 387,
Select Duplicates - 388

Chapter 10: Summaries and Outlines - 393

3-Step Summarizing - 393, STEP 1 - GROUP - 399, Subgroups - 399, Grand Total - 399,
Grouping by Week, Month, Quarter, or Year - 400, Manually Creating and Removing Summary Records - 402,
STEP 2 - CALCULATE - 403, Total - 403, Count - 403, Average - 403, Minimum - 403, Maximum - 403,
Recalculating Summaries - 404, Running Total - 404, Running Difference - 406, STEP 3 - OUTLINE - 409,
The Outline Level - 409, Collapsing vs. Selecting - 413, Expanding and Collapsing Specific Details - 413,
Sorting by Summary Value - 417, Sorting Within Groups - 421, Getting Rid of Summary Records - 421,
Getting Rid of Detail - 421, The Summaries & Outlines Wizard - 423, The Mini Statistics Wizard - 428

Chapter 11: Crosstabs - 433

Category and Tabulation Fields - 435, Creating and Setting Up a New Crosstab View - 436,
Crosstabs by Day, Month, Quarter or Year - 440, Adjusting Crosstab Column Widths - 441, Selecting Original Data - 441,
Crosstabs Based On Selected Data - 445, Crosstabs Containing Outlines - 445, Sorting a Crosstab - 447,
Removing and Renaming Crosstab Tables - 448, Exporting a Crosstab Table - 448,

Chapter 12: Data Processing - 451

Transforming Selected Data - 451, Filling a Field with a Fixed Value - 452, Filling a Field with a Formula - 453,
Filling Empty Cells - 463, Automatic Numbering - 464, Propagate - 465, UnPropagate - 469,
Using UnPropagate to Eliminate Duplicates - 470, Change (Find and Replace) - 472, Data Style and Color - 474

Chapter 13: Introduction to Forms - 481

Opening a Form - 482, Opening A Form in a New Window - 483, Form Modes: Data Access vs. Graphic Design - 485,
Form Operation: Individual Pages vs.View-As-List - 486, Creating a New Form - 487, Renaming a Form - 488,
Deleting a Form - 488, Browsing the Database With a Form - 488,
Browsing the Database With a View-As-List Form - 489

Chapter 14: Graphic Design - 491

Graphic Objects - 491, Creating a Graphic Object - 494, Customizing the Tool Palette - 497, SuperObjects - 501, Modifying Objects - 501, The Graphic Control Strip - 505, Rulers - 506, Moving a Single Object - 508, Changing the Size of a Single Object - 513, Removing Objects - 519, Solid, Outline and Hollow Objects - 520, Fill Pattern - 521, Line Pattern - 523, Line Width - 525, Color - 526, Font - 529, Text Size - 531, Text Style - 532, Object Type/Object Name - 533, The Object Properties Dialog - 534, Grouping Objects Together - 536, Cluster Resize - 541, Aligning Objects - 553, Adjusting Spacing Between Multiple Objects - 556, Duplicating Objects - 560, Cut, Copy, and Paste - 566, Overlapping Objects - 568, Locked Objects - 575, Alignment Grid - 577, Magnification and Reduction - 579, Form Background Colors - 582, Using the Form Explorer Wizard - 583

Chapter 15: Displaying and Editing Text - 587

Displaying Text - 587, Fixed Text Objects - 587, Text Font, Size and Style - 593, Text Alignment - 594, Displaying Data in Auto-Wrap Text - 595, Displaying Formulas in Auto-Wrap Text - 602, Text Display SuperObjects™ - 608, Using Formulas to Display Text - 621, Editing Text - 632, Working with Data Cell Objects - 635, Text Editor SuperObject - 639, Automatically Creating Rows or Columns of Data Cells or Text Editor SuperObjects - 662, Tab Order in Forms - 670, Word Processor SuperObject - 673, Using the Word Processor - 677, The Ruler - 680, Margins (Indents) - 682, Tab Stops - 685, Alignment - 690, Line Spacing - 691, Styles - 692, Selecting Text - 694, Configuring the Word Processor - 696, Printing Word Processor Documents - 719, Using the Mini Correspondence Wizard - 727

Chapter 16: Images & Movies - 739

Fixed Images - 739, Displaying and Printing EPS Images - 741, Flash Art™ - 748, The Flash Art Scrapbook (Gallery) - 762, Displaying Images Directly From Disk Files - 767, Displaying Non PICT Images (Enhanced Image Pack) - 773, Super Flash Art™ Options - 777, “Classic” Flash Art Objects - 806, Storing Images in a Field - 807, Displaying Movies in a Form - 810

Chapter 17: Buttons & Widgets - 813

Push Buttons - 813, Super Object Push Button - 813, “Classic” Push Buttons - 820, Flash Art™ Push Button SuperObjects™ - 823, Data Buttons - 827, Data Button SuperObjects™ - 828, Flash Art Data Button SuperObjects™ - 842, Sticky Push Button SuperObjects™ - 845, “Classic” Checkbox and Radio Buttons - 846, Pop-Up Menus - 850, Pop-Up Menu SuperObjects™ - 850, “Classic” Pop-Up Buttons - 861, Creating a Pop-Up Menu with a Procedure - 864, List SuperObjects - 869, Building the List - 884

Chapter 18: Form Goodies - 889

View-As-List Forms - 889, Elastic Forms - 912, Super Matrix Objects - 929, Building a Calendar - 960, Scroll Bars - 968, Balloon Help - 974, Displaying Balloon Help Text Directly on the Form - 981, Changing the Cursor Shape Over Different Areas - 980

Chapter 19: Charts - 983

Chart Data - 983, Creating a New Chart - 984, Bar Charts - 991, Line Charts - 993, Area Charts - 993, Pie Charts - 994, Scatter Diagrams - 995, Preparing the Database for Drawing a Chart - 996, Dressing Up Chart Appearance - 1004, Chart Font, Size, and Style - 1004, Vertical Legends - 1005, Output Patterns - 1006, Grid - 1015, Non-Zero Axis OK - 1016, Tick Mark Spacing - 1017, Chart Preview - 1018, Printing a Chart - 1033

Chapter 20: Printing Basics - 1037

Printing Different Views - 1037, Printing the Data Sheet - 1037, Printing Data Sheet Headers & Footers - 1038, Printing a Form - 1041, Preparing Data For Printing - 1042, The Page Setup Dialog - 1043, The Print Dialog - 1045, Print Preview - 1046, Print One Record - 1048

Chapter 21: Custom Reports - 1049

Working with Tiles - 1050, Tiles In Action - 1064, Margins - 1073, Headers and Footers - 1079, Page Numbers - 1088, Printing the Current Date and Time - 1091, The QuickReport Dialog - 1099, Printing Multiple Page Records - 1102, Printing Data that Overflows a Page - 1104, Variable Height Records - 1111, Printing Multiple Column Reports - 1120, Printing Summary Information - 1129, Printing Data Grouped by Month, Quarter or Year - 1136, Group Headers - 1139, Group Sidebars - 1143, Keeping a Group Together on a Column or Page - 1148, Even and Odd Page Layout - 1152, Special Paper Options - 1154

Chapter 22: Labels - 1157

Label Fundamentals - 1157, The QuickLabel Dialog - 1157, Printing Labels on Sheets - 1160, Printing 3 by 10 1” Labels (Avery 5160) - 1161, Aligning Labels on the Sheet - 1161, Printing Roll Labels - 1161

Chapter 23: Formulas - 1165

Formulas In Action - 1165, Displaying/Printing A Formula - 1166, Storing Formula Results in the Database - 1168, Using a Formula to Locate/Select Information - 1171, Using the Formula Wizard - 1175, Formula Components - 1188, Values - 1195, Constants - 1195, Fields - 1196, Variables - 1199, Special Characters - 1202, Working With Extremely Complex Formulas - 1204, Arithmetic Formulas - 1205, Text Formulas - 1212, Characters and ASCII Values - 1230, The ASCII Chart Wizard - 1232, Text Arrays - 1236, HTML Generating Functions - 1247, Time Arithmetic - 1255,SuperDates (combined date and time) - 1259, Reminders - 1260, True/False Formulas - 1265, Linking With Another Database - 1272, Zip Code Lookup - 1286, Graphic Co-Ordinates - 1287, Colors - 1294, Raw Binary Data - 1296, Disk Files and Folders - 1304, Import/Export Functions - 1312, System and Database Information Functions - 1314, Custom Functions - 1333

Chapter 24: Procedures - 1339

Introduction to (Panorama) Programming - 1339, Procedures - 1340, Statements - 1340, Creating a Procedure with the Recorder - 1348, Writing a Procedure from Scratch - 1352, Programming Reference Wizard - 1361, Assignment Statements - 1368, Variables - 1370, Control Flow - 1378, True/False Formulas - 1378, IF Statements - 1380, CASE Statements - 1382, LOOP Statements - 1382, Subroutines - 1384, Jumping to an Another Location in the Program - 1396, Stopping the Program - 1397, Building Subroutines On The Fly (The Execute Statement) - 1399, Custom Statements - 1408, Program Formatting - 1421, Organizing Large Procedures (The Mark Menu) - 1425, Suppressing Display of Text and Graphics - 1427, Debugging a Procedure - 1431, The Panorama Interactive Debugger - 1434, Procedure Debug Log - 1444, Cross Referencing - 1452, The Action Menu - 1458, Live Menus - 1465, Custom Menus (Resource Based) - 1478, Hidden Triggers - 1507

Chapter 25: Programming Techniques - 1527

Accessing Files - 1527, Files and Folders - 1527, Locating a File with Standard Dialogs - 1530, Opening a Panorama Database - 1533, Importing Text Files - 1537, Exporting Text Files - 1541, Smart Merge Synchronization - 1545, Directly Reading and Writing Disk Files - 1549, Working with Resources - 1560, Accessing the Windows Registry - 1568, Monitoring Memory Usage - 1571, Windows - 1572, Specifying the New Window Location - 1573, Temporary “Invisible” Windows - 1581, Window Clones - 1584, Alerts - 1591, Dialogs - 1607, Custom Dialogs - 1616, Preparing a Form for Use as a Dialog - 1616, Accessing and Modifying the Database Structure (Fields) - 1632, Database Navigation and Editing - 1637, Moving Data Between Files - 1659, Posting Data to Other Databases - 1673, Sorting - 1676, Locating Information - 1677, Live Clairvoyance™ - 1684, Summaries and Outlines - 1691, Transforming Big Chunks of Data - 1697, Processing/Transforming an Entire Array - 1716, Processing/Transforming Binary Data - 1723, Accessing the Internet - 1732, Programming Graphic Objects on the Fly - 1753, Drag and Drop - 1770, Program Control of SuperObjects™ - 1786, The Active SuperObject - 1787, Accessing and Modifying a SuperObject’s Internal Data - 1789, Text Editor SuperObject Commands - 1790, Word Processor SuperObject Commands - 1798, Super Flash Art Commands (Including Movie Control) - 1810, List SuperObject™ Commands - 1827, Auto Grow SuperObject™ Commands (Elastic Forms) - 1833, Super Matrix SuperObject™ Commands - 1834, Speech Synthesis - 1838, Printing - 1841, Writing Your Own Channel Modules - 1849

Panorama Reference - 5000

Introduction: - 5000, Online Reference: - 5000, A - 5010, B - 5070, C - 5080, D - 5141, E - 5184, F - 5214, G - 5285, H - 5338, I - 5348, K - 5461, L - 5465, M - 5517, N - 5535, O - 5555, P - 5588, Q - 5623, R - 5625, S - 5689, T - 5826, U - 5866, V - 5885, W - 5888, X - 5909, Y - 5912, Z - 5915

History of Panorama - 1

Version 5.0.0 - 1, Version 4.0.2 - 70, Version 4.0.1 - 76, Version 4.0 - 84, Version 3.1 - 101,
Version 3.0 - 104, Version 2.1 - 106, Version 2.0 - 106, Version 1.0, 1.1 and 1.5 - 106,
General Corporate History - 106

Additional Resources - 1877

On-Line Resources - 1877, Technical Support - 1881, Panorama Conferences - 1882, Publications - 1889,
Consulting Services - 1889

Copyright © 2004 ProVUE Development
All Rights Reserved

Table of Contents (Full)



– Click on any entry to jump to the page —

..... 1
..... 1
..... 1
..... 1
..... 1
..... 1
..... 1
..... 2

Table of Contents (Condensed) 3

Table of Contents (Full) 9

Welcome to Panorama! 41

What is a Database? 41
A Brief History of Database Technology 42
What is Panorama? 43
- Wizards 44
- Super Fast Searching and Sorting 45
- Phonetic Searching 45
- Easy Set-Up 46
- Crosstabs 47
- Data Outlines 47
- Data Entry Shortcuts 48
- Smart Dates™ 48
- Smart Program Recorder 48
- Formulas 48
- Compact Storage 49
- Charts 49
- Relational Links 49
- Internet Content 50
- Map, Phone, Email and VCard Support 50
- Sending Email 51
- Favorite Database Wizard 51
- Advanced Graphic Tools 52
- Mail Merge and Labels 53

Database Publishing 54
View-As-List Forms 55
Elastic Forms 56
Matrix Display 57
Images and Movies 58
High Speed Import 59
Flexible Text Export (including HTML) 59
Data Transformation 60
Select/Remove Duplicates 60
Client/Server 60
Complete Programming Language and Development Tools 61
Custom Menus, Buttons, and Dialogs 62
Seamless Cross Platform Operation 63
Getting Started With Panorama For First Time Users 64
Upgrading to Panorama V From an Earlier Version 65
Switching From Macintosh to Windows 66
Deploying Applications Built With Panorama 66
Panorama Direct 66
Panorama Engine 67
The Panorama Engine Licensing Process 67
Distributing Your Databases 67
Panorama Engine Restrictions 67
New Database Versions 68
License Fees (Commercial) 68
License Fees (Shareware/Freeware) 69
Panorama Engine vs. Panorama Direct 69

Tips for Using This Documentation 71

Why So Many Pages? 71
Printing This Book 71
Training Movies 71
Typographical Conventions 72
Opening the Documentation 72
Finding a Topic 74
Cross Reference Links 76
Using the Table of Contents 77
Searching the Manual 80
What About the Find Command? 82
Display Options 83
Smooth Text & Images Option 83
Single Page vs. Continuous Option 84
Magnification 86
Thumbnails 87

Installation & Activation 89

Getting Organized 89
Installing the Software 89
The Main Installer Window 90
Installation Package 91
Selecting the Installation Location 92
Installing the Software 93
Activating the Software 94
Moving Your Software to Another Computer (Deactivating Your Software) 97

Using Panorama With a Personal Use License 101
Setting Up and Using a Personal Use Password 101
What To Do If You Forget Your Password 104
Using Panorama's "Demo Mode" 105

Files and Memory 107

Files, Icons and the Desktop 107
Opening a Database 108
Databases and RAM 109
The **Favorite Databases** Wizard 109
Searching for a Favorite Database 111
Adding New Favorites 112
Removing a Favorite 113
Additional File Information 114
The Favorites Menu 116
The Show Menu 117
Advanced Database Opening Techniques 118
Opening a Database With Incorrect Type/Creator Information 118
Opening a Database Without Normal Initialization 119
Opening a Database with No Windows 119
Opening with Multiple Copies of Panorama on Your System 119
Creating a New Database 121
Using the **New Database** Wizard 122
Creating a Database with the Wizard 123
Creating Numeric and Date Fields 124
Default Values 124
Automatic Calculations 125
Line Items (Repeating Fields) 126
Starting with a New Database Template 128
Creating a Database from a Text File 129
Closing Panorama 131
Saving a Database 131
Saving Window Positions 132
Revert to Saved 133
Auto-Save 133
Pitfalls of Auto-Save 133
Backup Files 133
Opening Backup Files 134
On the Importance of Backing Up 134
Viewing and Modifying Database Metadata 135
Using Metadata Information 136
Working with Multiple Databases 137
Opening Multiple Files 137
File Sets 138
The AutoLoad File Set 140
Saving Multiple Files 141
Appending One Database to Another 141
Appending an Open Database 142
Appending Imported Data 142
Replacing Obsolete Data 143
Importing and Exporting Data 144
Working with Text Files 144
Importing a Text File 145

Importing into an Existing Database ... 149
Importing HTML Tables ... 150
Importing OverVUE Files ... 156
Re-Arranging Imported Data ... 156
Using the Text Import Wizard ... 157
Common Import Formulas ... 161
Import Templates ... 162
Choosing a Database to Import Into ... 164
Converting an Import Configuration into a Procedure ... 164
Exporting a Text File ... 165
Exporting with the **Text Export Wizard** ... **169**
Editing the Export Configuration ... 173
Common Export Formulas ... 175
Export Templates ... 176
Choosing a Database to Export From ... 177
Exporting HTML Tables ... 178
Using the Generated HTML Page ... 179
HTML Table Options ... 180
Monitoring Memory Usage ... 186
Memory Usage Details ... 187
Multiple Memory Statistic Windows ... 188
Adjusting Panorama's Memory Allocation (Windows and OS X) ... 189
Adjusting Panorama's Memory Allocation (Macintosh OS 9) ... 190
Changing Scratch Memory Size (Macintosh OS 9) ... 192

Windows ... 195

Window Components ... 195
Tool Palette ... 196
Scrolling the Tool Palette ... 197
Close Box ... 197
Drag Bar ... 197
Title ... 197
Zoom Box (Maximize) ... 197
Grow Box ... 198
Scroll Bars ... 198
Splitting a Window ... 198
Info Palette ... 199
Bringing a Window to the Front ... 200
Hiding Windows ... 201
Zooming Into a Box ... 201
Saving Window Positions ... 202
Saving with No Windows ... 202
Turning Window Components On and Off (**Window Tweak** Wizard) ... 203
Measuring a Window (**Window Size** Wizard) ... 205
Setting Exact Window Dimensions ... 206
Arranging All Open Windows at Once (Tiling and Stacking) ... 209
Saving and Restoring Window Positions ... 212
Choosing Tile Configurations ... 212
Bringing Windows to the Front ... 215

Views ... 217

Types of Panorama Views ... 217
Data Sheet and Form Views ... 218

Other Views 219
Switching Between Views 222
Opening More Than One Window Per Database 223
Window Options 226
The View Wizard 227
View Wizard Window Size and Options 231
Searching All Procedures 232
Form Modes: Data Access vs. Graphic Design 234
Form Operation: Individual Pages vs.View-As-List 235
Creating a New Form, Crosstab or Procedure 236
Renaming a Form, Crosstab or Procedure 237
Deleting a Form, Crosstab or Procedure 237
Changing the Order of Forms, Crosstabs or Procedures 237
The Privilege Dialog 238
User Levels vs. Save Window Positions 239
Hiding Sensitive Data 239

Records 241

Data Organization 241
Tables vs. Individual Pages 242
Special Records 242
Data Records 242
Summary Records 243
Invisible Records 244

Fields 247

The Setup Menu 248
Add Field 249
Field Properties 250
Delete Field 250
Changing the Width of a Field 251
The Design Sheet 252
Database “Generations” 252
Typical Design Sheet Operation 253
Field Properties 255
Adding New Fields Using the Design Sheet 256
Removing Fields Using the Design Sheet 257
Making a Copy of a Field 257
Re-Arranging Fields 259
Rules for Field Names 260
Multiple Line Field Names 260
Repeating Fields (Line Items) 262
Creating Line Item Fields 263
Modifying Line Item Fields 265
Adding More Line Item Fields 268
Learn More About Line Items 269
“Generic” Fields 270
Standard Generic Fields 270
Adding Generic Fields to an Existing Database 271
Assigning Multiple Database Fields to a Generic Field 273
Configuring Generic Phone Number Fields 273
Using Generic Fields with the VCard Wizard 276
Dragging Data to Other Applications or Databases 277

Dragging Data from other Applications or Databases 279
Displaying a Map 281
Sending an Email 281
Dialing the Phone 281
Exporting a Batch of VCards 282
Importing a Batch of VCards 283

Data Types 285

Data Types and Memory Usage 286
Setting Up a Field's Data Type 286
Data Type Conversion Problems 287
Numeric Data 289
Money 290
Numeric Output Patterns 290
Fixed Decimal Point Patterns 292
Numbers with Commas, Punctuation, and Measurement Units 292
Scientific Notation 292
Special Patterns for Negative Numbers 293
Leading Zeros 293
Numbers with Multiple Components 293
Phone Numbers 293
Plural Suffixes 294
Displaying Numbers as Words 294
Dates 294
Entering Dates 294
Default Year and Century 295
Date Output Patterns 295
Date Pattern Components 296
Common Date Output Patterns 297
Choices 298
Choice Data Entry (Choice Palette) 298
Creating the List of Choices 299
Exceptions 299
Generating a List of Choices Automatically 300
Updating the Choice List 301
Using Math Operations with Choices 301
Sorting Choices 301

Data Entry & Editing 303

Editing Records 303
Moving From Record to Record 303
Moving from Field to Field 305
Adding a New Record 306
Inserting a New Record 306
Deleting a Record 307
Deleting Multiple Records 308
Delete All 308
Duplicating a Record 309
The Clipboard Window 309
Moving a Record 309
Editing Data Within a Cell 310
The Input Box 310
Expanding the Input Box 311

Expanding a Right Justified Input Box ... 312
Editing Cells Within a Form ... 313
Tabbing from Cell to Cell ... 314
Tab Down ... 315
Tab Order in Forms ... 315
Tabbing with the Space Bar ... 317
Data Entry Accelerators ... 318
Automatic Capitalization ... 319
Changing Capitalization of Existing Data ... 319
Checking for Duplicate Data ... 320
Checking for Duplicates in Existing Data ... 321
Clairvoyance® ... 321
How Clairvoyance® Works ... 322
Turning Clairvoyance® On or Off ... 323
Clairvoyance® Helps Insure Data Consistency ... 323
Using Clairvoyance® With Dates ... 323
Clairvoyance® Across Multiple Files ... 323
Clairrows ... 327
Input Patterns ... 327
Entering Data with an Input Pattern ... 329
Using Input Patterns with Dates ... 329
Restricting Character Types ... 330
Custom Character Restrictions ... 331
Default Values ... 333
Default to Today's Date ... 334
"Ditto" Defaults Based on the Previous Record ... 335
Automatically Incrementing Defaults (1, 2, 3, …) Based on the Previous Record ... 336
Creating a Unique Record Number ... 337
Manually Changing the Record Number Counter ... 338
Automatic Time/Date Stamping ... 338
Automatic Calculations ... 340
Spreadsheet Mode Calculations ... 340
Procedure Mode Calculations ... 347
Automatically Triggering a Procedure ... 350
Pros and Cons of Spreadsheet vs. Procedure Mode ... 351
The Run Automatic Calculations Wizard ... 352
The Choice Palette ... 353
Changing the Shape of the Choice Palette ... 353
Creating the List of Choices ... 353
Exceptions ... 354
The Choice Palette vs. the Choices Data Type ... 354
Editing Tools within a Data Cell ... 354
Searching for Text Within the Input Box ... 355
Replacing Words or Phrases Within a Cell ... 356
Using the Spelling Checker within a Cell ... 357

Sorting ... 359

Basic Sorting ... 359
Sorting By More Than One Field ... 360
Sorting By Color ... 363
Undo Sorting ... 364
Sorting Numbers and Dates ... 364
Sorting Right Justified Text ... 365

Sorting Selected Data ... 365
Sorting Within Groups ... 365
Sorting Choices ... 365

Searching and Selecting ... 367

Finding vs. Selecting ... 367
The Find/Select Dialog ... 369
Locating Dates by Month, Quarter, or Year ... 373
Find and Find Next ... 374
Select ... 375
Multiple Find/Select Criteria ... 376
Select Within ... 377
Select Additional ... 377
Select Reverse ... 377
Undo Select ... 378
Permanently Removing Unselected Data ... 378
The Live Clairvoyance™ Wizard ... 379
Customizing the Search and Display Fields ... 380
Advanced Customization with Formulas ... 383
Changing the Target Database ... 383
The Search All Fields Wizard ... 384
Selecting From All Fields ... 386
Searching All Fields In Another Database ... 386
Formula Find/Select ... 387
The SEQ Function ... 387
The Select Summaries Command ... 388
Select Duplicates ... 388
Select Duplicates Using a Formula ... 389

Summaries and Outlines ... 393

3-Step Summarizing ... 393
STEP 1 - GROUP ... 399
Subgroups ... 399
Grand Total ... 399
The Group Command ... 400
Grouping by Week, Month, Quarter, or Year ... 400
Group by Color ... 401
Propagating Data into Summary Records ... 401
Manually Creating and Removing Summary Records ... 402
STEP 2 - CALCULATE ... 403
Total ... 403
Count ... 403
Average ... 403
Minimum ... 403
Maximum ... 403
Recalculating Summaries ... 404
Running Total ... 404
Using Running Total to Balance a Checkbook ... 404
Running Difference ... 406
Using Running Difference to Calculate Gas Mileage ... 407
STEP 3 - OUTLINE ... 409
The Outline Level ... 409
Collapsing vs. Selecting ... 413

Expanding and Collapsing Specific Details 413
Sorting by Summary Value 417
Sorting Within Groups 421
Getting Rid of Summary Records 421
Getting Rid of Detail 421
Printing Reports with Summary Information 422
The Summaries & Outlines Wizard 423
Using Summary/Outline Templates 425
Converting a Template into a Procedure 426
Printing the Summary Results 427
The Mini Statistics Wizard 428
Saving a Statistical Snapshot 431
Renaming and Deleting Snapshots 432
Printing a Statistical Analysis 432

Crosstabs 433

Category and Tabulation Fields 435
Creating and Setting Up a New Crosstab View 436
Crosstabs by Day, Month, Quarter or Year 440
Changing the Crosstab Design 440
Re-Calculating a Crosstab 441
Adjusting Crosstab Column Widths 441
Crosstab Font and Size 441
Selecting Original Data 441
Crosstabs Based On Selected Data 445
Crosstabs Containing Outlines 445
Sorting a Crosstab 447
Removing and Renaming Crosstab Tables 448
Exporting a Crosstab Table 448

Data Processing 451

Transforming Selected Data 451
Filling a Field with a Fixed Value 452
Filling a Field with a Formula 453
Numeric Calculations With Formula Fill 454
Using Formula Fill to Transform Characters 457
Date Calculations with Formula Fill 459
The SEQ Function 461
Filling Empty Cells 463
Automatic Numbering 464
Propagate 465
UnPropagate 469
Using UnPropagate to Eliminate Duplicates 470
Change (Find and Replace) 472
Changing with the Replace(Function 473
Data Style and Color 474
Displaying Data Style and Color in Forms 479
Accessing Style and Color in a Formula 479

Introduction to Forms 481

Opening a Form 482
Opening A Form in a New Window 483

Form Modes: Data Access vs. Graphic Design 485
Form Operation: Individual Pages vs.View-As-List 486
Creating a New Form 487
Renaming a Form 488
Deleting a Form 488
Browsing the Database With a Form 488
Browsing the Database With a View-As-List Form 489

Graphic Design 491

Graphic Objects 491
Types of Graphic Objects 491
Creating a Graphic Object 494
Creating Perfect Squares, Circles and Lines 496
Customizing the Tool Palette 497
Using the Keyboard to Select Common Tools 500
SuperObjects 501
Modifying Objects 501
Selecting a Single Object 501
Selecting Multiple Objects at Once 502
The Graphic Control Strip 505
Rulers 506
Moving a Single Object 508
Nudging an Object (or Objects) 509
Nudge “Auto Guides” 510
Viewing and Setting Exact Object Dimensions 512
Changing the Size of a Single Object 513
Nudging the Size of an Object 513
Nudge Size “Auto Guides” 514
Nudging to the Crosshair Cursor 515
Percentage Scaling 516
Resizing Without Handles 516
Changing the Radius of Round Corners 518
Removing Objects 519
Modifying Object Attributes 519
Solid, Outline and Hollow Objects 520
Fill Pattern 521
Line Pattern 523
Line Width 525
Color 526
Copying and Pasting Colors 528
Font 529
Maintaining Fonts across Multiple Computers and Platforms 529
Universal Fonts 530
Text Size 531
Text Style 532
Object Type/Object Name 533
The Object Properties Dialog 534
Working With Multiple Objects 536
Grouping Objects Together 536
Moving Multiple Objects 538
Fast Drag 538
Resizing Multiple Objects 540
Cluster Resize 541

Cluster Resize Troubleshooting 550
Setting Exact Dimensions of Multiple Objects 550
Aligning Objects 553
Adjusting Spacing Between Multiple Objects 556
Duplicating Objects 560
Duplicate 561
Drag Duplicating 561
Step and Repeat 562
Cut, Copy, and Paste 566
Copying Objects Between Forms 566
Copying Objects Between Files 566
Copying an Entire Form 566
Overlapping Objects 568
Changing the Stacking Order 569
Selecting a Completely Hidden Object 570
Making a Drop Shadow 573
Locked Objects 575
Ignoring Locked Objects 577
Alignment Grid 577
Magnification and Reduction 579
A Note About Measurement Accuracy 581
Form Background Colors 582
Using the Form Explorer Wizard 583

Displaying and Editing Text 587

Displaying Text 587
Fixed Text Objects 587
Editing Fixed Text 590
Moving and Resizing Fixed Text Objects 590
Text Font, Size and Style 593
Creating Reverse Type (White on Black) 593
Text Alignment 594
Displaying Data in Auto-Wrap Text 595
Data Merge Pop-Up Menu 596
Using Data Merge to Create Address Labels 597
Displaying Formulas in Auto-Wrap Text 602
The Build Formula Dialog 604
Text Display SuperObjects™ 608
Creating and Modifying Text Display SuperObjects 608
Text Display Options 611
Controlling Text Display Color and Style on the Fly 619
Using Formulas to Display Text 621
Combining Multiple Text Items Into One 621
Creating a Smart Formula 624
Eliminating Unnecessary Punctuation and Blank Areas With the Sandwich Function 625
Combining Numbers with Text 627
Displaying Dates 629
Merging Images Into Text 630
Editing Text 632
Types of Data Editing Objects 632
Working with Data Cell Objects 635
Data Cell Custom Output Patterns 638
Text Editor SuperObject 639

Creating and Modifying Text Editor SuperObjects 639
Text Editor Options 643
Converting Data Cells into a Text Editor SuperObjects 661
Automatically Creating Rows or Columns of Data Cells or Text Editor SuperObjects 662
Automatic Layout Options 663
Line Items in a Form 669
Tab Order in Forms 670
Tab Order for Variables 671
Field Setup in Graphics Mode 671
Word Processor SuperObject 673
Creating and Working With Word Processor SuperObjects 673
Using the Word Processor 677
The Ruler 680
Margins (Indents) 682
Tab Stops 685
Alignment 690
Line Spacing 691
Styles 692
Selecting Text 694
Configuring the Word Processor 696
Word Processor Document Storage Strategies 696
Storing a Collection of Documents 698
Searching for Text Within a Collection of Documents 701
Setting up Storage for a Template Document 702
Setting up Storage for Multiple Template Documents 703
Merging Data into Word Processing Documents 707
Forcing Merge Data to Update When Moving From Record to Record 714
Word Processor Options 716
Default Font and Text Size for New Documents 718
Printing Word Processor Documents 719
Printing Multiple Page Documents 724
Using the Mini Correspondence Wizard 727
Creating a New Letter 727
Printing a Letter 730
Printing a Mail Merge Letter 732
Viewing a List of Letters 734
Correspondence Templates 735
Merging Data into the Body of a Letter 737

Images & Movies 739

Fixed Images 739
Displaying and Printing EPS Images 741
Memory Requirements for Large Images 742
Tracing a Scanned Form 743
Flash Art™ 748
Creating Super Flash Art Objects 749
Using Flash Art to Display a Fixed Image 754
Using Flash Art to Display a Smart Background 755
Using Flash Art to Display a Color 759
The Flash Art Scrapbook (Gallery) 762
Adding a New Image to the Scrapbook 763
Locating an Image in the Flash Art Scrapbook 764
Removing an Image from the Flash Art Scrapbook 765

Renaming an Image 765
Re-Arranging the Image Order 766
Printing the Flash Art ScrapBook 766
Importing PICT Files into the Flash Art Scrapbook 766
Transferring the Flash Art Scrapbook to Another Database 767
Displaying Images Directly From Disk Files 767
Displaying Images in a Different Folder (Directory) 769
Displaying Non PICT Images (Enhanced Image Pack) 773
Image File Extensions in a Cross Platform Environment (MacOS and Windows) 775
Super Flash Art™ Options 777
Formula 777
Formula in a Variable 777
Default 778
Alt File 779
Include Pictures on Disk 780
Display Group of Pictures 780
Border 787
Drop Shadow 787
Overflow 788
Scroll Bars 788
Align 788
Displaying Images from Resource Files 793
Displaying Icons from Resource Files 799
Displaying Form Preview Pictures 799
Elastic Pictures 800
Using the Elastic Picture Workshop 801
Saving the Elastic Picture 804
Using the Elastic Picture 805
“Classic” Flash Art Objects 806
Storing Images in a Field 807
Displaying Movies in a Form 810

Buttons & Widgets 813

Push Buttons 813
Super Object Push Button 813
Push Button Styles 816
Button Title 818
Title Positioning 818
Standard Font/Size 818
3D Title 818
Hide Title 818
Click/Release 819
Color Options 819
“Classic” Push Buttons 820
Transparent Push Buttons 822
Flash Art™ Push Button SuperObjects™ 823
Data Buttons 827
Data Button SuperObjects™ 828
Creating a Group of Radio Buttons 831
Multiple Value Button Groups 835
Super Data Button Options 839
Data 839
Title 839

Value ... 840
Allow Multiple Values ... 840
Value Separator ... 840
"Radio" button ... 841
Procedure ... 841
Sample ... 841
Flash Art Data Button SuperObjects™ ... 842
Sticky Push Button SuperObjects™ ... 845
“Classic” Checkbox and Radio Buttons ... 846
Pop-Up Menus ... 850
Pop-Up Menu SuperObjects™ ... 850
The Pop-Up Menu Formula ... 853
Dividing Lines in the Menu ... 854
“Live” Pop-Up Menu Formulas ... 855
Pop Up Submenus ... 856
Pop-Up Menu Options ... 857
Data ... 857
Menu Formula ... 857
Menu Type ... 858
Display Options ... 860
Color ... 860
Procedure ... 860
Pop-Up Menu Font, Size and Dimensions ... 861
“Classic” Pop-Up Buttons ... 861
Creating a Pop-Up Menu with a Procedure ... 864
Where Will the Pop-Up Menu Appear? ... 864
The PopUp Statement ... 865
The PopUpButton Statement ... 865
The PopUpClick Statement ... 866
The PopUpFieldChoices Statement ... 866
The PopUpDoubleFieldChoices Statement ... 867
The PopUpByNumber Statement ... 868
The PopUpStyle Statement ... 868
“Live” Pop-Up Menu Formulas ... 868
List SuperObjects ... 869
Creating List SuperObjects™ ... 869
List Options ... 873
Data ... 873
Sep ... 875
Database ... 878
Sort Up ... 880
No Duplicates ... 881
Formula ... 881
Click Action ... 882
Grow Box ... 883
Thin Scroll Bars ... 884
Procedure ... 884
Click/Release ... 884
Building the List ... 884
“Hiding” Part of a List Item ... 887
Maximum List Size ... 888

Form Goodies 889

View-As-List Forms 889
How View-As-List Forms Work 890
Creating a View-As-List Form 892
Working with Tiles 898
Adding a View-As-List Header 899
Editable View-As-List Forms 903
View-As-List Background Colors 909
Buttons on a View-As-List Form 911
Elastic Forms 912
Theory of Elastic Forms 913
Building an Elastic Form 915
Defining the Quadrants 915
Maximum Window Size 919
Removing the Window's Scroll Bars 920
The Window Tweak Wizard 921
Opening Windows with a Procedure 923
Modifying an Elastic Form 923
Non-Rectangular Quadrants 924
Expanding Multiple Objects Proportionally 926
Elastic View-As-List Forms 927
Super Matrix Objects 929
The Matrix Template (and Frame Object) 930
Creating Super Matrix Objects 932
Linking with the Matrix Frame 934
Matrix Cell Borders & Background 935
Matrix Order 936
Matrix Rows and Columns 937
Designing a Matrix Template 939
Adjustable Size Templates 939
Tips for Adjustable Size Templates 942
Matrix Co-Ordinates and Data 943
Matrix Co-Ordinates (What cell is this?) 943
Co-Ordinates and Scroll Bars 944
Using a Matrix to Display an Array 944
Using the Matrix as a Button 945
What Cell Was Clicked? 945
Buttons Within Matrixes 945
Updating the Matrix Display 945
A Trick for Updating the Matrix Display Automatically 946
Super Matrix Case Studies 947
Displaying Music Tracks 947
Displaying Olympic Results 952
Television Station Guide 956
Building a Calendar 960
Scroll Bars 968
Scroll Bar "Theory" 968
Creating Scroll Bar SuperObjects™ 969
Scroll Bar Options 972
Data 973
Min 973
Max 973
Page Up/Down 973

16 Pixel ... 973
Procedure ... 973
Balloon Help ... 974
Creating Balloon Help Objects ... 975
Balloon Help Options ... 980
Changing the Cursor Shape Over Different Areas ... 980
Displaying Balloon Help Text Directly on the Form ... 981

Charts ... 983

Chart Data ... 983
Creating a New Chart ... 984
Setting Up Legend and Value Fields ... 988
Setting Up Additional Value Fields ... 989
Chart Types ... 990
Bar Charts ... 991
Line Charts ... 993
Area Charts ... 993
Pie Charts ... 994
Scatter Diagrams ... 995
Preparing the Database for Drawing a Chart ... 996
Ranking (Sorting) the Chart Values ... 999
Charts with “Other” ... 1001
Restoring the Original Data ... 1003
Maximum Number of Chart Points ... 1004
Dressing Up Chart Appearance ... 1004
Chart Font, Size, and Style ... 1004
Vertical Legends ... 1005
Output Patterns ... 1006
Graphic Attribute Icons ... 1007
Grid ... 1015
Non-Zero Axis OK ... 1016
Tick Mark Spacing ... 1017
Chart Preview ... 1018
Copying a Chart to Another Application ... 1019
Graphic Embellishments (Titles, Legends, Drop Shadows, etc.) ... 1020
Chart Flash Art ... 1022
Using Flash Art for Color or Blends ... 1026
Scatter Diagram Flash Art ... 1028
Connect Dots ... 1032
Printing a Chart ... 1033

Printing Basics ... 1037

Printing Different Views ... 1037
Printing the Data Sheet ... 1037
Printing Data Sheet Headers & Footers ... 1038
Printing a Form ... 1041
Preparing Data For Printing ... 1042
The Page Setup Dialog ... 1043
Fractional Fonts ... 1044
The Print Dialog ... 1045
Print Preview ... 1046
Print One Record ... 1048

Custom Reports 1049

Working with Tiles 1050
Creating Additional Tiles 1057
Creating A New Tile By Duplicating 1059
Tiles In Action 1064
Data Tiles 1064
Margins 1073
Top Margin Tile 1073
Left Margin Tile 1075
Right Margin Tile 1078
Bottom Margin 1078
Headers and Footers 1079
Header Tile 1079
Creating a Header Tile by Duplicating the Data Tile 1082
Footer Tile 1087
Page Numbers 1088
Printing the Current Date and Time 1091
First Page Header Tile 1093
BackDrop Tile 1095
Designing Headers and Footers For Changing Page Sizes 1097
The QuickReport Dialog 1099
Printing Multiple Page Records 1102
Selectively Printing Multiple Pages per Record 1103
Printing Data that Overflows a Page 1104
Variable Height Records 1111
Stacking Variable Height Objects 1117
The Expand/Shrink Option 1118
Mixing Variable Height Objects With Other Graphics 1119
Printing Multiple Column Reports 1120
Across or Down? 1122
Table Header and Table Footer Tiles 1123
Controlling the Number of Columns 1127
Spacer Tile 1128
Printing Summary Information 1129
Summary Tiles 1131
Printing Summaries Without Data 1135
Printing Data Grouped by Month, Quarter or Year 1136
Group Headers 1139
Group Sidebars 1143
Keeping a Group Together on a Column or Page 1148
Starting a Group on a New Column or Page 1151
Even and Odd Page Layout 1152
Special Paper Options 1154

Labels 1157

Label Fundamentals 1157
The QuickLabel Dialog 1157
Printing Labels on Sheets 1160
Printing 3 by 10 1” Labels (Avery 5160) 1161
Aligning Labels on the Sheet 1161
Printer Inaccuracy and Vertical Creep 1161
Printing Roll Labels 1161

Printing on 1-up 1” Roll Labels 1161
Printing Non 1” 1-up Labels 1162
Using Custom Page Size to Print Labels 1162
Using Standard Page Sizes to Print Labels 1162
2, 3, and 4-Up Roll Labels 1162
4-Up Cheshire Labels 1163
Selecting Font and Print Quality 1163

Formulas 1165

Formulas In Action 1165
Displaying/Printing A Formula 1166
Storing Formula Results in the Database 1168
Using a Formula to Locate/Select Information 1171
Formulas in Procedures 1174
Using the Formula Wizard 1175
Calculations with Database Fields 1176
Changing the Active Database 1178
Using Fields from Other Databases 1179
Topic and Functions Help Menus 1180
Function Search 1182
Special Formula Result Formats 1184
The Programming Reference Wizard 1187
Formula Components 1188
Formula Grammar 1188
Calculation Order and Parentheses 1189
Functions 1189
Multi-Parameter Functions 1190
Zero Parameter Functions 1190
Functions Menu 1191
Whitespace 1192
Grammar Errors 1193
Comments 1195
Values 1195
Constants 1195
Build in Constants: Pi, Carriage Return and Tab 1196
Fields 1196
Using the Current Field 1197
Line Item Fields 1198
Variables 1199
Variable Names 1199
What’s Inside A Variable? 1199
The Life Cycle of a Variable 1200
Creating Variables in a Procedure 1200
Initializing Variables 1201
Variables and Data Types 1201
SuperObject Variables 1201
Variable Name Conflicts 1202
Permanent Variable Tips 1202
Special Characters 1202
Working With Extremely Complex Formulas 1204
How Large Should the Buffer Be? 1204
Arithmetic Formulas 1205
Dividing by Zero 1206

Overflow/Underflow Problems 1206
Adding Line Item Fields 1207
Basic Numeric Functions 1207
Scientific Functions 1209
Financial Functions 1211
Text Formulas 1212
Gluing Strings Together 1212
Functions for Taking Strings Apart 1213
Taking Strings Apart (Text Funnels) 1214
Numeric Start and End Positions 1214
Specifying Numeric Length Instead of Position 1215
Start/End Positions by Character Matching 1215
Cascading Text Funnels 1216
Character Matching in Reverse Gear 1216
Stripping Out Individual Words 1217
Multiple Matching Characters for Start/End Position 1219
Non-Matching Character for Start/End Position 1220
Limitations of Text Funnels 1222
String Testing Functions 1223
String Modification Functions 1224
Converting Between Numbers and Strings 1227
Characters and ASCII Values 1230
Working with Character Values 1230
Invisible Characters 1231
The ASCII Chart Wizard 1232
Showing Character Ranges with the ASCII Wizard 1234
ASCII Character Constant Functions 1235
Text Arrays 1236
Picking a Separator Character 1236
Working With Arrays 1237
HTML Tag and Tag Parsing Functions 1243
Tag Parameter Functions 1245
HTML Table Parsing Functions 1246
HTML/URL Conversion Functions 1246
HTML Generating Functions 1247
Date Arithmetic 1248
Today’s Date 1248
Converting Between Dates and Text 1249
Date Functions 1250
Calendar Functions 1252
Time Arithmetic 1255
Converting Between Times and Text 1255
Time Calculations 1256
Time Calculations with Text 1258
Calculating Time Intervals Smaller Than One Second 1258
SuperDates (combined date and time) 1259
Reminders 1260
Appointments vs. To-Do’s 1261
Creating and Modifying a Reminder 1261
Reminder Functions 1263
Alarms 1264
True/False Formulas 1265
Comparison Operators 1265

A beginswith B ... 1266
A endswith B ... 1266
A contains B ... 1266
A notcontains B ... 1266
A soundslike B ... 1266
A match B ... 1267
A matchexact B ... 1268
A notmatch B ... 1268
A notmatchexact B ... 1268
A like B ... 1268
Combining Comparisons ... 1268
A and B ... 1268
A or B ... 1269
A xor B ... 1269
not A ... 1269
Equals Comparison vs. Assignment ... 1269
True/False Values ... 1270
The ? Function ... 1271
Linking With Another Database ... 1272
The Lookup Wizard ... 1273
Type Mismatch Problems ... 1276
Lookup Variations ... 1277
Looking Up Rates in a Rate Table ... 1278
Looking Up Multiple Fields From One Record ... 1278
The GrabData Function ... 1280
Looking Up Multiple Values at Once ... 1280
Linking Clairvoyance to the Lookup Key Field ... 1283
Looking Up Data in the Current File ... 1285
The Assign Function ... 1285
Zip Code Lookup ... 1286
US Post Office Abbreviation Functions ... 1287
Graphic Co-Ordinates ... 1287
Points ... 1288
Rectangles ... 1290
Colors ... 1294
Raw Binary Data ... 1296
The RPN Programmer's Calculator ... 1299
Converting Between Different Bases ... 1300
Calculations with Reverse Polish Notation ... 1301
Boolean Operators ... 1303
Disk Files and Folders ... 1304
Resource Files ... 1308
Import/Export Functions ... 1312
System and Database Information Functions ... 1314
System Information ... 1314
User Information ... 1317
Variable Information ... 1318
Database Information ... 1319
Window, Form and Report Information ... 1325
SQL Database Information ... 1332
Custom Functions ... 1333
The Custom Functions Wizard ... 1334
Function Names ... 1335

Parameter Names 1335
Advanced Topic: The FDF File 1335
Advanced Topic : Creating Custom Functions In A Procedure 1335
The Custom Functions (ProVUE) 1336

Procedures 1339

Programming Isn't Magic! 1339
Introduction to (Panorama) Programming 1339
Procedures 1340
Statements 1340
A Simple Procedure in Action 1341
Creating a Procedure with the Recorder 1348
Recording Mouse Clicks 1350
Non Recordable Menus and Tools 1351
Recording Data Entry 1351
Writing a Procedure from Scratch 1352
Writing Statements 1354
Trying Out a Procedure 1355
Checking for Mistakes 1357
Mysterious Errors 1359
Closing the Window When a Procedure is Finished 1359
Re-Opening a Procedure 1359
Font and Size 1360
Adding a Recording to an Existing Procedure 1360
Programming Reference Wizard 1361
Navigation Using the Search Panel and Topic List 1362
The Full Text Search Option 1363
Navigation Using the Topic, Statement and Function Menus 1365
Navigation Using HyperLinks 1365
Built In vs. Custom Statements and Functions 1366
Using the Template Panel 1366
Minimizing the Programming Reference Wizard 1367
Data Flow 1368
Assignment Statements 1368
Triggering Automatic Calculations 1368
The Define Statement 1369
Variables 1370
Creating a Variable 1370
Assigning a Value to a Variable 1371
Using a Variable in a Formula 1371
The Birth and Death of a Local Variable 1372
Long Life Variables 1372
Destroying a Variable 1372
Variable Accessibility 1373
Accessing "Dormant" Variables 1373
"Hidden" Variables and Fields 1374
Accessing Variables In Form Objects (Text or Images) 1374
Creating Variables with a SuperObject 1374
Permanent Variable Tips 1376
Displaying and Changing Variables 1376
Control Flow 1378
True/False Formulas 1378
Equals Comparison vs. Assignment 1379

True/False Values 1379
IF Statements 1380
ELSE Statements 1380
Nested if Statements 1380
Error Handling with if error 1381
CASE Statements 1382
LOOP Statements 1382
Stopping a Loop in the Middle 1383
Restarting a Loop in the Middle 1384
Subroutines 1384
CALL Statement 1384
Calling Procedures With Unusual Names 1385
Passing Values to a Subroutine (Parameters) 1386
Passing Values Back From a Procedure 1387
What if the parameters don't match the procedure? 1390
Calling a Subroutine in Another Database 1392
Terminating a Subroutine in the Middle 1392
Mini Subroutines within a Procedure 1393
Subroutines and Local Variables 1394
The UseCallersLocalVariables and UseMyLocalVariables Statements 1395
Recursive Subroutines 1396
Other Control Flow Statements 1396
Jumping to an Another Location in the Program 1396
Stopping the Program 1397
Aborting a Program 1397
Controlling the Abort Process 1398
Doing Nothing for a While 1399
Building Subroutines On The Fly (The Execute Statement) 1399
Tips for On-The-Fly Program Writing 1401
Execute and Local Variables 1403
Using Execute to Process Arrays 1403
Do It Yourself Data Merge 1405
On-The-Fly Subroutine Error Checking 1407
Building Parameters on the Fly (Parameters in a Variable) 1407
Catching Program Errors (Especially for Web and other Server Applications) 1407
Custom Statements 1408
The Custom Statements Wizard 1411
Creating Your Own Custom Statement Library 1413
Creating a New Custom Statement 1413
Setting Up a Procedure Information Block 1416
Processing Parameters 1418
Optional Parameters 1418
Repeating Parameters 1419
Raw Parameters 1419
Debugging a Custom Statement 1420
Accessing Forms & Procedures in the Library Database 1420
Advanced Topic: Using Libraries In Other Folders 1420
Program Formatting 1421
Notes To Yourself 1424
"Commenting Out" Statements 1424
Organizing Large Procedures (The Mark Menu) 1425
Suppressing Display of Text and Graphics 1427
Updating the Display After (or Within) a NoShow Block 1427

ShowPage 1428
ShowLine 1428
ShowFields field,field,…,field 1428
ShowColumns field,field,…,field 1429
ShowVariables var,var,…,var 1429
ShowRecordCounter 1429
ShowOther field,code 1430
Disabling the Watch Cursor 1430
Hide and Show 1430
Debugging a Procedure 1431
The Panorama Interactive Debugger 1434
The Debug Statement 1434
Using the Debugger 1434
Single Stepping 1435
Resuming Full Speed Execution 1437
Making Corrections to a Procedure 1437
Watching Computations 1437
Using the Inspector to Examine Fields, Variables and Formulas 1439
What Fields or Variables can be Displayed? 1442
Displaying Functions 1443
Procedure Debug Log 1444
The Procedure Log Window 1444
Recording a New Log 1445
Decoding Parameters and Assignment Statements 1449
The LogMessage Statement 1450
The Log Menu 1451
Cross Referencing 1452
The Cross Reference Wizard 1452
Opening a Form, Procedure or Crosstab 1454
Setting up a New Cross Reference 1456
Updating a Cross Reference 1457
50 Ways to Trigger a Procedure 1458
The Action Menu 1458
Action Menu Options 1459
Setting Different Menu Item Styles (Bold, Italic, etc.) 1459
Shortcuts/Command Key Equivalents 1459
Disabled Menu Items 1460
Separator Lines in a Menu 1460
Renaming the Action Menu 1462
Dividing the Action Menu into Multiple Menus 1463
“Unlisted” Procedures 1464
Live Menus 1465
The FileMenuBar Statement 1465
The .CustomMenu Procedure 1466
Programming the .CustomMenu Procedure 1467
The info("trigger") Function 1467
Processing Custom Menus with Simple IF’s 1468
Processing Custom Menus with Nested IF’s 1469
Splitting the Trigger into Menu/Item Names 1469
Menus with Modifier Keys 1470
Building Menus from Arrays 1470
Command Key Equivalents 1471
Menu Styles 1472

Disabled Menu Items and Separator Lines ... 1472
Submenus (Heirarchical Menus) ... 1473
Multiple Column Menus ... 1474
The WindowMenuBar Statement ... 1475
Advanced Topic: Live Menus Behind the Scenes ... 1475
Menu Titles ... 1475
Menu Items ... 1476
Submenus ... 1477
Formula Errors ... 1477
The menu(, menuitems(, arraymenu(and checkarraymenu(functions ... 1477
Helper Functions for Standard Menus ... 1478
Custom Menus (Resource Based) ... 1478
Important Note for Custom Menu Developers ... 1478
Custom Menu Overview ... 1478
Preparing a Resource File ... 1479
Creating a New Resource File ... 1480
Editing Within a Menu ... 1481
Command Key Equivalents/Shortcuts ... 1487
Adding and Removing Entire Menus ... 1488
Opening and Closing Resource Files ... 1489
Saving Resource Files ... 1490
Opening a Resource File in Panorama ... 1490
Sharing A Resource File Between Databases ... 1491
Assigning Custom Menus to a Form ... 1491
The .CustomMenuProcedure ... 1494
Submenus (Hierarchal Menus) ... 1494
Changing Custom Menus on the Fly ... 1497
Specifying Menus and Menu Items ... 1497
Menu Marks (Checkmarks, etc.) ... 1497
Checkmark On/Off Toggle ... 1498
Checking One Item in a Group ... 1498
Groups with Other… ... 1499
Disabling Menu Items ... 1500
Changing Menu Text on the Fly ... 1501
Rebuilding Entire Menus ... 1502
Reassigning Menus in the Menu Bar ... 1503
Custom Menu Troubleshooting ... 1504
Buttons ... 1505
Hidden Triggers ... 1507
Creating Hidden Trigger Procedures ... 1507
.About ... 1508
AutoGrow ... 1508
.ClearRecord ... 1509
.CloseWindow ... 1509
.CurrentRecord ... 1510
.CustomMenu ... 1510
.DeleteRecord ... 1510
.DialogKeyDown ... 1510
.Help ... 1511
.Initialize ... 1511
.KeyDown ... 1511
.ModifyRecord ... 1512
.NewRecord ... 1513

.OutOfBounds 1514
.ZoomFailed 1515
Data Entry Triggers 1515
Data Entry Triggers (Part Two) 1517
Hot Key Procedures 1518
HotKeys with Modifiers 1519
Universal HotKey Procedure 1519
Triggering a Procedure Every Second 1519
Triggering a Procedure Every Minute 1521
Triggering a Procedure As Soon As Possible 1521
Event Handler Procedures 1522
Text Editor SuperObject ..Handler Option 1523
Focus Procedure 1523
..OpenForm 1524
..ActivateForm 1524
..CustomAbout 1525
..CloseDatabase 1525

Programming Techniques 1527

Accessing Files 1527
Files and Folders 1527
Combined Folder Location and File Name 1528
Folder ID’s and Paths 1529
Locating a File with Standard Dialogs 1530
Customizing the Standard File Dialogs 1533
Opening a Panorama Database 1533
Suppressing the Default Extension 1533
Appending Databases End-to-End 1534
Eliminating Duplicates in Appended Data 1534
Replacing the Data in a Database 1534
Saving a Panorama Database 1535
Closing a Database 1535
Shutting Down Panorama 1536
Importing Text Files 1537
Carriage Returns in the Data 1537
Importing a Text File into an Existing Database 1537
Importing from a Variable 1538
Importing HTML Tables 1538
Re-Arranging the Order of Imported Data 1538
Building the ImportUsing Formula on the Fly 1540
Exporting Text Files 1541
Exporting Line Items as Separate Records 1542
Analyzing Line Items 1543
Exporting Array Elements as Separate Records 1543
Opening a Document in Another Application 1544
Smart Merge Synchronization 1545
How Smart Merge Synchronization Works 1545
Adding Smart Merge to Your Database 1545
The Modified Field 1546
Adding New Records 1546
The Smart Merge Procedure 1547
Directly Reading and Writing Disk Files 1549
What’s in a File? 1549

Reading Data Files 1550
Writing Data Files 1551
Copying Data Files 1552
Using FileSave and ArrayBuild to Export Data 1552
Reading and Writing Resource Forks 1555
Erasing a File 1556
Changing a File's Name 1556
Changing a File's Type and Creator 1556
Creating a New Folder 1556
Getting Information about a File 1557
Getting and Setting Additional File Information 1558
Accessing and Modifying File Permissions 1558
Building a List of Files or Folders 1558
Building a List of Disks (Volumes) 1559
Working with Resources 1560
Opening and Closing Resource Files 1562
Opening a Resource File in the .Initialization Procedure 1562
Reading a Resource 1563
Reading STR and STR# Resources 1563
Writing a Resource 1564
Deleting a Resource 1565
Renumbering a Resource 1565
Listing Resources 1565
Working with Multiple Resource Files 1567
Accessing the Windows Registry 1568
Getting Information About Registry Items 1568
Modifying Registry Entries 1569
Deleting a Registry Entry 1570
Monitoring Memory Usage 1571
Changing the Scratch Memory Allocation 1571
Windows 1572
Opening a Window 1572
Specifying the New Window Location 1573
New Window Options 1575
Non Standard Window Styles 1576
Changing a Window's Position/Options 1577
Changing a Window's View 1578
Changing the Name of a Window 1578
Scrolling Inside a Form Window 1578
Closing a Window 1579
Trapping the Close Box 1580
Changing The Window Order (Who's on Top?) 1580
Temporary "Invisible" Windows 1581
Databases Without Windows 1582
"Magic" Windows 1583
Window Clones 1584
Designing A Clone Window Application 1585
Alerts 1591
Displaying a Message 1591
Alerts With Multiple Buttons 1593
The Alert Statement 1596
Obsolete Alert Statements 1598
Suppressing Alerts 1599

The SuperAlert Statement 1599
The DisplayData Alert 1606
Dialogs 1607
Basic Text Entry Dialogs 1607
The SuperGetText Statement 1609
Obsolete Text Entry Statements 1612
The SuperChoiceDialog Statement 1612
Custom Dialogs 1616
Preparing a Form for Use as a Dialog 1616
Customizing the Dialog Code 1622
Options to the RunDialog Statement 1626
Editing Database Information with a Dialog 1628
Custom Dialog Menus 1630
Accessing and Modifying the Database Structure (Fields) 1632
Getting Information About Field Structure 1632
Modifying Field Structure Directly 1633
Working With the Design Sheet 1634
Updating Database Structure From Another Database 1635
Transferring Permanent Variables 1636
Verifying Database Identity 1636
Database Navigation and Editing 1637
Moving Up and Down in the Database 1637
Moving Left and Right 1640
Moving “Left” and “Right” on a Form 1641
Moving to an Empty Line Item Field 1643
Adding and Deleting Records 1644
Modifying the Database One Cell at a Time 1647
Accessing and Modifying the Current Cell 1647
Accessing and Modifying the Clipboard 1648
Triggering Automatic Calculations 1648
Triggering Automatic Procedures 1649
The Set Statement 1649
The FormulaCalc Statement 1650
Opening the Input Box 1650
“Natural” Data Entry 1652
Natural Data Display 1653
Natural Data Entry 1655
Validating a Credit Card Number 1658
Moving Data Between Files 1659
Cross Database Assignment Statements 1659
Identifying Data to Move 1659
Transfer Function Parameters 1660
Single Record Transfer Functions 1662
grabdata(target database,data field) 1662
lookup(target database,key field,key value,data field,default value,summary level) 1662
lookuplast(target database,key field,key value,data field,default value,summary level) 1662
lookupselected(target database,key field,key value,data field,default value,summary level) 1663
table(target database,key field,key value,data field,default value,summary level) 1663
Clairvoyance and Lookups 1665
The SpeedCopy Statement (Multiple Assignments in One Statement) 1665
Multiple Record Transfer Functions 1667
lookupall(target database,key field,key data,data field,separator) 1667
lookupcalendar(target database,key field,key data,data field,separator) 1667

lookuprtime(target database,key field,key data,pattern,separator) 1667
After a Lookup…Modifying the Original Data 1668
Using Lookups for Display/Printing 1669
Using ArrayBuild to Transfer Data Between Files 1670
Posting Data to Other Databases 1673
The Post Statement 1673
The PostAdjust Statement 1674
Sorting 1676
Reducing Screen “Flashing” 1676
Making Sorts Even Faster 1676
Locating Information 1677
Finding Information 1677
A Handy Universal Find Procedure 1678
Find Next 1680
Selecting Information 1682
Handling Empty Selections 1683
Selecting Duplicates 1684
Live Clairvoyance™ 1684
Adding a Cancel Search Button 1690
Clicking on the Live Clairvoyance List Object 1690
Summaries and Outlines 1691
Summary/Outline Examples 1692
Calculating Grand Totals 1694
Running Total 1696
Running Difference 1696
Transforming Big Chunks of Data 1697
Making Transformations Even Faster 1698
Numeric Calculations with FormulaFill 1698
Suppressing Zero’s 1699
Fill vs. FormulaFill 1700
Using FormulaFill to Transform Text 1702
Date Calculations with Formula Fill 1704
The SEQ Function 1705
Filling Empty Cells 1706
Automatic Numbering 1708
Propagate and UnPropagate 1709
Using UnPropagate to Eliminate Duplicates 1709
Change (Find and Replace) 1709
Changing with the Replace(Function 1712
Data Style and Color 1713
Accessing Style and Color in a Formula 1715
Processing/Transforming an Entire Array 1716
“Filtering” an Array 1716
Stripping Blank Elements From An Array 1717
Reversing the Order of an Array 1718
Using Regular Text Functions with Arrays 1718
Sorting an Array 1718
Removing Duplicate Items from an Array 1719
Building an Array from a Database 1719
Appending an Array to a Database 1720
Copying Between Multiple Variables and an Array 1721
Editing an Array using Separate Variables 1723
Processing/Transforming Binary Data 1723

Bits ... 1723
Bytes ... 1724
Words ... 1724
Long Words ... 1724
Creating Binary Values ... 1725
One Dimensional Arrays of Binary Values ... 1725
The CharacterFilter Statement ... 1726
The ChunkFilter Statement ... 1726
The TextFilter Statement ... 1727
Data Dictionaries ... 1728
Key/Value Pairs ... 1728
Storing a Key/Value Pair in a Dictionary ... 1728
Accessing Dictionary Entries ... 1729
Another Technique For Initializing a Dictionary ... 1730
Additional Methods for Modifying Dictionary Entries ... 1731
Looking Up a Dictionary Key Given Its Value ... 1731
Accessing the Internet ... 1732
Basic Web Access ... 1732
Fetching Web Pages ... 1732
Parsing Web Pages ... 1733
Fetching Images ... 1738
Relative URLs ... 1738
Submitting Forms ... 1739
Cookies ... 1741
Accessing Web Content ... 1742
Maps ... 1742
General Zip Code Information ... 1742
Street Address Information ... 1743
White Pages ... 1744
FedEx Shipment Tracking ... 1745
Controlling Web and E-Mail Clients ... 1746
Displaying a Web Page ... 1746
Displaying a Web Page on a Local Hard Drive ... 1747
Displaying a Map ... 1747
Sending an E-Mail ... 1748
Sending E-Mail ... 1749
Channel Configuration ... 1750
Sending a single e-mail ... 1751
Sending multiple e-mails ... 1751
Programming Graphic Objects on the Fly ... 1753
Basics of Graphic Object Programming ... 1753
Selecting an Object by Name ... 1753
Selecting Multiple Objects ... 1753
Getting Information About Individual Objects ... 1754
Modifying Selected Objects ... 1759
Getting Information About Selected Objects ... 1762
Object ID Values ... 1763
Redrawing an Object ... 1763
Dragging a Rectangle ... 1764
Movable Dividers ... 1768
Drag and Drop ... 1770
Drag Items and Flavors ... 1770
The Dropalyzer Wizard ... 1771

Dragging Items from Panorama ... 1772
Dragging a Single Flavor ... 1772
Dragging Multiple Flavors ... 1773
Receiving Dragged Data ... 1774
The .DropProcedure ... 1774
Dropping Files and Folders on Panorama ... 1776
VCard Drag and Drop ... 1776
Drag and Drop (Obsolete Method) ... 1778
Program Control of SuperObjects™ ... 1786
The Active SuperObject ... 1787
Accessing and Modifying a SuperObject's Internal Data ... 1789
Internal Data Types ... 1789
Text Editor SuperObject Commands ... 1790
Text Editor Internal Data ... 1796
Text Display SuperObject Internal Data ... 1797
Word Processor SuperObject Commands ... 1798
Word Processor Internal Data ... 1809
Super Flash Art Commands (Including Movie Control) ... 1810
Super Flash Art Internal Data ... 1813
Converting Between Image Formats ... 1814
Building Web Like HyperText Systems with Super Flash Art ... 1816
Preparing Pictures with Extractable Text ... 1816
Programming a HyperText Engine ... 1818
Extracting All Text of a Specific Style ... 1820
Creating Multi-Page Pictures ... 1821
Push Button Internal Data ... 1822
Flash Art Push Button Internal Data ... 1823
Data Button SuperObject Internal Data ... 1823
Flash Art Data Button SuperObject Internal Data ... 1824
Sticky Push Button SuperObject Internal Data ... 1825
Pop-Up Menu SuperObject Internal Data ... 1826
List SuperObject™ Commands ... 1827
Using Drag and Drop to Change the Order of Items in a List ... 1831
List SuperObject Internal Data ... 1832
Auto Grow SuperObject™ Commands (Elastic Forms) ... 1833
Auto Grow SuperObject Internal Data ... 1833
Super Matrix SuperObject™ Commands ... 1834
Super Matrix SuperObject Internal Data ... 1835
Scroll Bar SuperObject™ Commands ... 1836
Speech Synthesis ... 1838
The Speak Statement ... 1838
Embedded Speech Commands ... 1838
The StopSpeaking Statement ... 1838
The info("speaking") Function ... 1838
Buffered Speech ... 1839
Speaking Using the Speech Wizard ... 1840
Printing ... 1841
Selecting a View for Printing ... 1841
Selecting a Printer ... 1841
Adjusting Page Setup ... 1841
Preparing Data For Printing ... 1841
Printing the Database ... 1841
Printing a Single Record ... 1842

Print Preview ... 1842
Printing Using an Alternate Form ... 1843
Printing Data in an Array ... 1844
Form Comments ... 1845
The FormSelect Statement ... 1847
Reading and Modifying Form Comments in a Procedure ... 1848
Writing Your Own Channel Modules ... 1849
The ModuleInformation Procedure ... 1849
Channel Specific Procedures ... 1850
The Channel Workshop Wizard ... 1851
Previewing the ModuleInfo Procedure ... 1853
Creating the Module ... 1854

Cross Platform Databases ... 1855

File Type/Creator vs. Extensions ... 1855
Platform Converter Wizard ... 1856
Converting from Macintosh to PC ... 1857
Converting Resources ... 1857
Converting from PC to Macintosh ... 1857
Converting from Panorama 3.x to V (Macintosh) ... 1858
Fixing a Database With Missing or Incorrect Type/Creator Information ... 1859
Sharing Databases Across a Cross Platform Network ... 1859
Cross Platform vs. Older Versions of Panorama ... 1859
Cross Platform Font Usage ... 1860
Cross Platform Programming ... 1860
File Name Extensions and the OpenFile Statement ... 1860
Name Extensions and Window Names ... 1861
Flash Art Formulas ... 1861
Using Partial Paths to Reference SubFolders ... 1861
Hard Coded Folder Locations ... 1861
Is It a Mac or a PC? ... 1862

AppleScript ... 1863

Learning Basic AppleScript ... 1863
Script Debugger ... 1864
AppleScript and Panorama ... 1864
Using AppleScript to Control Other Applications from Panorama ... 1865
Launching an External Script from Panorama ... 1865
Building a Script Within a Panorama Procedure ... 1866
Getting the Result from an AppleScript ... 1866
Using Panorama Field and Variable Values within an AppleScript ... 1867
Using the UNIX Command Line ... 1867
Using AppleScript to Control Panorama from Other Applications ... 1868
Everything You Really Need to Know… ... 1868
Value of Cell ... 1868
Executing Panorama Procedures ... 1869
Transferring Data Between AppleScript and a Panorama Program ... 1869
Working with Lists ... 1870
AppleScript & Panorama… The Rest of the Story ... 1870
The Required Suite ... 1871
The Core Suite ... 1872
The Objects ... 1873

Additional Resources 1877

On-Line Resources 1877
Signing Up For Panorama News Via E-Mail 1878
Signing Up to Join Other Panorama Users On-Line (QNA List) 1879
QNA Digest Mode 1879
QNA Log Database 1880
Technical Support 1881
Getting the Most from Technical Support 1881
Panorama Conferences 1882
ProVUE 98 Conference 1883
Panorama Skills Track 1883
Advanced Track 1884
Programming Track 1884
Internet Track 1885
Cross Platform Track 1885
ProVUE 99 Conference 1886
Web Track 1886
Basic Skills Track 1887
Intermediate Skills Track 1887
Advanced Skills Track 1888
Publications 1889
Panorama Real World Programming Guide 1889
Panorama Security Handbook 1889
Panorama Partner/Server Handbook 1889
Consulting Services 1889

Welcome to Panorama!

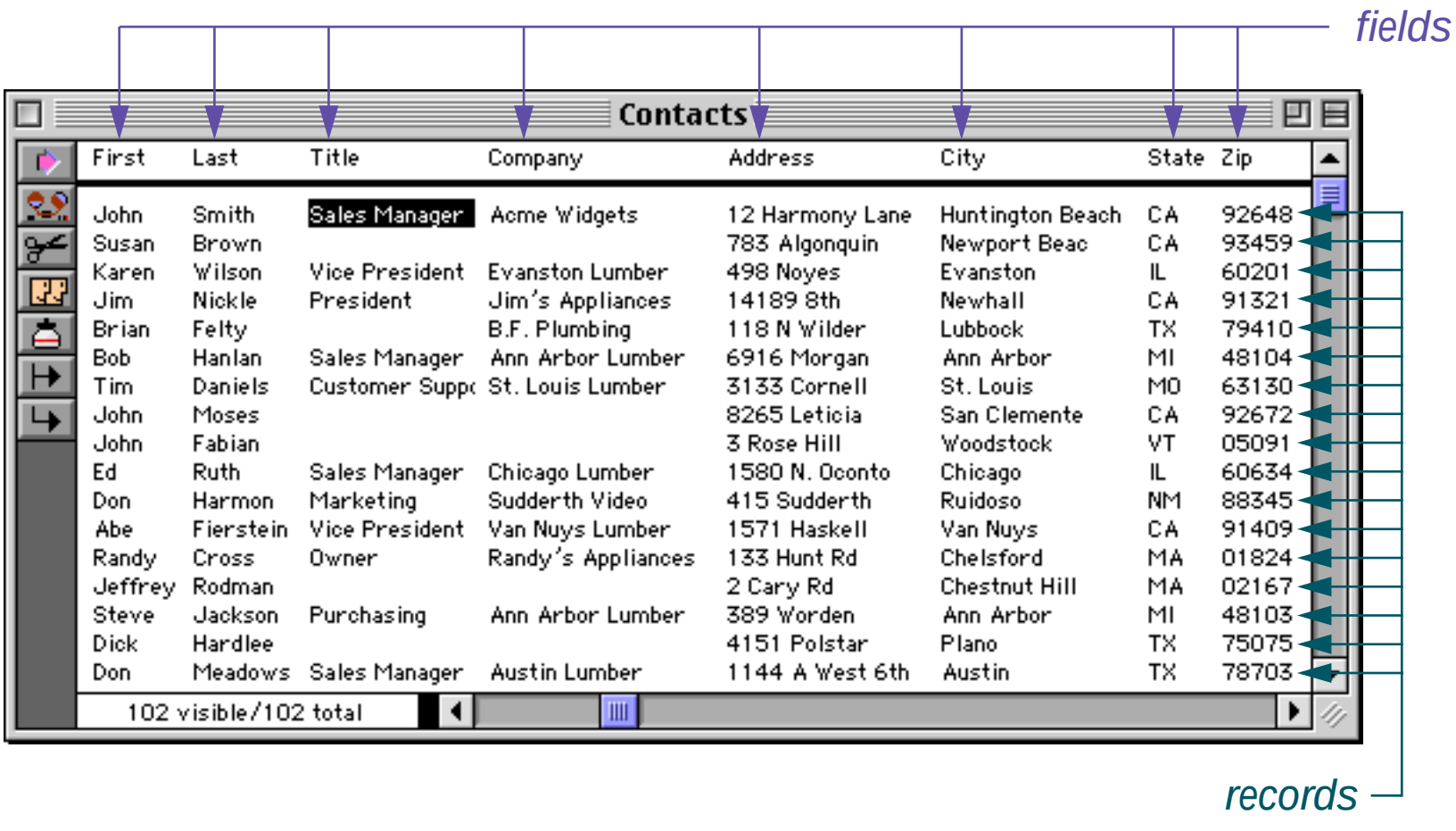


Congratulations! You are about to get acquainted with Panorama, a powerful tool for organizing and understanding information. With Panorama you can store, retrieve, categorize, summarize, chart, merge and print your information.

This book explains how to use Panorama. It assumes that you are already familiar with the basics of operating your computer. You should be familiar with pointing, clicking and dragging with the mouse, copying files, choosing commands from pull down menus, using scroll bars and editing text. If you are not familiar with these topics please review the training material that came with your computer.

What is a Database?

The right knowledge at the right time can advance a career, a company, a civilization. Obtaining the right information at the right time is not often an easy task, especially when you are confronted with large amounts of data. Data that is arranged randomly (for example a pile of receipts) won't do you much good. When a collection of data is organized it is called a **database**. A computer program for entering and manipulating the information in a database is called a database program or a database management program.



Every database— whether on paper or in a computer — is composed of records. A **record** contains a collection of information about a particular person, company or entity. For example, each record in a mailing list database would contain the name and address of a particular person, while each record in an invoice database would contain all of the information collected for a single order. To learn more about records see “Records” on page 241.

Each record is divided into fields. A **field** contains a single piece of information about the subject of the record, for example a name, a street address, a phone number, etc. Fields are what make a database more than a hodgepodge of random information. Each field appears in the same place within every record in the database. To learn more about fields see “Fields” on page 247.

A database program like Panorama helps you design, manage and use data that is structured into records and fields. Before you begin you must tell the database program how you want the fields to be set up. As you enter new data Panorama helps make sure that everything goes in the right place and the structure remains intact. Once data is entered, Panorama can quickly scan the data to find the information you need, or categorize and summarize the information you need for a report. Panorama can also re-organize the data (for example, sorting) or even change the database field structure as your needs change — without having to start over from scratch. Virtually any job that can be done manually with a filing cabinet, card file or paper list can be done faster and more efficiently with a computerized database program like Panorama.

A Brief History of Database Technology

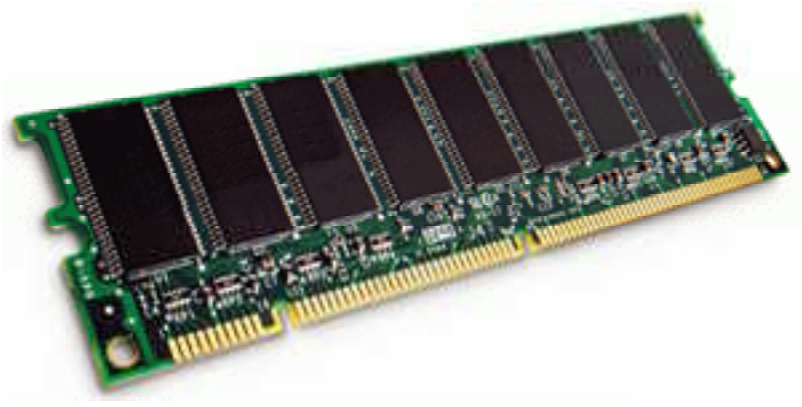
The need for data organization goes back long before computers. Before computers data was usually organized with paper forms and filing cabinets. Unfortunately, the information in a row of filing cabinets is hardly at your fingertips — at best an item might be located in a minute or two, while at worst a misfiled item might never be found at all.



In the 1960’s and 1970’s disk based database systems revolutionized the collection and storage of data. Instead of storing the data in a filing cabinet it is stored on a spinning magnetic disk (hard drive). Depending on how the information is filed an item can be located in as little as a second. Today’s popular database programs, including Access and FileMaker, are based on this hard drive technology that was originally developed 30 to 40 years ago.



Storing information on a spinning disk is a fantastic improvement over filing cabinets, but it still relies on a mechanical system. Although disk drives have become faster and faster over the years, the speed of rotation and the movement of the head over the disk platter are still a bottleneck, just as feet, hands and fingers were a bottleneck when accessing data in a filing cabinet. Fortunately, there is an alternative. Besides the hard drive, your computer contains a large internal electronic memory bank (RAM) that allows the computer to work with large quantities of information at pure electronic speeds. In the past this electronic memory was too small for large databases, but today's computers have enough RAM for all but the largest database tasks.

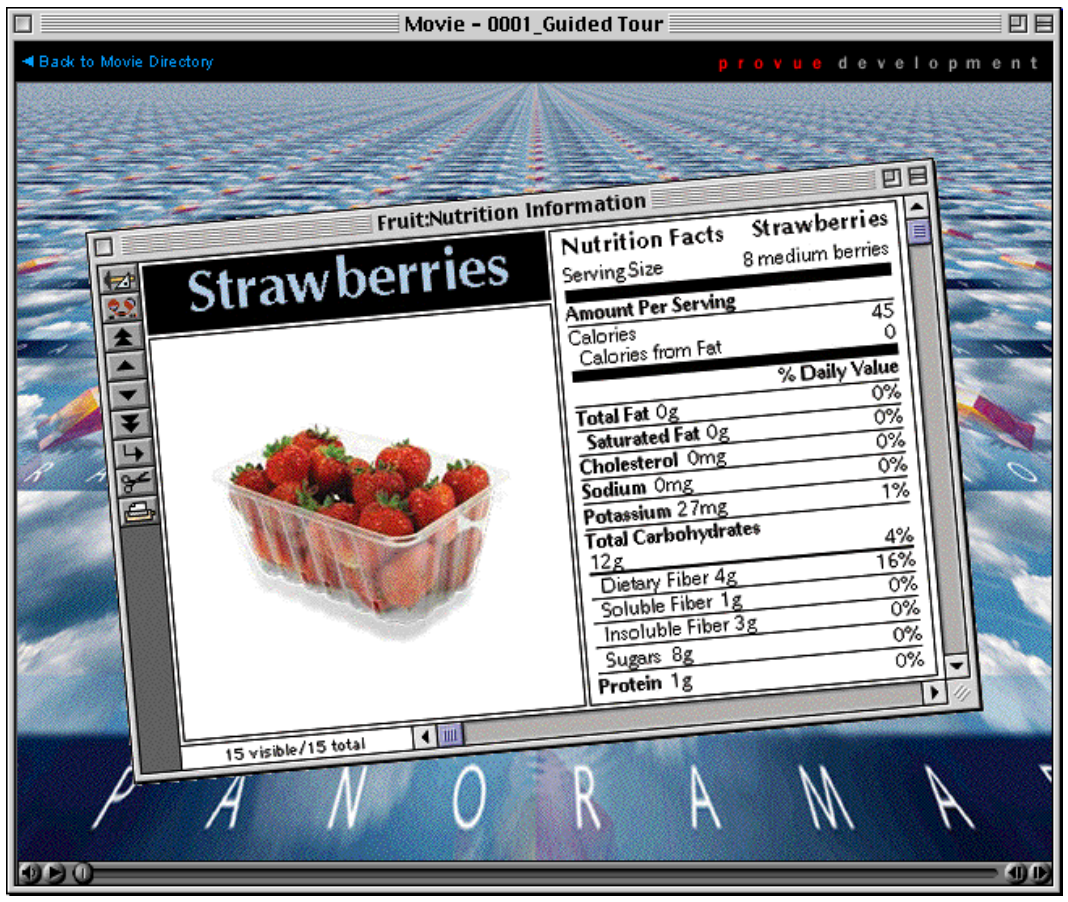


As you have probably guessed by now, Panorama is designed to leapfrog existing disk based database software by using your computers electronic memory for ultra fast operation, making Panorama faster, easier to use, and more powerful than any other database software previously available. Panorama uses the hard drive only for permanent data storage. Searching, sorting, summarizing and other data processing tasks are performed entirely in RAM.

What is Panorama?

In addition to its blazing RAM based speed, Panorama includes a number of unique capabilities that set it apart from other database programs. Panorama is easy to learn and use, is blazingly fast, has powerful tools for analyzing financial data, and is powerful enough for even the most demanding database jobs.

If you are just getting acquainted with Panorama, be sure to check out the guided tour movie included on the CD. Just sit back and relax while we show you Panorama's unique tools for organizing information. All of the movies are recorded digitally and allow you to pause, back up, or skip ahead to the topics that most interest you (see "Once you have clicked on all of the blue bold characters you can press the Continue button to continue with your work." on page 106).



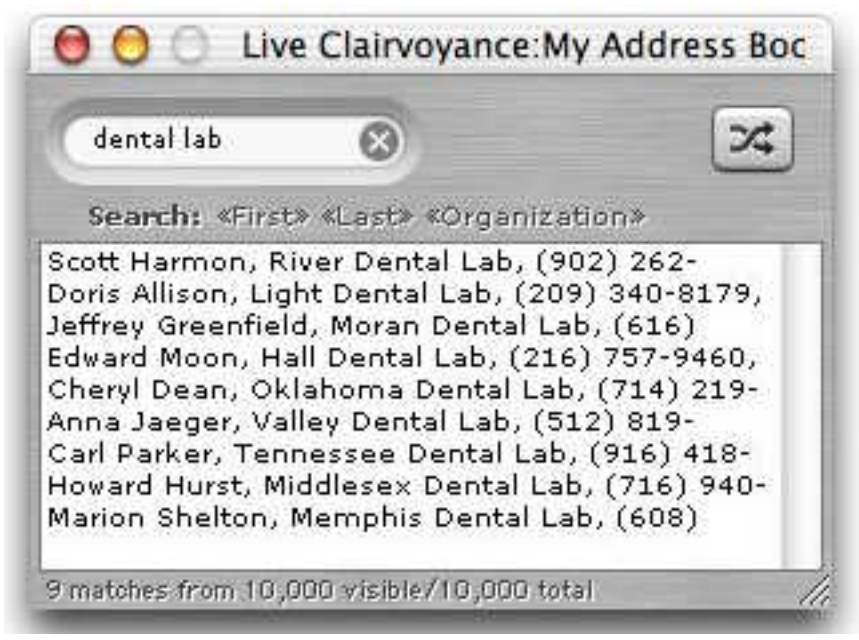
Wizards

Panorama's **Wizard** menu contains pre-built databases for automating common tasks and enhancing productivity when using Panorama (see "Guide to Wizards & Demo Files" on page 1). General productivity wizards include databases for organizing your contacts, calendar, correspondence and tracking your time. Panorama also includes wizards for importing and exporting data, arranging windows, locating favorite files, creating new databases and much more.



Super Fast Searching and Sorting

Panorama's RAM based speed makes searching and sorting faster and more flexible than ever before (see "Sorting" on page 359 and "Searching and Selecting" on page 367). Searching is not limited to full word or begins with matches — you can search for data that contains a word or phrase or even perform formula based searches (for example "find all names longer than 12 characters" or "find all invoices where shipping is more than 10% of the order total" (see "Formula Find / Select" on page 387). The **Live Clairvoyance™** feature allows you to perform "live" searches on any Panorama database. The search results are updated dynamically as you type, allowing you to "hone in" on just the information you are looking for. See "The Live Clairvoyance™ Wizard" on page 379 and "Live Clairvoyance™" on page 1684.

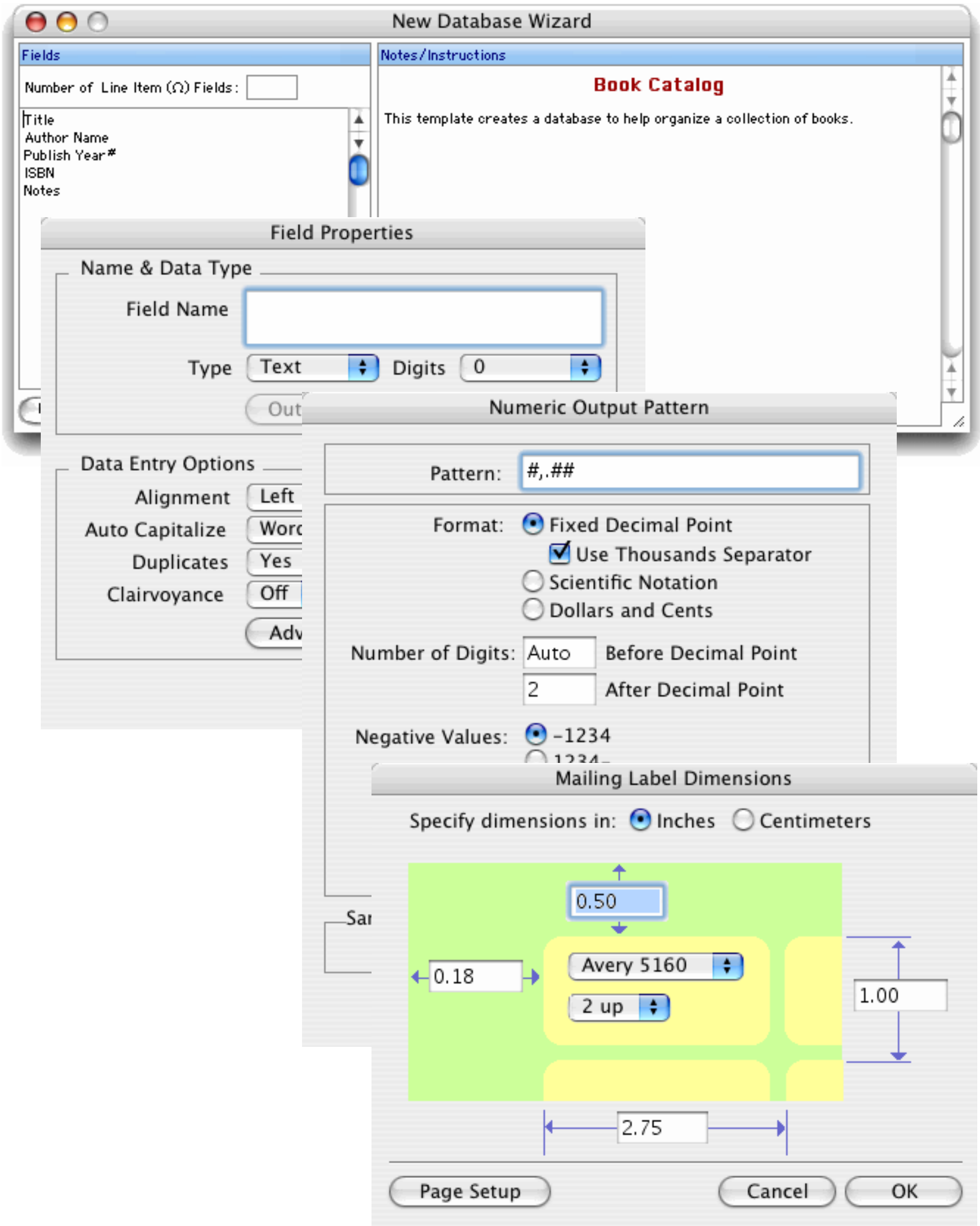


Phonetic Searching

Panorama's "sounds like" option allows you to search for data phonetically (see "The Find / Select Dialog" on page 369). For example a search for "sounds like Alan" will turn up anyone named Alan, Allan, or Allen.

Easy Set-Up

Simple step-by-step dialogs make it easy to define database fields (see "The Setup Menu" on page 248), print mailing labels (see "Labels" on page 1157), and print custom reports (see "Custom Reports" on page 1049).



Crosstabs

Panorama's crosstab feature allows it to quickly and automatically convert raw data into a tabular summary; for instance turning raw checkbook data into a monthly budget. Crosstabs are one of the most powerful tools yet for analyzing financial data (see "Crosstabs" on page 433).

Corporate Checkbook:XTABS:Spending by Quarter

xtab	1q98	2q98	3q98	4q98	TOTAL
Advertising	8,592.43	7,681.09	5,185.44	8,552.43	30,011.39
DEPOSIT	0.00	0.00	0.00	0.00	0.00
Fixed Assets	3,338.75	3,555.61	2,064.40	1,063.90	10,022.66
Insurance	3,763.50	3,763.50	3,763.50	3,763.50	15,054.00
Legal		282.44	1,186.08	1,425.11	2,893.63
Office Supplies	3,495.79	1,655.83	1,577.07	2,556.82	9,285.51
Payroll	23,408.67	22,866.11	22,440.16	23,504.18	92,219.12
Purchases	11,289.42	7,409.52	6,013.66	8,813.83	33,526.43
Rent	4,740.00	4,740.00	4,740.00	4,740.00	18,960.00
Shipping	2,067.84	3,428.15	2,890.98	2,746.63	11,133.60
Telecom	1,392.75	1,272.40	1,322.30	1,362.93	5,350.38
Utilities	694.84	684.57	610.40	600.91	2,590.72
+TOTAL	62,783.99	57,339.22	51,793.99	59,130.24	231,047.44

Data Outlines

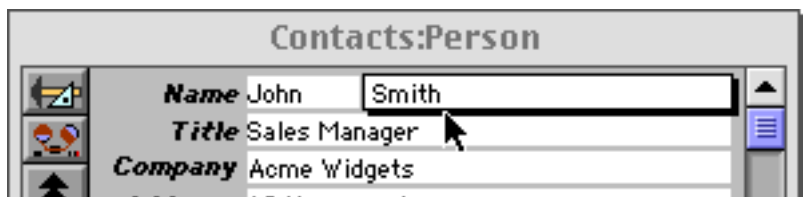
Panorama's outlining feature is another powerful tool for analyzing financial data, and goes much further than ordinary subtotal calculations (see "Summaries and Outlines" on page 393). You can hide and reveal different levels of subtotals, identify problems or opportunities and then zoom in on specific details. You can also perform further operations or calculations on subtotals as if they were data.

			Rent	**35,026.34**
02/09/99	1962	Airborne Express	Shipping	35.40
		Airborne Express		**35.40**
03/26/99	2018	AIRS	Shipping	138.07
		AIRS		**138.07**
02/09/99	1971	American Customs House	Shipping	34.00
		American Customs H		**34.00**
07/16/99	2191	B J D Trucking Inc.	Shipping	37.50
		B J D Trucking Inc.		**37.50**
09/19/99	2277	Burlington Air Express	Shipping	95.17
		Burlington Air Expre		**95.17**
03/16/99	2005	Consolidated Freight	Shipping	150.20
		Consolidated Freight		**150.20**
01/30/99	1929	Federal Express	Shipping	178.75
03/20/99	2015	Federal Express	Shipping	170.00
03/28/99	2032	Federal Express	Shipping	150.00
05/14/99	2101	Federal Express	Shipping	170.00
		Federal Express		**668.75**
01/31/99	1930	U P S	Shipping	52.97
02/09/99	1946	U P S	Shipping	122.60
03/20/99	2014	U P S	Shipping	25.00
03/28/99	2034	U P S	Shipping	197.42
04/17/99	2056	U P S	Shipping	25.00
05/04/99	2079	U P S	Shipping	25.00
05/08/99	2088	U P S	Shipping	25.00
07/24/99	2207	U P S	Shipping	45.80
08/21/99	2250	U P S	Shipping	155.00
09/19/99	2281	U P S	Shipping	95.32
		U P S		**769.11**
			Shipping	**1,928.20**
			Taxes	**5,152.79**
			Telephone	**5,690.50**
			Utilities	**2,054.89**
				183,651.22

46 visible/563 total

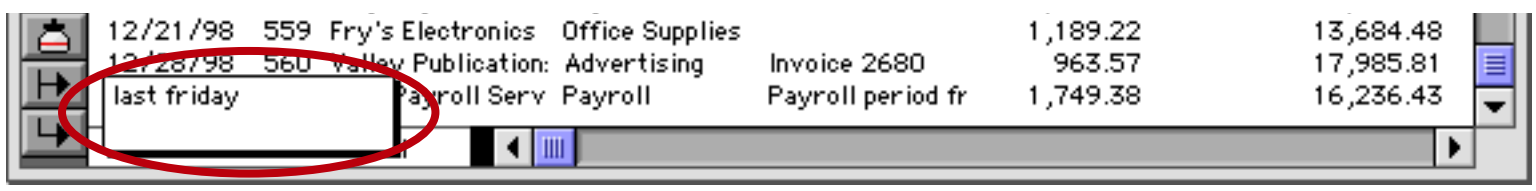
Data Entry Shortcuts

Panorama's data entry shortcuts reduce keying errors and data entry errors (see "Data Entry & Editing" on page 303). Panorama's unique Clairvoyance® feature automatically finishes typing for you (see "Clairvoyance®" on page 321). Auto-capitalization, data entry buttons, smart defaults, and optional spelling checker and zip code lookup save even more time.



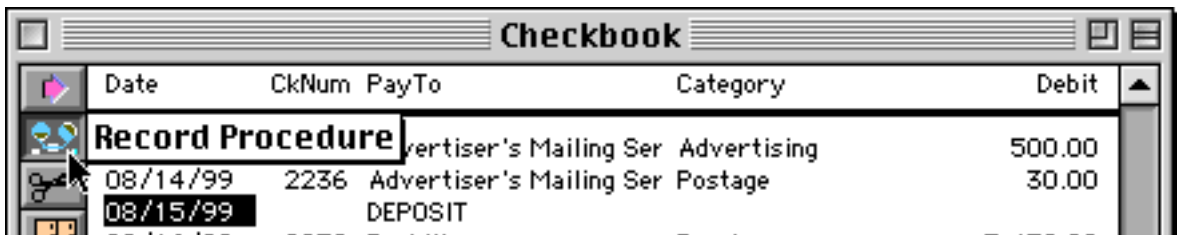
Smart Dates™

Panorama understands dates the way you do—as part of weeks, months, quarters, or years (see "Entering Dates" on page 294). You can easily locate or summarize information by any of these date periods.



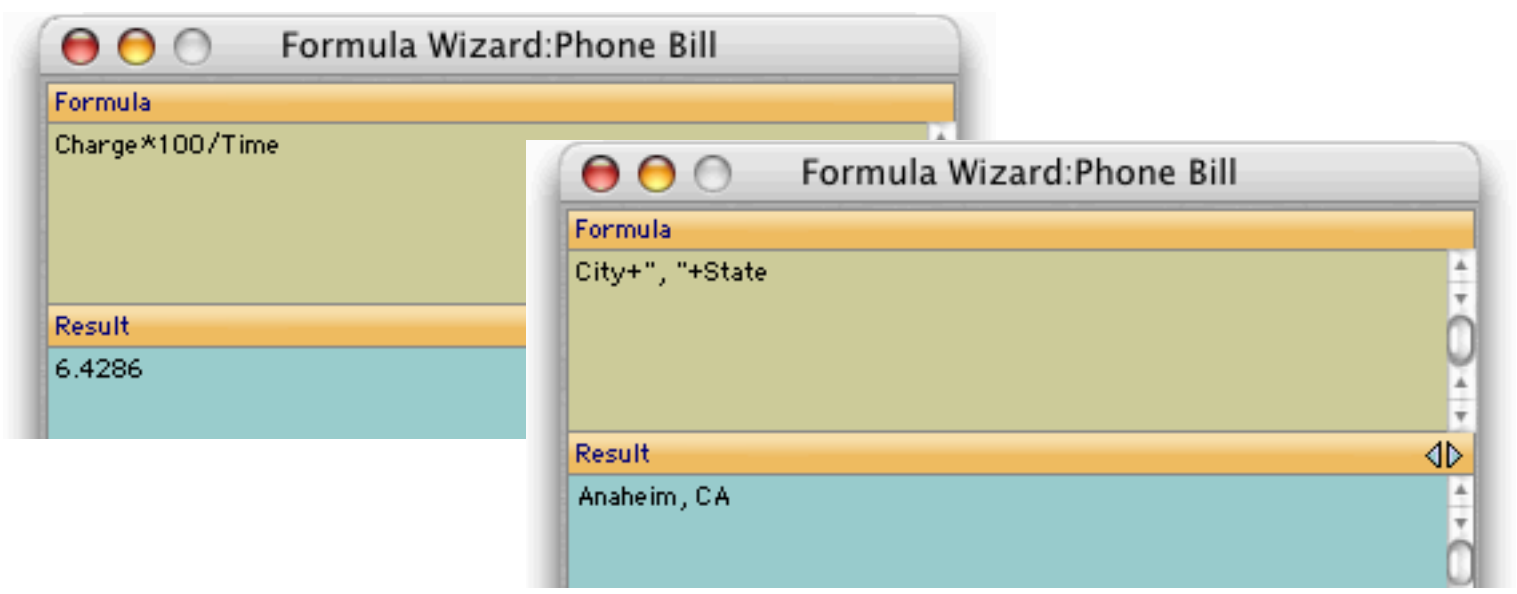
Smart Program Recorder

Panorama's program recorder allows you to record multi-step operations and play them back later with a single click or keystroke. It's as easy to use as a cassette recorder—just start the recorder then do your work (see "Creating a Procedure with the Recorder" on page 1348). The recorder doesn't just record mouse clicks and keystrokes as is, but automatically converts them into simple English-like commands that can be edited later if necessary.



Formulas

Panorama can perform simple and complex calculations on numbers, text, and dates, and includes a wizard to help you build and test formulas (see "Formulas" on page 1165).

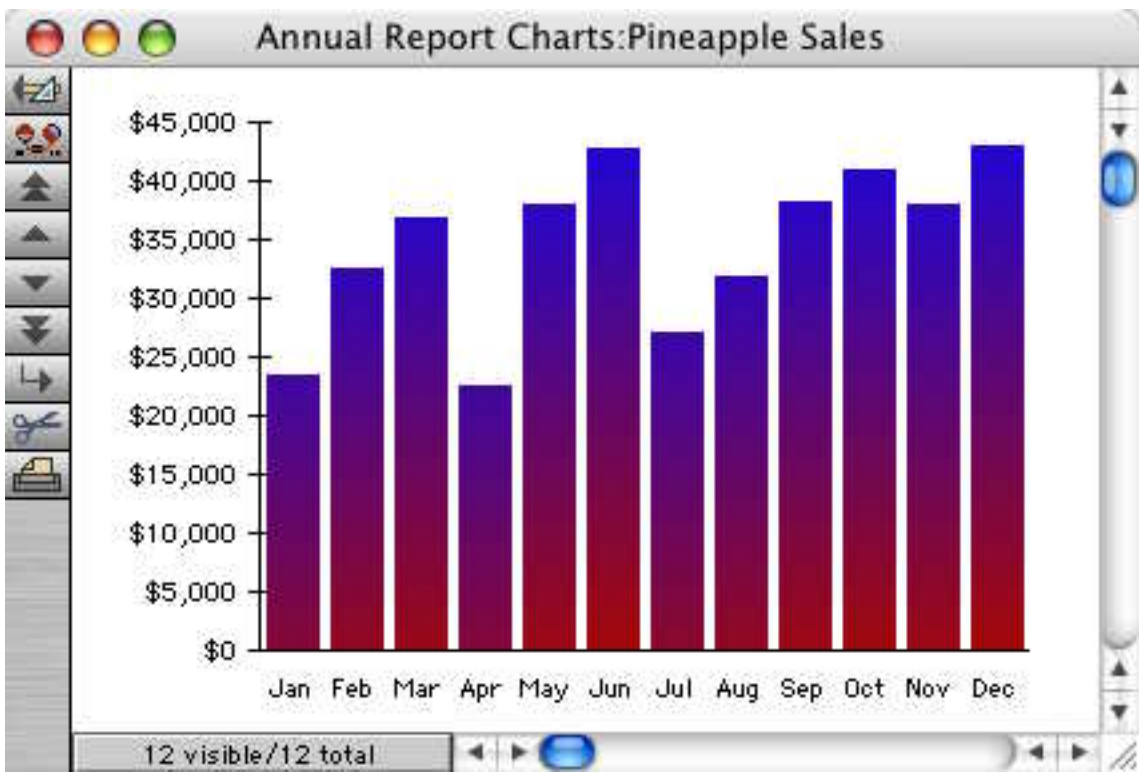


Compact Storage

Panorama uses up to 85% less disk space than other database programs for the same data.

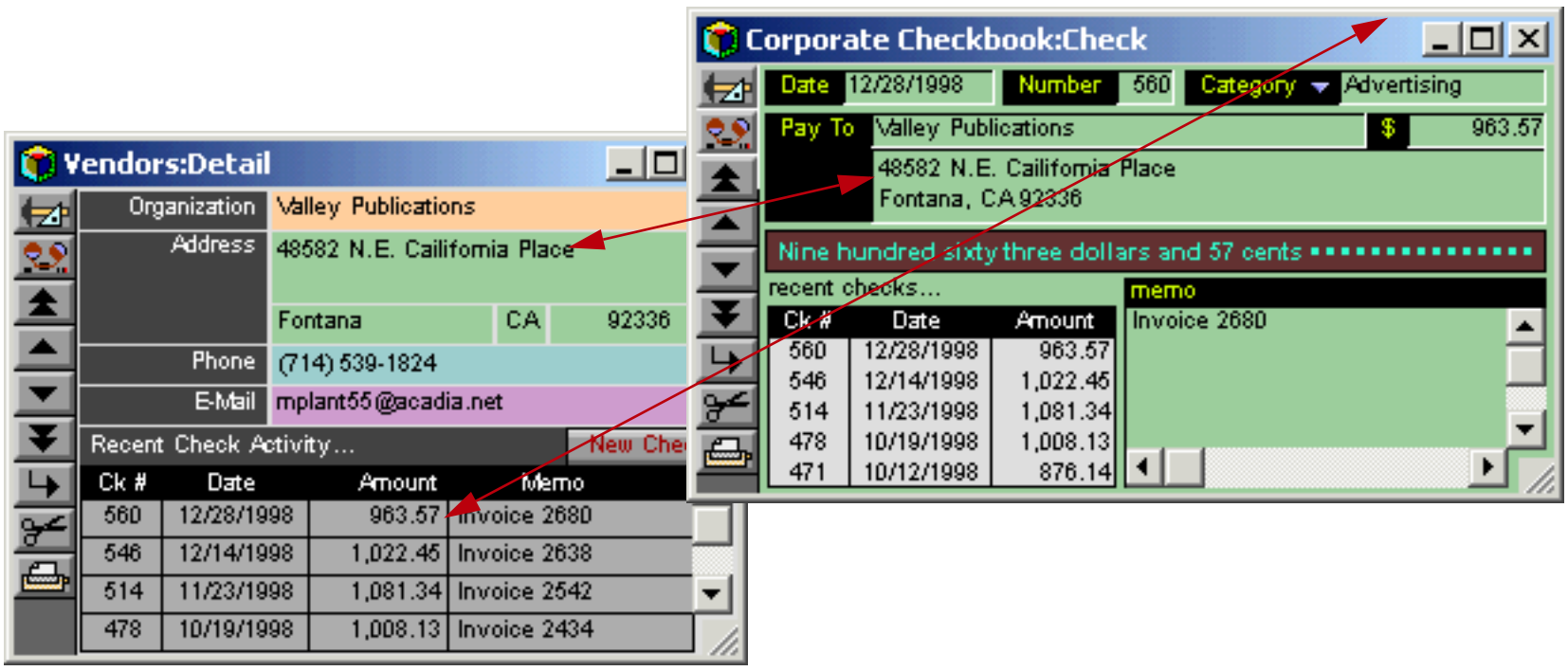
Charts

Panorama's built in charts (bar, line, area, pie, and scatter) can visually reveal trends and relationships that are often hidden in a conventional report (see "Charts" on page 983).



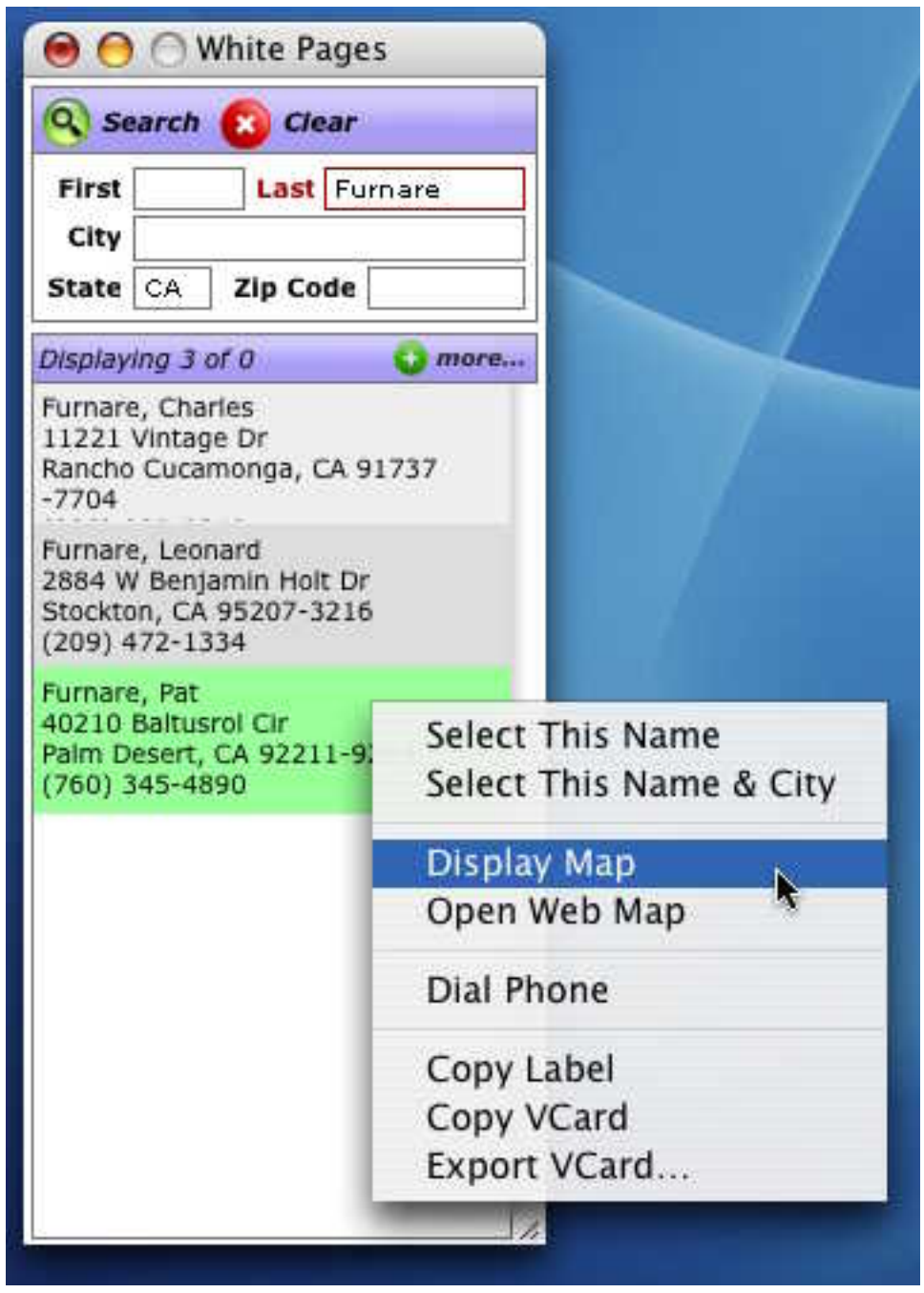
Relational Links

Panorama can relate the information in two or more databases, insuring consistency and simplifying complex tasks like order entry, billing, payroll, sales lead tracking, and more (see "Linking With Another Database" on page 1272).



Internet Content

Panorama includes built-in access to web content, including maps, phone and zip code information, fedex tracking and more. If data is on the web, you can access it with Panorama. See "White Pages" on page 35, "Address Info" on page 27, "Fedex Tracking" on page 32 and "Basic Web Access" on page 1732



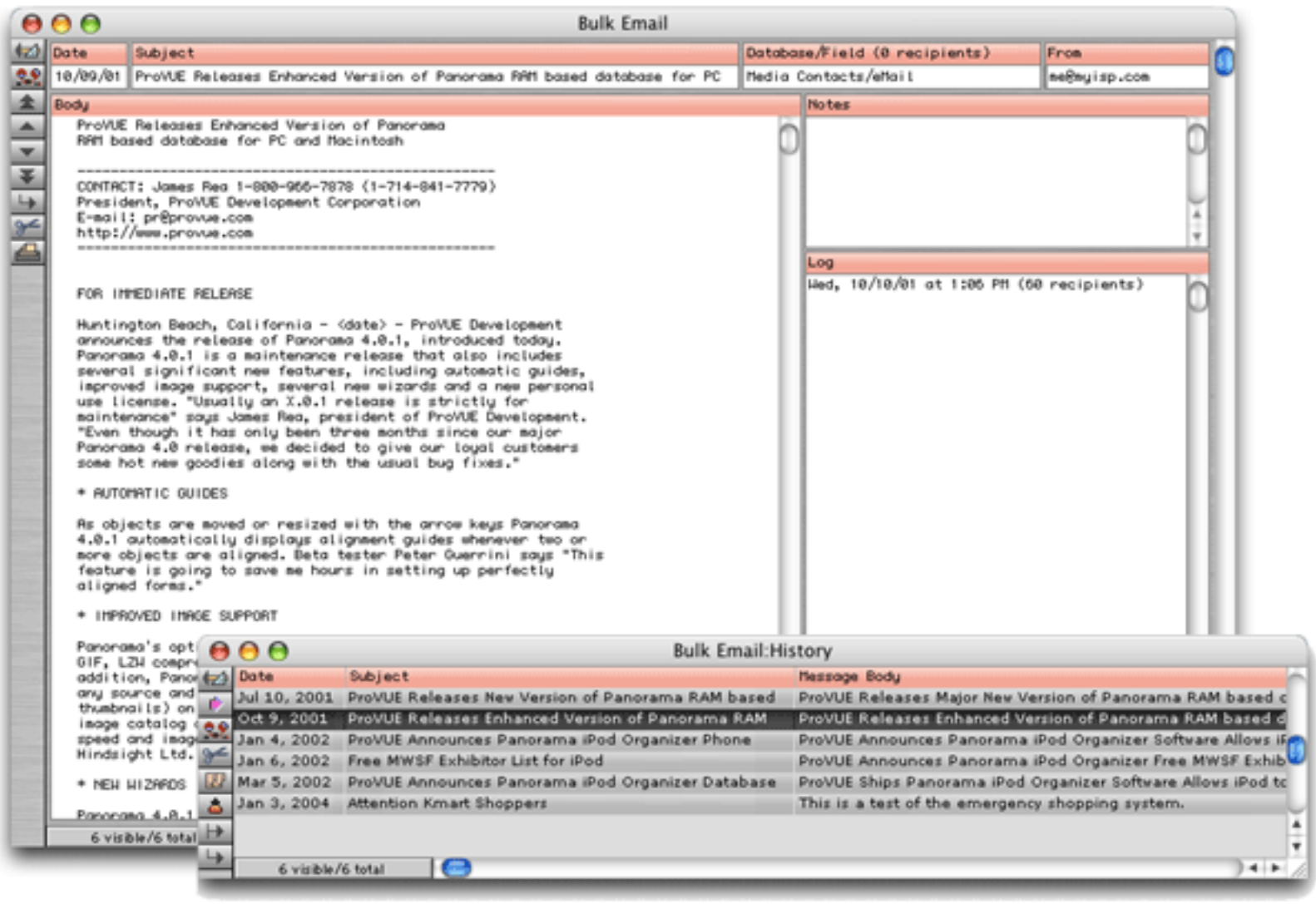
Map, Phone, Email and VCard Support

Based on information in your databases Panorama can automatically display maps, dial the phone, send e-mail and drag and drop between Panorama and any other VCard enabled application (like Address Book).



Sending Email

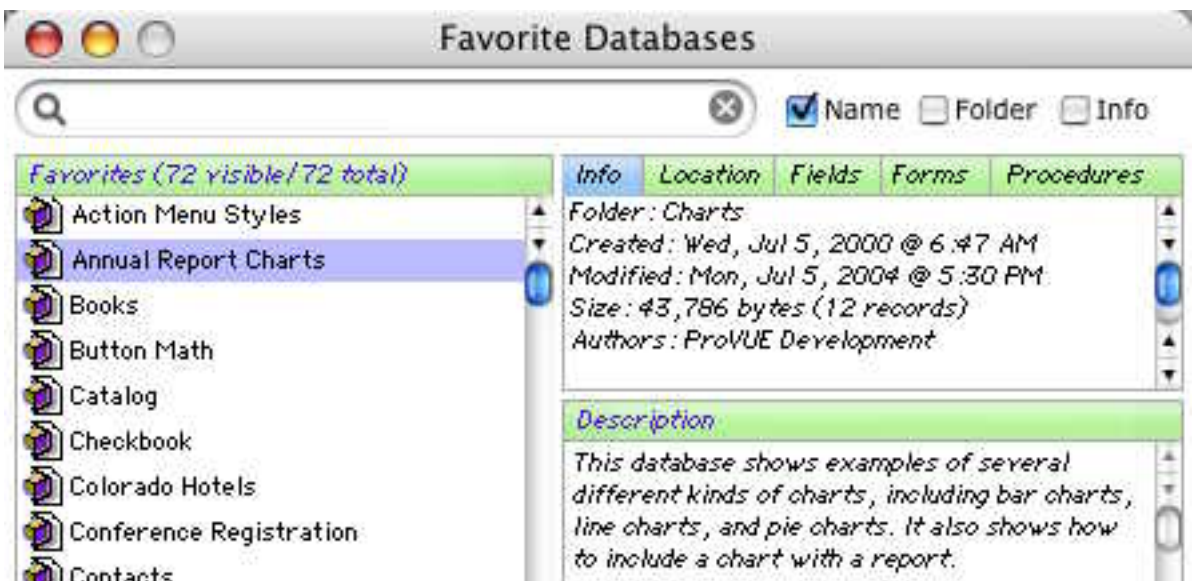
Panorama can send e-mail to individual or bulk recipients. You can program e-mail transmission into your own database (for example setting up an invoice database to automatically e-mail a copy of each invoice) or use the **Bulk Email** wizard. This wizard makes it easy to send and keep track of bulk emails. It keeps all of the previous e-mails you've sent organized, and can automatically extract e-mail addresses from one or more other databases. The wizard has two primary windows. The Bulk Email window displays a single e-mail message, and allows you to configure and modify that message. The History window displays a list of the previous e-mails.



For more details about this wizard see “Bulk Email” on page 30.

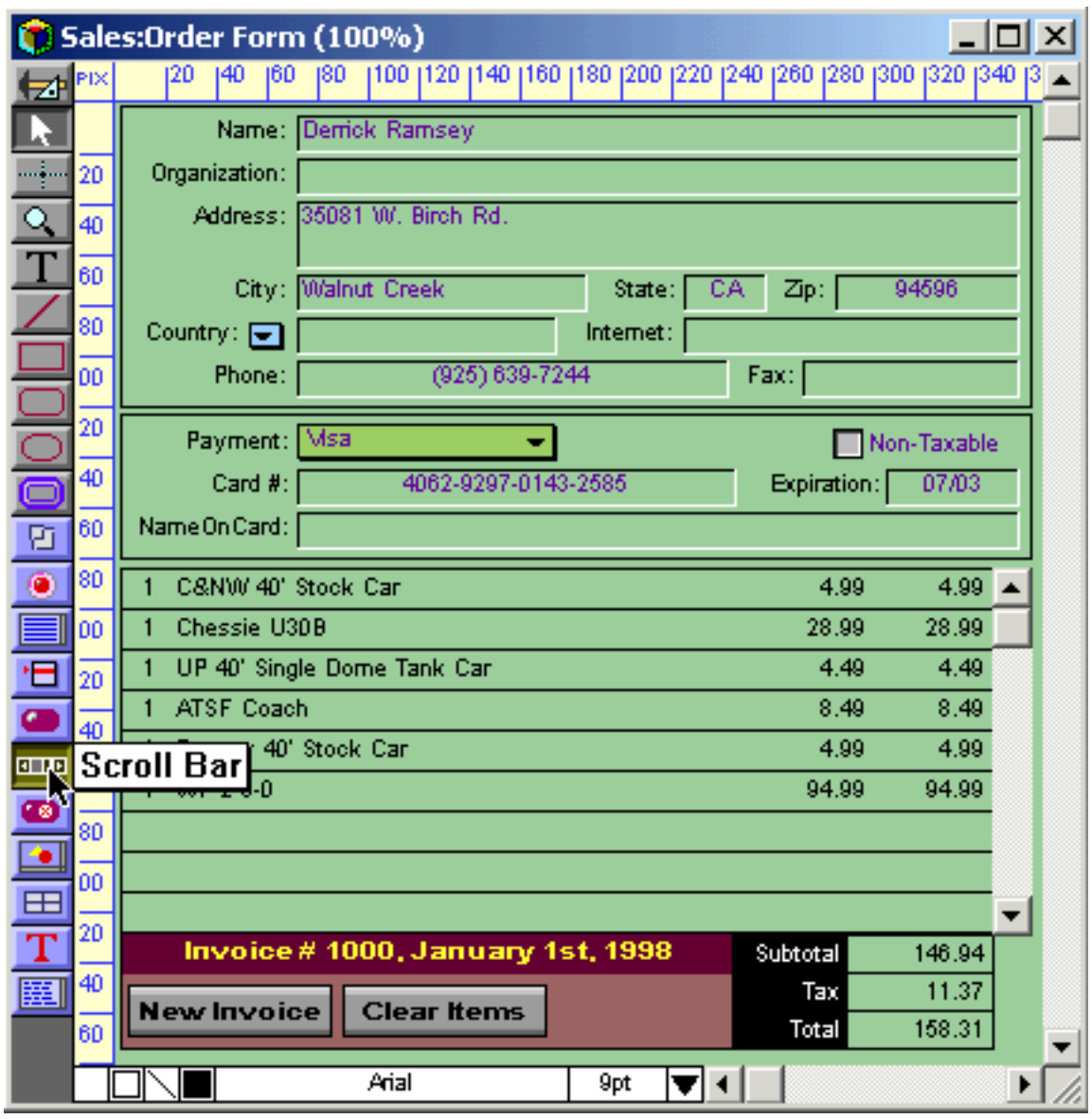
Favorite Database Wizard

The Favorite Database Wizard makes it easy to keep all of your databases at your fingertips (see “The Favorite Databases Wizard” on page 109).



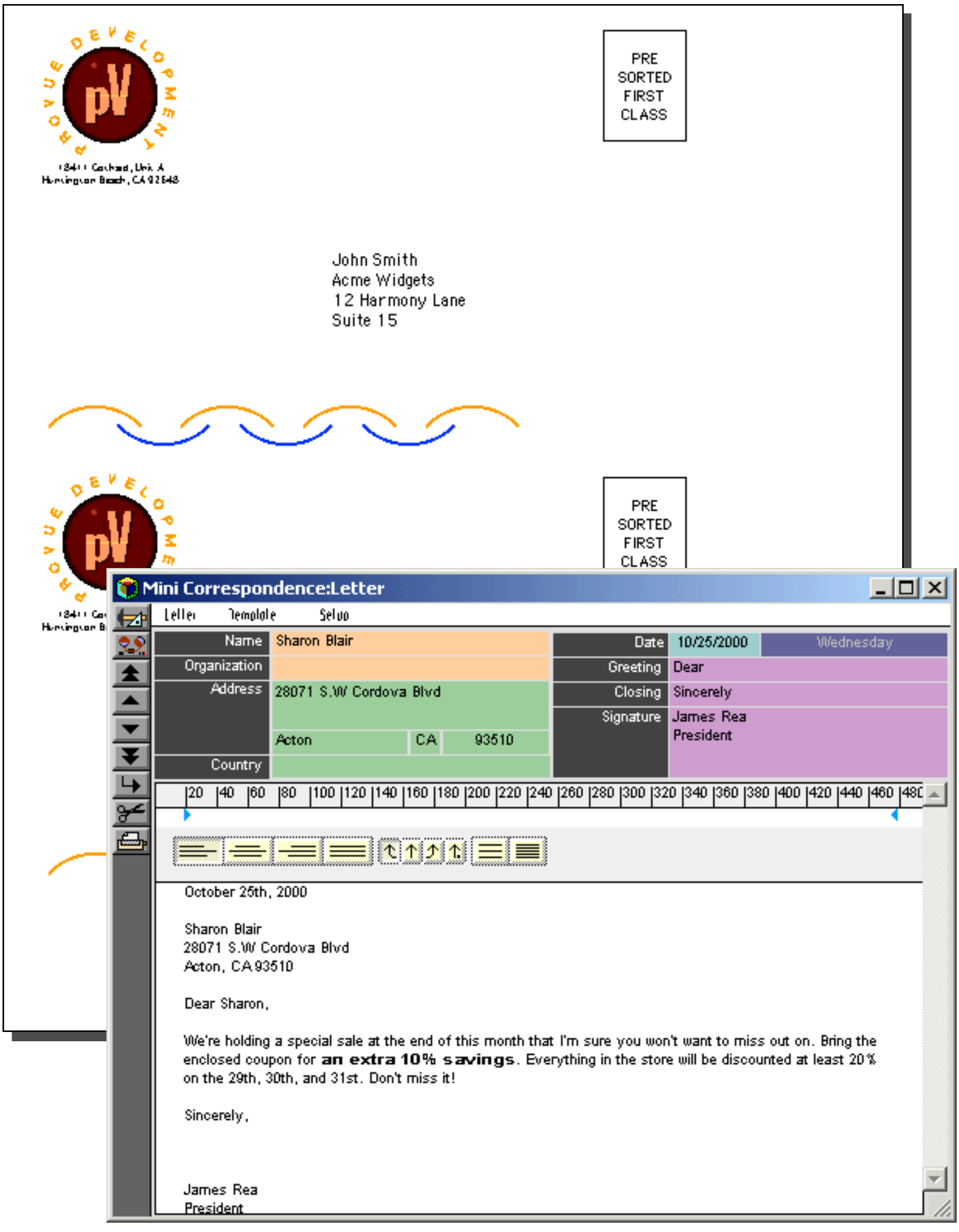
Advanced Graphic Tools

Panorama's graphic capabilities build on standard Macintosh drawing features and add special tools and dialogs that are designed specifically for creating and modifying tables within forms and reports (see "Graphic Design" on page 491).



Mail Merge and Labels

Panorama's built-in mail merge can produce all the components of a direct mailing—including custom form letters, mailing labels, and postcards or envelopes (see "Labels" on page 1157 and "Custom Reports" on page 1049). Panorama includes a built-in word processing program, reducing both the complexity and time required to put together a direct mailing (see "Word Processor SuperObject" on page 673).



Database Publishing

Panorama can automatically create complex database reports including catalogs, directories, bibliographies, and more. Reports can include both fixed and variable height elements (including images), and Panorama can automatically control page and column breaks to eliminate widows and orphans (see “Custom Reports” on page 1049).

CALIFORNIA



Cupertino Christy Alpert, Signal Research
Arthur Clairmont, South Coast Office Products
Harold Cobb, Cobb Associates
Sherry Grossman, Pablo Distribution
Peter Parks, Hamilton Press
Michelle Adams, Sceptre

Long Beach Kathy Schwartz, Wendover Insurance Group

Los Angeles Charles Arrow, Arrow, Inc.
Dave Elko, First Row Group

Palo Alto Cindy Blunden, Hot Lines, Inc.
Robert Dorn, Valley Services
Roxie Jacobsen, Alpha Pic
Donna Brady, Challenger Air Cargo

San Diego Jan Morgan, McCormick-Ridder
Frank Pearce, Taylor & Associates
Ray Cavalier, Stagg Instant Press

Windows Annoyances

David A. Karp, 280 pages



Windows is full of all kinds of oddities and frustrating behavior. This book is full of complaints, but it is also full of positive suggestions for getting around the roadblocks. Some of the suggestions are simple, but little known, while others are much more technical. Covers Windows 95 and NT 4.0, though of course much of the material also applies to Windows 98. Don't read this as your first Windows book, but after you've been using Windows for a while you'll get a lot out of this relatively slim book.

Designing for the Web : Getting Started in a New Medium

Jennifer Niederst,Edie Freedman , 180 pages

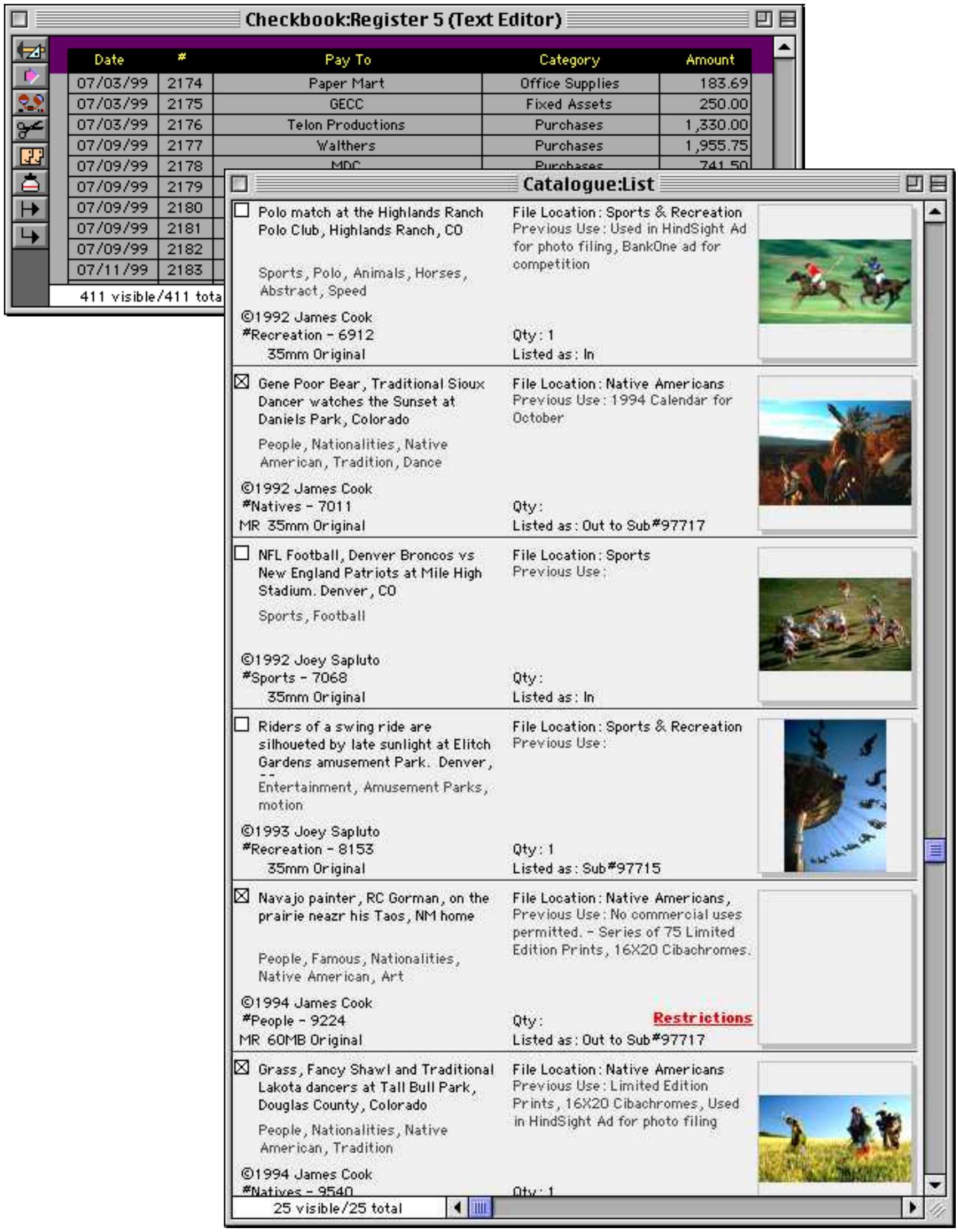


This book provides an excellent introduction to HTML. It is especially targeted towards graphic artists. Unlike many other HTML books that are heavy enough to use as lethal weapons, Ms. Niederst has filtered out the essentials that you really need to know first, while leaving out the overwhelming mass of detail that is usually included in most other web tutorials. (When using this book with SiteWarrior, keep in mind that SiteWarrior automatically adds the document header and footer for you, so you don't need to worry about including the <html>, <head>, <title> and <body> tags.)

Although this book is an excellent introduction to HTML, it does not cover more advanced topics. The biggest ommision is tables, which are mentioned, but not explained. Nevertheless, this is one of the best books on the market for someone who is just getting started with HTML for the first time.

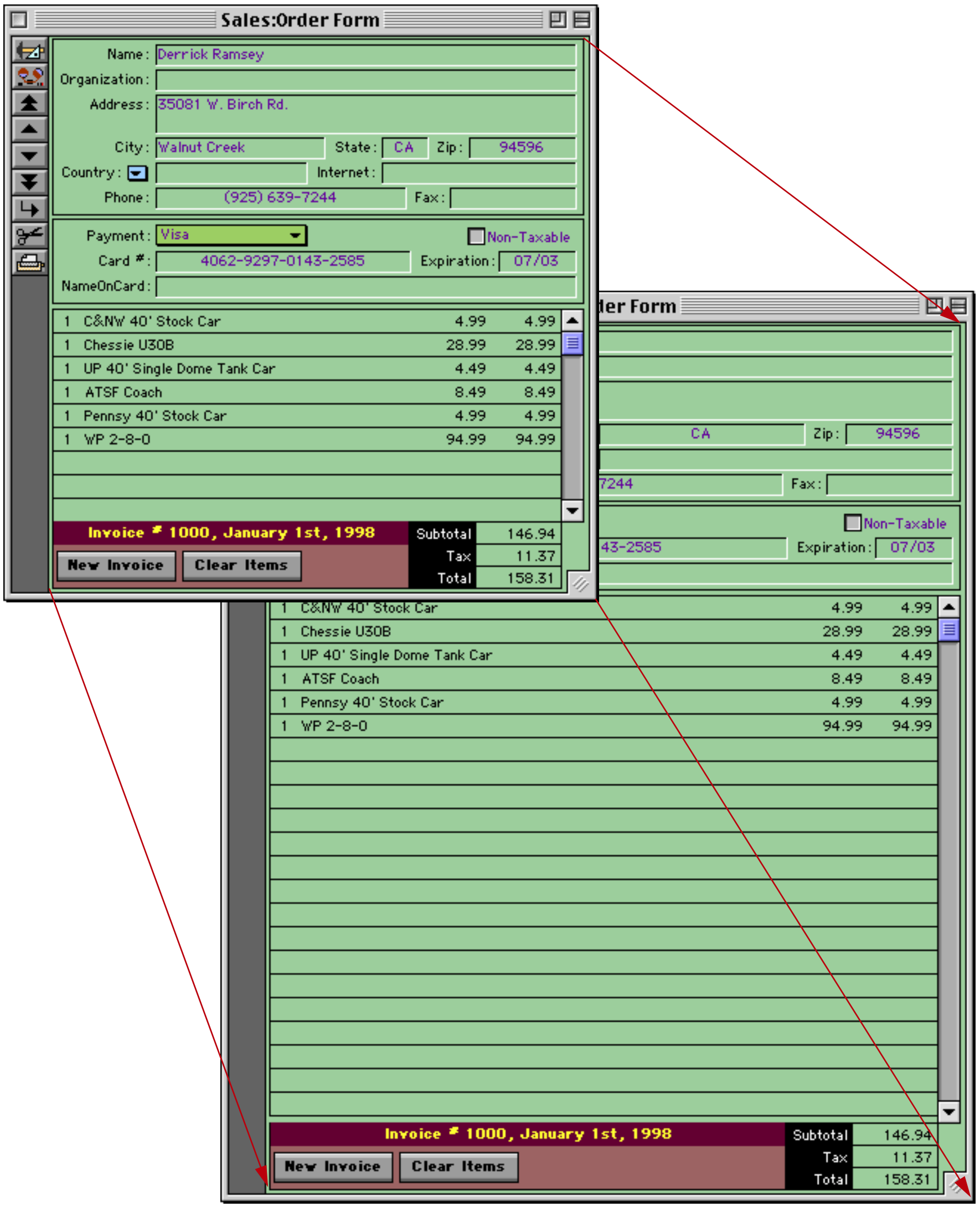
View-As-List Forms

Panorama allows you to display forms as separate pages, or as a continuous sheet (see “View-As-List Forms” on page 889).



Elastic Forms

Elastic forms adjust intelligently when the window containing the form is resized or zoomed. When the form is designed, you decide how the individual elements will expand or shift as the form changes size (see “Elastic Forms” on page 912).



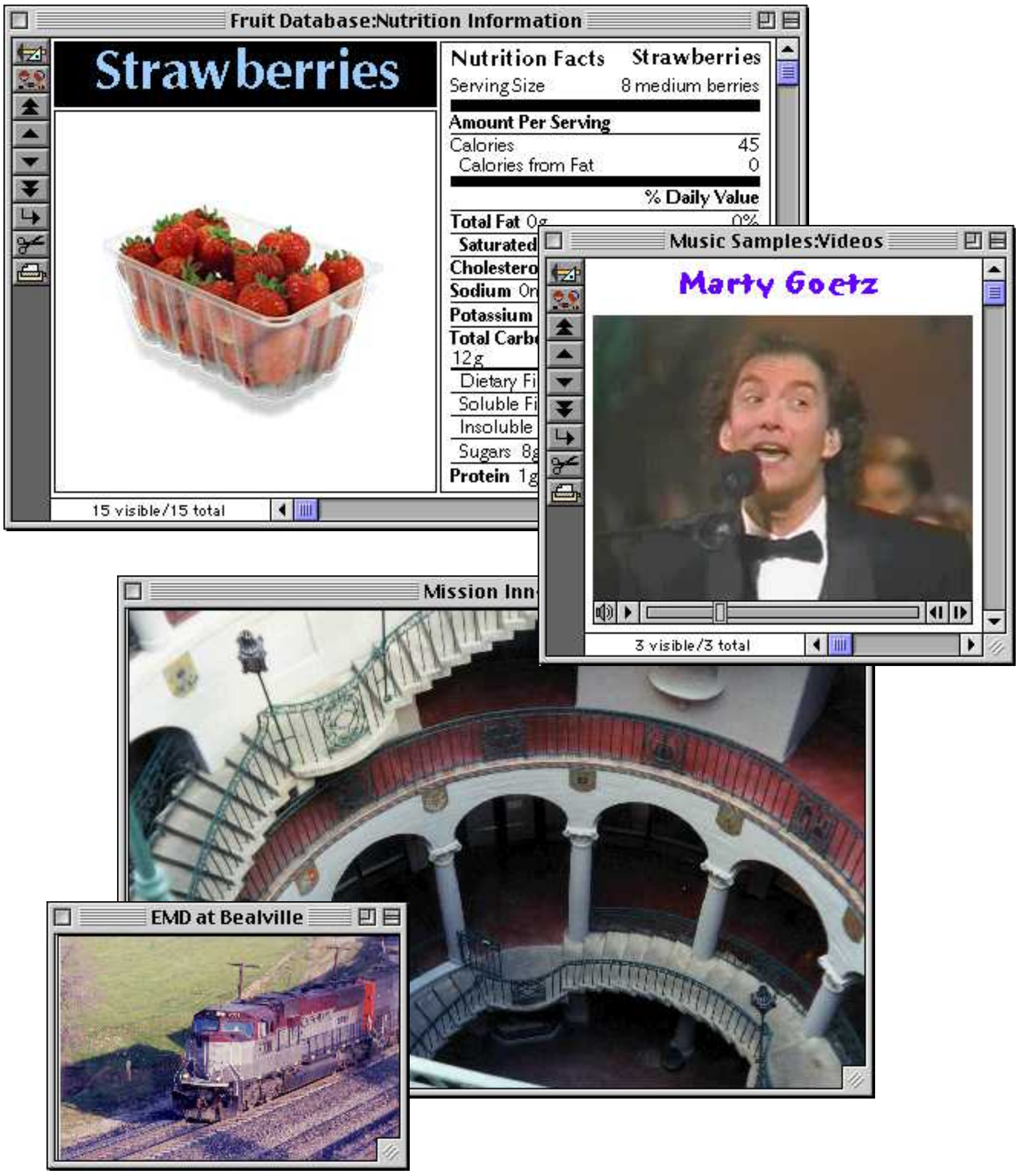
Matrix Display

Panorama matrix object makes it easy to create repeating grids including calendars, catalogs and invoices (see "Super Matrix Objects" on page 929).



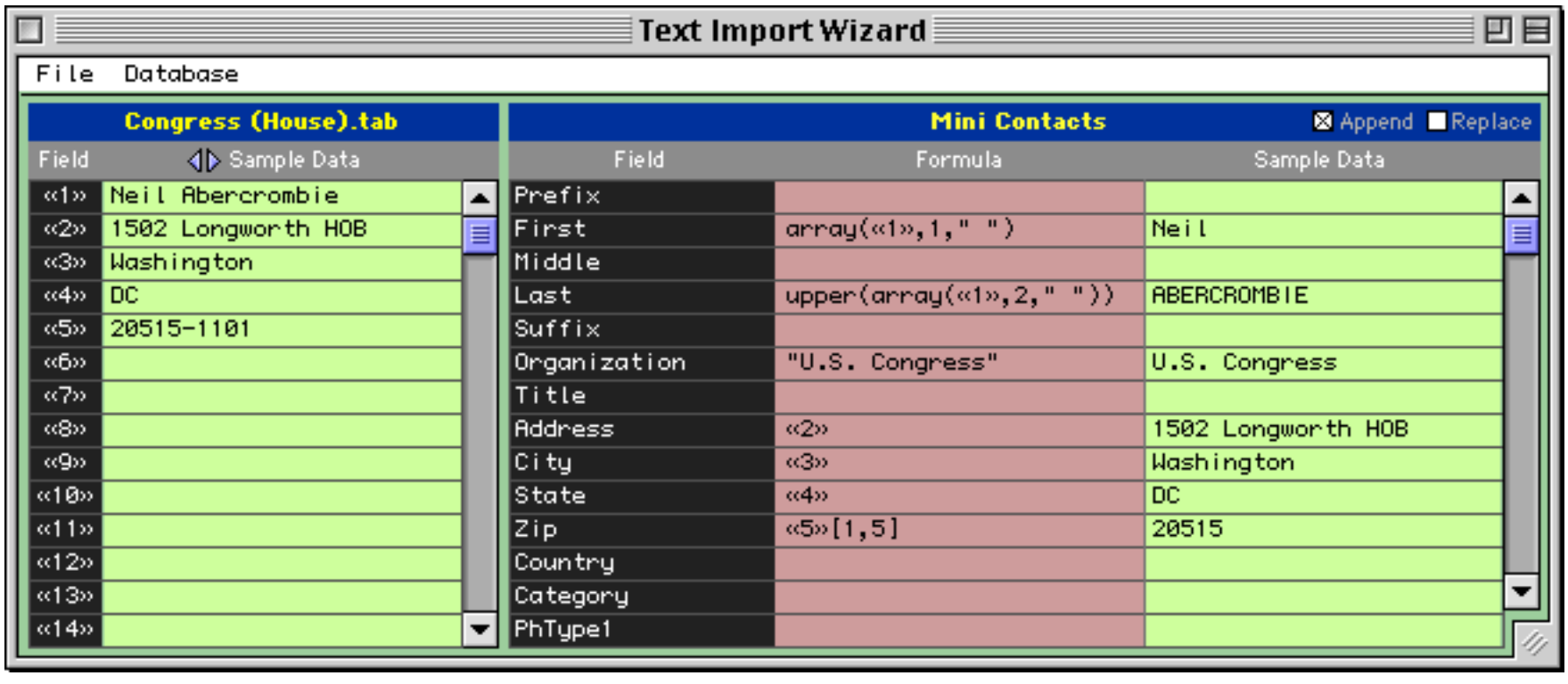
Images and Movies

A Panorama form can display images and movies from a wide variety of sources (see “Images & Movies” on page 739). An image may be fixed (for example a logo or background) or variable (changing from record to record - for example personnel photos or maps associated with individual records). Variable images may be included in the database or (more commonly) displayed directly from files on the disk. With the optional enhanced image pack Panorama can display nearly two dozen different image formats, including JPEG, TIFF, PNG, PCX and TARGA.



High Speed Import

Panorama can import data at rates approaching 1000 records per second. Importing data from mainframe or minicomputer systems is fast and convenient (see “Importing a Text File” on page 145).



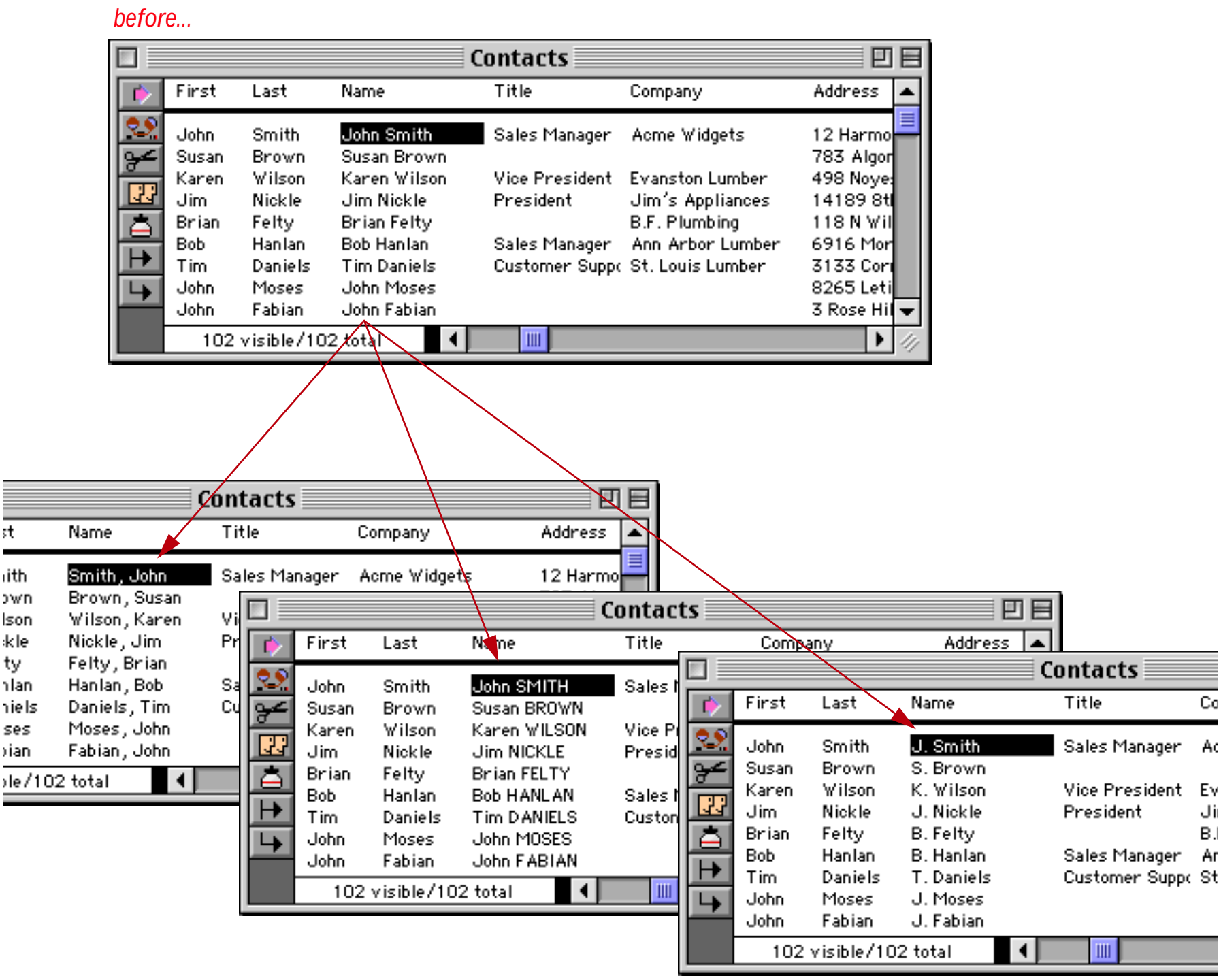
Flexible Text Export (including HTML)

Panorama can export text files in tab separated, comma separated or HTML table formats. When exporting HTML you can choose fonts, colors, column widths and titles (see “Exporting a Text File” on page 165).



Data Transformation

Panorama can quickly transform large amounts of data. Examples include re-arranging characters or words, capitalizing, and transformations based on patterns in the data (see "Transforming Selected Data" on page 451).



Select/Remove Duplicates

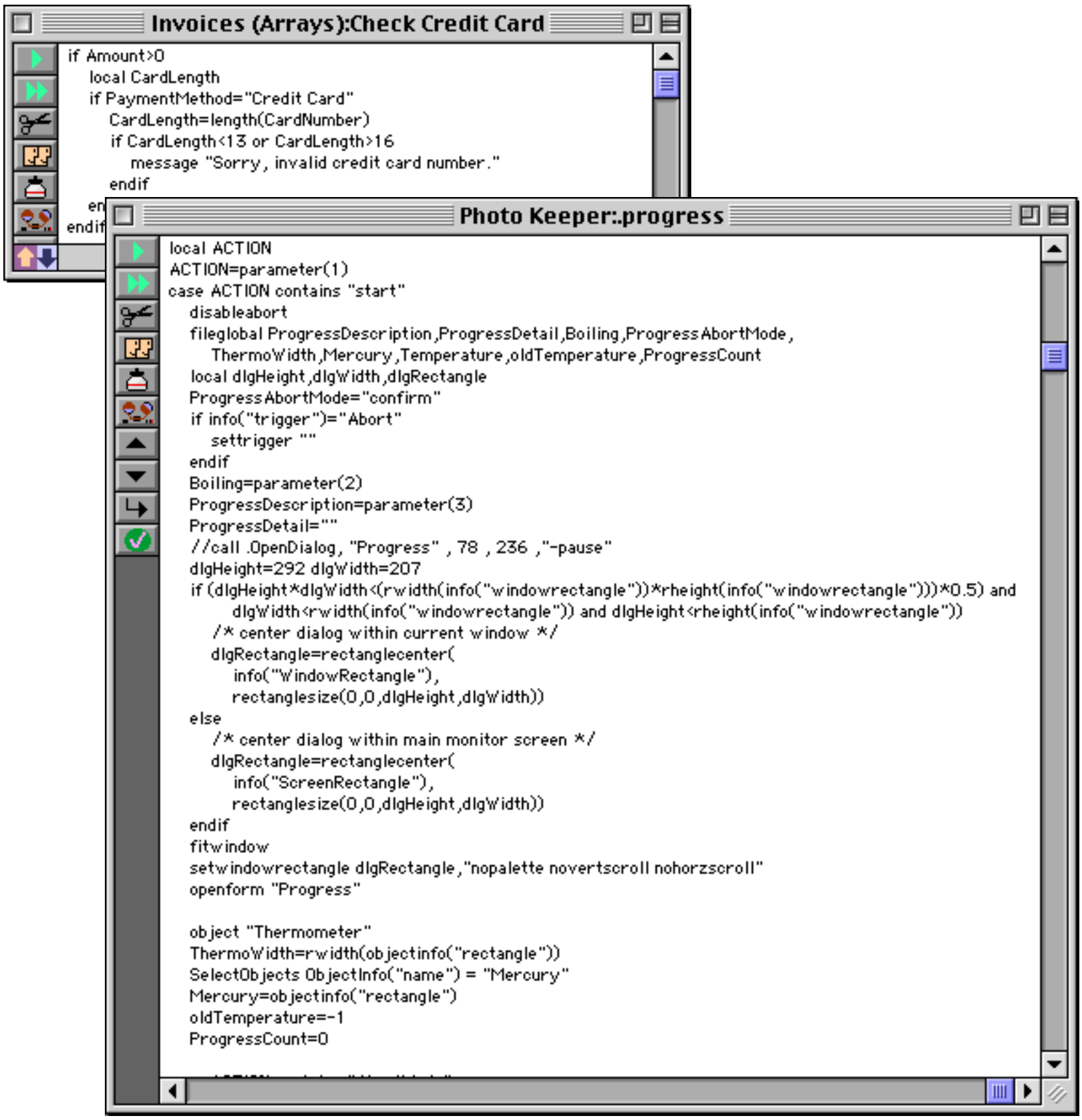
Scan even the largest databases for duplicate information with the **Select Duplicates** command (see "Select Duplicates" on page 388). Duplicates can also be removed automatically based on rules set up in advance (see "Using UnPropagate to Eliminate Duplicates" on page 470).

Client/Server

Panorama introduces a whole new dimension in client/server database management. Instead of a "dumb" client that simply displays forms and allows data to be edited, our **Partner/Server**™ system combines the best of Panorama's incredibly fast single user RAM based database technology with an industry standard SQL server for co-ordinating data sharing across multiple computers. (This feature requires an optional SQL server, sold separately and currently available only for MacOS based networks). The Client/Server system is documented in the separate Panorama Partner/Server Handbook, which is included with your optional SQL server software.

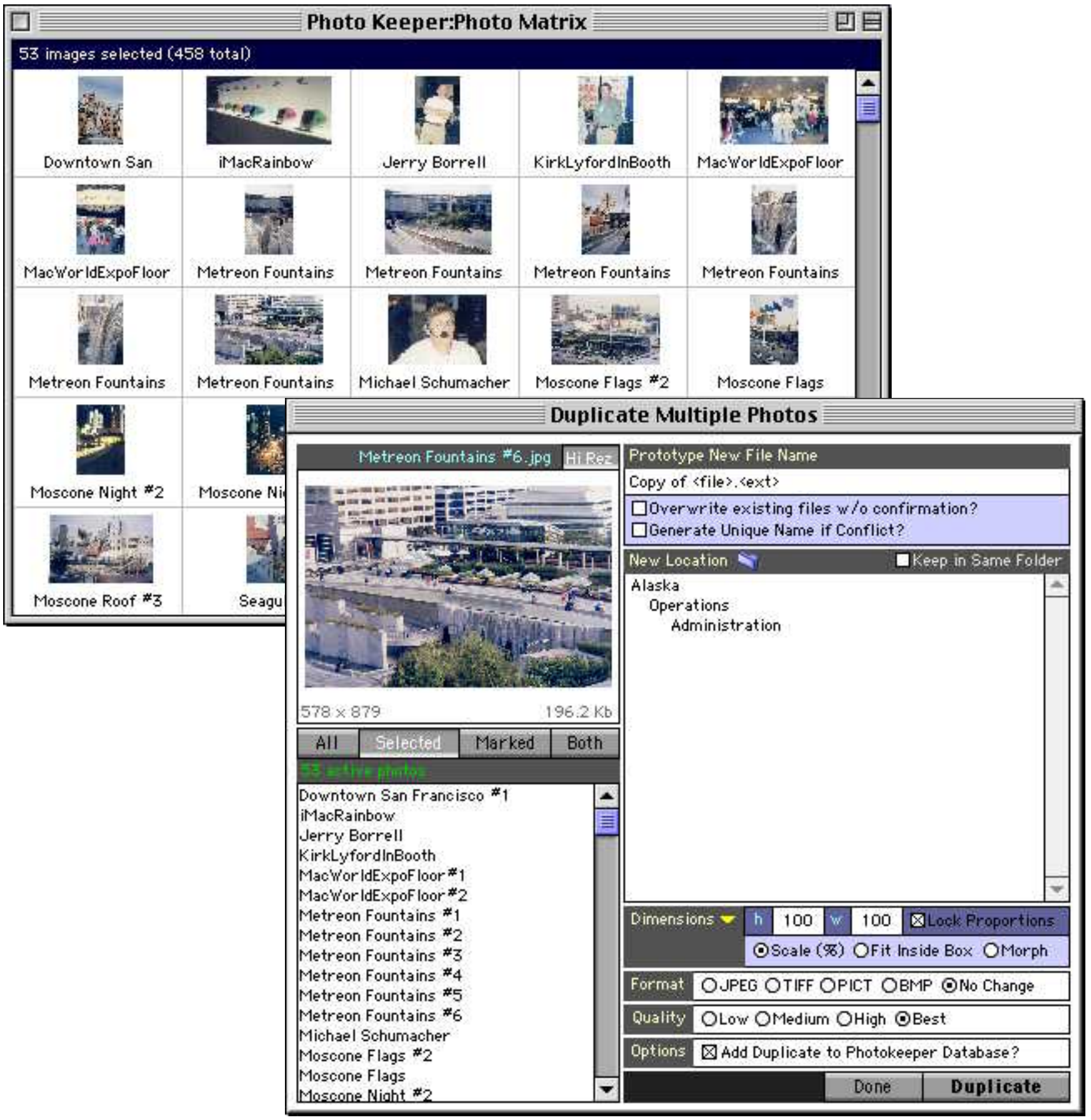
Complete Programming Language and Development Tools

If Panorama doesn't have a feature already, you can add it yourself with Panorama's built in programming language (see "Writing a Procedure from Scratch" on page 1352). Panorama includes powerful programming features like variables (both local and global), if-then-else, case switching, loops, subroutines and a built in interactive debugger. You can even extend Panorama's programming language with your own custom statements and functions.



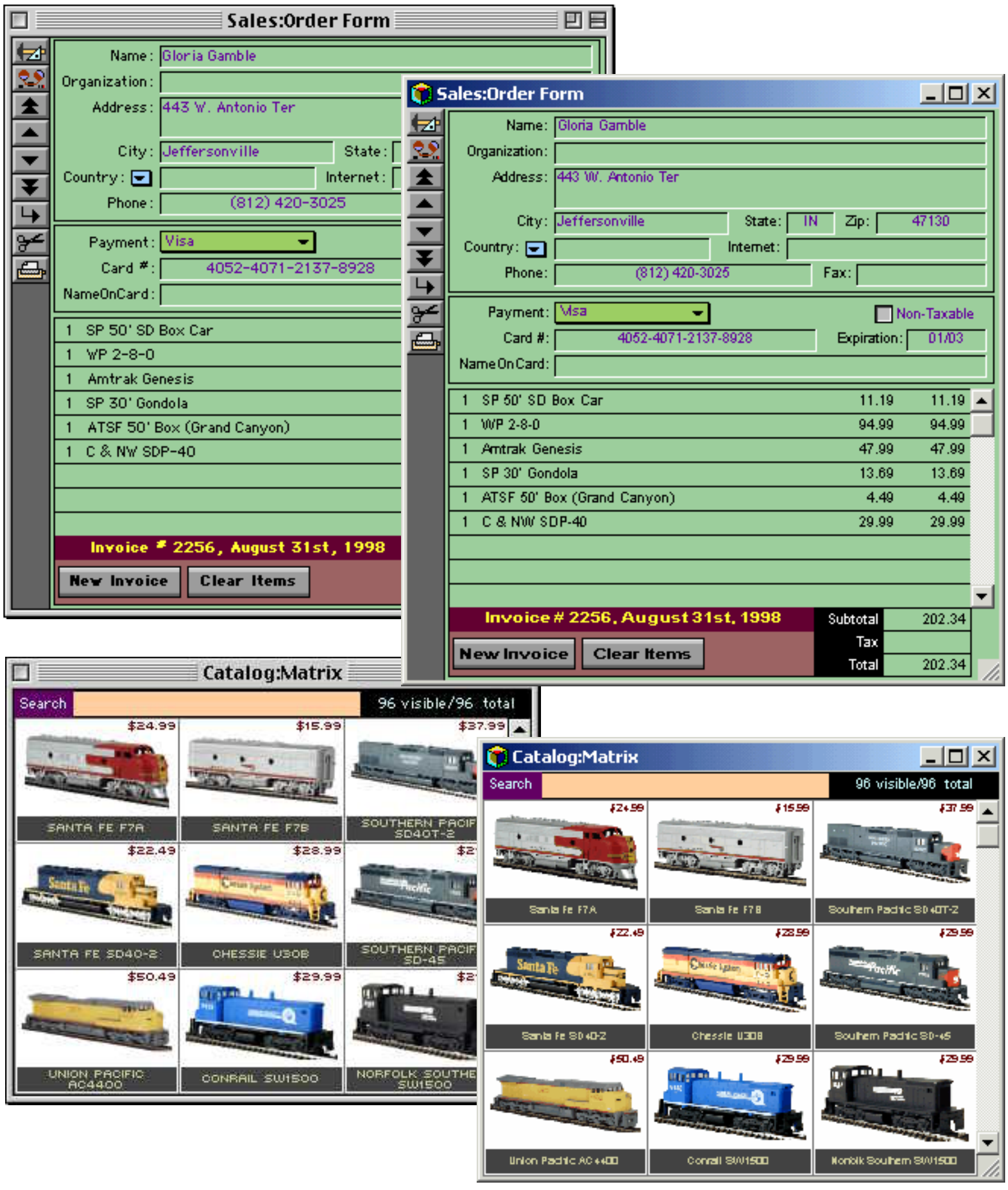
Custom Menus, Buttons, and Dialogs

Panorama gives you the capability to create your own custom menus (see “The Action Menu” on page 1458 and “Custom Menus (Resource Based)” on page 1478), buttons (see “Buttons & Widgets” on page 813) and dialogs (see “Dialogs” on page 1607), making professional quality custom applications possible.



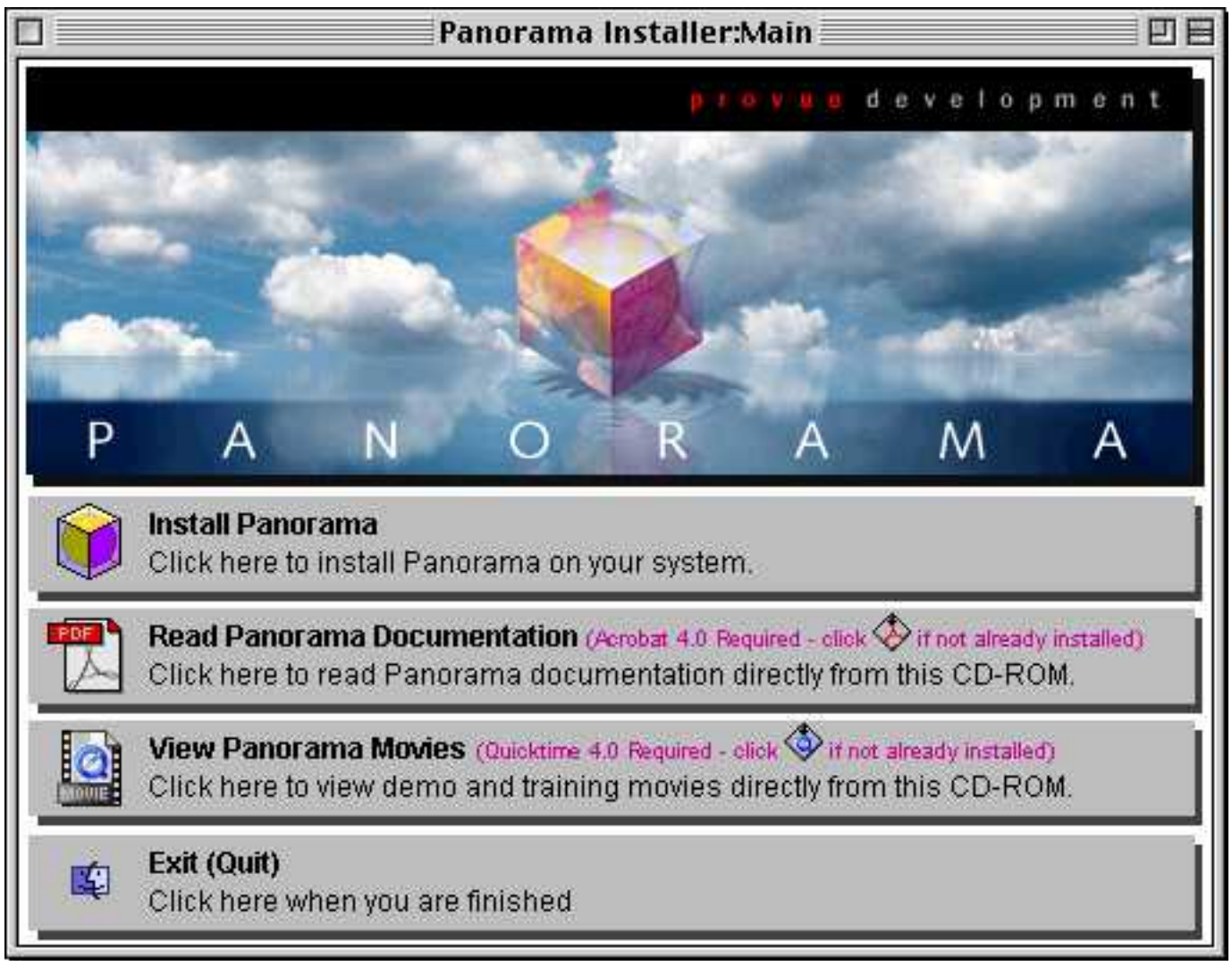
Seamless Cross Platform Operation

Panorama databases are 100% compatible with both the Windows or MacOS platforms and may be transferred back and forth freely between the two platforms. You can even share databases on a single server across a cross platform network.

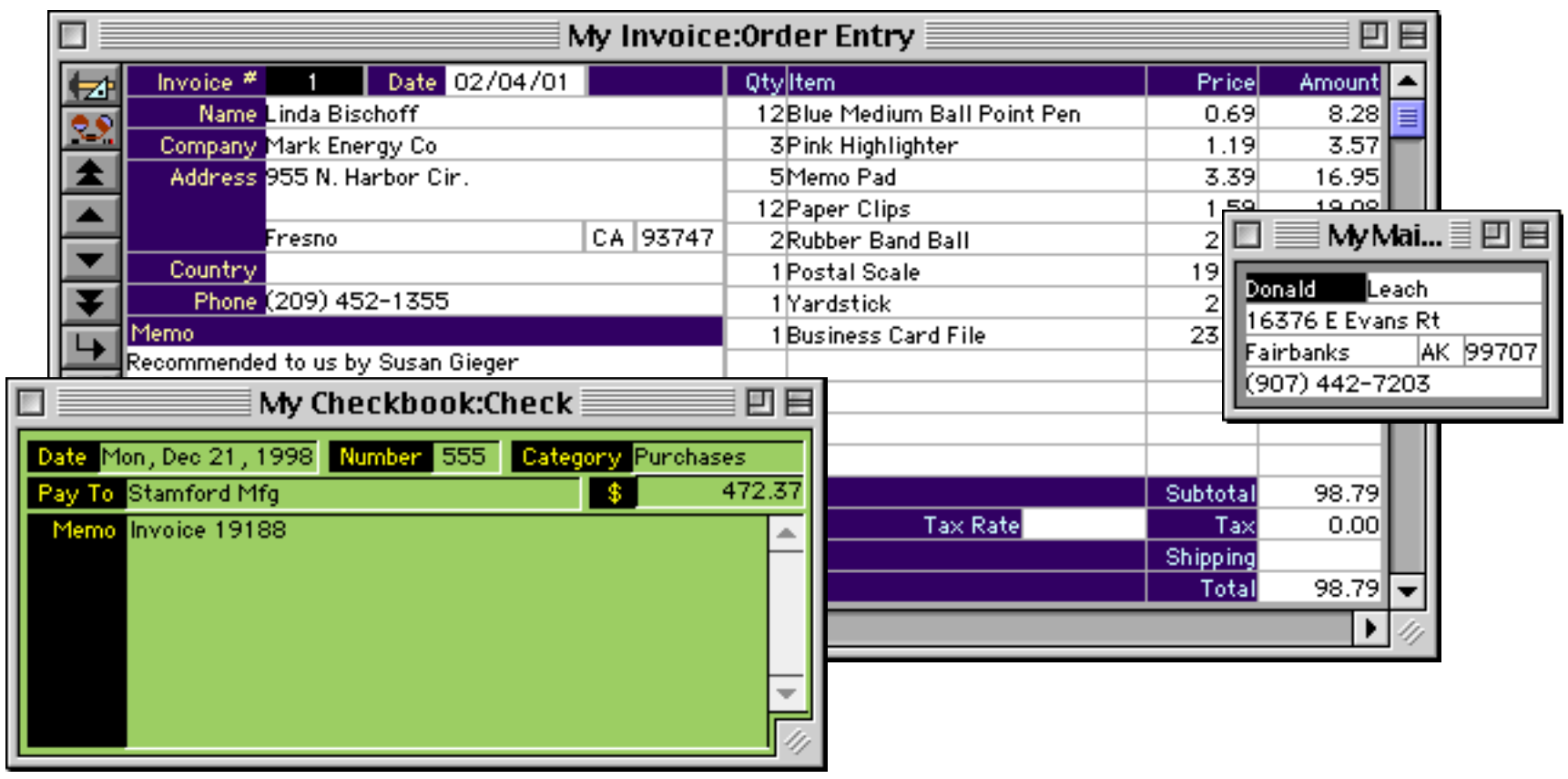


Getting Started With Panorama For First Time Users

If you haven't done so already, start by installing Panorama on your system (see "Installation & Activation on page 89").



As a newcomer to Panorama we recommend that you start with the step-by-step tutorials that walk you through the creation and use of four typical databases — a mailing list, a checkbook, an invoice and a price list.



You can either watch a movie of these tutorials (see "Once you have clicked on all of the blue bold characters you can press the Continue button to continue with your work." on page 106) and / or view them in PDF format (see "Step-by-Step Tutorials" on page 1). We've also included finished versions of these databases that you can use for learning and inspiration. As you build and use these databases you'll acquire the skills you'll need to develop custom database applications for your own business, organization, school, hobby or household.

As you become more involved with Panorama you may want to take advantage of some of the additional resources available, including technical support (see "Technical Support" on page 1881) and various e-mail discussion groups (see "On-Line Resources" on page 1877). You may even want to attend one of the periodic conferences we hold here in Los Angeles (see "Panorama Conferences" on page 1882).

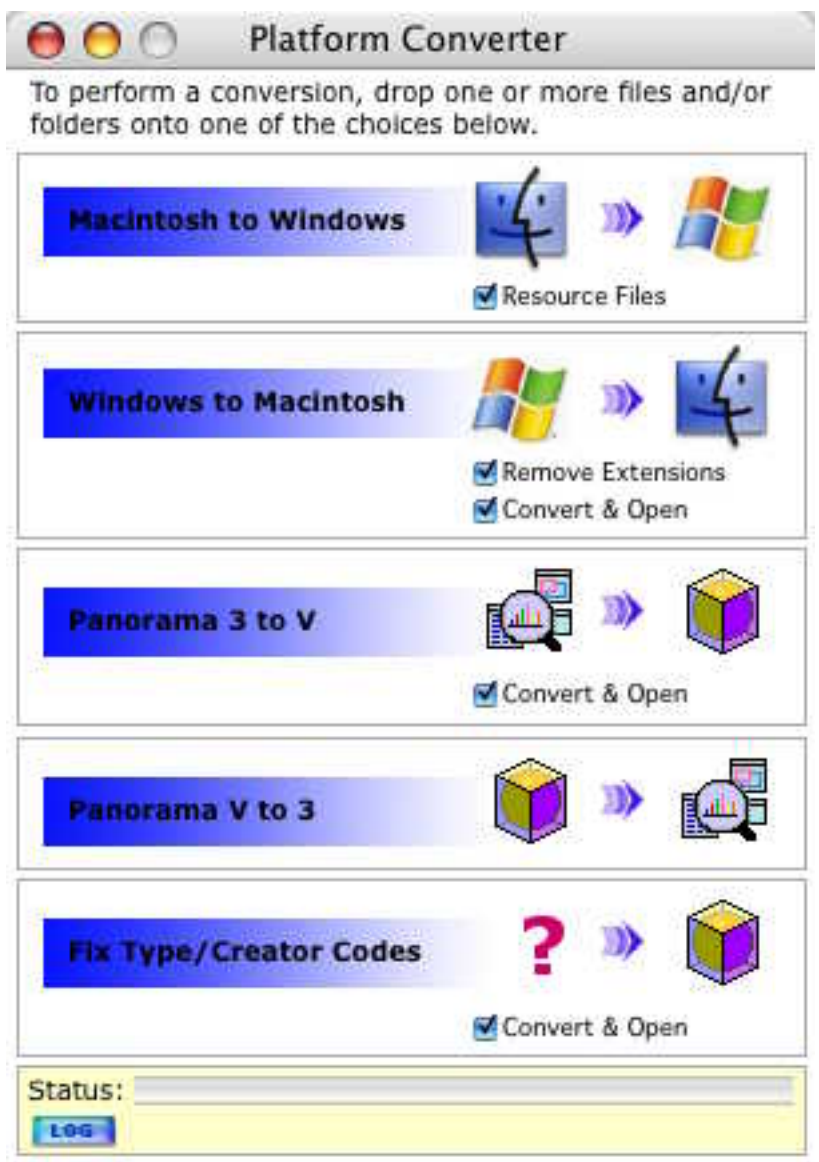
Upgrading to Panorama V From an Earlier Version

If you haven't done so already, start by installing Panorama on your system (see "Installation & Activation" on page 89). The activation / registration process is quite different from previous versions, so you may want to read this material even though you've installed Panorama before. (By the way, you can leave your old version of Panorama on your hard drive if you want to — you can even use both the old and new versions at the same time (not with the same database at the same time, of course)!)

To help you quickly learn about what's has changed since the version of Panorama you are familiar with we have prepared a summary of the changes in each version (see "History of Panorama" on page 1). The summary has links to the actual topics within the body of the manual so that you can quickly focus on the new material important to you. (If you are upgrading from Panorama 3 or earlier you may want to review the entire manual anyway at some point, because many topics have been rewritten and expanded to make them clearer than in earlier manuals. For example, the section on the Word Processor SuperObject has been expanded from 3 to 57 pages!)

Switching From Macintosh to Windows

Panorama has been designed to be extremely compatible between Macintosh and Windows PC systems. Over 99.9% of all databases created on the Macintosh work perfectly on Windows systems without modifications. If you are transitioning from the Macintosh to a Windows or cross platform environment, we've prepared a special supplement to help you. See "Cross Platform Databases" on page 1855 to learn more.



Deploying Applications Built With Panorama

ProVUE has two options for affordable deployment of Panorama based applications — **Panorama Direct** and **Panorama Engine**. Panorama Direct is designed to be used as a "run-time" engine that allows the economical deployment of database applications created with the full version of Panorama. Panorama Direct prices start at under $100 and is available in economical 3, 6, 12, 25 and 50 packs for as little as $30 per seat. The Panorama Engine allows unlimited royalty free distribution of a Panorama database for a one time fee.

Panorama Direct

Panorama Direct is a limited, low cost version of Panorama that can run any database created with the full version of Panorama. Database files can be exchanged back and forth between full Panorama and Panorama Direct with no conversion. The Panorama Direct user can enter and modify data in the database, sort, select, print and run procedures. The user can even make minor adjustments to forms and reports. However the user will not be able to examine, create or modify procedures, create or modify SuperObjects within a form, modify cross tab set up, set up or modify charts, etc. (If desired, the database creator can lock down any database to prevent anyone from modifying forms, reports, procedures, etc., whether they are using Panorama Direct or a full copy of Panorama.)

Panorama Direct is ideal for distributing pre-built databases either in-house or around the world. Many companies save money by purchasing full copies of Panorama only for users that actually set up and design databases, while purchasing Panorama Direct for data entry positions. Panorama Direct is fully compatible with the Panorama SQL Server and may be used as a client with other copies of Panorama and/or Panorama Direct. In addition, a growing number of third-party developers use Panorama Direct to distribute commercial applications developed in Panorama. Panorama Direct makes it possible for developers to price their products aggressively while retaining all the benefits of rapid application development with Panorama.

Panorama Engine

Panorama is normally sold on a "per-machine" basis. Once you have purchased a copy of Panorama or Panorama Direct, you are free to use it on an unlimited basis on a single machine. On that single machine you can create as many databases as you like, and change the design of any database any way you like at any time.

The **Panorama Engine** introduces a new way to purchase Panorama's outstanding capabilities. Instead of purchasing unlimited use of Panorama on a single machine, you can now purchase an unlimited "run-time" distribution license for a single database (or collection of databases). Once you have purchased this unlimited distribution license you can distribute as many copies of your database (or collection of databases) as you like, without having to purchase Panorama or Panorama Direct for each machine. The one-time license fee is very reasonable, and for shareware authors, almost zero!

The Panorama Engine Licensing Process

The first step in the licensing process is to create your database or collection of databases. Once this is complete, contact ProVUE by phone or e-mail to purchase your unlimited distribution license. Once the license is purchased, you will be instructed to send your files to:

```
license@provue.com
```

ProVUE will process your files to add the unlimited distribution license "tag" to each file. This tag enables the database for unlimited royalty-free distribution. The modified files will be e-mailed back to you.

Distributing Your Databases

A database with the unlimited distribution license "tag" can run on any computer with the Panorama Engine installed. The Panorama Engine is simply an ordinary copy of Panorama that has not been activated yet (see "Using Panorama's "Demo Mode"" on page 105). (Your database will also work with ordinary copies of Panorama.) You can distribute the Panorama Engine yourself, or you can tell your users to download the Panorama Engine from our web page.

Panorama Engine Restrictions

When used with a database with the unlimited distribution tag, the Panorama Engine can perform almost all Panorama operations, including saving and printing. In other words, the Panorama Engine is not a demo - you can build real working applications. There is absolutely no limit to database size (other than the amount of RAM available) or limitations on data entry or saving data. However, there are some operations that are not available when running databases with the Panorama Engine.

Cannot create new databases (either with Open dialog or Save As command)
Cannot add fields, remove fields, change field attributes, or make any other database structure changes
Cannot modify forms or open graphics mode

Cannot modify or look at procedures
Cannot connect to SQL server (Partner / Server databases require Panorama or Panorama Direct)
Cannot save an "untitled" database
Cannot import with Open dialog (can import using a pre-defined procedure)
Cannot export with Save as dialog (can export using a pre-defined procedure)
Cannot manually change Save As dialog options

If a database does not have the unlimited distribution tag, it will not run in demo mode if the software is not activated (see “Using Panorama’s “Demo Mode”” on page 105).

New Database Versions

The unlimited distribution license is for one version of your database only. You are allowed to make minor "bug fix" changes to existing forms and reports (a full copy of Panorama is required to make these changes). However, if you want to create a completely new version of your database with new features, you will need to re-license the new version of the database for unlimited distribution. Performing any of the following modifications to your database will invalidate the unlimited distribution license.

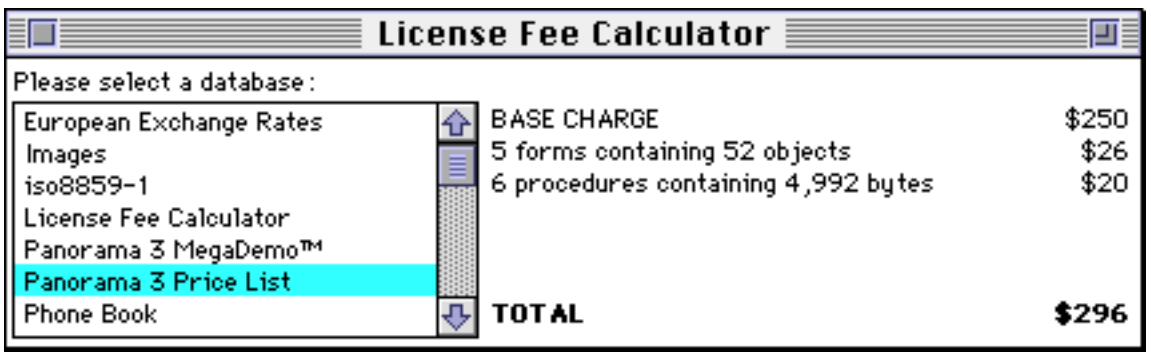
Adding or removing fields
Adding or removing forms
Adding or removing procedures
Adding or removing crosstabs

If you make any of these changes to a database, Panorama automatically invalidates the unlimited distribution tag.

License Fees (Commercial)

For commercial distribution, the license fee depends on the complexity of your database. The base fee is $250 (a database with data sheet only). Additional charges are added as you add forms and procedures.

To calculate the exact charges for any database, use the License Fee Calculator database included on the CD-ROM (use the **Favorite Databases Wizard** to locate it). Here is an example of an actual license fee calculation. (Note: ProVUE reserves the right to change the license fee schedule at any time.)



The calculation shown above is for a single database. If your application has multiple files, the total charge is the sum of the charges for the individual databases.

License Fees (Shareware/Freeware)

For shareware / freeware distribution, the license fee is a flat $25 per database. To qualify as shareware / freeware your software must meet the following qualifications.

You must supply a concise (100 words or less) description of your package.
You must supply complete documentation in an on-line format.
You must agree to allow ProVUE to distribute your shareware / freeware package and documentation via our web site, bulletin board, CD-ROM's or any other method we find appropriate.

Panorama Engine vs. Panorama Direct

Panorama developers now have two ways to deploy a Panorama database: Panorama Direct or the Panorama Engine. How do you decide which to use for your project?

If you are planning to distribute your database to hundreds (or thousands) of users, then the unlimited distribution license with the Panorama Engine is the way to go. The Panorama Engine makes it possible to create low cost mass-market applications using Panorama technology.

If you are planning to distribute your database to a small group of users, Panorama Direct is a better approach. Panorama Direct is also more appropriate if you plan to change the design of your database frequently, or if your application requires features not included in the Panorama Engine (SQL, graphics mode, etc.).

Tips for Using This Documentation



This online PDF manual was designed both to be used on screen with the Adobe Acrobat Reader and to be printed. We've taken several steps to help make this manual more on screen friendly. First of all, we expanded the page size and all fonts by 25% to make everything more readable. With only a couple of exceptions, all screen shot illustrations are included at 100% of their original size so all of the details are sharp and clear at the same magnification used when reading the text. You won't have to zoom in and out as you often have to do when reading PDF documents, you can simply leave the magnification at 100% at all times. To make navigation easier we've included an outline and we've also enabled Acrobat's search feature.

Why So Many Pages?

Did you notice that this book has a LOT of pages? There's really both more and less than meets the eye. The primary reason why there are so many pages is because this book was designed primarily for on-screen use, not for printing. When writing a printed book the author always has the final page count in mind. Each page costs money! So as you write you are constantly thinking — do I really need a more detailed explanation here? Can I eliminate some or all of these screen shots? Check out the Adobe Photoshop manual — this is a very visual program and yet there is only one small screen shot every few pages. Keeping the number of pages affordable (both for printing and shipping) often becomes more important than clarity or comprehension.

When we started this book we decided not to think that way. Extra pages in a PDF file are essentially free, so we took full advantage of this new opportunity. There are literally thousands of screen shots, all full size for maximum clarity. This probably doubles the page count compared to a typical printed manual. We also used an extra large (14 point) font size for maximum readability, which also increases the page count. Every topic is covered in depth, often with more than one approach discussed. We've included a 900 page reference and a 200 page tutorial. We think the end result is a better experience for you, the end user.

Printing This Book

You may decide that you want to print a portion of the documentation for further study. If you are using a Macintosh you'll need to use the **Page Setup** dialog to set the Scale option to 75%. If you are using a Windows PC computer this is taken care of for you. Either way, the documentation is designed to be printed on standard 8 1/2 by 11 inch paper so that it can easily be bound in a standard three ring binder.

Training Movies

Don't like to read? Join the club. You can get started with Panorama without reading at all! Just sit back and watch the tutorial movies included on your Panorama CD-ROM (see "Panorama Movies" on page 21). Of course these movies don't cover every Panorama feature, so you still probably need to do some reading at some point if you want to get the maximum productivity from your Panorama investment. Nevertheless, these movies can get you up and running quickly with the primary skills you need.

Typographical Conventions

Throughout this manual we've used different type styles to indicate special meanings within the text. This table shows the different styles and their meanings.

Example	Meaning	Description
Page Setup	Menu	This style indicates the text is a menu command.
OK	Button	This style indicates the text is the name of a button, usually in a dialog.
Scale	Option	This style indicates the text is an option, usually in a dialog.
`addrecord`	Code	This style indicates the text is part of a program or formula
Company	Literal	This style indicates the text is a literal value. For example it might be a field name, a typical data value or a number.
Return	Keys	This style indicates the text is the name of a key on the keyboard, for example **Return**, **Enter** or **Tab**.
See "Cross Reference Links" on page 76	Link	This style indicates the text is a link to another section of the manual. You can click on the link to jump to the cross referenced topic.
Wizard	New Term	This style indicates that a new term is being introduced. If the new term is a proper name (for example **Formula Wizard**) we will usually use this style every time it is used, otherwise it is only used the first few times the term is being introduced.

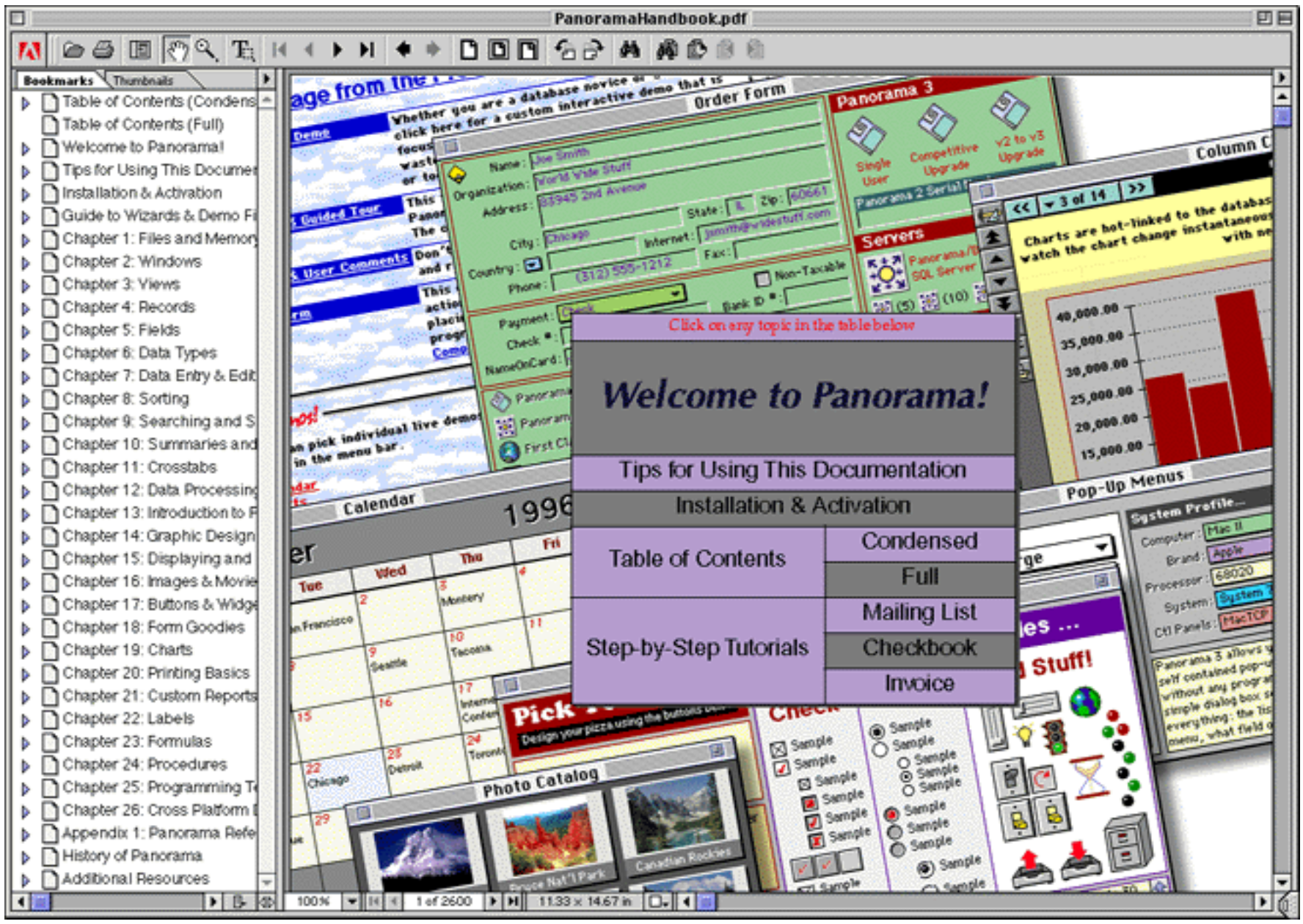
Opening the Documentation

To open the documentation locate and double click on the file PanoramaHandbook.pdf.

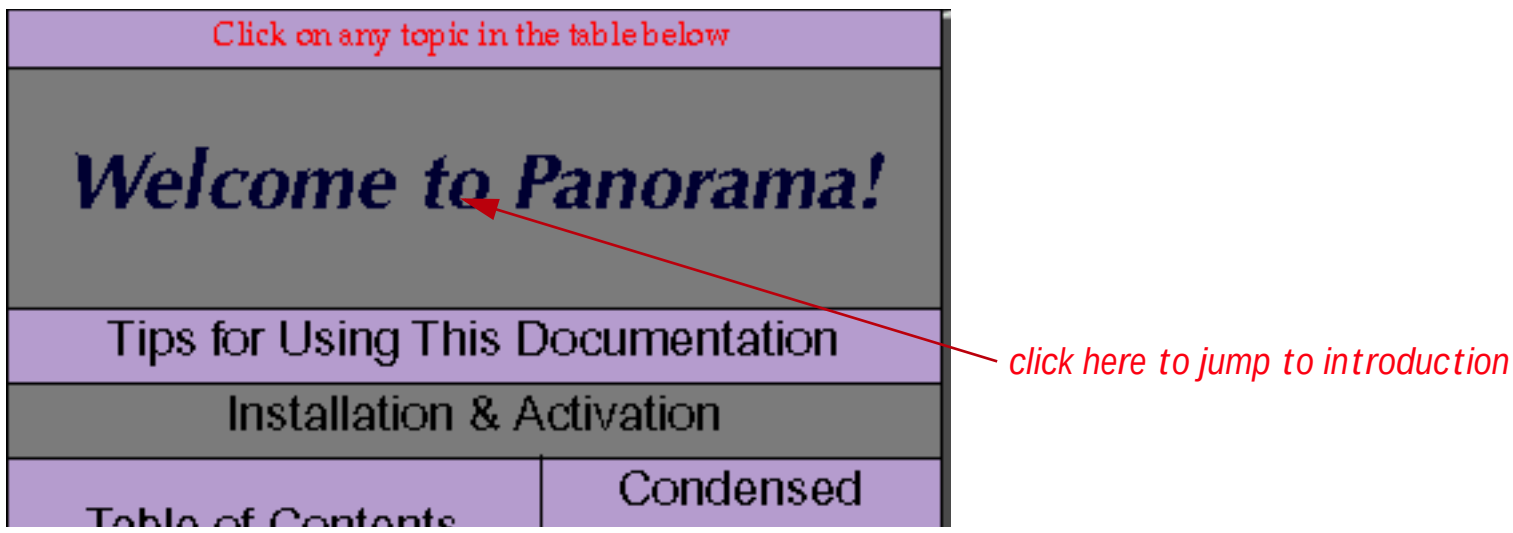


You can also open the documentation directly from the Panorama Wizards menu (in the Documentation submenu).

When you first open the file the first page looks like this.

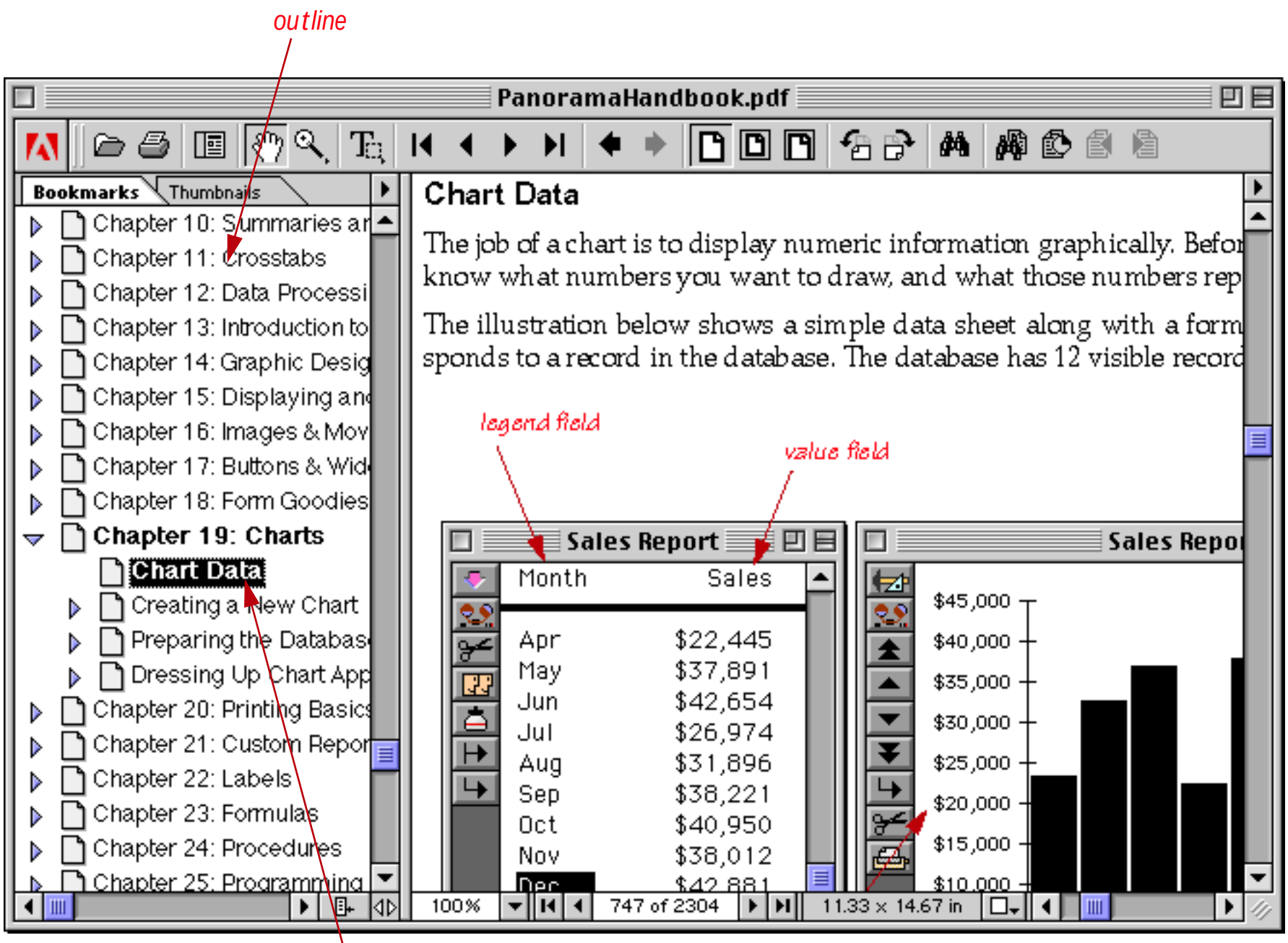


In the center of the page is a table that allows you to quickly jump to major sections of the documentation. For example, if you want to start with the introduction to the manual click on Welcome to Panorama!

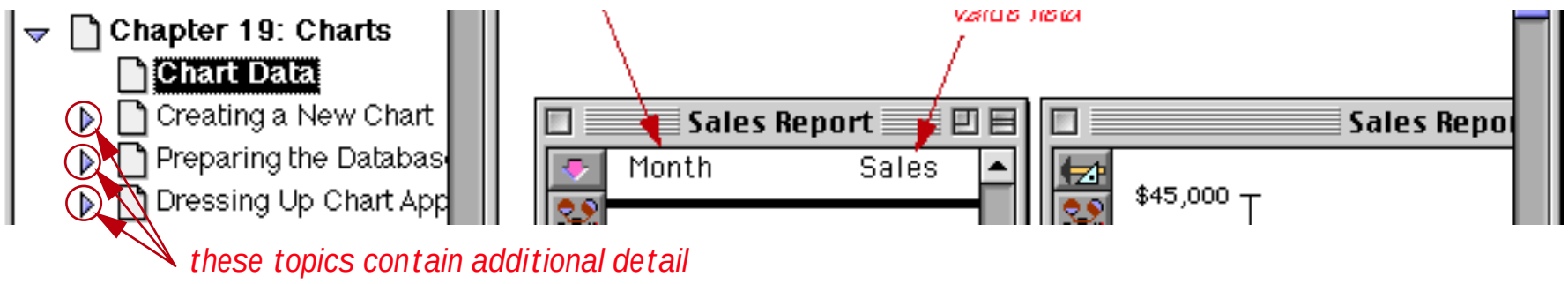


Finding a Topic

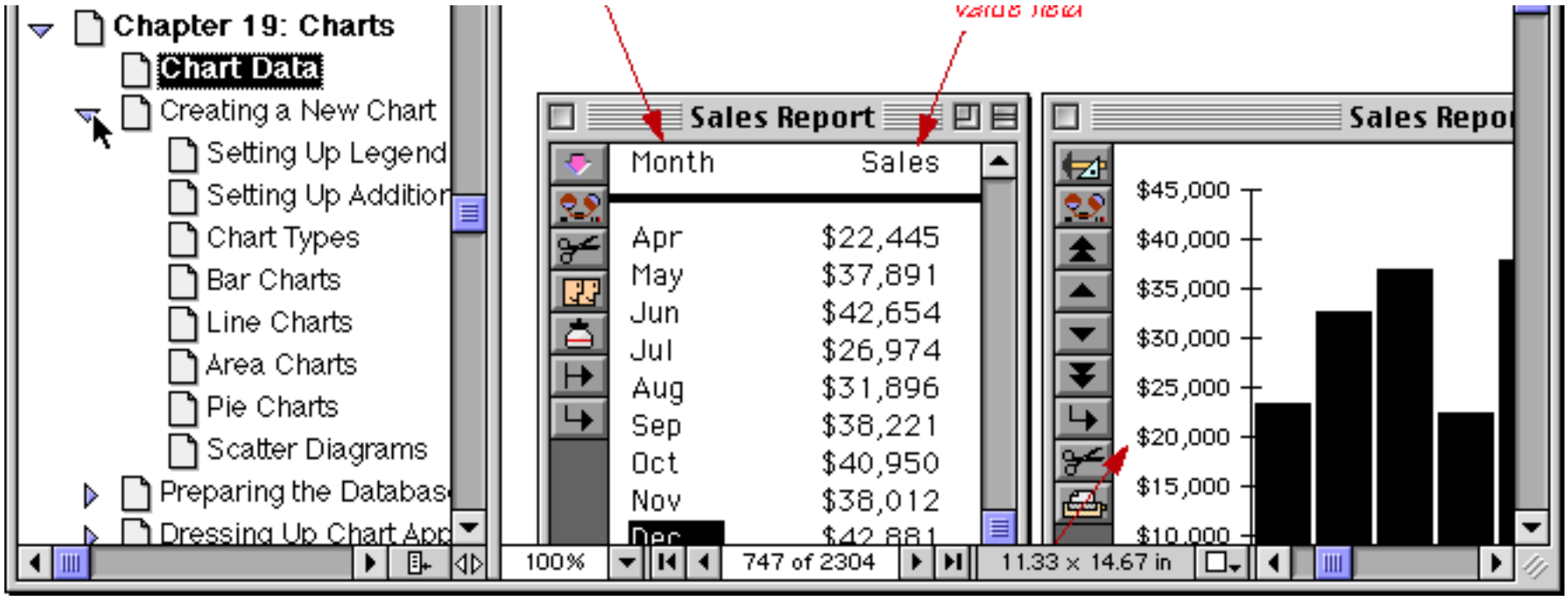
The Panorama Tutorial & Handbook includes several navigation aids to help you find material quickly. The primary navigation aid is the outline on the left hand side of the window. This outline allows you to quickly navigate to any spot in the documentation and also shows you the context of what you are viewing at any time.



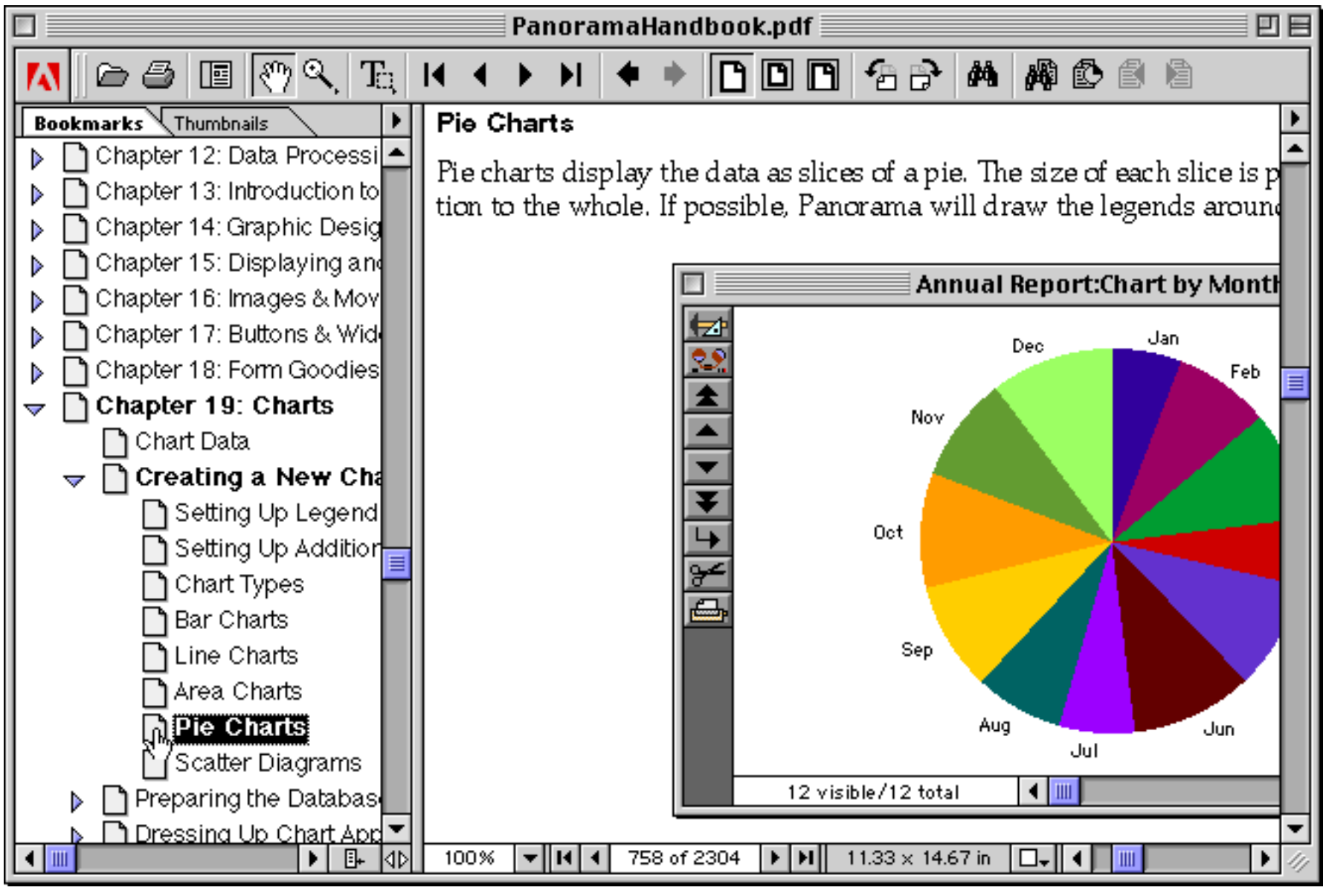
A triangle next to the topic name indicates that the topic has one or more subtopics.



Click on the triangle to reveal the subtopics associated with any topic.

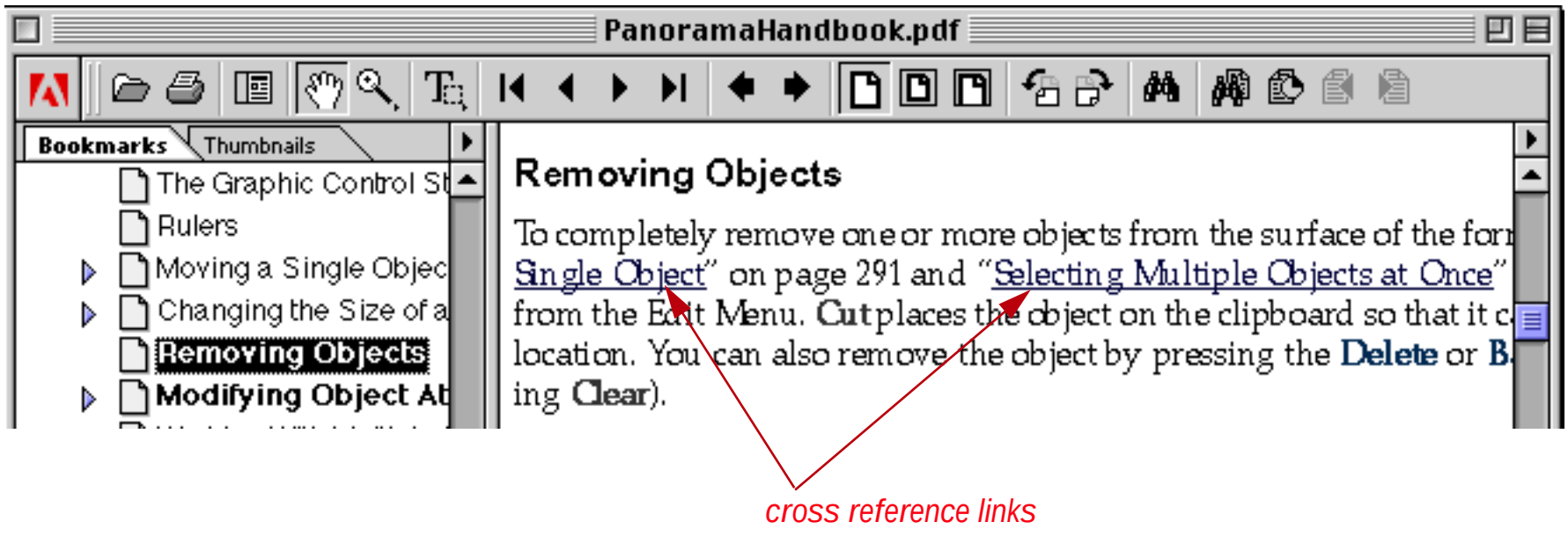


To move directly to any topic simply click on the topic name.



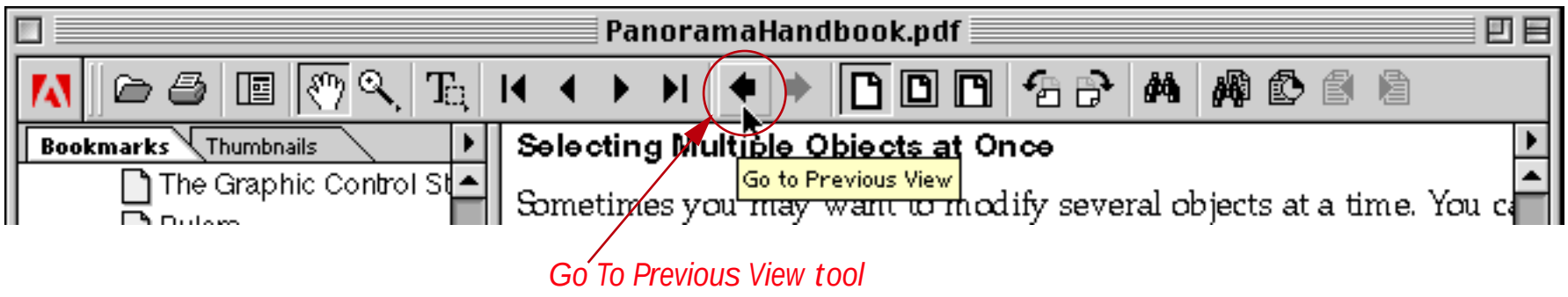
Cross Reference Links

Where appropriate, the documentation includes cross references to other sections of the manual that explain material related to the current subject. These cross references look and act much like links in a web page. Just like most web page links, these cross references appear in blue underlined text.



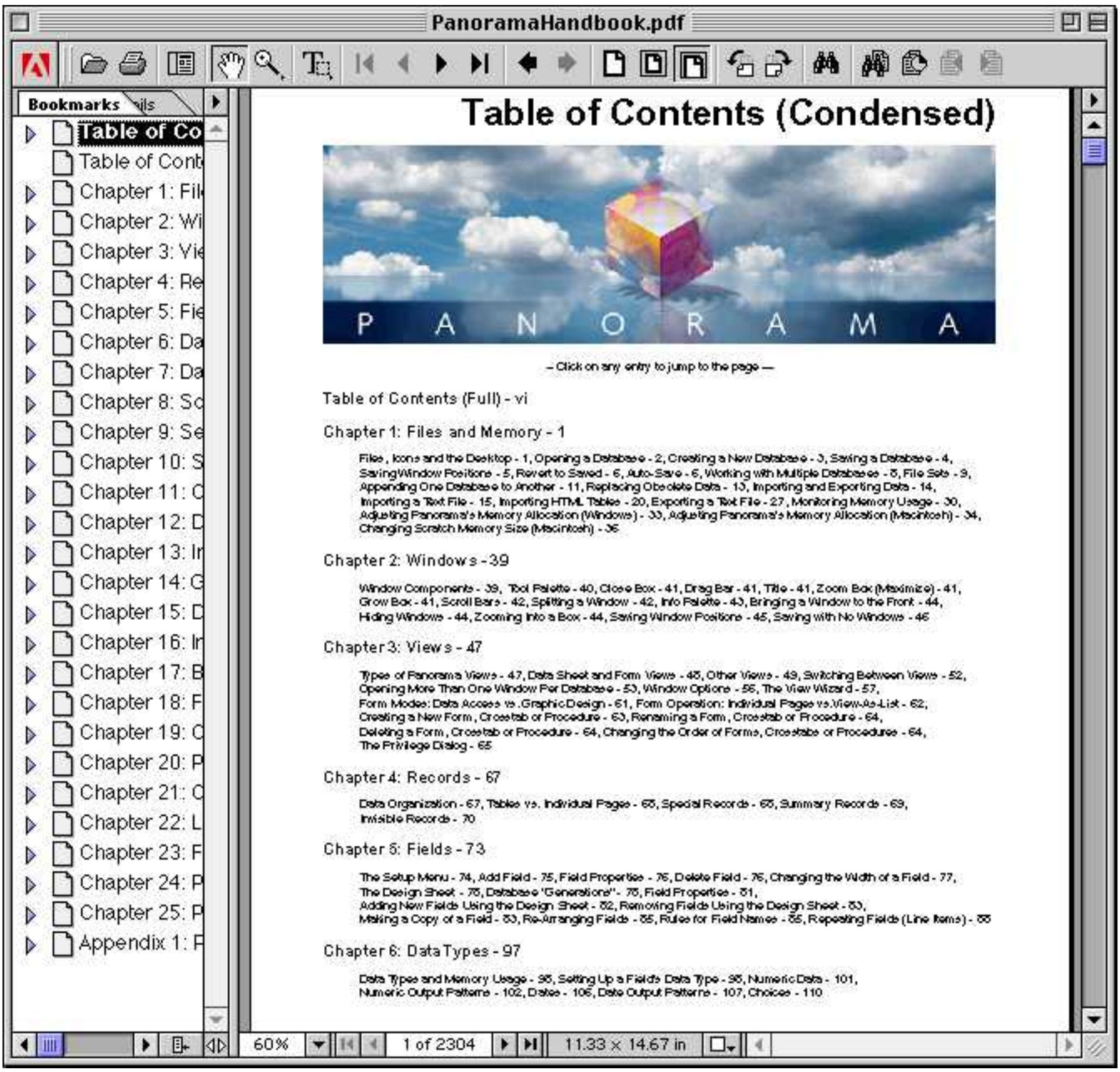
To jump to the cross reference page simply click on the underlined text, just as you would in a web browser. (Unfortunately, Adobe Acrobat does not allow you to open the link in a new window. You can only view one spot in the documentation at a time.) For example, if you click on the link **Selecting Multiple Objects at Once** the page describing this selection technique will appear.

If you want to go back to the original spot you can either select the **Go Back** command from the View menu or you can press the **Go To Previous View** icon in the tool bar.

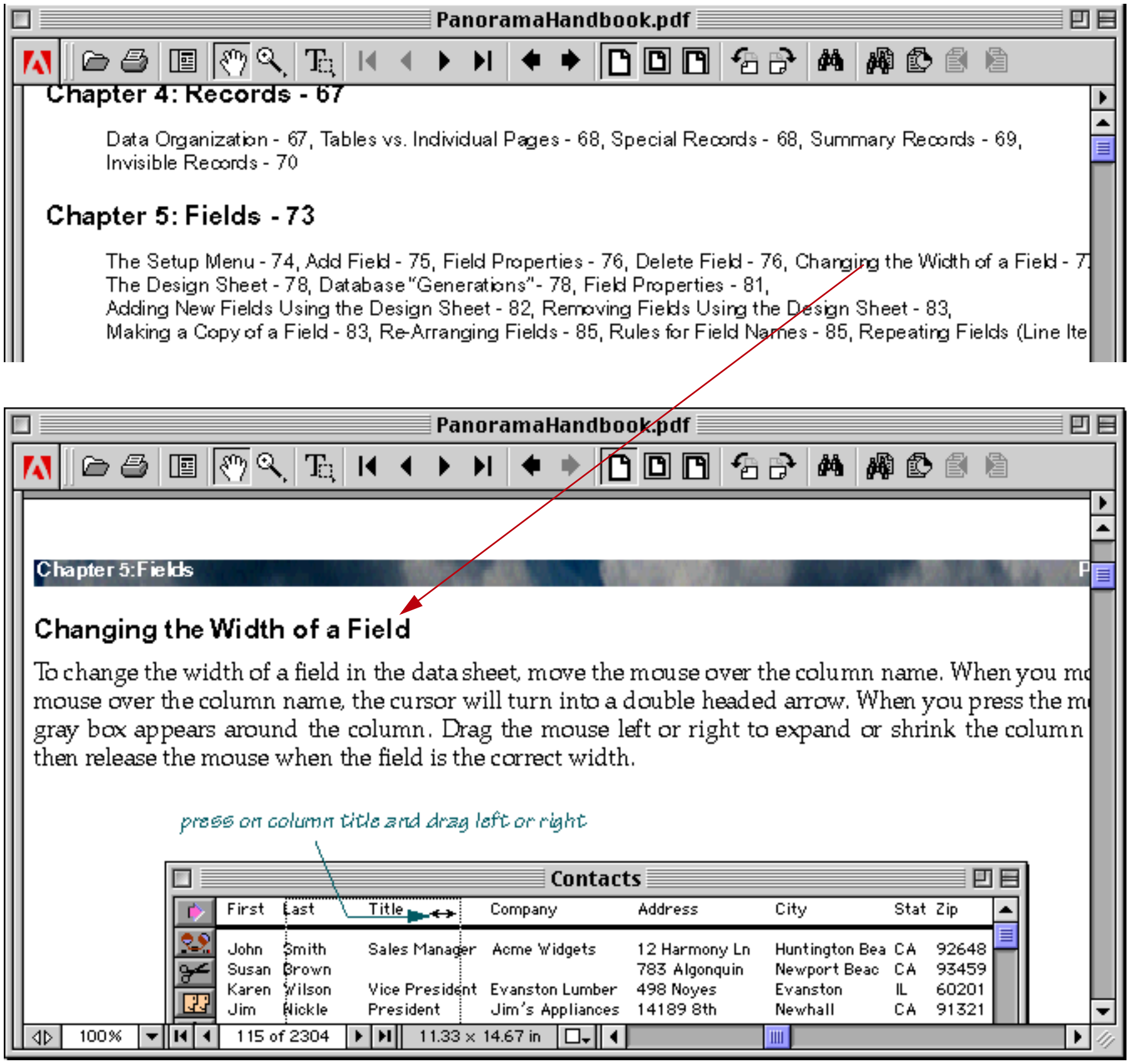


Using the Table of Contents

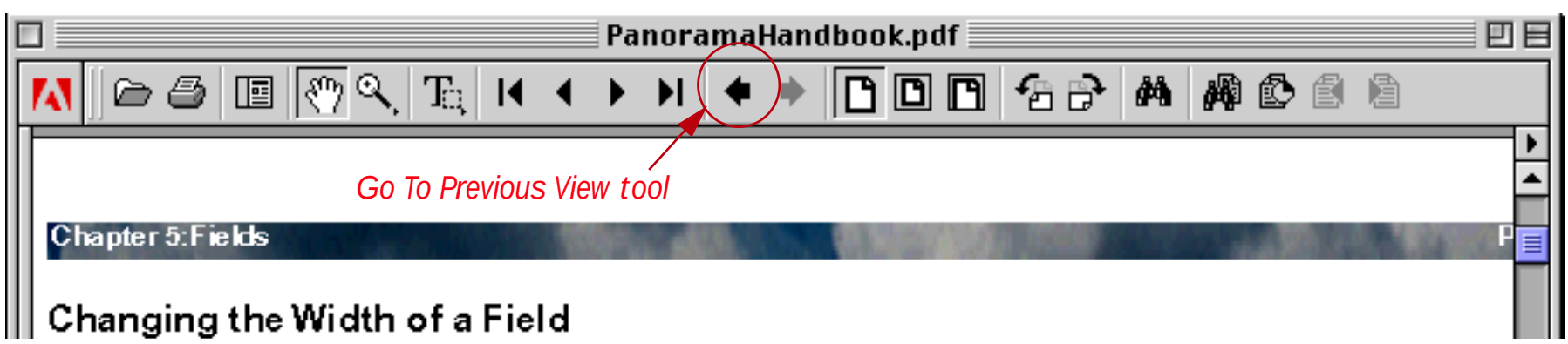
There are actually two different tables of contents in this manual (three if you count the outline on the left). The first is the condensed version. This version contains only the highlights in a compact format.



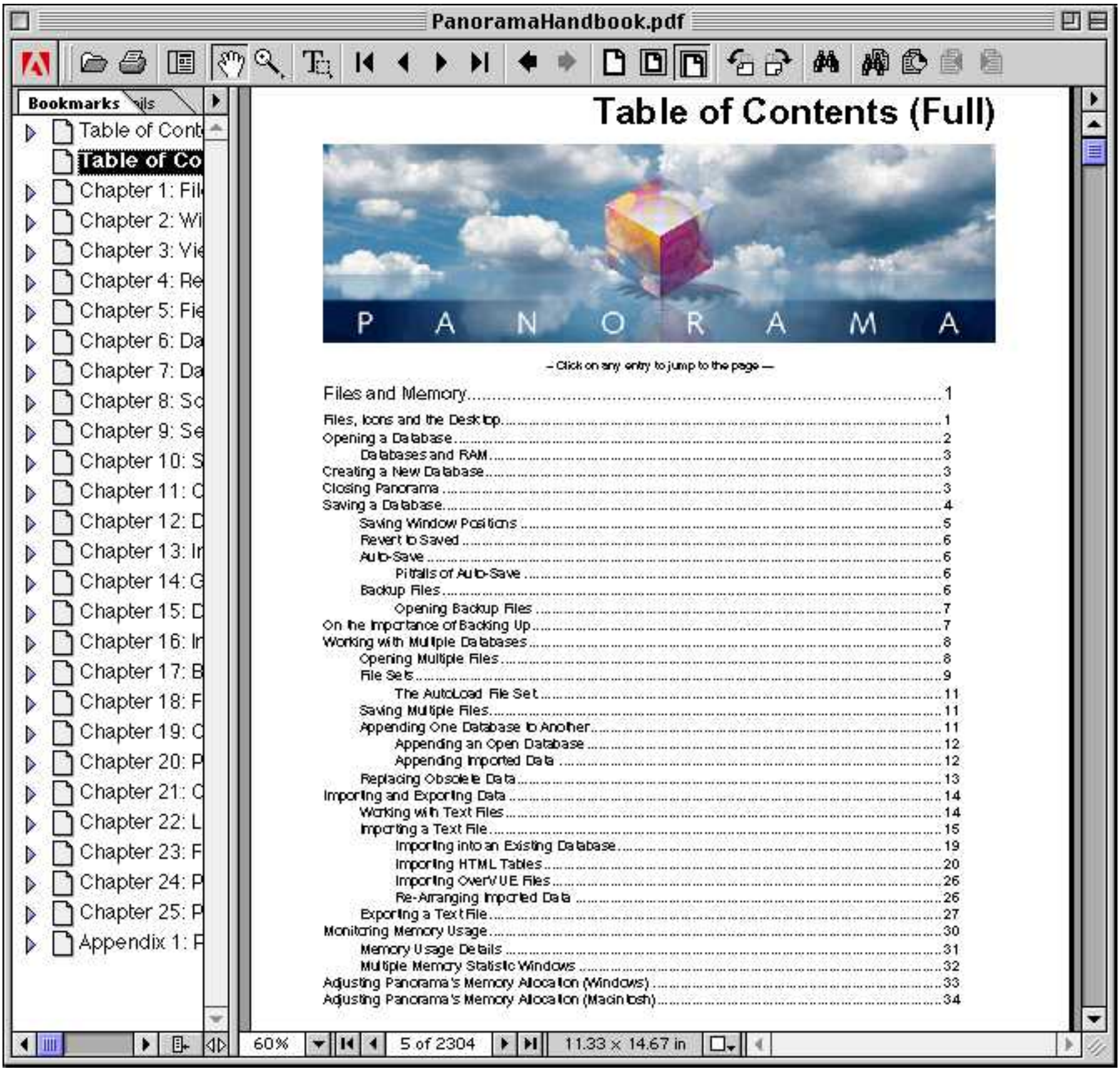
To jump to any topic simply click on the topic's name. All of the topics are links (even though they are not underlined).



If you want to go back to the table of contents you can either select the **Go Back** command from the View menu or you can press the **Go To Previous View** icon in the tool bar.

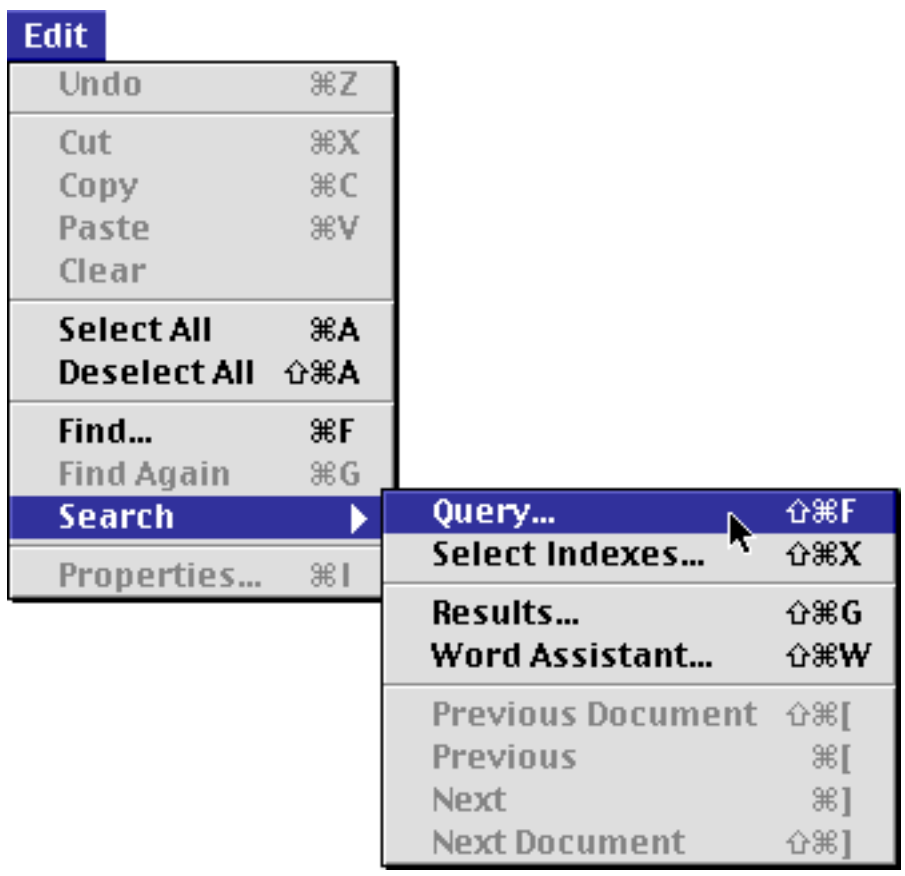


After the condensed table of contents there is a full table of contents. This table of contents has a complete list of every section in the entire manual. Just like the condensed version you can click on any topic to jump to that topic.



Searching the Manual

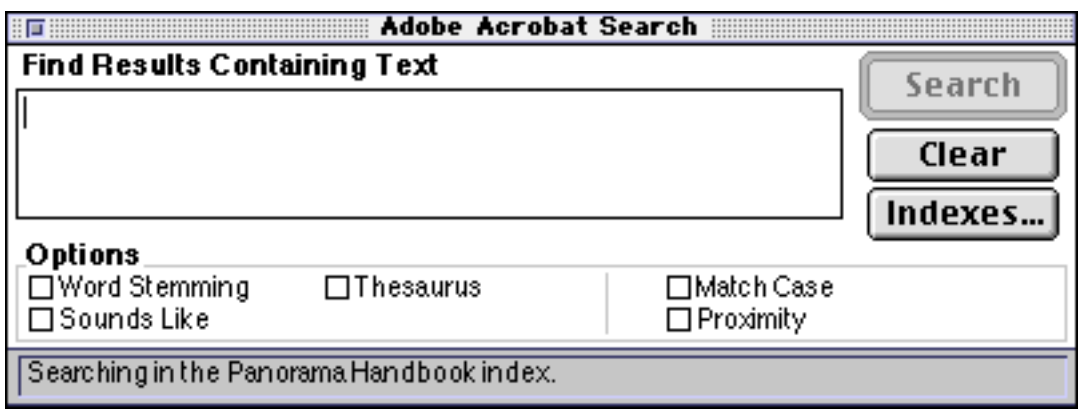
To search for a word or phrase choose the **Query** command from the **Search** submenu in the Edit menu.



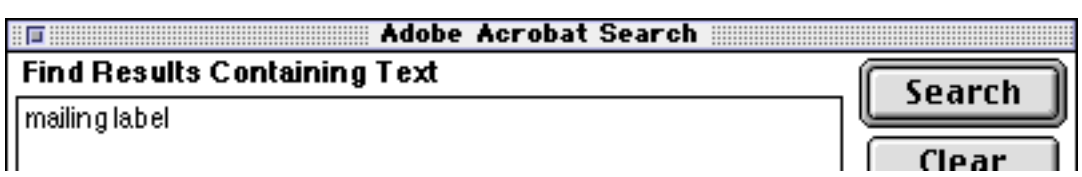
Another way to start a search is to click on the query icon in the tool bar.



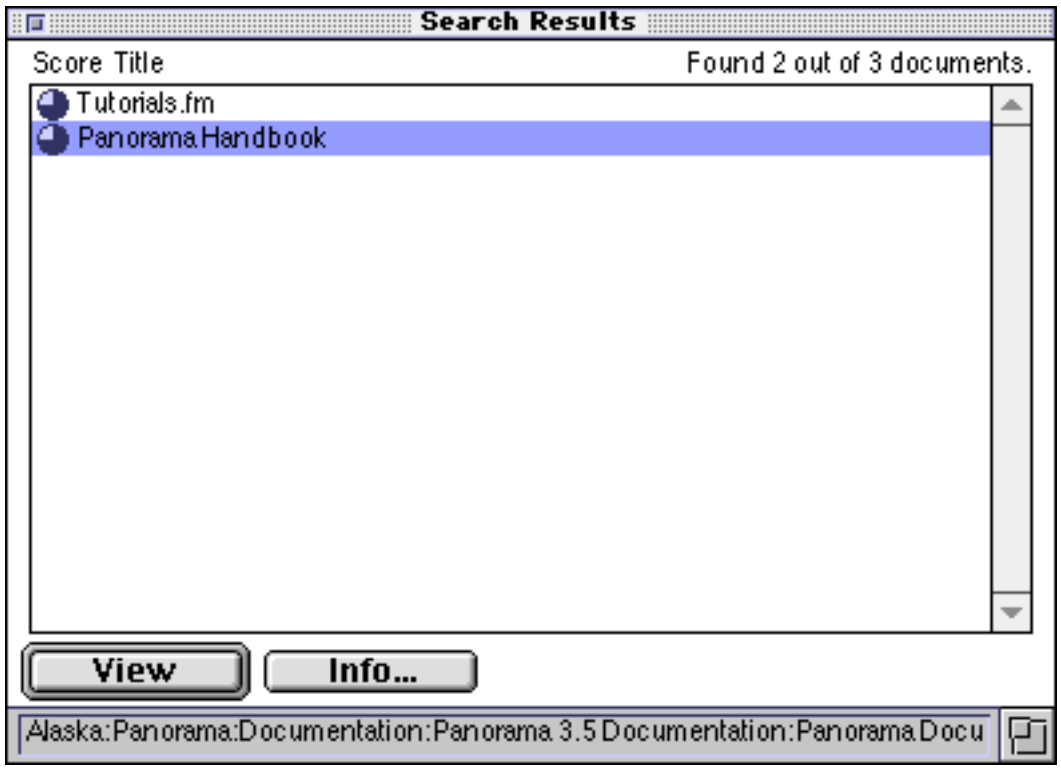
Either way the **Acrobat Search** dialog appears.



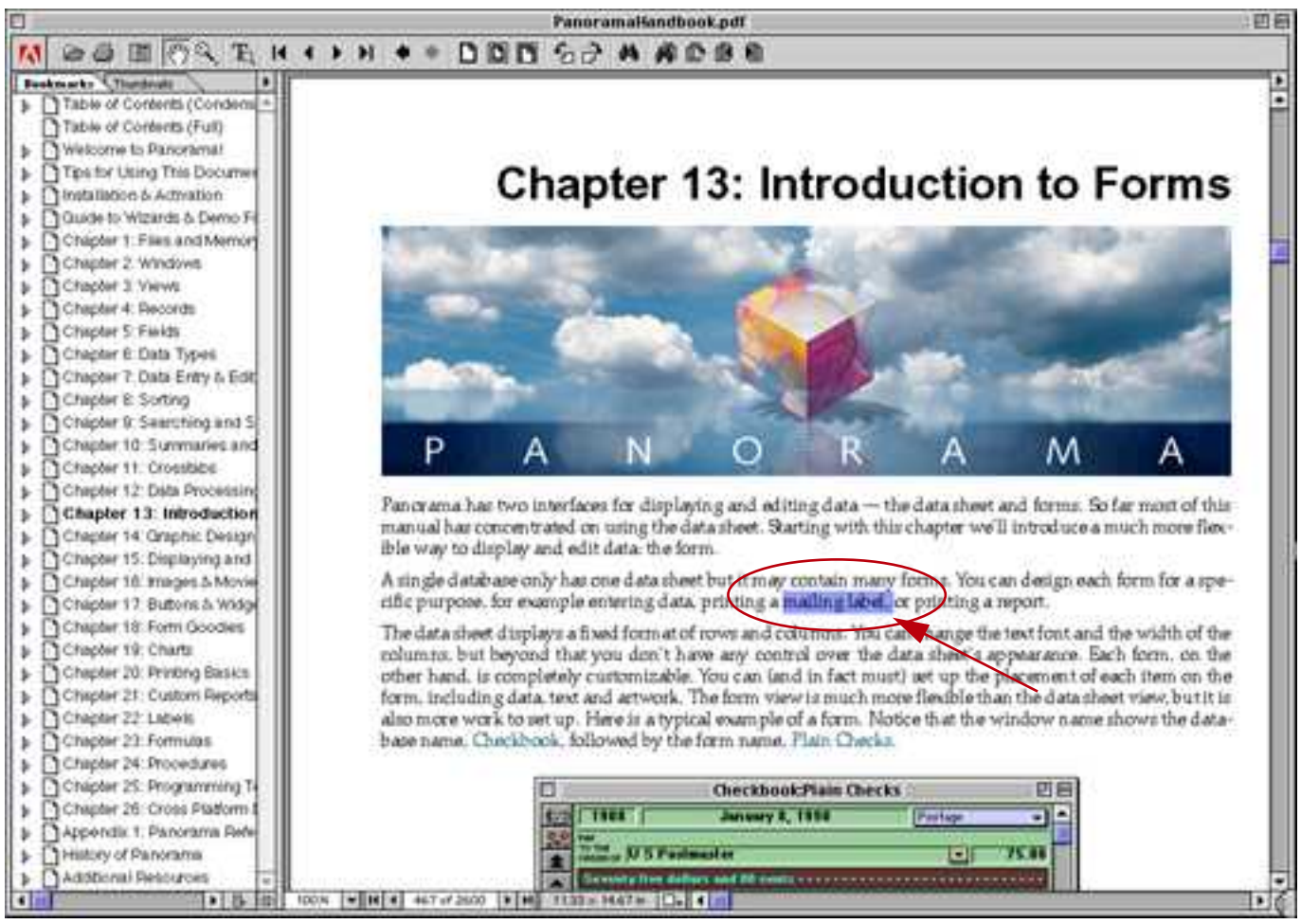
Type in the word or phrase you want to search for.



A dialog appears. If there is more than one file select Panorama Handbook and press the **View** button.



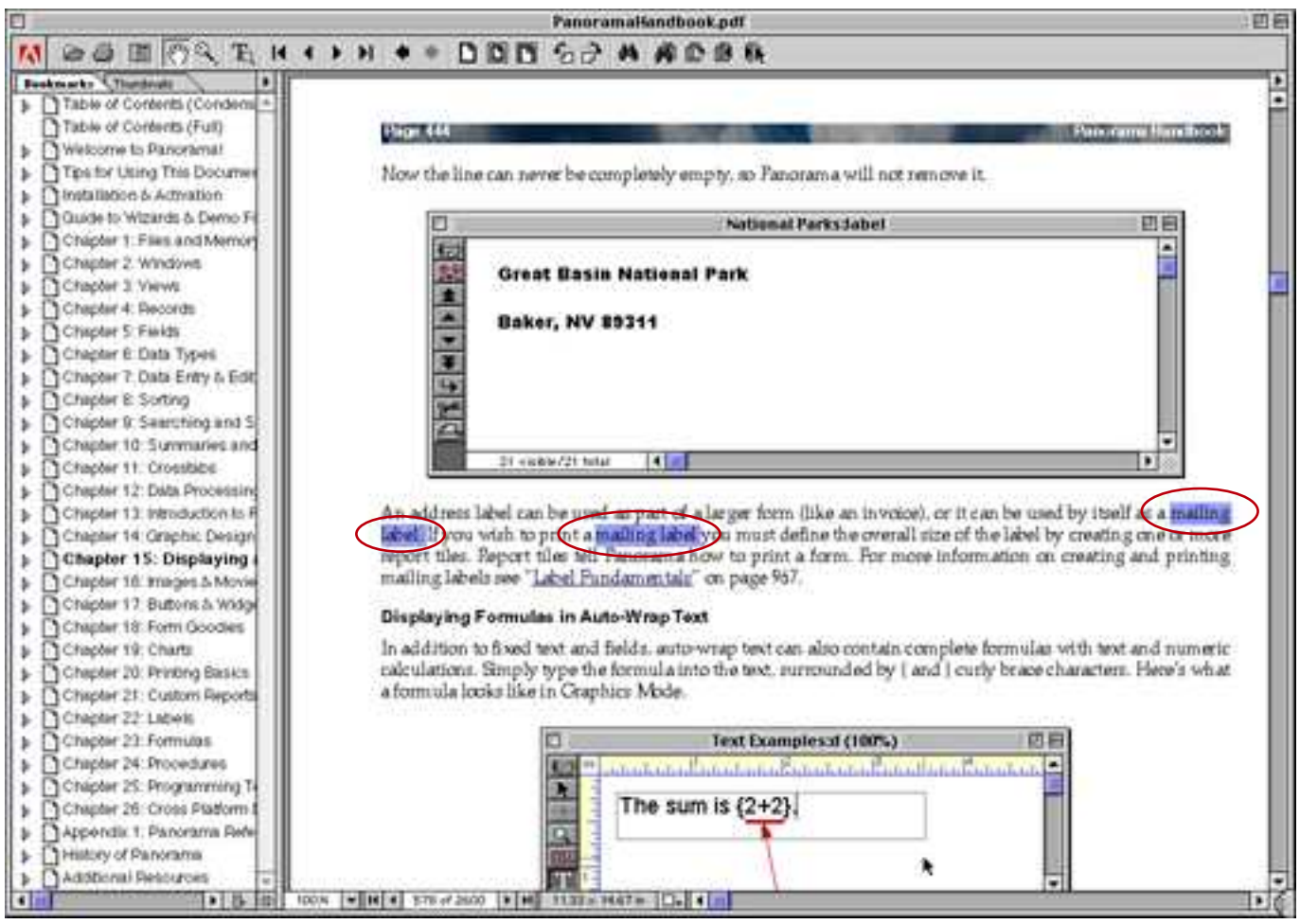
When you press the **View** button the view will instantly jump to the first page where the word or phrase appears. The word or phrase will be highlighted.



Use the icons to locate additional matches, or go back to previous matches.



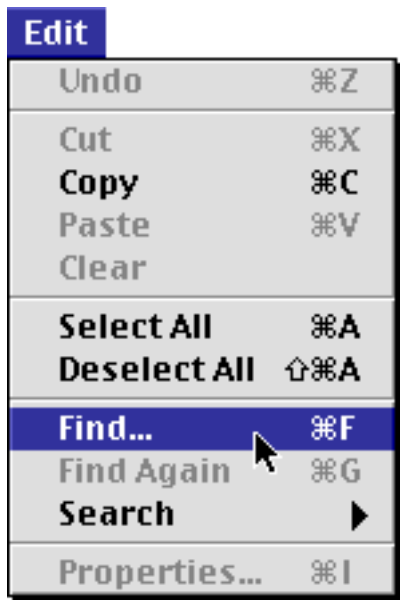
Here is the next page that contains the word or phrase. Actually, this page contains two copies of the phrase, and both of them are highlighted.



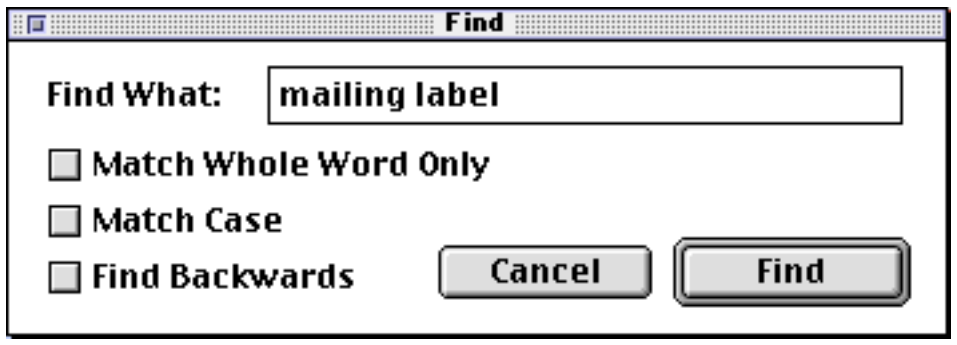
You can continue until you have located all of the occurrences of the word or phrase.

What About the Find Command?

In addition to the **Query** command Acrobat also has a **Find** command.



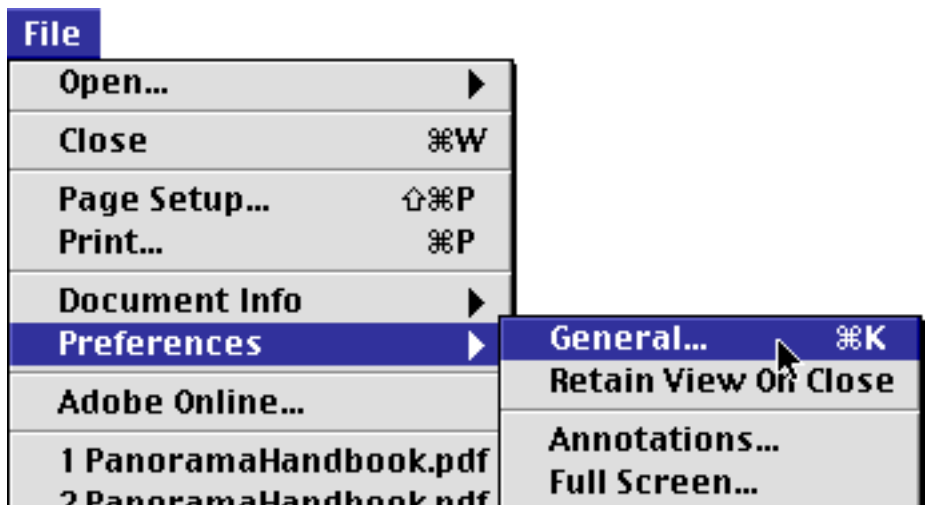
Just as with the **Query** command, the **Find** command allows you to search for a word or phrase.



The **Find** command, however, has a major drawback — it's very slow. Instead of finding the word or phrase instantaneously the **Find** command can take a long time, as much as a minute. Because of this you'll probably want to stick with the **Query** command described in the previous section.

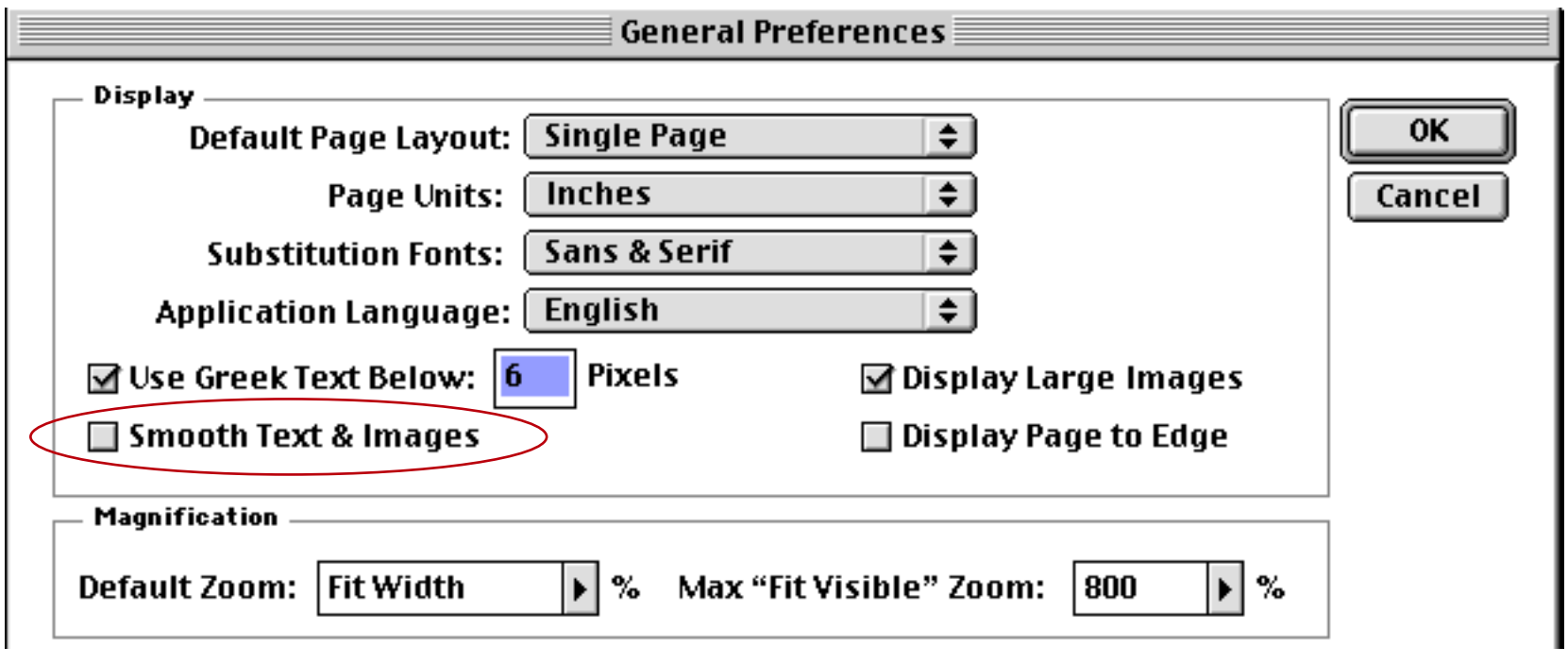
Display Options

Acrobat has many options for customizing the way a document is displayed. We'll mention a couple of options that we have found interesting. Most of these options are found in the **General Preferences** dialog.



Smooth Text & Images Option

One option you may want to adjust is Smooth Text & Images.

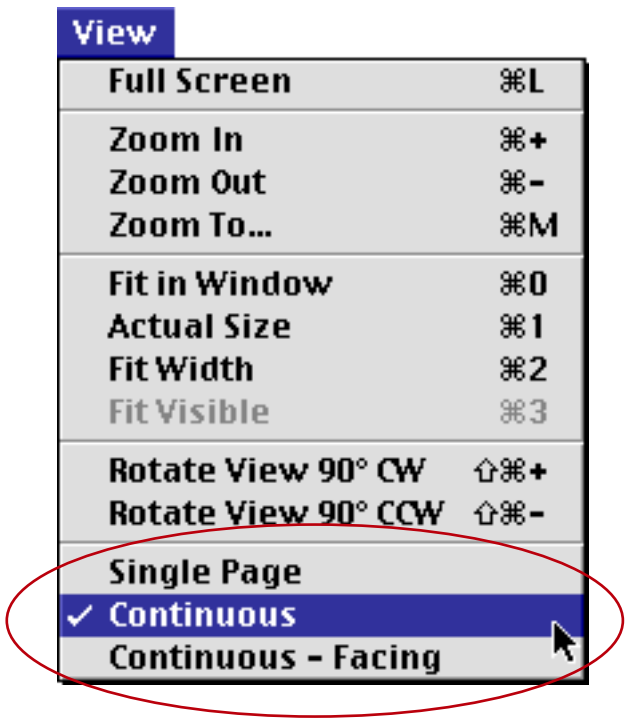


The table below shows what the text looks like with this option turned on and turned off. Personally we prefer it turned off, but it's your call.

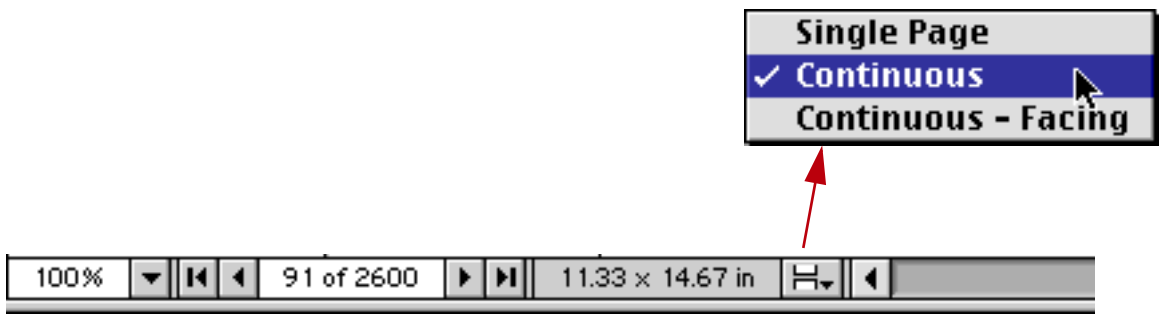
Smooth Text & Images OFF	Smooth Text & Images ON
Displaying the Book Covers The database uses **Flash Art** to display the book stored in RAM as part of the database but are kep playing Images Directly From Disk Files" on pag number of the corresponding book.	**Displaying the Book Covers** The database uses **Flash Art** to display the book stored in RAM as part of the database but are kep playing Images Directly From Disk Files" on pag number of the corresponding book.

Single Page vs. Continuous Option

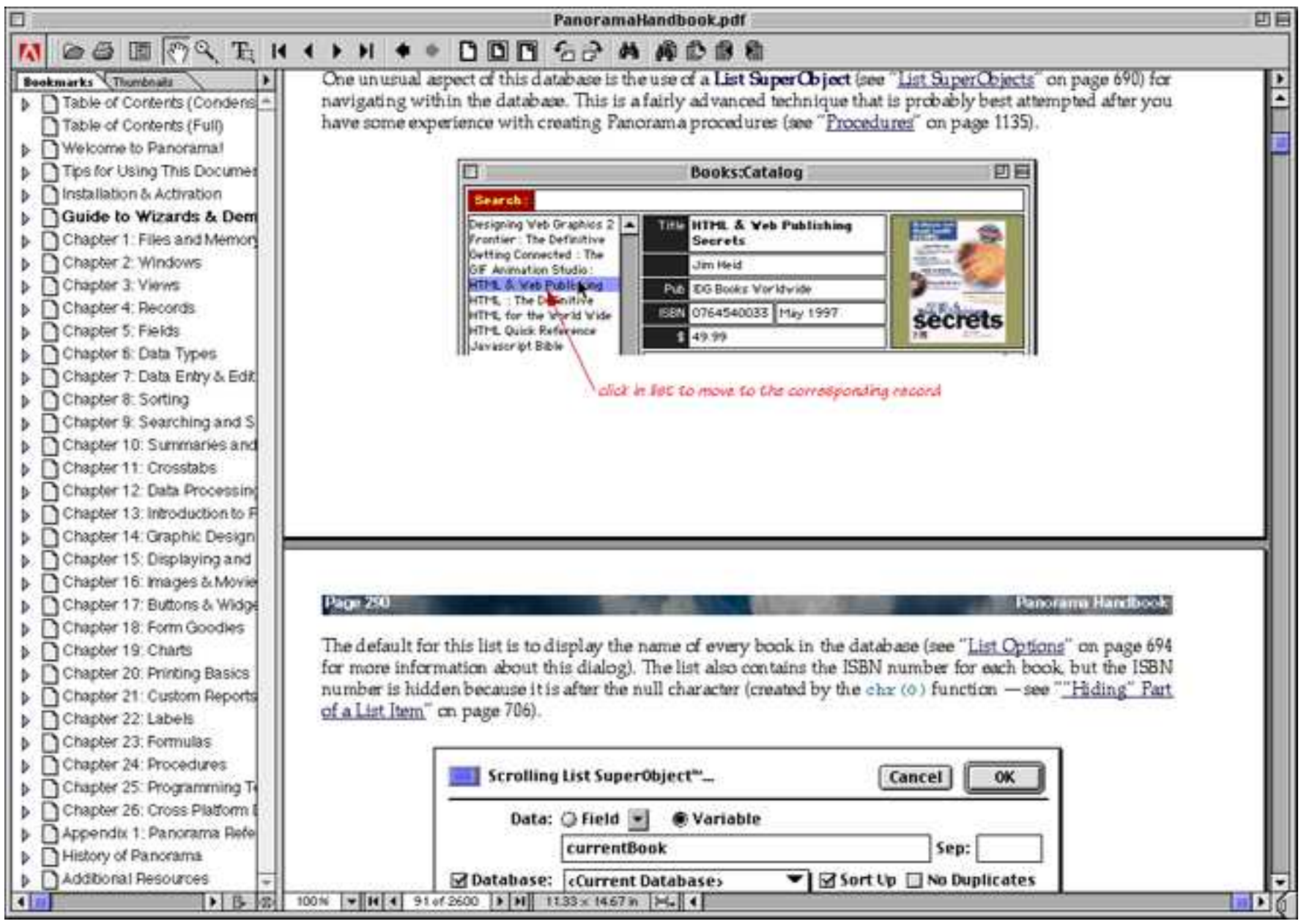
Another option you may want to consider customizing is Single Page vs. Continuous.



You can also adjust this option by clicking on the icon at the bottom of the window.

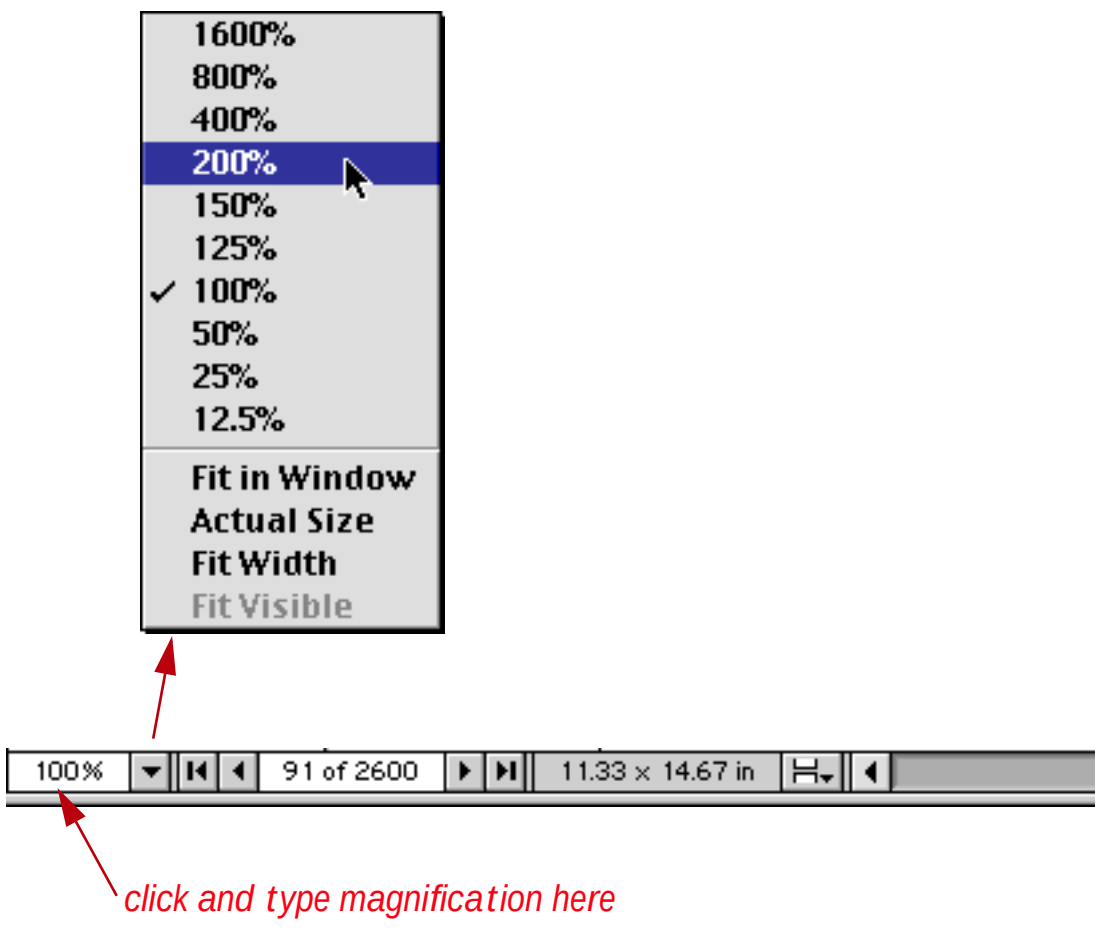


In the Single Page mode Acrobat will never show more than one page at a time. When the Continuous option is set Acrobat will scroll the document continuously as if it was a long sheet instead of as separate pages. As you can see in the illustration below this allows you to see the bottom of one page at the same time as the top of the next page.



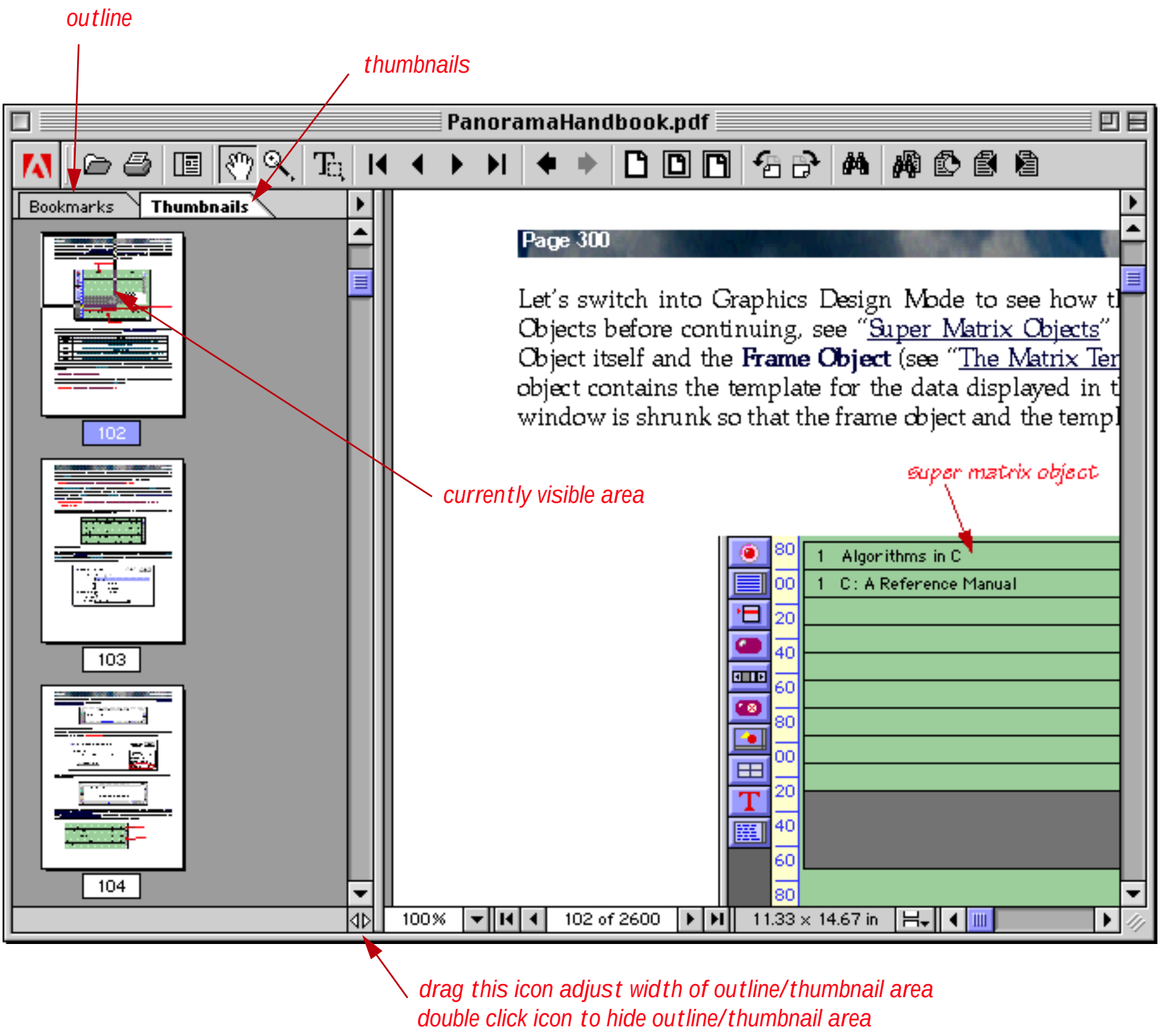
Magnification

The Panorama documentation is designed to be viewed at 100%, and automatically opens at that magnification. All of the text and graphics are optimized for viewing at that magnification. If for some reason you want to zoom in or out you can use the pop-up menu at the bottom of the window, or you can simply click and type in the magnification you want.



Thumbnails

Acrobat normally displays an outline of the manual on the left hand side of the window. By clicking on the tab you can change this to a thumbnail view showing miniature pages. You can click on any section of a page to view that section.



You can also adjust the width of the area reserved for outlines and thumbnails, as shown above. If the area becomes wide enough then two or more columns of thumbnails will be displayed. Or if you want to completely hide this area, double click on the icon. Double click again to make it re-appear.

Installation & Activation



Congratulations on your purchase of Panorama for Windows or Power Macintosh. If you are new to Panorama, welcome. If you have used a previous version of Panorama, the installation procedure has changed significantly.

Getting Organized

Before you begin the installation, make sure you have all of the items you'll need. Of course you'll need the installation CD-ROM, or the copy of Panorama that you have downloaded. If you are installing the Panorama Engine, that is all you will need, and you can proceed to the first step.

If you are installing Panorama or Panorama Direct, you'll need the serial number and product code. These are printed on the paperwork that came with your order (if you purchased Panorama directly from ProVUE Development you will also have received an e-mail with these codes). If you are installing Panorama on more than one machine you'll need a separate set of serial numbers and product codes for each machine.

You may have additional installation codes, for example for the enhanced image pack, spelling dictionary or zip code lookup. You'll need these codes to install these add-on products.

Installing the Software

The first step in the installation process is to insert the CD-ROM into your CD-ROM drive. On Windows machines this should automatically launch the installer program. (If for any reason it doesn't, you can manually launch the installer by choosing **Run** from the Start Menu, then typing D:\SETUP.EXE and pressing the **Return** key [if your CD-ROM drive is not D: you should substitute the letter used by your CD-ROM drive].)

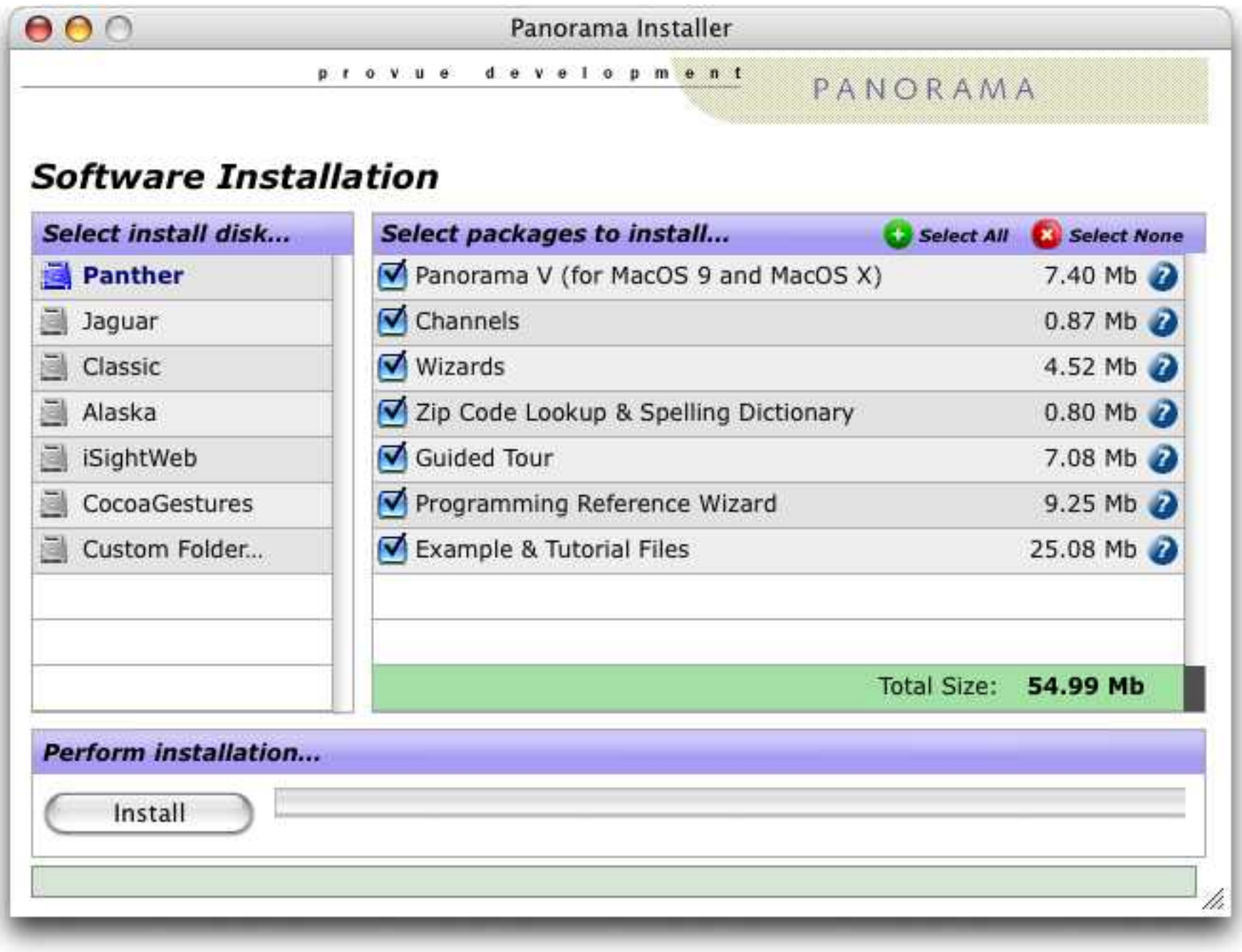
If you are installing on the Macintosh, double click on the Install Panorama icon to open the installer.



Unlike some other installers, it is not necessary to disable system extensions before using the Panorama installer.

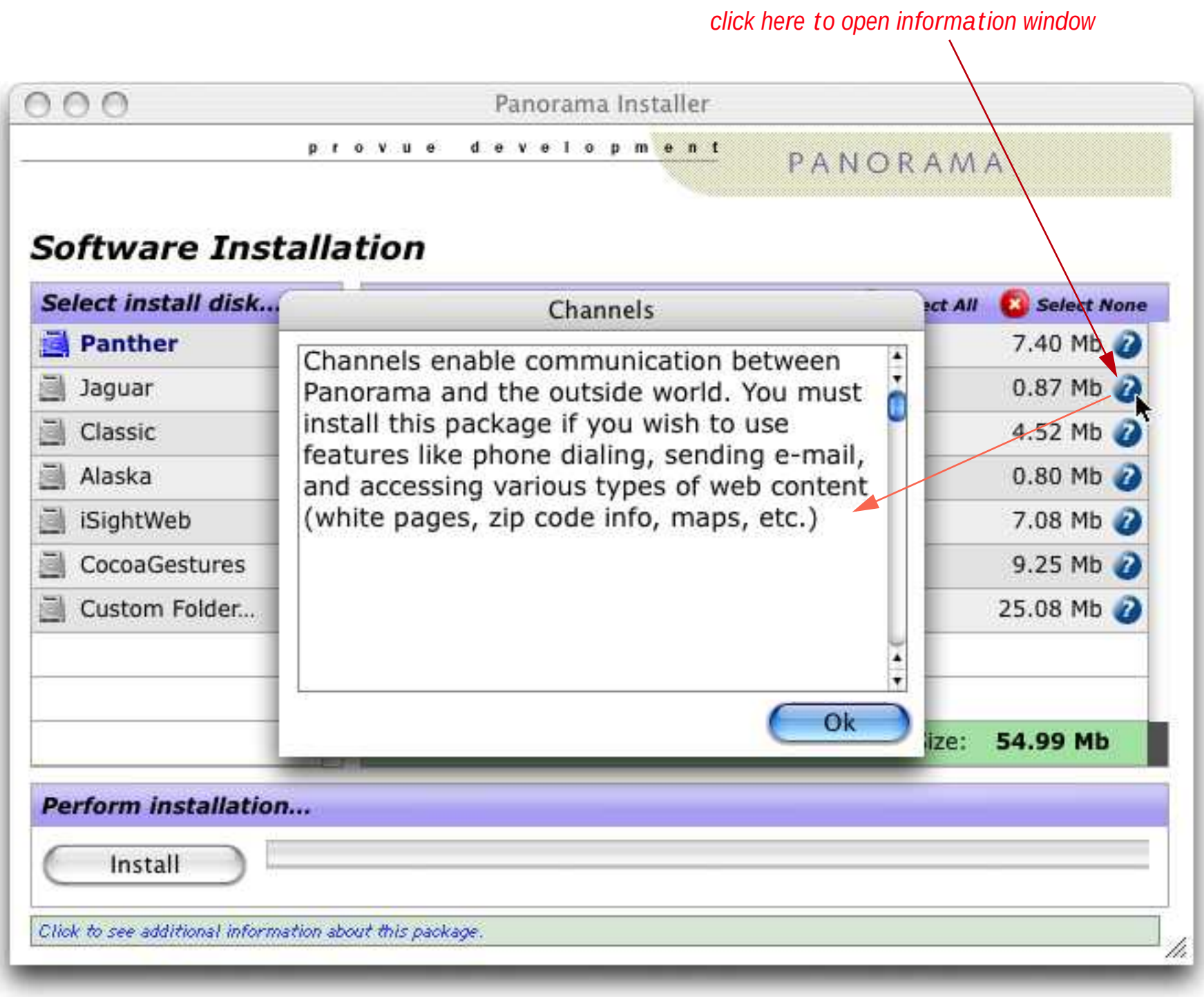
The Main Installer Window

The main installer window allows you to choose the installation location and packages you want to install.



Installation Package

The right side of the window displays the packages available for installation, along with their sizes. There may be more or less packages listed than those shown here. If a package you need is not listed you should go back and download the most deluxe version from our web site (or purchase the CD). The window lists the installed size of each package. For more information on a specific package click on the **?** button.



Selecting the Installation Location

The installer normally places the software in a folder inside your Program Files folder (Windows) or Applications folder (Macintosh) on your main hard drive. If you wan to put this folder on another drive just click on the drive name. If you want to put the folder in a completely different location click on the Custom Folder… option.



You can either type in the folder location or drag a folder from the Finder or Desktop onto the dialog (MacOS only).

Installing the Software

Once the packages and location are selected you are ready to go. Just press the **Install** button and let the installer do all the work. If you are installing packages that require authorization codes to install (zip code, spelling dictionary) you will be prompted for those codes at this time. If you don't have them handy you can always go back and install them later. When the install is finished it will display a report listing what it has done.



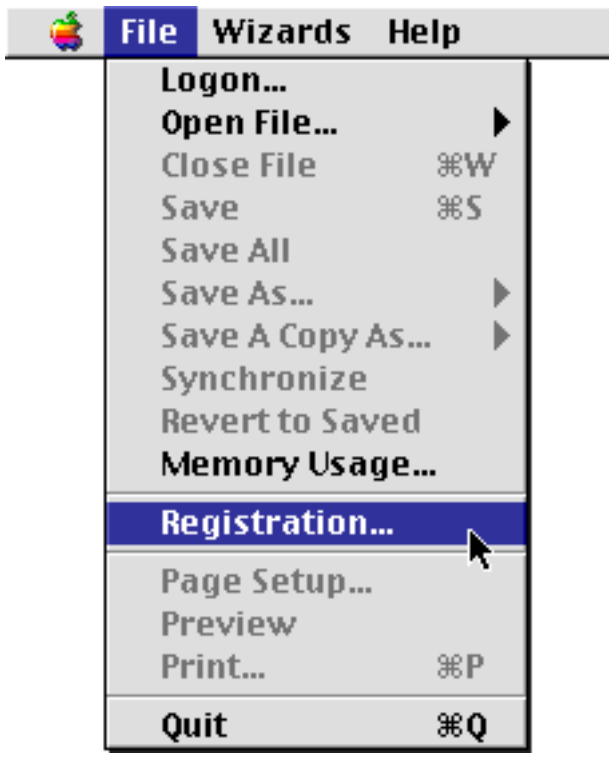
At this point you will usually want to press the **Exit Installer** button. (If you wish you can save this report for later by pressing the **Save Report** button before exiting.) If Panorama was already activated on this computer (see below) then you are ready to go.

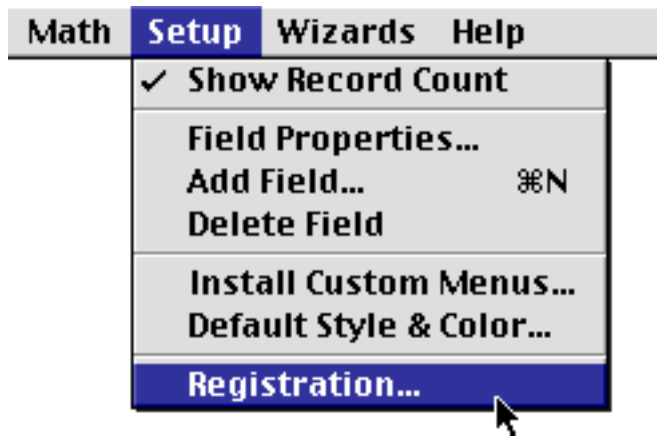
Activating the Software

When you first install Panorama on a computer it initially acts as a demo version. If you haven't purchased Panorama yet you can use this demo version to evaluate your potential Panorama purchase (see "Using Panorama's "Demo Mode"" on page 105). If you are using Panorama as a demo then you don't need to worry about activating it for the moment.

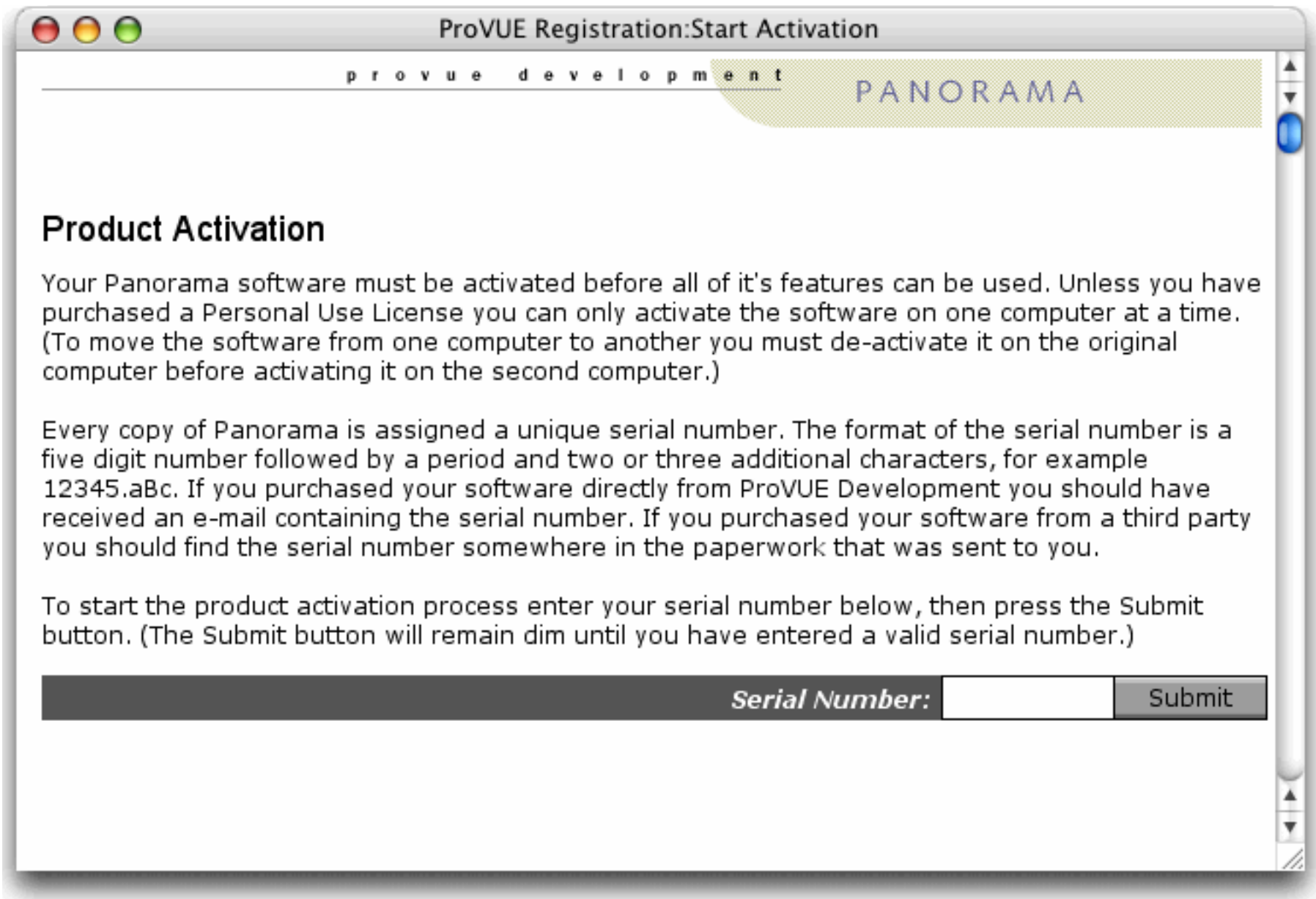
If you have purchased **Panorama** or **Panorama Direct** you'll want to activate your software as soon as possible. If you have access to the Internet you can do this in a few minutes at any time of the day or night. (If you do not have access to the Internet you'll need to do the activation process during ProVUE business hours.)

To activate the software you need to open the **ProVUE Registration** database. If Panorama is not already activated, the installer will automatically select the **Open Activation Window** option and automatically start the activation process when you press the **Exit Installer** button. If you need to do this later you can do so by choosing the **Registration** menu command. You'll find this command either in the **Setup** menu (when one or more databases is open) or in the **File** menu (when no databases are open).

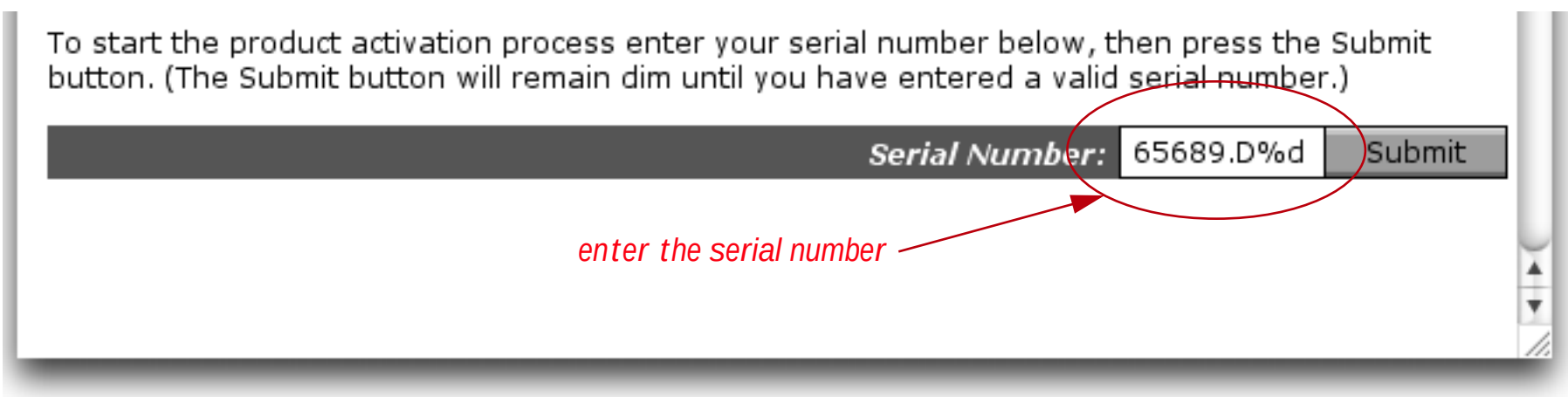




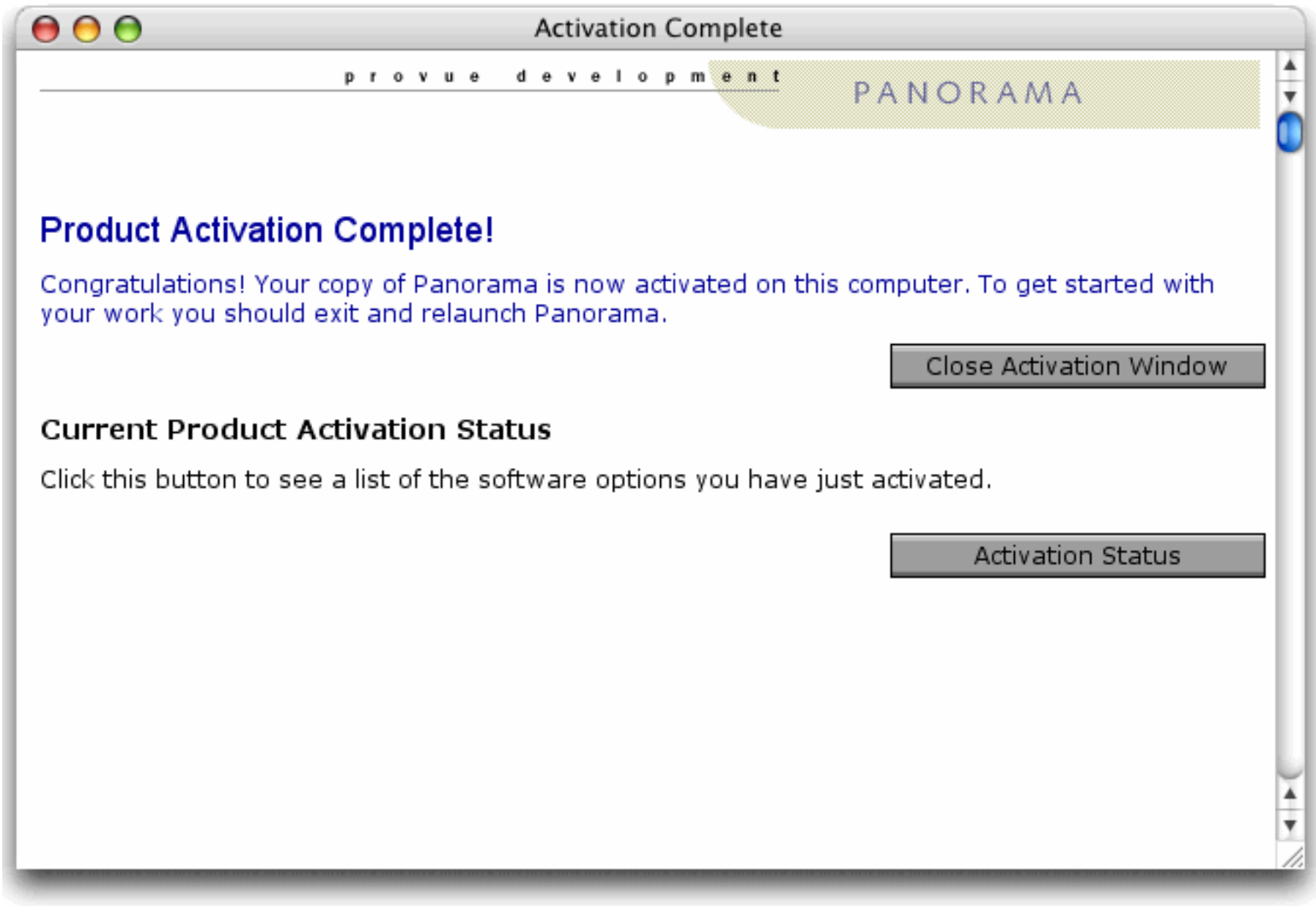
The first time you open the ProVUE Registration database it displays the window shown below.



Enter your serial number to start the activation process. The format of the serial number is a five digit number followed by a period and two or three additional characters, for example 12345.aBc. If you purchased your software directly from ProVUE Development you should have received an e-mail containing the serial number. If you purchased your software from a third party you should find the serial number somewhere in the paperwork that was sent to you.



When you press the **Submit** button the window there will be a delay as Panorama contacts the ProVUE web server. (The only information transmitted to the ProVUE web server is the serial number you just typed in. No other information is transmitted.) If the serial number is valid, the Activation Complete window will appear. (Note: If you have purchased a personal use license, you will be asked to enter the last four digits of the credit card used to purchase the license.)

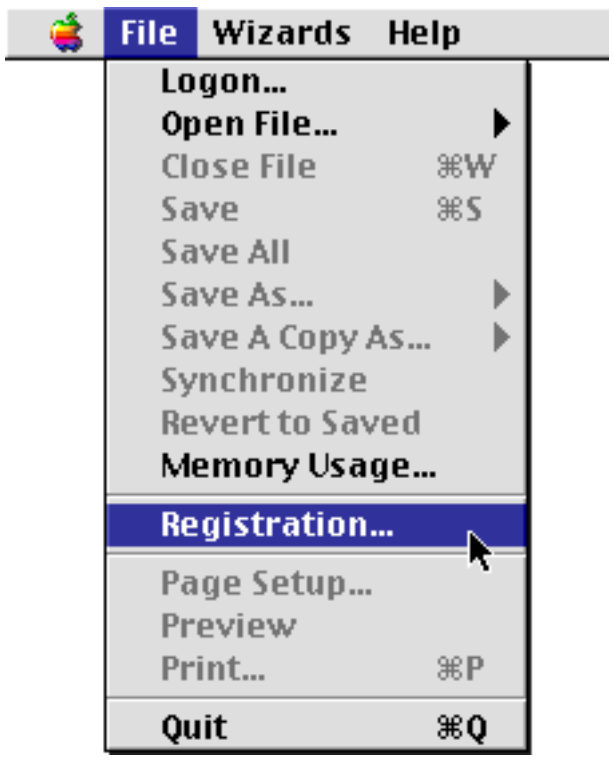


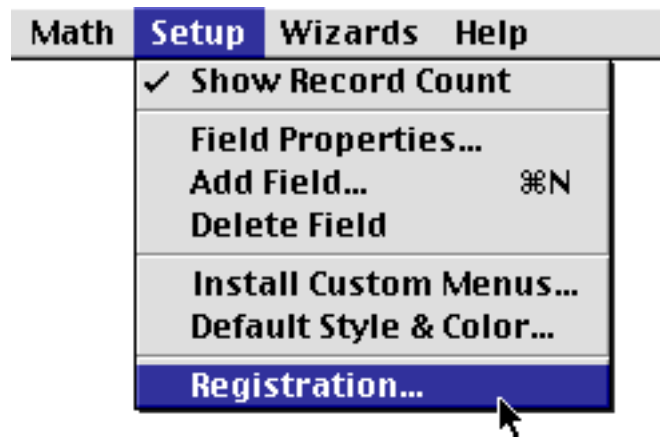
That’s all there is to it! Your copy of Panorama is ready to use.

If you don’t have an internet connection you will have to perform the activation manually. Panorama will automatically detect that you don’t have an internet connection, and give you on-screen step-by-step instructions for completing the process.

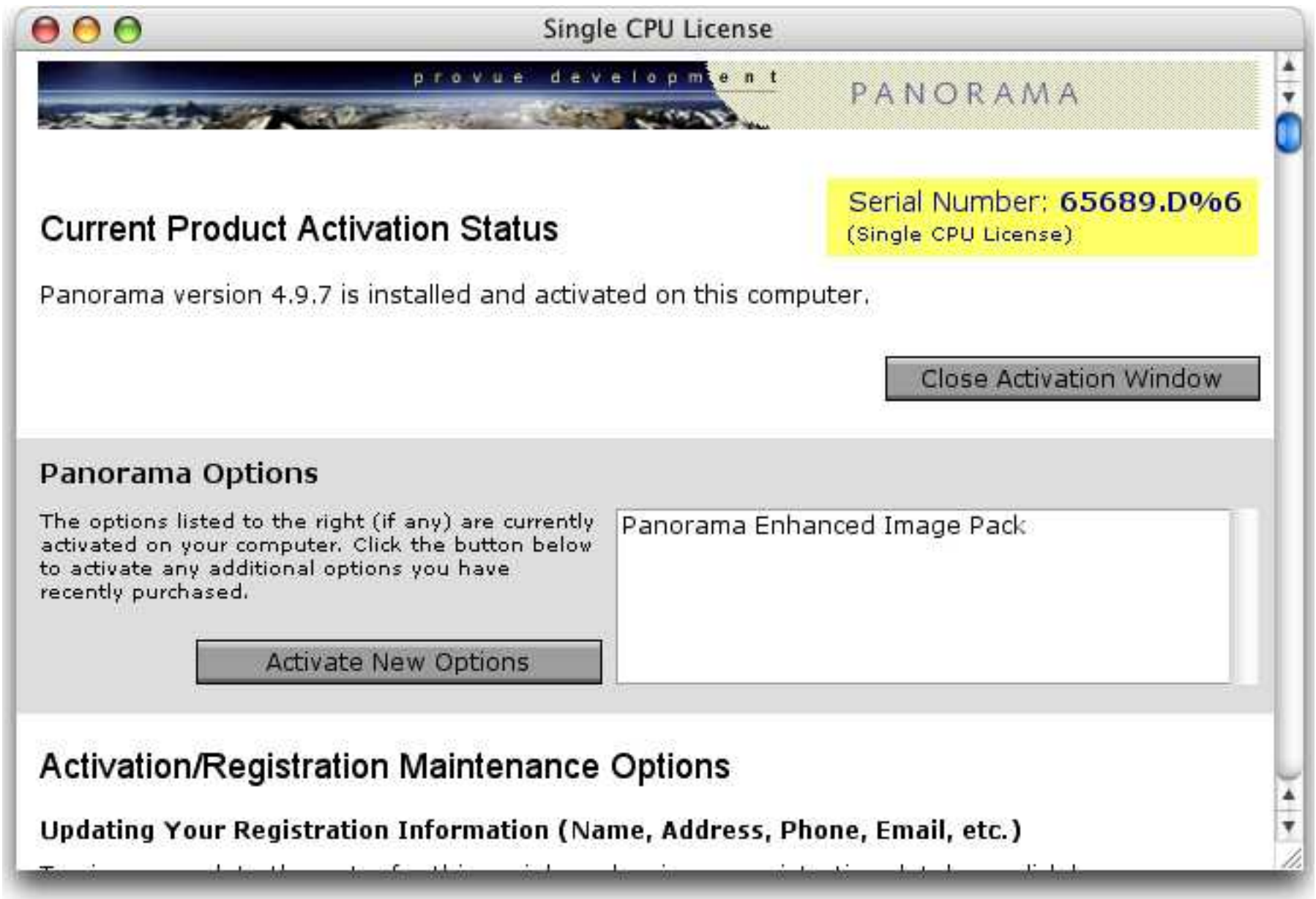
Moving Your Software to Another Computer (Deactivating Your Software)

Unless you are using a personal use license, your ProVUE software may only be activated on one computer at a time. If you want to transfer this software to a different computer, you must first de-activate the software on the current computer. Then you can transfer the software to the second computer and activate it. To start the deactivation process open Panorama and open the **ProVUE Registration** database the ProVUE Registration database again using the **Registration** command. You'll find this command either in the **Setup** menu (when one or more databases is open) or in the **File** menu (when no databases are open).

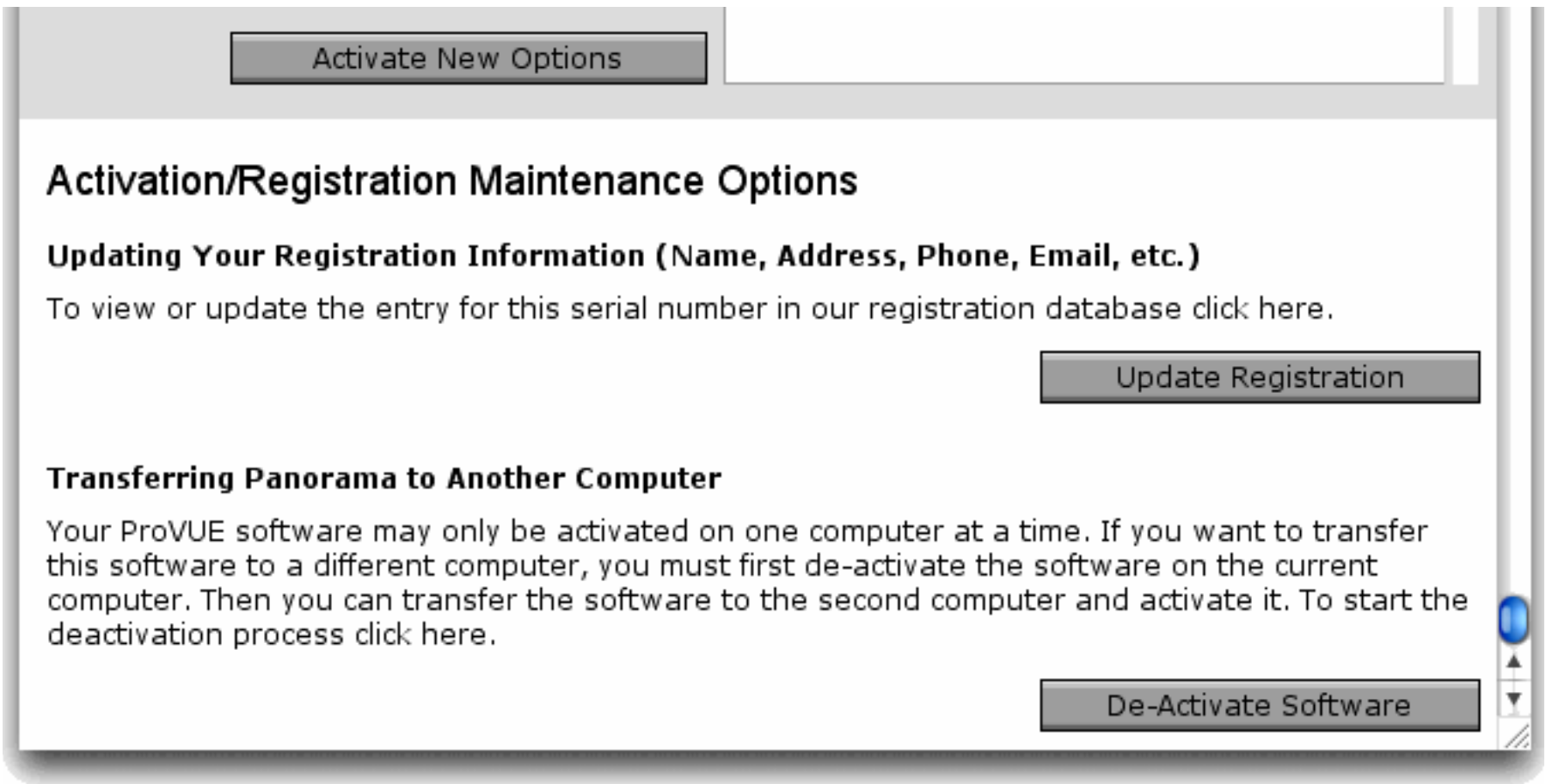




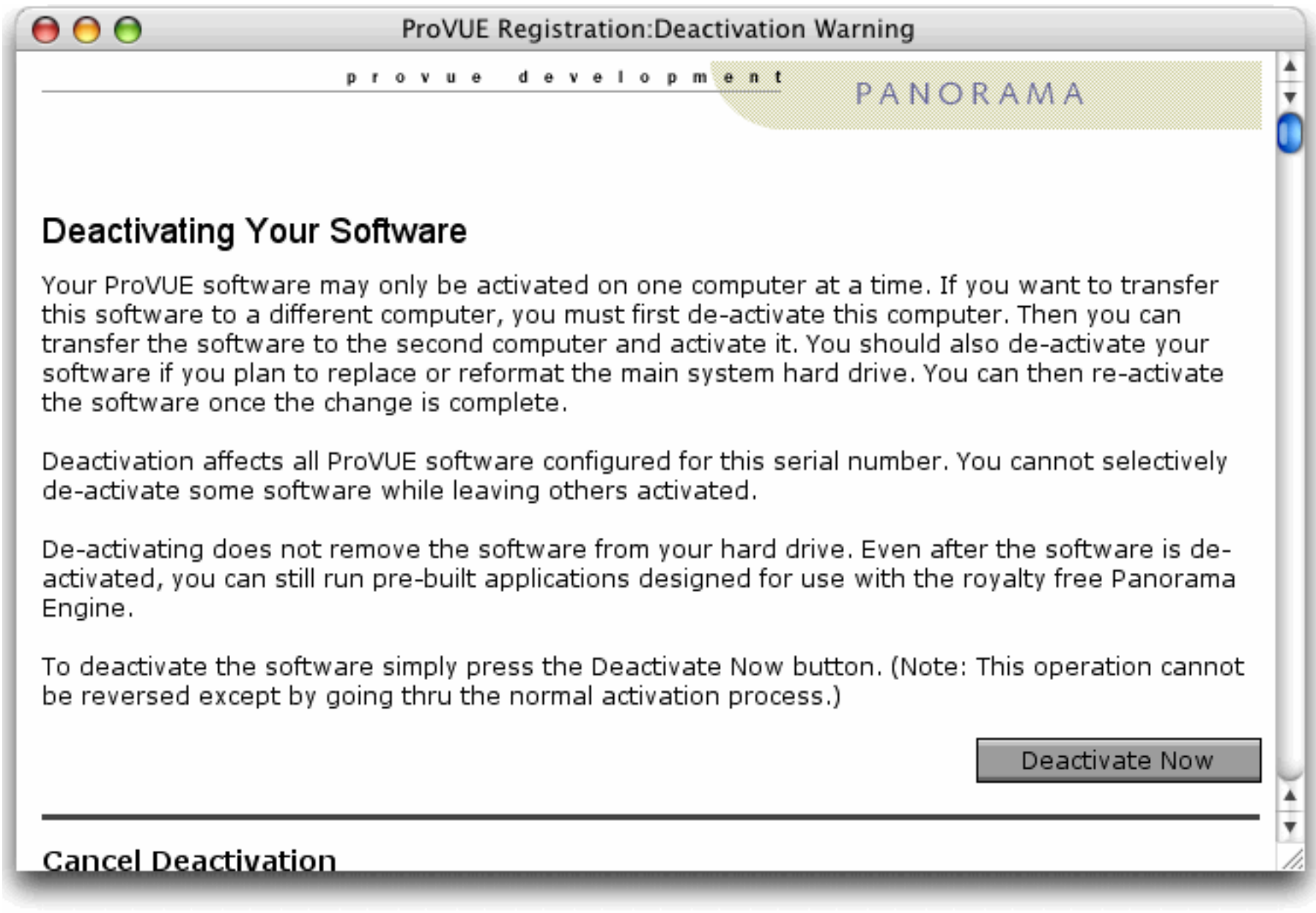
This opens a Product Activation Status window that will look something like this.



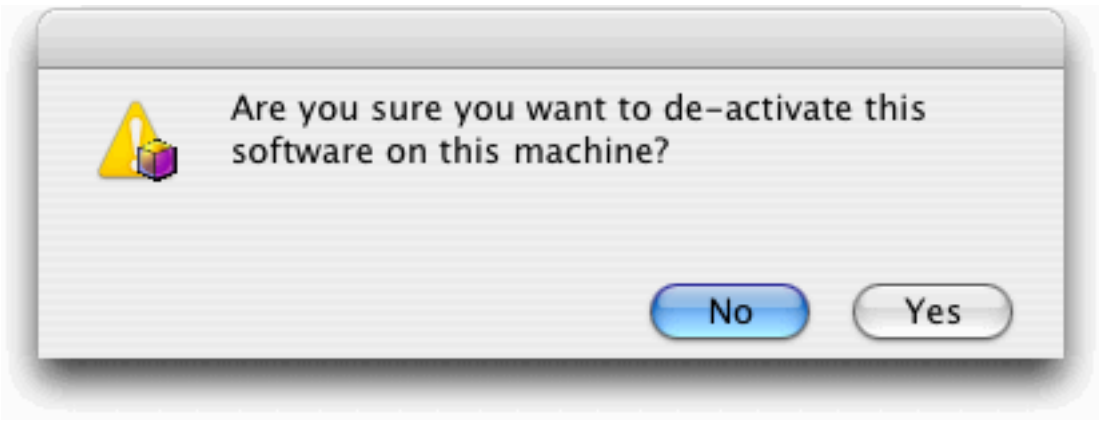
Scroll down to the bottom of this window.



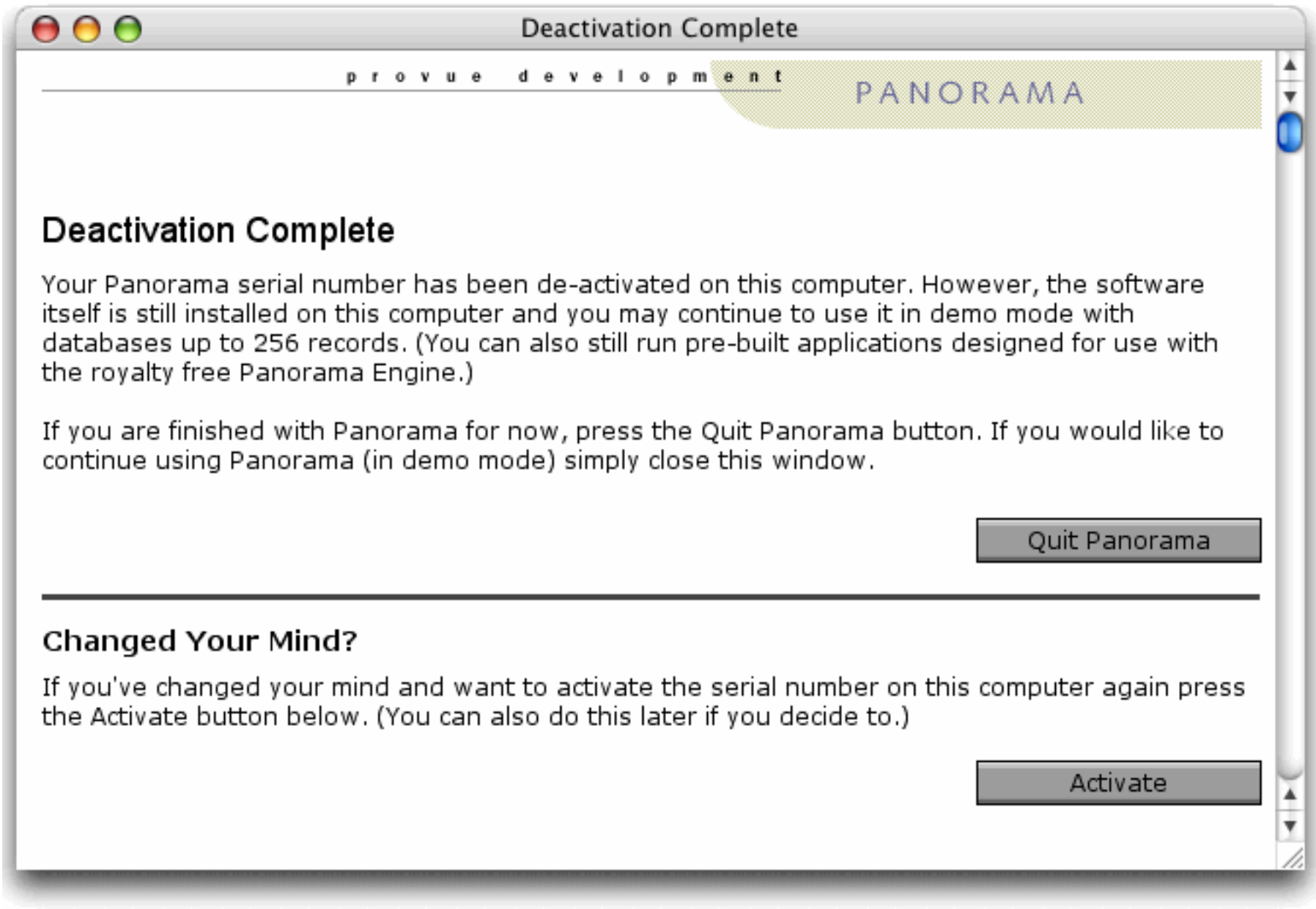
To deactivate Panorama on this computer press the **De-Activate Software** button. Before actually deactivating the software Panorama displays a warning page. You can still back out by scrolling down and pressing the **Cancel** button (not visible in the image below).



To go ahead and deactivate, press the **Deactivate Now** button. Panorama gives you one last chance to back out.



To proceed, press the **Yes** button. If your computer is connected to the Internet the deactivation will be completed automatically and this page will appear. (If you don't have internet access you must submit the deactivation code to the ProVUE manually. Follow the instructions Panorama displays to complete this process.)



Now that the software is de-activated on your old computer you can go ahead and activate it on your new computer. De-activating does not remove the software from the hard drive of your old computer. Even after the software is de-activated, you can still run the software in demo mode (see "Using Panorama's "Demo Mode"" on page 105).

Using Panorama With a Personal Use License

When using Panorama with the personal use license the **About Panorama** dialog displays your personal use license information. The illustration below simulates what the **About Panorama** dialog on your computer will look like.



The personal license information is stored in a file named Panorama Personal License.dat. This file is stored in the same folder as the Panorama application.

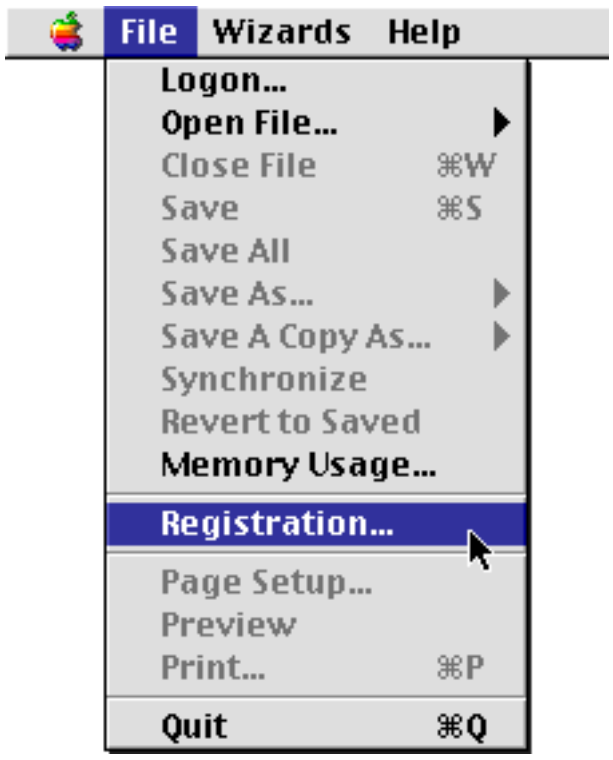


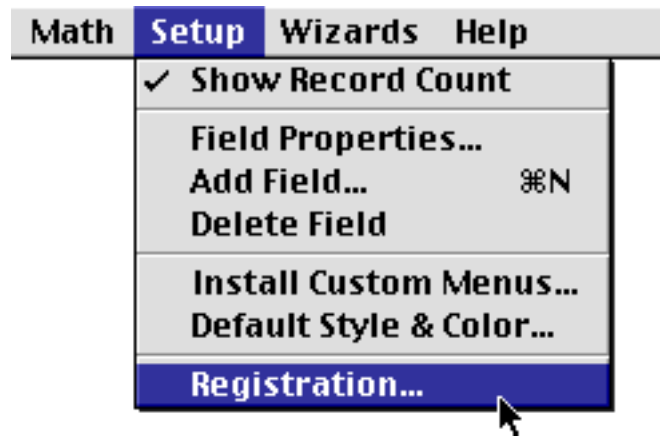
If you want to you can copy this file to your other personal computers, or back up this file. The information in the file is encoded so that it cannot be read without the Panorama application. Of course you should be careful not to give a copy of this file to anyone else.

Setting Up and Using a Personal Use Password

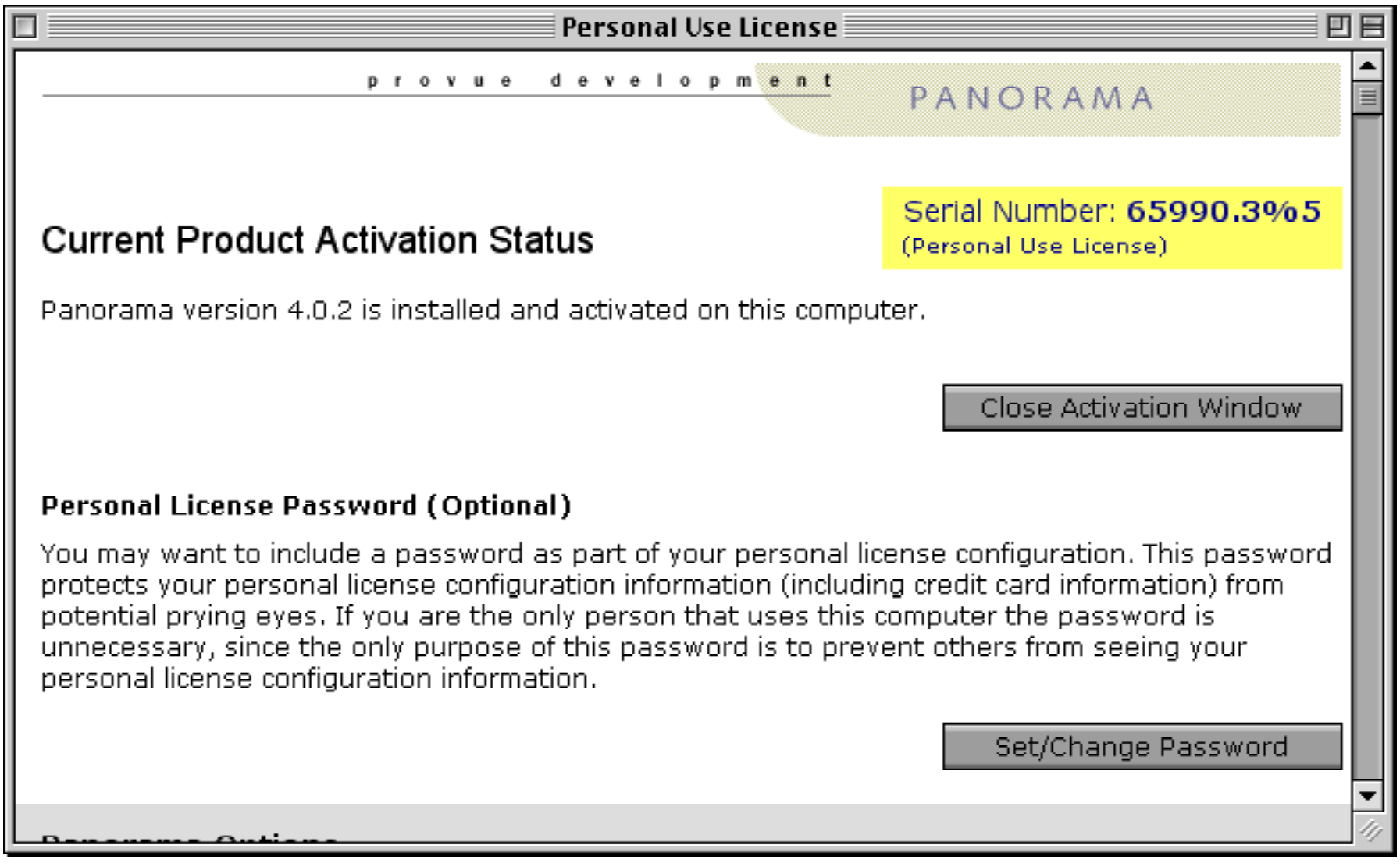
Once you install the personal use license you can optionally set up a password on your copy of Panorama. If someone attempts to use Panorama without the password Panorama will behave as if it is not registered at all, and the **About Panorama** dialog will not display the personal use license information.

If you've already set up the personal license information, start by opening the ProVUE Registration database again using the **Registration** command. You'll find this command either in the **Setup** menu (when one or more databases is open) or in the **File** menu (when no databases are open).

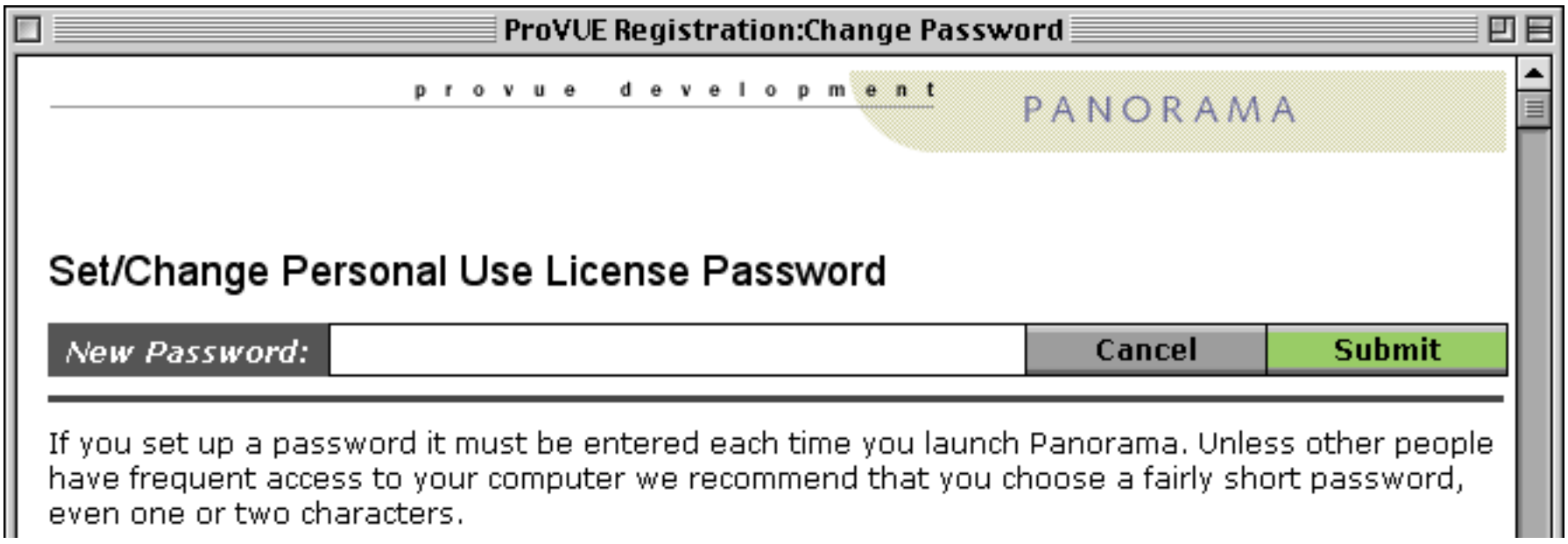




This opens a Product Activation Status window. If this copy of Panorama is currently activated with a personal use license the window will look something like this.



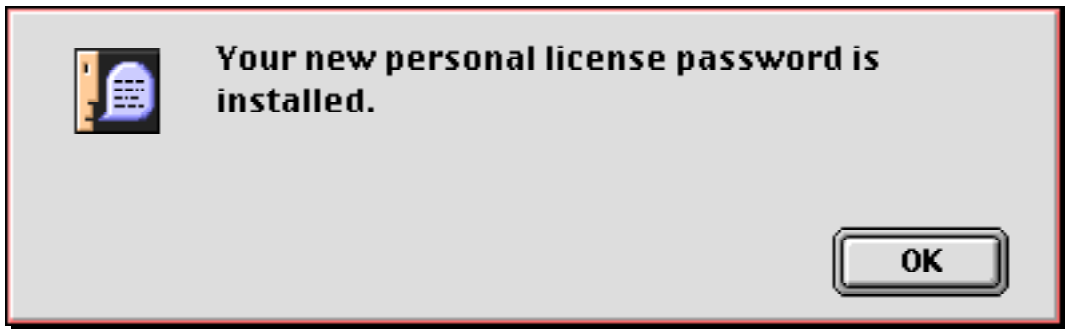
To set or change the password, press the **Set/Change Password** button. This opens the window shown below.



You'll have to type this password each time you open Panorama, so you'll probably want to keep it as short as possible, and it will be easier if you stick to numbers and lower case letters, as shown in the example below.



After a short delay Panorama will confirm that your password is now installed.



When you press the **OK** button Panorama will switch back to the Product Activation Status window. There is no indication in this window that a password has been set.

To try out your new password first quit Panorama then re-open it. As Panorama opens it will stop and pause when the introductory screen appears.



At this point you need to go ahead and type in the password. As soon as you press the last key of the password Panorama will continue normally. You do not need to press the **Return** or **Enter** keys, just the password itself. Panorama will not display the keys as you type them, so just keep typing until the password is complete.

If you type an incorrect key, click on the mouse or wait more than six seconds without pressing a key Panorama will beep and then continue. However, in this case Panorama will behave as if it has not been activated. If you attempt to save or print a database with more than 250 records Panorama will "nag" you to purchase and activate the program (see “Using Panorama’s “Demo Mode”” on page 105), and the personal license information will not appear in the **About** dialog. This allows you to prevent someone else from seeing your personal license information, and also prevents them from using Panorama. If you make a mistake as you type in the password you'll need to quit Panorama and re-open it to try again.

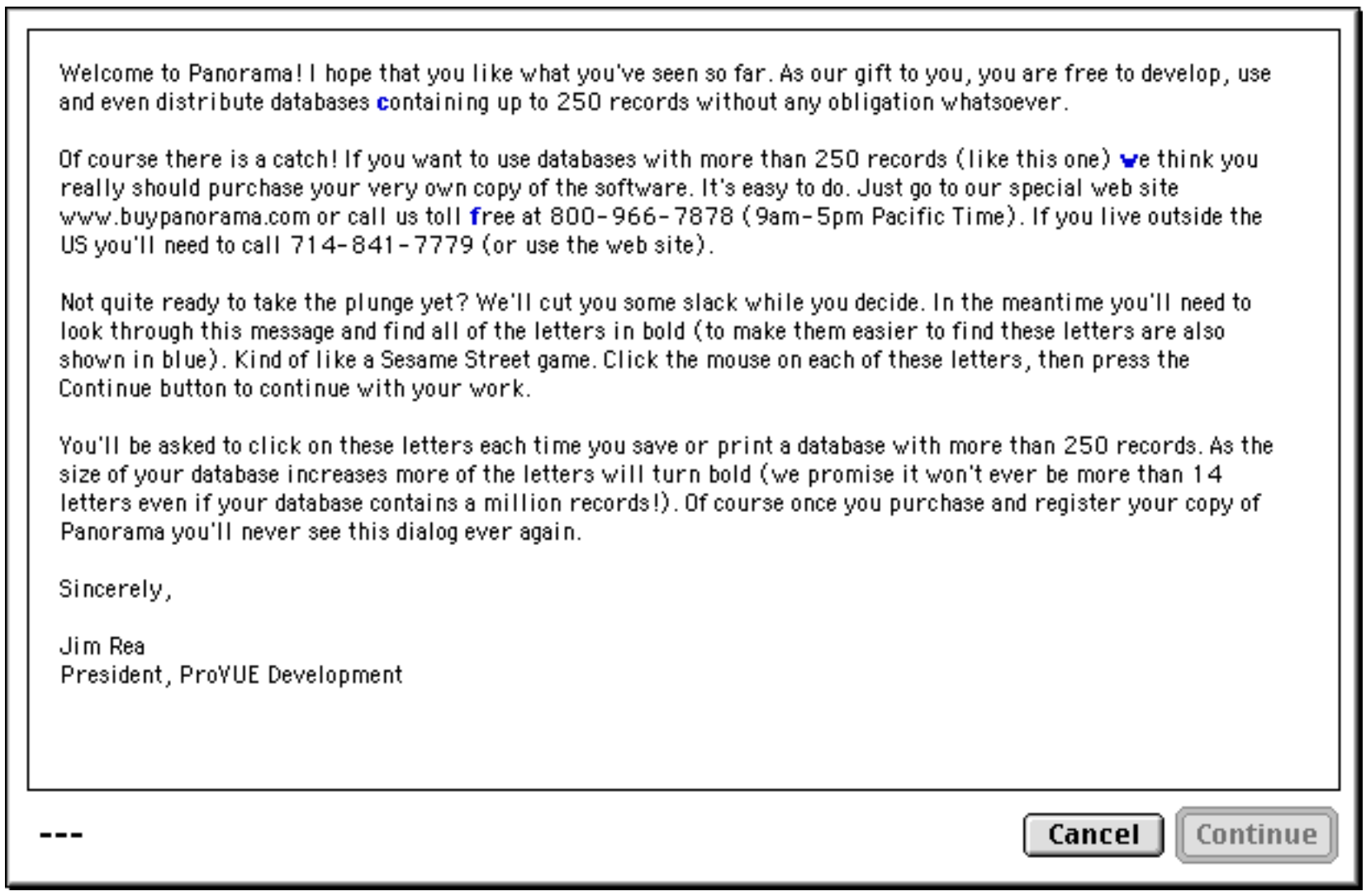
What To Do If You Forget Your Password

If you forget your password simply delete the Panorama Personal License.dat file and re-activate the software from scratch (following the instructions earlier in this chapter).

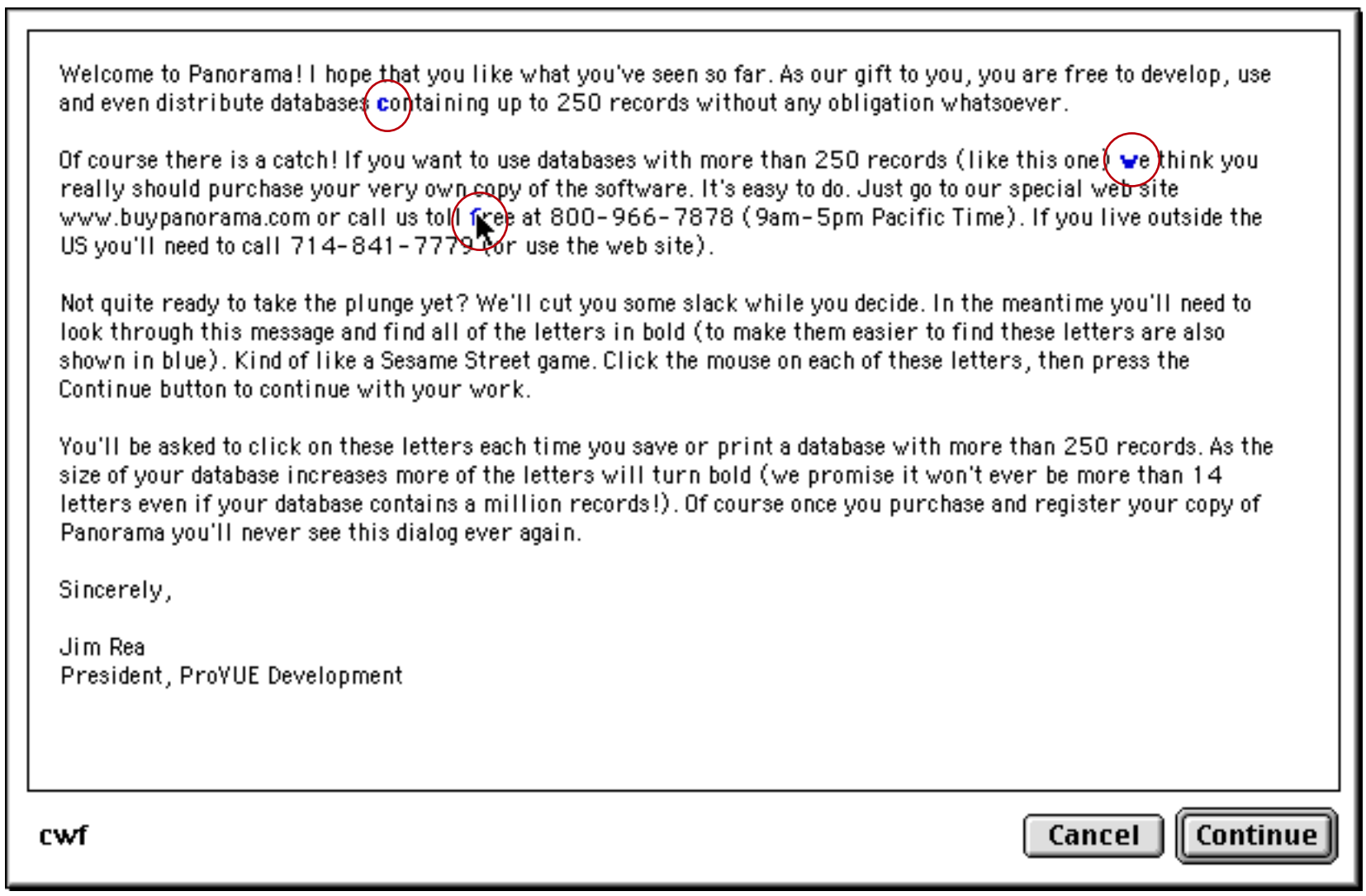
Using Panorama's "Demo Mode"

Before you purchase and activate Panorama (see "Activating the Software" on page 94) you can use the software in "demo mode." This mode allows you to evaluate Panorama before you purchase it. In this mode you can create and use databases with up to 250 records.

If you attempt to save or print a database with more than 250 records the following dialog will appear.



Before you can actually save or print the database you must click on each of the blue bold characters. Depending on the size of the database there will be at least three and as many as 14 of these characters.



Once you have clicked on all of the blue bold characters you can press the **Continue** button to continue with your work.

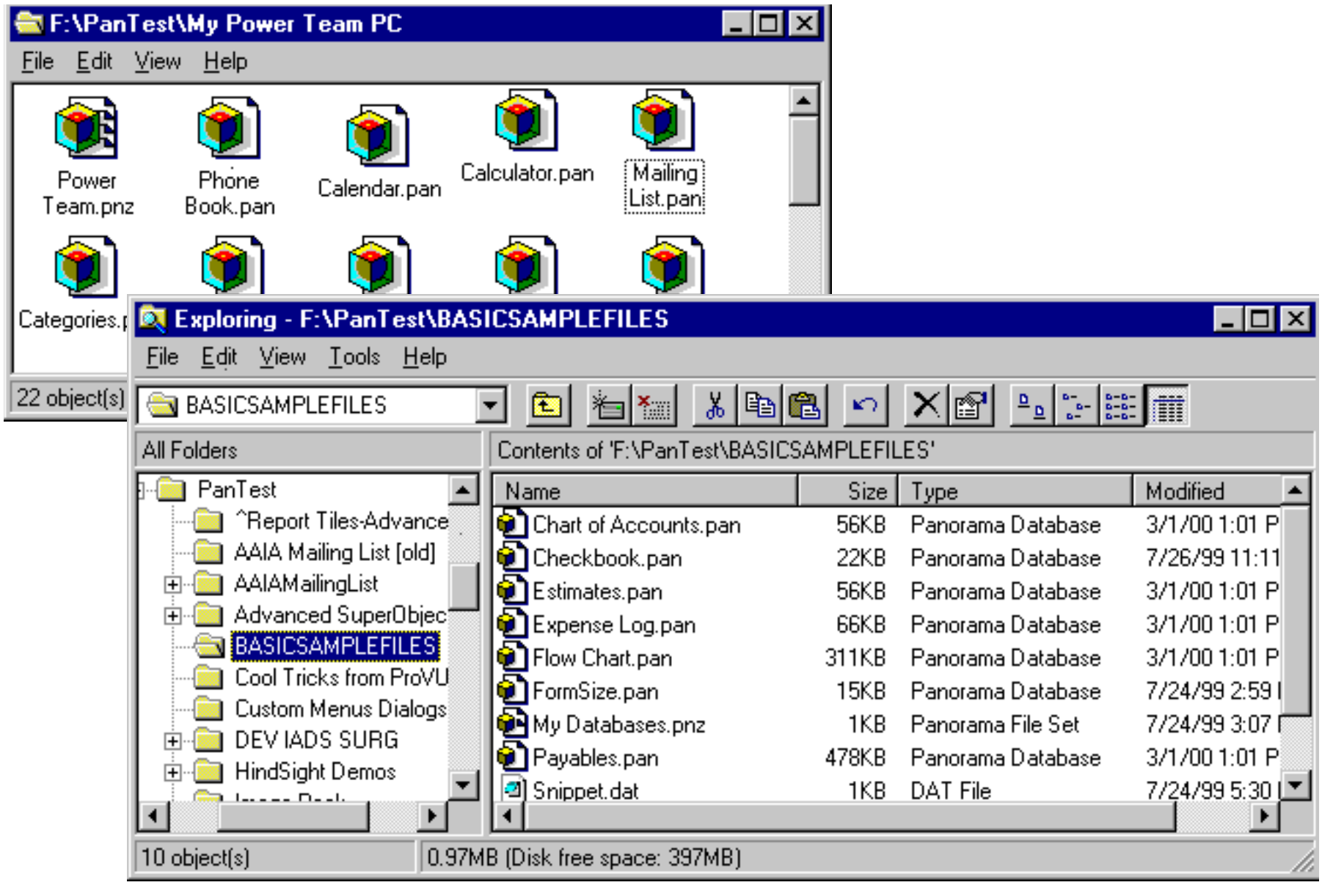
Chapter 1: Files and Memory



Panorama databases are permanently stored in files on a disk drive. (In fact, you'll often find that the words **database** and **file** are used interchangeably.) Each Panorama file contains all the components needed to use the database.

Files, Icons and the Desktop

Before you begin to use Panorama you should be familiar with the basic operation of your computer. Whether you are using a Macintosh or a Windows based computer, files appear in a "desktop" environment that allows them to be located, moved, copied and opened. On the Macintosh this desktop environment is called the **Finder**. On Windows computers this is simply called the desktop, which you can view with **My Computer** or using the **Windows Explorer**. Here are two typical views of folders full of files on the desktop.



There are three different kinds of Panorama icons: **databases**, **file sets**, and the Panorama application itself.

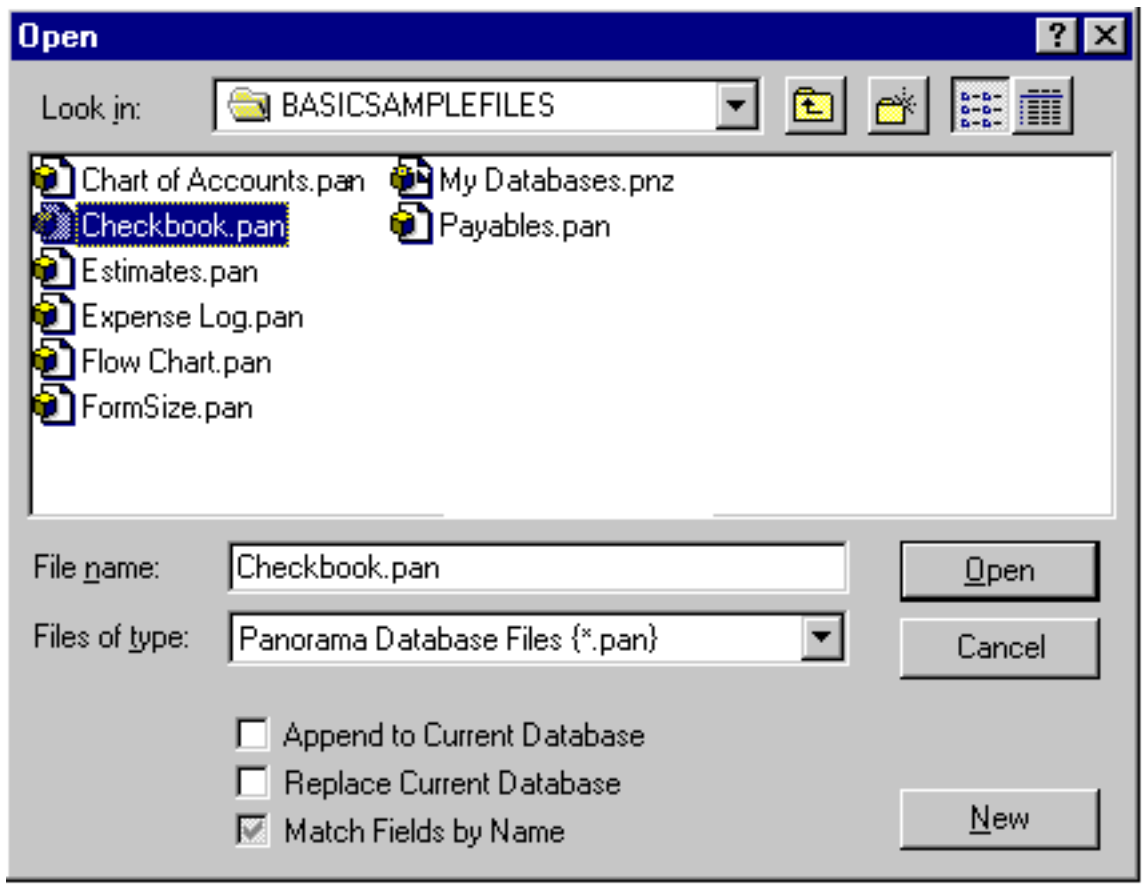
	This is the icon for a single Panorama database. On Windows machines, these files have the extension **.pan**. Double click on this icon to open the database.
	This is the icon for a set of Panorama databases (called a file set). On Windows machines, these files have the extension **.pnz**. Double click on this icon to open the entire set of databases at once.
	This is the icon for the Panorama application itself, which is usually called **Panorama** (Mac) or **Panorama.exe** (Windows). You can double click this application when you want to create a new database without opening an existing database first.

You can manipulate these icons on the desktop any way you like, just like any other files. (However, you should avoid moving or copying the Panorama application itself. Although your disk may contain many database files for different tasks, there should only be one copy of the Panorama program itself.)

Opening a Database

Before you can work with a database you have to open it. From the desktop the quickest way to do this is to simply double click on the file's icon. Or you can select the icon and choose **Open** from the desktop's File Menu.

If Panorama is already running you can use the **Open File** command (File Menu) to open the file. Simply locate the file you want to open and double click on it. Or select the file and press the **Open** button.



This dialog has several additional buttons for importing, combining and creating new databases. For now you can simply ignore these options.

Databases and RAM

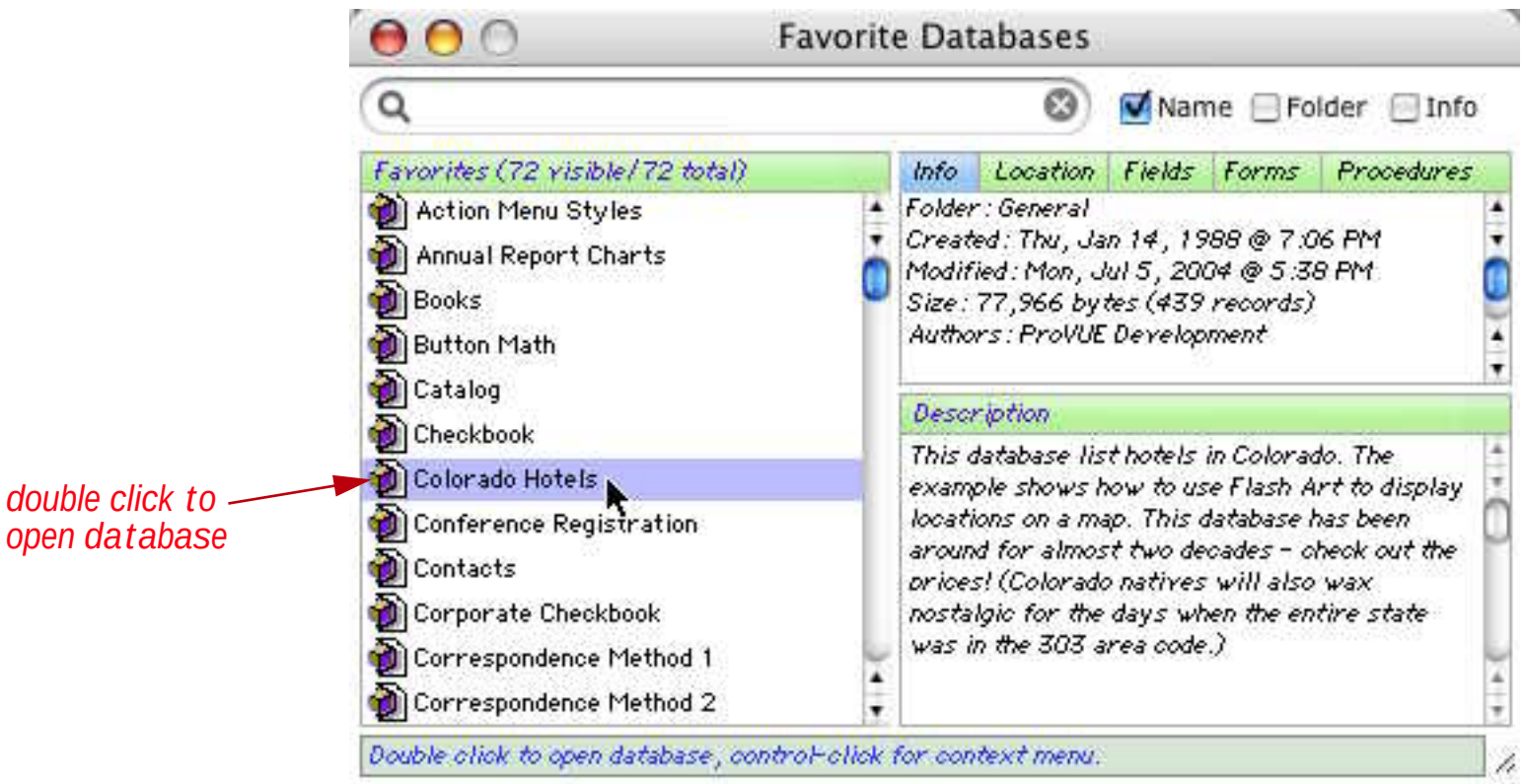
When a database is opened, Panorama copies the information from the disk into the computer's internal electronic memory (RAM). Everything you do to a file takes place in RAM; including data entry, sorting, calculating, and drawing. If you want to store your work permanently, you must save it from RAM back to the disk using the **Save** command.

Most database programs don't take the extra step of copying the database from the disk into RAM before working with it. Since your computer can access data in RAM hundreds or even thousands of times faster than data on the disk, bringing the data into RAM makes Panorama much faster than most other database programs. If you've used other database programs you'll immediately notice how much "zippier" Panorama is compared to the programs you are used to.

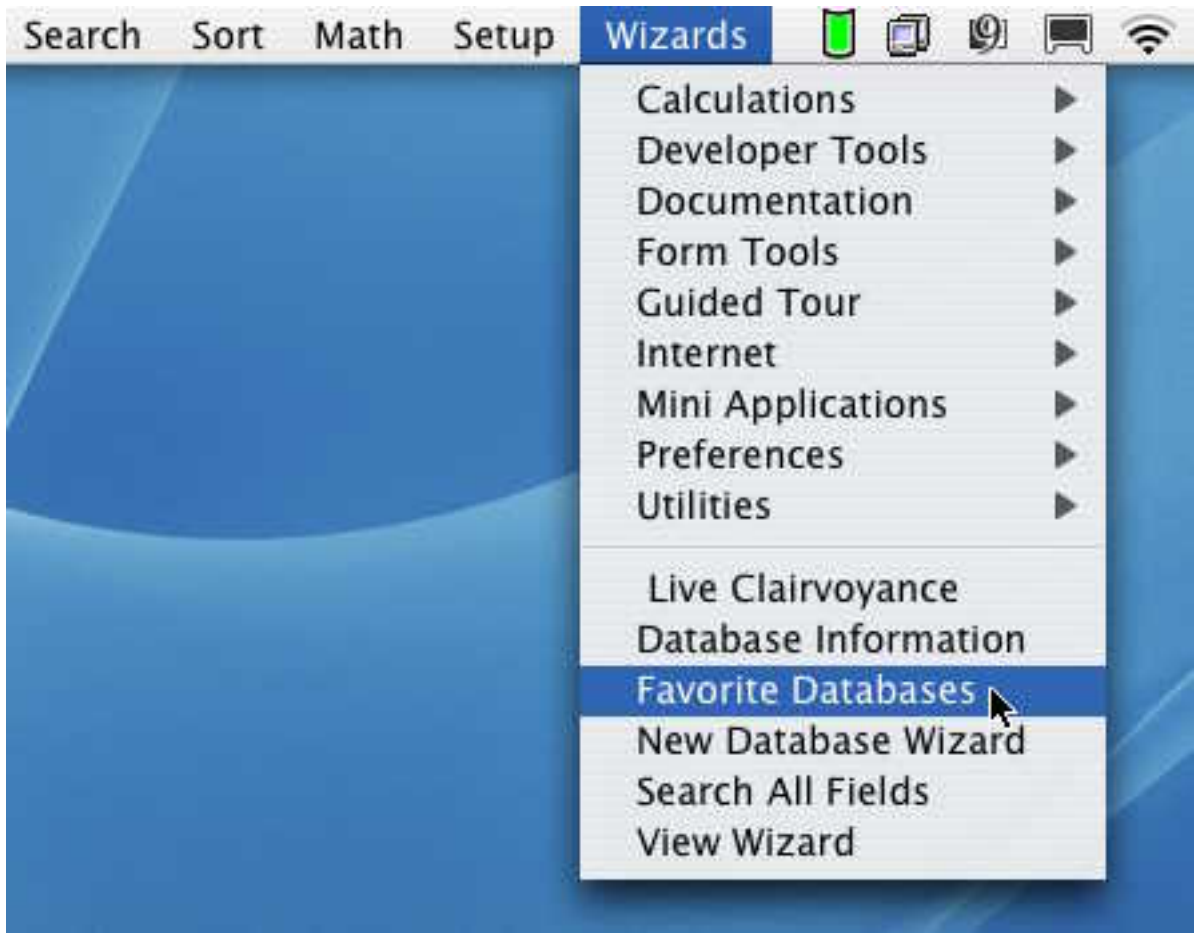
If your computer has enough RAM available, you can open several Panorama databases at the same time. For more information about working with multiple files see "Opening Multiple Files" on page 137.

The Favorite Databases Wizard

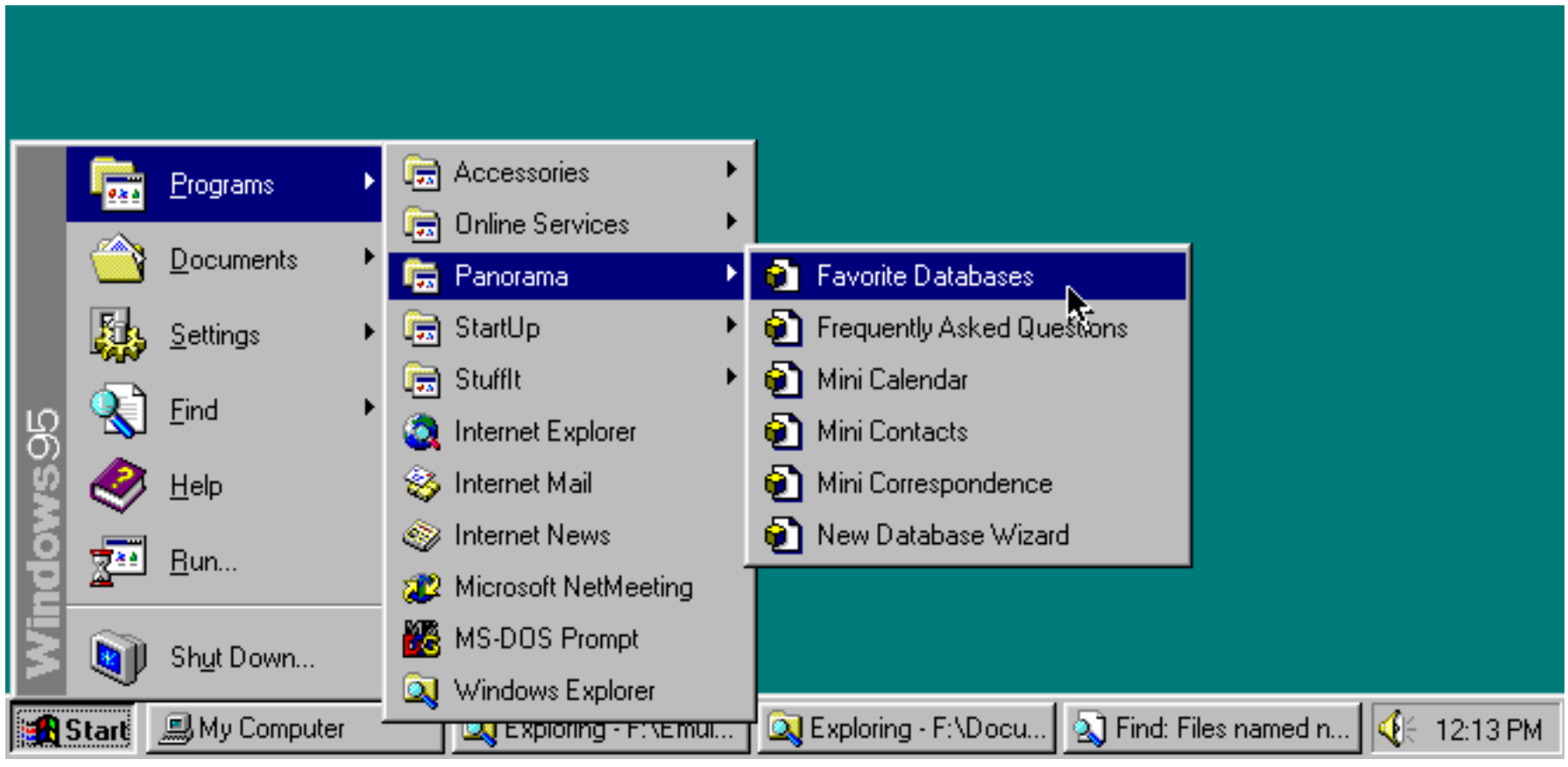
If you use a database frequently you can add it to the **Favorite Databases** wizard. This wizard shows a list of your favorite databases on the left, along with information about the selected database on the right. To open a database simply double click on it. If you hold down the **Control** key (Mac) or double click with the right mouse button (Windows PC) Panorama will automatically close the **Favorite Databases** window after it opens the selected database.



If Panorama is already open you can use the **Wizards** menu to open the **Favorite Databases** wizard.

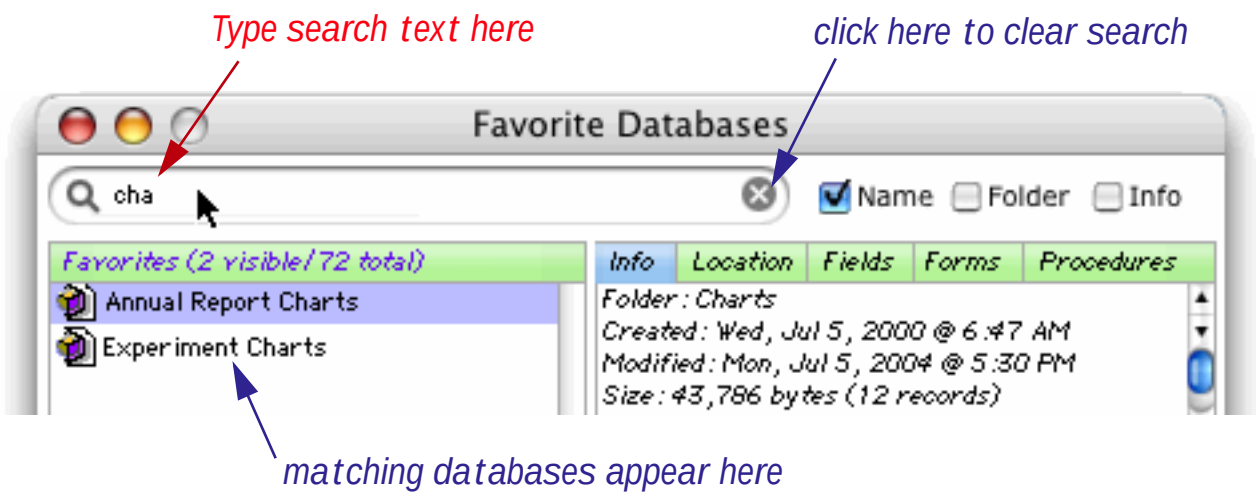


If you are using a Windows computer you can also open the **Favorite Databases** wizard directly from the Windows **Start Menu**.



Searching for a Favorite Database

To search for a favorite database type one or more characters into the search box at the top of the window. As you type each character Panorama will immediately display the favorites that match what you have typed.



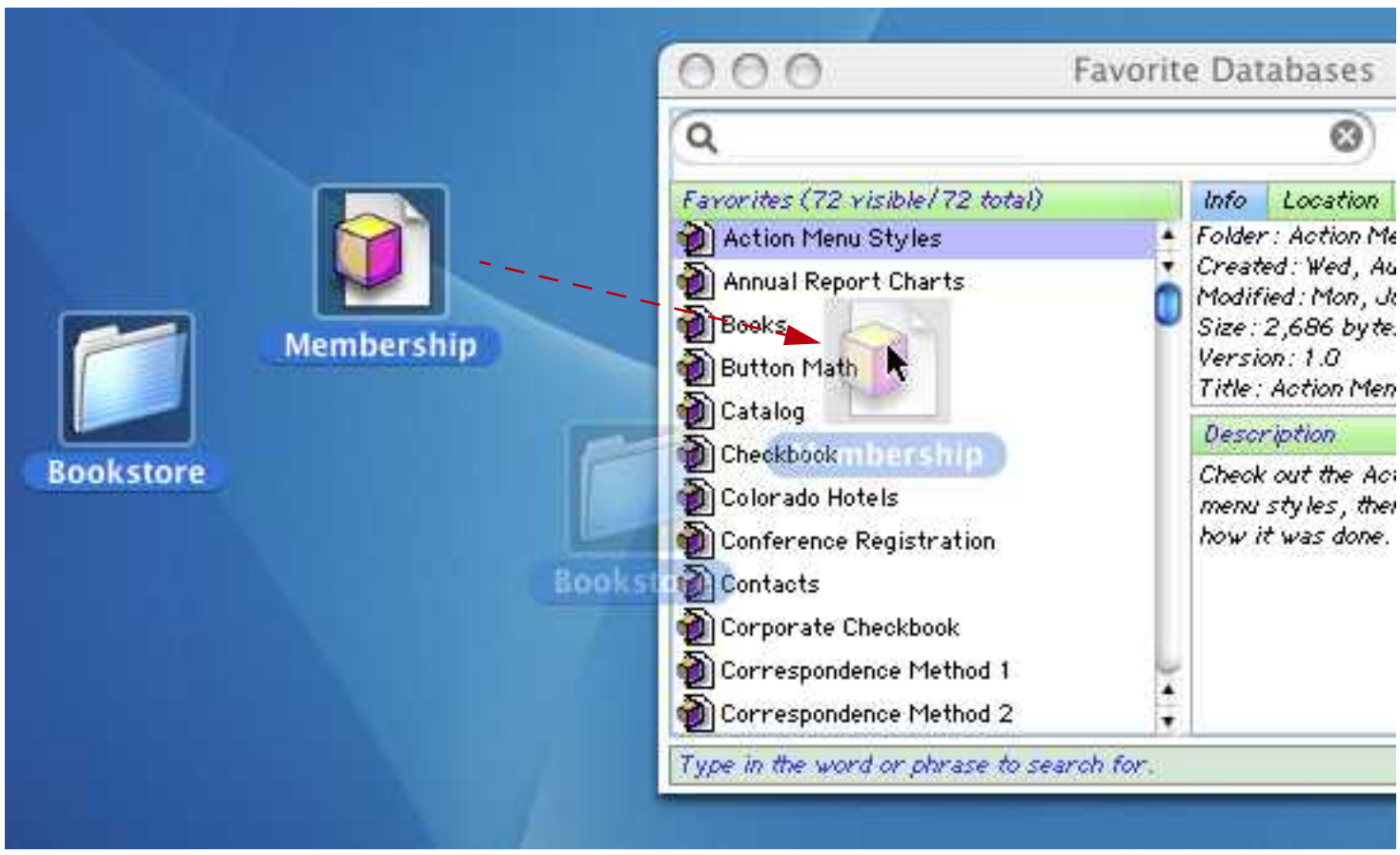
Just to the right of the search box you'll see three checkboxes that allow you to customize the search process. If the **Name** box is checked, the wizard will search the database names. If the **Folder** box is checked, the wizard will search the names of the folders the databases are contained it. If you know the name of the folder the database you want to find is in, this option will help you find the database. If the **Info** box is checked, the wizard will search the additional metadata for each favorite database (see "Using Metadata Information" on page 136). This option allows you to search the favorite databases by title, author, keywords, description and other metadata information.

Another way to open a favorite is to use the **Open** menu. This menu lists all of the favorite databases in multiple columns (if necessary). Just select the database from the menu to open it.

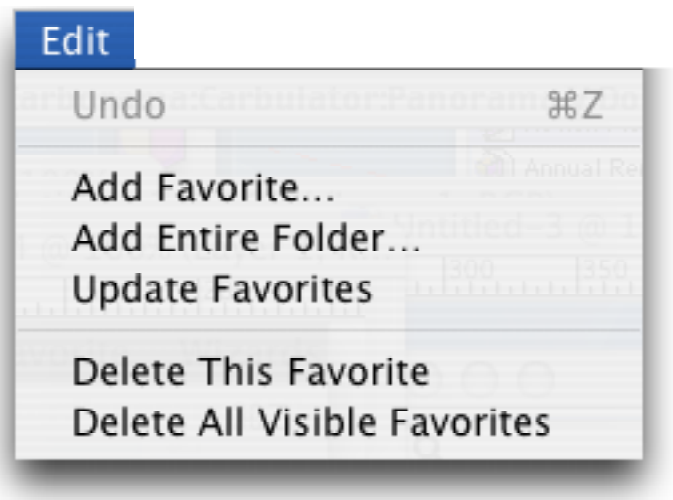


Adding New Favorites

To add a new favorite simply drag the database onto the **Favorite Databases** window. You aren't limited to dragging just one database at a time — you can select and drag a group of databases, or even folders. If you drag a folder all the databases in that folder (and any subfolders) will be added to the favorites as well. (If one or more of the databases you drag are already in the favorites they won't be added again.)



You can also add favorites with the **Edit** menu.



Use the **Add Favorite** command to add a single database. A dialog opens to allow you to select the database you want to open.

The **Add Entire Folder** command adds more databases in the same folder as the currently favorite. When you select this command the wizard will display a list of the databases in this folder. Press the **Add** button to add all of these databases to the Favorites wizard (again, any databases that are already in the Favorites wizard will not be added again).

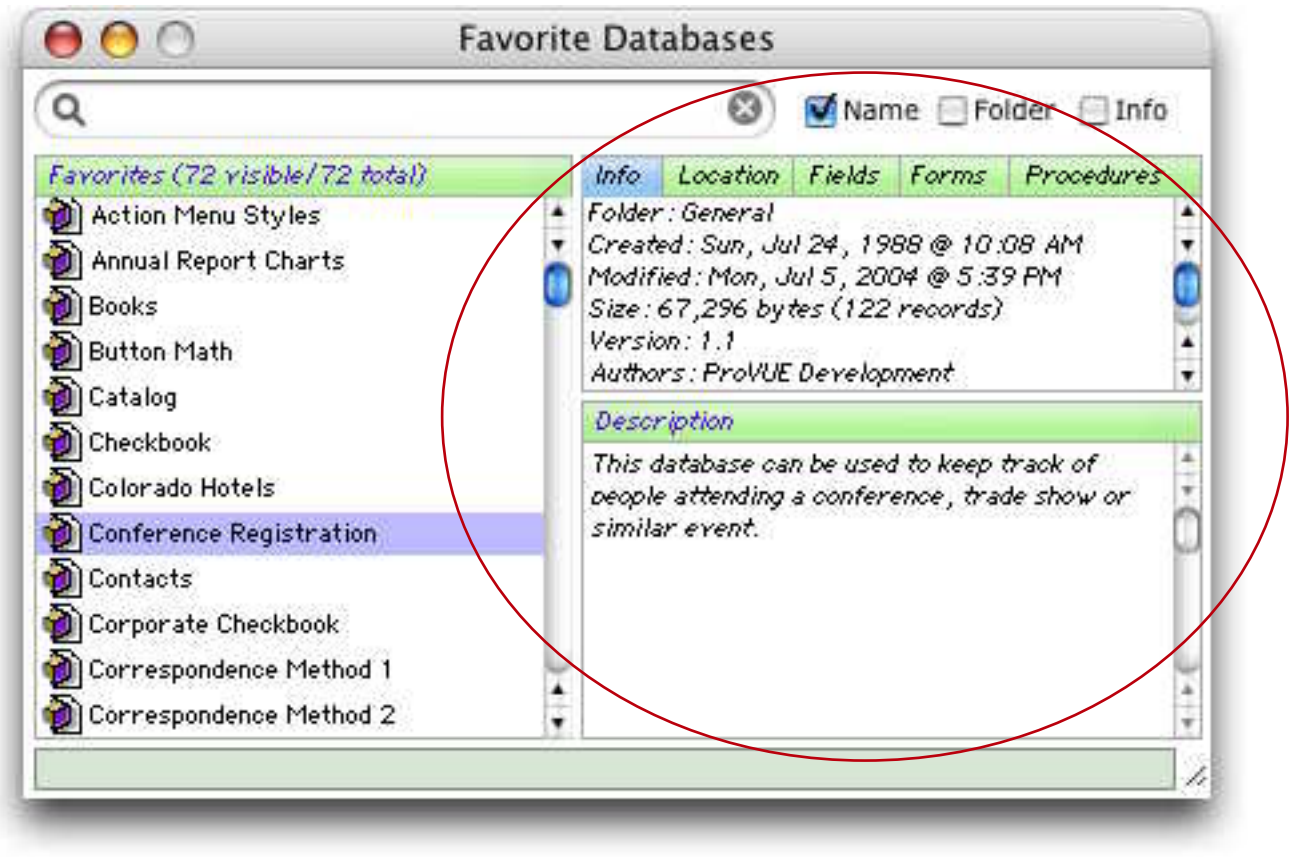


Removing a Favorite

To remove a favorite database from the list simply click once on the name in the list to select it, then choose **Delete This Favorite** from the Edit menu. If you want to delete all of the visible favorites, choose **Delete All Visible Favorites**.

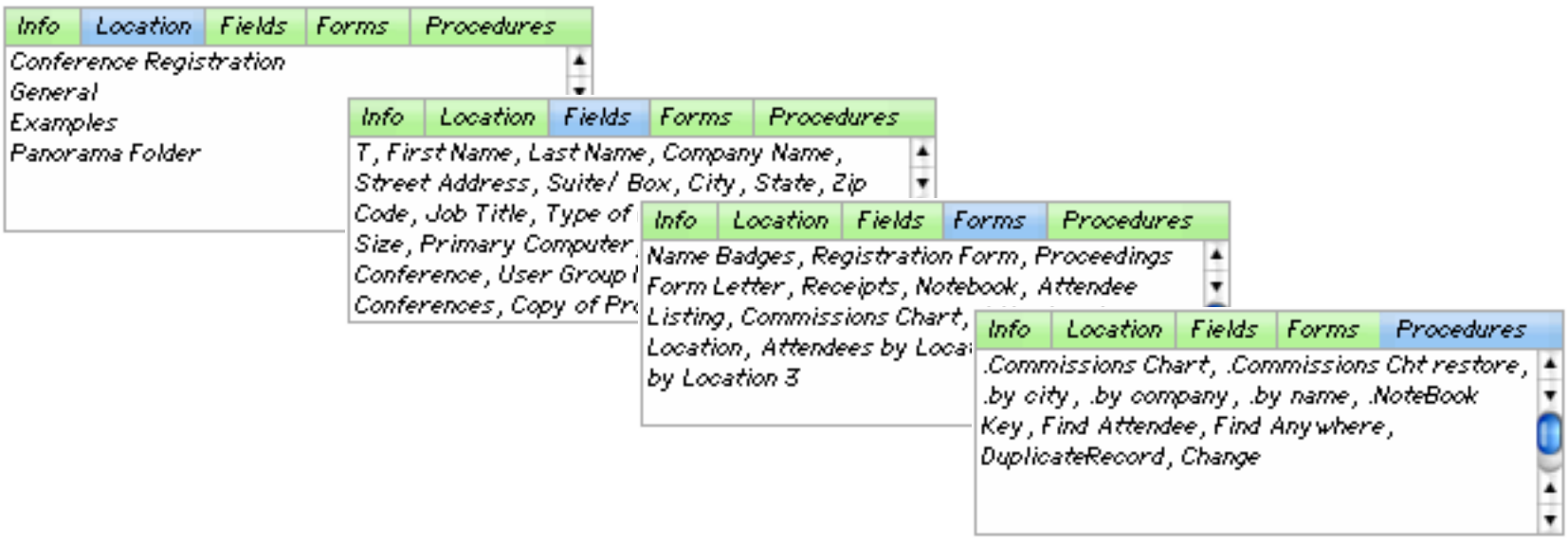
Additional File Information

When you click on a favorite database Panorama will display information about that file on the right hand side of the window, including the file's folder, creation and modification dates and the number of records, fields and bytes (size) and much more.

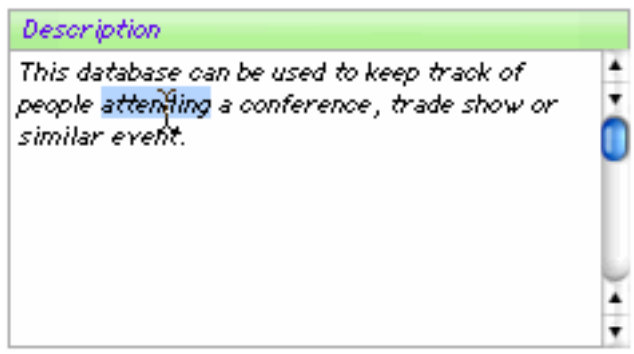


The top section can display five different types of file information. In the illustration above the **Info** button is pressed. This shows that the file is in a folder named General, was created in 1988 and last modified in 2003, contains 122 records, etc.

To see the exact location of this database or a list of fields, forms or procedures in this database click on the appropriate buttons. This database has many fields (see "Fields" on page 247) and forms (see "Introduction to Forms" on page 481).



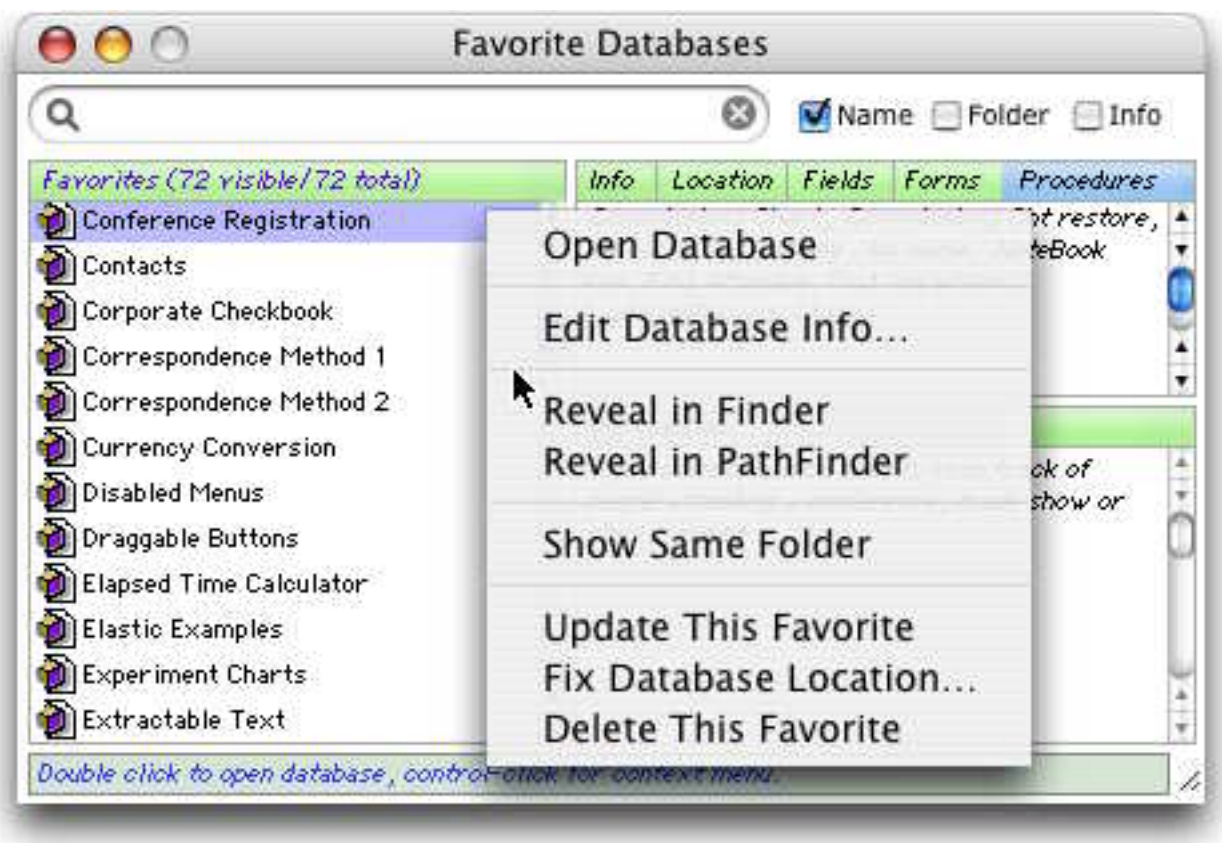
The **Description** section allows you to type in your own description or comments about this database. Click in this area to edit the text.



Press the **Enter** key when you have completed editing the description. (Note: The description is actually stored in the database's metadata, see "Using Metadata Information" on page 136.)

The Favorites Menu

The **Favorites** menu has about a half dozen commands that apply to the currently selected database. This menu appears in the menu bar, and also appears when you hold down the **Control** key and click on a favorite (on most two-button mice this is the same as right-clicking on the favorite).



The **Open Database** command opens the database. This is the same as double clicking on the favorite.

The **Edit Database Info** command opens the dialog window for editing metadata for this database (including the database title, author, description, and keywords, see “Using Metadata Information” on page 136.)

The **Reveal in Finder** command opens the Finder window for the folder containing this database. If you are using the CocoaTech PathFinder shareware application the menu also contains a **Reveal in PathFinder** command.

The **Show Same Folder** command shows all of the favorites in the same folder as the currently selected favorite.

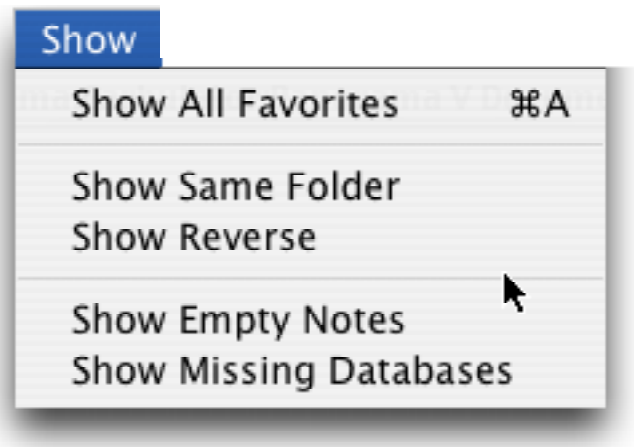
The **Update This Favorite** command checks the original database and updates some of the information about this file, including the creation and modification date, number of records, field names, etc.

The **Fix Database Location** command can be used if a database has been moved to a different drive or folder. When you use this command a dialog appears that allows you to relocate the database. You may want to use this command in combination with the **Show Missing Databases** command in the **Show** menu, which displays all of the favorites that have been moved from their original locations.

The **Delete This Favorite** command removes this database from the Favorites wizard. This command is also in the **Edit** menu.

The Show Menu

The **Show** menu is another way to search for favorites.



The **Show All Favorites** command displays all of the favorites. This is the same as clicking on the cancel search button.

The **Show Same Folder** command shows all of the favorites in the same folder as the currently selected favorite.

The **Show Reverse** command makes visible all of the favorites that are invisible. The currently visible favorites become invisible.

The **Show Empty Notes** command displays all of the favorites that do not have

The **Show Missing Database** command displays the databases that have moved since they were added to the Favorites database. If there are any missing databases, they can be fixed with the **Fix Database Location** command in the **Favorites** menu.

Advanced Database Opening Techniques

The standard techniques for opening a database (double clicking, Open File dialog, Favorite Databases, etc.) will work fine in 99,999 of 100,000 cases. Sometimes, however, you may need to use a more specialized technique to open a database. For example, if a database has lost it's MacOS type/creator information (perhaps by sending it through an e-mail client that doesn't properly support this information, a common problem) the standard techniques will not work. In this section we'll describe the advanced techniques for these very unusual cases.

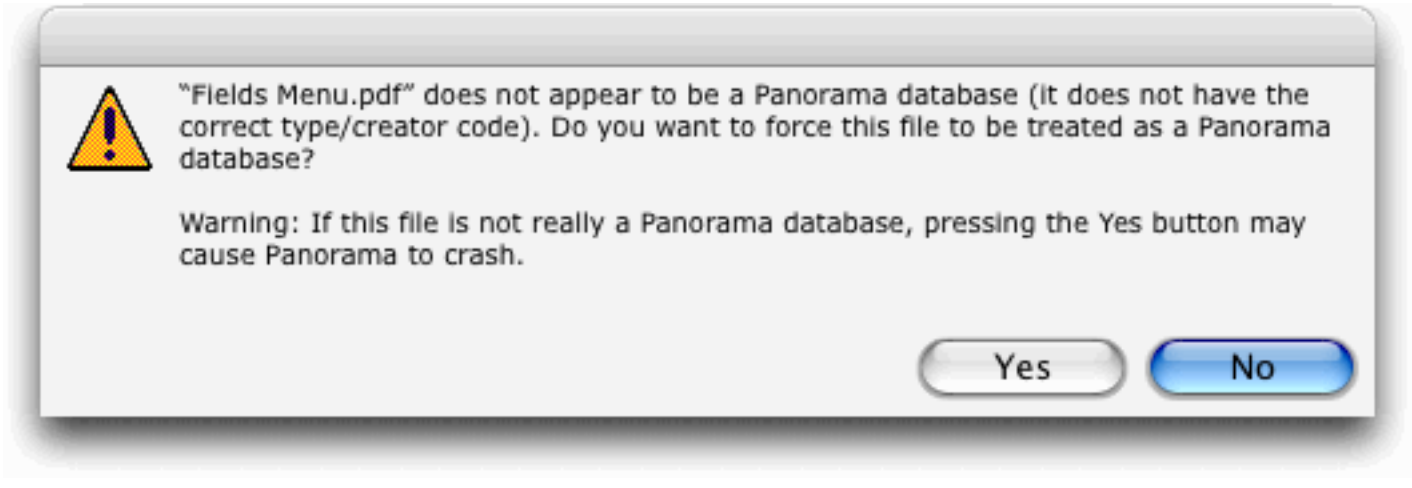
Opening a Database With Incorrect Type/Creator Information

On MacOS systems each type of database is identified by special "type and creator" codes that are kept in the disk directory. Panorama normally sets up these codes when a database is first created so you never have to worry about them. However, if you transfer a Panorama database from a PC computer it will not have the correct "type and creator" codes. These codes can also become missing or corrupted when transferring a file via some e-mail programs. (To avoid this, we recommend compressing the database with a program like Stuffit before sending it via e-mail.)

If you have a file that you know is a Panorama database, but the type and creator codes are missing or wrong, you can still open the database. To do this, open the **Open Database** wizard (in the Utilities submenu). This wizard has a single window with three icons:



To open the database, simply drag it over the icon on the left (labeled Normal). If the type and creator information are Ok, the database will simply open (just as if you had double clicked on it). If the type and creator information are missing or corrupted, a warning dialog will appear:



If you are sure that this file is a Panorama database (unlikely in the example above) go ahead and press the **Yes** button. The wizard will set the type and creator codes to indicate that this is a Panorama database and then it will go ahead and open the file. The next time you need to open this file you should be able to do so normally.

Opening a Database Without Normal Initialization

When Panorama opens a database it normally performs an initialization process. If the window positions were saved (see "Saving Window Positions" on page 132) Panorama will open any windows that were open the last time the database was saved (forms, procedures, etc.) In addition, Panorama will run any custom initialization program that you have set up for this database (see ".Initialize" on page 1511).

In certain rare instances you may want Panorama to skip this initialization process. The most common reason for doing this is if you have made a mistake in the custom initialization program that you have set up. If this mistake prevents the database from opening properly, you can't even get the database open to change the program. Quite a pickle! Fortunately the **Open Database** wizard can handle this situation. Simply drag the database onto the Data Sheet Only icon.



The database will open, but Panorama will skip the normal initialization process, and only the data sheet window will open (no forms).

Opening a Database with No Windows

To open a database with no windows at all, drag it onto the No Windows icon in the **Open Database** wizard. (This opens a database with no windows once. If you want a database to always open with no window use the **Save As** command, see "Saving with No Windows" on page 202).

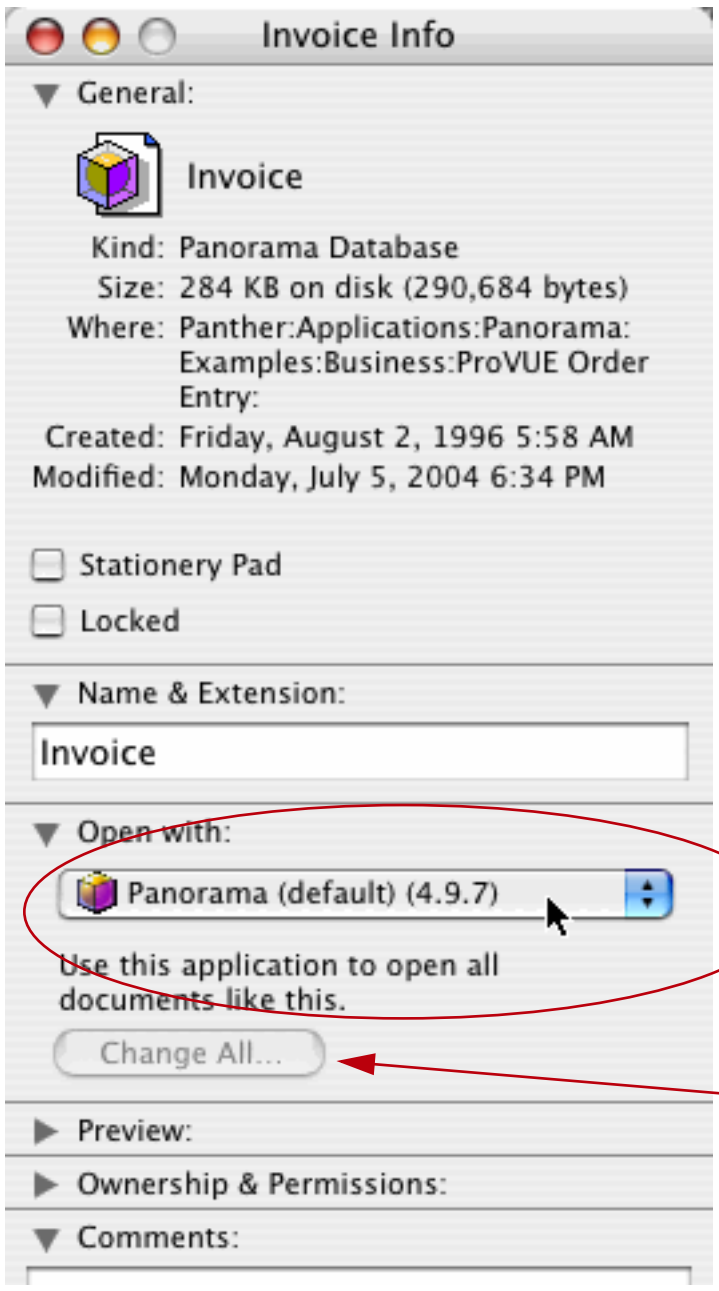
Opening with Multiple Copies of Panorama on Your System

In most cases you'll have only one copy of Panorama on your computer, and you can simply skip this section. However, on MacOS (either OS 9 or OS X) it is possible to have multiple copies of Panorama installed at once. Unless you have a special need, we recommend that you install only one copy of Panorama on your computer and remove all others. (For example, the technical support staff here at ProVUE has both Panorama 4 and Panorama 5 installed at the same time, and the programming staff has additional development versions installed.) However, if you do have a special need to use multiple copies of Panorama you'll want to control which copy of Panorama you are using at different times.

The simplest method to control which version of Panorama is in use is to locate the Panorama application you want to use, double click on it, then use the **Open File** dialog (see above) to open the database you want to use. If you are using OS X you can put multiple copies of Panorama in the dock, and click on the one you want to use, then use the **Open File** dialog to open each database.

Using the **Open File** dialog is a bit awkward. Another method for opening a database with a specific copy of Panorama is to drag the database onto the actual specific Panorama icon itself (or onto the dock icon for that copy of Panorama).

If you are going to use one copy of Panorama most of the time you can tell Mac OS X to open that version automatically when you click on any database. To set this up, click once on the icon of any Panorama database, then choose **Get Info** from the File menu. This opens an Info window for that database.



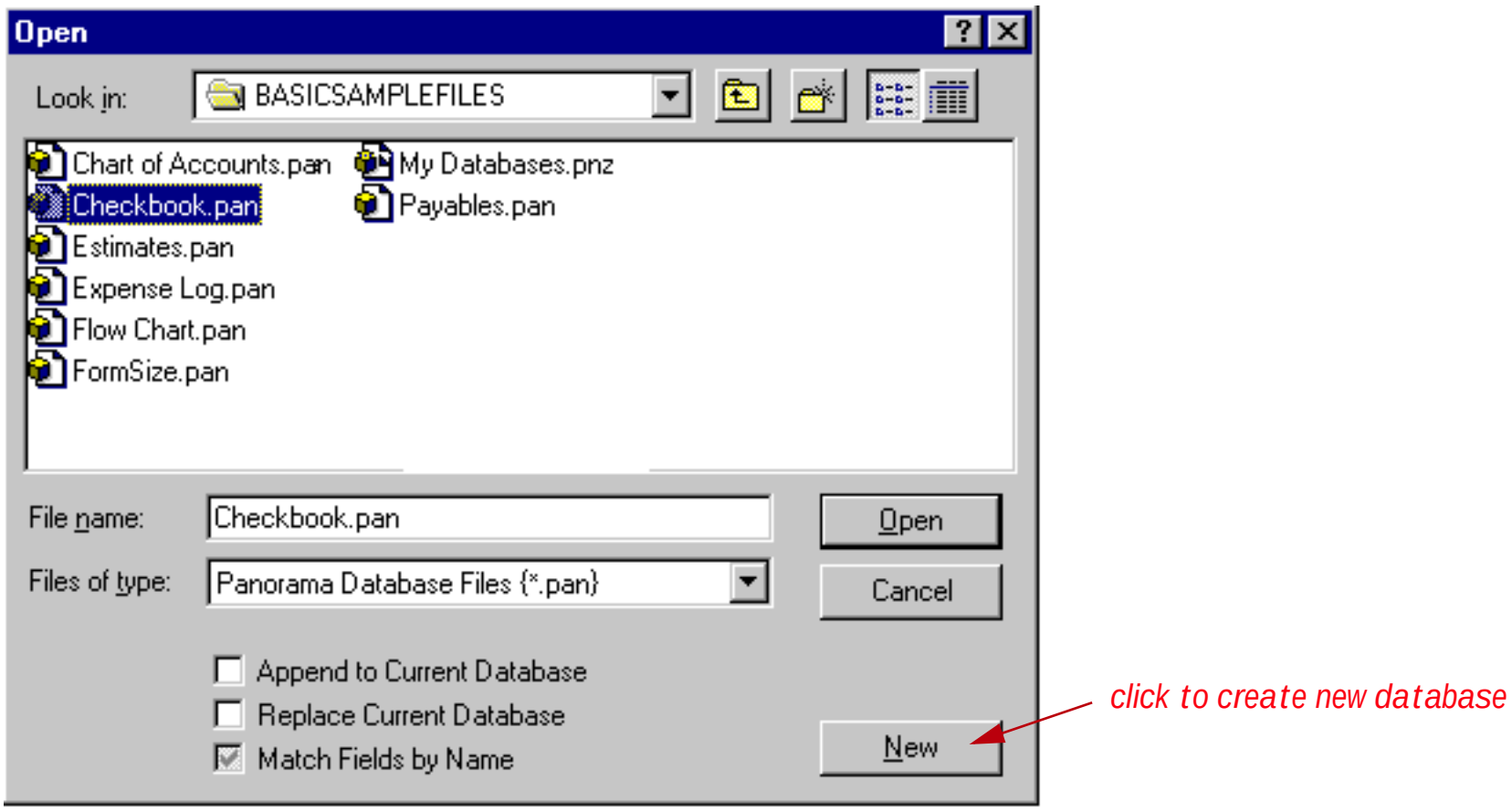
use pop-up menu to select specific copy of Panorama you want to use

Click here to make this the default version of Panorama when double clicking on any database.

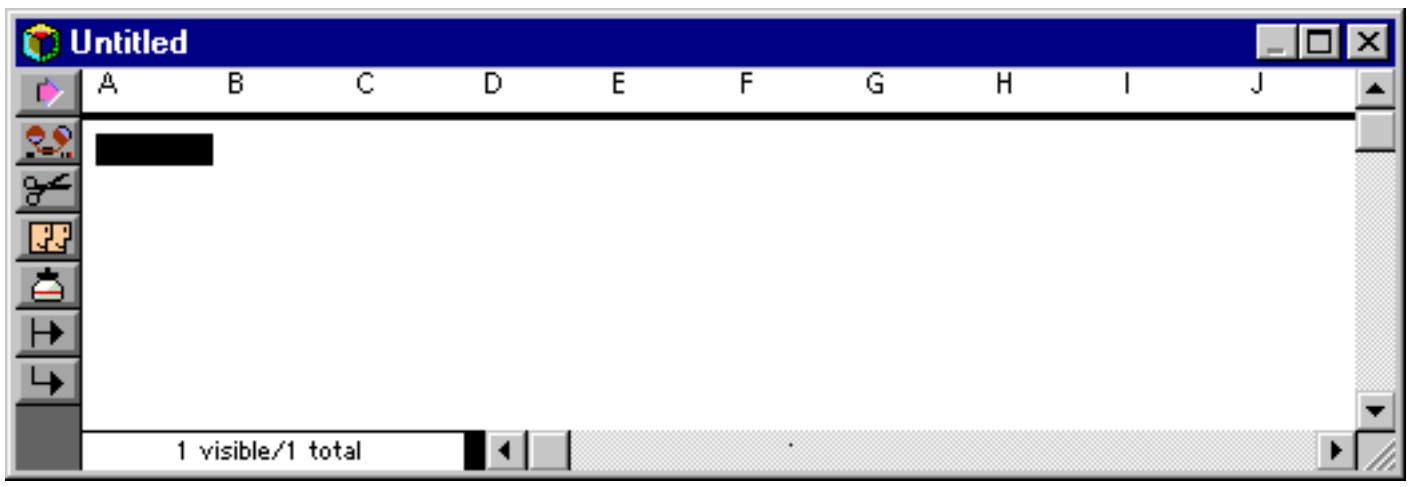
As shown in the illustration above, you can use the Open with: pop-up menu to select which version of Panorama you want to launch when you double click on this database. Press the **Change All** button if you want this copy of Panorama to launch when you click on any Panorama database.

Creating a New Database

To create a new Panorama database choose **Open File** from the File Menu, then press the **New** button.



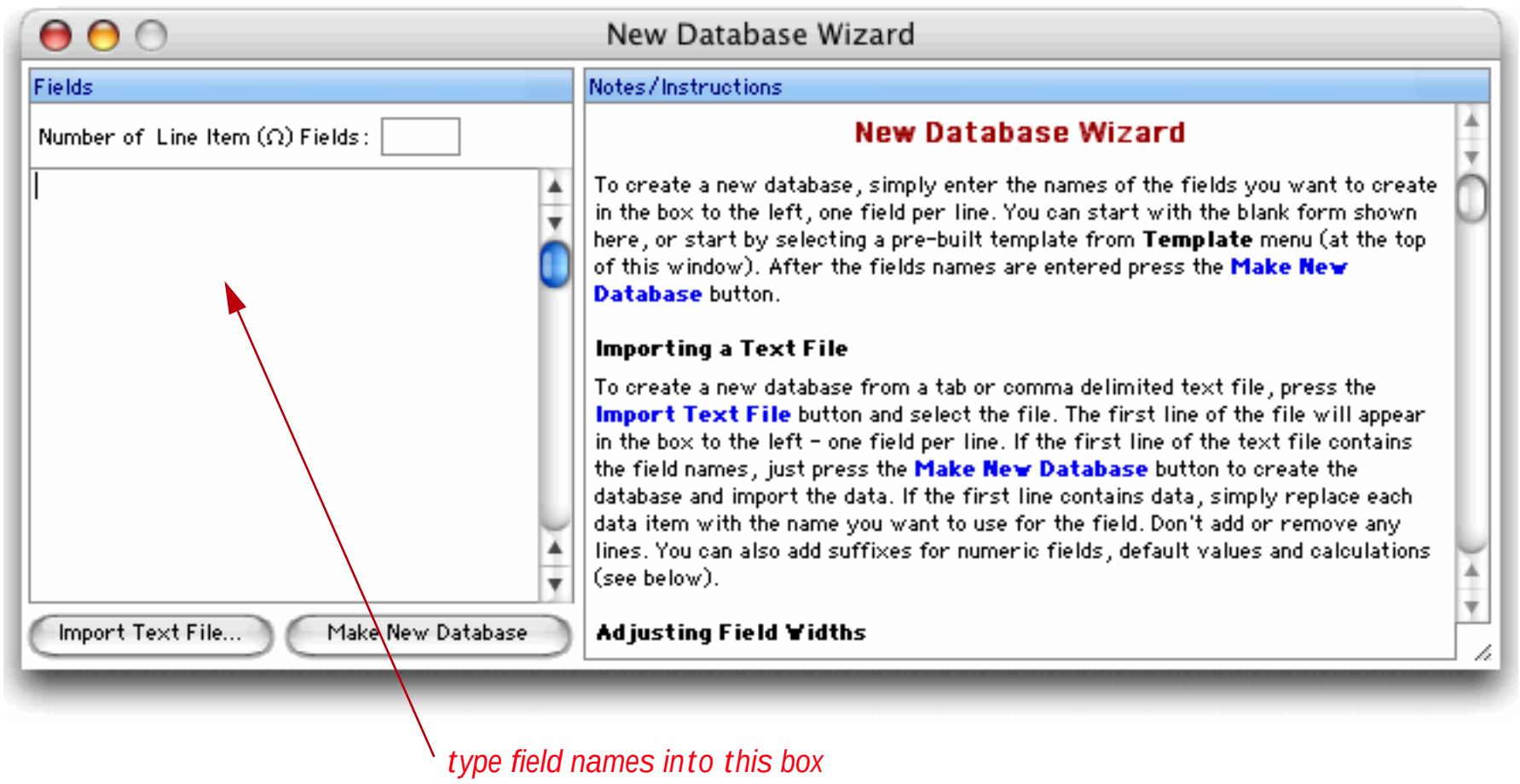
A new database has ten fields and is called Untitled. These fields are named alphabetically: A through J (You can change the names later). You can start entering data right away or you can change the fields before you begin (See “Fields” on page 247).



If you are at the desktop, you can create a new file by double clicking on the Panorama program icon. When you click on the Panorama icon, the Open dialog appears automatically. Press the **New** button to create the new file.

Using the New Database Wizard

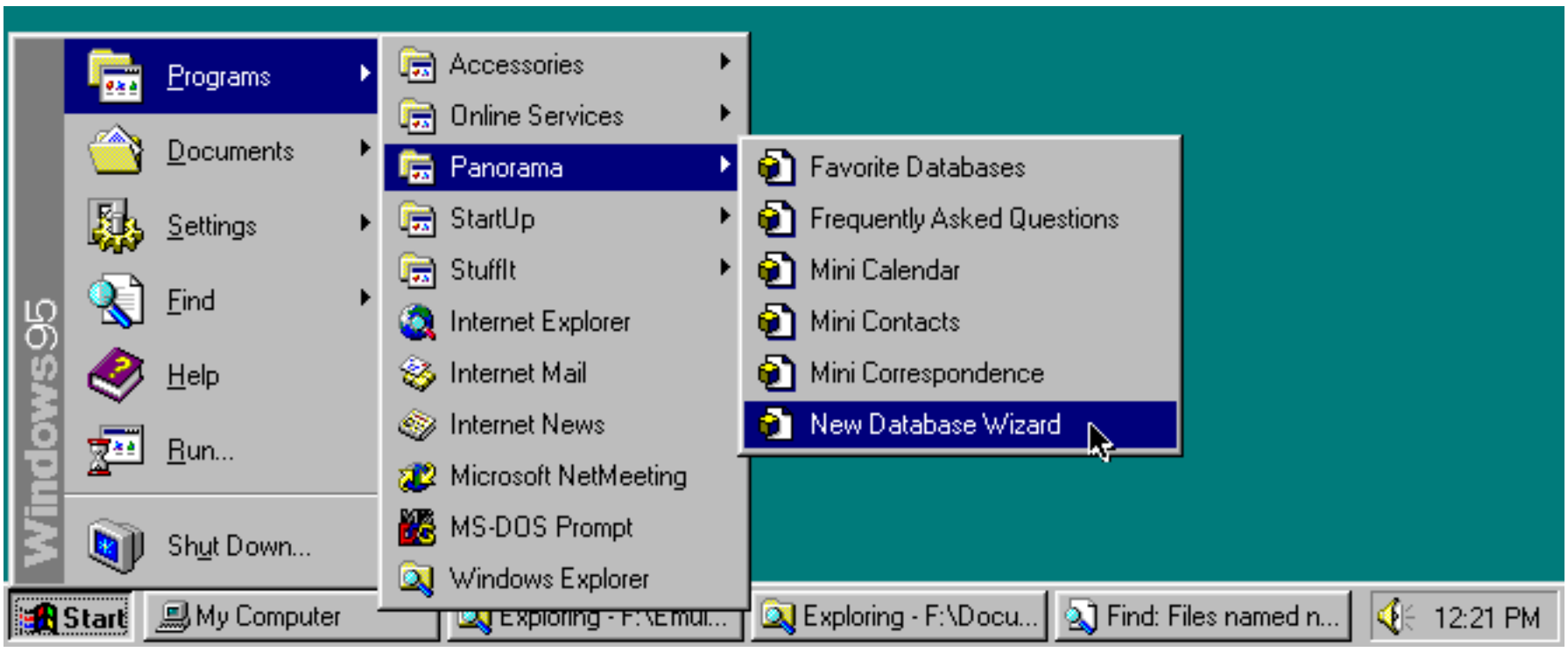
To help make creating new databases easier Panorama includes “wizard” for creating new databases.



If Panorama is already open you can use the **Wizards** menu to open the **New Database Wizard**.

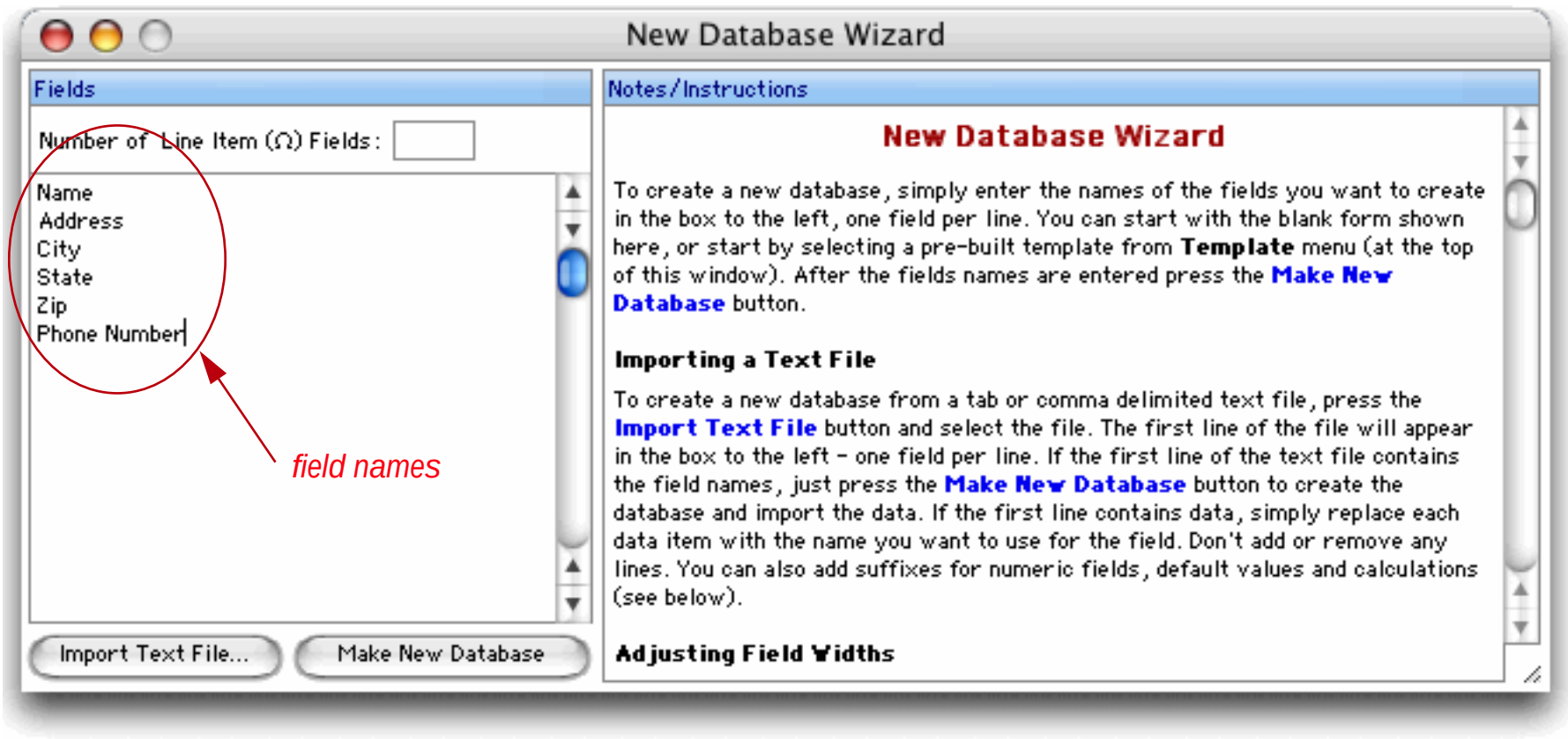


If you are working on a Windows computer you can also open the **New Database Wizard** directly from the **Start Menu**.

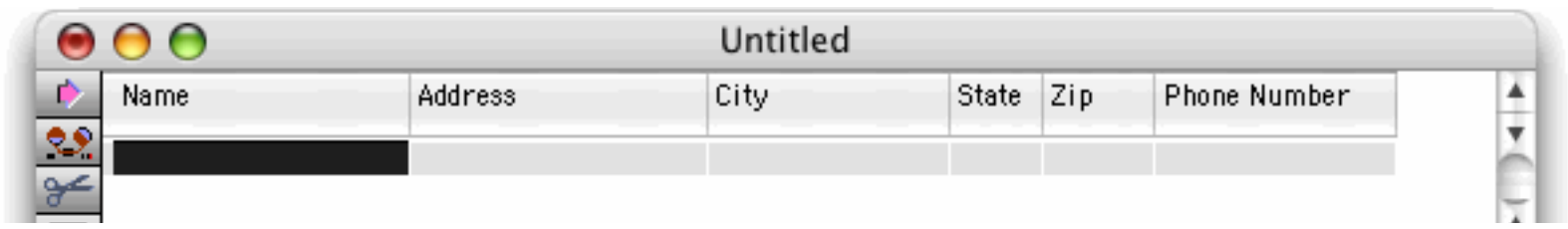


Creating a Database with the Wizard

To create a database simply type in the name of each field, one per line.



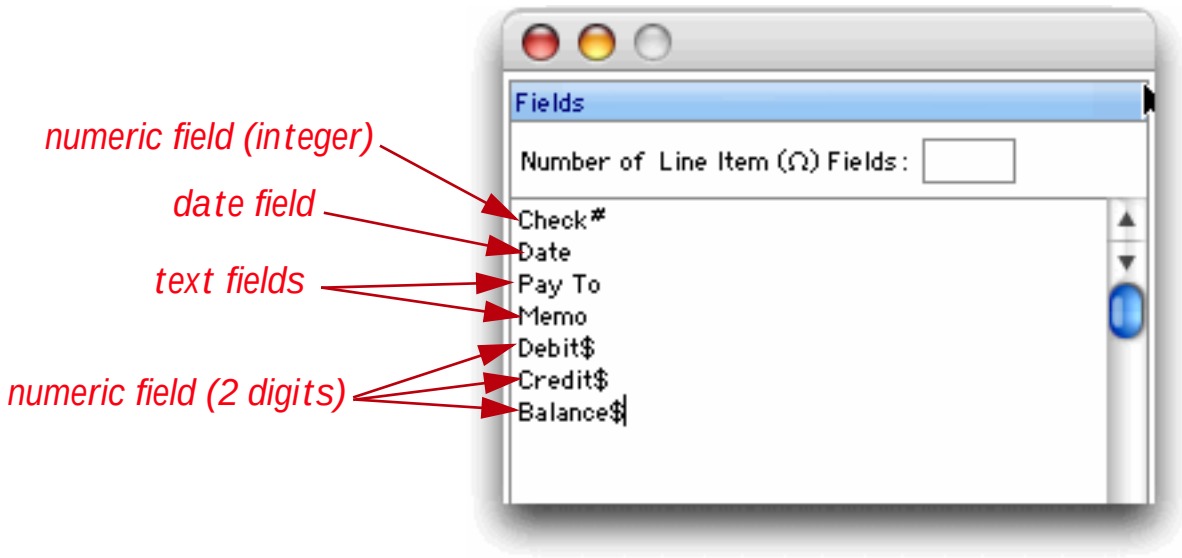
To create the database simply press the **Make New Database** button.



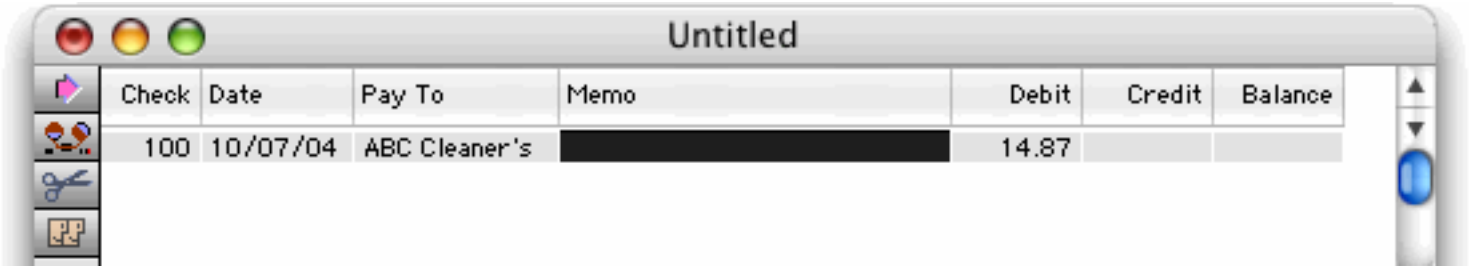
Panorama automatically sets the width of each field. See "Changing the Width of a Field" on page 251 to learn how to manually adjust the width of a field. Before you continue you should save your new database (see "Saving a Database" on page 131).

Creating Numeric and Date Fields

In Panorama, all data is not the same. To get the most out of a database, Panorama needs to know what type of data you intend to store in each field — text, numbers or dates (see “Data Types” on page 285). The **New Database Wizard** normally creates fields designed for holding text. Adding #, $ or . at the end of a field name tells the wizard to create a numeric field (see “Numeric Data” on page 289). A field with a name ending in # (for example Check#) can hold integer values (for example 1, 23 or 456). A field with a name ending in $ (for example Price$) can hold numbers with two places after the decimal point (for example 4.87 or 783.98). A field with a name ending in . (for example Weight.) can hold any floating point value.



Press the **Make New Database** button to create the new database. Notice that the # and $ suffixes that were typed into the wizard are not included in the final field names. We’ve typed in one line of data to show the different types of data stored in each field.



Any field who’s name contains the word Date (for example Date, Start Date or Check Date) will be created as a field for date values (see “Dates” on page 294). In the example above the second field (Date) is just such a field.

Default Values

To assign a default value to a field (see “Default Values” on page 333) follow the field name with an = symbol followed by the default value. For example:

```
Country=USA

ShipMethod=Federal Express
```

To repeat the previous value in this field (ditto) use the " symbol.

```
City="

Date="
```

To automatically increment a numeric or date field use +, or plus followed by a number (+1, +2, +5, etc.).

```
Check Number#=+1

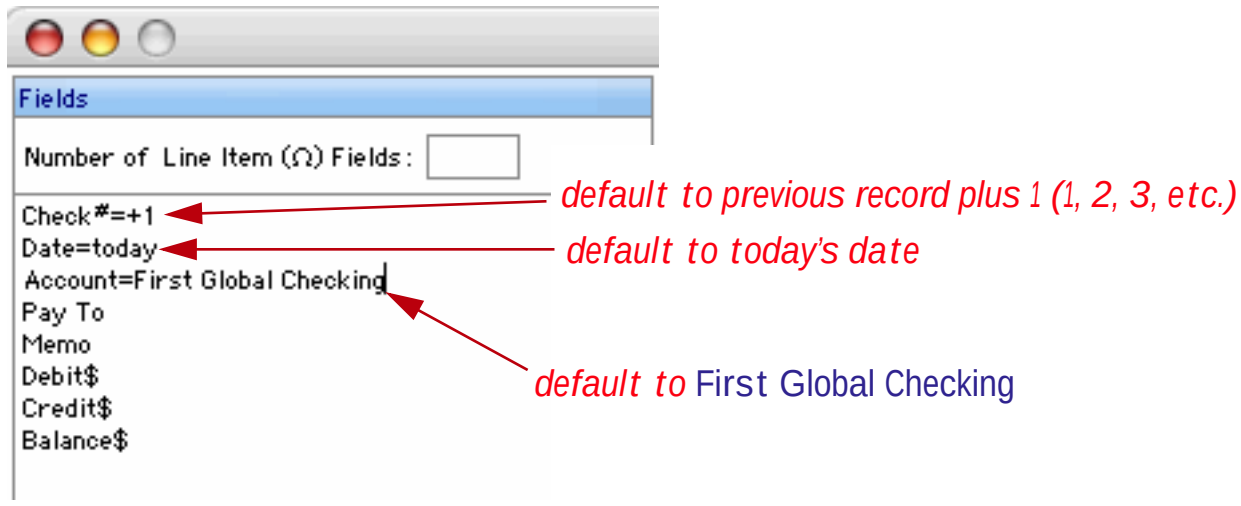
Invoice#=+
```

To default to today's date use =today.

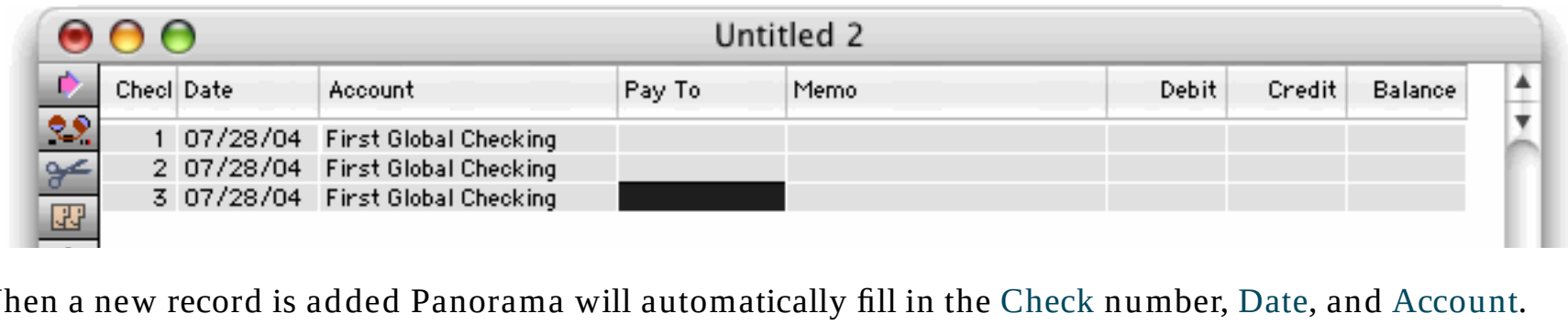
```
Date=today

InvoiceDate=today
```

In this example three fields have default values.



This shows the result of creating this database and adding several fields.



When a new record is added Panorama will automatically fill in the Check number, Date, and Account.

Automatic Calculations

To assign an automatic calculation to a field (see “Automatic Calculations” on page 340) follow the field name with an = symbol followed by the formula. The formula must be surrounded by (and) parentheses. Using the wizard you cannot assign both a default value and a calculation to the same field. Use Panorama's Design Sheet (see “The Design Sheet” on page 252) if this is necessary. For example:

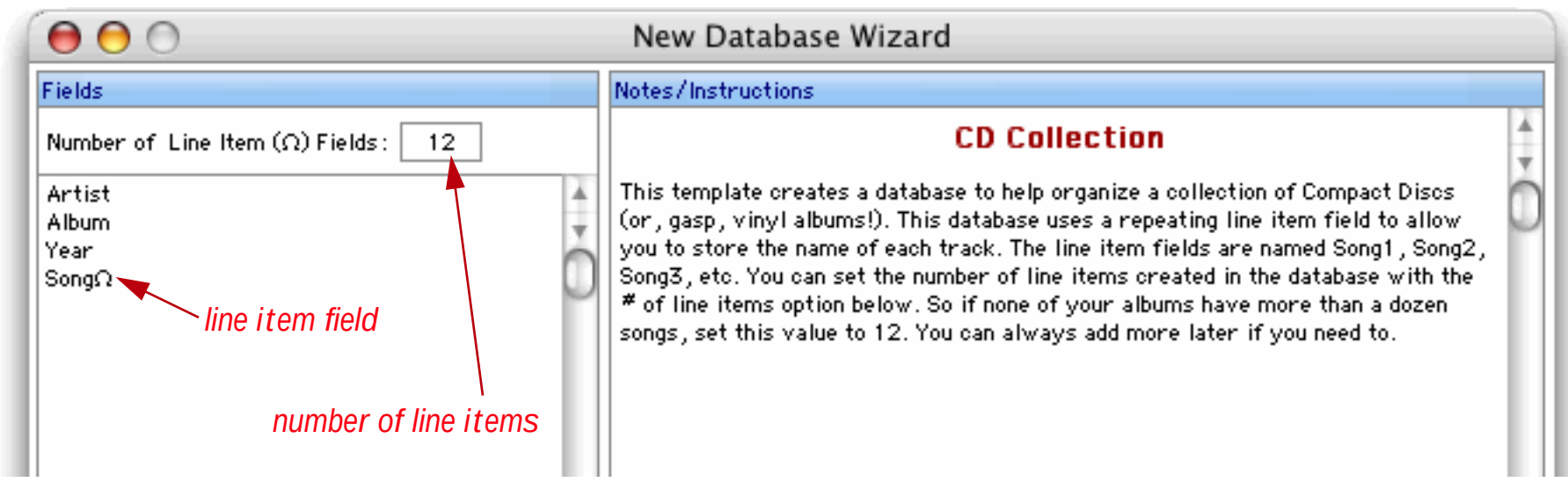
```
Total$=(Subtotal+Shipping+Tax)

Amount$=(Qty*Price)
```

Line Items (Repeating Fields)

Some databases contain several similar fields repeating within each record. For example, an invoice usually contains several quantities, product descriptions, product prices, etc. These fields are often called **Line Items** because they repeat for each line on the invoice (see "Repeating Fields (Line Items)" on page 262). In Panorama these line item fields are created by adding a numeric suffix to the root field name, for example `Qty1`, `Qty2`, ... `Qty15`. In the **New Database Wizard** you don't have to type in each of these multiple field names. Instead, you can simply enter the root field name followed by the Ω symbol, for example **QtyΩ**. If you are using a Macintosh computer press **Option-Z** to create the Ω symbol. On a Windows PC press **Alt-0166** to create the symbol.

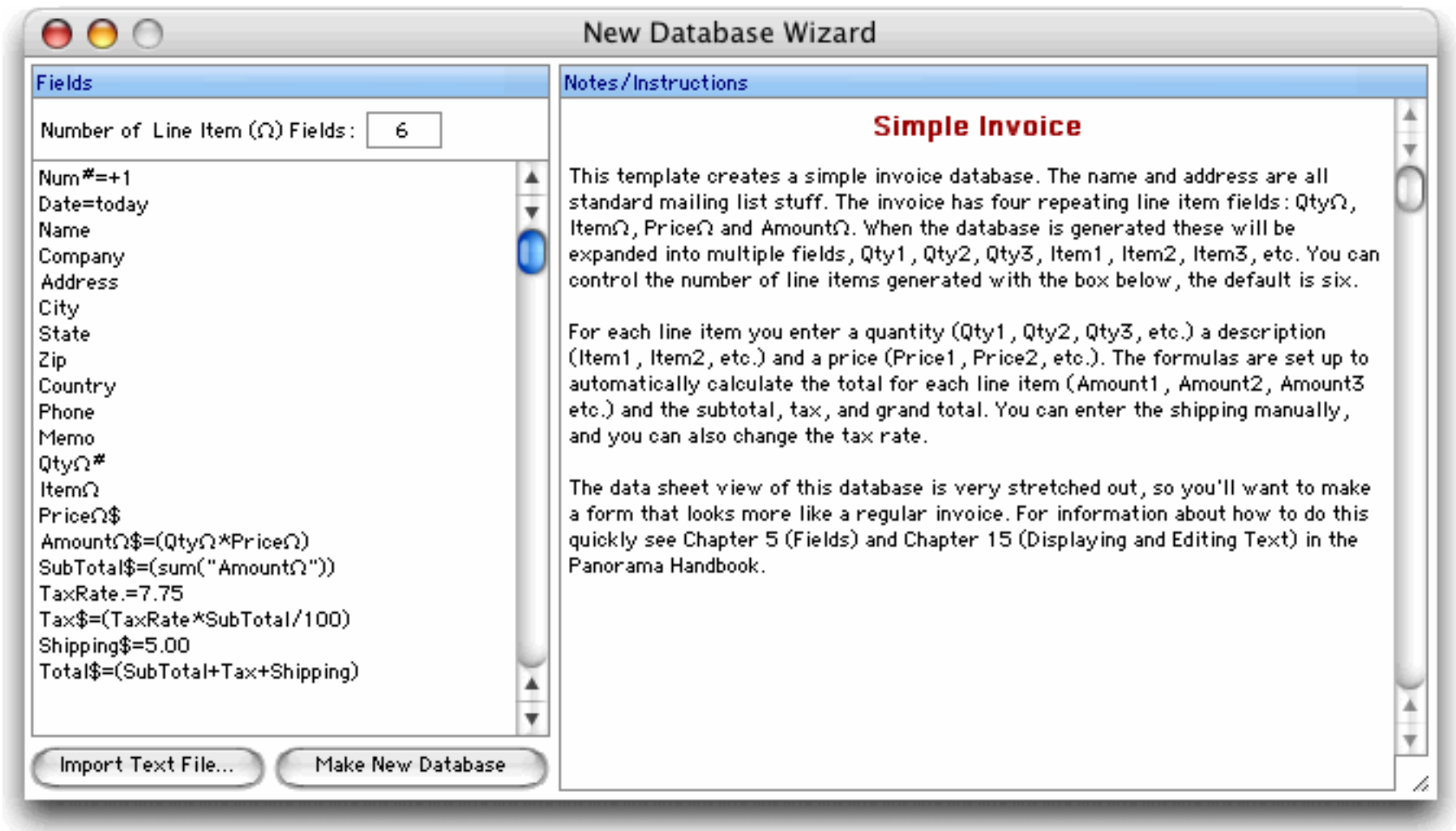
Here is an example of a database with a single line item field. You must type in the number of line item fields that you want to create (in this case 12).



The actual database (created by pressing **Make New Database**) contains 12 separate Song fields, Song1, Song2, Song3 etc.



Here's a more complex example that creates an invoice database. This database will have four line item (repeating) fields, Qty, Item, Price and Amount. Notice that the Ω symbol can be used in a formula (see "Line Item Fields" on page 1198).



This illustration shows a small portion of the resulting database.



This database is extremely wide and doesn't work very well in the Data Sheet view. Instead you'll probably want to create an invoice form something like this (we've cut out the middle to fit it on this page).

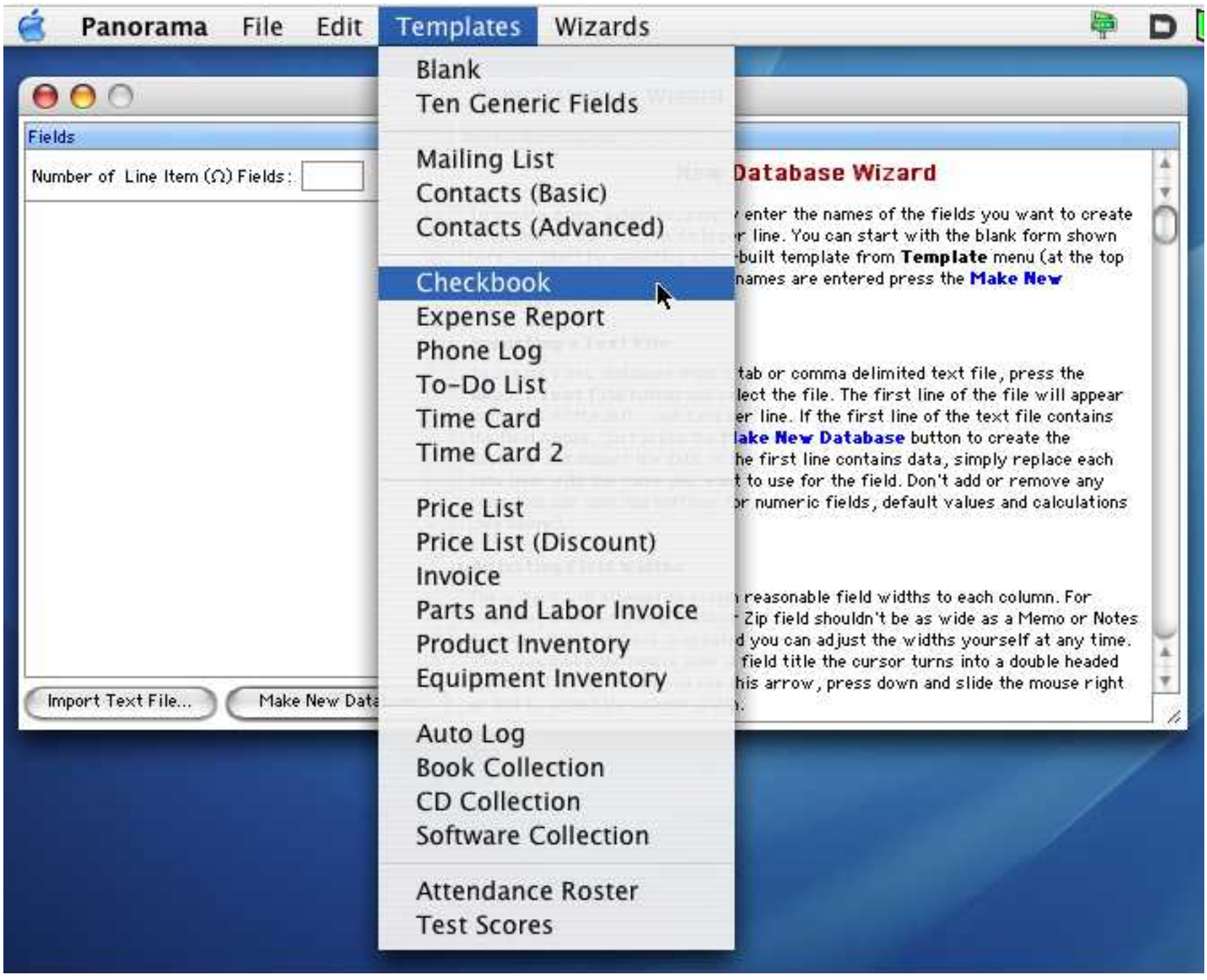
Qty	Description	Price	Total
2	Box Car	4.75	9.50
1	Oil Tank	18.00	18.00
1	Steam Passenger Engine	84.95	84.95
1	Diesel Passenger Engine	29.95	29.95
6	Insulated Rail Joiner (24)	0.90	5.40
1	Semaphore	18.99	18.99

Invoice	101	Subtotal	174.59
Date	November 3, 1990	Tax	8.73
◯ Cash ◉ Check ◯ Visa/MC		Total	183.32

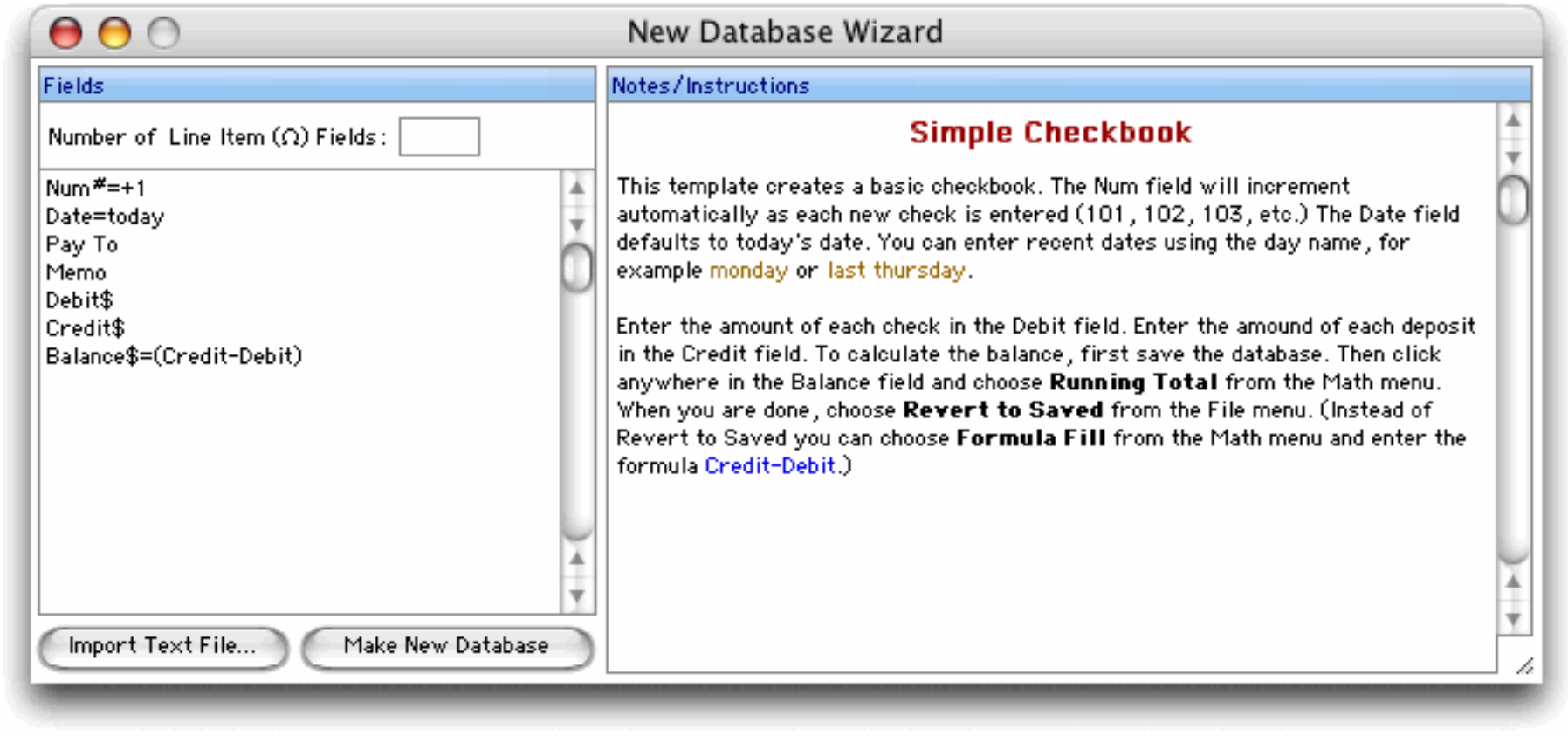
To learn how to create a form like this see "Line Items in a Form" on page 669.

Starting with a New Database Template

To help you get started the **New Database Wizard** includes a number of templates you can access from the **Templates** menu



When you select a database the wizard automatically sets up all of the field definitions, and also displays some notes about this database design on the right.



You can use the template as is or modify the field definitions for your application. When the field definitions are set up the way you want press **Make New Database** to actually create the database.

Creating a Database from a Text File

The **New Database Wizard** usually creates an empty database, but it can also create a database from a text file and automatically import the text (see "Importing a Text File" on page 145). To start this process press the **Import Text File** button at the bottom of the window. The wizard will display a dialog box asking you to select the text file you want to use.

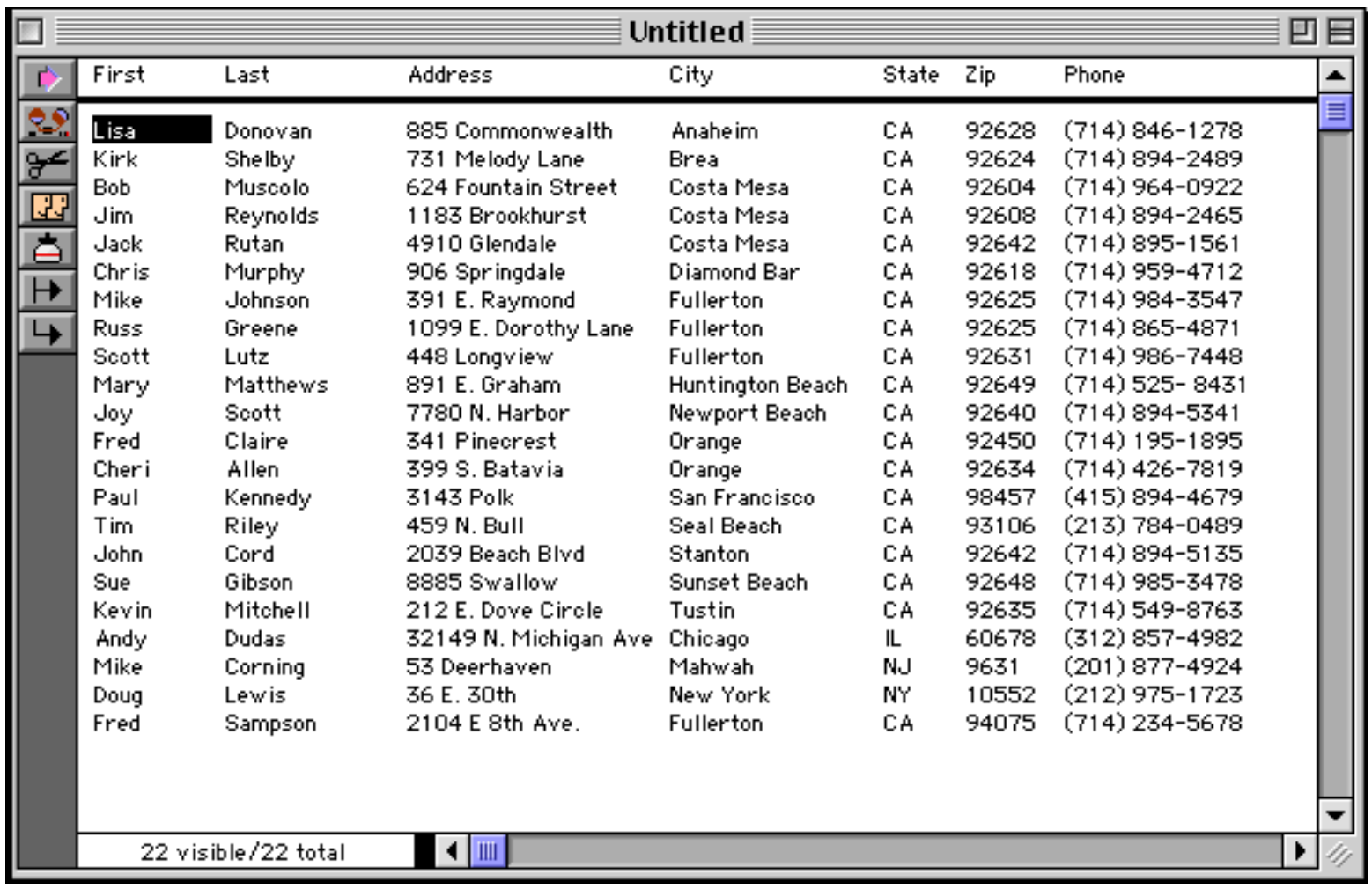
In this case we have selected the file Mailing List Data.csv, a comma separated text file. The wizard displays the data from the first line of the text file.



Using the data as a template, replace the data with the field names you want to use.



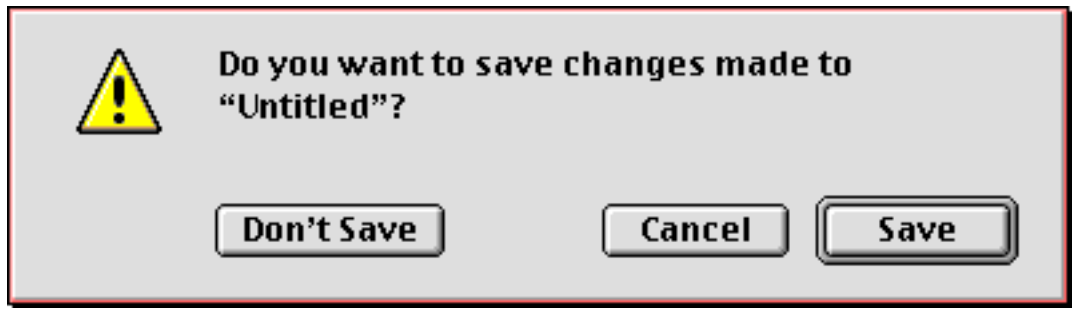
When you press the **Make New Database** button the wizard automatically creates the new database and imports the data.



Be sure to save the new database (see “Saving a Database” on page 131) before you continue.

Closing Panorama

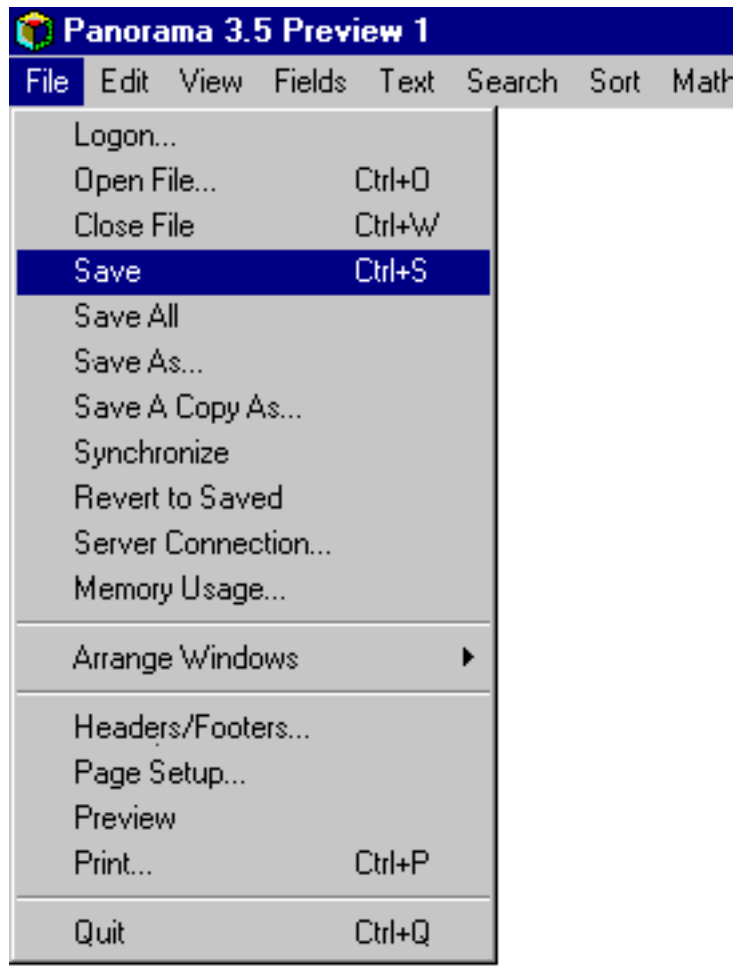
When you are finished with Panorama, use the **Quit** command (File Menu) to return to the desktop. If you have not saved your work, Panorama will display an alert and ask if you would like to save the work now.



Simply press the **Save** button to permanently save the database on the disk and return to the desktop.

Saving a Database

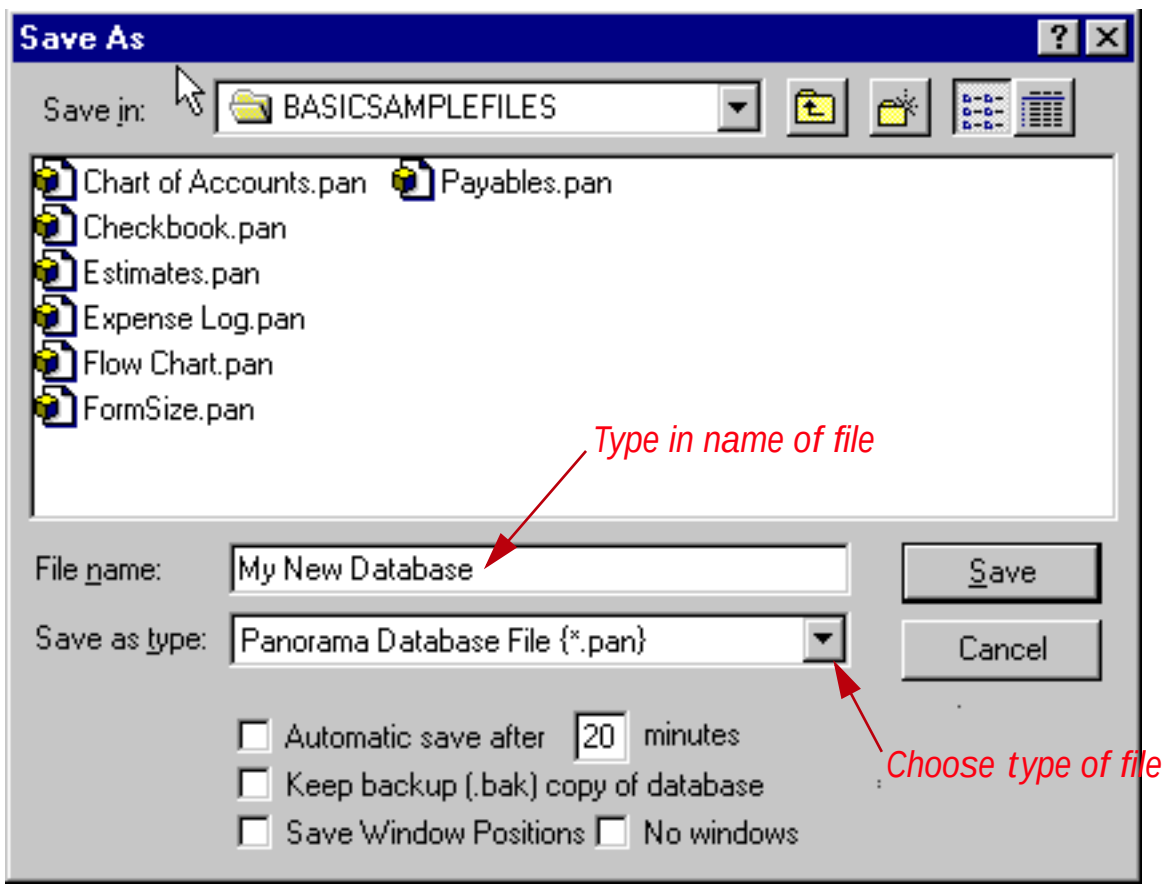
You can save your work permanently on the disk at any time with the **Save**, **Save As**, or **Save a Copy As** commands in the File menu. The **Save** command saves the database permanently on the disk, then allows you to continue with your work. You should save your work often.



The **Save A Copy As** command makes a copy of the database under a new name. It leaves the original copy in memory so you can continue to work on it. This command is like duplicating a sheet of paper and then continuing to work on the original.

The **Save As** command also makes a copy of the database under a new name. The **Save As** command, however, leaves the new copy in memory—not the original. This command is like duplicating a sheet of paper and then working on the copy while setting the original aside.

The **Save As** command allows you to choose the location where you want to save the file, the name of the new file, and several file options.



On Windows computers all Panorama database names end with **.pan**. This is called the **extension**, and it tells the system that this file is a Panorama database. You should not type the extension into the Save As dialog box—Panorama will automatically add the extension for you.

You can save the database as a regular Panorama file, as a text only file (see "Exporting a Text File" on page 165), or create a file set (see "File Sets" on page 138). You can also use the **Save As** command to turn on the Save Window Positions, Auto-Save and Keep Backup Copy options described later in this chapter.

Saving Window Positions

If you check the Save Window Positions option, Panorama will save the positions of each open window that belongs to the file being saved. The next time the file is opened, Panorama will automatically open the same windows in the same positions. Each time you save the file, Panorama will save the new window positions.

It is also possible to lock in window positions so that they will always be open in the same positions, even if you move the windows and save again. To do this, first choose **Save As** and check the Save Window Positions option. Move the windows into the desired positions and then choose **Save** from the File Menu. Now, open the **File Privileges** dialog and change from Author to User or Custom level. (To open this dialog on a Macintosh computer, hold down the Command or Option key and choose **About Panorama** from the Apple menu. To open this dialog on a Windows system, hold down the Control or Alt key and choose **About Panorama** from the Help menu. See "The Privilege Dialog" on page 238 for details on this command.) Close the dialog, and choose Save again. The window positions are now locked in place. To unlock them, switch back to Author mode.

Revert to Saved

The **Revert to Saved** command (File Menu) recopies the original file from the disk into RAM. This will undo all the changes made since the last time the file was saved. By all changes we mean all changes: data entry, sorting, formulas, graphic editing, creating/deleting forms, crosstabs or procedures—every single thing you've done to this database since the last time you saved. Before Panorama actually goes ahead with this command it asks you to verify that you really want to do this.



Be absolutely sure before you press the **Revert** button. Any work you have done since the last time you saved will be gone forever. On the other hand, this command is a great safety net that can let you recover from mistakes. Even if you accidentally delete all the data in your entire database, you can easily get it back with this command. Remember, however, that this safety net only goes back to the last time you have saved. If you accidentally delete all the records and then **Save**, you are out of luck (unless you have made some other backup).

Auto-Save

If you wish, you can ask Panorama to automatically save your file every few minutes. To turn on this feature, choose **Save As**, then check the Automatic Save option and enter how often you want Panorama to save the file. When you are done, press the **Save** button. (If you are using a Macintosh computer and you have saved this file before, Panorama will ask you if you want to replace the existing file. Press **Yes**).

Once you have turned on the Automatic Save option it remains on unless you **Save As** again and turn it off.

Pitfalls of Auto-Save

If you think you might need to use **Revert to Saved**, you should not use auto-save. When you are using auto-save keep in mind that you no longer control when the file will be saved. This lack of control can cause problems if Panorama saves the file when you didn't want it to. In particular, if you have deleted items from your database you won't be able to get them back using **Revert to Saved** once the file has been automatically saved.

Because they can remove large amounts of data from your database, the **Remove Unselected** and **Remove Detail** commands are especially dangerous with auto-save. To help reduce this danger, these commands give you the option to temporarily suspend auto-save. Once you have suspended auto-save it will remain off until you manually save the file (with the **Save** command).

Backup Files

Like most programs, Panorama normally stores only one copy of each database on the disk. If you wish, however, you can tell Panorama to keep a second copy of a database. This extra backup copy can help protect you from mistakes.

The backup copy of a database is a copy of what was saved the second to the last time. Having a backup copy of the database means that if you make a mistake and save when you didn't mean to, you can still get back to the previous version of the database.

The backup copy of the file has the same name as the original with.b added. On Macintosh computers the.b is added at the end of the filename (for example the backup file for Checkbook is Checkbook.b). On Windows computers the .b is added just before the .pan extension (for example the backup file for Contacts.pan is Contacts.b.pan).

Macintosh computers are limited to 32 character file names. Because of this limitation, the original filename cannot exceed 29 characters if the Keep Backup option is used. (If the filename is more than 29 characters, Panorama will ignore the Keep Backup option.)

To tell Panorama to keep a backup copy, use the **Save As** command to save the file and check the Keep Backup Copy of Database option. Once this option is checked, Panorama will keep a backup copy whenever the file is saved.

Opening Backup Files

You can open a backup file by double clicking on it, just like any other Panorama database. On the Macintosh however, you cannot open backup files with the **Open File** command, because the backup files are not displayed in the dialog.

On the Importance of Backing Up

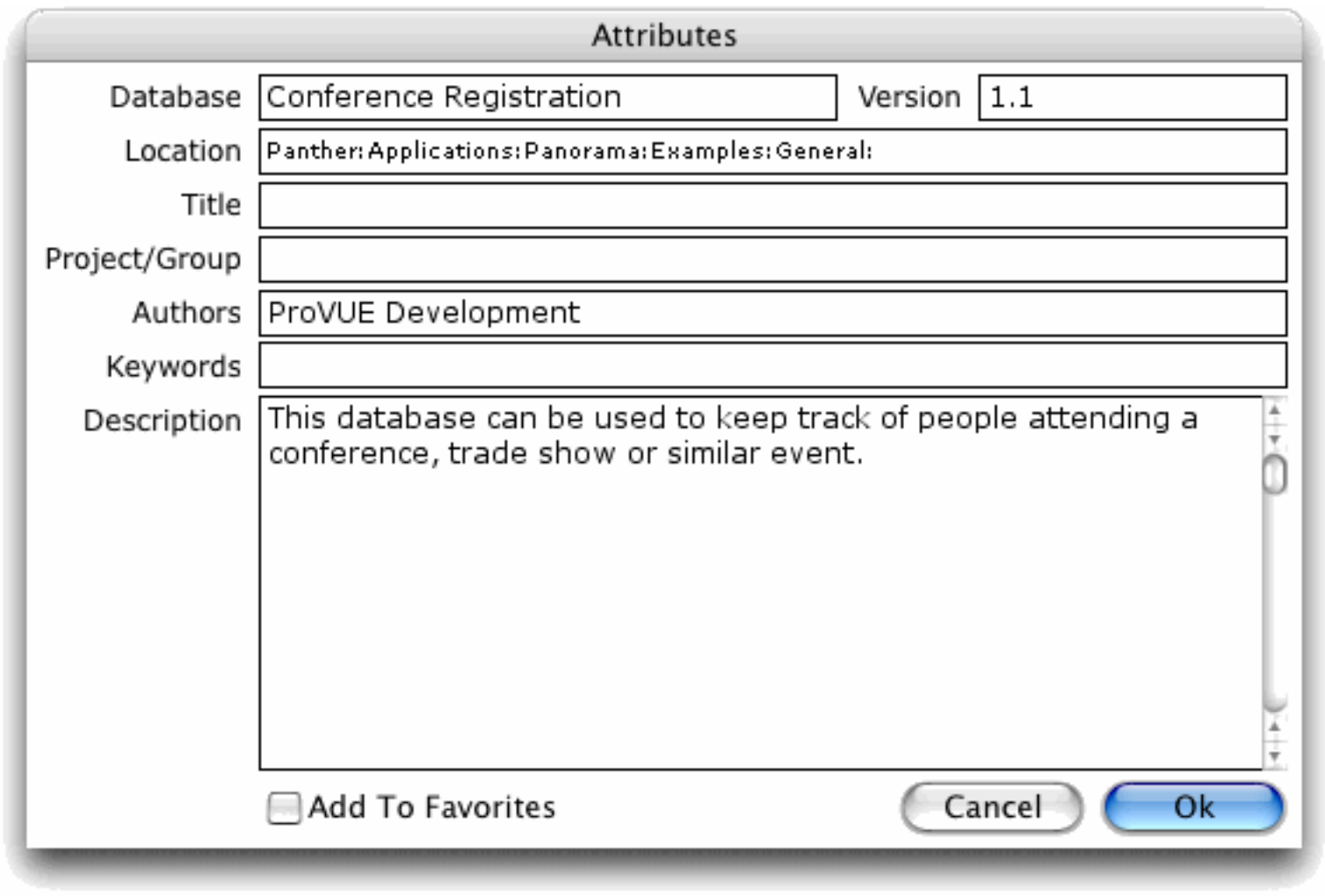
You've heard it before, you'll hear it again—there's no substitute for regular backups. Panorama's automatic backup file option is great for recovering from your mistakes, but it won't do you any good if your hard disk fails, or even gets stolen. Yes, it can happen to you! So be sure to back up your files regularly to a removable media (floppy disk, Zip or Jazz drive, CD-ROM, tape etc.), perhaps even daily if you are doing a lot of work. To protect from fire or theft you should keep your backups at a separate location. Any experienced user will tell you that "Having good backups means never having to say you're sorry!"

Of course some of you will ignore this advice until it happens to you. We get too many sad calls from people who have lost thousands of records due to a hardware failure. Don't take a chance on being one of them.

Viewing and Modifying Database Metadata

Metadata is defined as data that describes other data. The **Database Information** wizard allows you to view and modify descriptive information about any database: the title, author name, description, keywords, etc.

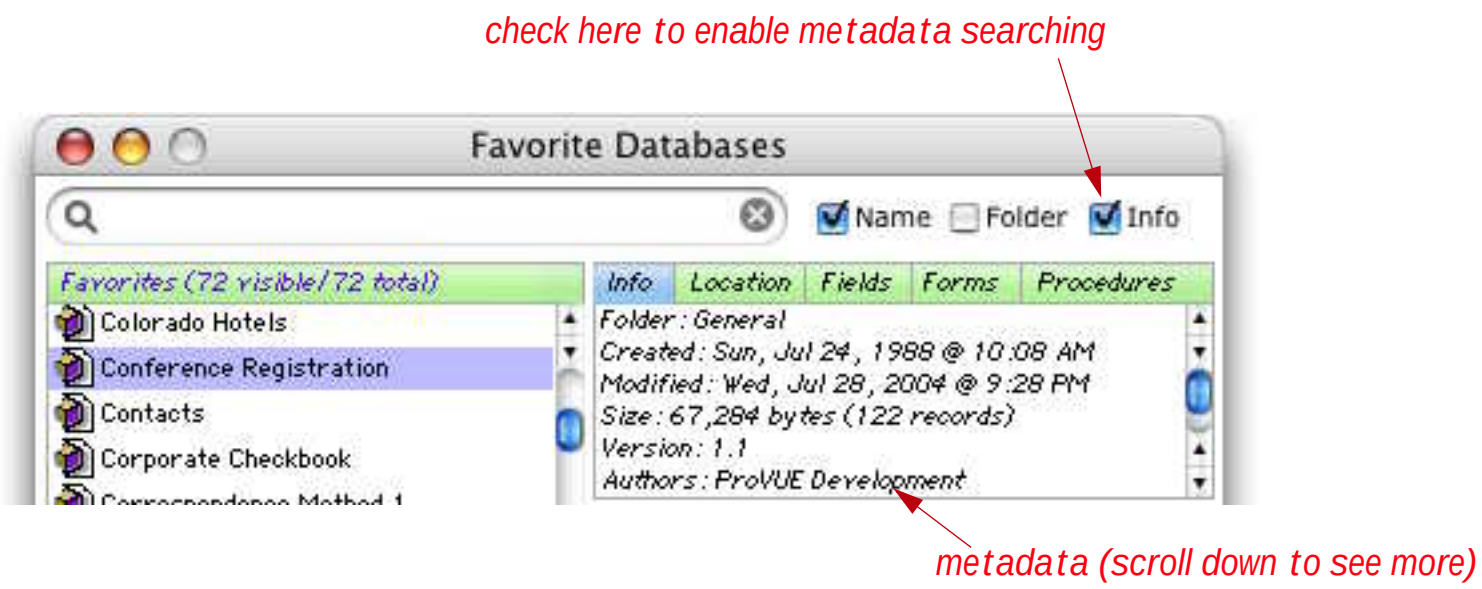
The **Database Information** wizard always works with whatever database is displayed in the topmost window. To view or modify the descriptive information simply choose **Database Information** from the Wizard menu. Unlike most wizards, this one opens a dialog window which displays the metadata information.



The Database and Location fields are always filled in for you, and cannot be changed. You can modify all of the other fields to contain any text you like. When you press the **Ok** button the metadata is stored in the database itself (in this case inside the Conference Registration database). Warning: Just like other database information, the metadata isn't saved permanently until you use the **Save** command.

Using Metadata Information

In addition to viewing the metadata information with the **Database Information** wizard this information can also be viewed and searched with the **Favorite Databases** wizard (see "The Favorite Databases Wizard" on page 109). When a database is added to the **Favorite Databases** wizard the metadata information is automatically imported into the **Favorite Databases** wizard. To enable searching this information check the Info checkbox at the top of the **Favorite Databases** wizard.



A database can also be designed to use metadata within a formula or procedure using the `dbmetatag(` function. For example this could be used to print the database title and/or author names on a report. See the **Programming Reference** wizard for details about this function.

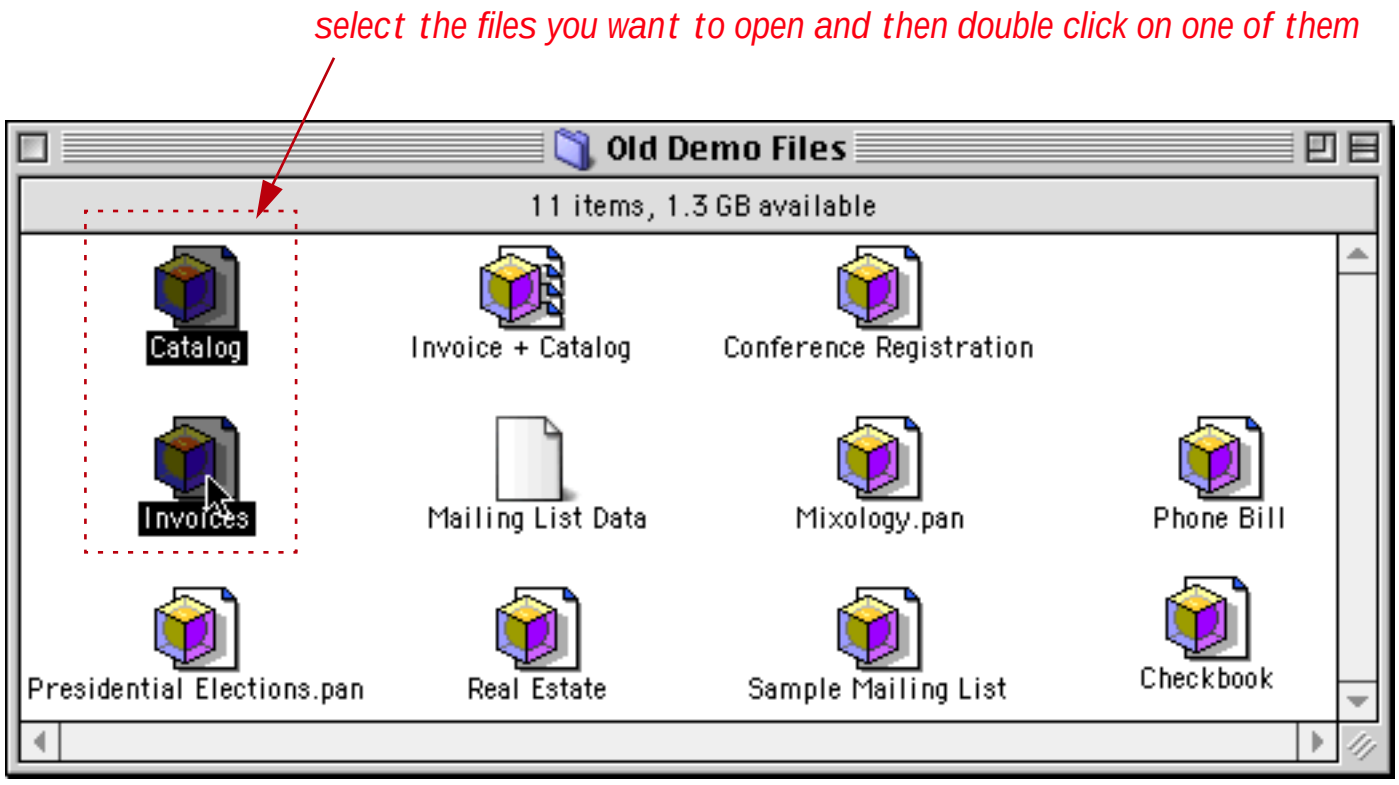
Working with Multiple Databases

Panorama can work with several databases at the same time. The databases can be independent, or they may contain related information.

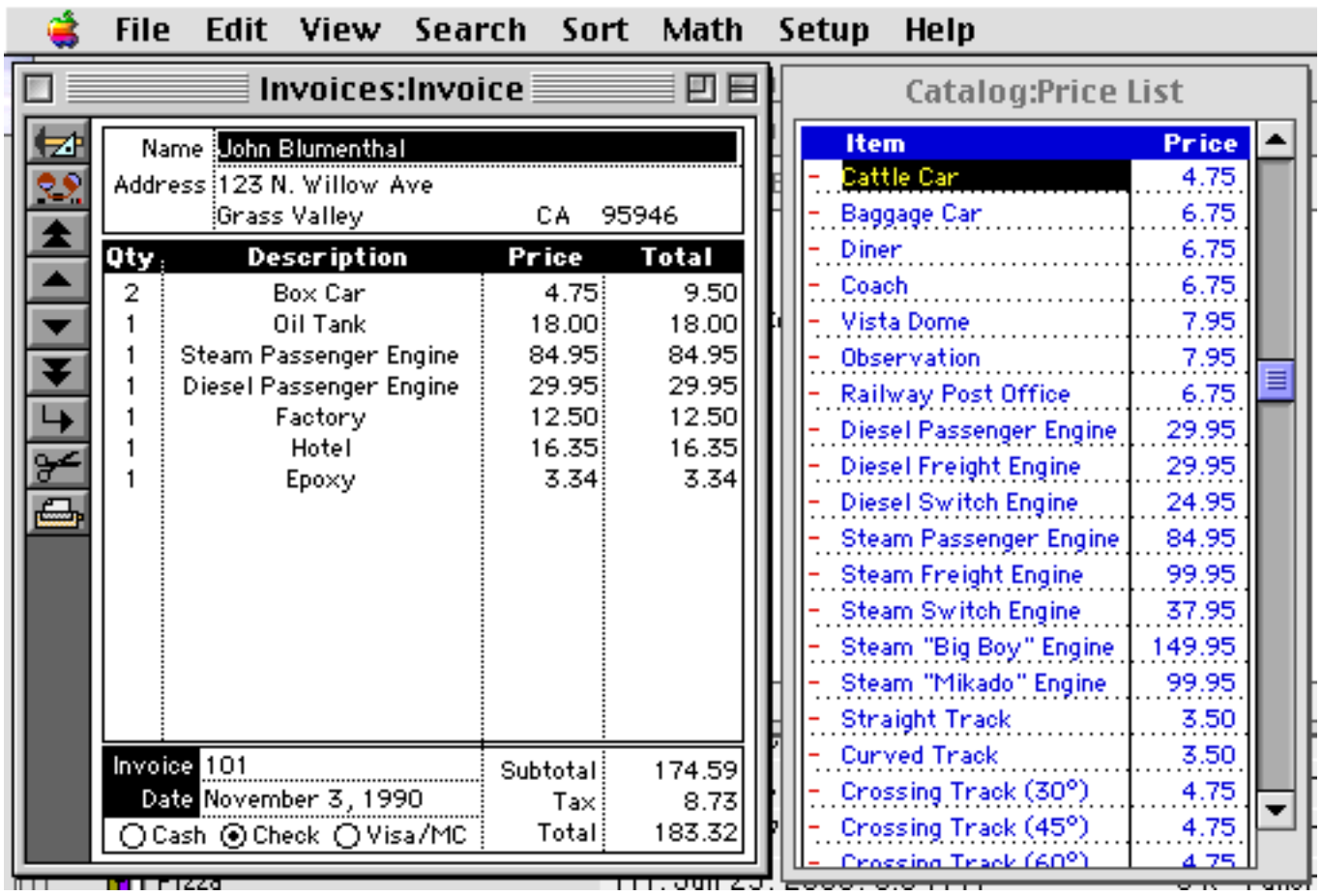
As you use Panorama, you'll probably encounter many tasks that require working with several databases at once. For example, when you are working with an invoice database, you may need to have the customer and price list databases available. As you are working with a checkbook database, you may need to access information in vendor and accounts payable databases. When you are working with a class scheduling database, you may also need the course catalog and student registration databases. Panorama can handle all these tasks.

Opening Multiple Files

You can open multiple files all at once using the desktop (Explorer), or one at a time after Panorama is open. From the desktop, first select the files and then double click on one of them. (To select several files, either hold down the **Shift** key as you click on each file, or drag the mouse around the files.)



Panorama will open all of the databases you have selected.



To open an additional file from inside Panorama, use the **Open File** command in the File Menu. You can continue to open files until you run out of memory. (You are also limited to 32 open windows.)

File Sets

If you plan to use a group of files together, you can create a special document that represents the entire group. This special document is called a File Set. File Sets have their own unique icon.

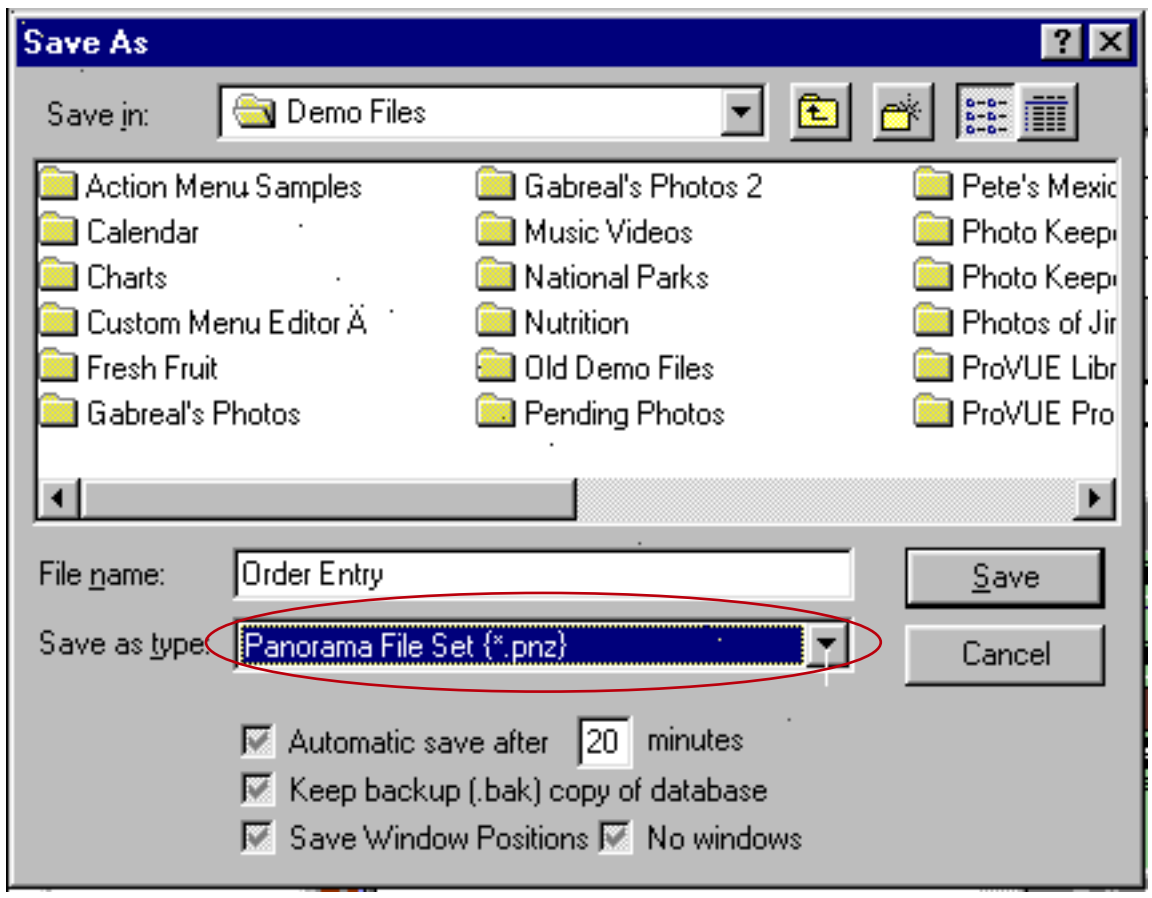


When you double click on a file set icon, Panorama will automatically open all the files included by the set. For example, you could create an Order Entry file set that automatically opens three databases—Invoice, Customer List, and Price List. A file set can also be opened with the **Open File** dialog.

To create a file set, first open the databases you want to include in the set. (Make sure that no other files that you don't want included in the set are open. Every open file will be included in the set. If you are not sure what files are currently open you can use the Memory Usage command to find out — see "Monitoring Memory Usage" on page 186.) Then use the **Save As** command in the File menu. Type in the name of the file set. If you are using a Macintosh computer check the Set radio button.



If you are using a Windows PC computer use the combo box to choose the Panorama File Set {*.pnz} option



Press the **Save** button to create the file set document.

All the databases in a file set must be in the same folder as the file set itself. The files will be opened in the same order that they were originally opened when you created the set. If you want to make sure that the files open in a certain order, you should open the files manually (with the **Open File** command) in that order before saving the file set.

A file set cannot be edited or changed once it is created. To change a file set, you must use **Save As** to create it over again.

Tip: It's important to realize that the file set document does not contain the actual databases themselves—the databases are still in separate files. If you copy the file set to another folder or disk, you also need to copy the actual database files.

Tip: You cannot save the window positions associated with a file set. Instead, you must save the window positions of each individual database within the set. See "Saving Window Positions" on page 132 for more information on saving window positions.

The AutoLoad File Set

Panorama allows you to create a special file set that loads automatically when Panorama is launched. This file set must be called either **AutoLoad** (Macintosh) or **AutoLoad.pnz** (Windows PC) and must be in the same folder as the Panorama application itself. However, this file set only loads automatically if no other database is opened. In other words, if you double click on the Panorama application itself, the AutoLoad file set will open, but if you double click on a Panorama database or file set, the AutoLoad file set will not open. The AutoLoad file set will also open automatically if Panorama is launched automatically (for example from the Startup Items folder, an AppleScript or by configuring a CD to automatically run Panorama from the CD when the disk is entered).

The AutoLoad file set must be in the same folder as Panorama. If that is not possible, you can create an alias (Mac) or shortcut (Windows PC) to the AutoLoad file set and move the alias into the Panorama folder. The alias / shortcut must be named AutoLoad, not AutoLoad alias or any other name.

Saving Multiple Files

The **Save** command only saves one file at a time—the file associated with the top window. To save all the open files, use the **Save All** command. (Note: Only databases that have actually been changed will be saved.)

When you use the **Quit** command, Panorama will check each open database to see if it has been modified. Panorama will ask you if you want to save your changes before shutting down Panorama.

Appending One Database to Another

To append a database to the end of the current database, use the Append To Current Database option in the **Open File** dialog.



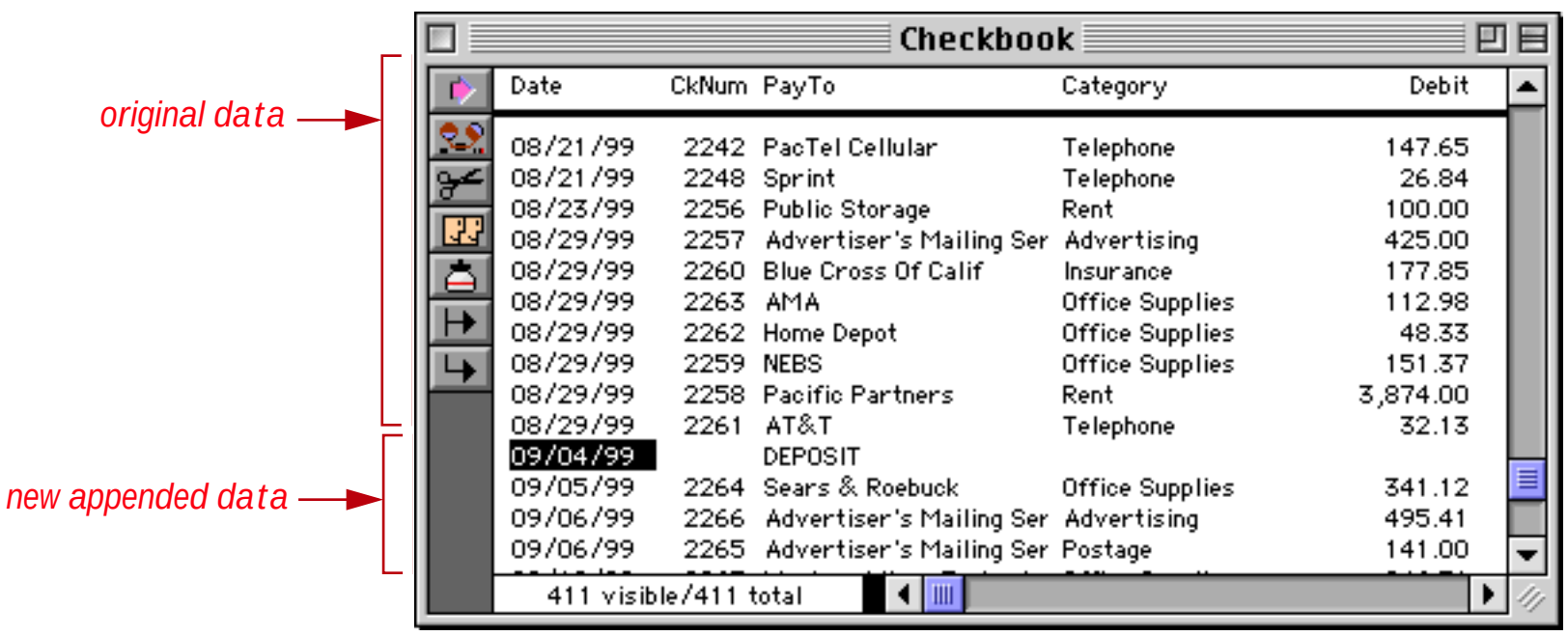
When you press the **Open** button the data in the selected file is appended to the end of the current file. (Only the data is appended, not the forms, procedures, crosstabs, or Flash Art.)

If the Match Fields by Name option is checked Panorama will examine the two databases and look for fields with the same names. Only the data in these fields will be appended. For example if both databases contain fields named Address, City, State and Zip then the information in these fields will be appended. However if one database has a field named Zip and the other has a field called PostalCode, the data in these fields will not be appended. The field names must match exactly — Company and COMPANY will not match.

If the Match Fields by Name option is not checked Panorama will append fields according to their order. In other words, the first field of the second database will be appended to the first field of the current database, the second field to the second field, etc. Usually appending two databases this way makes sense only if both databases have the same fields in the same order. If they don't, you can open the second database, re-arrange the fields and then append (see below).

If the Match Fields by Name option is not checked and the database being appended has more fields than the current database, the extra fields will be ignored. If the data types in the two files are incompatible, some data may be lost. For example, data will be lost if you try to append text into a numeric field. Panorama will alert you if this happens, but it cannot tell you exactly what data has been lost.

After the append is finished, Panorama positions the database at the first new record. All the records above this position are part of the original file. All the records at and below this position are part of the new appended data.



Appending an Open Database

Panorama can append a database that is already open. Panorama will append the copy of the database that is open in memory, not the original on disk. This is handy if you want to modify a file before you append it.

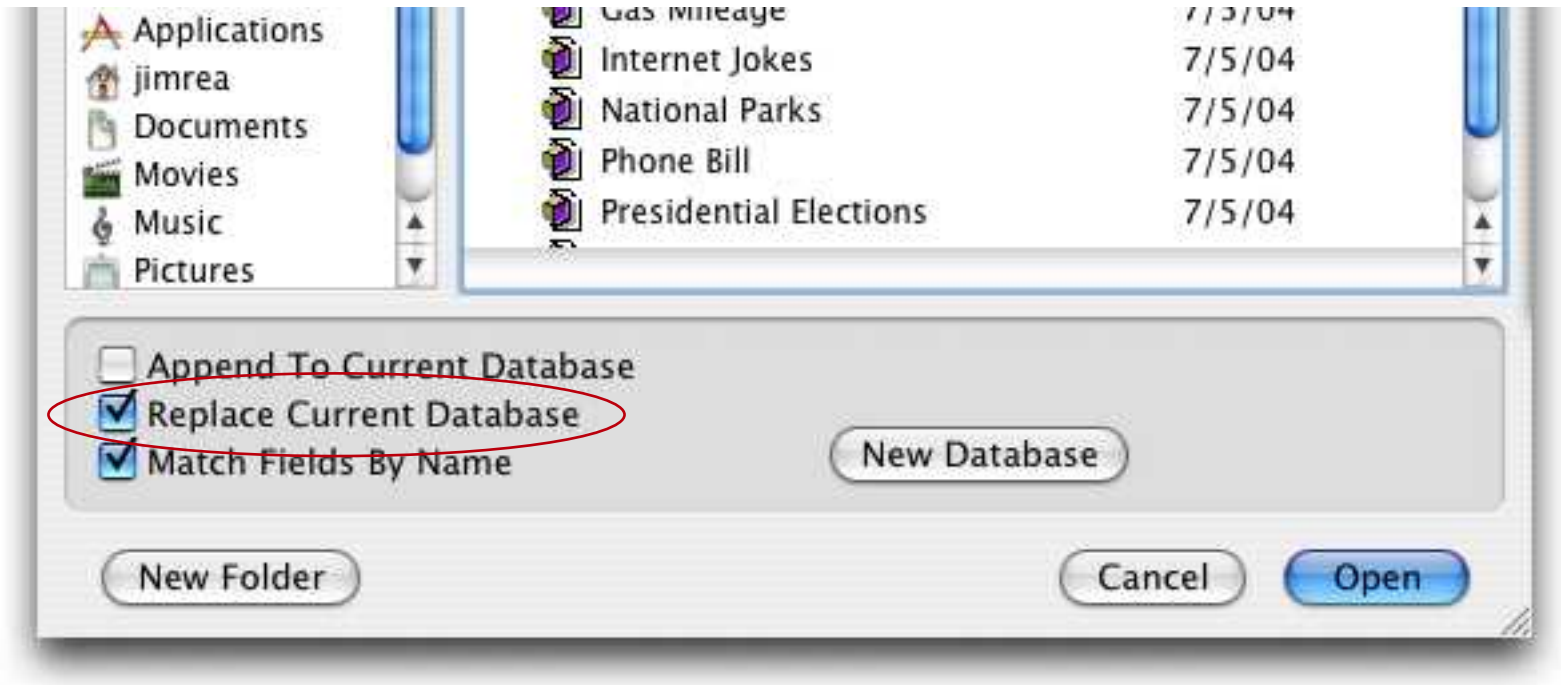
Panorama can also append a database to itself. In this case Panorama will append the original copy of the database on disk. For example if you open a database with 1,000 records, delete 500 records without saving, and then append the database to itself, you will wind up with 1,500 records.

Appending Imported Data

By checking the OverVUE or Import radio buttons (PC) or selecting the **Text Files (Import)** option from **Enable** pop-up menu (Macintosh) in the **Open File** dialog, you can import data and append it to the current database in a single step. Either tab or comma delimited TEXT files can be appended. The data being imported should contain the same fields in the same order as the current database. If the fields do not match, some data may be lost. For a complete discussion of importing data into Panorama, see "Importing a Text File" on page 145.

Replacing Obsolete Data

Another **Open File** dialog option is Replace Current Database.



This option allows you to completely replace the data in the current database with the data in another database. This is useful if the current database contains obsolete data, but has forms or procedure you want to use.

Depending on whether or not the Match Fields by Name option is checked the new data should either have the same field names as the old database or it should have the same fields in the same order as the current database. If the fields do not match, some data may be lost. The Replace Current Database option works exactly like the Append option, except that Panorama removes all the data in the current database before appending the data.

You can also replace the current data with imported data—just check the Replace Current Database option and also check OverVUE or Import (PC) or select **Text File (Import)** from the Enable pop-up menu. The data being imported should contain the same fields in the same order as the current database. If the fields do not match, some data may be lost.

Importing and Exporting Data

Panorama allows you to freely exchange information between it and other applications. The common terms for these exchanges are **importing** and **exporting**.

Importing means to transfer information from another program or computer into a Panorama file. The data can then be manipulated using Panorama's menu commands and tools. Panorama's import capabilities allow you to take advantage of databases that have already been keyed in or databases on other computers (for instance, on minicomputers or electronic bulletin boards). See "Importing a Text File" on page 145.

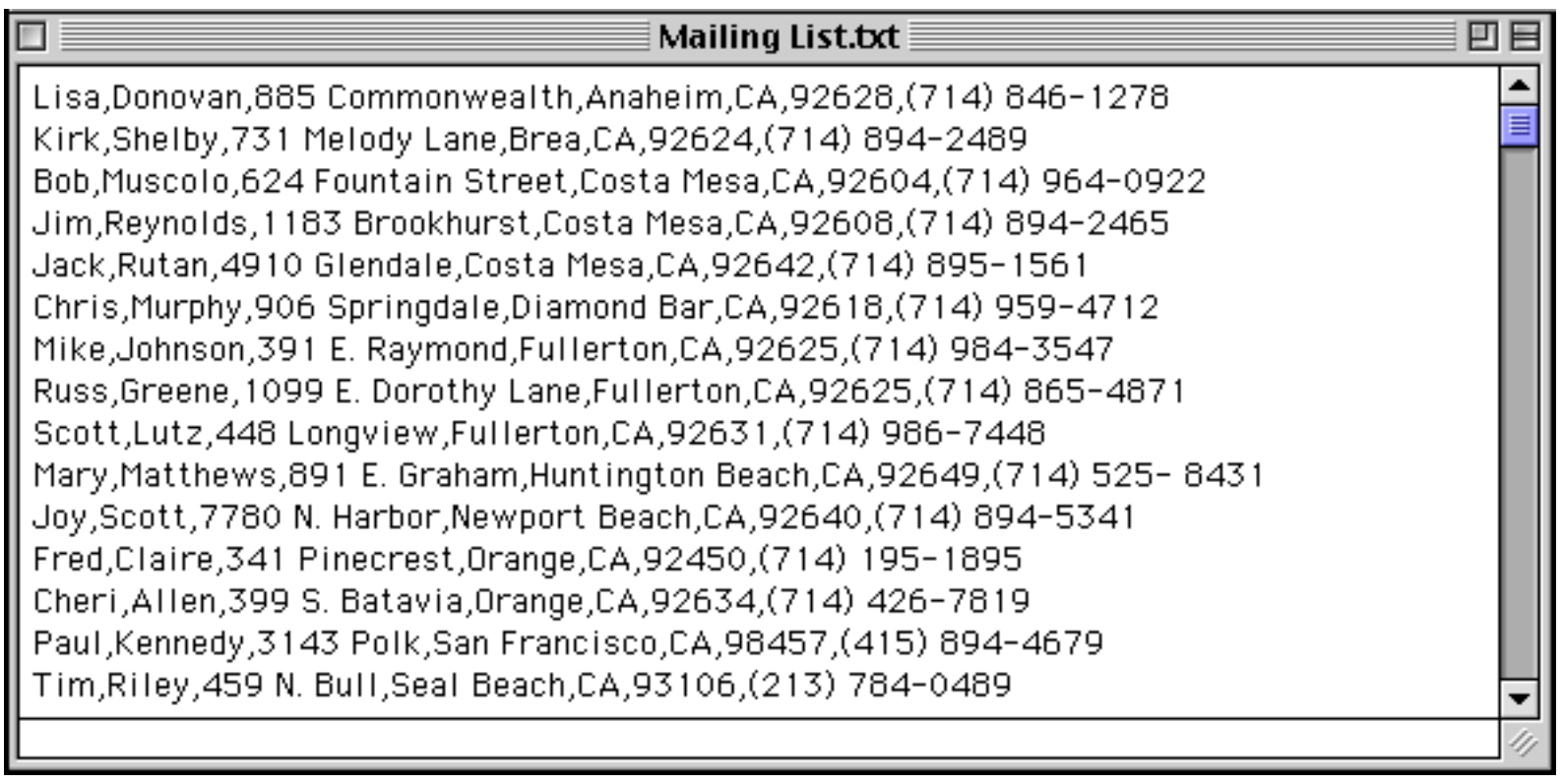
Exporting is the exact opposite of importing. To export data from Panorama means to take data from a Panorama file and make it accessible to another "foreign" program. For example, Panorama data can be exported to Excel so that it can be included in a spreadsheet. See "Exporting a Text File" on page 165.

In addition to the manual techniques described in this chapter it is also possible to set up a procedure to automatically import or export data. To learn more about this see "Importing Text Files" on page 1537 and "Exporting Text Files" on page 1541.

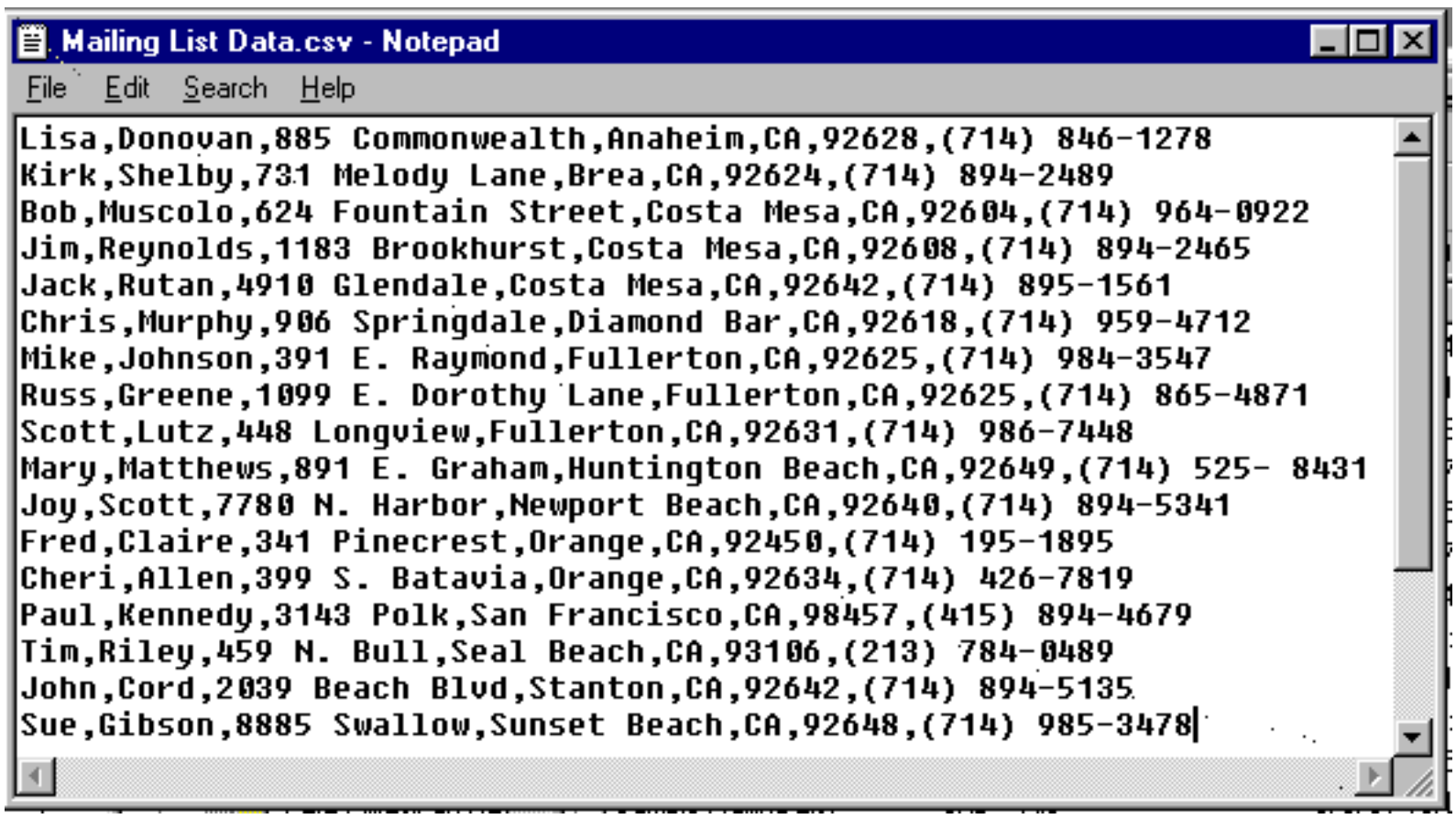
Working with Text Files

Panorama cannot directly access information in database or spreadsheet files created by other programs. Exchanging data between Panorama and another program requires an intermediate **text file**. A text file is very basic because it contains just the data—no forms, procedures, graphics, or anything else. Because text files are so simple, they provide a common interchange format for different programs. Virtually all database, spreadsheet, and word processing programs can read and write text files. This makes transferring data between Panorama and another program a two step process. Let's take Excel as an example. To transfer data from Panorama to Excel you first must export the data from Panorama as a text file. Then you go into Excel and import the text file. To transfer data from Excel to Panorama you start by exporting the data from Excel as a text file. Once the text file has been created you can go into Panorama and import the data from the text file.

In addition to importing and exporting text files you can also edit them directly. On the Macintosh the **SimpleText** application is provided free with every computer. Here's what a typical text file looks like in this application.



On PC systems text files can be edited with the **Notepad** or **WordPad** applications.



On PC systems text files often have a three letter filename extension of **.txt**, for example My Data.txt. However, text files may use other extensions as well, such as the **.csv** (short for **c**omma **s**eparated **v**alues) file shown above. Panorama can work with text files with any extension.

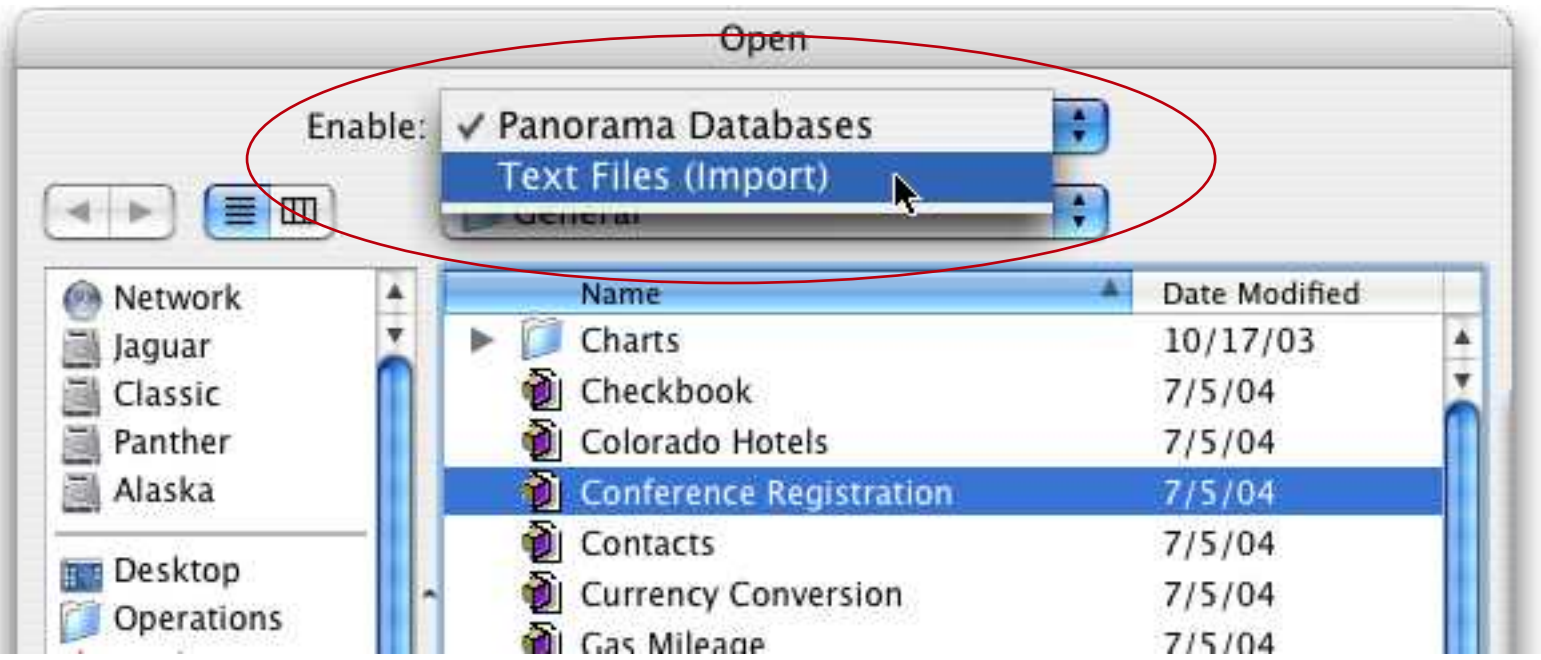
On Macintosh systems no extension is required. However we recommend adding **.txt** to the end of the filename anyway. This makes it easier to remember what kind of data is in the file and also improves compatibility in case the file is ever transferred to a PC system.

Importing a Text File

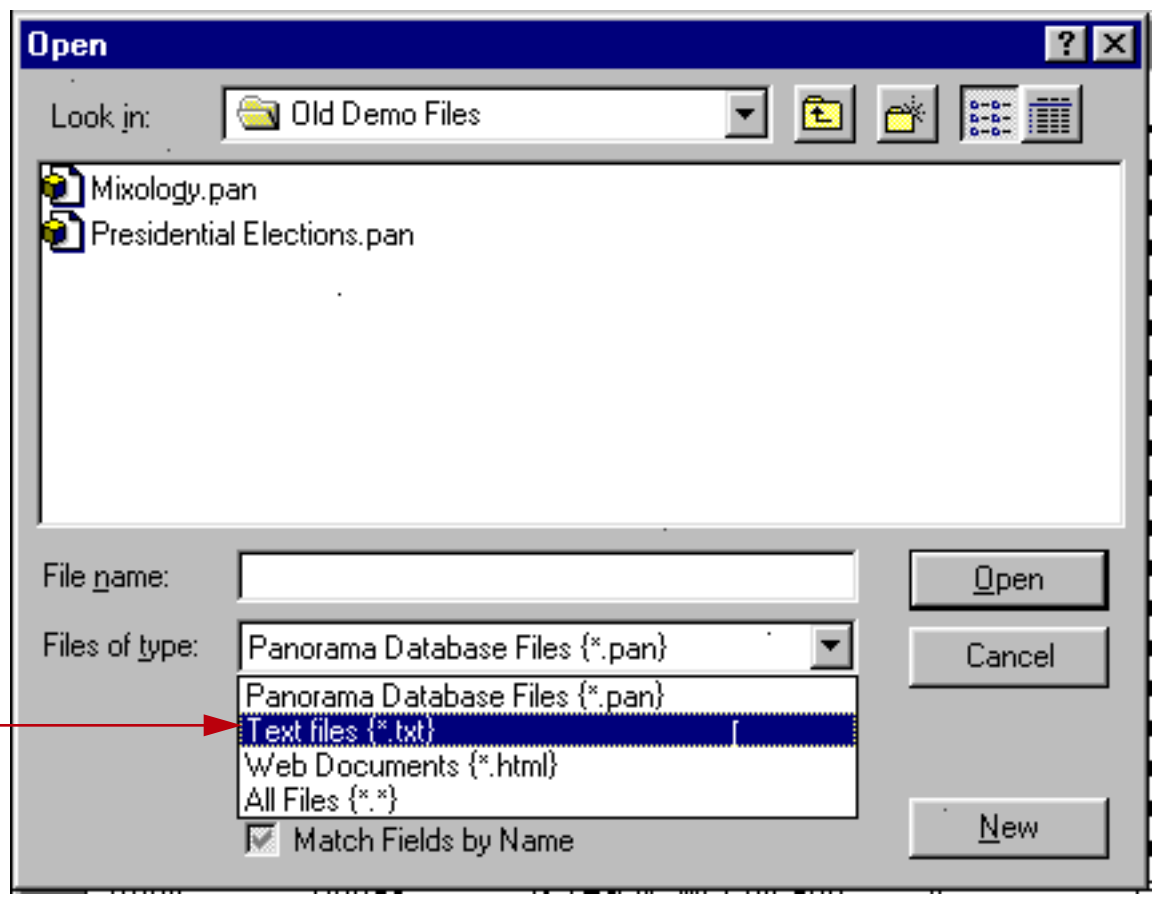
The first step in importing a text file is to prepare the text file. You can create it manually using a text editor like **SimpleText** or **Notepad**, but usually the text file is created by exporting from another database or spreadsheet program.

Macintosh	PC
 Mailing List.txt	 Mailing List.txt  Mailing List Data.csv

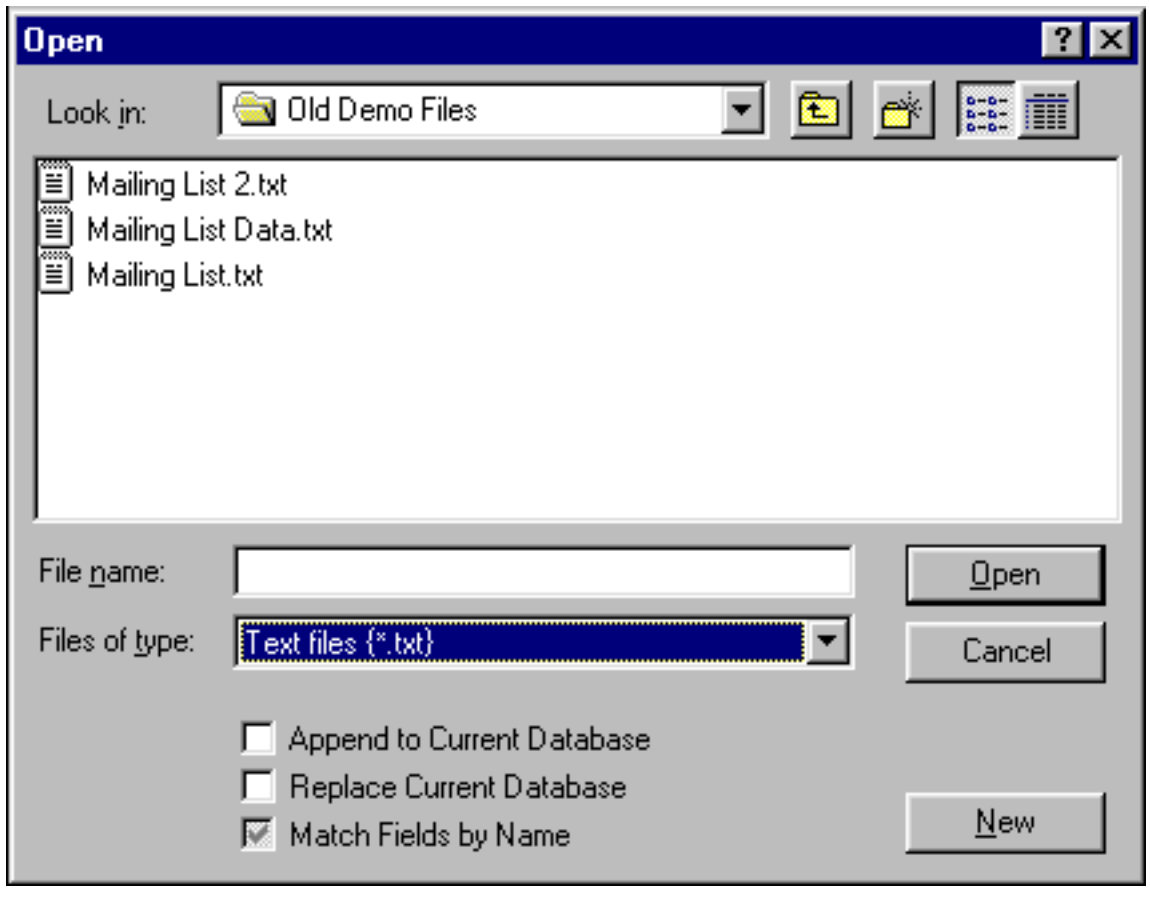
Once the text file is prepared go into Panorama and choose **Open File** from the File menu. This is the same dialog that is used for opening Panorama databases. On the Macintosh, choose **Text Files (Import)** from the Enable pop-up menu to see a list of text files that can be imported.



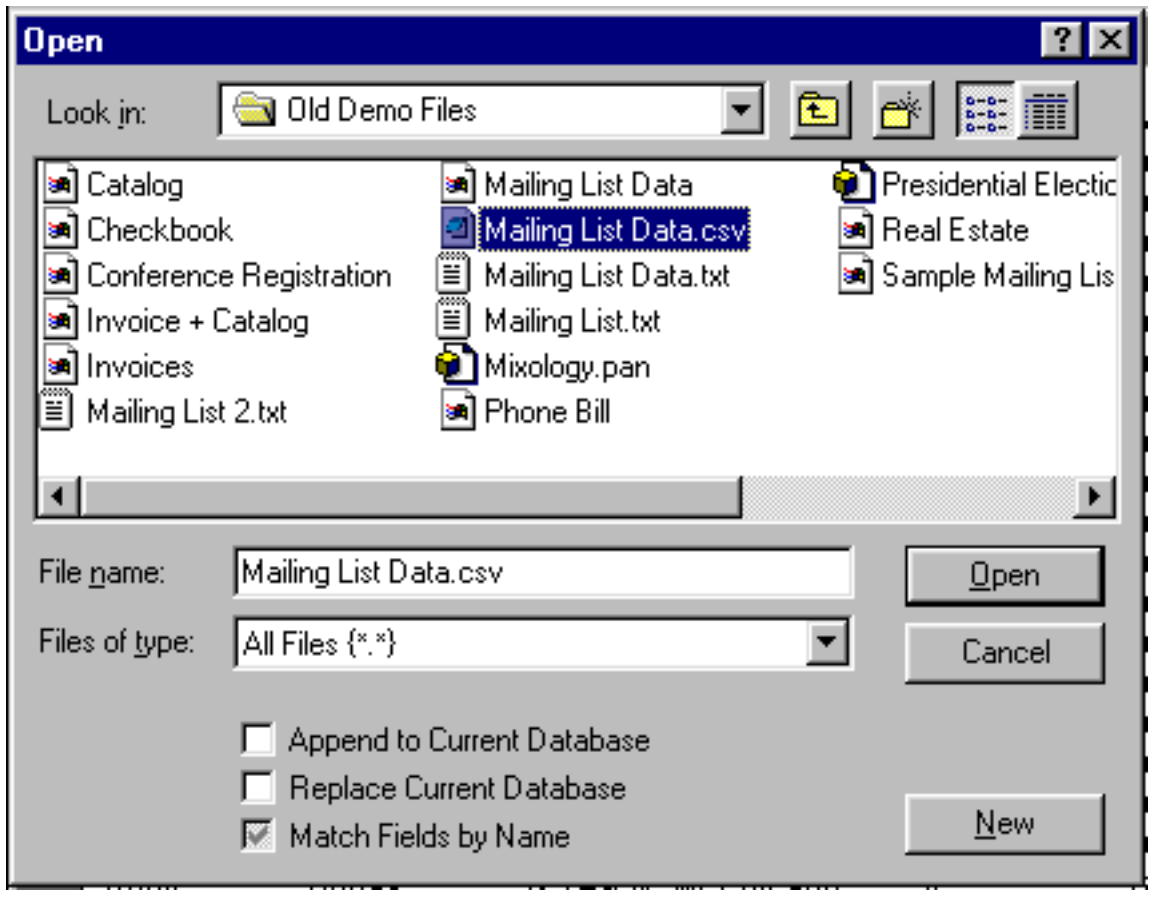
If you are using a Windows PC computer then use the combo box to choose Text files {*.txt}.



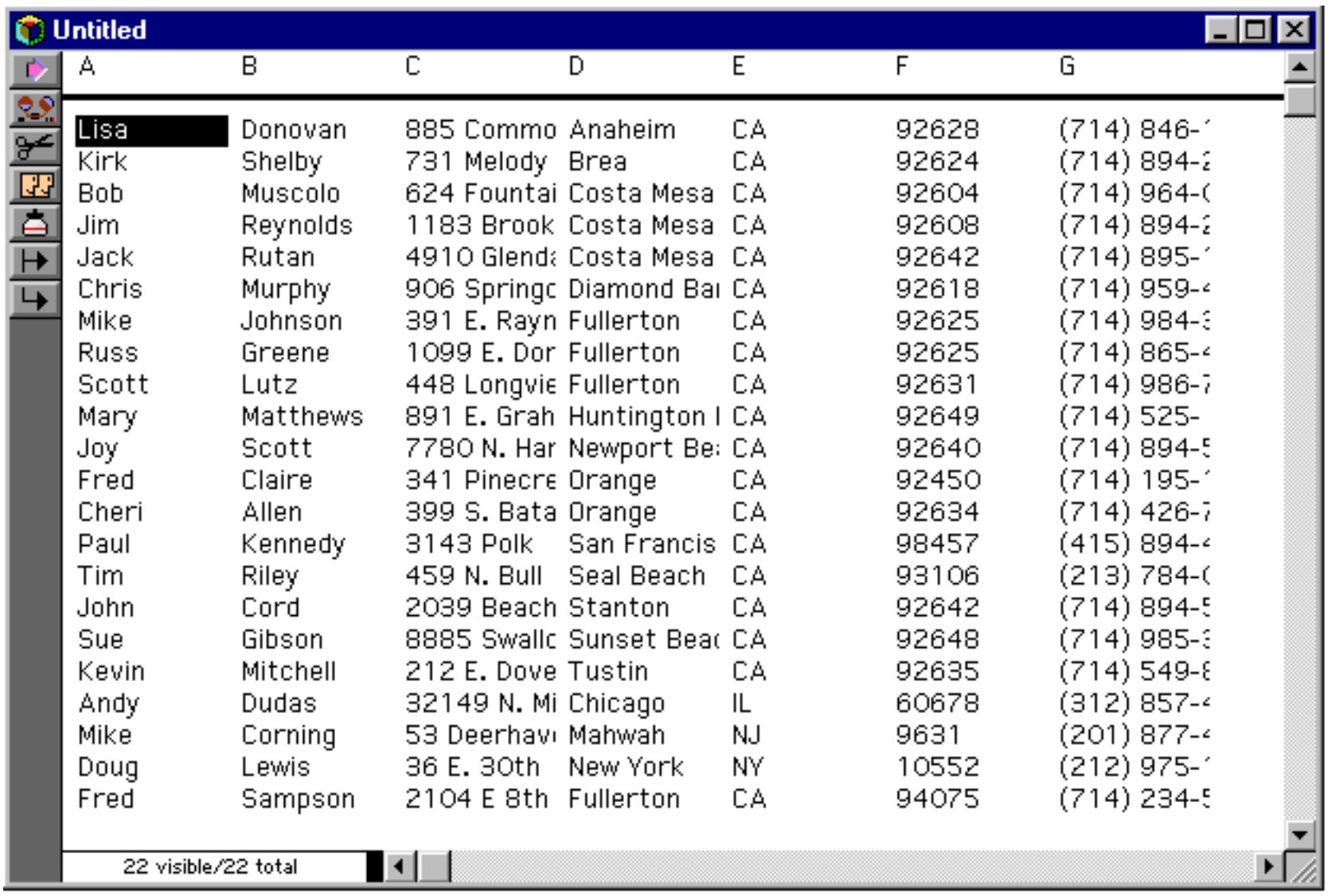
When you choose this option the dialog will list all of the **.txt** files available to be imported.



If you want to import a text file that does not use the .txt extension then use the combo box to choose All files {*.*}. This causes all files to be listed. In this illustration the All files option is being used to allow a **.csv** file to be selected for import.



Once the text file you want to import is visible simply click on it and press the **Open** button. Panorama will create a new database and fill it with the data from the text file. Panorama automatically examines the text file to decide whether the data is tab or comma separated and then creates the number of columns required to hold the data.

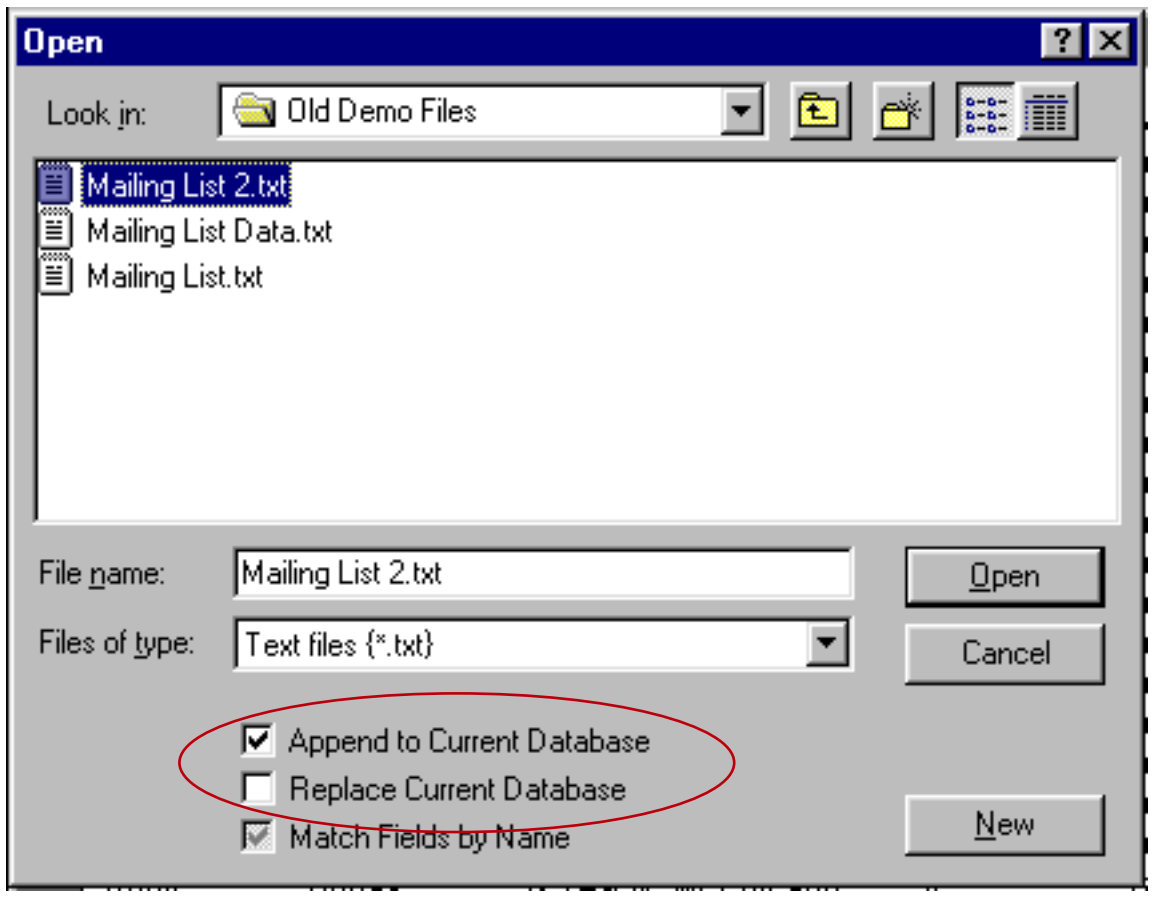


The new database is initially assigned the name Untitled (or if that is taken, Untitled 2, Untitled 3, etc.). The first time you **Save** the database Panorama will ask you to assign the actual name to the file.

The columns in the new database are initially assigned the names A, B, C, etc. To assign real names to these columns you can use the **Field Properties** dialog (see “Field Properties” on page 250) or the Design Sheet (see “The Design Sheet” on page 252)

Importing into an Existing Database

When you import a text file Panorama normally creates a brand new database, as described in the previous section. However, it is also possible to import data into an existing database. To do this you must check either the Append to Current Database or the Replace Current Database option in the **Open File** dialog.



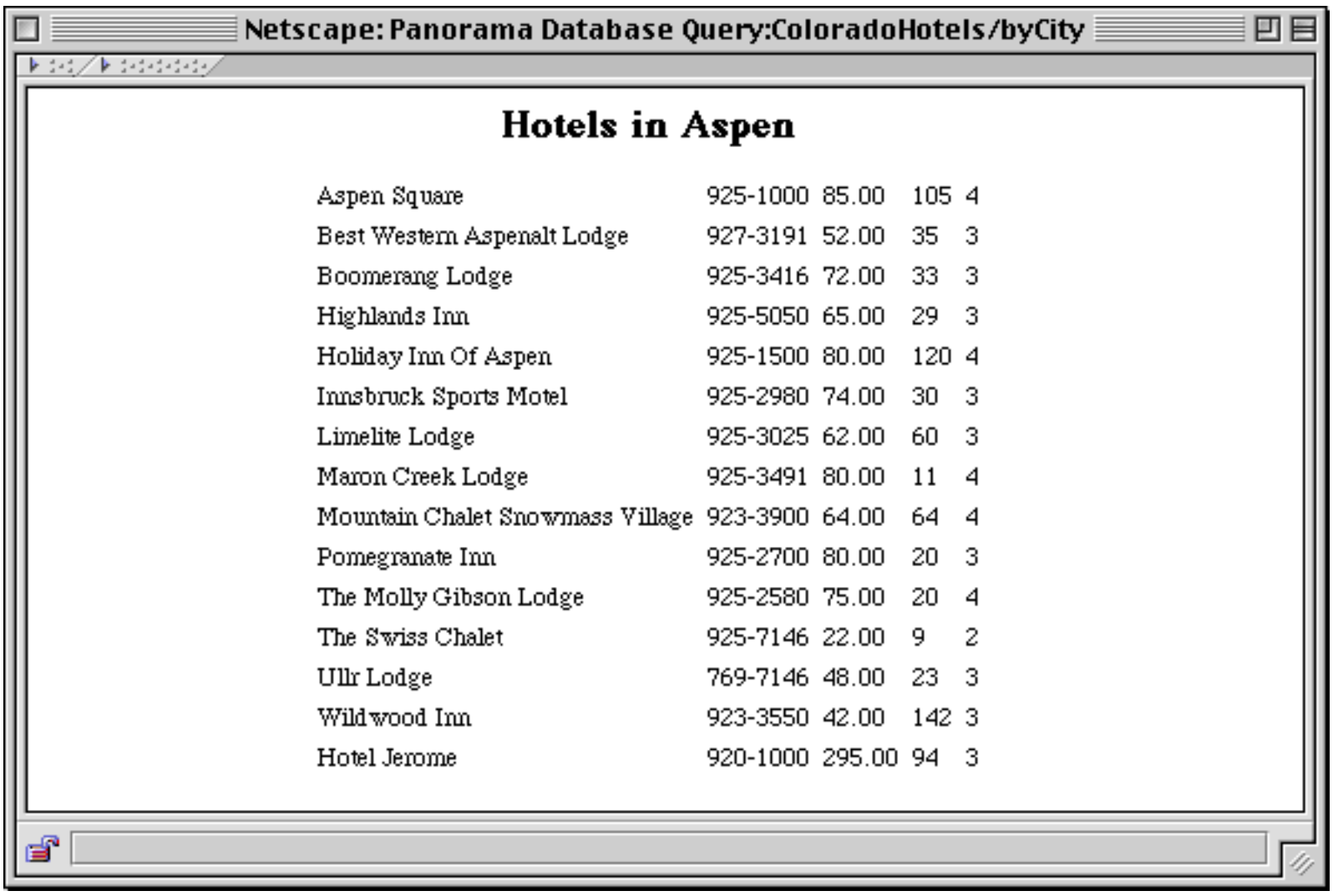
The data being imported should have the same fields in the same order as the current database. If the fields are not in the same order you should use the **Text Import Wizard** (see "Using the Text Import Wizard" on page 157) to import the text. Otherwise some data may be lost.

If the Append to Current Database option is checked, the imported data will be appended to the end of the current database. Use this option to add new data to an existing database.

If the Replace Current Database option is checked, the imported data will completely replace the current data. The old data will be erased, then replaced with the new imported data. Use this option to update a database with the latest information, while leaving all the forms, reports, crosstabs, and procedures intact.

Importing HTML Tables

Panorama can automatically import tables from a text file that contains HTML with the `<table>` tag. As an example consider the HTML page shown below.



To import the data on this page you must first save the page as a text file. Consult the documentation for your browser to learn how to do this. A sample HTML file named Aspen Hotels.html is supplied with your copy of Panorama — locate this file now if you want to follow along.



If you examine this file with a text editor you should see the <table> tag followed by the data and finished up with the </table> tag.

Aspen Hotels.html

```
<html>
<head>
<title>Panorama Database Query :ColoradoHotels/byCity</title>
</head>
<body bgcolor="FFFFFF" text="000000" text="0000FF" text="336666">
<center>
<h2>Hotels in Aspen</h2>
<table>
<tr><td>Aspen Square</td><td>925-1000</td><td>85.00</td><td>105</td><td>4</td></tr>
<tr><td>Best Western Aspenalt Lodge</td><td>927-3191 </td><td>52.00</td><td>35</td><td>3</td></tr>
<tr><td>Boomerang Lodge</td><td>925-3416</td><td>72.00</td><td>33</td><td>3</td></tr>
<tr><td>Highlands Inn</td><td>925-5050</td><td>65.00</td><td>29</td><td>3</td></tr>
<tr><td>Holiday Inn Of Aspen</td><td>925-1500</td><td>80.00</td><td>120</td><td>4</td></tr>
<tr><td>Innsbruck Sports Motel</td><td>925-2980</td><td>74.00</td><td>30</td><td>3</td></tr>
<tr><td>Limelite Lodge</td><td>925-3025</td><td>62.00</td><td>60</td><td>3</td></tr>
<tr><td>Maron Creek Lodge</td><td>925-3491 </td><td>80.00</td><td>11</td><td>4</td></tr>
<tr><td>Mountain Chalet Snowmass Village</td><td>923-3900</td><td>64.00</td><td>64</td><td>4</td></tr>
<tr><td>Pomegranate Inn</td><td>925-2700</td><td>80.00</td><td>20</td><td>3</td></tr>
<tr><td>The Molly Gibson Lodge</td><td>925-2580</td><td>75.00</td><td>20</td><td>4</td></tr>
<tr><td>The Swiss Chalet</td><td>925-7146</td><td>22.00</td><td>9</td><td>2</td></tr>
<tr><td>Ullr Lodge</td><td>769-7146</td><td>48.00</td><td>23</td><td>3</td></tr>
<tr><td>Wildwood Inn</td><td>923-3550</td><td>42.00</td><td>142</td><td>3</td></tr>
<tr><td>Hotel Jerome</td><td>920-1000</td><td>295.00</td><td>94</td><td>3</td></tr>
</table>
</center>

</body>
</html>
```

You can import this file just like any other text file. Panorama will scan the file and notice the <table> tag. When it finds this tag it switches gears and goes into HTML table import mode. It automatically analyzes the tags and extracts the data in the table. Voila!

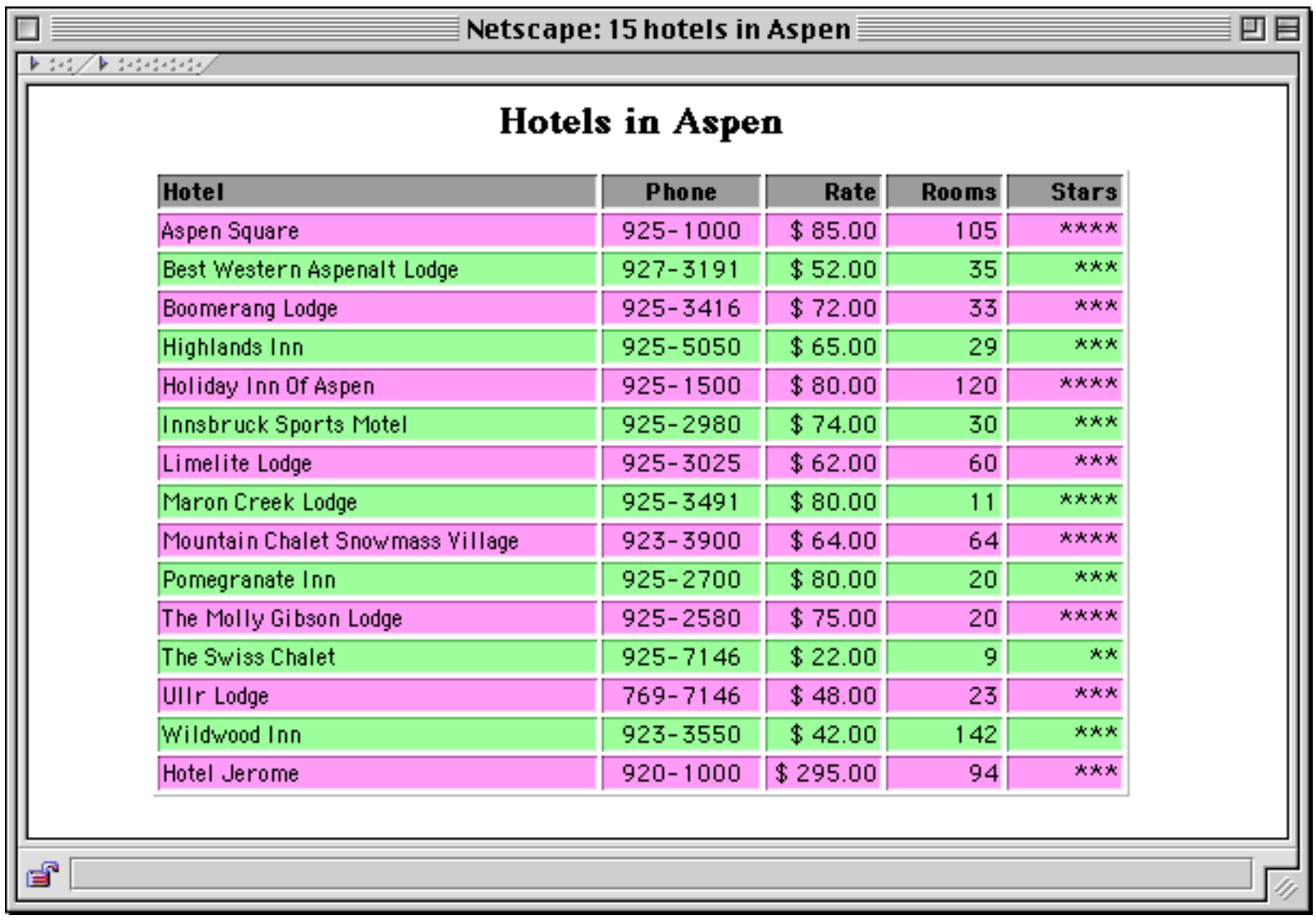
Untitled

A	B	C	D	E
Aspen Square	925-1000	85.00	105	4
Best Western A	927-3191	52.00	35	3
Boomerang Lod	925-3416	72.00	33	3
Highlands Inn	925-5050	65.00	29	3
Holiday Inn Of	925-1500	80.00	120	4
Innsbruck Spor	925-2980	74.00	30	3
Limelite Lodge	925-3025	62.00	60	3
Maron Creek Lc	925-3491	80.00	11	4
Mountain Chale	923-3900	64.00	64	4
Pomegranate Ir	925-2700	80.00	20	3
The Molly Gibs	925-2580	75.00	20	4
The Swiss Cha	925-7146	22.00	9	2
Ullr Lodge	769-7146	48.00	23	3
Wildwood Inn	923-3550	42.00	142	3
Hotel Jerome	920-1000	295.00	94	3

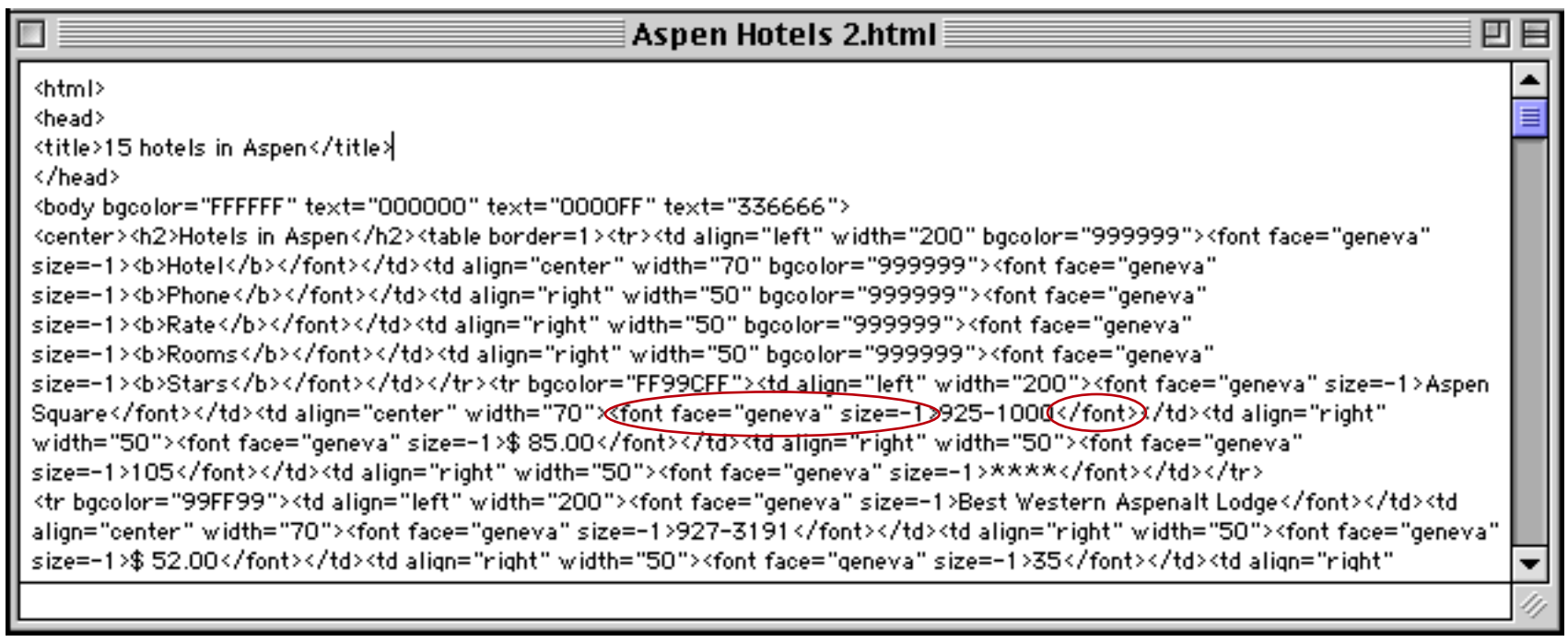
15 visible/15 total

If the **.html** file contains more than one `<table>` tag Panorama will only import the first one. You may need to manually edit the file before you import to make sure that the data you want to import is the first table in the file.

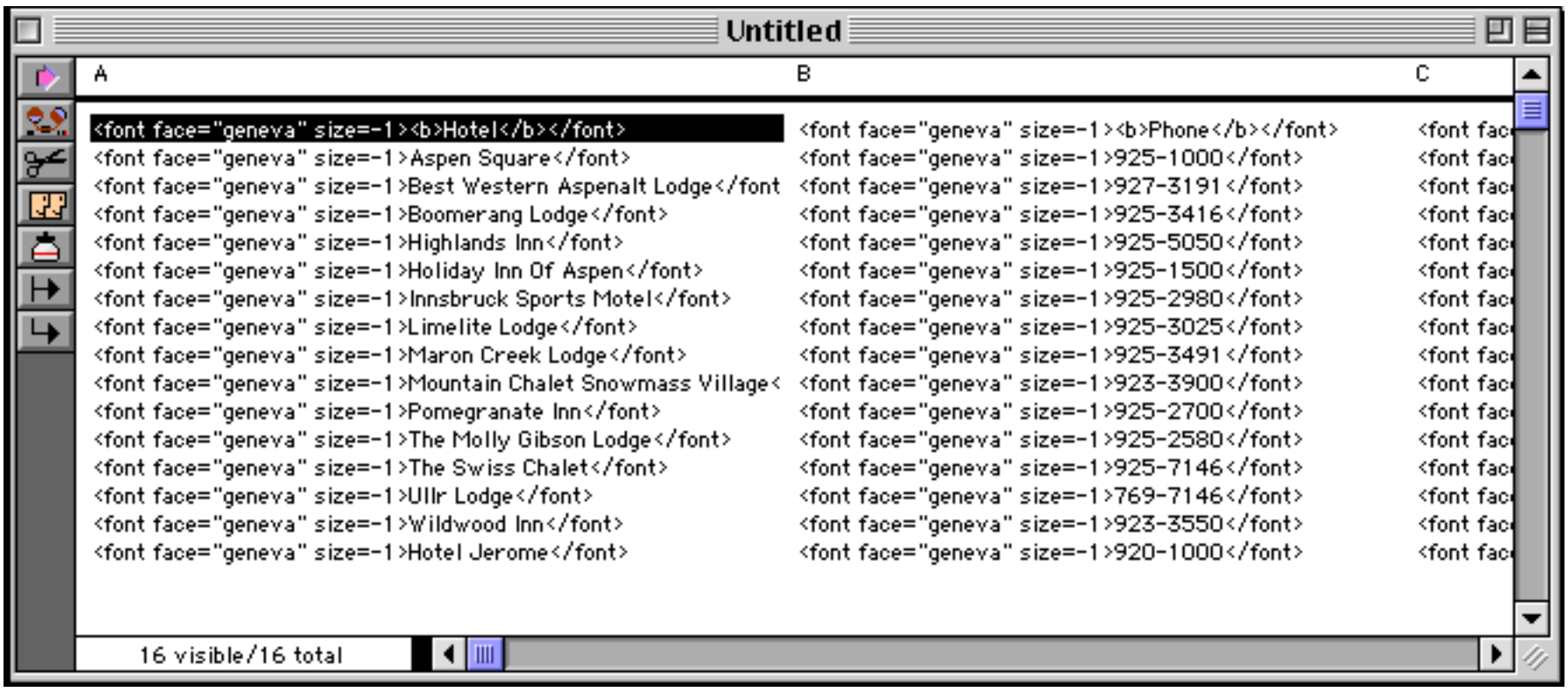
In some cases you may need to do further work to clean up the data after it is imported. Consider the table shown here.



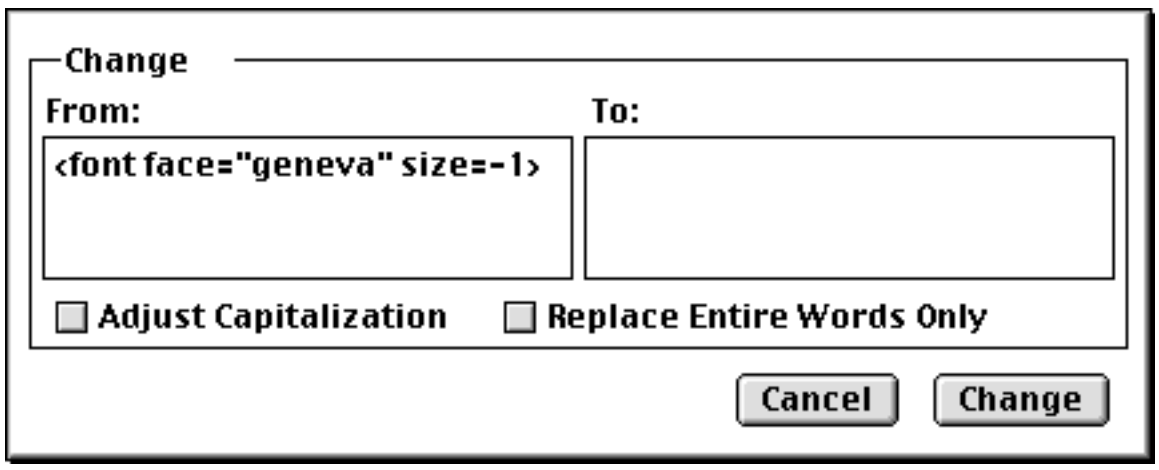
In this case the html is much more complex, and each data cell contains tags for setting the font.



When the data is imported, these tags are also imported.



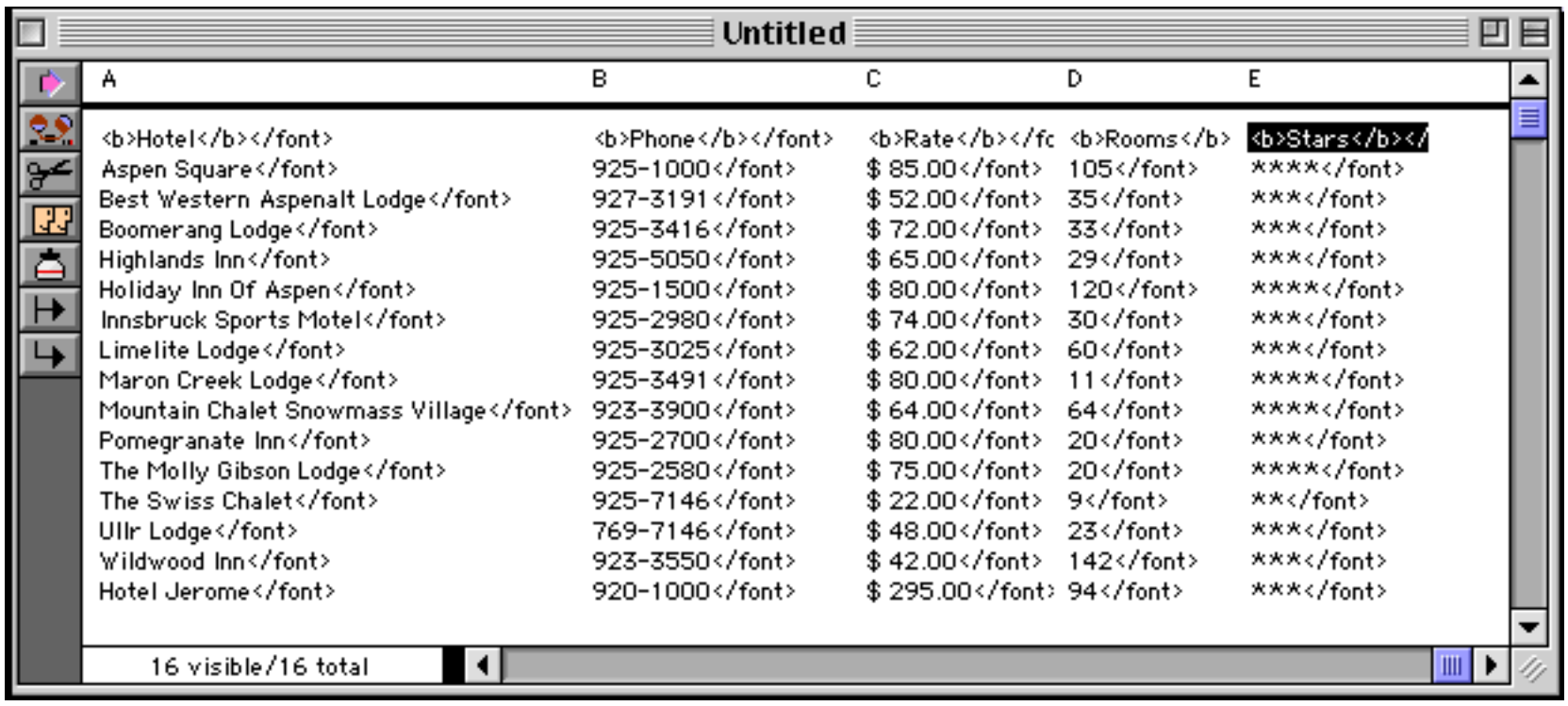
If you wish you can use the **Change** command to get rid of these extra tags (see "Change (Find and Replace)" on page 472). Start by replacing with nothing.



The command removes this tag from the first column.

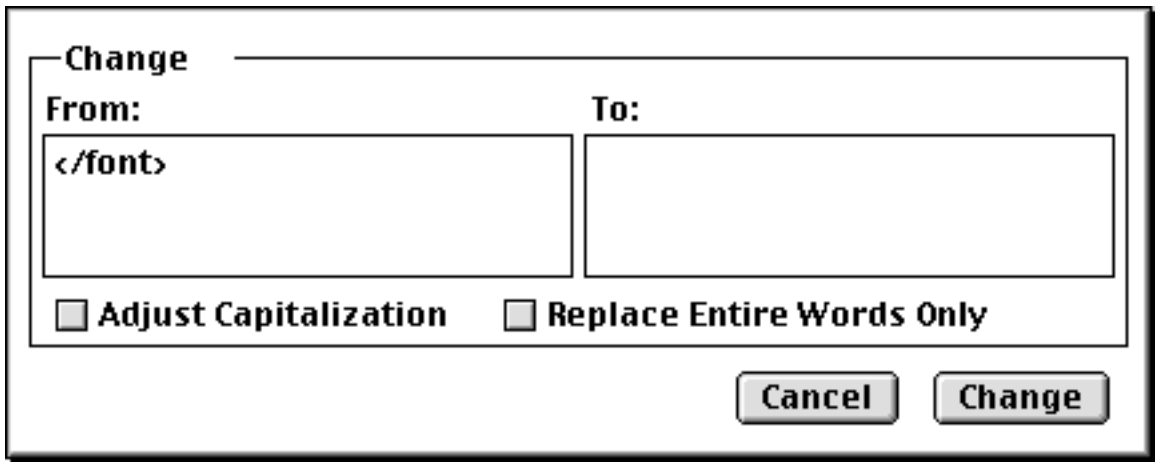


Repeat this process for each additional column. (This is an excellent potential application for automation with a procedure — see “Procedures” on page 1339.)



As you can see we’ve also adjusted the column widths. See “Changing the Width of a Field” on page 251 to learn how to do this.

Now we need to go back to the first column and replace with nothing.



The first column is now pretty much cleaned up.

Untitled

A	B	C	D	E
Hotel	Phone	Rate</fc	Rooms	Stars</
Aspen Square	925-1000	$ 85.00	105	****
Best Western Aspenalt Lodge	927-3191	$ 52.00	35	***
Boomerang Lodge	925-3416	$ 72.00	33	***
Highlands Inn	925-5050	$ 65.00	29	***
Holiday Inn Of Aspen	925-1500	$ 80.00	120	****
Innsbruck Sports Motel	925-2980	$ 74.00	30	***
Limelite Lodge	925-3025	$ 62.00	60	***
Maron Creek Lodge	925-3491	$ 80.00	11	****
Mountain Chalet Snowmass Village	923-3900	$ 64.00	64	****
Pomegranate Inn	925-2700	$ 80.00	20	***
The Molly Gibson Lodge	925-2580	$ 75.00	20	****
The Swiss Chalet	925-7146	$ 22.00	9	**
Ullr Lodge	769-7146	$ 48.00	23	***
Wildwood Inn	923-3550	$ 42.00	142	***
Hotel Jerome	920-1000	$ 295.00	94	***

16 visible/16 total

Repeating the process for the other columns.

Untitled

A	B	C	D	E
Hotel	Phone	Rate	Rooms	Stars
Aspen Square	925-1000	$ 85.00	105	****
Best Western Aspenalt Lodge	927-3191	$ 52.00	35	***
Boomerang Lodge	925-3416	$ 72.00	33	***
Highlands Inn	925-5050	$ 65.00	29	***
Holiday Inn Of Aspen	925-1500	$ 80.00	120	****
Innsbruck Sports Motel	925-2980	$ 74.00	30	***
Limelite Lodge	925-3025	$ 62.00	60	***
Maron Creek Lodge	925-3491	$ 80.00	11	****
Mountain Chalet Snowmass Village	923-3900	$ 64.00	64	****
Pomegranate Inn	925-2700	$ 80.00	20	***
The Molly Gibson Lodge	925-2580	$ 75.00	20	****
The Swiss Chalet	925-7146	$ 22.00	9	**
Ullr Lodge	769-7146	$ 48.00	23	***
Wildwood Inn	923-3550	$ 42.00	142	***
Hotel Jerome	920-1000	$ 295.00	94	***

16 visible/16 total

Cleaning up this data was a fair amount of work but was much easier than typing in the data all over again, and less prone to errors.

Importing OverVUE Files

OverVUE was ProVUE's first database program and the predecessor to Panorama. However, since the last version of OverVUE is almost twenty years old, Panorama V no longer supports import of OverVUE documents.

Re-Arranging Imported Data

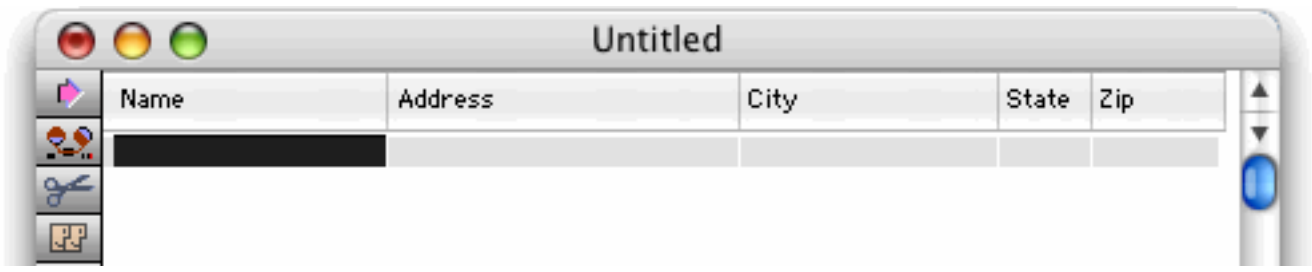
Sometimes the data you want to import is not set up the way you want. Perhaps the fields are in the wrong order, or the data is not split into fields correctly. The easiest way to handle this is to use the **Text Import Wizard** (see "Using the Text Import Wizard" on page 157). If you don't want to use the wizard, Panorama has lots of great tools for transforming the imported data after it is imported.

If the fields are simply in the wrong order, you can use the **Design Sheet** to re-arrange them. See "Re-Arranging Fields" on page 259 for details on re-arranging the field order.

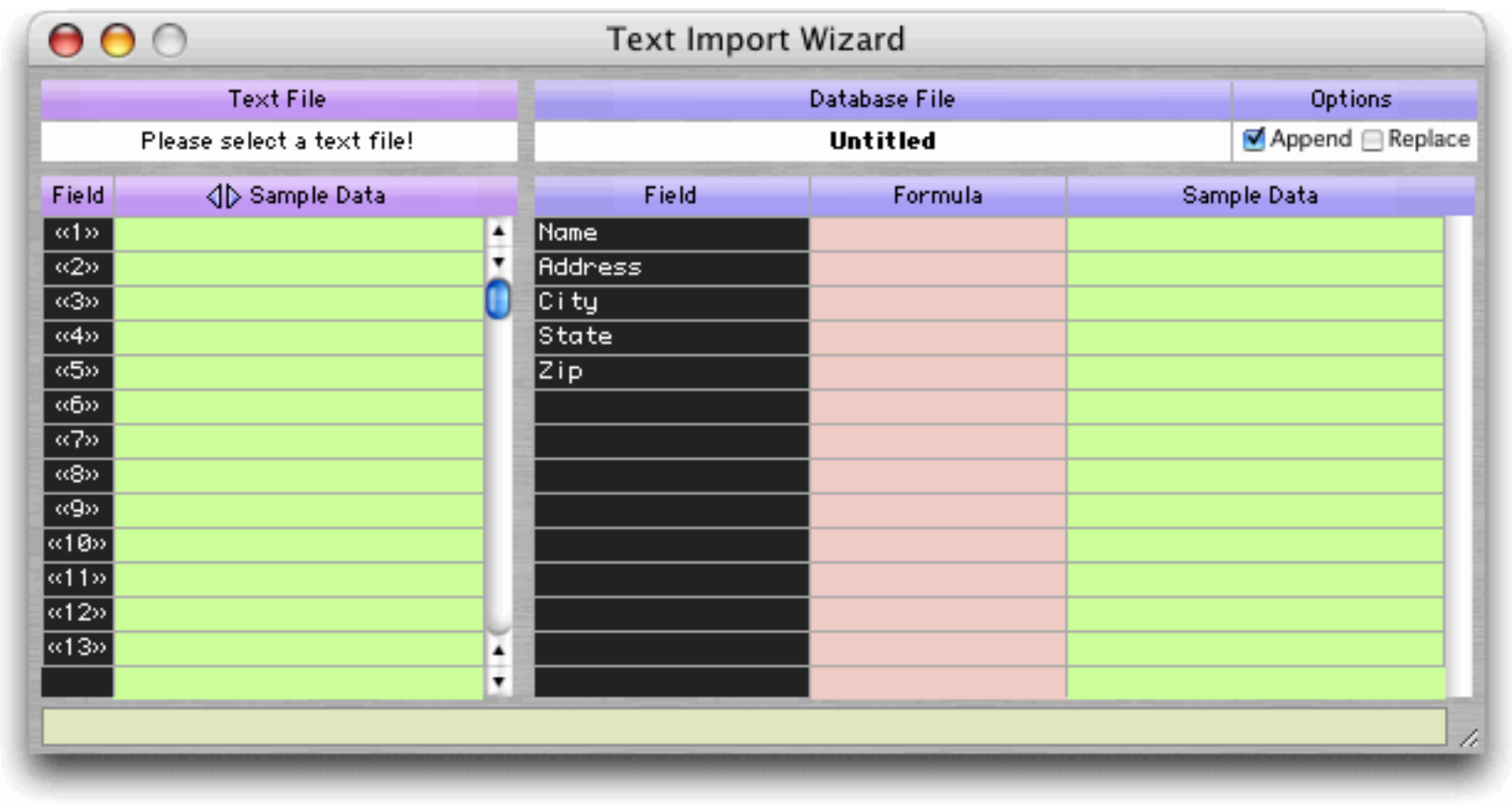
If the data is not split into fields properly using tabs or commas, you can use the **Formula Fill** command to massage the data into shape. See "Filling a Field with a Formula" on page 453 to learn more about this command.

Using the Text Import Wizard

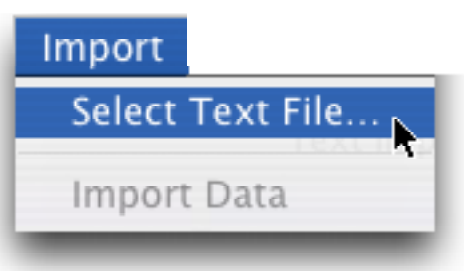
The **Text Import Wizard** makes it easy to import a text file into an existing database, even if the fields in the text file are not in the same order as the database fields. Usually you'll start the process by opening the database you want to import the data into. To illustrate this wizard we'll use a very simple mailing list file.



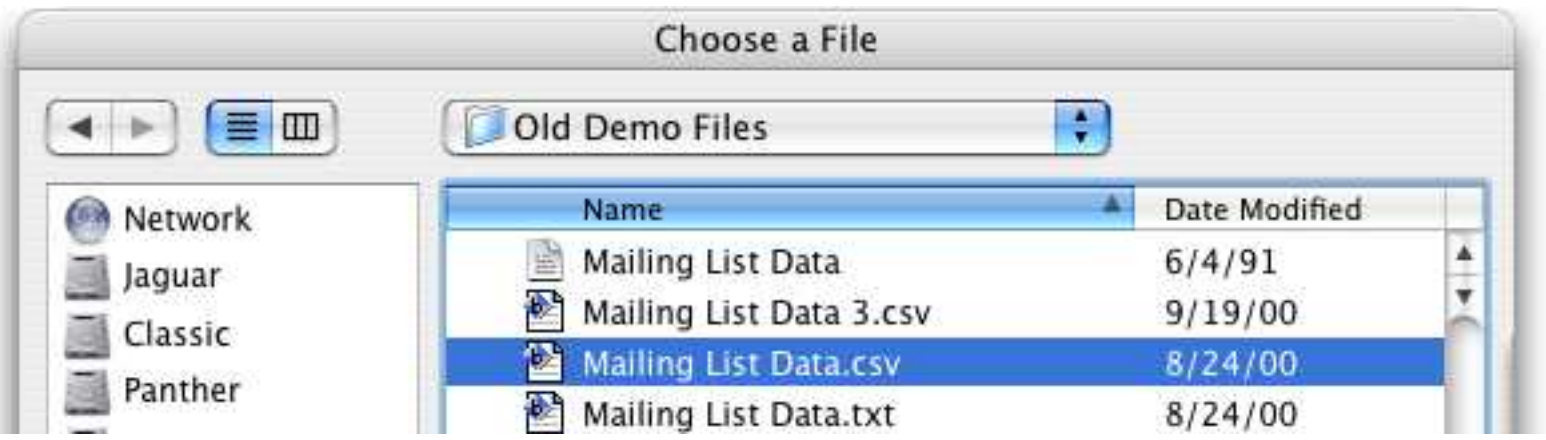
To start the import process open the **Text Import Wizard** from **Utilities** submenu in the **Wizard** menu. As you can see the right hand side of the window shows that we are going to import into the Mailing List file, and lists the five fields in this database.



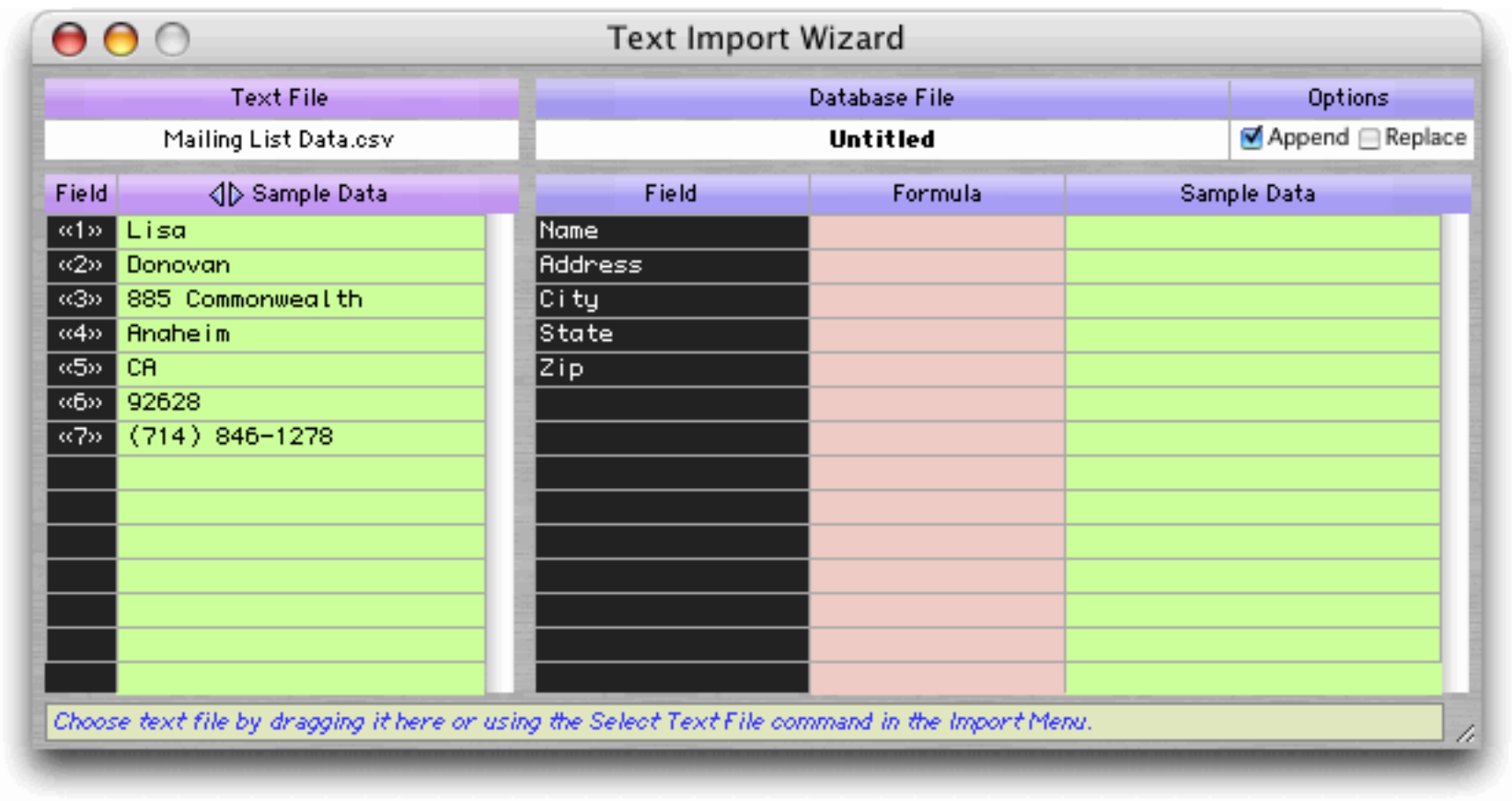
The next step is to select the text file you want to import into this database. To do this choose **Select Text File...** from the Import menu.



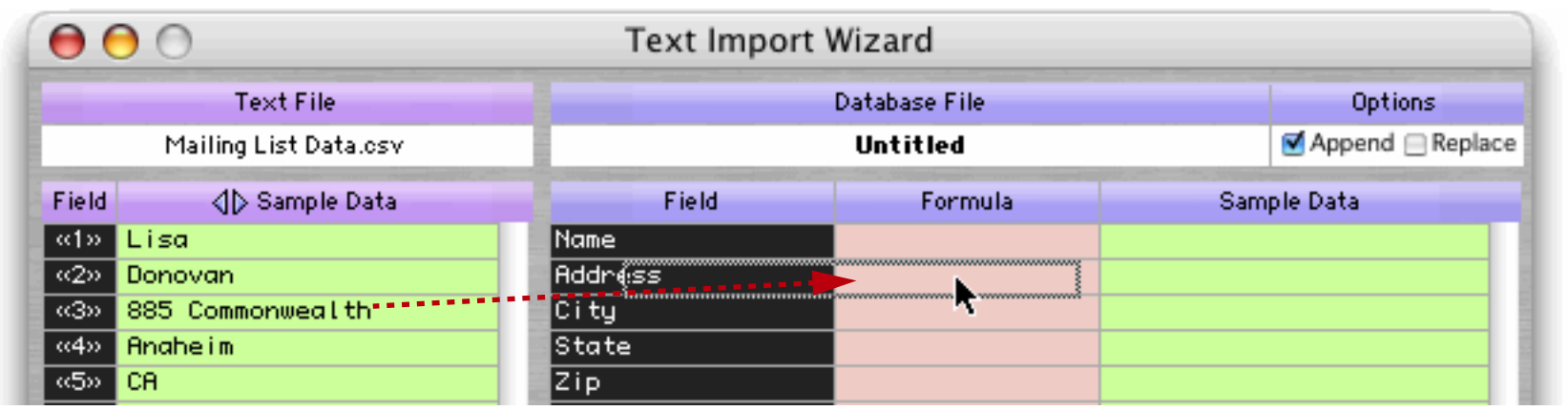
Use the dialog to select the text file you want to import.



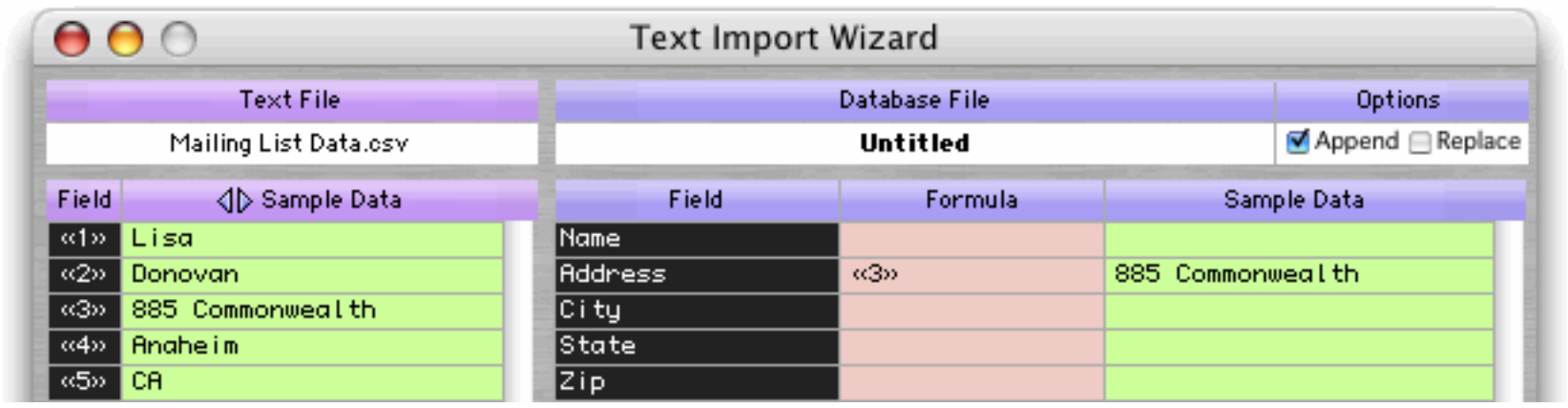
After you have selected the text file the wizard will display the name of the file (in this example Mailing List Data.csv) and contents of the first line.



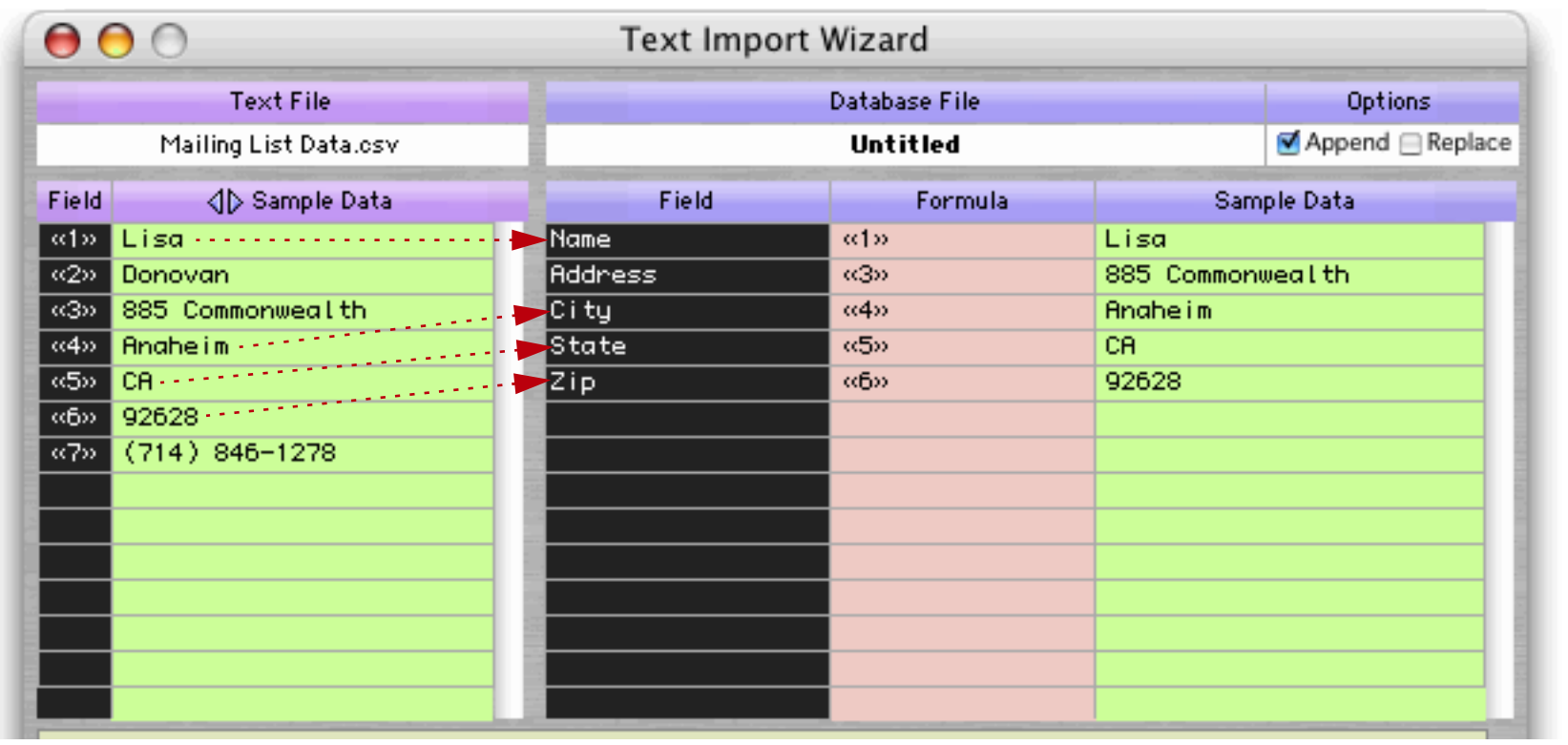
To set up the import configuration you'll need to drag from the left hand side onto the right hand side.



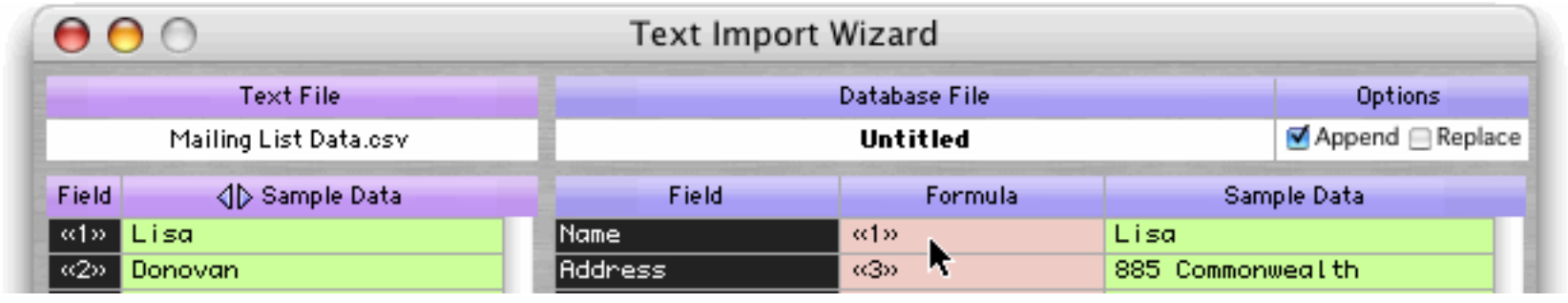
When you release the mouse the wizard will update the import configuration.



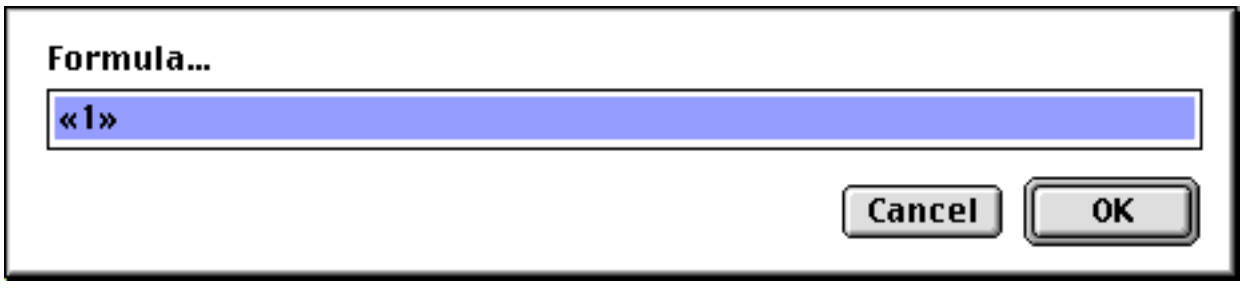
You'll need to drag each field you want to import across from the left to the right.



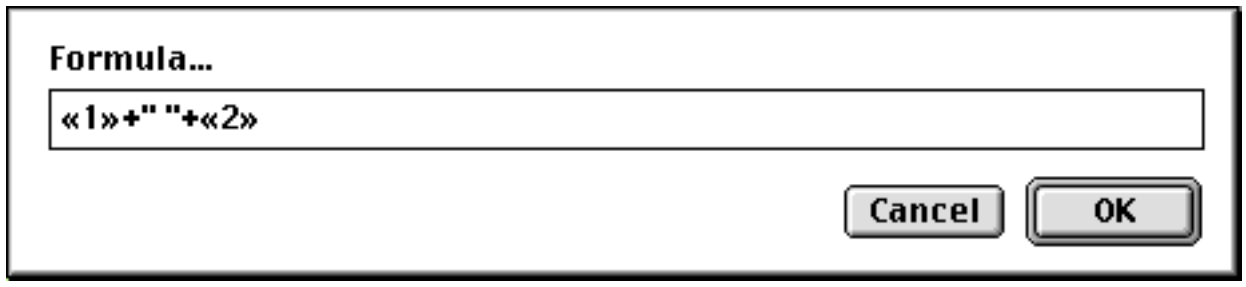
Sometimes the data you want to import doesn't match the fields in the database. In this case the database has only a single Name field, but the import data contains separate first and last names. Somehow these separate fields will need to be combined as the data is imported. The import wizard allows you to do this with a **formula** (see "Formulas" on page 1165). To edit the formula for a field click anywhere on the field's line.



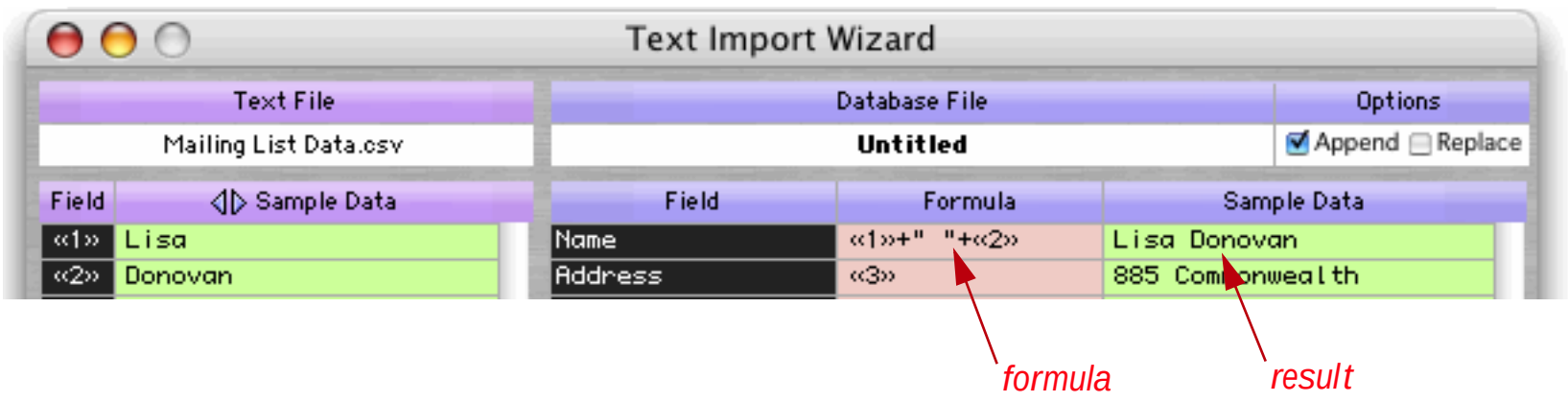
A dialog appears allowing you to edit the formula.



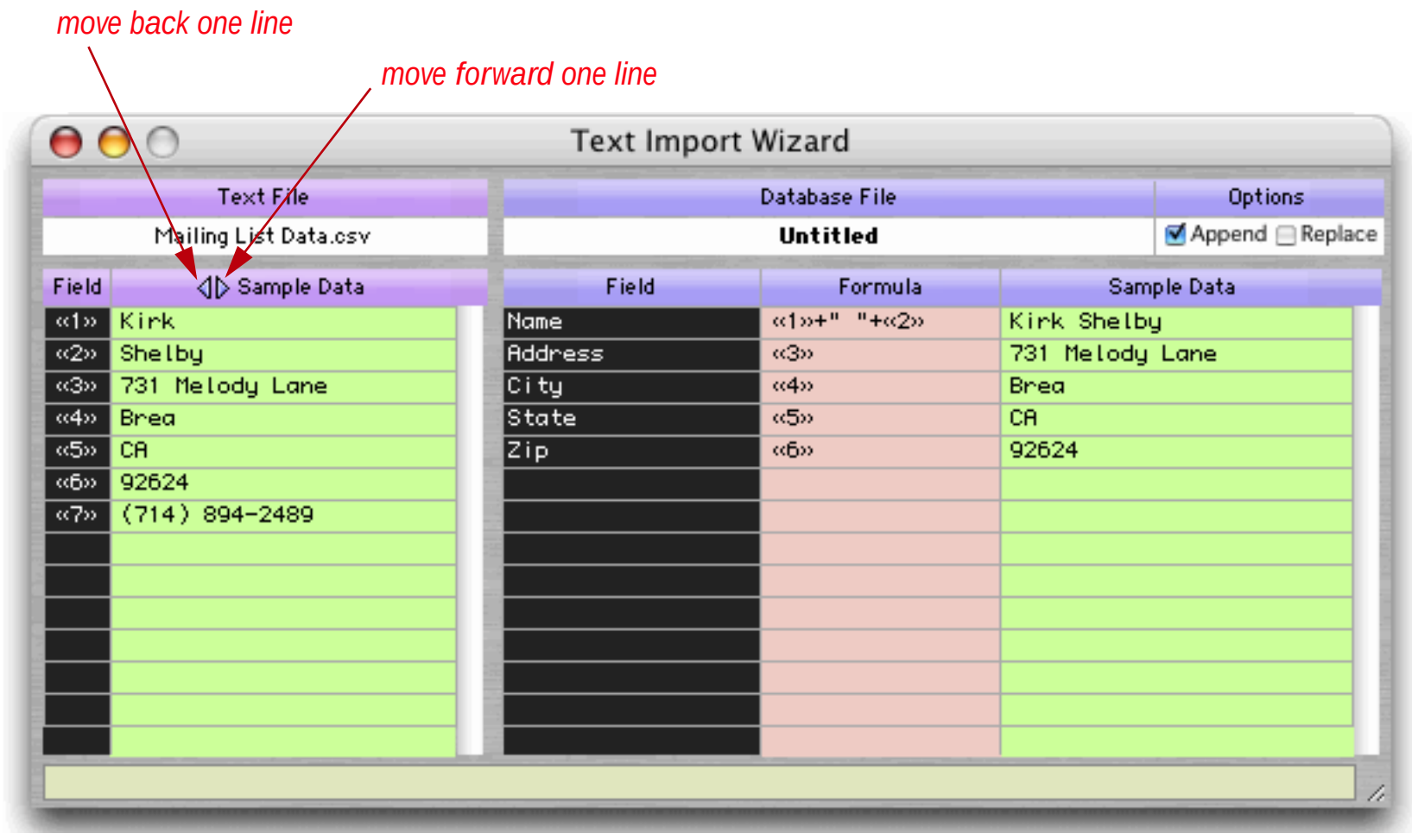
Within this formula you can include any import field by typing the field number in between « and » characters (see “Special Characters” on page 1202). To “glue” two text items together you can use the + symbol (see “Gluing Strings Together” on page 1212). To include constant text in the formula put the text inside quotes (see “Constants” on page 1195). The illustration below shows a formula that combines the first and last name into a single field (with a space in between).



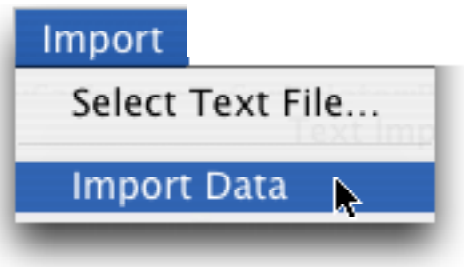
Press the **OK** button to preview the result of this formula.



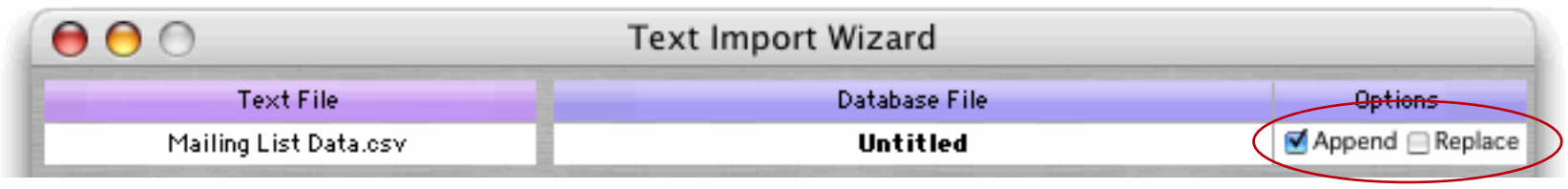
If you want to you can use the small arrows to preview additional lines in the imported data (not just the first line).



When all the fields you want to import are set up choose **Import Data** from the Import menu inside the window.



(Before you do you should make sure that the Append or Replace option you want is selected. Append will append the new data to whatever data is already in the database, while Replace will erase and replace the existing data.)



The wizard imports the data into the original database.



Common Import Formulas

Using a formula you can combine import fields, split import fields, convert to upper case or lower case, and much much more. You've already learned how to combine two or more import fields together with the + symbol like this (see "Gluing Strings Together" on page 1212).

```
«1»+" "+«2»

«2»+", "+«1»
```

To pick out a single word (for example a first or last name) you can use the `array(` function (see "Text Arrays" on page 1236). Here is a formula for picking the first name from a combined name field (assumed to be the first field, «1», and using the format First Last, for example John Smith).

```
array(«1»,1," ")
```

This formula extracts the last name.

```
array(«1»,2," ")
```

To convert text to upper case use the `upper(` function (see "String Modification Functions" on page 1224). This formula extracts the last name and converts it to upper case.

```
upper(array(«1»,2," "))
```

To extract only a limited number of characters use a **text funnel** (see "Taking Strings Apart (Text Funnels)" on page 1214). This formula extracts the first five characters from a zip code.

```
«6»[1,5]
```

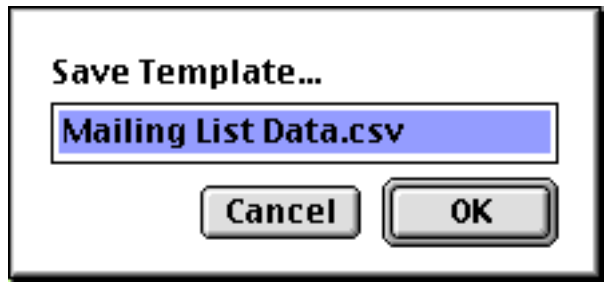
Here is an example that uses several of these techniques.



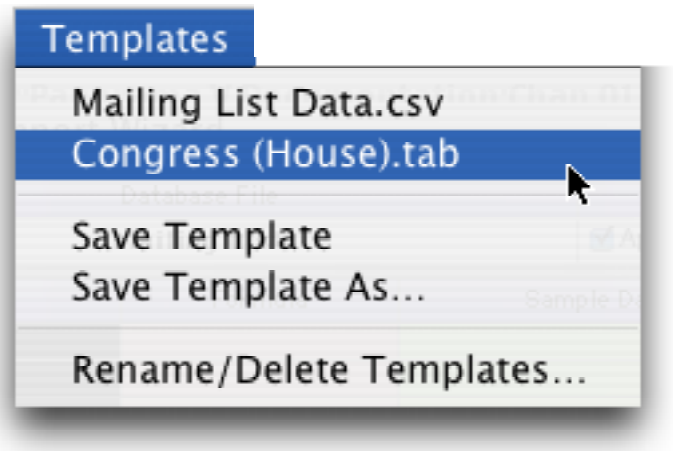
To learn more about Panorama formulas see "Formulas" on page 1165.

Import Templates

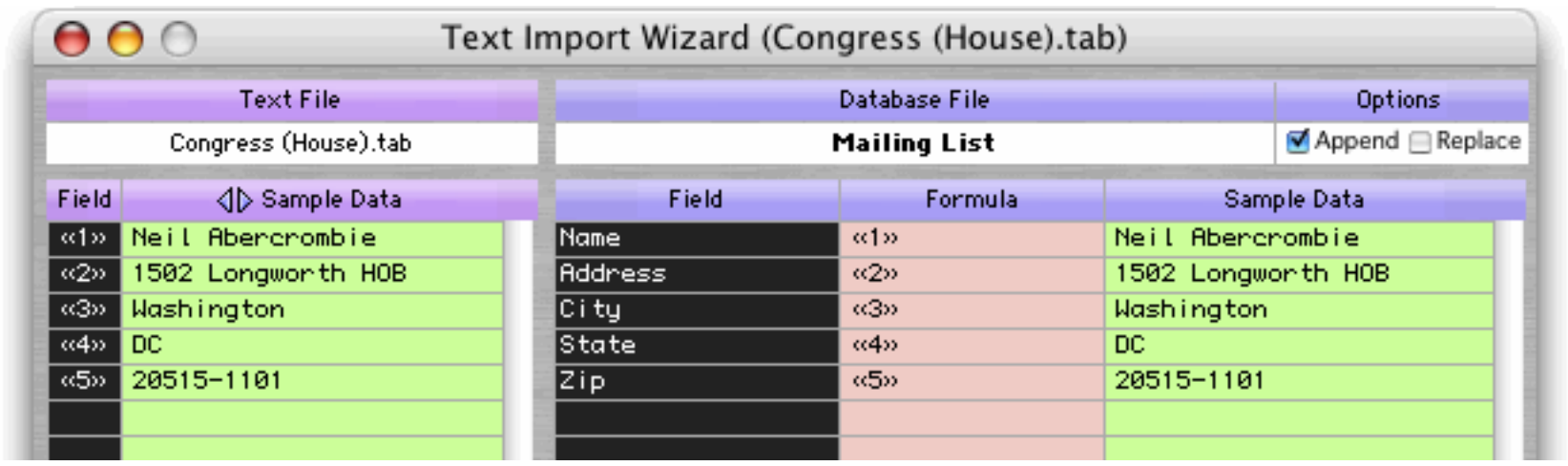
If you think you'll need to use an import configuration more than once you can save it as a **template**. The first step is to set up the configuration (as described in the previous section). Once the configuration is set up you can save it with the **Save Template** or **Save Template As…** commands in the Template menu. The wizard will prompt you to type in a name for the new template (the default is the name of the text file being imported).



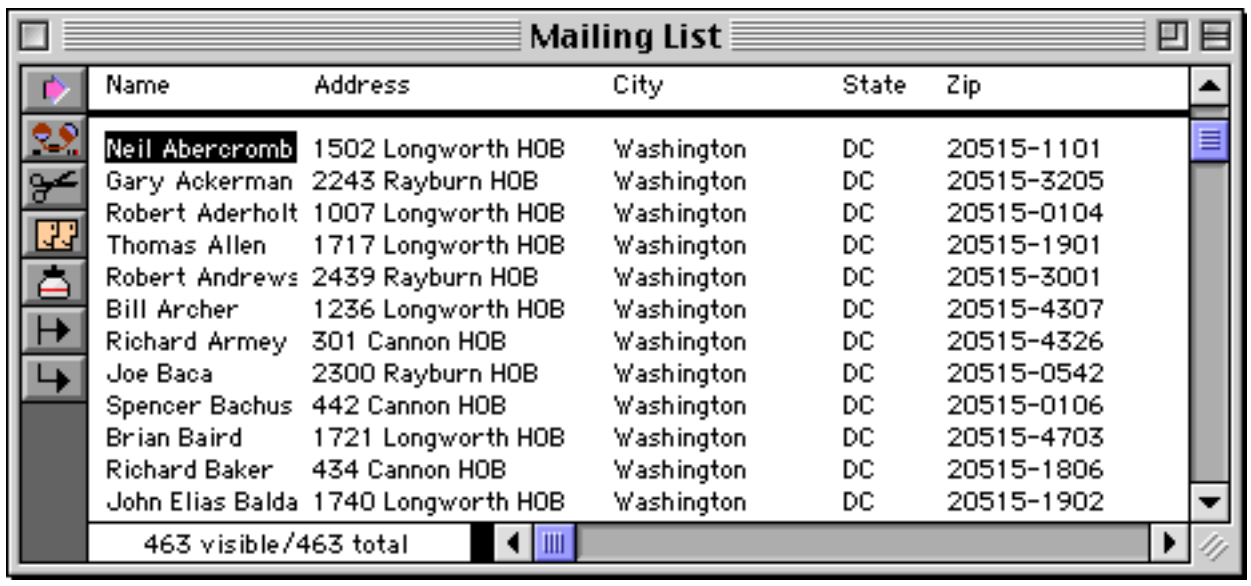
Once a template has been saved you can open it again by selecting it from the **Template** menu. (Note: The template is actually stored in the database being imported into (in this case Mailing List). The template is only available when that database is being imported into. Each database may contain its own separate set of templates, which makes sense since the import configuration used with one database is not likely to work with any other database.)



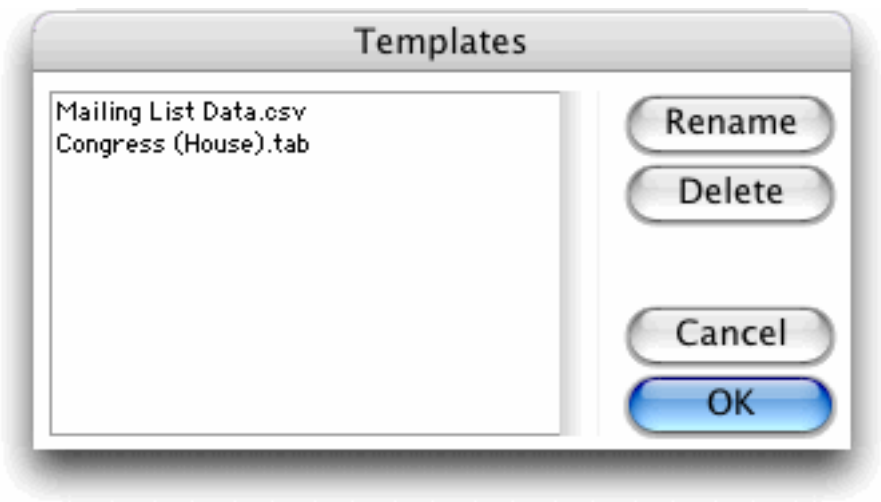
The wizard loads the entire import configuration, ready to go.



You can use the configuration as is or modify it before you actually import the data.



If you want to delete or rename a template choose the **Rename/Delete Templates...** command from the Template menu. This opens the dialog shown below. To rename a template first click on it and then press the **Rename** button. A dialog appears allowing you to type in a new name. To delete a template press the **Delete** button.



When you are done press the **OK** button.

Choosing a Database to Import Into

The **Text Import Wizard** normally imports into database that was active when the wizard was opened. However, you can use the **Database** menu to choose to import into any open database. Simply choose the database you want to import into and then set up the configuration.

Converting an Import Configuration into a Procedure

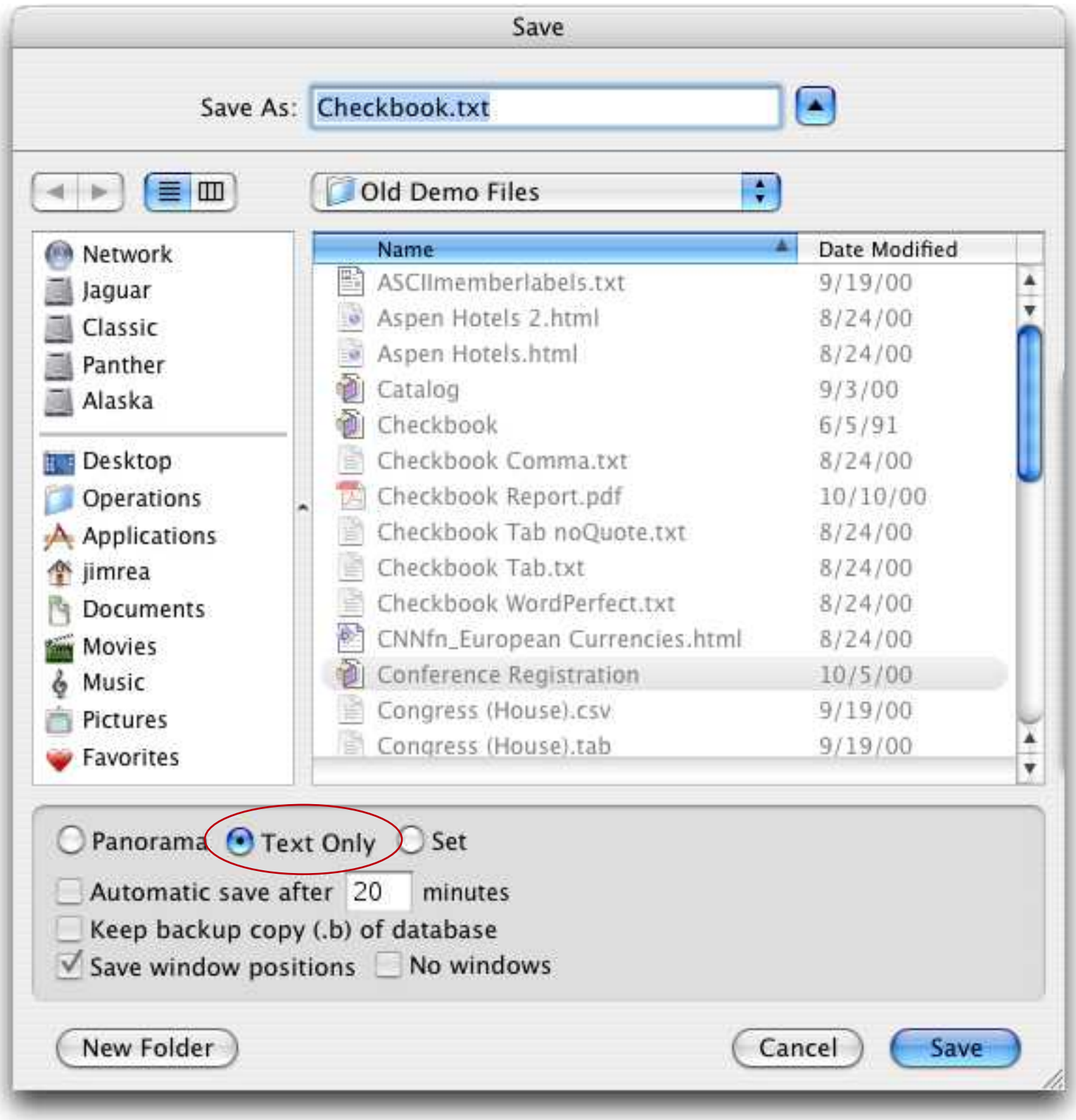
To convert the current import configuration into a procedure choose the **Copy Procedure to Clipboard** command from the Special menu. Now create a procedure (see "Writing a Procedure from Scratch" on page 1352) and **Paste** the automatically generated procedure into it.



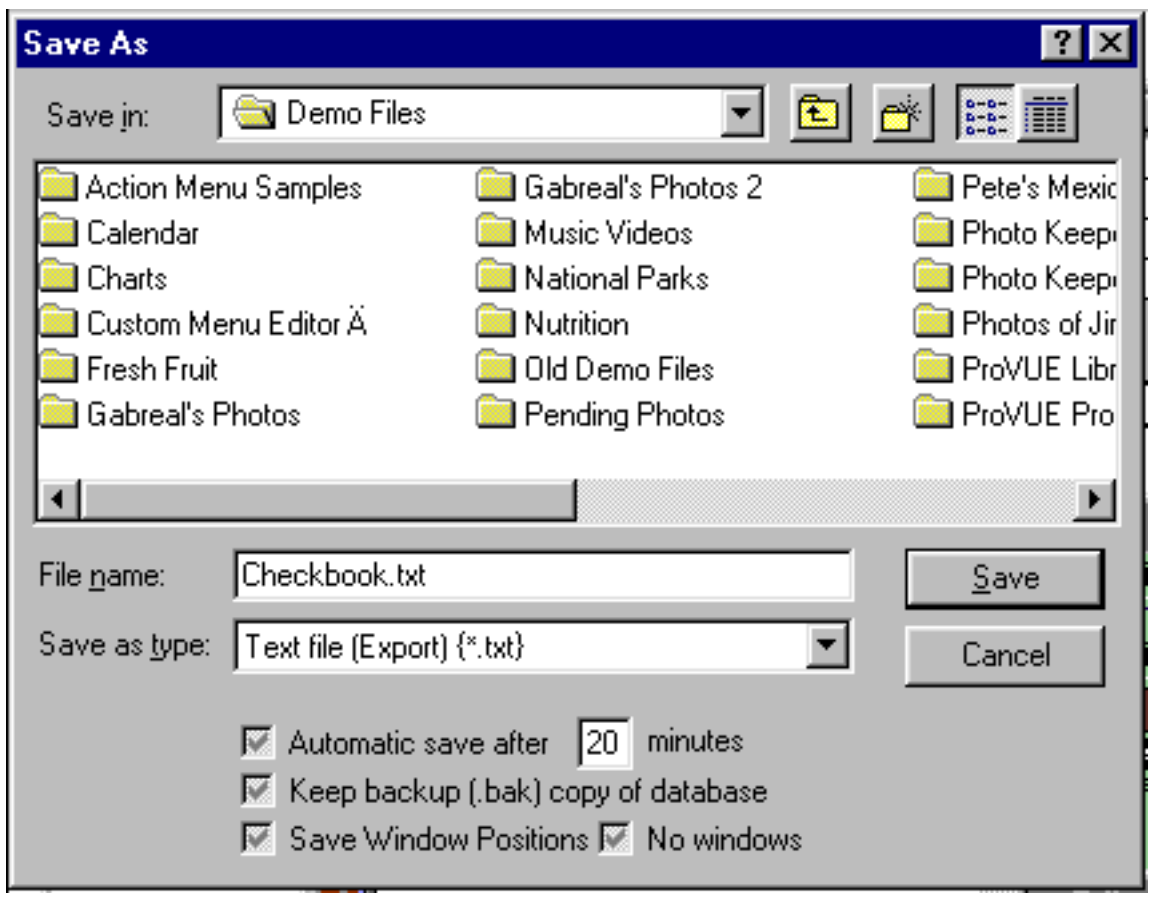
For more information on creating and using procedures see "Procedures" on page 1339.

Exporting a Text File

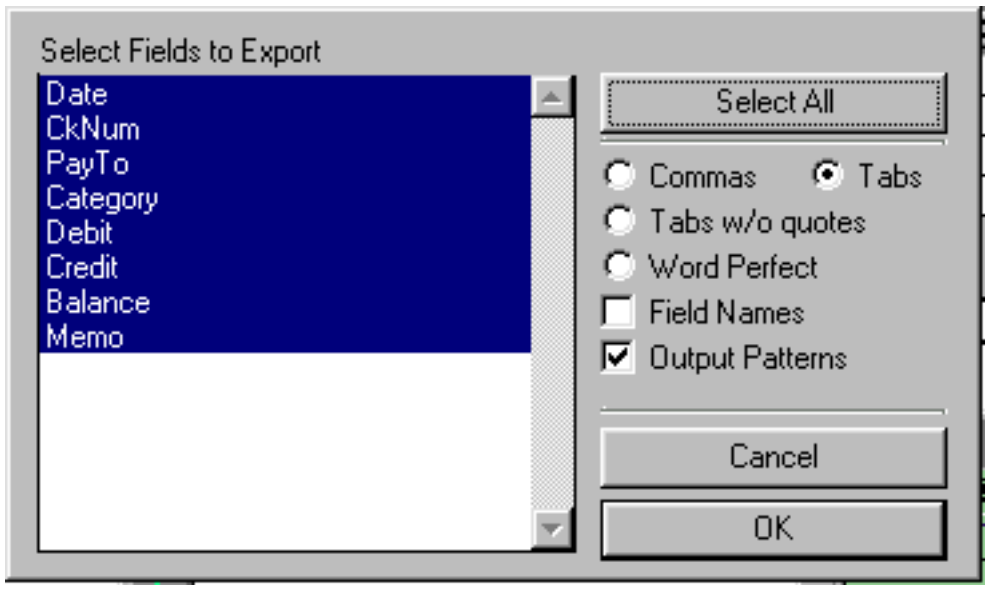
To export the selected records (see “Select” on page 375) in the current database into a text file, use the **Save As** command in the File menu. The standard save dialog will appear on the screen. If you are using a Macintosh computer select the folder and name of the text file and click the Text Only radio button. (The name must be different from the database name.)



If you are using a Windows PC computer use the combo box to choose the Text file (Export) {*.txt} option.



Once the name and text only option are set up, click the **Save** button. This opens a second dialog that allows you to specify which fields to include in the exported data and what format to use.



Only the fields you select will be included in the text file. To select a field, click the mouse on the field name. To select additional fields, click on them also. To un-select a field, click on the field again. If you want to export all the fields in the database, you can quickly select them all with the **Select All** button.

The four radio buttons—Commas, Tabs, Tabs w/o quotes, and Word Perfect—let you pick how the text file is formatted.

Option	Description	Example
Commas	Exported file will have commas between each field and a carriage return at the end of each line. If a data cell contains a comma, it will be surrounded by quotes (").	01/08/90,1907,Northern Illinois Mold,96.05 01/08/90,1908,U S Postmaster,75.00 01/08/90,1909,"Advertiser's Mailing Service, Inc.",390.80 01/16/90,1910,"Coudert Brothers, Attorney's At Law",223.52 01/16/90,1911,Paramount Stationers,105.84 01/17/90,1912,California Capitol,36.00 01/17/90,1913,California Capitol,28.00 01/17/90,1914,U S Postmaster,75.00 01/17/90,1915,Sacramento Bee,795.00 01/22/90,1916,Walthers,"12,463.00" 01/22/90,1917,Blue Cross Of Calif,279.03
Tabs	Exported file will have an invisible tab character between each field and a carriage return (Mac) or carriage-return/line feed (Windows) at the end of each line. If a data cell contains a comma, it will be surrounded by quotes (").	01/08/90 1907 Northern Illinois Mold 96.05 01/08/90 1908 U S Postmaster 75.00 01/08/90 1909 "Advertiser's Mailing Service, Inc." 390.80 01/16/90 1910 "Coudert Brothers, Attorney's At Law" 223.52 01/16/90 1911 Paramount Stationers 105.84 01/17/90 1912 California Capitol 36.00 01/17/90 1913 California Capitol 28.00 01/17/90 1914 U S Postmaster 75.00 01/17/90 1915 Sacramento Bee 795.00 01/22/90 1916 Walthers "12,463.00" 01/22/90 1917 Blue Cross Of Calif 279.03
Tabs w/o quotes	Exported file will have an invisible tab character between each field and a carriage return (Mac) or carriage-return/line feed (Windows) at the end of each line. No quotes are added even if the data contains commas.	01/08/90 1907 Northern Illinois Mold 96.05 01/08/90 1908 U S Postmaster 75.00 01/08/90 1909 Advertiser's Mailing Service, Inc. 390.80 01/16/90 1910 Coudert Brothers, Attorney's At Law 223.52 01/16/90 1911 Paramount Stationers 105.84 01/17/90 1912 California Capitol 36.00 01/17/90 1913 California Capitol 28.00 01/17/90 1914 U S Postmaster 75.00 01/17/90 1915 Sacramento Bee 795.00 01/22/90 1916 Walthers 12,463.00 01/22/90 1917 Blue Cross Of Calif 279.03

Option	Description	Example
Word Perfect	Exported file will use Word Perfect's unique mail merge format, which requires a Control-R / Carriage Return between each field and a Control-E / Carriage Return between each record.	

The Field Names checkbox lets you choose whether you want the first record of the text file to list the field names. If this button is checked, an extra line will be added to the top of the text file. This extra line will contain the names of each field.

```
Date,Num,Pay To,Debit
01/08/90,1907,Northern Illinois Mold,96.05
01/08/90,1908,U S Postmaster,75.00
01/08/90,1909,"Advertiser's Mailing Service, Inc.",390.80
01/16/90,1910,"Coudert Brothers, Attorney's At Law",223.52
01/16/90,1911,Paramount Stationers,105.84
01/17/90,1912,California Capitol,36.00
01/17/90,1913,California Capitol,28.00
01/17/90,1914,U S Postmaster,75.00
01/17/90,1915,Sacramento Bee,795.00
01/22/90,1916,Walthers,"12,463.00"
01/22/90,1917,Blue Cross Of Calif,279.03
```

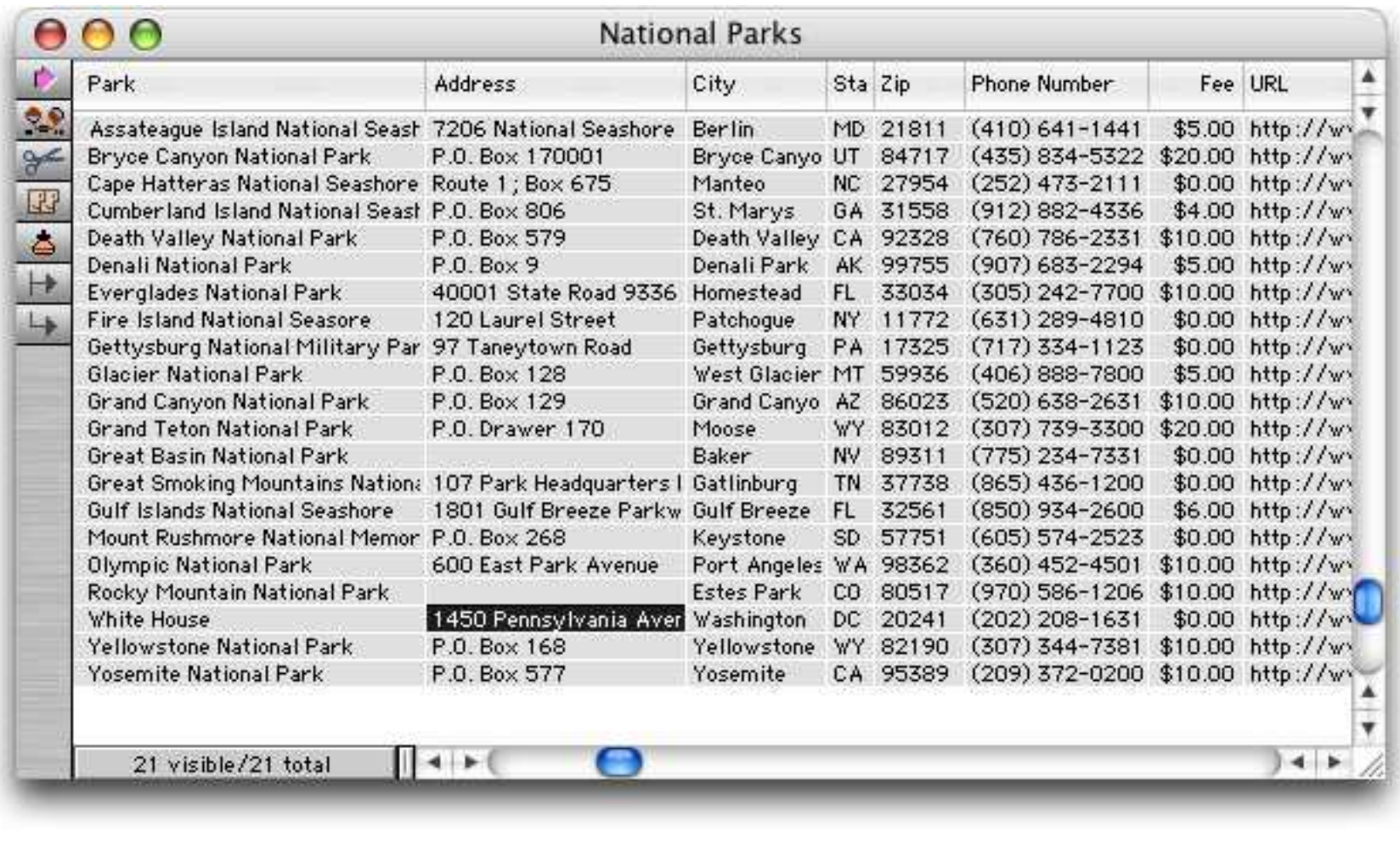
The Output Patterns checkbox controls whether the data is exported using the output patterns set up for each field (see "Numeric Output Patterns" on page 290 and "Date Output Patterns" on page 295), or in the standard numeric and date formats. Checking this box will cause numeric and date columns to be exported in using the output patterns you have set up for each field.

Once you have selected the fields you want exported and checked the appropriate buttons and boxes, pressing the **OK** button will export the data to a text file on the disk.

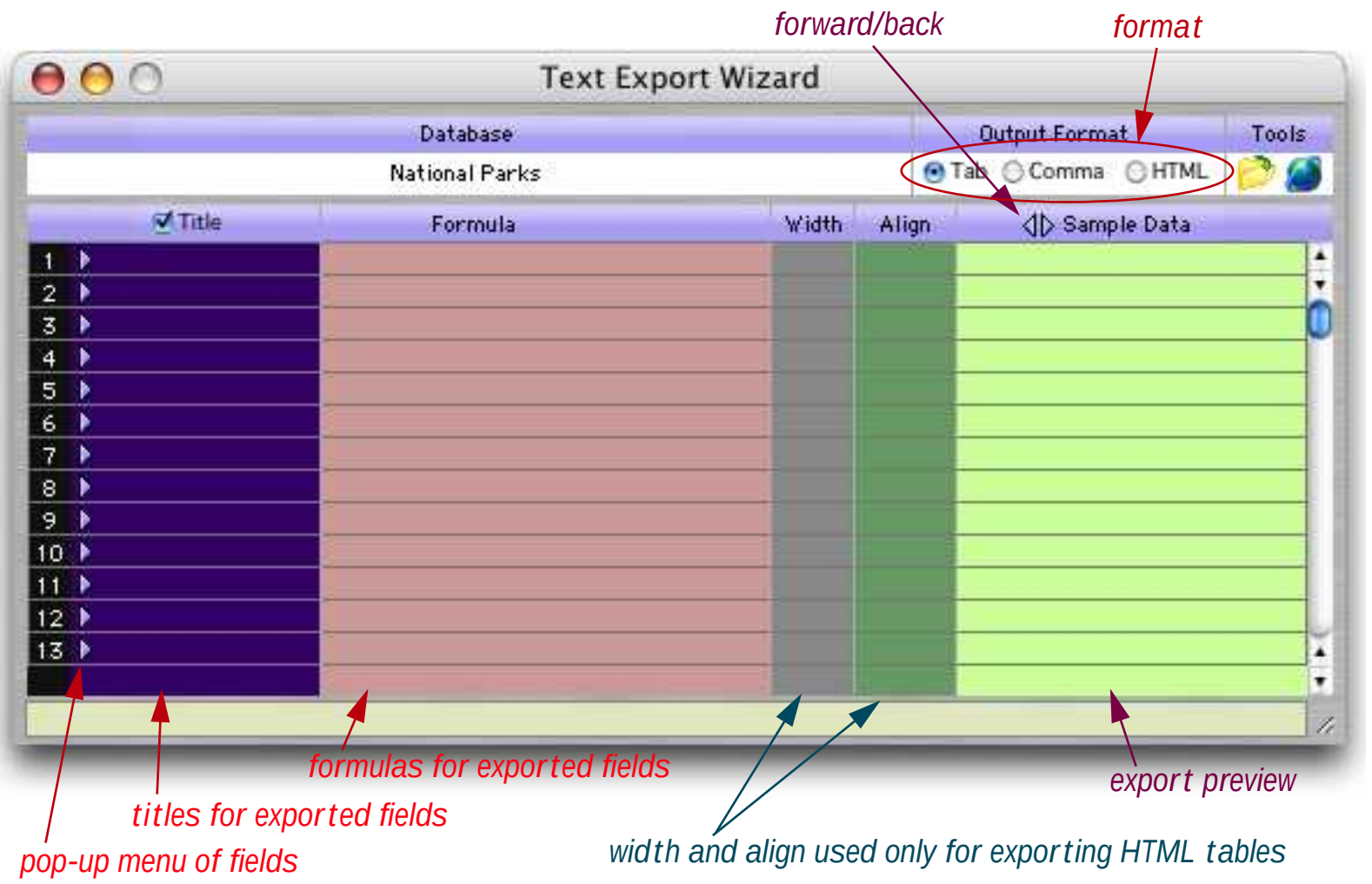
The **Save As** command exports only the selected records in the database. If you don't want to export the entire database, use the **Find/Select** command to choose just the records you want (see "Select" on page 375) before using the **Save As** command. Only the selected records will be exported.

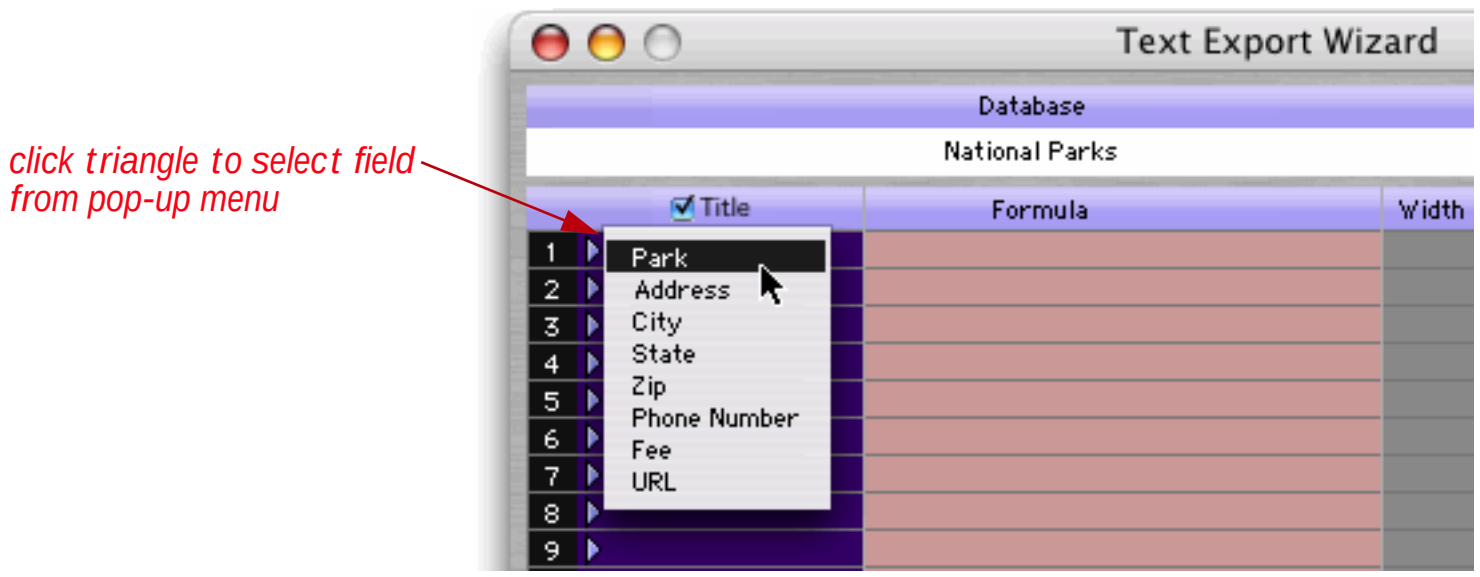
Exporting with the Text Export Wizard

The **Save As** dialog gives you limited control over the format of the text you are exporting. For more control you can use the **Text Export Wizard**. This wizard allows you to specify the order of the fields being exported, and to manipulate the data as it is being exported (converting it to upper case, for example, or combining several database fields into one export field). The wizard can even be used to convert the database into an HTML table. To illustrate this wizard we will use this database of national parks.



To begin the export process choose **Text Export Wizard** from the Utilities submenu of the Wizard menu.





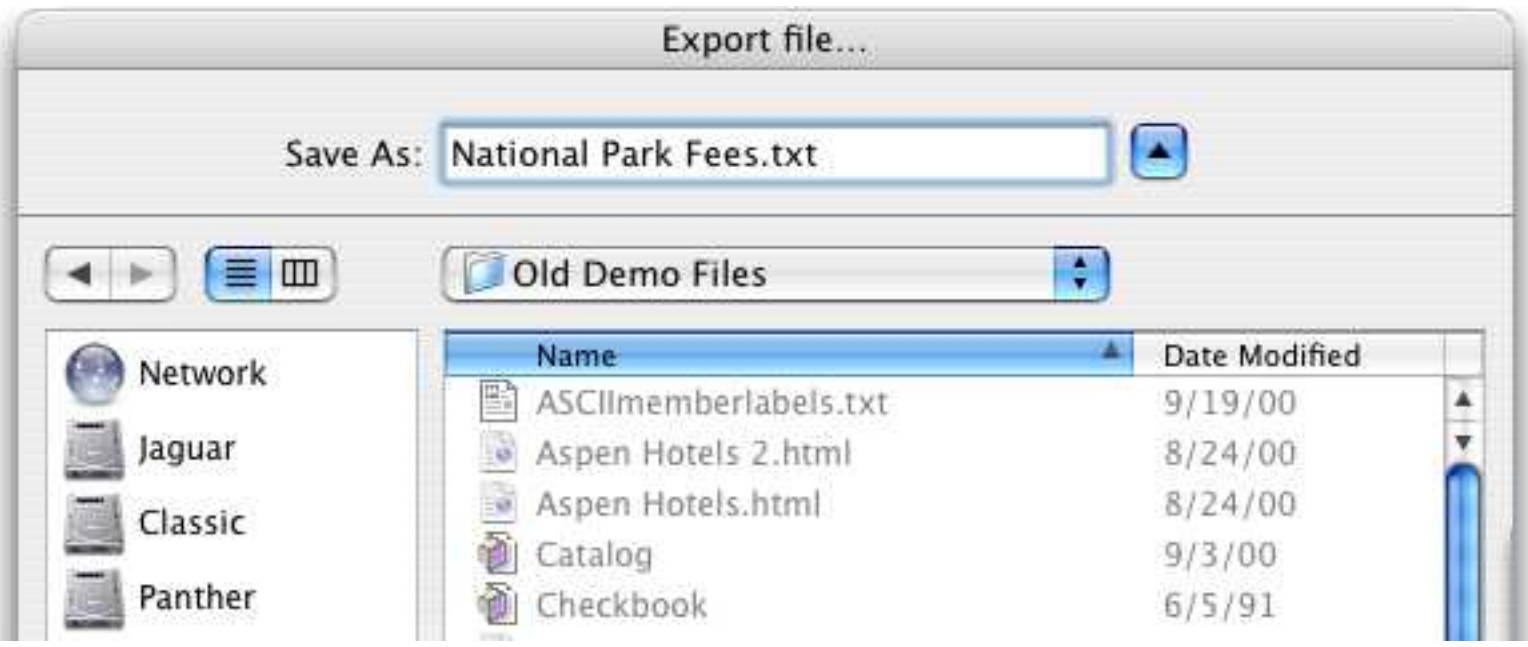
When you make your choice the wizard will fill in one line of the configuration.



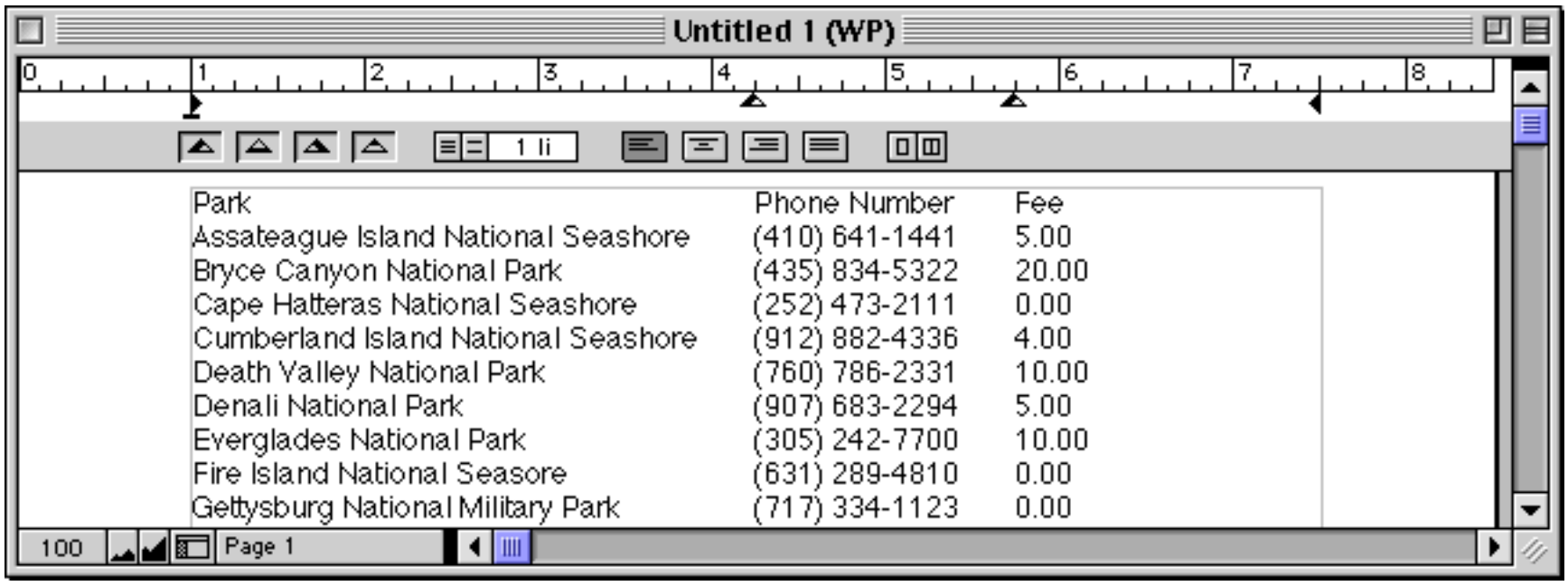
Repeat the process to select all of the fields you want to export. If a numeric field is selected the wizard will use the `pattern(` function (see "Converting Between Numbers and Strings" on page 1227) to convert the number into text, as shown below. If a date field is selected the `datepattern(` function will be used to perform the conversion (see "Converting Between Dates and Text" on page 1249).



Once the configuration is set up select the **Export Text File** command in the Export menu to actually export the data. (You can also click on the export icon in the Tools box.) This command will ask you to select a folder for the exported file, and to type in a file name.



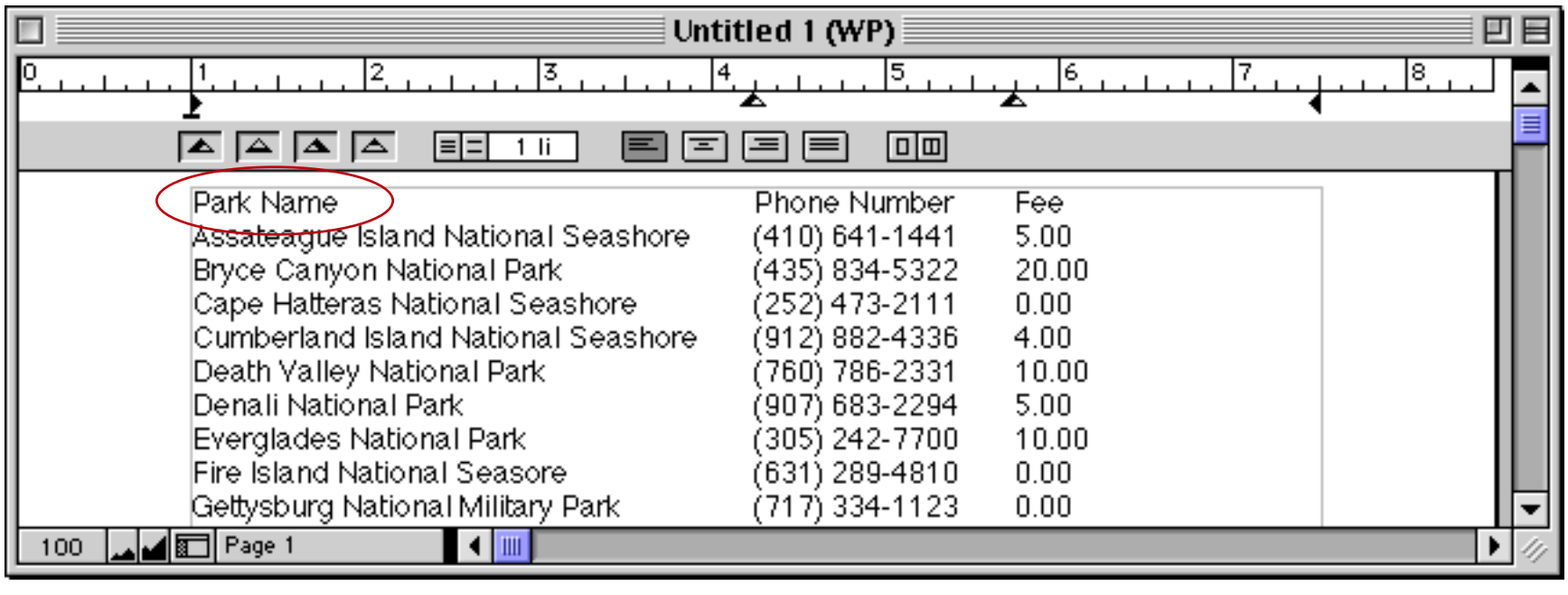
When you press the **Save** button the file will be exported. Here's what the resulting text file looks like when it is opened with a word processor (in this case ClarisWorks™). We've set the tab stops to make it easier to see how the data has been exported as separate, tab separated columns.



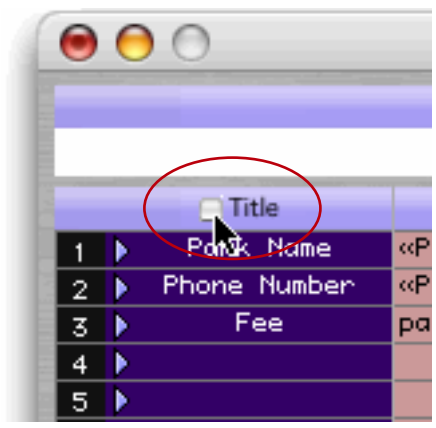
The first line of this exported file contains the title of each column (Park, Phone Number, Fee). You can edit a title simply by clicking on it.



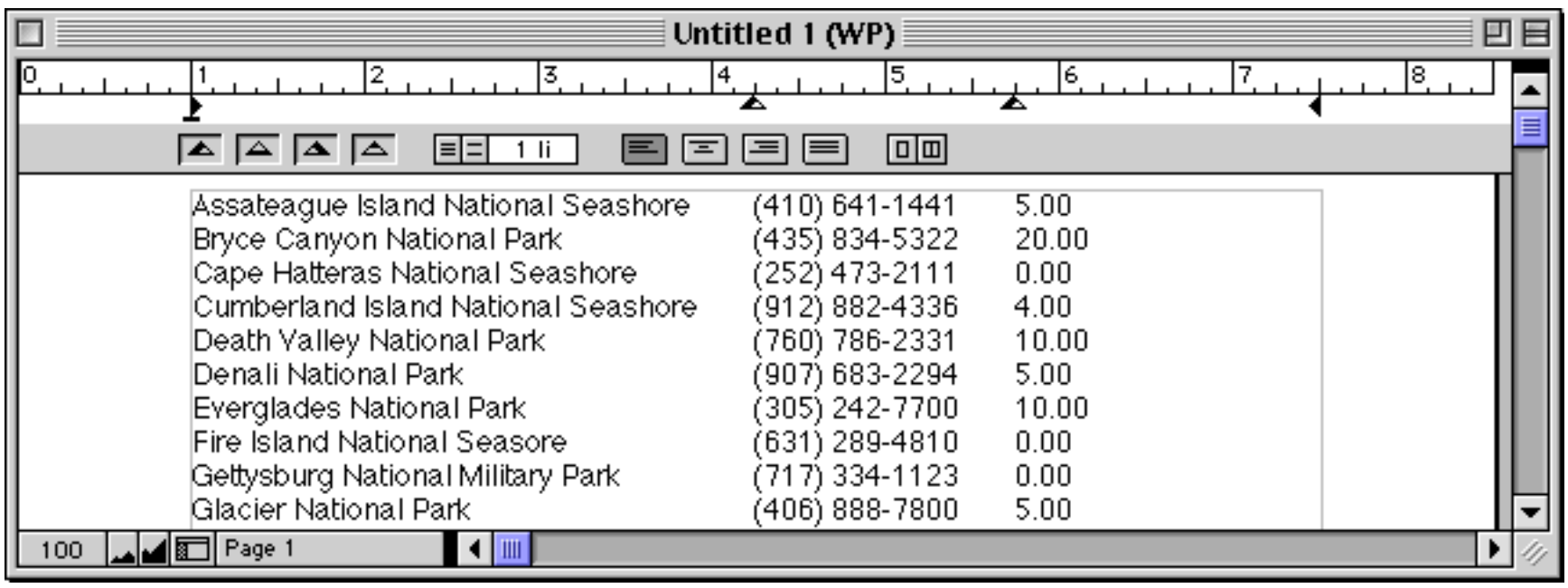
When you are done press the **Enter** key. If you re-export the data and examine it you will see the new title.



If you want to leave out the first line of titles simply un-check the Title option.

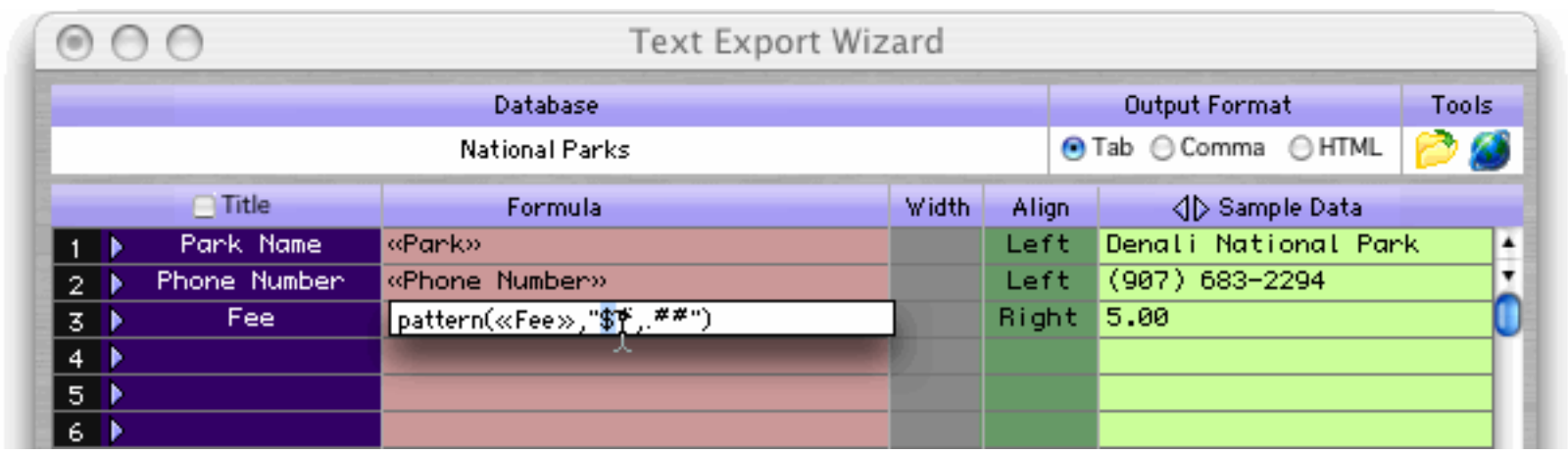


With this option turned off the exported data starts right on the first line, with no titles at all.



Editing the Export Configuration

To edit the title, formula, width or alignment simply click on the item you want to edit. A pop-up editing window will appear (see “The Input Box” on page 310).

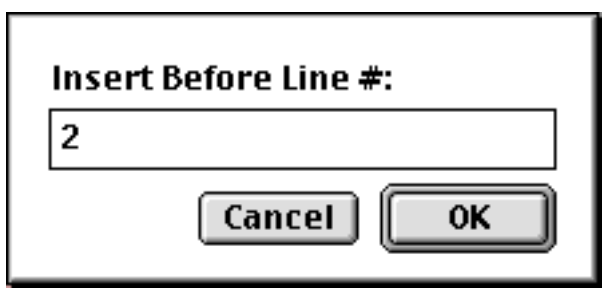


If necessary you can expand the input box by dragging on the bottom right hand corner (see “Expanding the Input Box” on page 311).



Press the **Enter** key when you have completed editing the item.

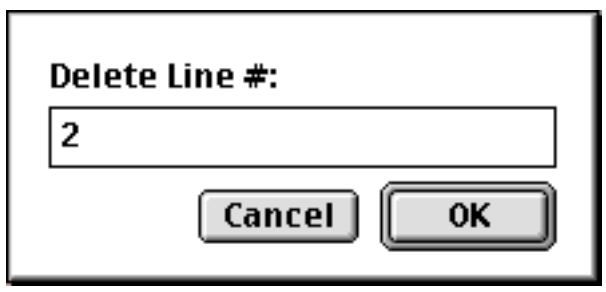
To insert a new item in the middle choose **Insert Line...** from the Edit menu. The wizard will ask you where you want the new line inserted.



When you press **OK** the new line is inserted.



To delete an item use the **Delete Line...** command in the Edit menu. The wizard will ask you which line you want to delete.



Enter the line number and press **OK** to delete the line.

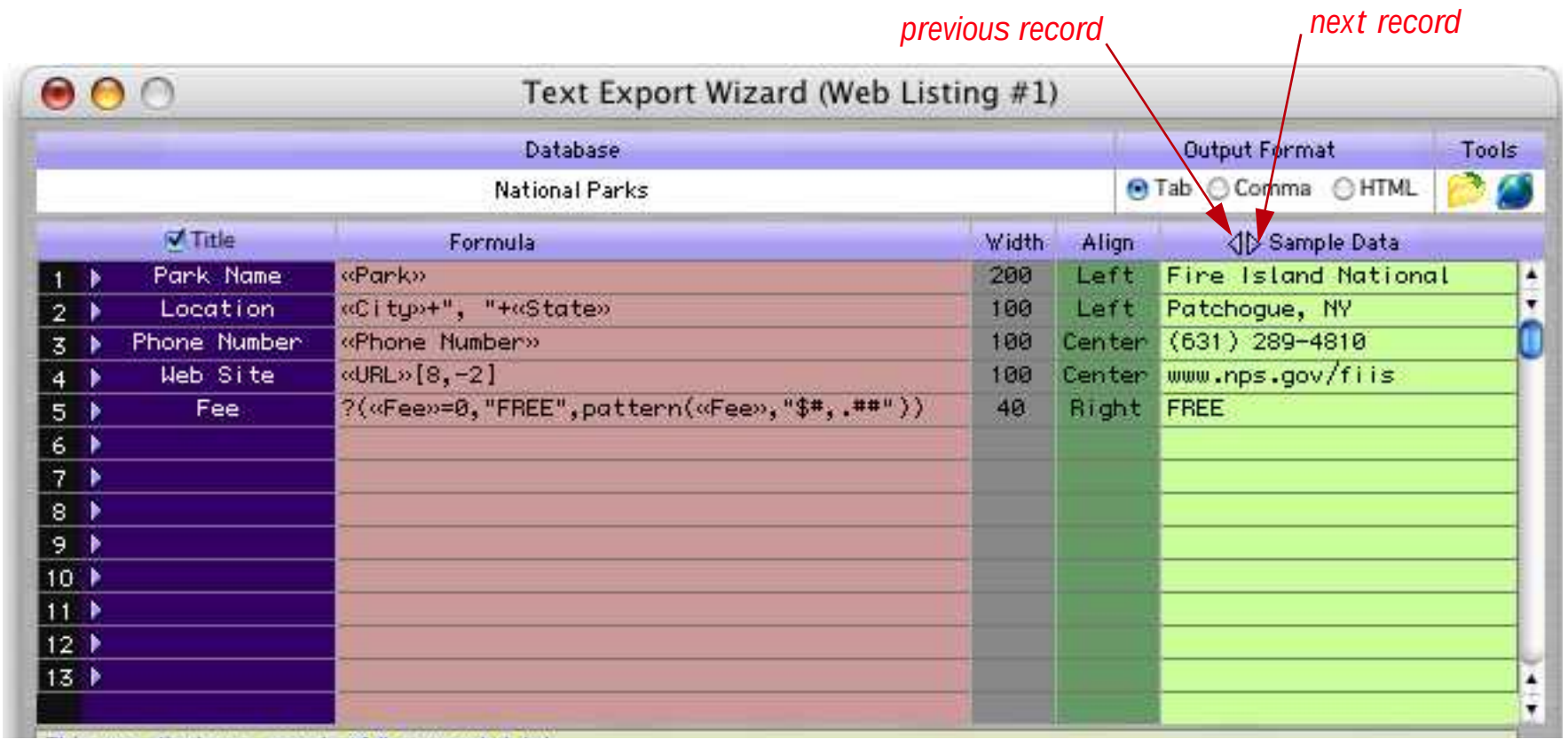
Common Export Formulas

The **Text Export Wizard** allows you to use any valid Panorama formula (see "Formulas" on page 1165) to create each item. The example below contains several formulas.



The second export field, Location, combines two database fields, City and State (see "Gluing Strings Together" on page 1212). The fourth export field, Web Site, uses a text funnel to remove http:// and / from each URL (see "Taking Strings Apart (Text Funnels)" on page 1214).The final export field, Fee, uses the ?(function to check for parks where the access fee is zero. If the fee is zero then the word FREE is exported instead of $0.00 (see "The ? Function" on page 1271). This formula also uses the pattern(function (see "Converting Between Numbers and Strings" on page 1227). See "Using Formulas to Display Text" on page 621 for more examples of formulas that are useful for exporting data.

The right hand side of the window displays a sample of the exported data for the current record in the database. This preview can be very handy for checking the result of your formulas. Using the small arrows you can move back and forth to preview other records in the database.

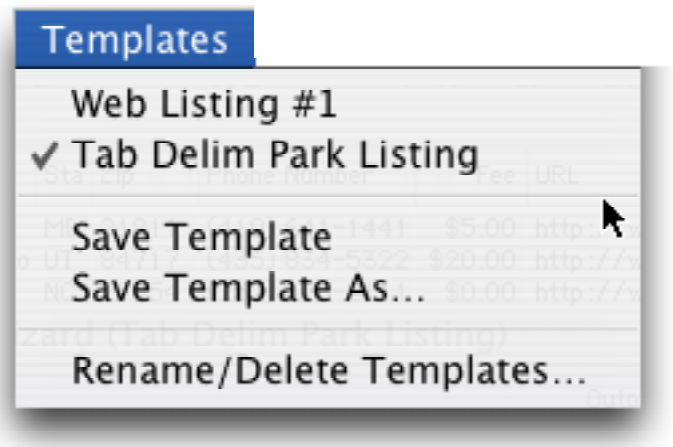


Export Templates

If you think you'll need to use an export configuration more than once you can save it as a **template**. The first step is to set up the configuration (as described in the previous section). Once the configuration is set up you can save it with the **Save Template** or **Save Template As...** commands in the Template menu. The wizard will prompt you to type in a name for the new template (the default is the name of the text file being imported).

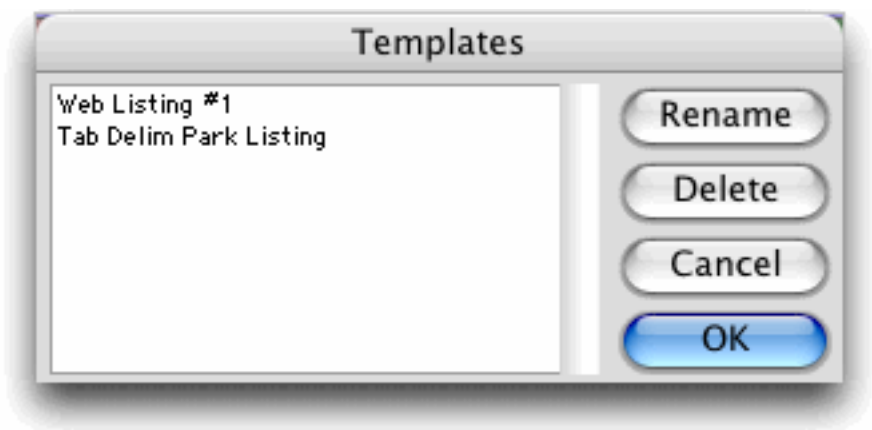


Once a template has been saved you can open it again by selecting it from the Template menu. (Note: The template is actually stored in the database being imported into (in this case National Parks). The template is only available when that database is being exported from. Each database may contain its own separate set of templates, which makes sense since the export configuration used with one database is not likely to work with any other database.)



The wizard loads the entire export configuration, ready to go. You can use the configuration as is or modify it before you actually import the data.

If you want to delete or rename a template choose the **Rename/Delete Templates...** command from the Template menu. To rename a template first click on it and then press the **Rename** button. A dialog appears allowing you to type in a new name. To delete a template press the **Delete** button.



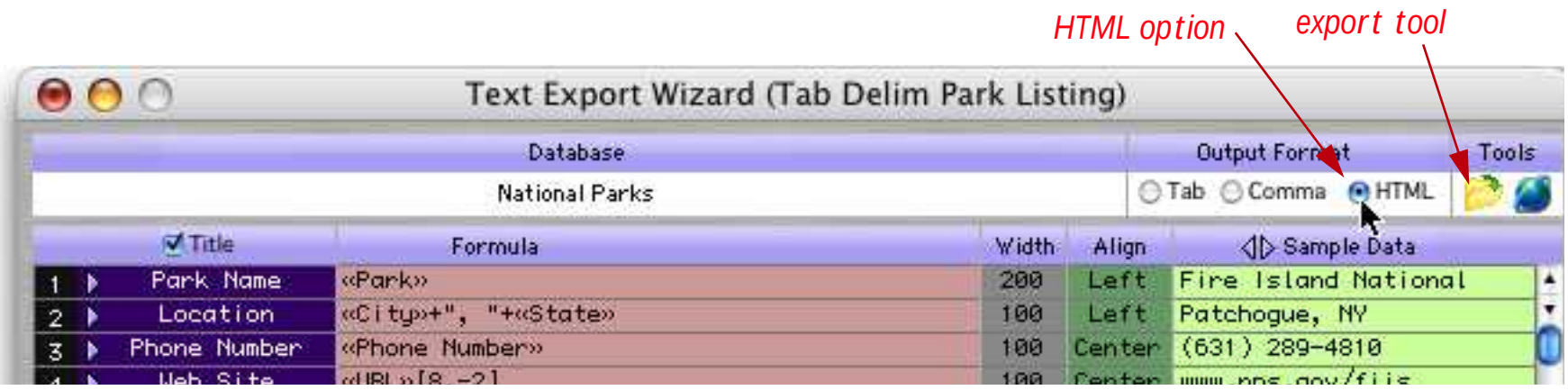
When you are done press the **OK** button.

Choosing a Database to Export From

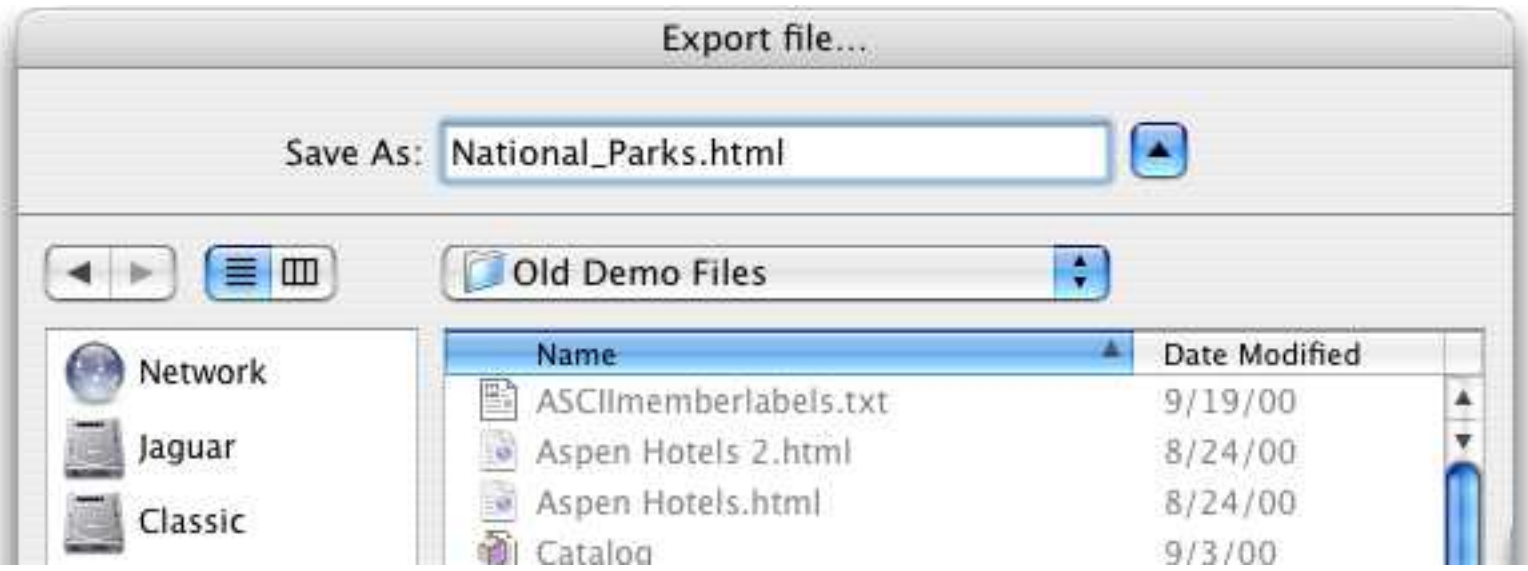
The **Text Import Wizard** normally exports from database that was active when the wizard was opened. However, you can use the **Database** menu to choose to export from any open database. Simply choose the database you want to export from and then set up the configuration.

Exporting HTML Tables

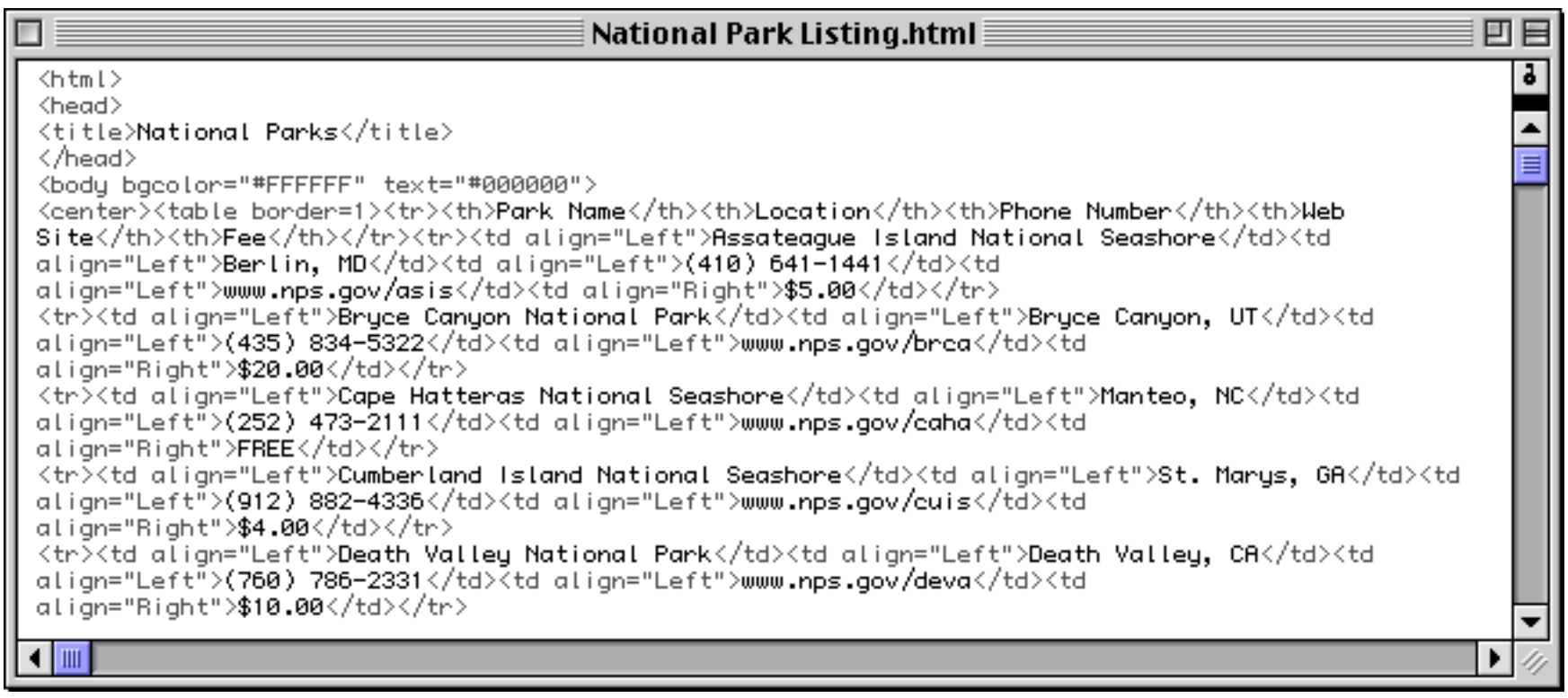
To export a database as an HTML table all you have to do is check the HTML option and choose **Export Text File...** from the Export menu (or click on the export icon in the Tools box.).



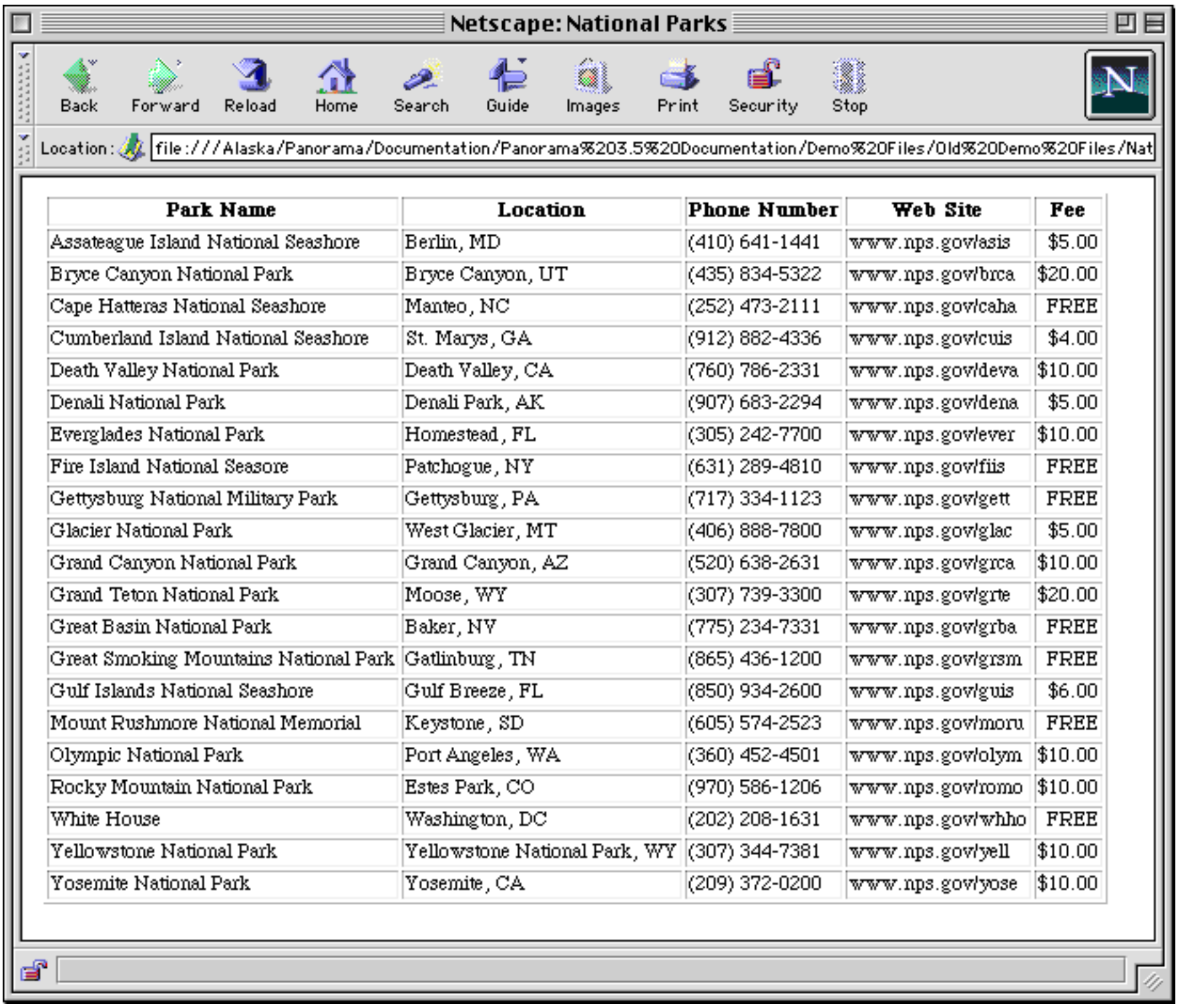
When the wizard prompts you for the name of the file to be exported, use .html as the suffix.



When viewed in a text editor you can see that the wizard has included all of the tags necessary to display a table in your web browser.



Here's what the same page looks like when previewed in a web browser.



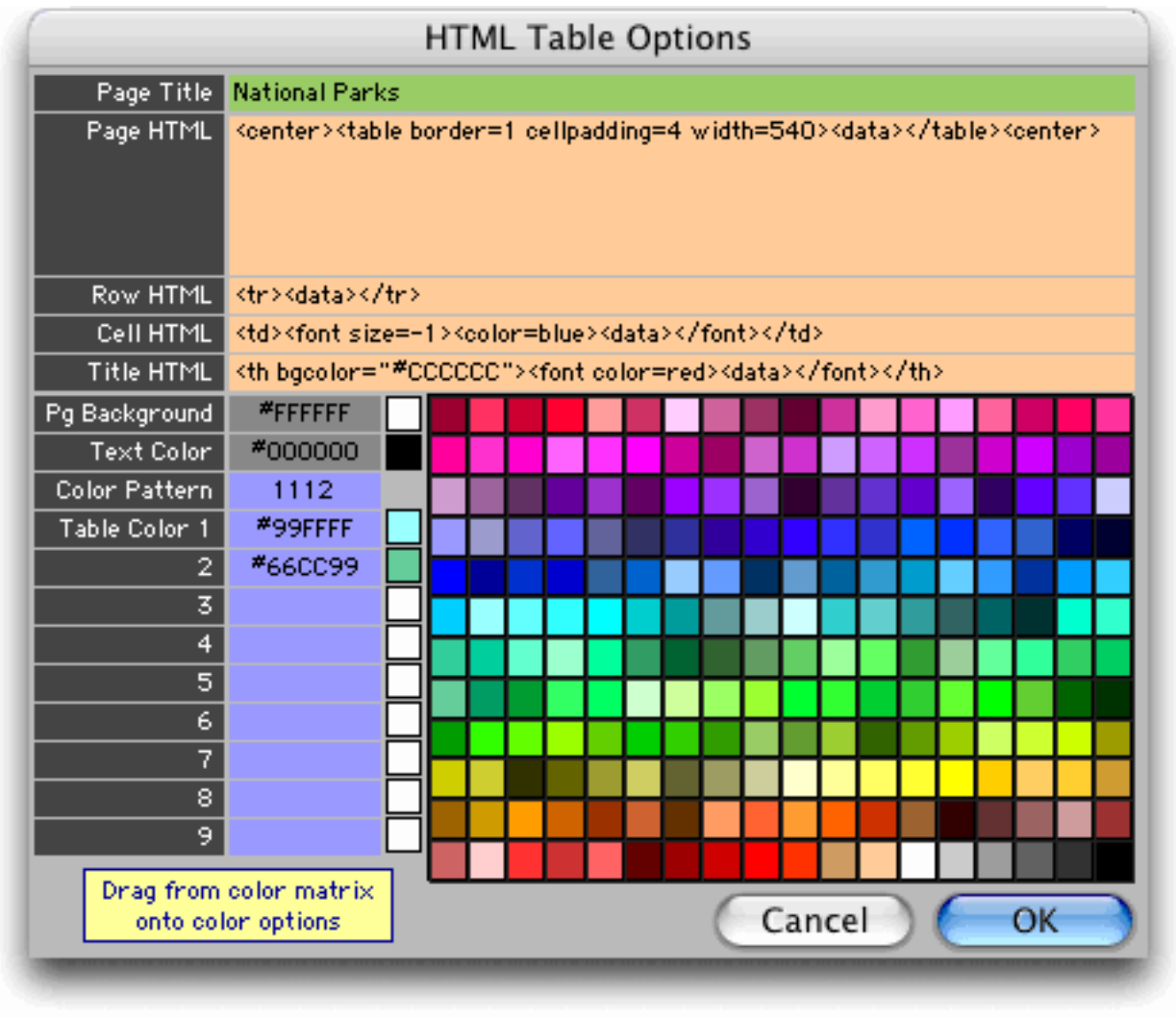
Park Name	Location	Phone Number	Web Site	Fee
Assateague Island National Seashore	Berlin, MD	(410) 641-1441	www.nps.gov/asis	$5.00
Bryce Canyon National Park	Bryce Canyon, UT	(435) 834-5322	www.nps.gov/brca	$20.00
Cape Hatteras National Seashore	Manteo, NC	(252) 473-2111	www.nps.gov/caha	FREE
Cumberland Island National Seashore	St. Marys, GA	(912) 882-4336	www.nps.gov/cuis	$4.00
Death Valley National Park	Death Valley, CA	(760) 786-2331	www.nps.gov/deva	$10.00
Denali National Park	Denali Park, AK	(907) 683-2294	www.nps.gov/dena	$5.00
Everglades National Park	Homestead, FL	(305) 242-7700	www.nps.gov/ever	$10.00
Fire Island National Seasore	Patchogue, NY	(631) 289-4810	www.nps.gov/fiis	FREE
Gettysburg National Military Park	Gettysburg, PA	(717) 334-1123	www.nps.gov/gett	FREE
Glacier National Park	West Glacier, MT	(406) 888-7800	www.nps.gov/glac	$5.00
Grand Canyon National Park	Grand Canyon, AZ	(520) 638-2631	www.nps.gov/grca	$10.00
Grand Teton National Park	Moose, WY	(307) 739-3300	www.nps.gov/grte	$20.00
Great Basin National Park	Baker, NV	(775) 234-7331	www.nps.gov/grba	FREE
Great Smoking Mountains National Park	Gatlinburg, TN	(865) 436-1200	www.nps.gov/grsm	FREE
Gulf Islands National Seashore	Gulf Breeze, FL	(850) 934-2600	www.nps.gov/guis	$6.00
Mount Rushmore National Memorial	Keystone, SD	(605) 574-2523	www.nps.gov/moru	FREE
Olympic National Park	Port Angeles, WA	(360) 452-4501	www.nps.gov/olym	$10.00
Rocky Mountain National Park	Estes Park, CO	(970) 586-1206	www.nps.gov/romo	$10.00
White House	Washington, DC	(202) 208-1631	www.nps.gov/whho	FREE
Yellowstone National Park	Yellowstone National Park, WY	(307) 344-7381	www.nps.gov/yell	$10.00
Yosemite National Park	Yosemite, CA	(209) 372-0200	www.nps.gov/yose	$10.00

Using the Generated HTML Page

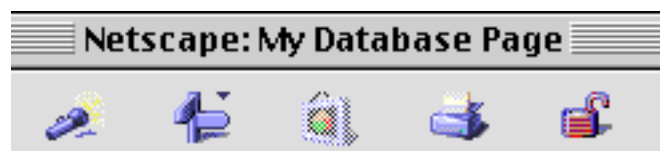
Just like any other HTML file created with a web authoring tool the file you have created may be copied onto a web server and used as part of a web site. The **Text Export Wizard** creates fixed "static" pages that represent a "snapshot" of the database at a certain point in time. If you want use a Panorama database to create "dynamic" web pages (for example searching a database on the web or allowing a database to be updated via the web) you must use our optional **PanSTAR** web server plug-in. Contact ProVUE Development for details about this plug-in.

HTML Table Options

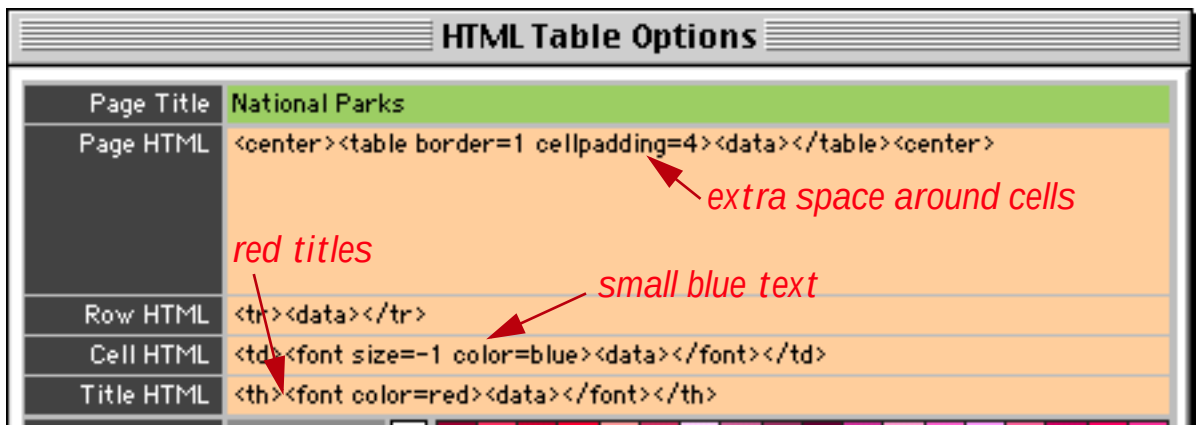
As you can see the wizard is capable of making a fairly nice table right out of the box. However if you have some basic knowledge of HTML you can also customize the generation of the table. Most of the customization options are accessible from the **HTML Table Options** dialog, which can be opened by selecting it from the Export menu or by double clicking on the HTML checkbox. The dialog allows you to specify about a dozen options for the generated HTML page.



The top third of the dialog controls how HTML is generated for each cell, row, and for the overall page.

Option	Examples	Description
Page Title	My Database Page	This is the title for the page. Whatever you type here will appear in the title bar of the browser, like this. 
Page HTML	<table border=1><data></table>	This is the template for the overall HTML of the body of the page (the wizard will create the header and footer automatically. At a minimum it should include the tags <table><data></table>. The wizard will automatically replace the <data> tag with the body of table generated from the database.
Row HTML	<tr valign=center><data></tr>	This is the template for each line in the table. At a minimum it should include the tags <tr><data></tr>. The wizard will automatically replace the <data> tag with the body of the line (which has been built up from individual cells, see the next entry).
Cell HTML	<td><data></td>	This is the template for each cell in the table. At a minimum it should include the tags <td><data></td>. The wizard will automatically replace the <data> tag with the data (an individual cell) from the database. As it does so the wizard will automatically prepare the data for HTML display, for example, converting non 7-bit characters to the appropriate HTML entity wherever possible.
Title HTML	<th><data></th>	This is the template for each title in the table. At a minimum it should include the tags <th><data></th> or <td><data></td>. The wizard will automatically replace the <data> tag with the column title as specified in the wizard.

Here is an example of customized table options. Consult an HTML reference to learn about the different tag options you can use when building a table.

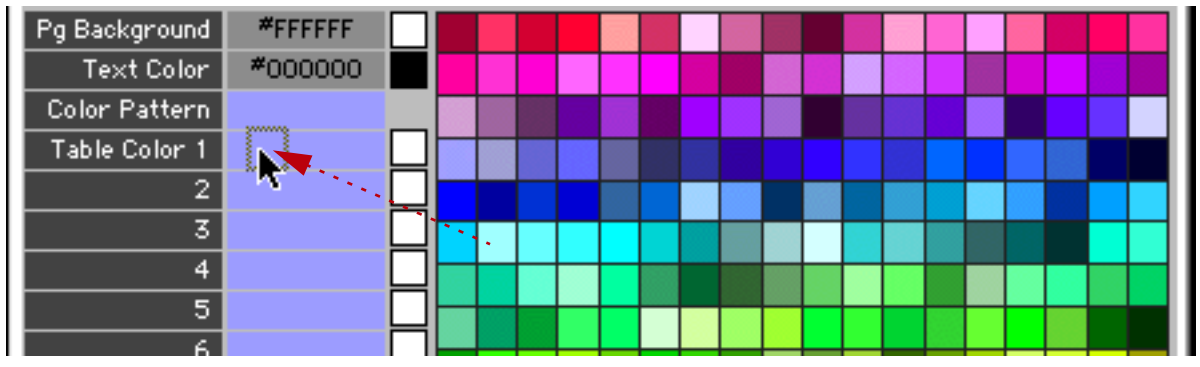


Here's what this table looks like when generated (use **Export Text File** in the File menu to generate the page).

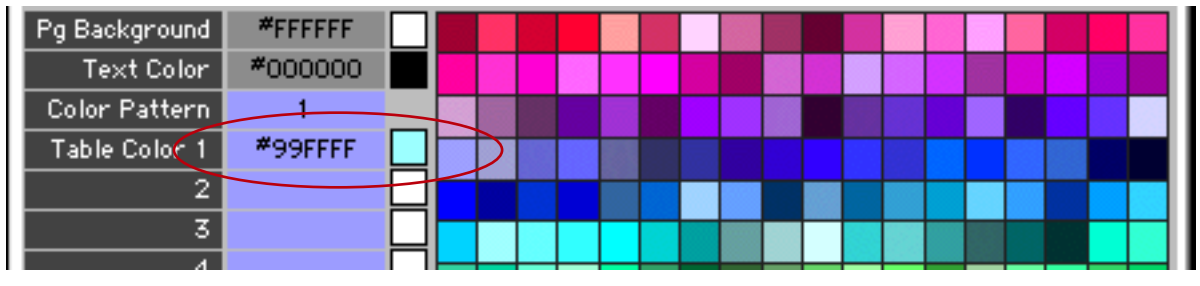


Park Name	Location	Phone Number	Web Site	Fee
Assateague Island National Seashore	Berlin, MD	(410) 641-1441	www.nps.gov/asis	$5.00
Bryce Canyon National Park	Bryce Canyon, UT	(435) 834-5322	www.nps.gov/brca	$20.00
Cape Hatteras National Seashore	Manteo, NC	(252) 473-2111	www.nps.gov/caha	FREE
Cumberland Island National Seashore	St. Marys, GA	(912) 882-4336	www.nps.gov/cuis	$4.00
Death Valley National Park	Death Valley, CA	(760) 786-2331	www.nps.gov/deva	$10.00
Denali National Park	Denali Park, AK	(907) 683-2294	www.nps.gov/dena	$5.00

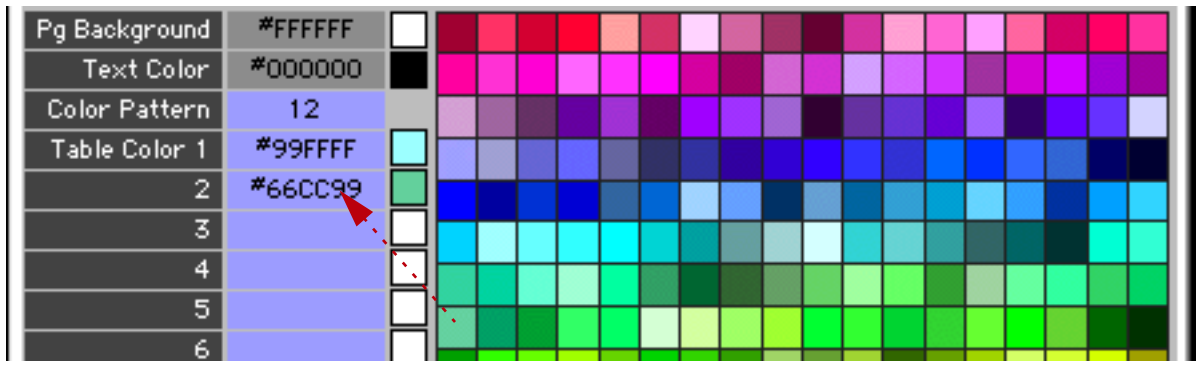
The bottom section of the dialog controls the colors used in the generated page. You can set a page background color, a default text color, and up to nine table background colors. When multiple table background colors are defined you can set up a table with an alternating pattern of background colors. To specify a color you can either type in a six character HTML color (for example #CC0033 or #6699FF) or you can drag a color from the matrix of “safe” colors on the right.



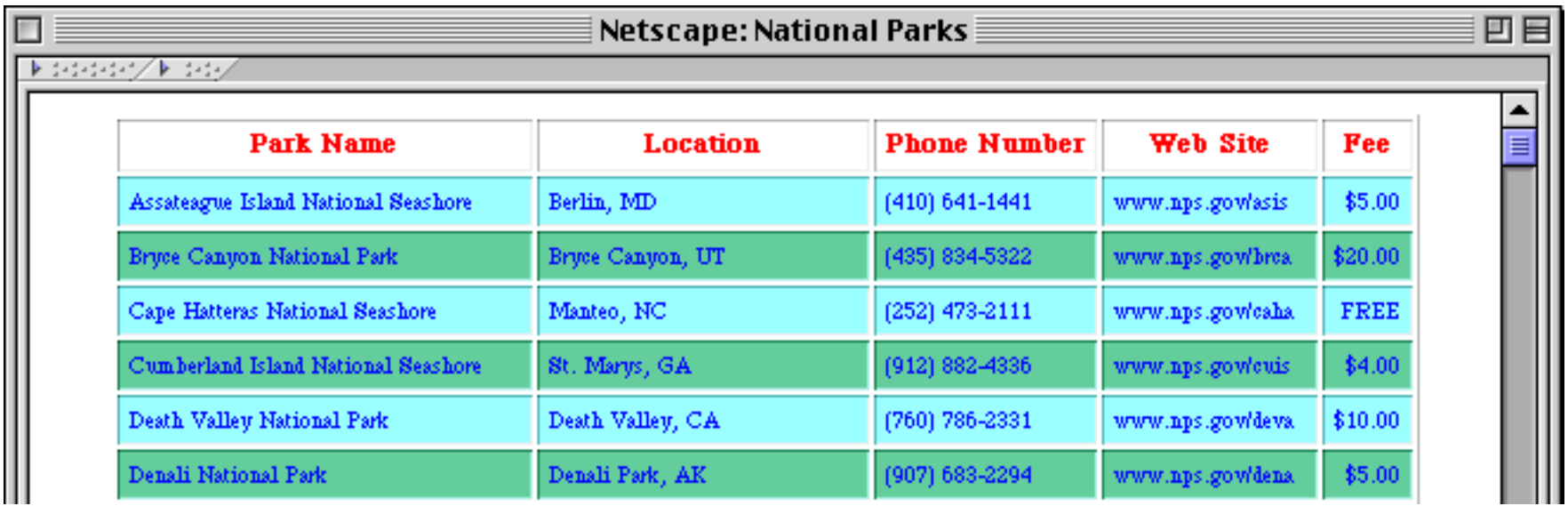
When you release the mouse the color appears in the box you have selected, along with a preview.



Repeat the process for each background color you want to use.

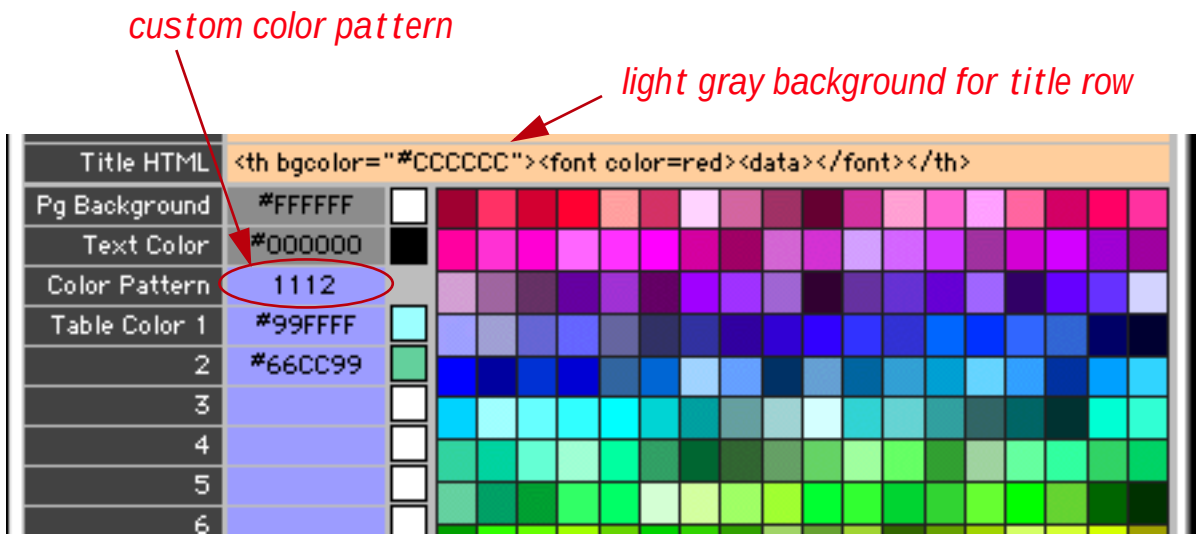


Here’s what this table looks like when generated (use **Export Text File** in the File menu to generate the page).



Park Name	Location	Phone Number	Web Site	Fee
Assateague Island National Seashore	Berlin, MD	(410) 641-1441	www.nps.gov/asis	$5.00
Bryce Canyon National Park	Bryce Canyon, UT	(435) 834-5322	www.nps.gov/brca	$20.00
Cape Hatteras National Seashore	Manteo, NC	(252) 473-2111	www.nps.gov/caha	FREE
Cumberland Island National Seashore	St. Marys, GA	(912) 882-4336	www.nps.gov/cuis	$4.00
Death Valley National Park	Death Valley, CA	(760) 786-2331	www.nps.gov/deva	$10.00
Denali National Park	Denali Park, AK	(907) 683-2294	www.nps.gov/dena	$5.00

By revising the Color Pattern you can change the pattern the wizard uses for alternating the background color. In the example below the pattern has been revised to use the first color three times and then switch to the second color for one line. We've also revised the Title HTML to make the title appear with a light gray background.

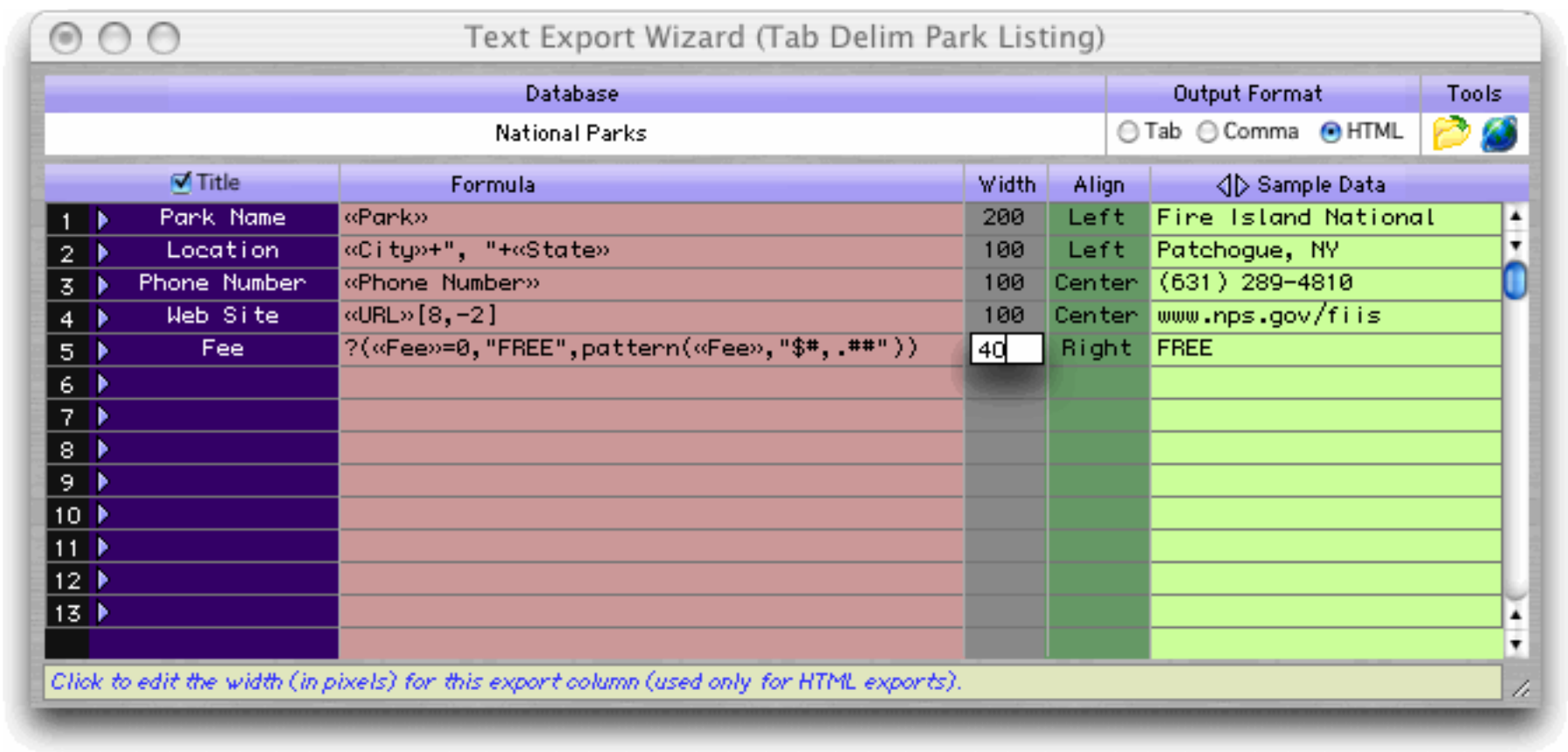


Here's what the revised table looks like in a browser.

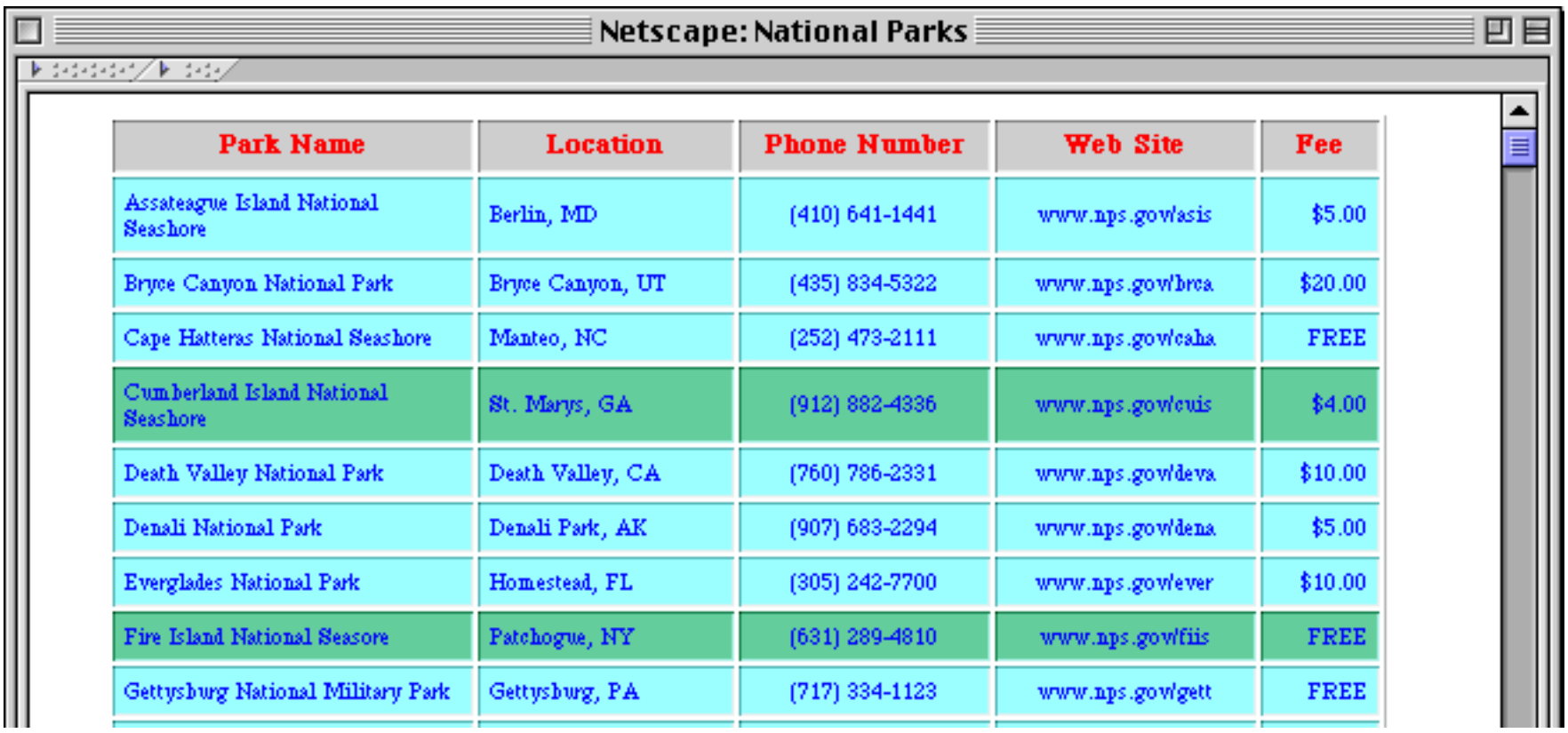


Park Name	Location	Phone Number	Web Site	Fee
Assateague Island National Seashore	Berlin, MD	(410) 641-1441	www.nps.gov/asis	$5.00
Bryce Canyon National Park	Bryce Canyon, UT	(435) 834-5322	www.nps.gov/brca	$20.00
Cape Hatteras National Seashore	Manteo, NC	(252) 473-2111	www.nps.gov/caha	FREE
Cumberland Island National Seashore	St. Marys, GA	(912) 882-4336	www.nps.gov/cuis	$4.00
Death Valley National Park	Death Valley, CA	(760) 786-2331	www.nps.gov/deva	$10.00
Denali National Park	Denali Park, AK	(907) 683-2294	www.nps.gov/dena	$5.00
Everglades National Park	Homestead, FL	(305) 242-7700	www.nps.gov/ever	$10.00
Fire Island National Seasore	Patchogue, NY	(631) 289-4810	www.nps.gov/fiis	FREE
Gettysburg National Military Park	Gettysburg, PA	(717) 334-1123	www.nps.gov/gett	FREE
Glacier National Park	West Glacier, MT	(406) 888-7800	www.nps.gov/glac	$5.00
Grand Canyon National Park	Grand Canyon, AZ	(520) 638-2631	www.nps.gov/grca	$10.00
Grand Teton National Park	Moose, WY	(307) 739-3300	www.nps.gov/grte	$20.00
Great Basin National Park	Baker, NV	(775) 234-7331	www.nps.gov/grba	FREE
Great Smoking Mountains National Park	Gatlinburg, TN	(865) 436-1200	www.nps.gov/grsm	FREE
Gulf Islands National Seashore	Gulf Breeze, FL	(850) 934-2600	www.nps.gov/guis	$6.00
Mount Rushmore National Memorial	Keystone, SD	(605) 574-2523	www.nps.gov/moru	FREE
Olympic National Park	Port Angeles, WA	(360) 452-4501	www.nps.gov/olym	$10.00
Rocky Mountain National Park	Estes Park, CO	(970) 586-1206	www.nps.gov/romo	$10.00
White House	Washington, DC	(202) 208-1631	www.nps.gov/whho	FREE
Yellowstone National Park	Yellowstone National Park, WY	(307) 344-7381	www.nps.gov/yell	$10.00

If you don't assign any column widths the browser will try to assign appropriate widths for you. If you don't like the widths the browser has picked you can specify the widths to be used. The widths are specified in pixels. Be forewarned, however, that sometimes a browser may sometimes ignore the widths you specify and go ahead and use whatever widths it wants!

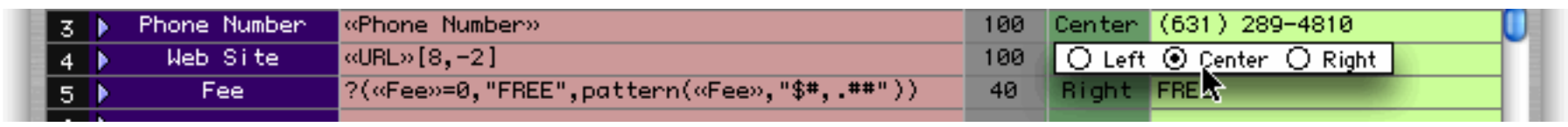


Here is the revised page as seen in the browser.



Park Name	Location	Phone Number	Web Site	Fee
Assateague Island National Seashore	Berlin, MD	(410) 641-1441	www.nps.gov/asis	$5.00
Bryce Canyon National Park	Bryce Canyon, UT	(435) 834-5322	www.nps.gov/brca	$20.00
Cape Hatteras National Seashore	Manteo, NC	(252) 473-2111	www.nps.gov/caha	FREE
Cumberland Island National Seashore	St. Marys, GA	(912) 882-4336	www.nps.gov/cuis	$4.00
Death Valley National Park	Death Valley, CA	(760) 786-2331	www.nps.gov/deva	$10.00
Denali National Park	Denali Park, AK	(907) 683-2294	www.nps.gov/dena	$5.00
Everglades National Park	Homestead, FL	(305) 242-7700	www.nps.gov/ever	$10.00
Fire Island National Seasore	Patchogue, NY	(631) 289-4810	www.nps.gov/fiis	FREE
Gettysburg National Military Park	Gettysburg, PA	(717) 334-1123	www.nps.gov/gett	FREE

You can also set the alignment of each column.



The width and alignment are only used when exporting HTML tables — they are ignored when exporting tab or comma delimited text.

You can customize the formula for a column to generate HTML directly, for example to insert a link tag.

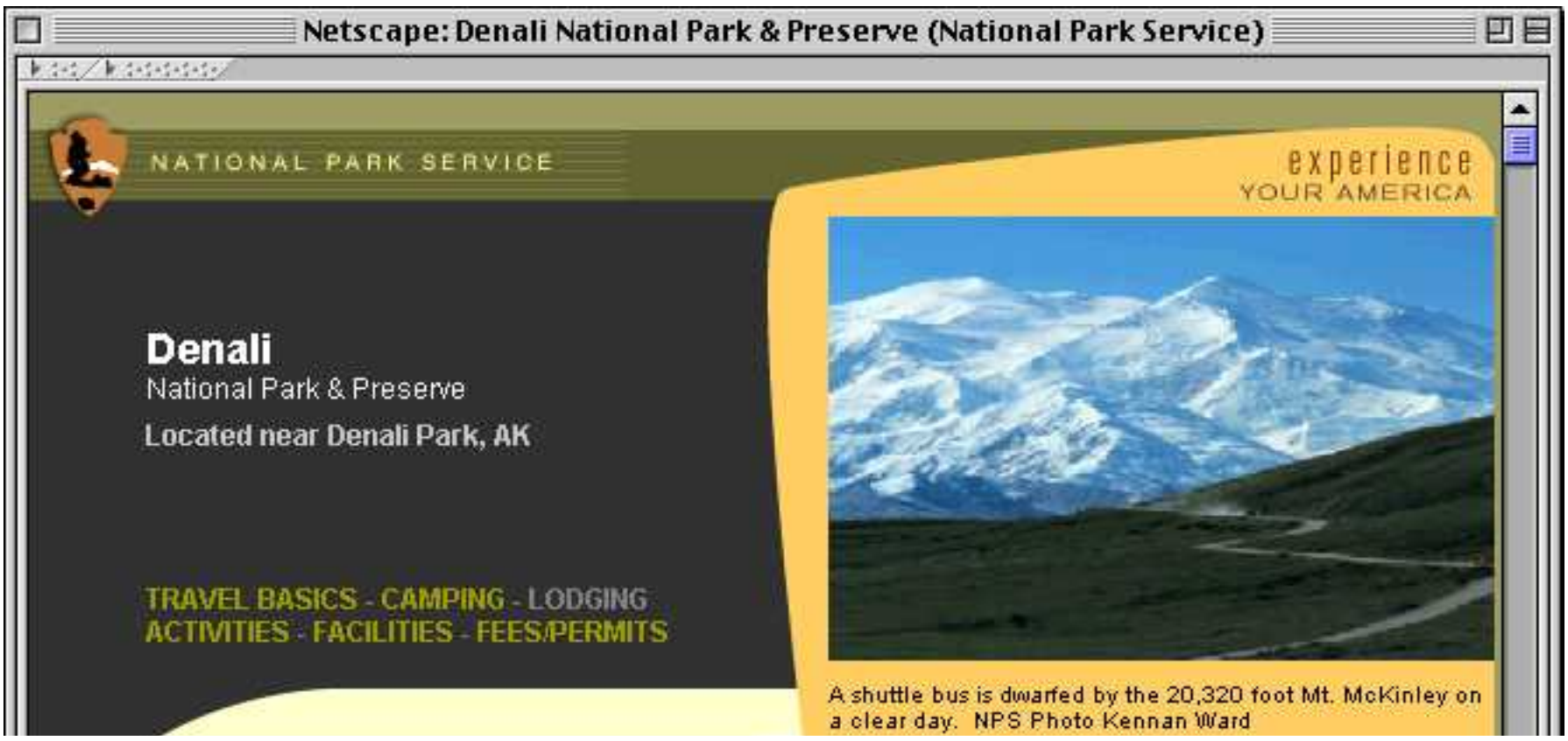
	Title	Formula	Width	Align	Sample Data
1	Park Name	«Park»	200	Left	Fire Island National
2	Location	«City»+", "+«State»	100	Left	Patchogue, NY
3	Phone Number	«Phone Number»	100	Center	(631) 289-4810
4	Web Site	""+«URL»[8,-2]+""	100	Center	www.nps.gov/fiis
5	Fee	?(«Fee»=0,"FREE",pattern(«Fee»,"$#,.##"))	40	Right	FREE
6					

The wizard will use the formula to generate links automatically.

Netscape: National Parks

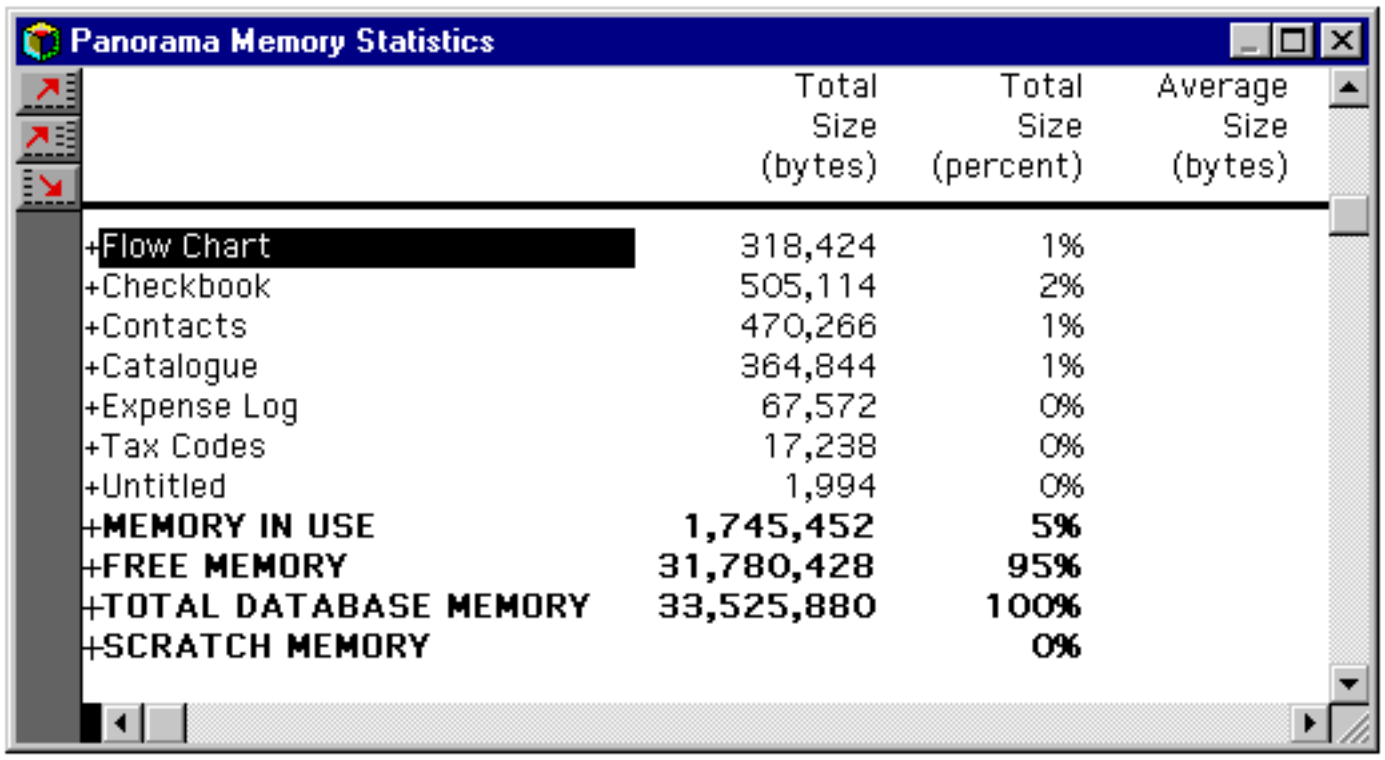
Park Name	Location	Phone Number	Web Site	Fee
Assateague Island National Seashore	Berlin, MD	(410) 641-1441	www.nps.gov/asis	$5.00
Bryce Canyon National Park	Bryce Canyon, UT	(435) 834-5322	www.nps.gov/brca	$20.00
Cape Hatteras National Seashore	Manteo, NC	(252) 473-2111	www.nps.gov/caha	FREE
Cumberland Island National Seashore	St. Marys, GA	(912) 882-4336	www.nps.gov/cuis	$4.00
Death Valley National Park	Death Valley, CA	(760) 786-2331	www.nps.gov/deva	$10.00
Denali National Park	Denali Park, AK	(907) 683-2294	www.nps.gov/dena	$5.00
Everglades National Park	Homestead, FL	(305) 242-7700	www.nps.gov/ever	$10.00

Just click on a link to jump to the corresponding web page!



Monitoring Memory Usage

Each database you open with Panorama is copied into the RAM memory of your computer. For most typical databases you'll have plenty of RAM available. You can use the **Memory Usage** command in the File Menu to see how much memory is in use and how much is available for expansion or for opening additional databases. This command opens a statistics window that displays the current memory usage of every open database, along with overall memory usage statistics.



At the top of the memory statistics data sheet, each open file is listed with the amount of memory used by that file (in bytes and as a percentage of the total memory). The **Memory In Use** line shows the total amount of memory used by all of the open databases (in this case just under 2 megabytes for 7 databases). The **Free Memory** line shows the amount of free memory available for database expansion or for loading more databases (in this case over 32 megabytes). The **Total Database Memory** line shows the total amount of memory available for Panorama databases, including the memory that is already in use.

At the bottom of the statistics window the **Scratch Memory** line shows the size of the scratch memory. The scratch memory is used on Macintosh OS 9 systems when Panorama has temporary memory needs for fonts, pictures, menus, and other system needs. Panorama does not use scratch memory on Windows or OS X systems, which is why that line is blank in the illustration above. On Macintosh OS 9 systems the Total Size (bytes) column shows the total amount of memory dedicated to scratch memory ("Changing Scratch Memory Size (Macintosh OS 9)" on page 192). The Total Size (percent) column shows the percentage of this scratch memory actually in use when the statistics window was opened. This is usually a very low percentage (under 10%) because scratch memory is mostly heavily used at other times.

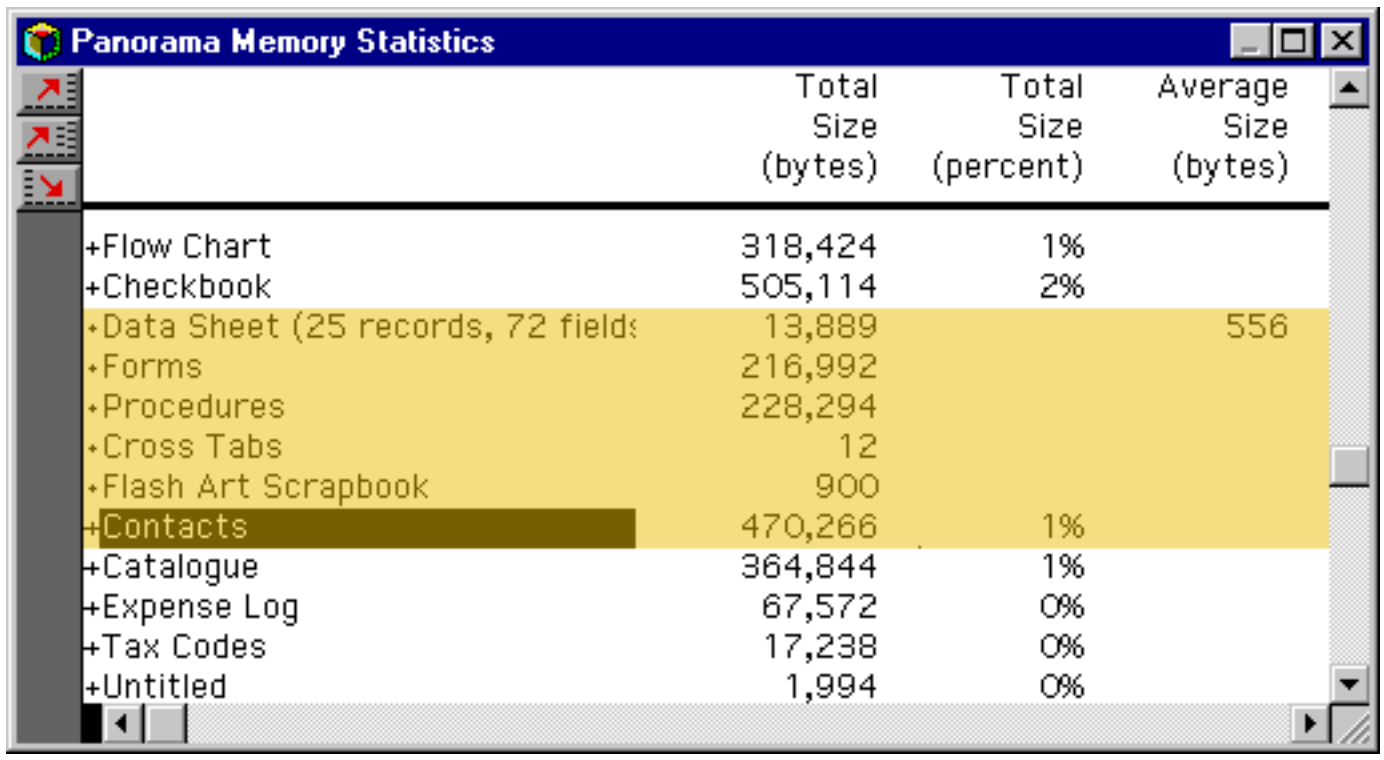
The memory statistics window has four columns. The first column shows the name of each memory area. The second column contains the actual number of bytes used by each area, while the third column displays each area's memory size as a percentage of the total database memory. The fourth column, Average Size, will be explained later.

When you are done with the memory statistics window, you can put it away by pressing the close box.

Memory Usage Details

The memory statistics window shown above doesn't show much detail—only the overall memory usage for each file. You can use Panorama's outline tools (see "Expanding and Collapsing Specific Details" on page 413) to expand to greater levels of detail. To see more detail about a particular file, click on that file and choose the **Expand** tool.

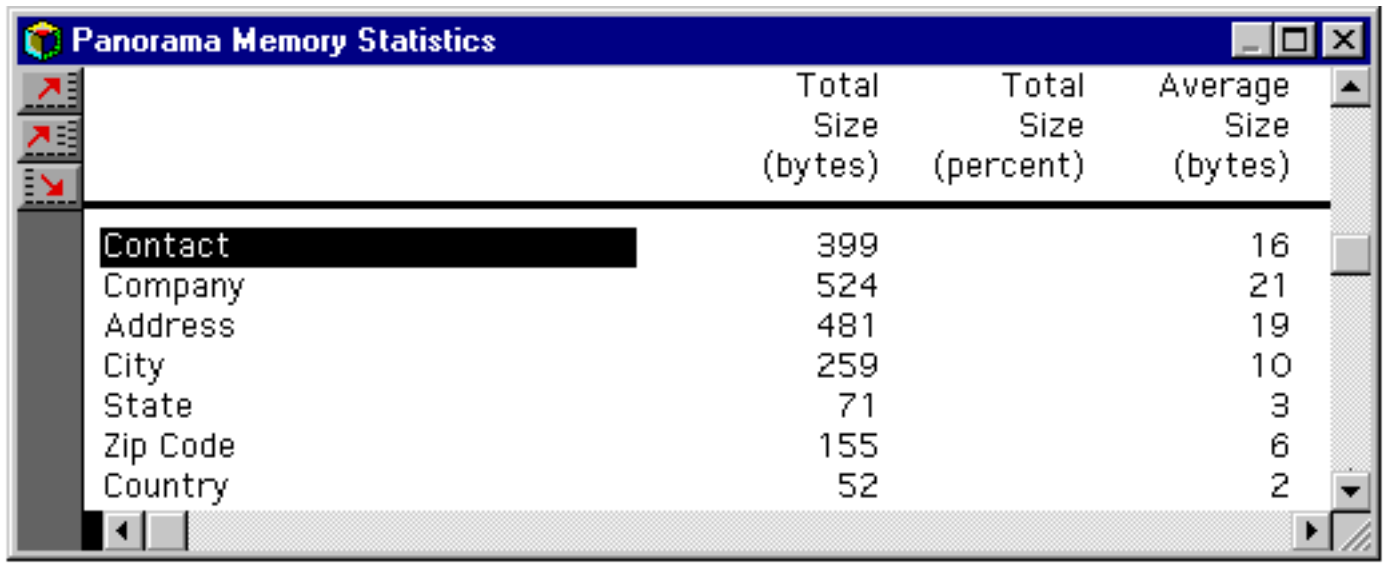
Click on the database you want to examine and press the  *Expand tool*



The database statistics for this file are now divided into five subdivisions: Data Sheet, Forms, Procedures, Cross Tabs, and Flash Art Scrapbook. (Note: The yellow highlight in the window above is for illustration purposes only, it does not appear in the real window.)

The Data Sheet portion of the file contains the actual data itself. This line shows the number of records in the database, the total amount of data occupied by the data itself, and the average character length of each record in the database. In this case the database contains 25 records that are an average of 556 bytes long.

The Data Sheet line can be expanded further to show the amount of memory used by each field in the database, as shown below.



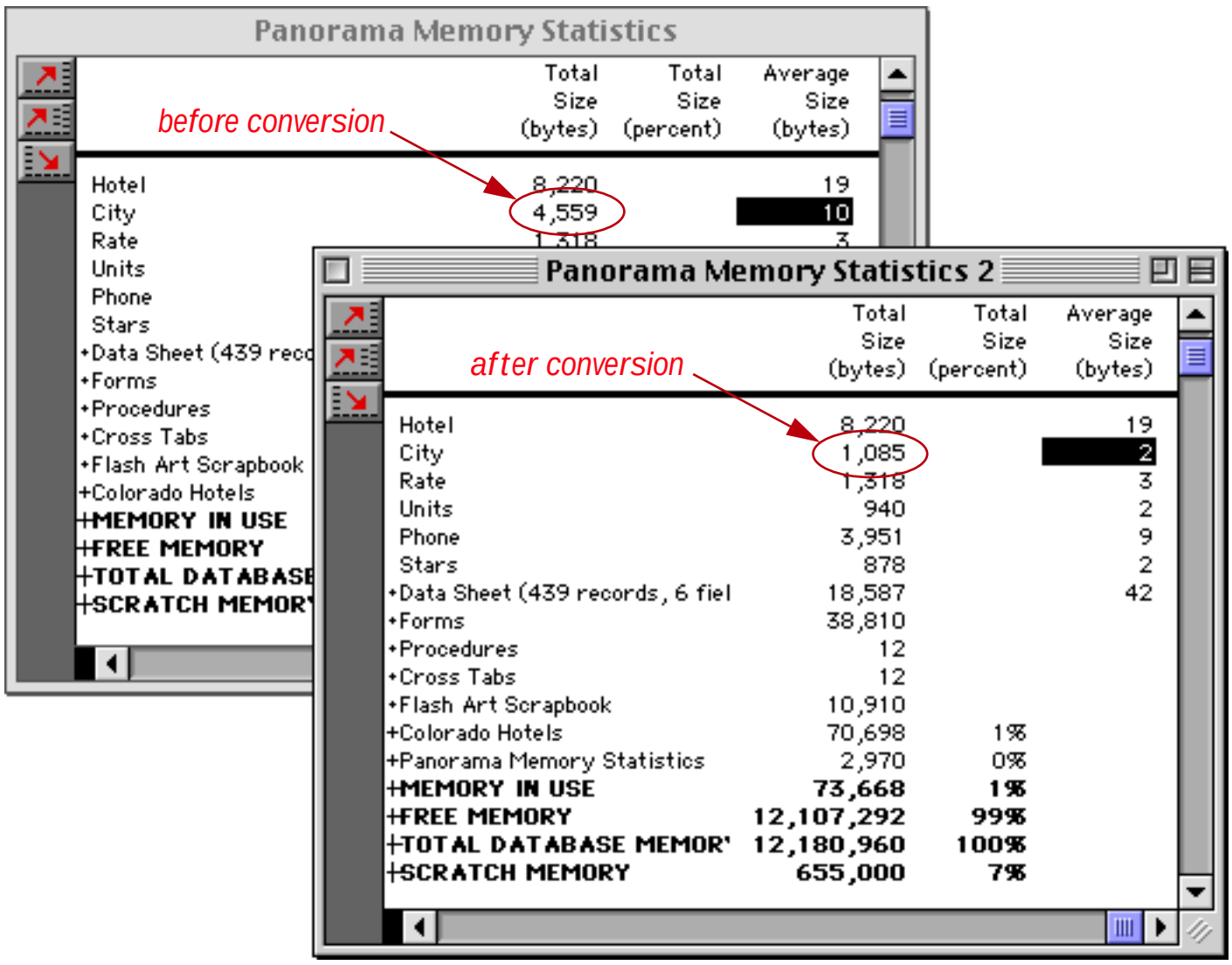
Each line shows the total amount of memory used by one field of the database, and the average size of one cell. For example, the Contact field (Name) is an average of 16 characters long, while the Company field is an average of 21 characters long.

The Forms, Procedures, and Cross Tabs lines can be expanded to show the actual amount of memory used by each individual form, procedure, and crosstab within the database. The Flash Art Scrapbook line can be expanded to show the size of each picture in the Flash Art Scrapbook. You usually won't need this kind of detail, but it can be useful if a file seems unusually large and you need to find out why.

Multiple Memory Statistic Windows

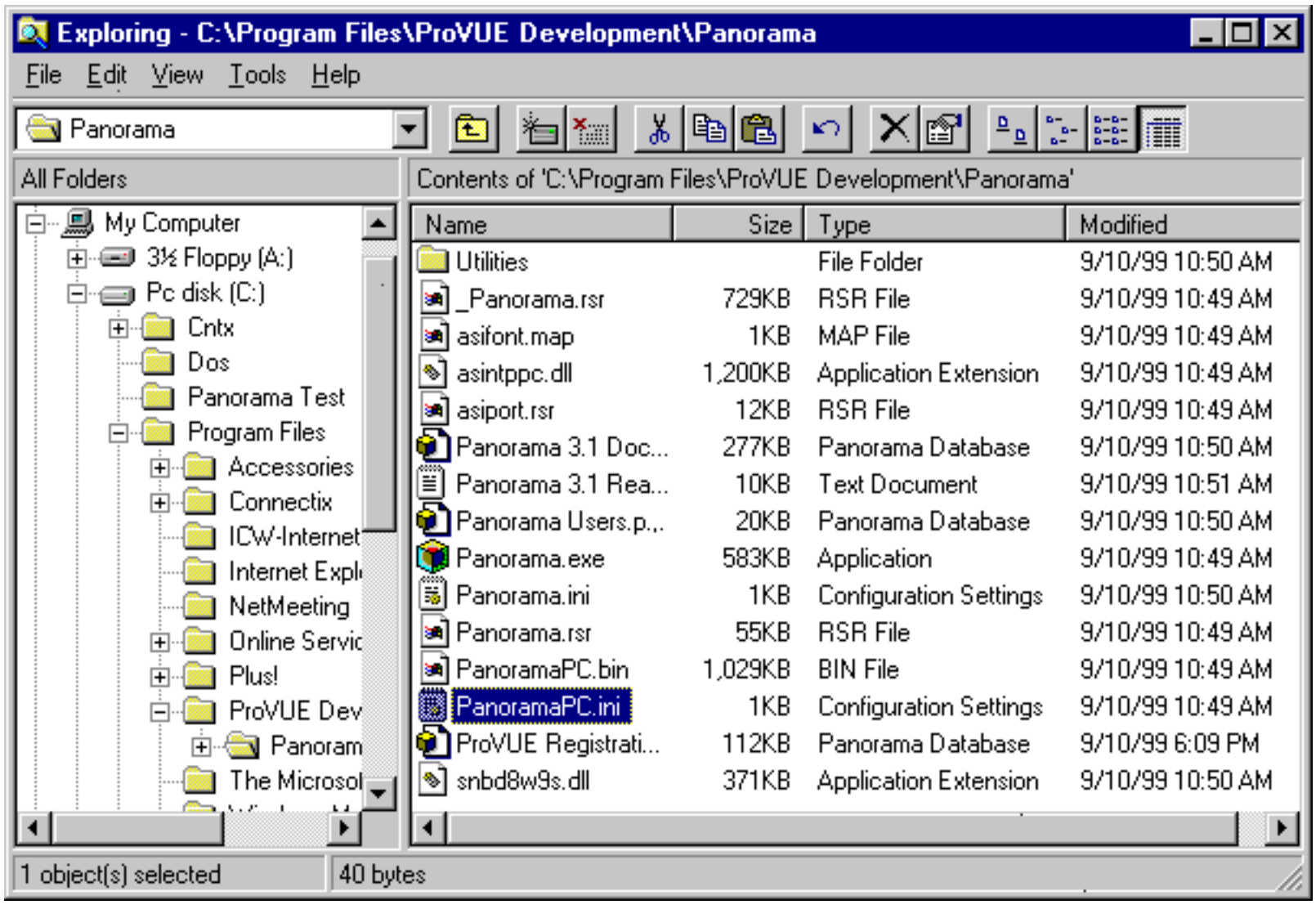
The memory statistics window displays a snapshot of Panorama's memory usage at a single point in time. The window will not be updated as memory usage changes. However, it is possible to take another snapshot to see how usage has changed. In fact, you can open over a dozen memory usage snapshot windows on the screen at one time, so that they can easily be compared with each other.

As an example, suppose you wanted to see how much memory is saved by converting the City field in a mailing list database from text to choice data type. (Using the choice type will save memory by storing the actual name only once, no matter how many times it is used in the database.) Start by taking a **Memory Usage** snapshot to see the original size of the City field. Then use the design sheet to convert the field from text to choice. Use the **Automatic Choices** command to create the list of cities (See "Generating a List of Choices Automatically" on page 300). Finally, choose **Memory Usage** from the File Menu again to take another memory snapshot. You can compare the two memory snapshots to see the actual effect of the change (in this example, 3,474 bytes saved).

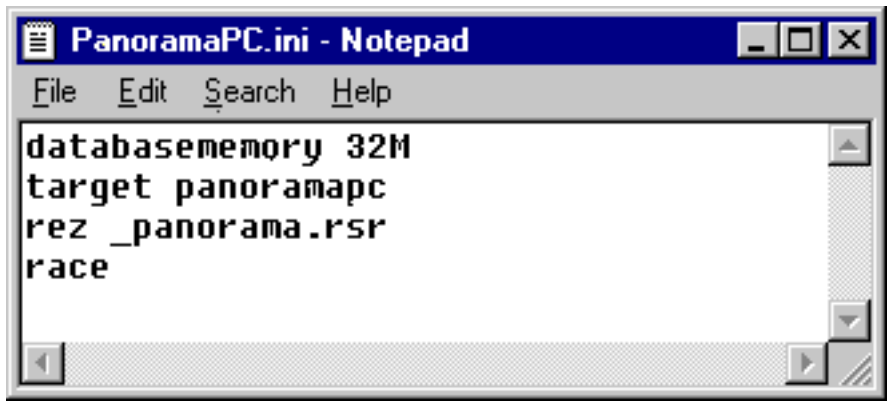


Adjusting Panorama's Memory Allocation (Windows and OS X)

As shipped from the factory, Panorama normally allocates 32 megabytes of memory for databases. For most applications this is more than enough. However, if you wish to use extremely large databases you may need to increase this allocation. To do this, locate the `PanoramaPC.ini` (WIndows) or the `PanoramaPowerPC.ini` (Mac OS X) file. As shown in the illustration below, this file is normally located in the `C:\Program Files\ProVUE Development\Panorama` folder on Windows systems. On Mac OS X systems it is normally in the `Applications:Panorama` folder. In either case it is always in the same folder as the Panorama application itself.



Double click this file to open it in the Notepad or TextEdit. (Note: Be sure you open `PanoramaPC.ini` (Windows) or `PanoramaPowerPC.ini` (OS X) and not `Panorama.ini`. If you open the wrong file, don't worry, just close it and open the correct file.) The file should look something like this:

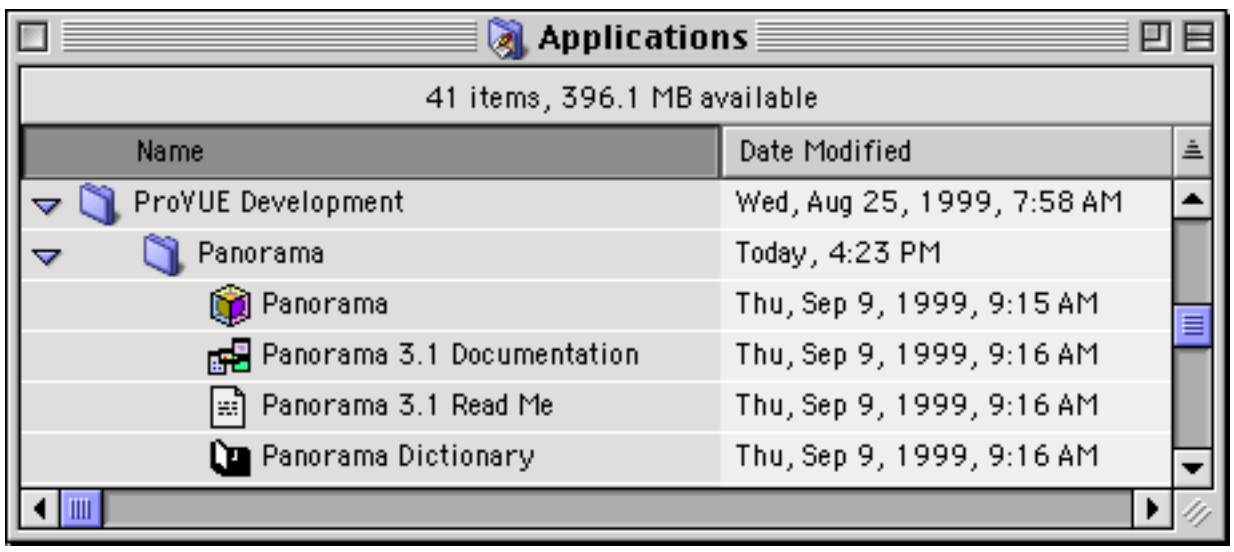


The **databasememory** line controls the amount of memory allocated by Panorama for databases. You may set this to any value from 3M to 999M. (Don't forget the M!). However, if you set this value to larger than the physical amount of memory available on your computer, you may reduce the amount of virtual memory available for other applications. We do not recommend opening databases that are larger than the physical memory size of your computer. Panorama will open the file and operate correctly, but it's performance may be severely degraded. Once you have set the new value save and close the window, then relaunch Panorama if necessary.

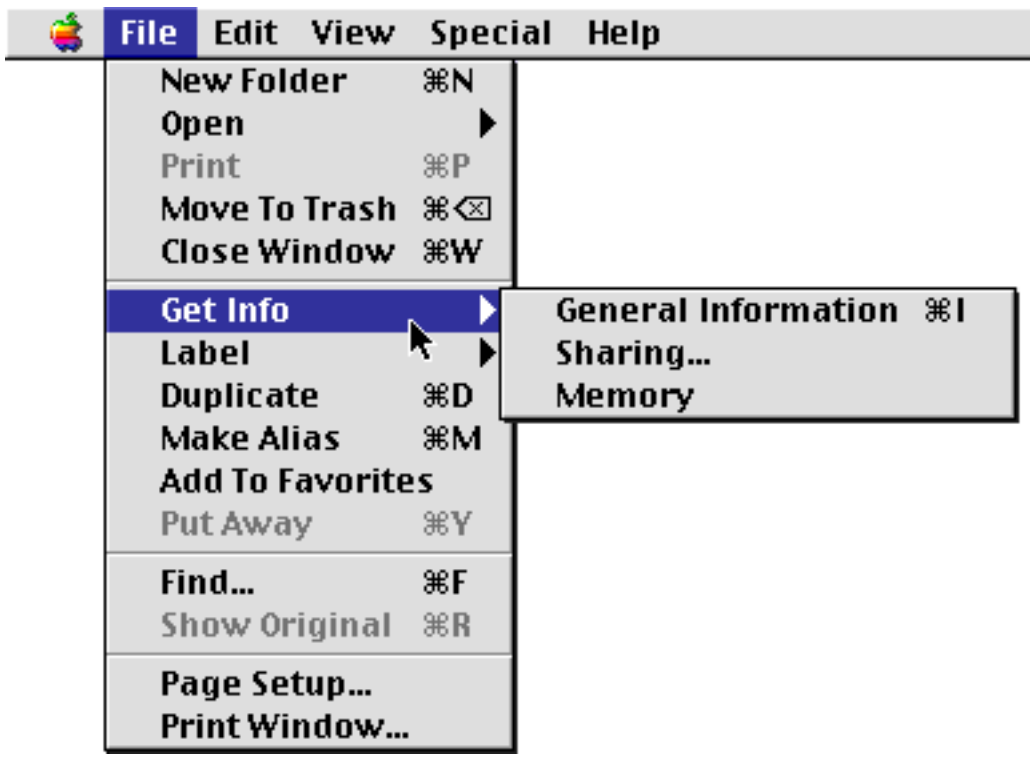
Adjusting Panorama's Memory Allocation (Macintosh OS 9)

As shipped from the factory, Panorama normally allocates 15 megabytes of memory on Macintosh computers. Not all of this memory is available for database use—about 2.5 megabytes are used by Panorama itself, plus another half megabyte for scratch memory (described below). That leaves about 12 megabytes for databases. This number may be increased (to allow you to open larger databases) or reduced (which allows other programs more memory to operate, but reduces the size of databases that you can work with).

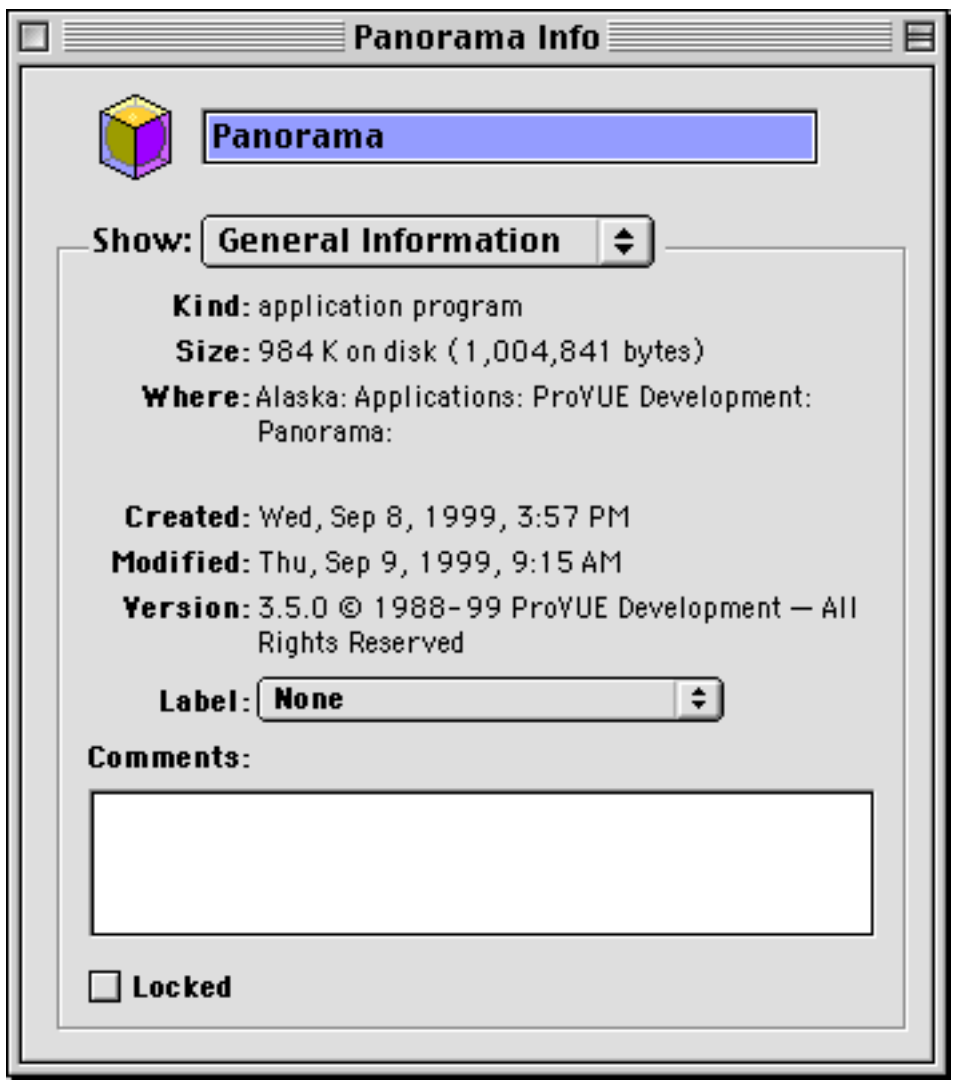
To adjust Panorama's memory allocation, locate the Panorama application. If you haven't moved it, you should be able to find it inside the `Applications` folder, as shown below. Make sure you have located the application itself, and not an alias to the application.



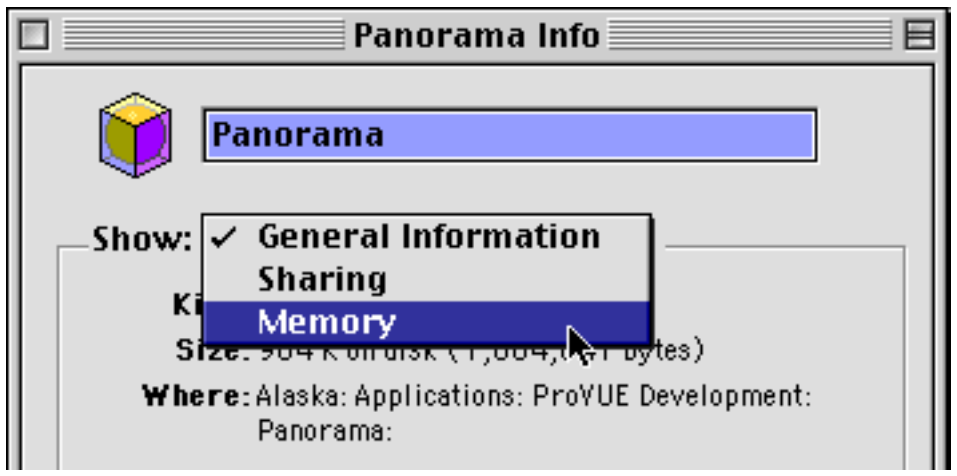
Click **once** on the Panorama icon, then choose **Get Info** from the File menu.



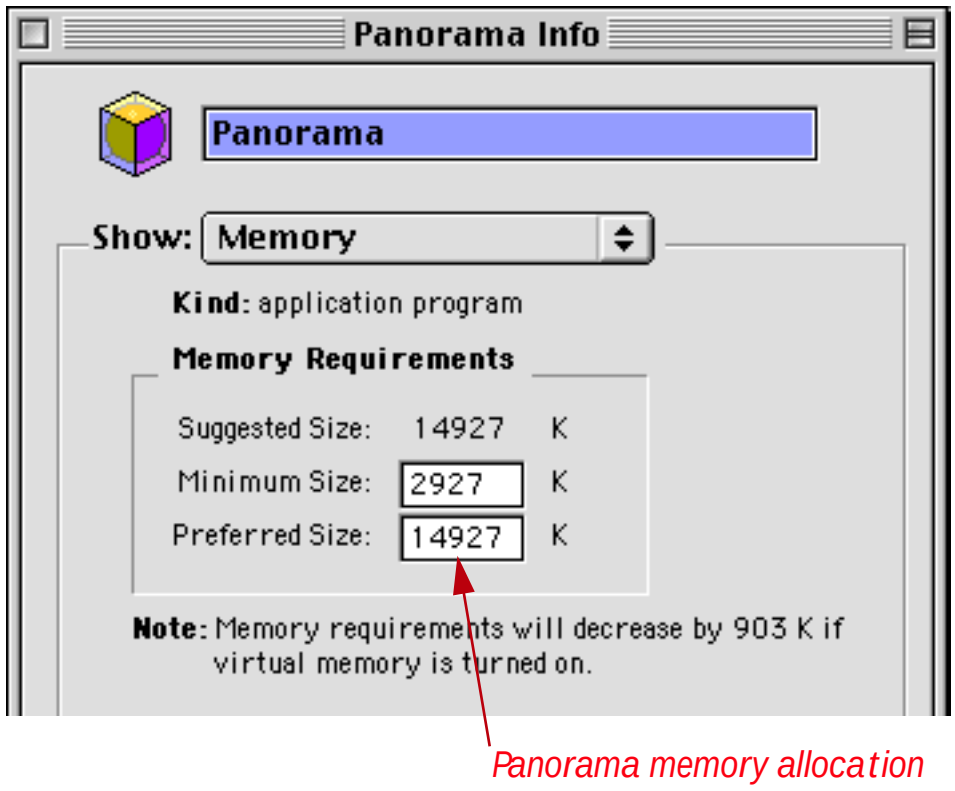
You should see a window that looks like this:



If the Kind is not application program, you don't have the right file. Go back and find the correct file. If you've got the correct file, click on the pop-up menu to select the **Memory** panel.



Now you can set the new Preferred Size for Panorama. Increase it to allow larger databases, or decrease it if you are running short of memory for other applications.



When you have changed the value simply close the window and re-launch Panorama.

Changing Scratch Memory Size (Macintosh OS 9)

Panorama normally reserves 650,000 bytes (650K) for **scratch memory**. Panorama uses scratch memory as temporary memory for fonts, pictures, sorting, printing, desk accessories, etc. Usually 650k of scratch memory is sufficient, but some applications require more scratch memory. Remember, however, that increasing the scratch memory decreases the amount of memory available for working with databases.

Color Pictures—If your database uses medium to large color pictures, the scratch memory will have to be expanded. A full size 8 bit color picture can be up to 300k. Scratch memory should be at least 100k larger than the largest picture you want to draw. If you are using a compressed image format like JPEG you'll need even more memory - enough for an uncompressed copy of the image (up to 8 times larger than the disk file).

Large Variables—Panorama's programming language uses scratch memory to hold variables. If you want to use extremely large variable values, you may need to increase the scratch memory allocation.

Unusual Printers—Some unusual printers (like the GCC laser printer and the HP DeskJet) may require more scratch memory.

To change the amount of scratch memory, hold down the **Shift** key and choose **Memory Usage** from the File Menu. Panorama will display the current scratch memory size, and allow you to enter a new size. Enter the new size and press **Ok**.



When you press **Ok**, Panorama will attempt to change the scratch memory immediately. Sometimes this may not be possible. If the scratch memory cannot be changed immediately, Panorama will ask you to quit and re-open Panorama. You do not need to reboot the entire machine. Simply re-opening Panorama will be sufficient. You may also want to increase Panorama's overall memory allocation (See "Adjusting Panorama's Memory Allocation (Macintosh OS 9)" on page 190).

It's also possible to set up a procedure to automatically modify the scratch memory allocation. See "Changing the Scratch Memory Allocation" on page 1571 to learn how to do this.

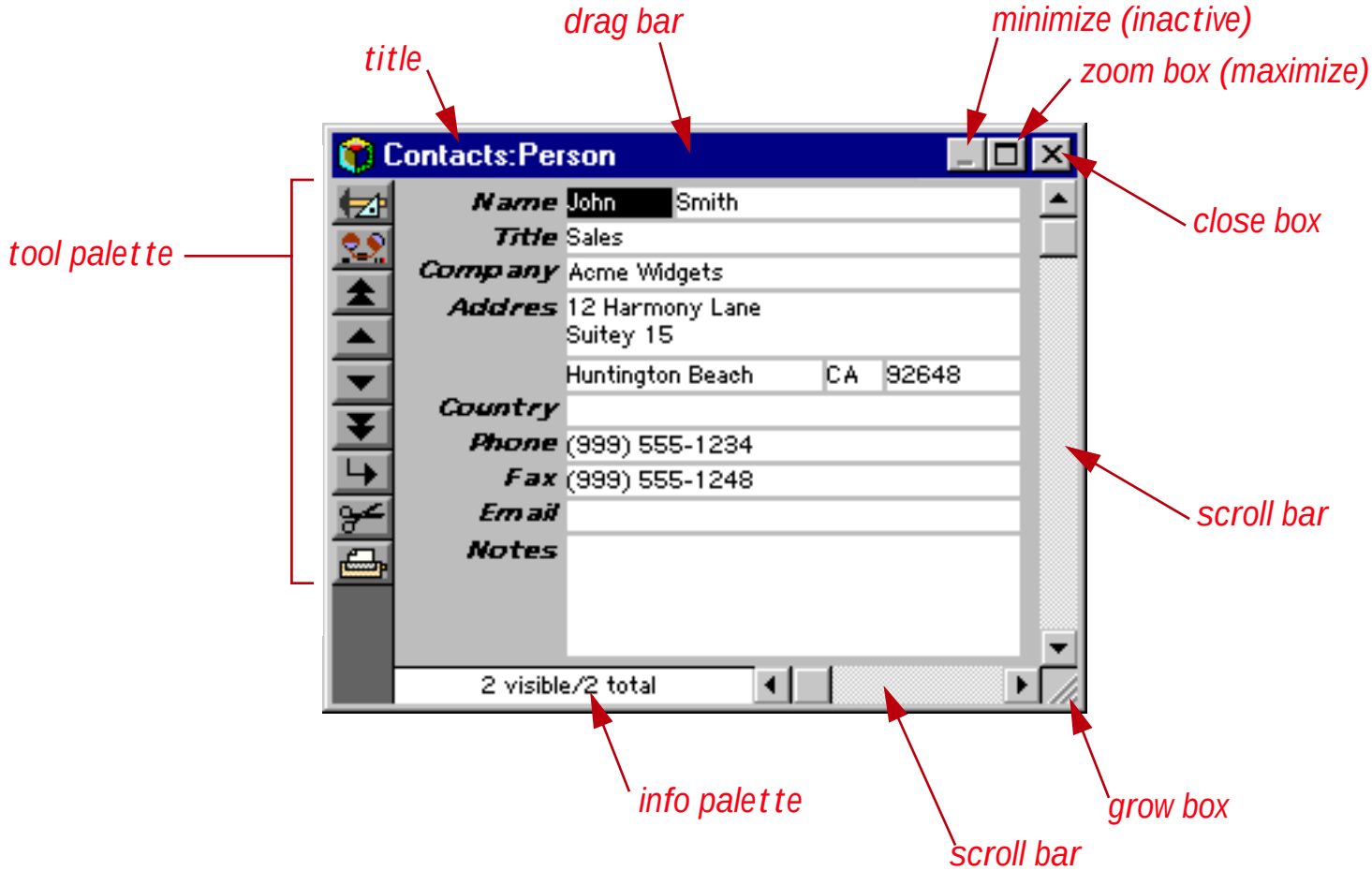
Chapter 2: Windows



As you use Panorama most of the action takes place inside windows. Panorama uses standard windows with a few unique touches, including a tool palette with pop-up help on the left side of each window. Panorama also has an enhanced zoom box that lets you zoom a window to a specific position. This chapter covers both the standard and enhanced window features.

Window Components

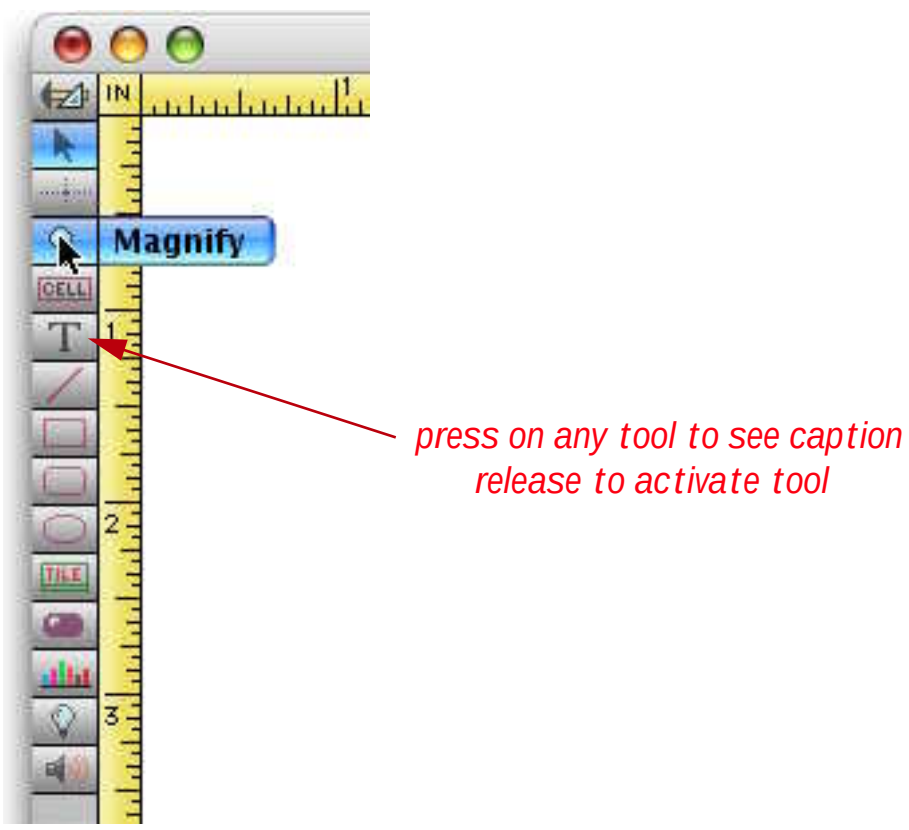
Each Panorama window has about a dozen components for configuring and controlling the window. Each of these components is activated by clicking or dragging with the mouse. Here's what a typical Panorama window looks like on a PC system.



On the Macintosh windows look very similar, except that the close box is on the left and the title is centered.

Tool Palette

The left side of each window contains a tool palette. To help you learn and remember the function of each tool, the tool palette displays a pop-up caption whenever you click on a tool. The caption will remain visible as long as you hold the mouse down over the tool.



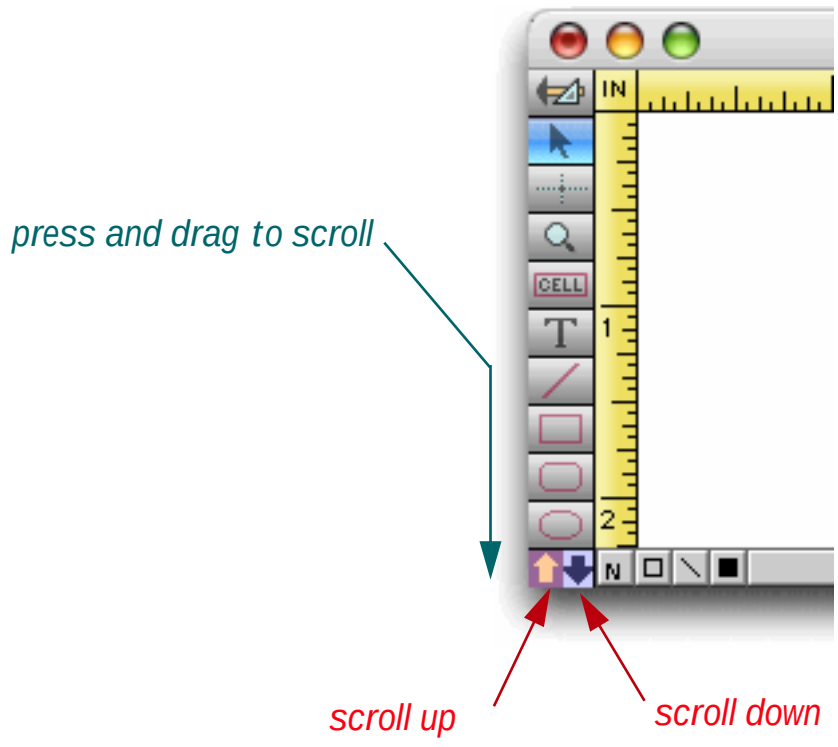
Unlike most tool palettes that perform a function when you click on a tool, Panorama's tool palette doesn't perform the function until you release the mouse. This allows you to look at the caption before you activate the tool. (Of course, you don't have to hold down the mouse and look at the caption. Once you have memorized the tool icons you can simply click on the tool you want—just like your favorite graphics or page layout program.)

You can scan through the tool captions by dragging the mouse up or down the palette (just as you would scan across the menu bar). The caption for each tool pops up as the mouse is dragged across it. You can stop at any time and release the mouse to activate a tool.

Remember, to activate a tool you must release the mouse over the tool itself—not over the pop-up caption. The pop-up caption is not a menu.

Scrolling the Tool Palette

What if all the tools don't fit in the window? You can scroll the tools by pressing on any tool and then dragging to the top or bottom of the palette. When you reach the edge of the palette the tools will scroll. You can also scroll the tools by clicking on one of the small arrows at the bottom of the palette. Each click scrolls by one tool.



If you have a scroll wheel mouse you can also scroll the tool palette by moving the mouse over the palette and turning the scroll wheel (OS X only).

Close Box

Clicking on the close box makes the window go away. If you hold down the **option** key (Mac) or **alt** key (PC) while you click the close box, Panorama will close the current file along with all of its windows.

Drag Bar

The drag bar allows you to move the window to a new position. When you press the drag bar a dotted outline of the window appears. Drag the outline to the new position, then release the mouse. Moving the window does not change the contents of the window. (You can also move the window by zooming into a spot—see below.)

Title

The window title displays the name of the database and information about the view being displayed in the window (form name, magnification, etc.).

Zoom Box (Maximize)

Clicking on the zoom (maximize) box allows you to quickly zoom the window to cover the entire screen. Clicking the zoom box again pops the window back into its original position. The zoom box is very handy when you want to temporarily concentrate on a specific window.

The zoom box can also be used to move a window to a specific spot on the screen. See "Zooming Into a Box" on page 201.

Grow Box

The grow box adjusts the size of the window. When you press on the grow box a dotted outline of the window appears. Drag the bottom right hand corner of this outline and then release to set the new window size. (You can also set the size of a window by zooming into a spot—see above.) On PC systems you can also adjust the size of a window by dragging any side of the window.

Scroll Bars

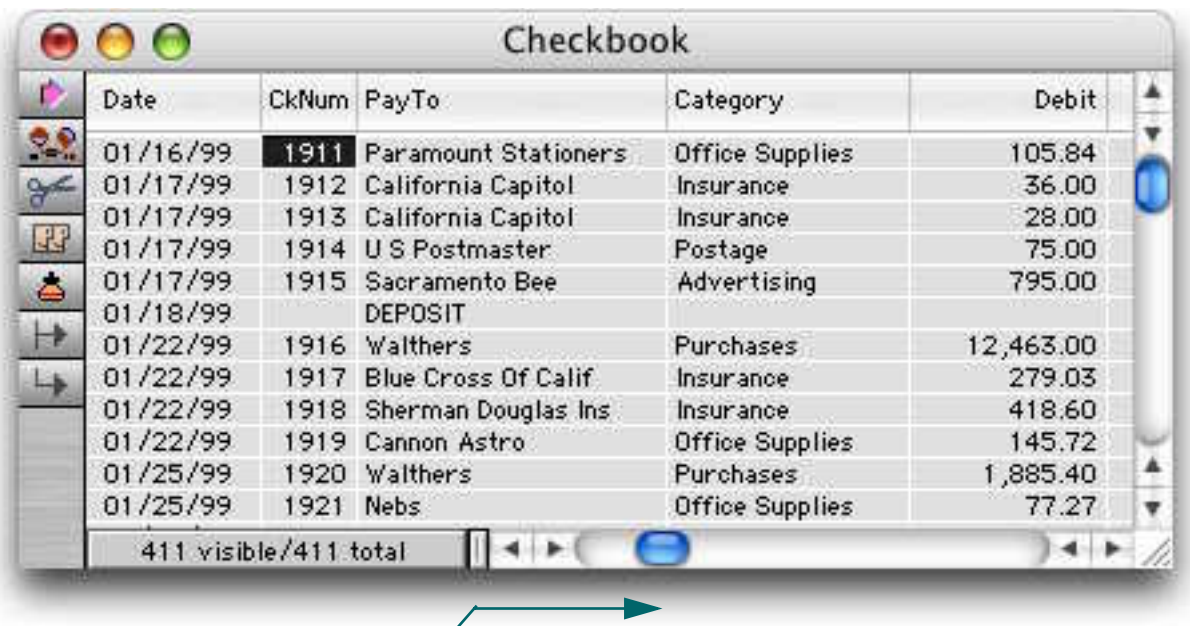
The scroll bars are used to shift the information or graphics displayed in the window. The vertical scroll bar (on the right edge of the window) shifts the display up and down, while the horizontal scroll bar (on the bottom edge of the window) shifts the display left and right. The sliding box inside each scroll bar shows the current position of the display.

If you are using OS X and a mouse with a scroll wheel, Panorama supports the scroll wheel. This wheel normally causes the window to scroll vertically when turned. If the **Shift** key is held down, the wheel will scroll the window horizontally.

When using OS X or OS 9, the **Option** key changes the way the scroll bars work. If you hold down the **Option** key and click on the up or down arrows, the window will scroll up or down an entire page. If you hold down the **Option** key and click in the main body of the scroll bar, the window will scroll directly to that spot.

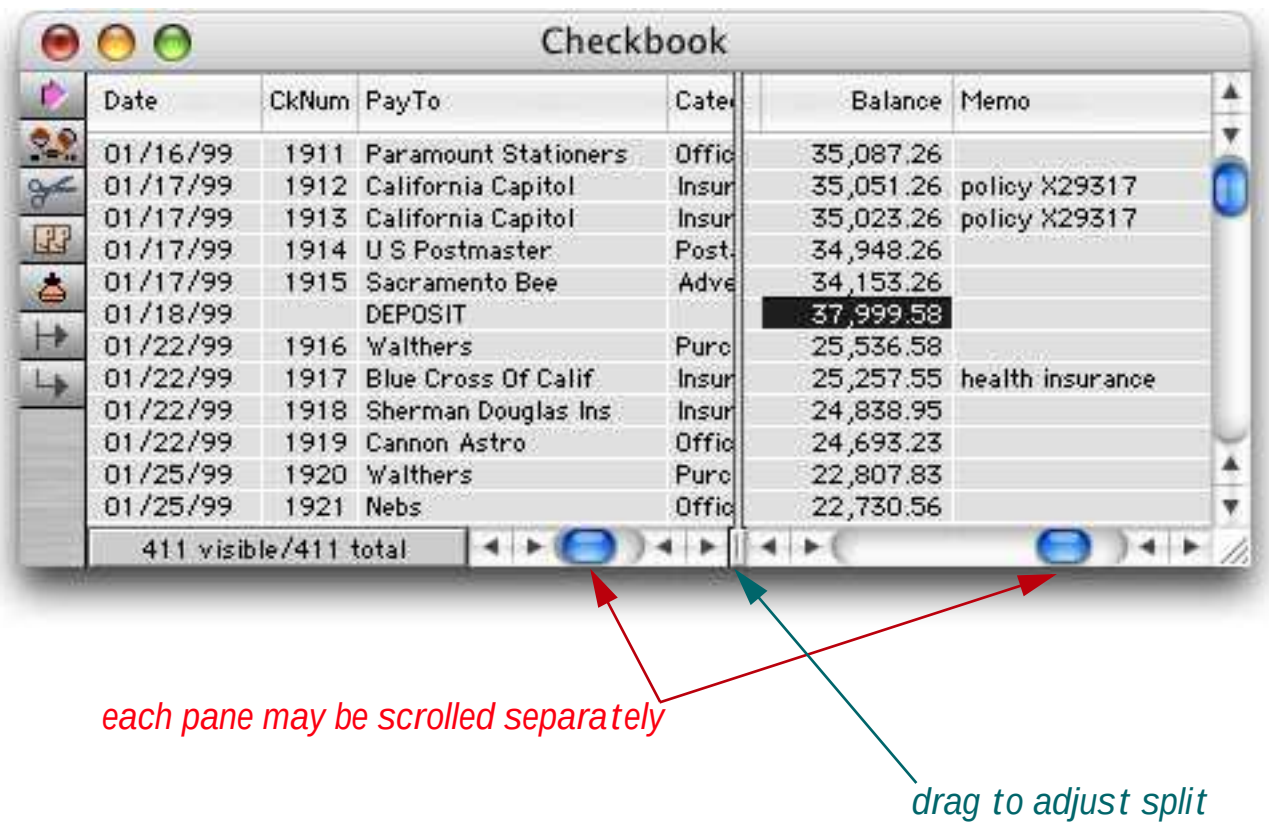
Splitting a Window

Some Panorama windows can be split into two side by side panes. Each pane displays a different area of the database. The two panes are locked together vertically, but each pane has its own horizontal scroll bar.



press and drag to split window

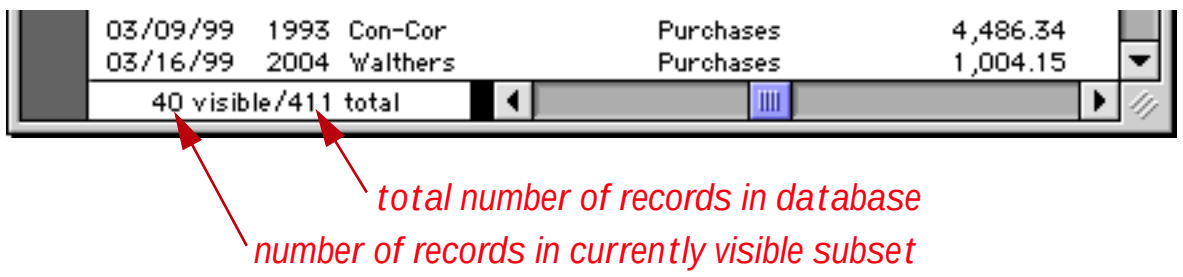
To split a window, drag the **splitter** to the right. (The **splitter** is the small black rectangle to the left of the horizontal scroll bar.)



To remove the split, drag the splitter back to the left edge. To adjust the split location, drag the splitter into position.

Info Palette

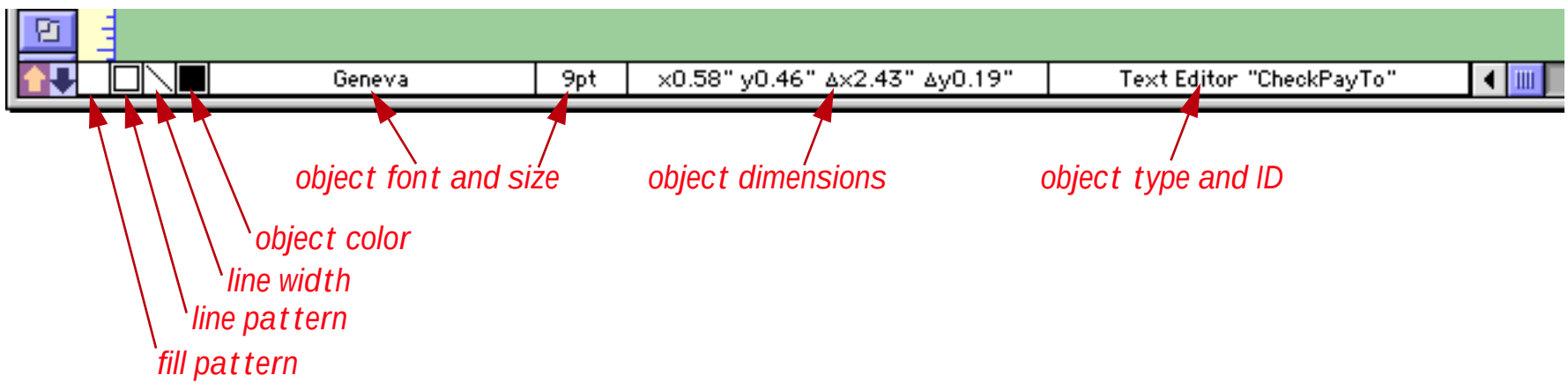
Many Panorama windows display an **Info Palette** along the bottom left corner of the window. In windows that are used for displaying and editing data (data sheet, forms, crosstabs), the info palette displays the current number of database records in the bottom left corner of each window. Panorama displays both the total number of records and the number of visible (selected) records.



If the window is less than 3 inches wide, the record count will not appear. You can also turn the record count on or off with the **Show Record Count** command in the Setup menu.

Clicking anywhere in the record count opens the **Find/Select** dialog. This dialog allows you to locate information within the database. See “The Find / Select Dialog” on page 369 for details on this dialog.

When a form window is switched into **Graphic Display Mode** the Info Palette displays a graphic control strip. This strip displays the graphic attributes of the currently selected object (or objects), and allows you to change the attributes with pop-up menus.

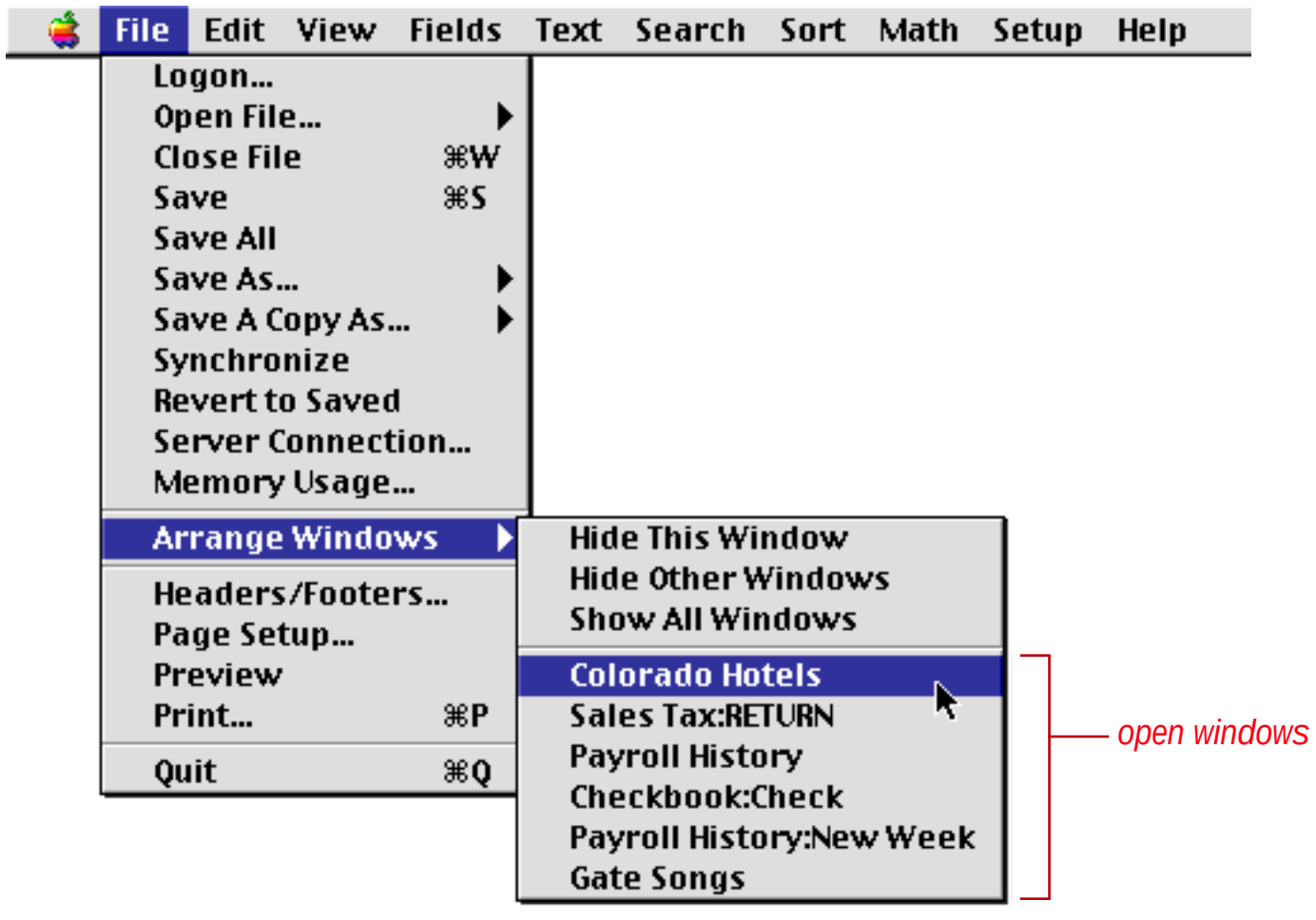


See "The Graphic Control Strip" on page 505, for more information on the graphic control strip.

Procedure windows also display status information along the bottom of the window. When writing a procedure Panorama displays error messages here. When single stepping through a procedure Panorama displays the results of any assignments into fields or variables in this area (see "The Panorama Interactive Debugger" on page 1434).

Bringing a Window to the Front

Each new window you open appears on top of the other windows. To bring another window to the front, simply click anywhere on the window. If the window is hidden you can bring it to the front with the **Arrange Windows** sub-menu in the File Menu.



Another way to bring a window to the front is to use the **Arrange Windows** wizard (see "Bringing Windows to the Front" on page 215).

Hiding Windows

The **Hide This Window** command (in the File:Arrange Windows submenu) temporarily hides the current window. The window becomes invisible, but can be made to re-appear by selecting it from the Arrange Windows submenu.

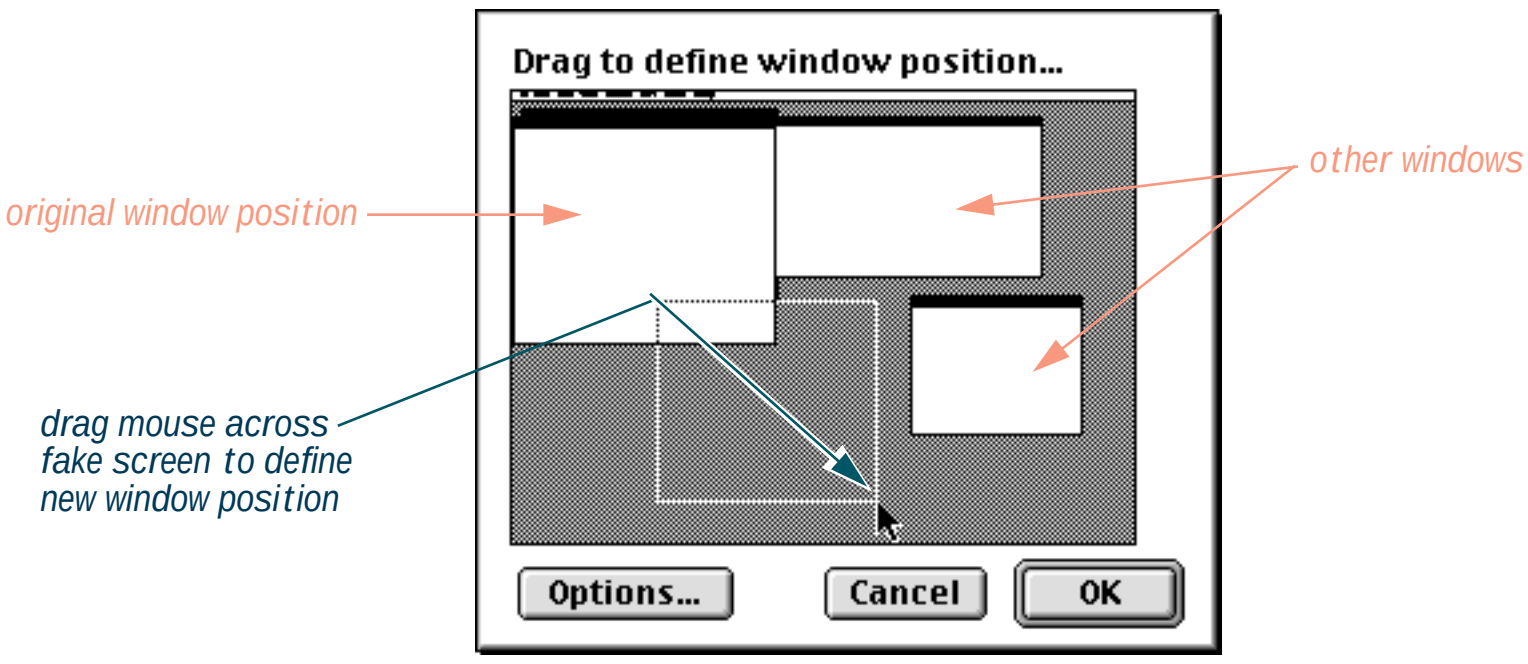
The **Hide Other Windows** command (in the File:Arrange Windows submenu) temporarily hides all Panorama windows except for the current window.

The **Show All Windows** command (in the File:Arrange Windows submenu) makes all invisible windows visible again.

Zooming Into a Box

The zoom box can also be used to zoom the window into a pre-defined box on the screen. This allows you to move and resize the window in one step.

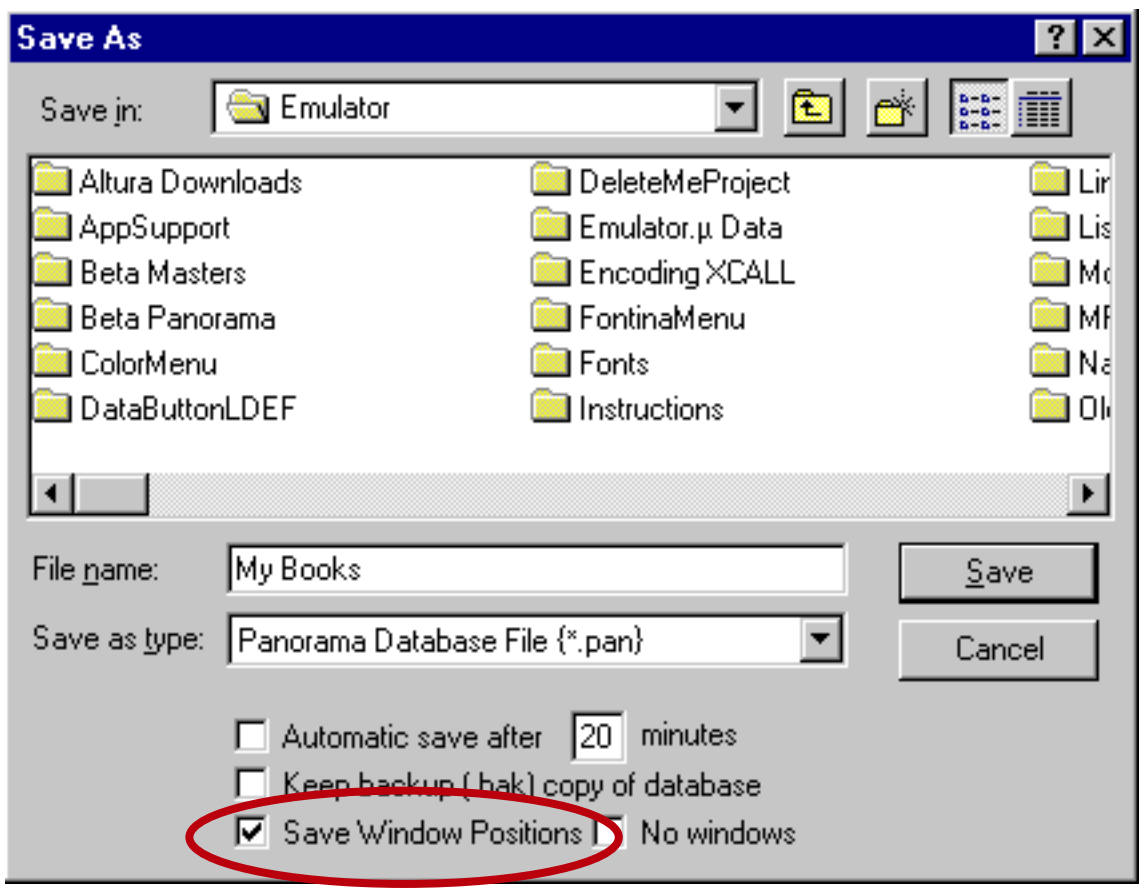
To zoom a window to a specific spot you must hold down a special key while you click on the zoom box. On Windows (PC) systems this is the **Control** key, on Macintosh systems this is the **Command** key. When you click on the zoom box while holding down this key the window options dialog will appear.



Drag the mouse across this dialog to define the new location for the window. When you press **Ok**, the window will “hop” into the new location.

Saving Window Positions

If you check the Save Window Positions option in the **Save As** dialog, Panorama will remember the position of each open window in the file being saved. The next time the file is opened, Panorama will open the same windows in the same positions.



Note: The Save Window Positions option is only available when you save an individual file. You cannot save window positions as part of a file set. If you want the files in a file set to open with predetermined window positions, you must save each individual file with the Save Window Positions option enabled.

Saving with No Windows

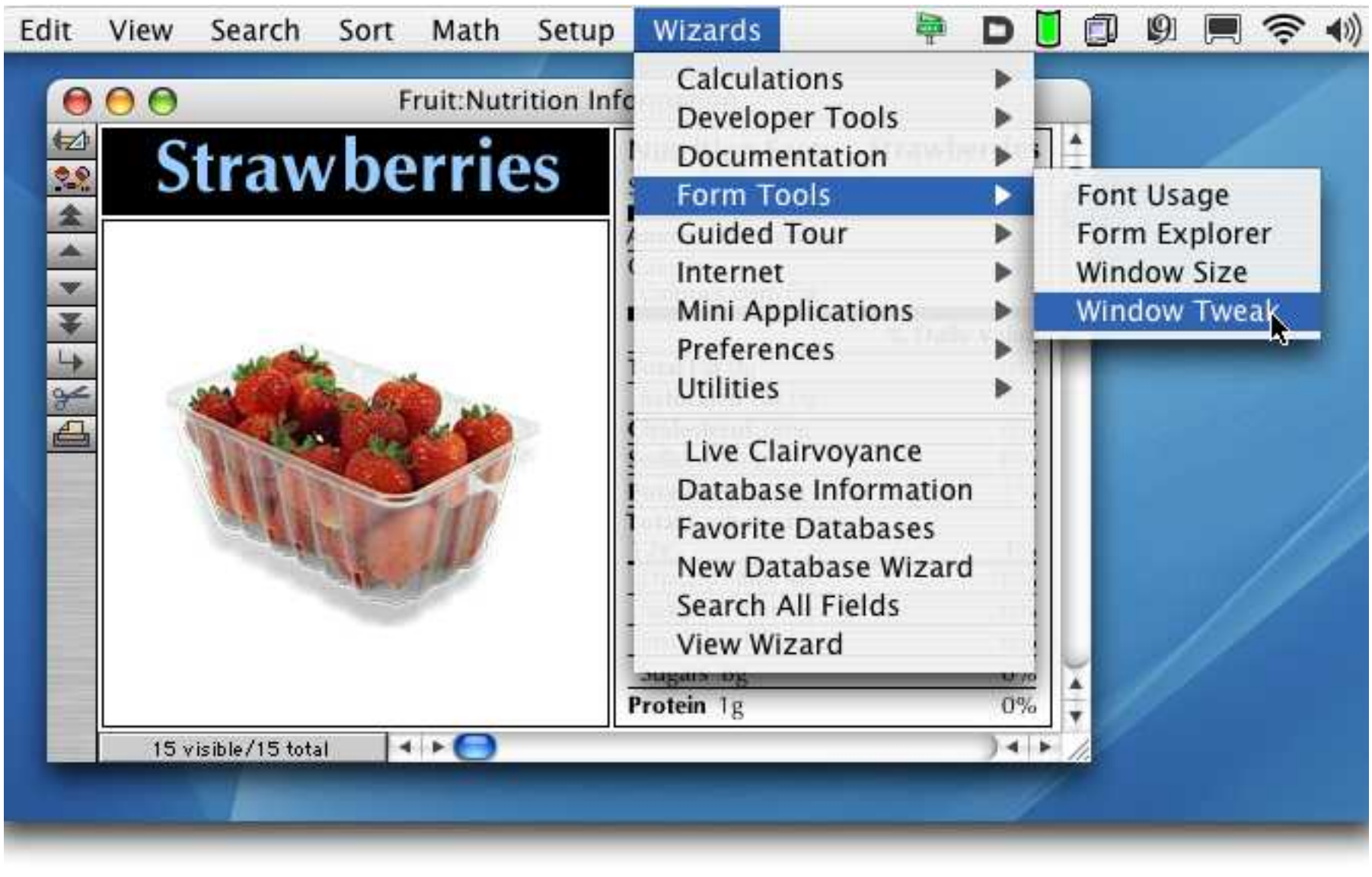
If you check the No Windows option the database will open without any open windows at all. Why would you want a database without any windows? It could be used as an invisible reference to store information that is needed by other databases but never (or rarely) changes, for example a tax or shipping rate table (see “Linking With Another Database” on page 1272). A database with no windows can also be referenced by a procedure using “secret windows” (see “Temporary “Invisible” Windows” on page 1581).

If you want to open a window in a database that has been saved with the No Windows option you can use the **View Wizard**. See “The View Wizard” on page 227 to learn how to open windows in any database. You can also open a window in an invisible database with a procedure. See “Databases Without Windows” on page 1582 to learn how to bring back windows using this technique.

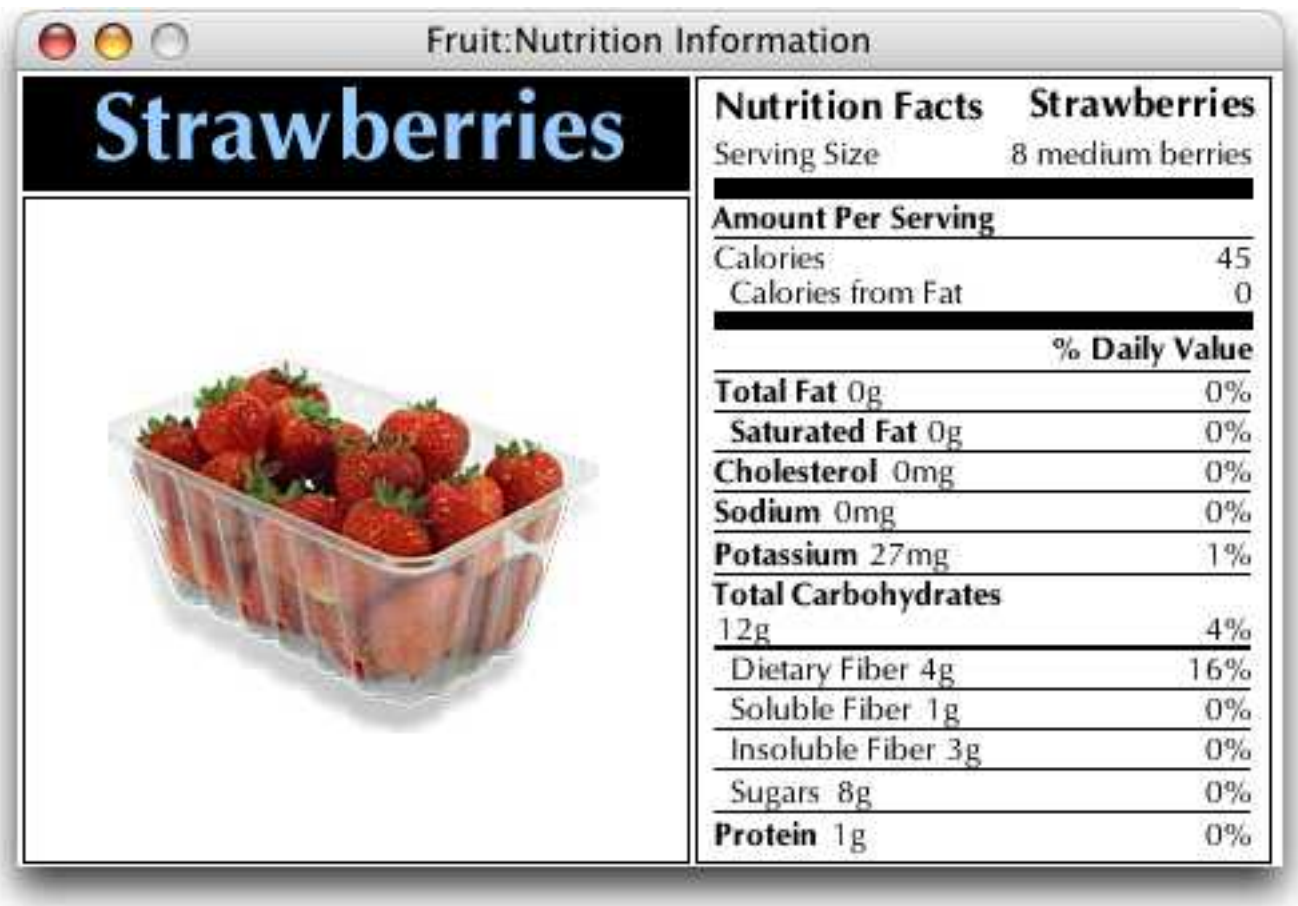
As an alternative to saving a database with the No Windows option you can instead open the database with a procedure using the `opensecret` statement (see “OPENSECRET” on page 5582). This statement opens the database without any windows.

Turning Window Components On and Off (Window Tweak Wizard)

Using the **Window Tweak** command in the **Wizard** menu you can enable and disable the tool palette and scroll bars in a form. Start with a normal form window.



When you choose **Window Tweak** the wizard will remove the tool palette and scroll bars from the window.



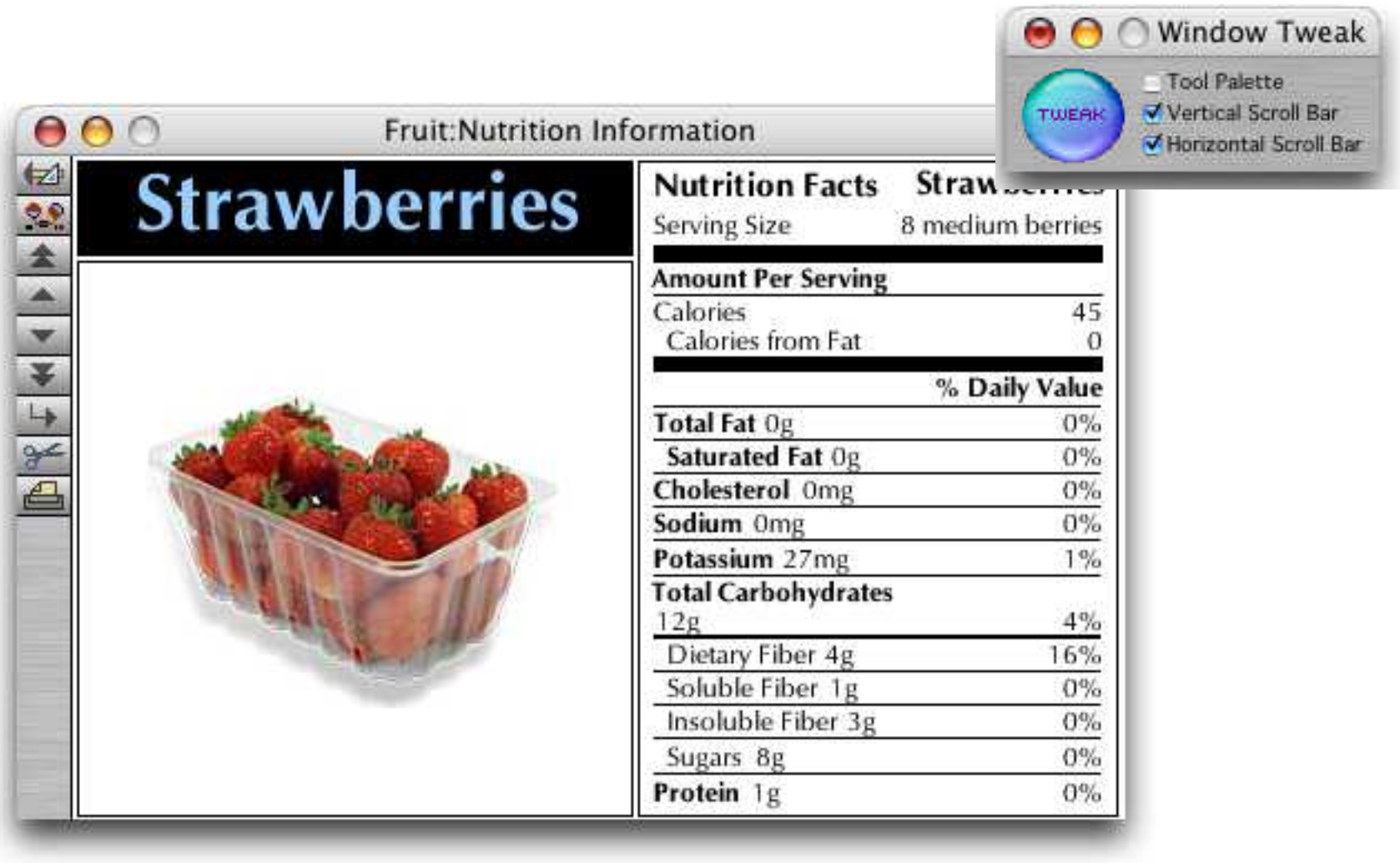
Choose **Window Tweak** again to bring the tool palette and scroll bars back again.

In addition to "tweaking" the current window the **Window Tweak** wizard also opens a small window.



When this window is open you can quickly tweak any form window. Simply bring the form window to the front, then click on the big round **Tweak** button.

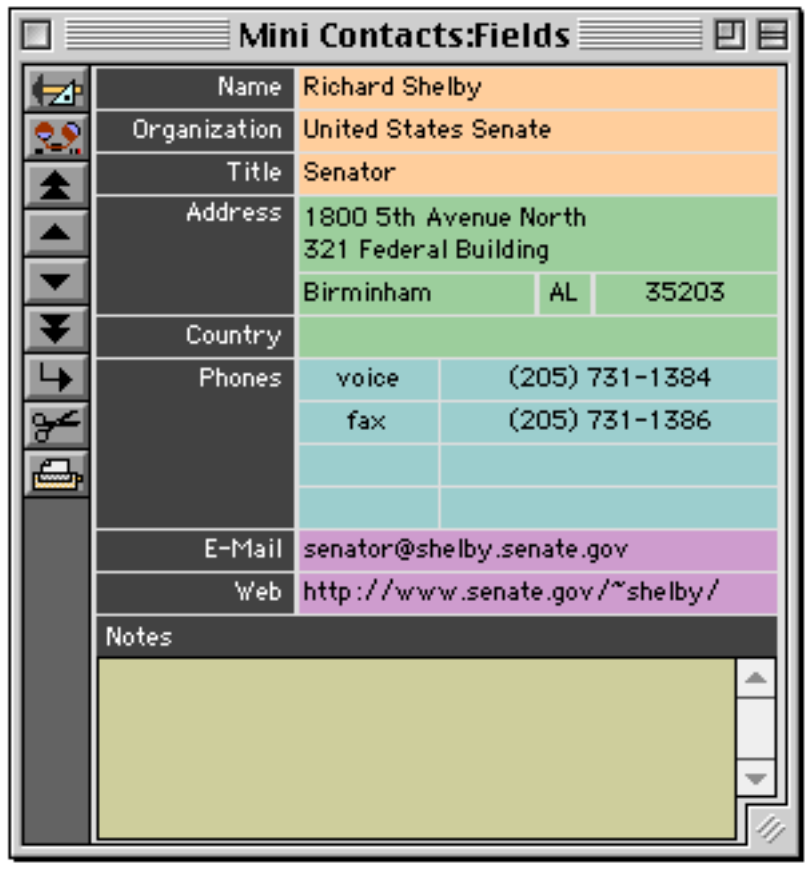
You can also use this window to control which window components get "tweaked." Normally both scroll bars and the tool palette get tweaked. However, you can disabled some of these options. A disabled option doesn't get tweaked. For example, if the Tool Palette option is disabled then the tool palette will not be removed. Here is a window with both scroll bars "tweaked" but the tool palette has been left alone.



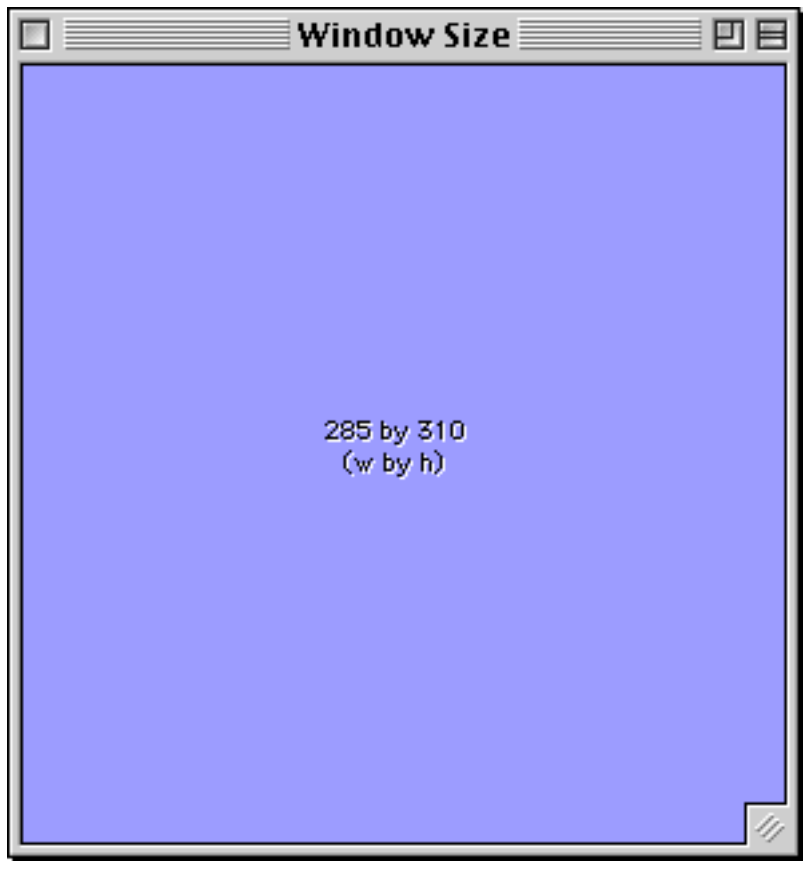
When you press the **Tweak** button again the scroll bars will re-appear.

Measuring a Window (Window Size Wizard)

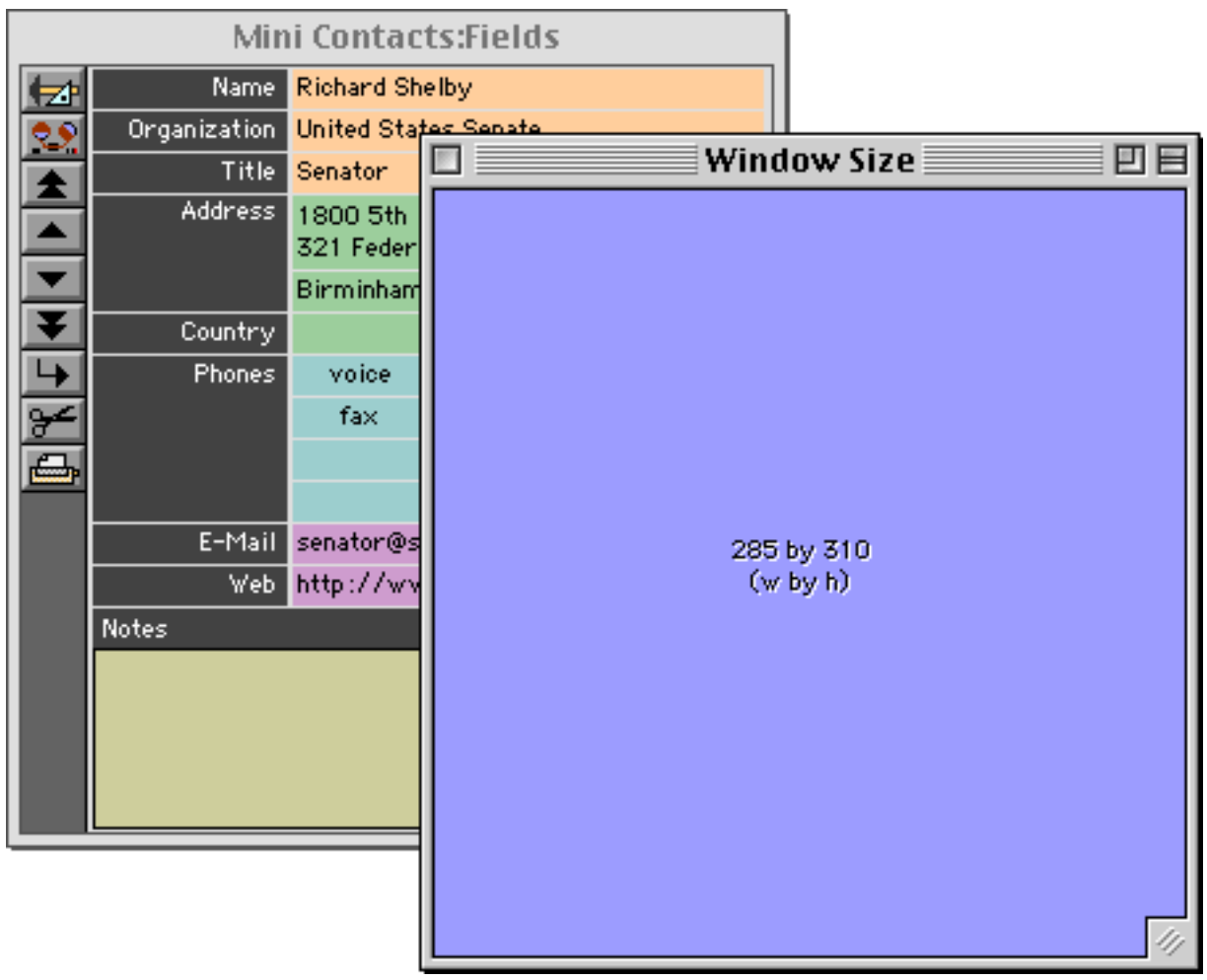
Sometimes you may need to measure the height and width of a window (for example if you want to write a procedure to re-open the window the same size, see “Specifying the New Window Location” on page 1573). It’s easy to measure a window with the **Window Size** wizard. First bring the window you want to measure to the front, then select **Window Size** from the Form Tools submenu of the Wizard menu.



The wizard will cover the window and show the dimensions of the window.



Don’t worry, your original window is still there. It’s just hidden behind the Window Size wizard, as you can see by dragging the wizard to the side.



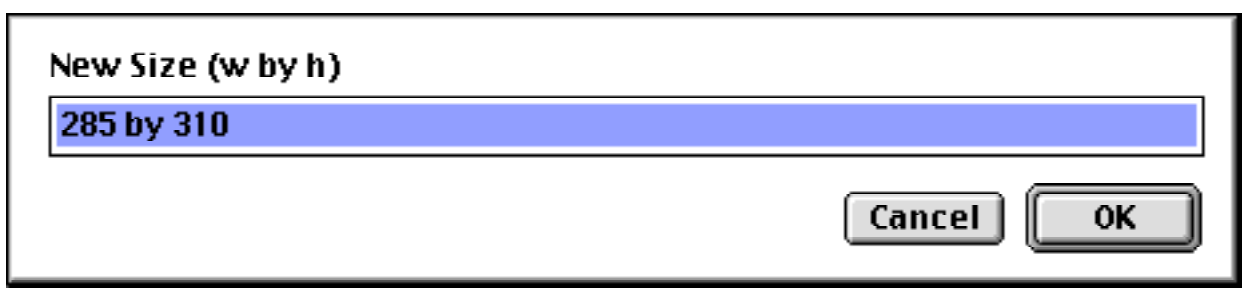
You can drag and resize the window anywhere you like. Or you can simply click on a window and select **Window Size** from the Wizard menu and the size window will jump to cover the window you have selected. When you’re done with the window you can simply close it.

Setting Exact Window Dimensions

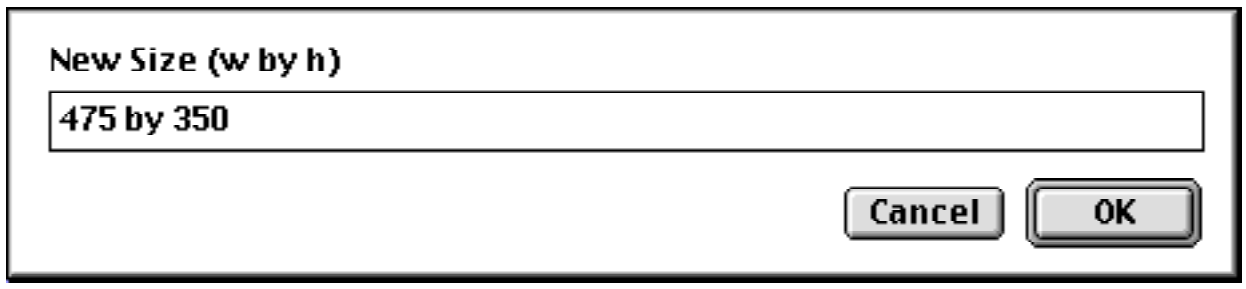
In addition to displaying the current window dimensions the **Window Size** wizard can also be used to precisely set new dimensions. Start by opening the wizard, as described in the previous section. Then choose the **Set Size** command from the **WindowSize** menu.



This command displays a dialog that shows the current window dimensions.



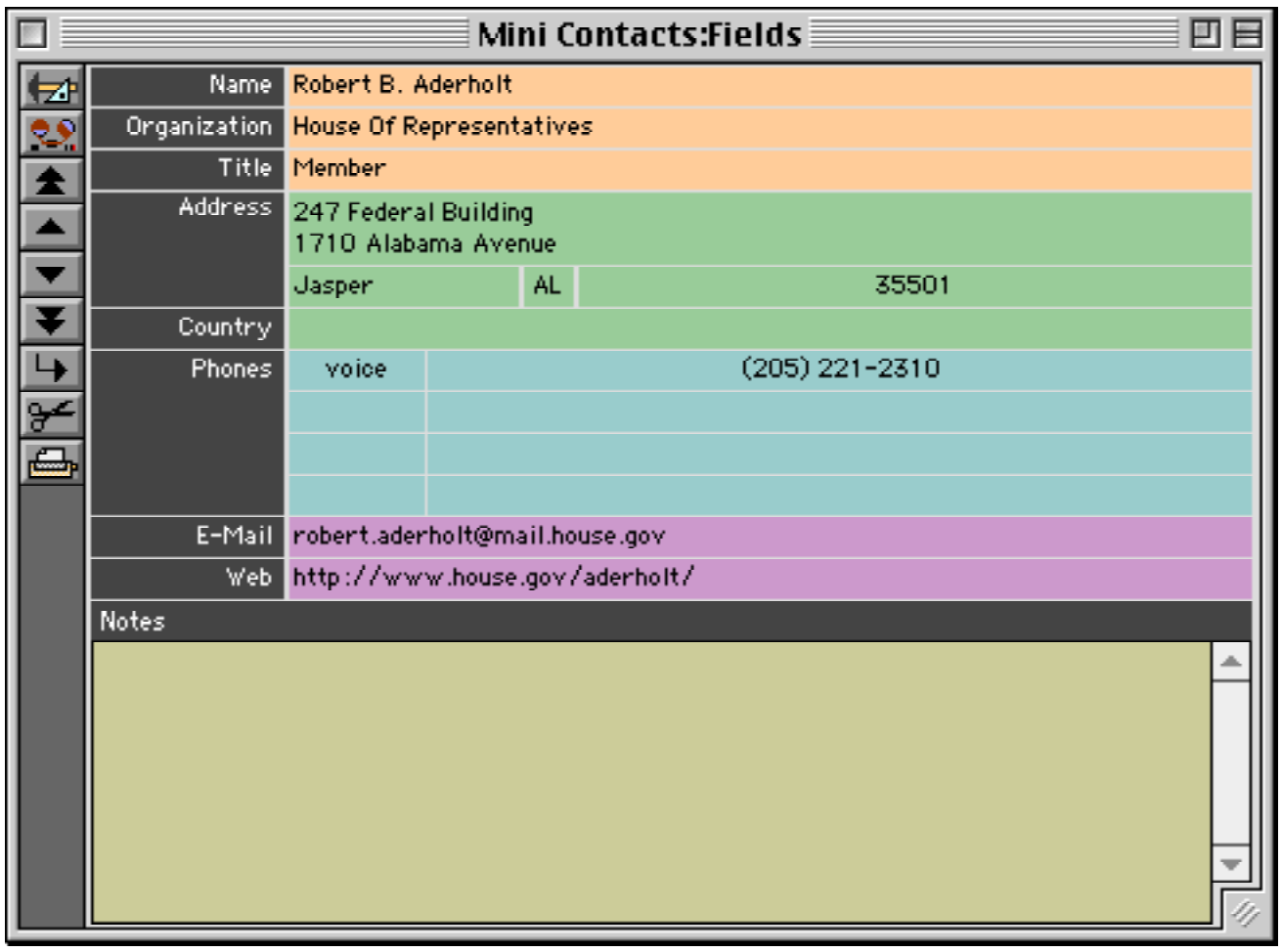
Use this dialog to type in the new dimensions you want.



When you press the **OK** button Panorama will adjust the size of the wizard's window to the new dimensions.



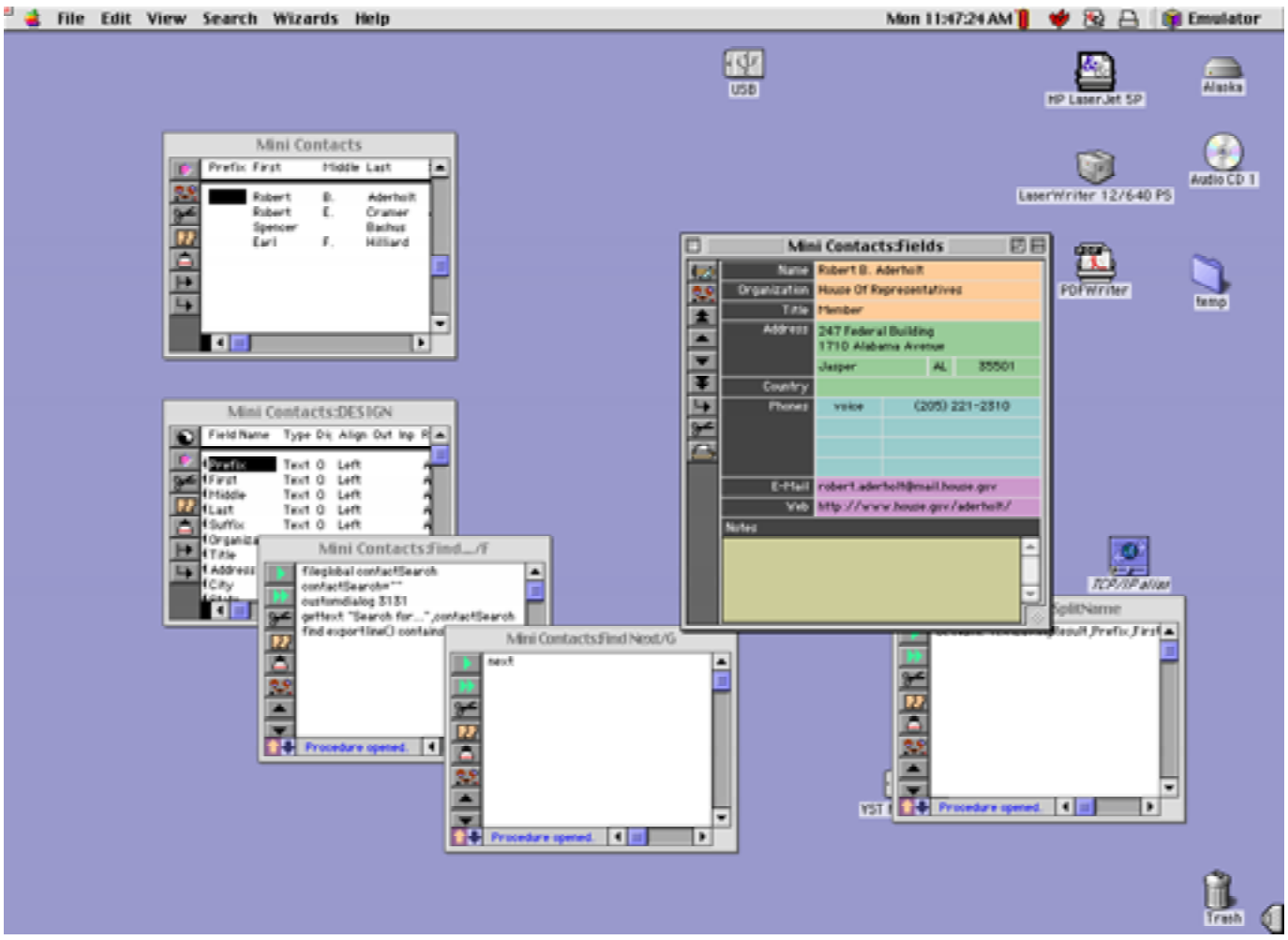
When you close the wizard's window (or move it to the side) you will see that the original window has also been adjusted to the new dimensions.



Arranging All Open Windows at Once (Tiling and Stacking)

Normally you manipulate one window at a time. The **Arrange Windows** wizard allows you to arrange all of the open Panorama windows into a regular pattern, either side by side (tiled) or piled on top of each other with a slight offset.

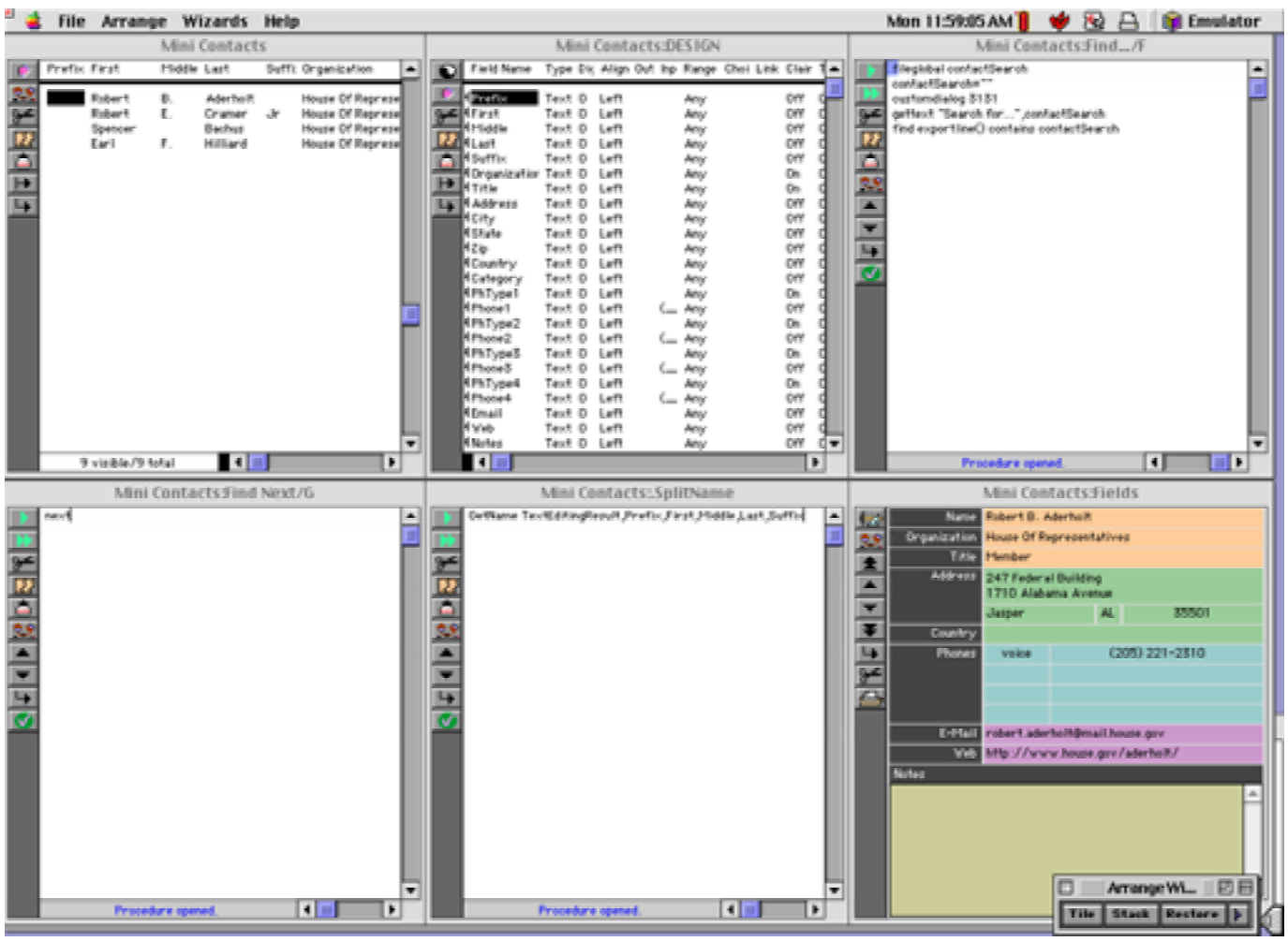
To illustrate the operation of this wizard, let's suppose that there are currently six Panorama windows open in more or less random locations, like this.



The first step is to choose **Arrange Windows** from the Wizard menu. This displays the **Arrange Windows** options.

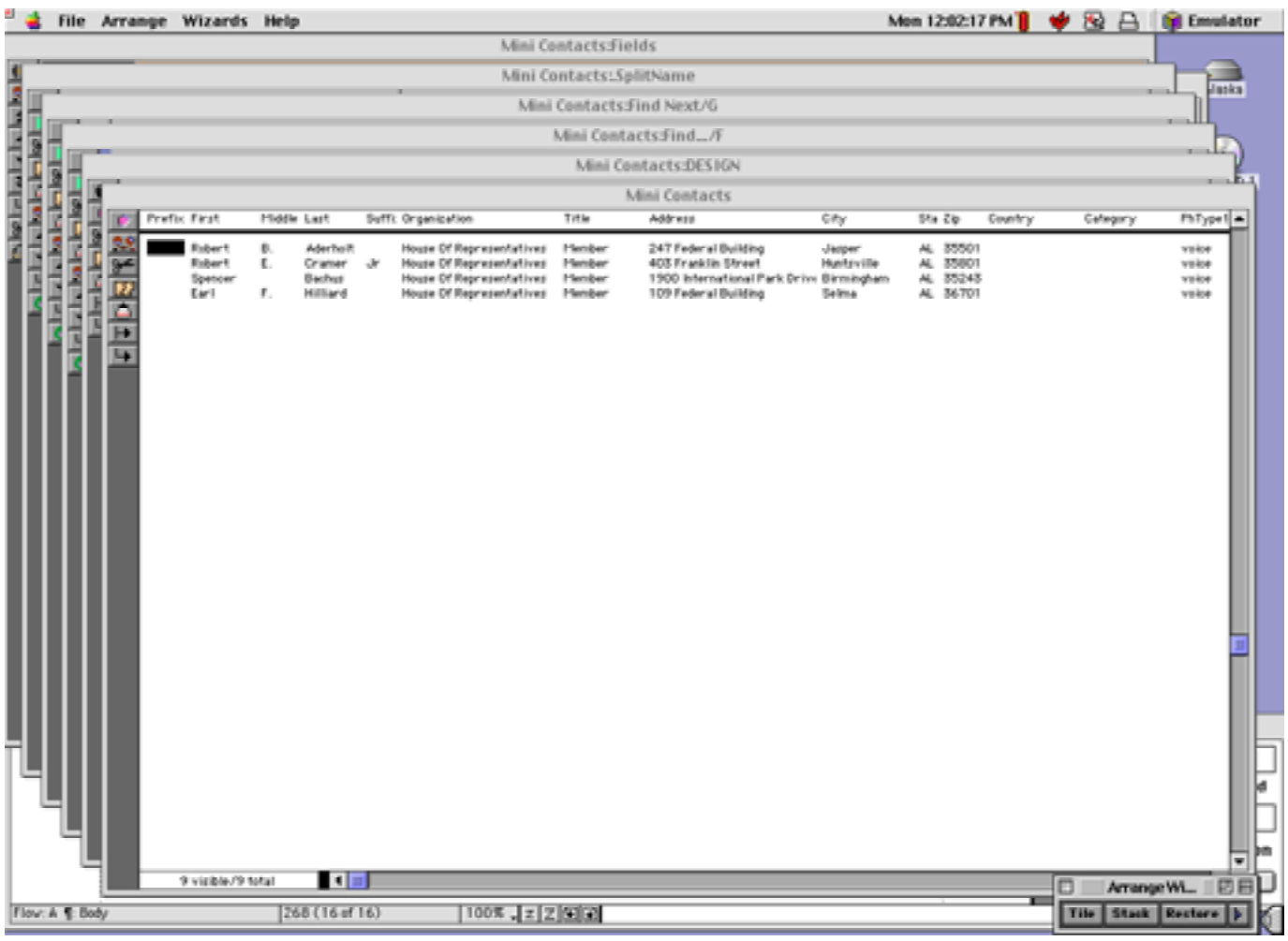


To arrange the windows in a grid (like floor tiles) press the **Tile** button.

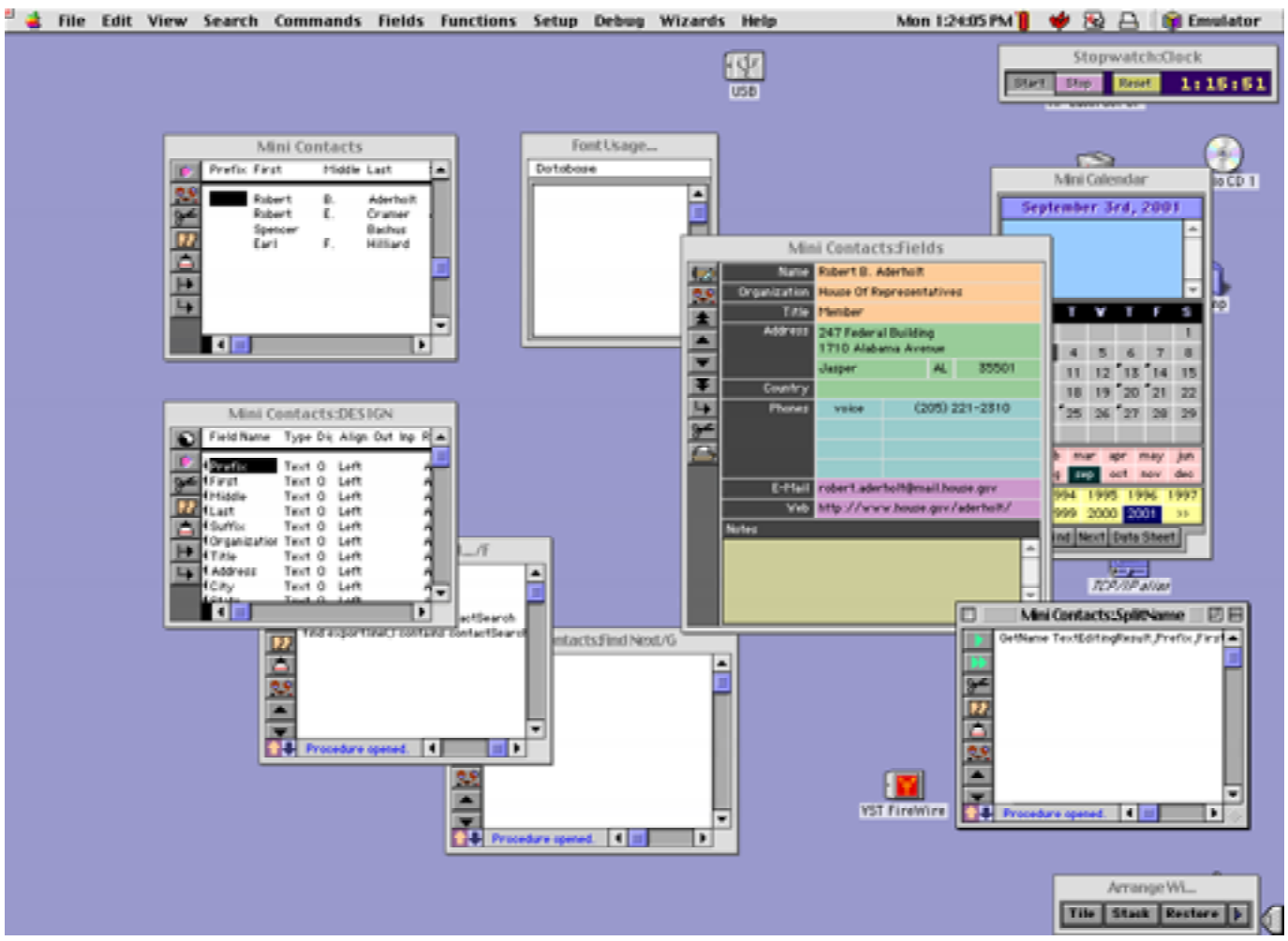


If you decide you don't like this new arrangement you can revert to the original window positions by pressing the **Restore** button.

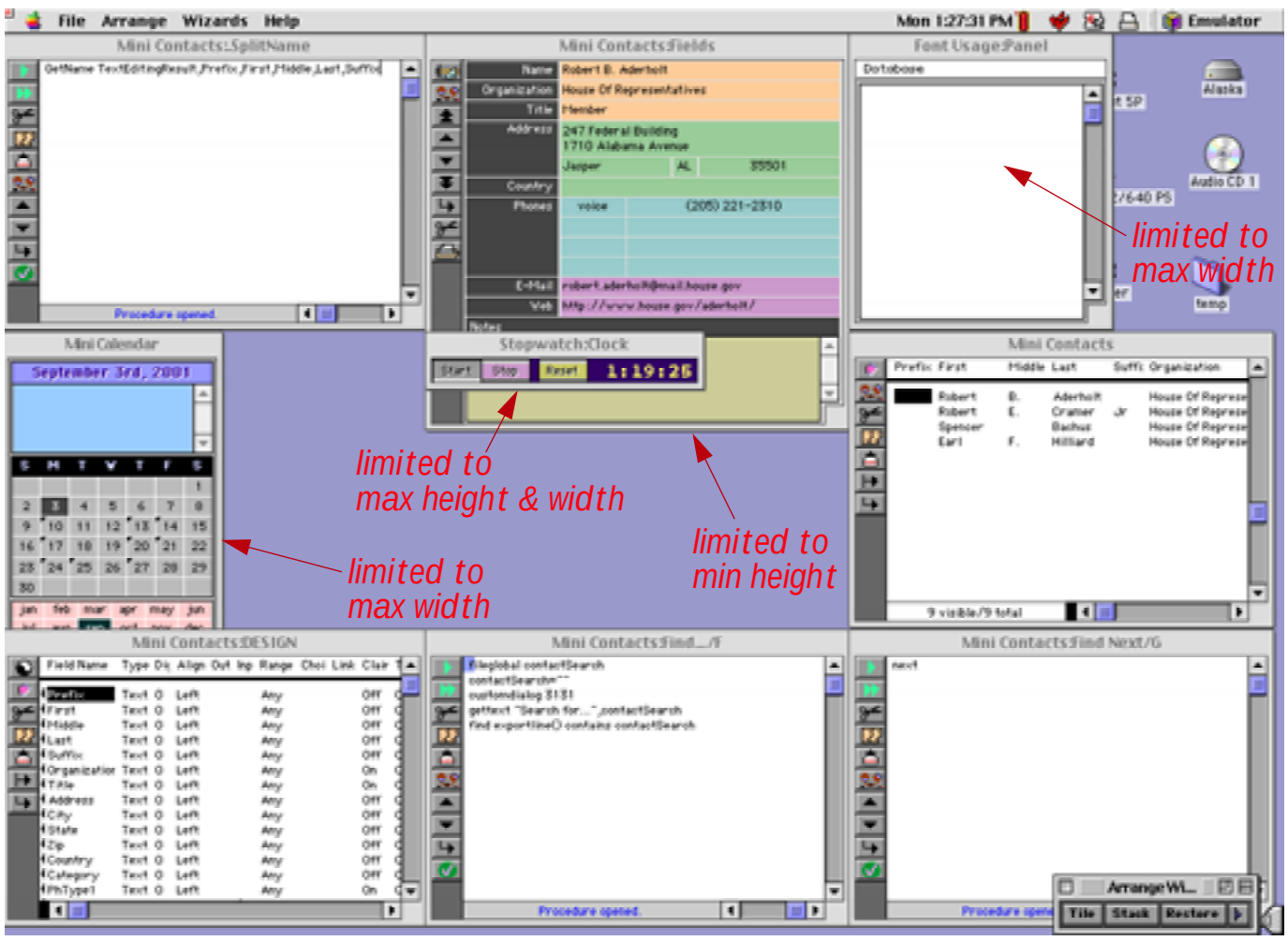
To make all the windows the same size, arranged in a staggered pattern, press the **Stack** button.



When Panorama stacks or tiles the windows it checks the attributes of each window. It won't make any window smaller than it's minimum size or larger than it's maximum size. Sometimes this can prevent the tile or stack patterns from aligning perfectly. For example, suppose you start with the windows shown here.

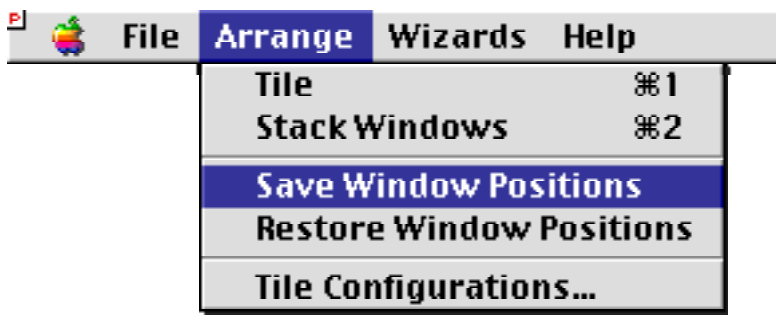


Because of restrictions on the minimum and maximum window dimensions these windows cannot fit perfectly in the tile pattern. Panorama will leave some gaps and some overlaps.



Saving and Restoring Window Positions

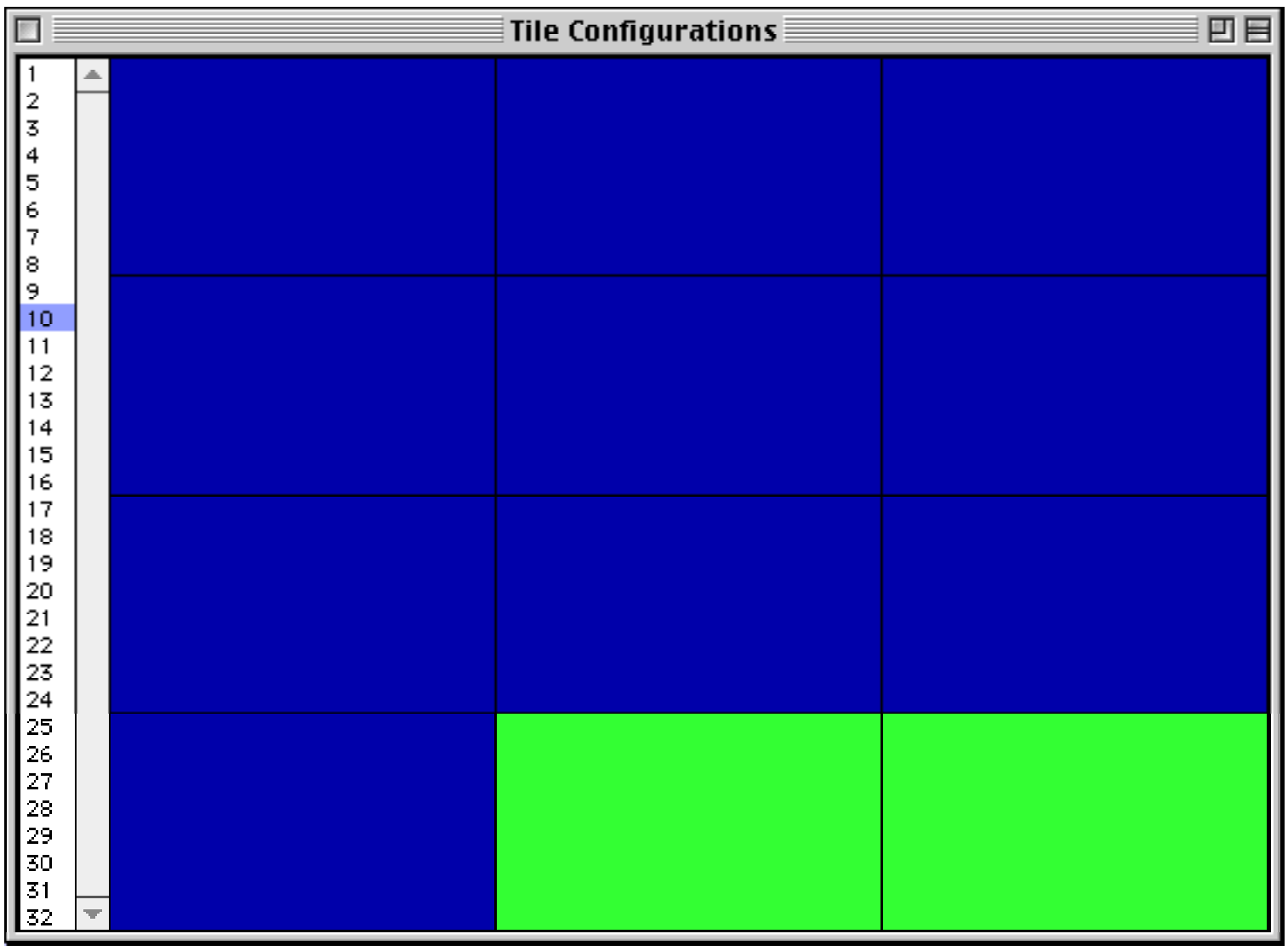
The **Arrange Windows** wizard automatically saves window positions each time you **Tile** or **Stack** the windows, allowing you to go back to the original window configuration. You can also ask the wizard to save the current window positions at any time.



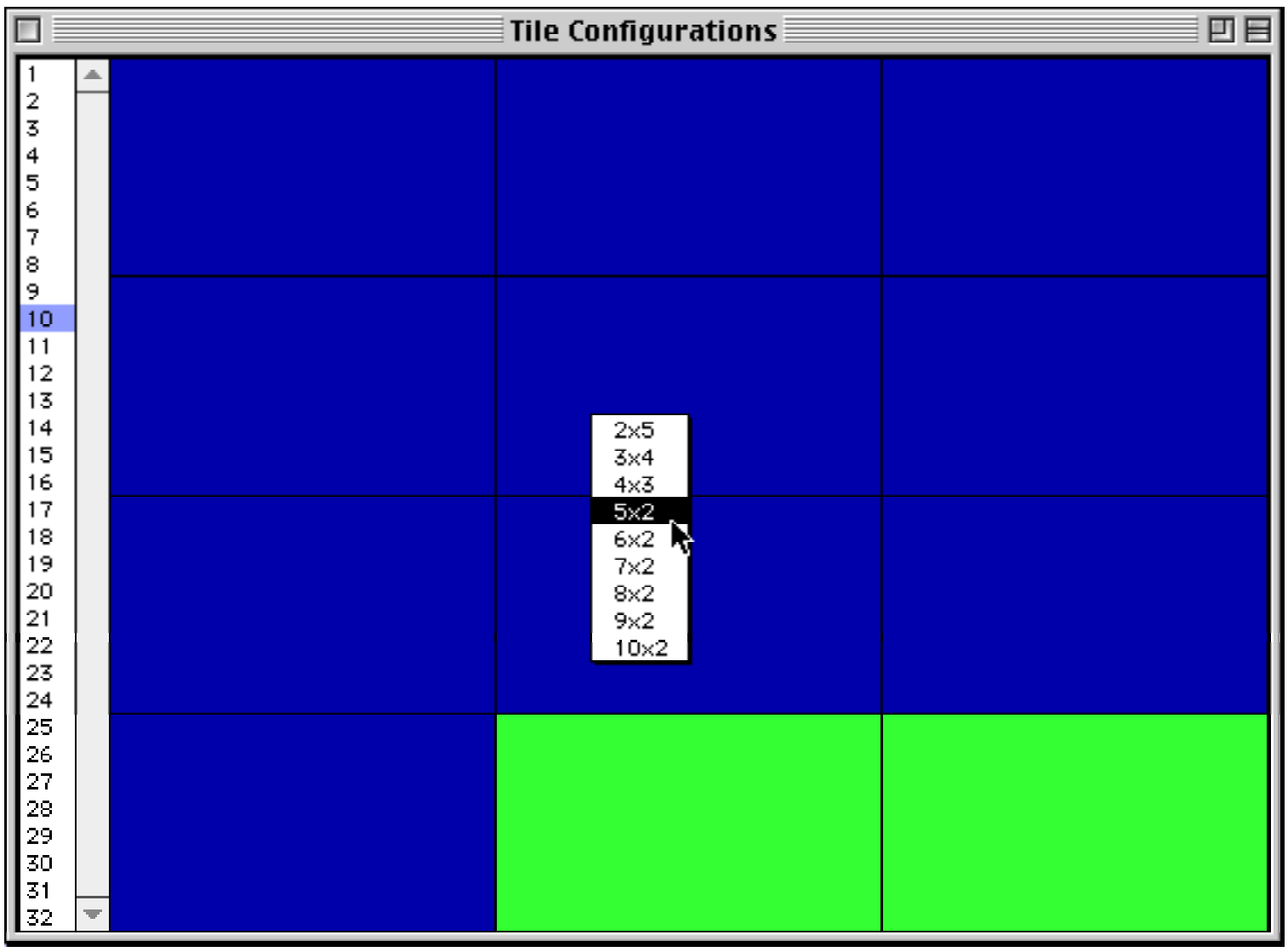
Later you can restore the original window positions by choosing **Restore Window Positions**. Please note, however, that this only affects the windows that were open when the window positions were saved. If a window has since been closed, **Restore Window Positions** will not re-open it. Conversely, if additional windows have been opened since the window positions were saved these new windows will not be affected by the **Restore Window Positions** command.

Choosing Tile Configurations

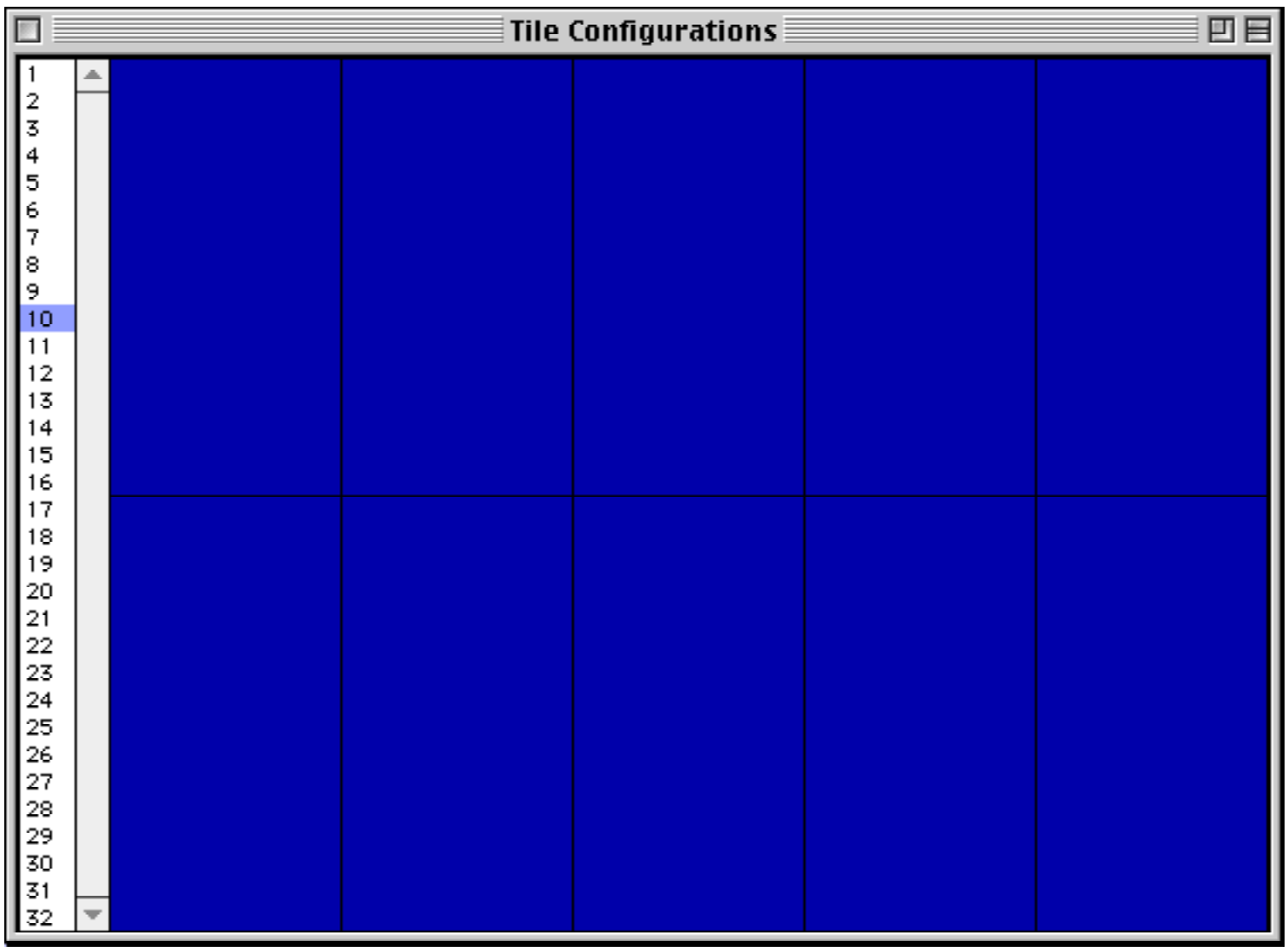
The Arrange Windows wizard allows you to choose pattern used for tiling windows. For example, suppose you have ten windows. Should the tiles be 4 by 3, or 5 by 2, or 2 by 5 or some other design? We've supplied a default arrangement, but you can open the **Tile Configurations** window to change the design at any time. When you open this window it shows you the current configuration for the current number of windows. In this example there are currently 10 windows open, and they will be tiled into a 4 high by 3 wide grid. Any empty slots are displayed in light green.



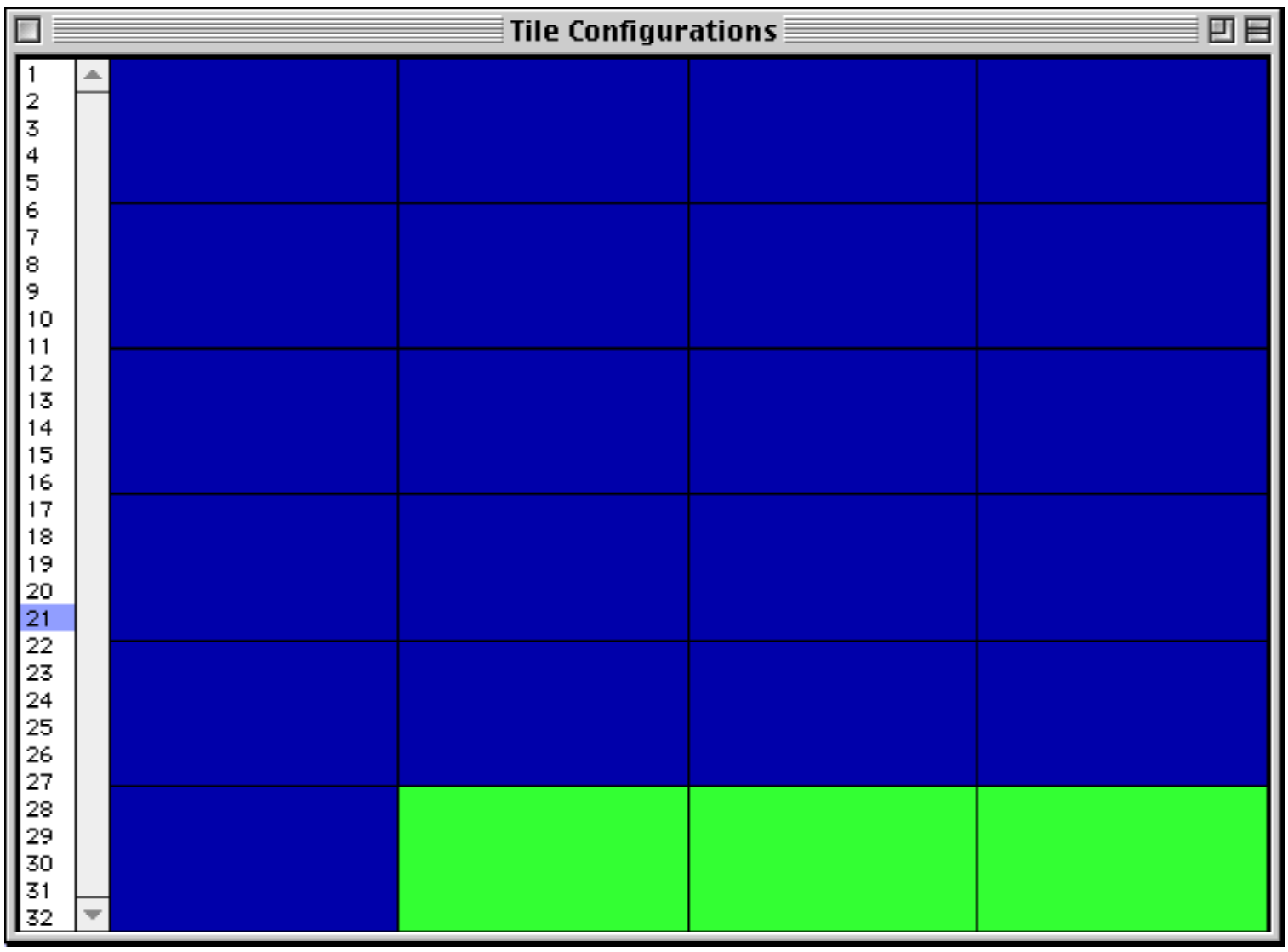
To change the configuration simply press the mouse anywhere on the grid. A pop-up menu listing possible arrangements will appear.



Select the arrangement you want from the pop-up menu. After you make your selection the grid will be updated to reflect your choice.



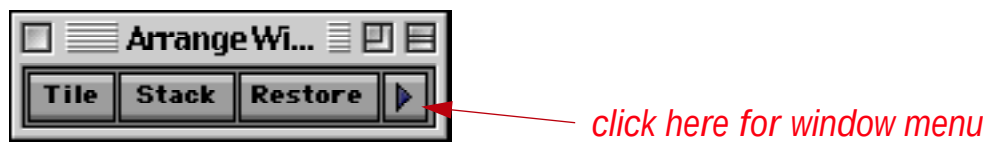
You can use the list on the left side of the window to set up the configuration for any number of windows (up to Panorama's maximum of 32 windows).



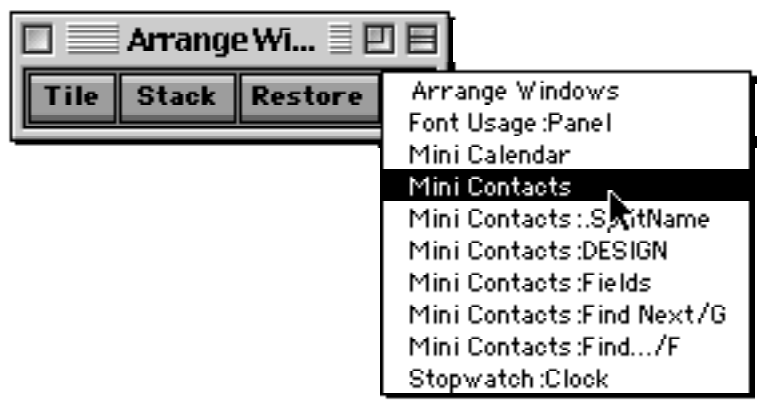
When you are done simply close the Tile Configuration window and press the **Tile** button.

Bringing Windows to the Front

The Arrange Windows wizard also provides an alternative method for bringing a window to the front. To bring any window to the front, click on the arrow button.



When you press on this button a menu listing all the open windows will appear. Simply select the window you want to bring to the front.



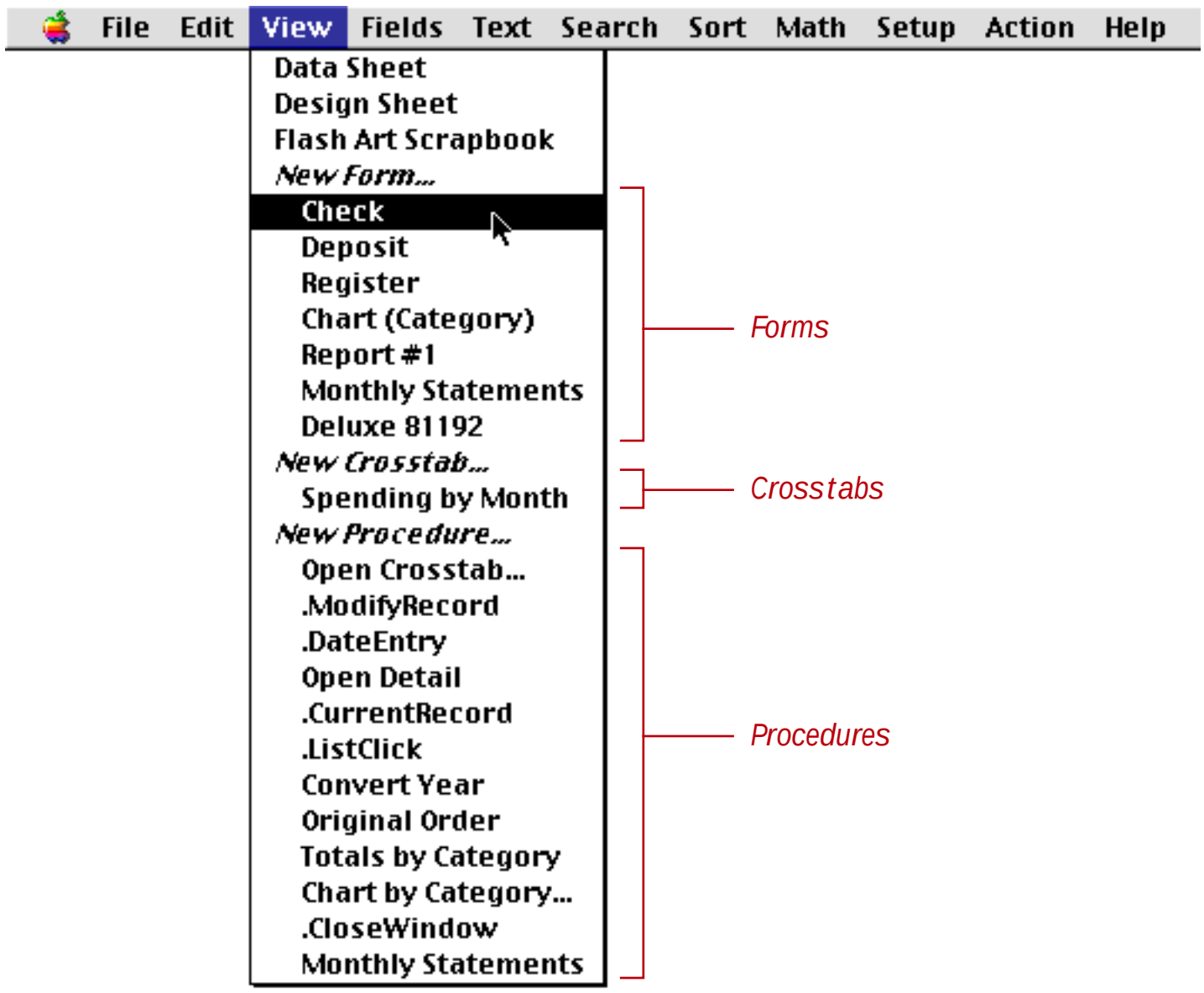
Of course you can also bring a window to the front by clicking on it or using the **Arrange** submenu in the File menu (see "Bringing a Window to the Front" on page 200).

Chapter 3: Views



A Panorama database can have up to six elements: data sheet, forms, design sheet, procedures, crosstabs, and flash art. Each window shows a view of one of these elements.

The **View** menu is just to the right of the Edit menu.



Use this menu to pick which view you want to see in the window. Choosing different views from this menu is like pointing a camera in different directions.

Types of Panorama Views

Panorama has six different kinds of views. Each type of view gives you access to a different element of the database or gives you a different perspective on your data (for example, form vs. data sheet).

A new database starts with a data sheet, design sheet, and an empty flash art scrapbook. Views for forms, procedures, and crosstabs can be added if desired.

Data Sheet and Form Views

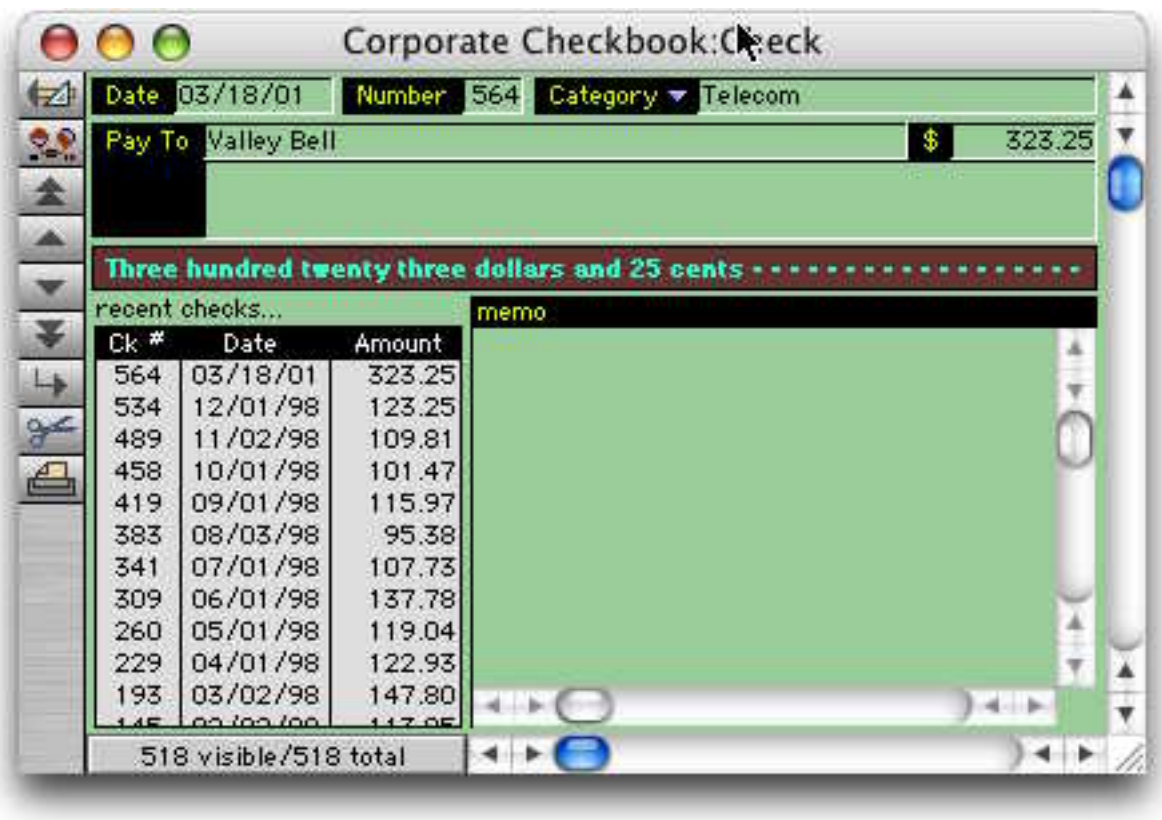
The most important views are the data sheet and forms. These views give you access to the actual data. Use the data sheet view when you want to display the database as a sheet of rows and columns. The data sheet view has a fixed format very much like a spreadsheet, as shown below. Although you can make minor alterations like changing the font size or the width of a column, you cannot add graphics or change the overall arrangement of the data sheet view.

Checkbook

Date	Ck #	Pay To	GL Category	Debit	Credit	Balance
01/01/90		OPENING BALANCE			35,978.47	35,978.47
01/08/90	1907	Northern Illinois Mold	Equipment Rental	96.05		35,882.42
01/08/90	1908	U S Postmaster	Postage	75.00		35,807.42
01/08/90	1909	Advertiser's Mailing Service, I	Advertising	390.80		35,416.62
01/16/90	1910	Coudert Brothers, Attorney's A	Legal Fees	223.52		35,193.10
01/16/90	1911	Paramount Stationers	Office Supplies	105.84		35,087.26
01/17/90	1912	California Capitol	Insurance	36.00		35,051.26
01/17/90	1913	California Capitol	Insurance	28.00		35,023.26
01/17/90	1914	U S Postmaster	Postage	75.00		34,948.26
01/17/90	1915	Sacramento Bee	Advertising	795.00		34,153.26
01/18/90		DEPOSIT			3,846.32	37,999.58
01/22/90	1916	Walthers	Purchases	12,463.00		25,536.58
01/22/90	1917	Blue Cross Of Calif	Insurance	279.03		25,257.55
01/22/90	1918	Sherman Douglas Ins	Insurance	418.60		24,838.95
01/22/90	1919	Cannon Astro	Office Supplies	145.72		24,693.23
01/25/90	1920	Walthers	Purchases	1,885.40		22,807.83
01/25/90	1921	Nebs	Office Supplies	77.27		22,730.56
01/25/90	1922	Ramona Drinking Water	Office Supplies	98.10		22,632.46
01/25/90	1923	Pacific Partners	Rent	4,070.83		18,561.63
01/29/90	1924	Athearn	Purchases	1,906.32		16,655.31
01/29/90	1925	Advertiser's Mailing Service, I	Advertising	860.22		15,795.09
01/29/90	1926	PacTel Cellular	Telephone	141.09		15,654.00
01/30/90	1927	State Board Of Equalization	Taxes	549.00		15,105.00
01/30/90	1928	Walthers	Purchases	828.70		14,276.30

411 visible/411 total

Use a form view when you want complete control over the arrangement and appearance of your data. Instead of appearing in a fixed row and column grid, the data can be arranged any way you want. Graphics can be added for clarity or to simulate an actual paper form. The form view is much more flexible than the data sheet view, but it is also more work to set up. Here is a typical example of a form. Notice that the window name shows the database name, Checkbook, followed by the form name, Plain Checks.



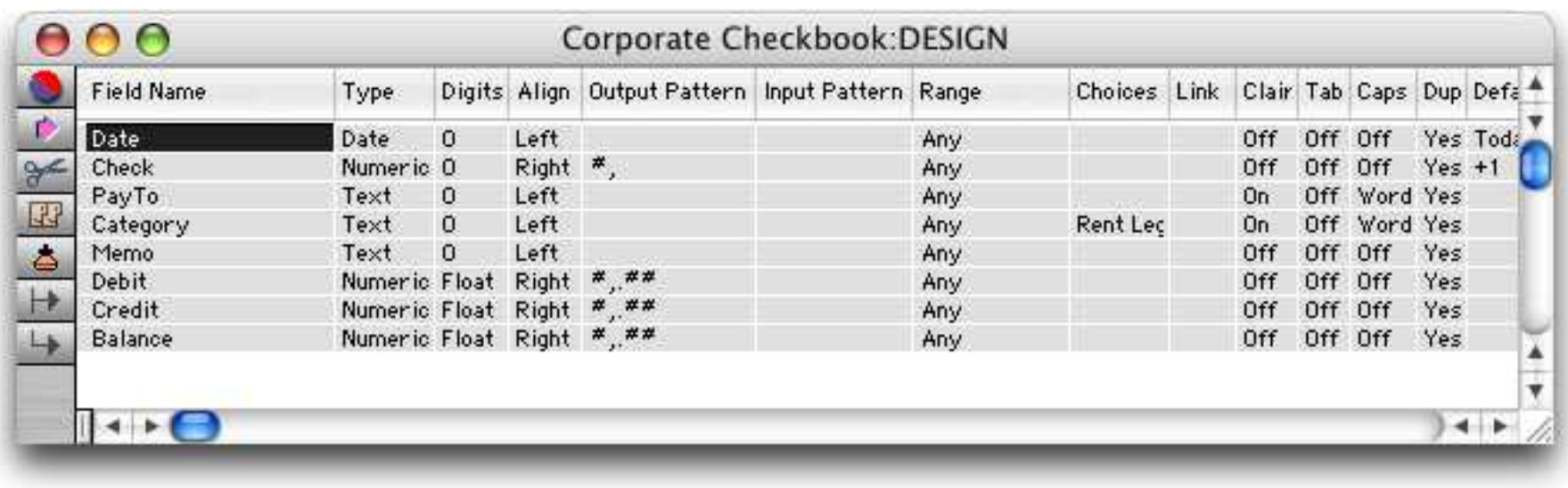
Every Panorama database has a single data sheet view, but can have any number of form views. A simple database might not have any form views, while a complex database may have dozens. You create as many forms as you need. Each form is identified by a unique name.

Since the data sheet and form views are based on the same underlying data, any action or change made to the data in one of these views will immediately appear in the other views. Of course you'll only notice this when several windows are open at once.

Other Views

In addition to the data sheet and form views Panorama has four other kinds of views. Instead of accessing the actual data, these views let you access the other components of a Panorama file.

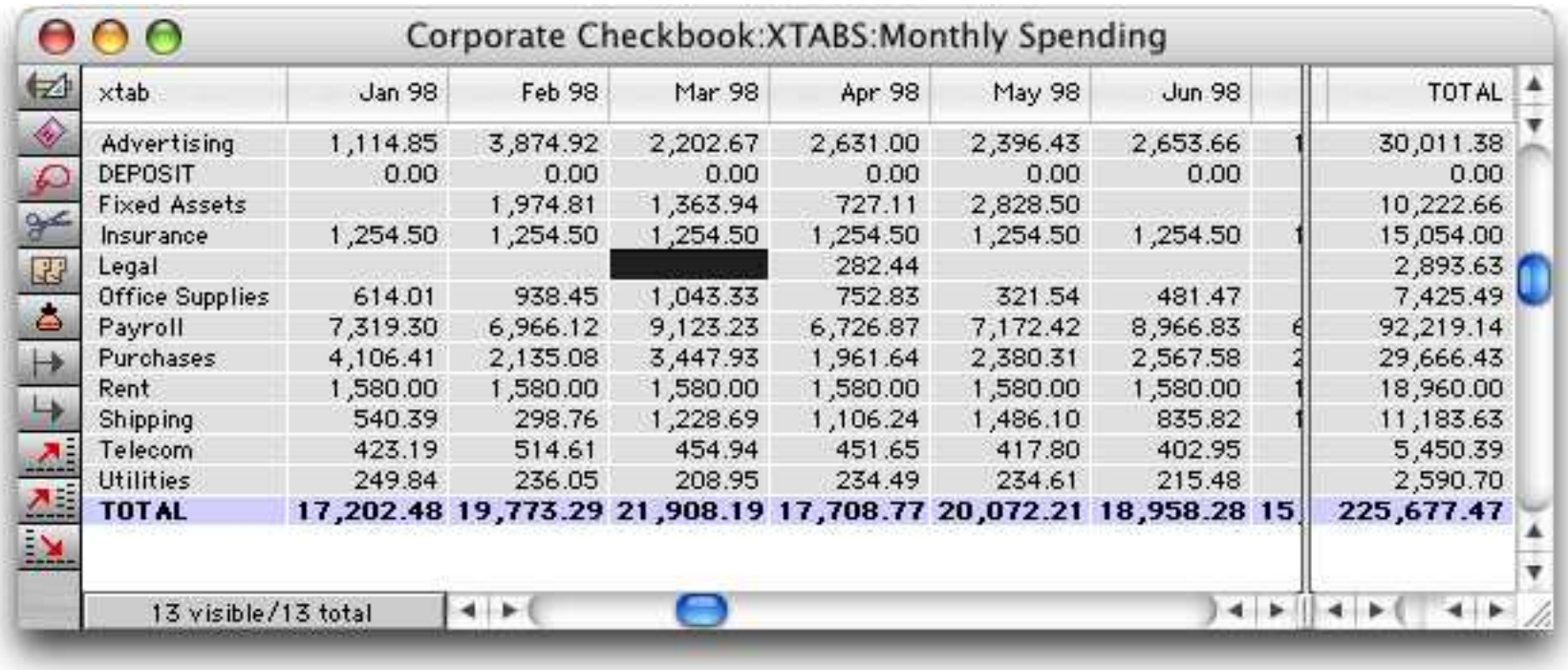
The design sheet view is the DNA or blueprint of the database. It contains the actual structure of the database fields. You can add and remove fields and make other minor modifications to the database structure without using the design sheet, but the design sheet provides the ultimate control over the structure of your database. See "The Design Sheet" on page 252 to learn more.



The flash art scrapbook view contains a catalog of pictures that can be displayed in a form or report. Using the flash art scrapbook can sometimes save a significant amount of memory over pasting pictures directly into the database, and it can be easier too. To learn more about the flash art scrapbook see “The Flash Art Scrapbook (Gallery)” on page 762.

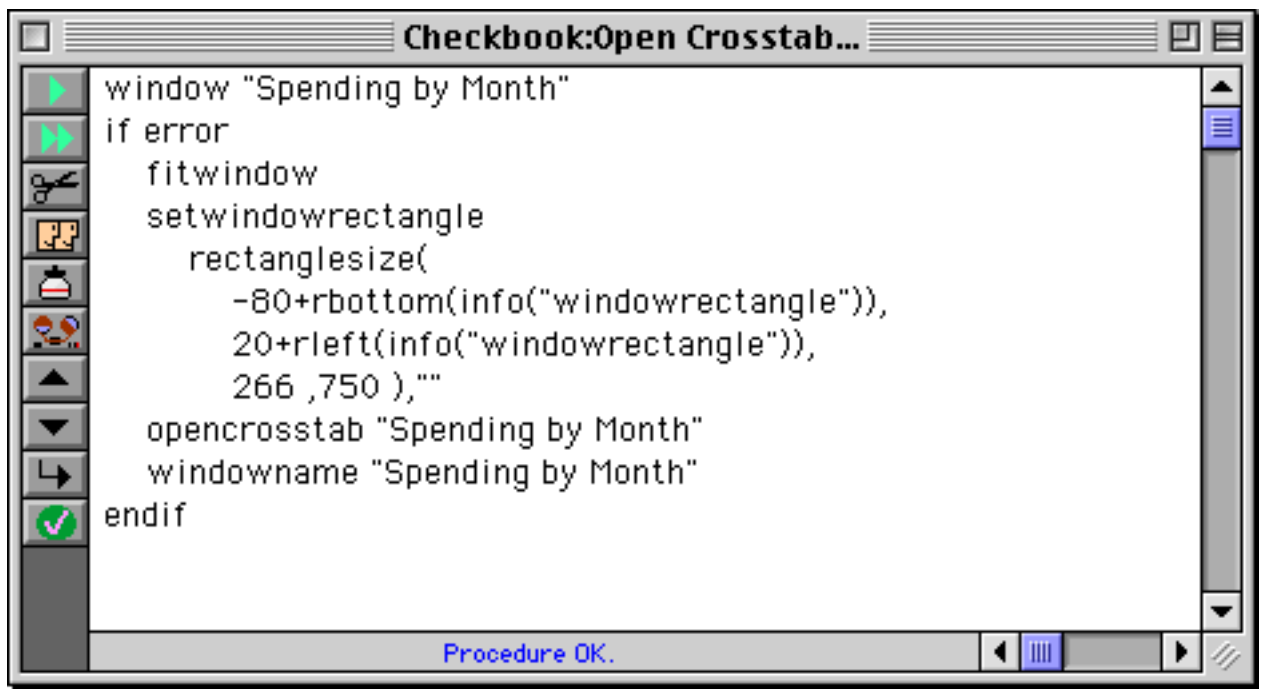


Crosstab views display a special 2-way summarization of the information in a database. Crosstabs display the relationships between data in different fields, exposing trends that might be invisible in the normal database views (data sheet and forms). Like form views, a database can contain any number of crosstab views—it may have none or several, and each crosstab has a name. The window name shows the database name (for example Checkbook), followed by XTABS, followed by the crosstab name (for example Spending by Month). To learn more about crosstabs see “Crosstabs” on page 433.



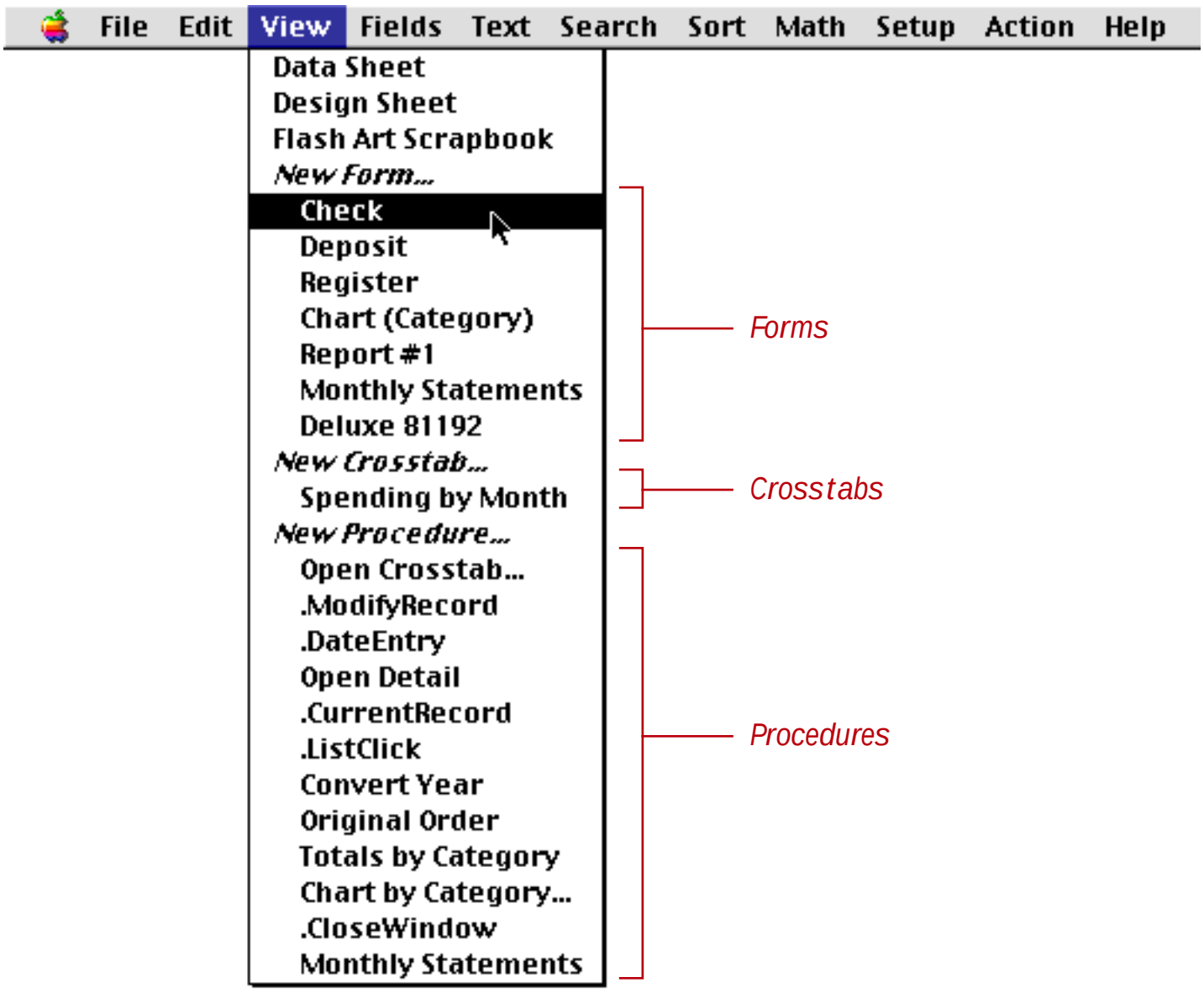
xtab	Jan 98	Feb 98	Mar 98	Apr 98	May 98	Jun 98		TOTAL
Advertising	1,114.85	3,874.92	2,202.67	2,631.00	2,396.43	2,653.66	1	30,011.38
DEPOSIT	0.00	0.00	0.00	0.00	0.00	0.00		0.00
Fixed Assets		1,974.81	1,363.94	727.11	2,828.50			10,222.66
Insurance	1,254.50	1,254.50	1,254.50	1,254.50	1,254.50	1,254.50	1	15,054.00
Legal				282.44				2,893.63
Office Supplies	614.01	938.45	1,043.33	752.83	321.54	481.47		7,425.49
Payroll	7,319.30	6,966.12	9,123.23	6,726.87	7,172.42	8,966.83	6	92,219.14
Purchases	4,106.41	2,135.08	3,447.93	1,961.64	2,380.31	2,567.58	2	29,666.43
Rent	1,580.00	1,580.00	1,580.00	1,580.00	1,580.00	1,580.00	1	18,960.00
Shipping	540.39	298.76	1,228.69	1,106.24	1,486.10	835.82	1	11,183.63
Telecom	423.19	514.61	454.94	451.65	417.80	402.95		5,450.39
Utilities	249.84	236.05	208.95	234.49	234.61	215.48		2,590.70
TOTAL	**17,202.48**	**19,773.29**	**21,908.19**	**17,708.77**	**20,072.21**	**18,958.28**	**15**	**225,677.47**

Procedure views contain sequences of instructions for Panorama to follow. You don't have to remember each step—Panorama will remember for you. Once a procedure is set up, it can be activated several different ways. You can choose the procedure from a menu, press a button, or use a **Command** key combination (Macintosh) or **Control** key shortcut (PC). Procedures can even be activated automatically when special events occur. Like form and crosstab views, a database can contain any number of procedure views. Each procedure has its own name, which is shown in the window title. To learn more about procedures see "Procedures" on page 1339.



Switching Between Views

The **View** Menu lists all the views in a database. The pre-defined views appear at the top—data sheet, design sheet, and flash art scrapbook. Next come the views you've created—forms, crosstabs, and procedures. The View Menu also contains commands for creating your own new views—new form, new crosstab, and new procedure.

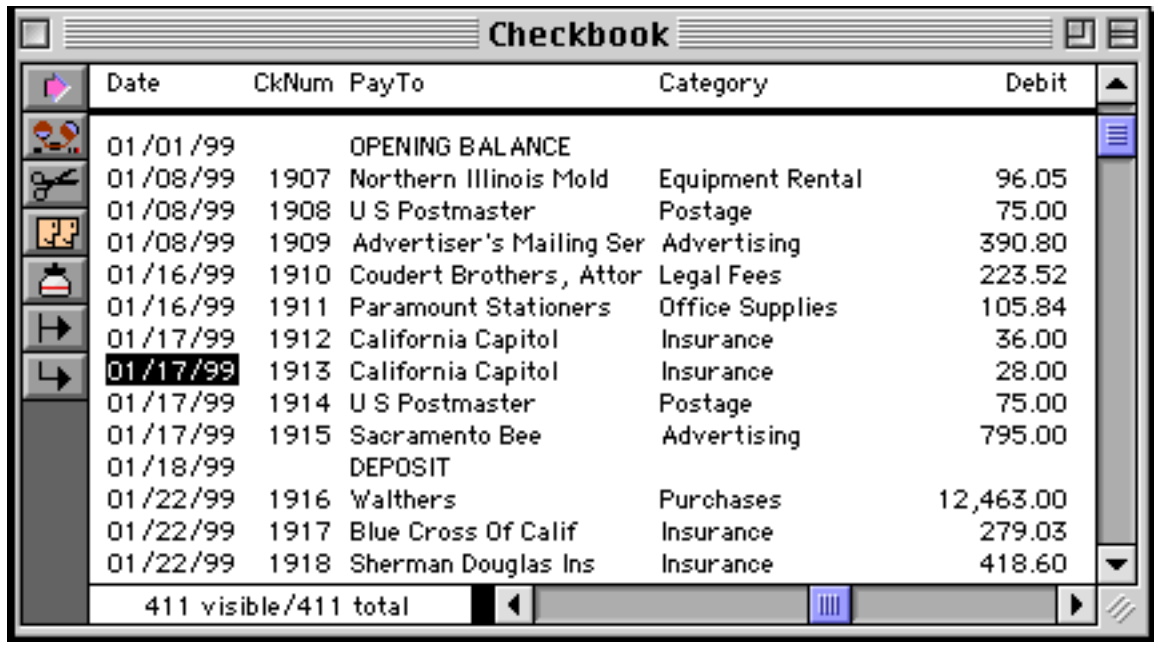


To switch to a different view within the same window, simply choose the view from the menu and release the mouse. You can flip back and forth between views at any time.

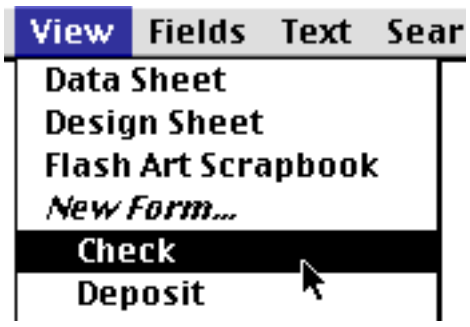
Opening More Than One Window Per Database

You can also use the view menu to open a new window, allowing you to see two views of the database at once. To open a second window the same size as the current window, hold down the **Alt** key while you select from the View Menu. (If you are using a Macintosh, hold down the **Control** key or the **Option** key.) The new window will appear slightly below and to the right of the original window.

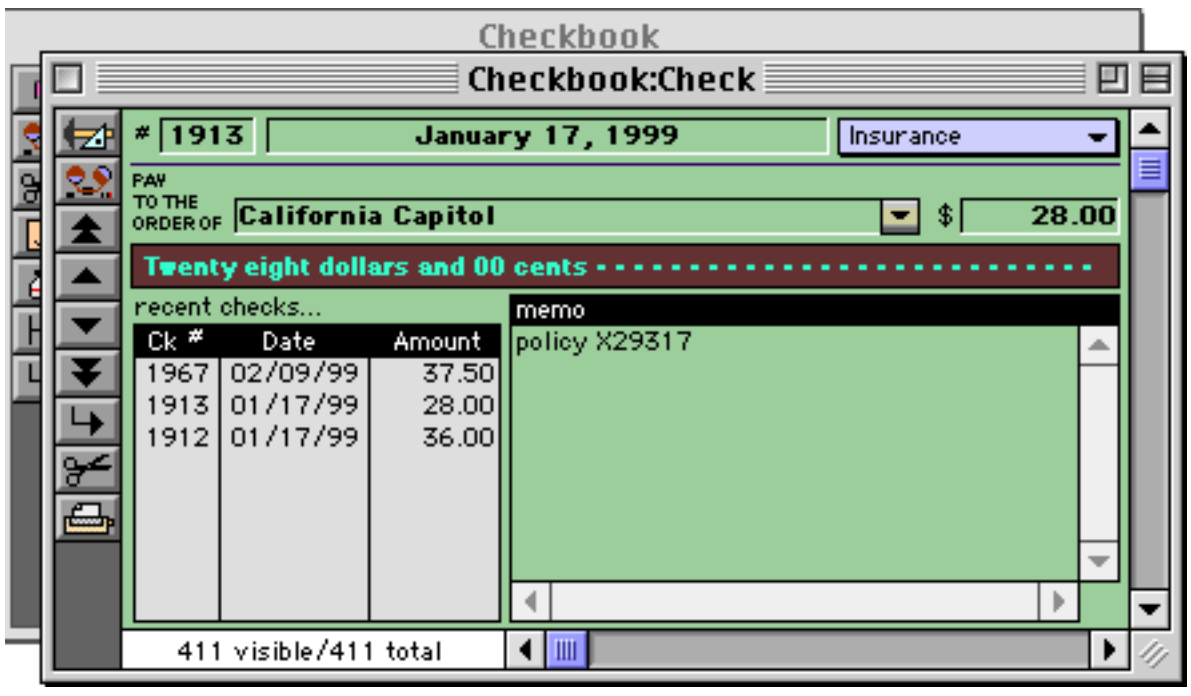
1) Start with one window



2) While holding down the Alt key (PC) or the Control key (Mac), make a selection from the View menu. On the Mac you can also use the Option key.

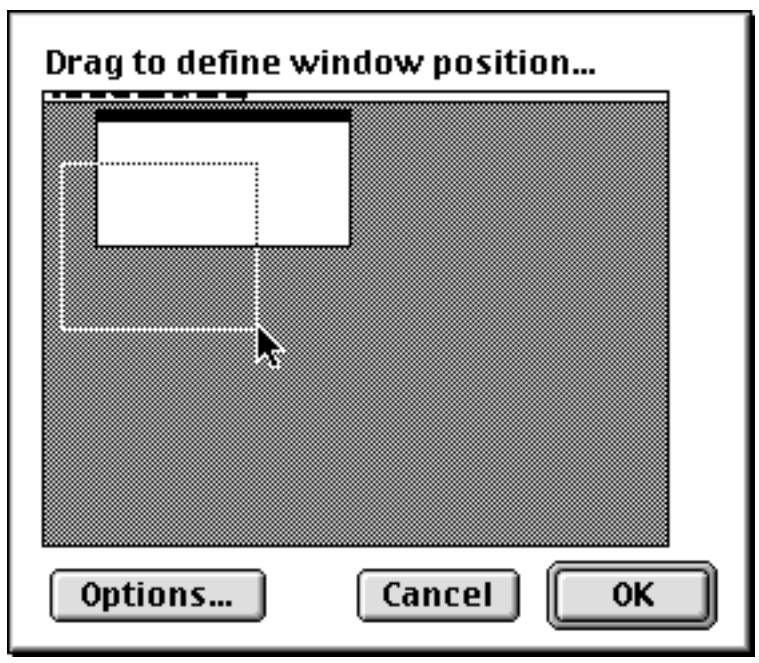


3) The new window appears sightly below and to the right...

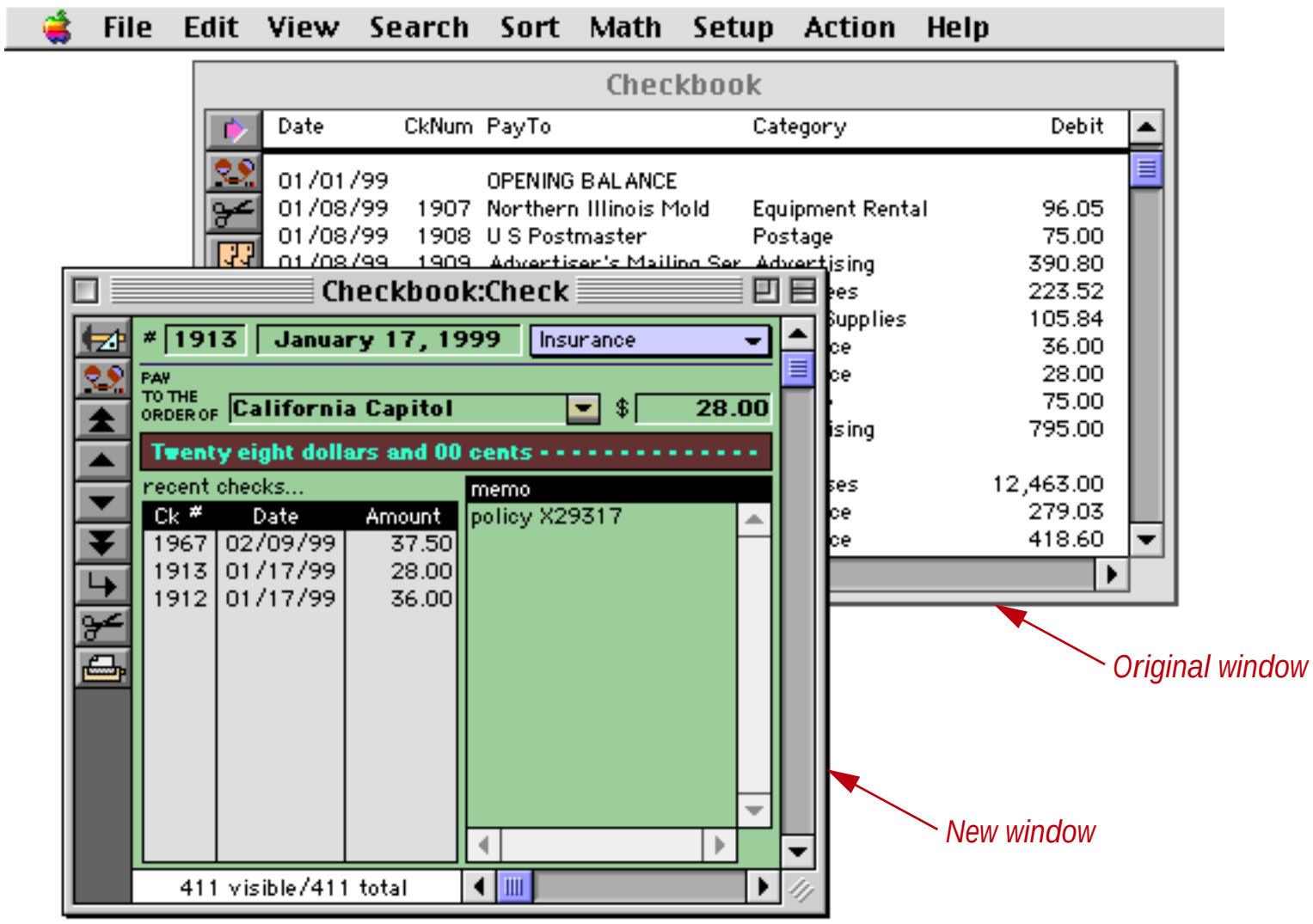


The new window will track the original window. In fact, all windows associated with a database will track each other automatically. Any changes made in one window automatically appear in all other windows, and when any navigation is done in one window (moving up or down within the database) all of the other windows will follow along.

Another technique allows you to control the exact size and position of the new window in advance. (Of course you can always drag and resize it after it has been opened.) To use this technique, hold down the Control key while you select from the View Menu. (If you are using a Macintosh, hold down the Command key.) After you choose the view you want to open, the Window Options dialog will appear shown below. This dialog shows a miniature view of the entire computer screen, along with the positions of every window.

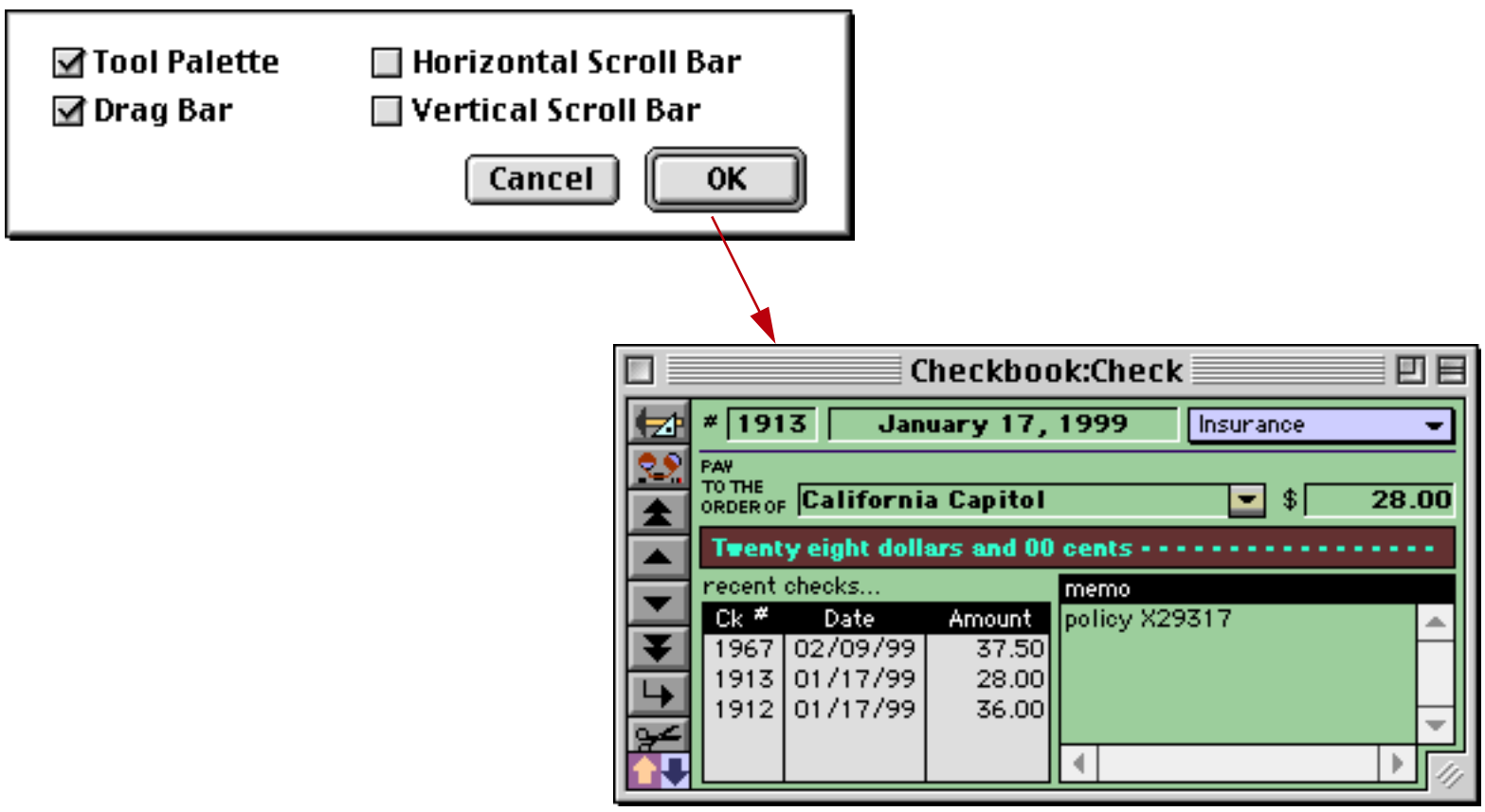


To define the position and size of the new window, simply drag a rectangle across the miniature screen, as shown in the illustration above. If you don't get the position quite right, simply drag again. (Of course you can also adjust the position and size later.) When you press the Ok button the new window will open in the location you have specified.



Window Options

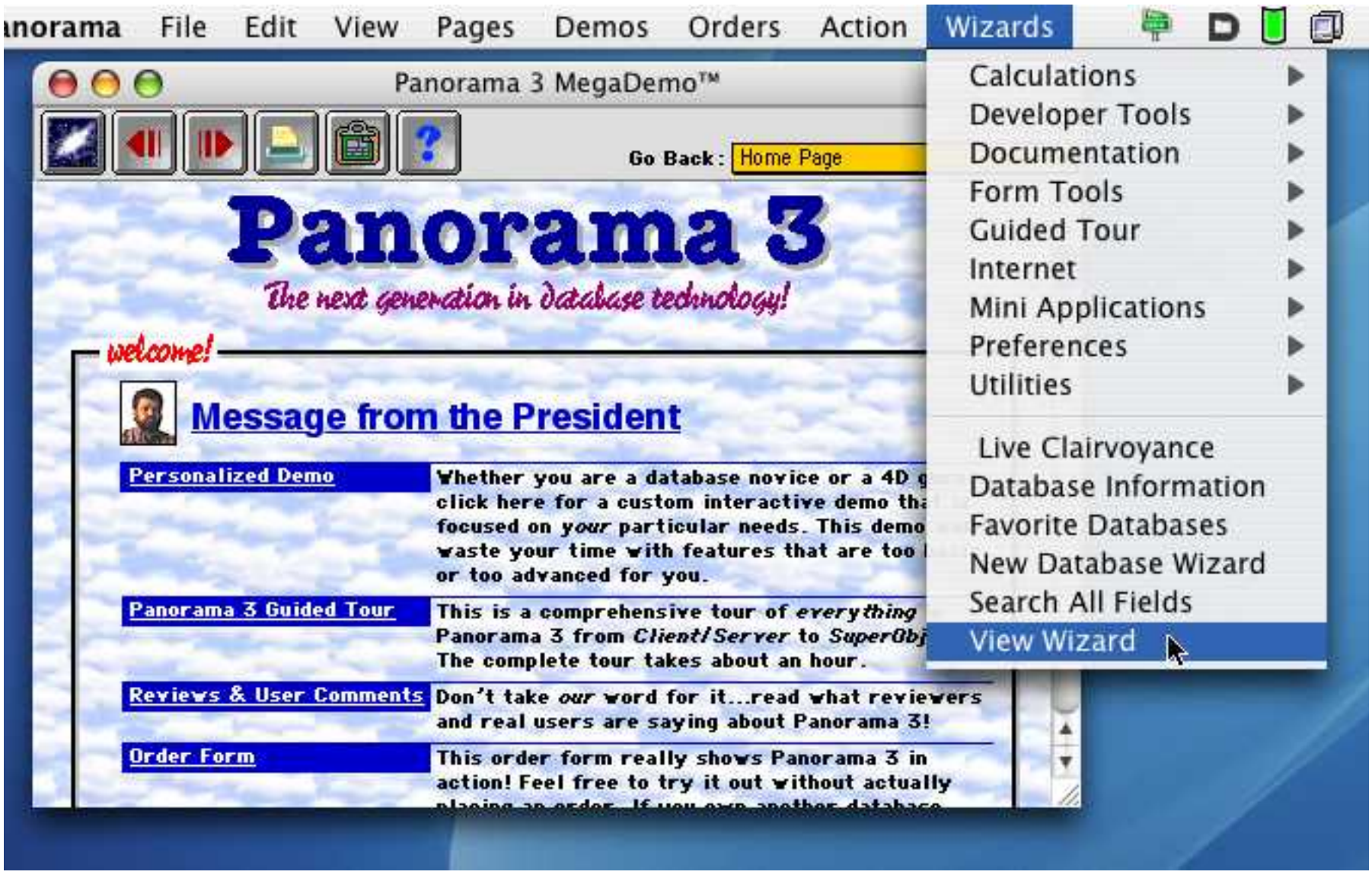
The Options button in the Window Options dialog allows you to selectively eliminate up to four components from a new window—the tool palette, scroll bars, and drag bar (you can also turn these components on and off with the **Window Tweak** wizard, see "Turning Window Components On and Off (Window Tweak Wizard)" on page 203). This illustration shows a form with the scroll bars removed.



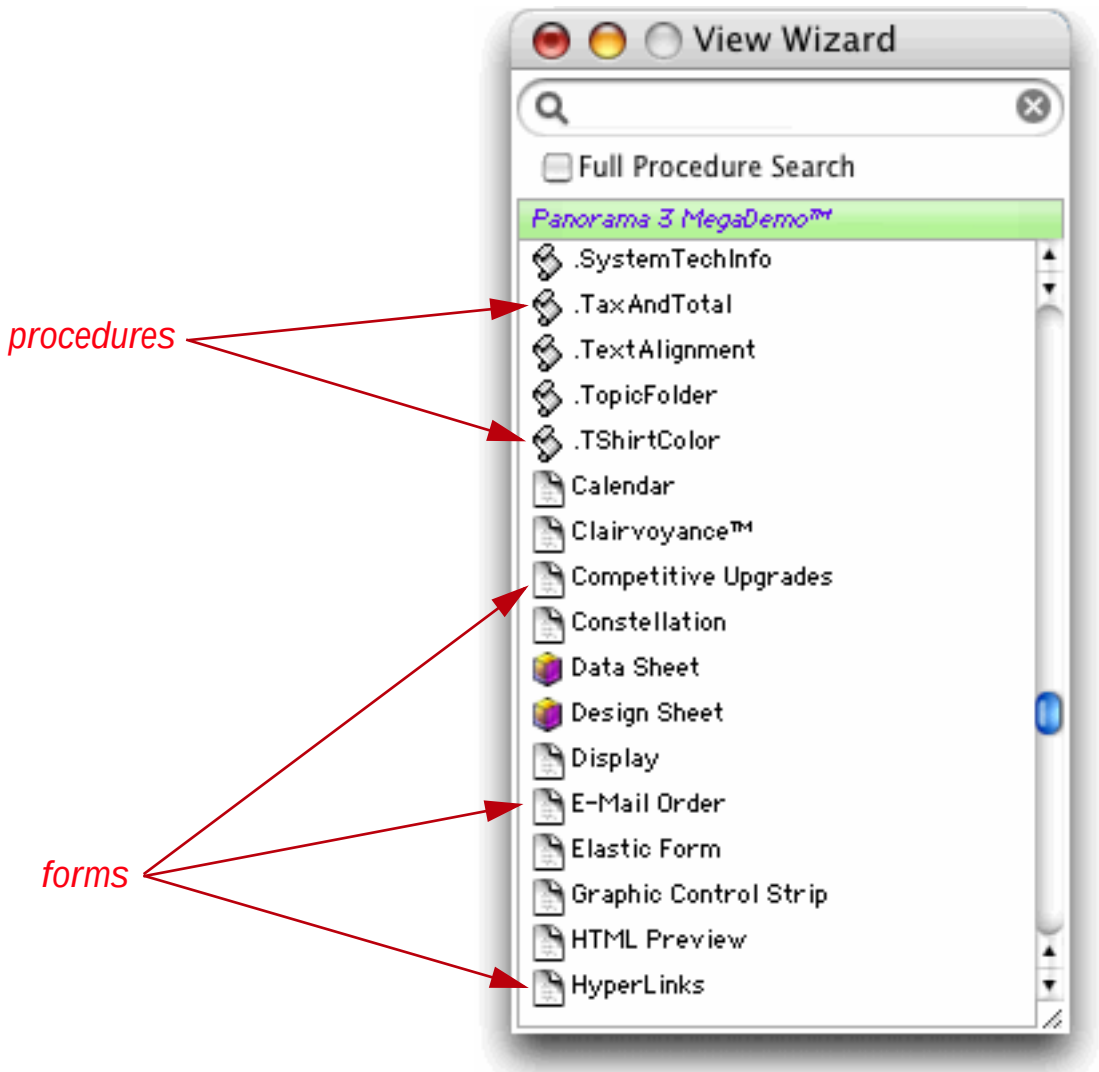
Some views will not work properly if components are eliminated—for example, you should not eliminate the vertical scroll bar from a data sheet. Be very careful if you remove the drag bar. If the drag bar is removed, the window cannot be manually moved, resized, or closed. (It can be closed by programming a procedure to close the window.)

The View Wizard

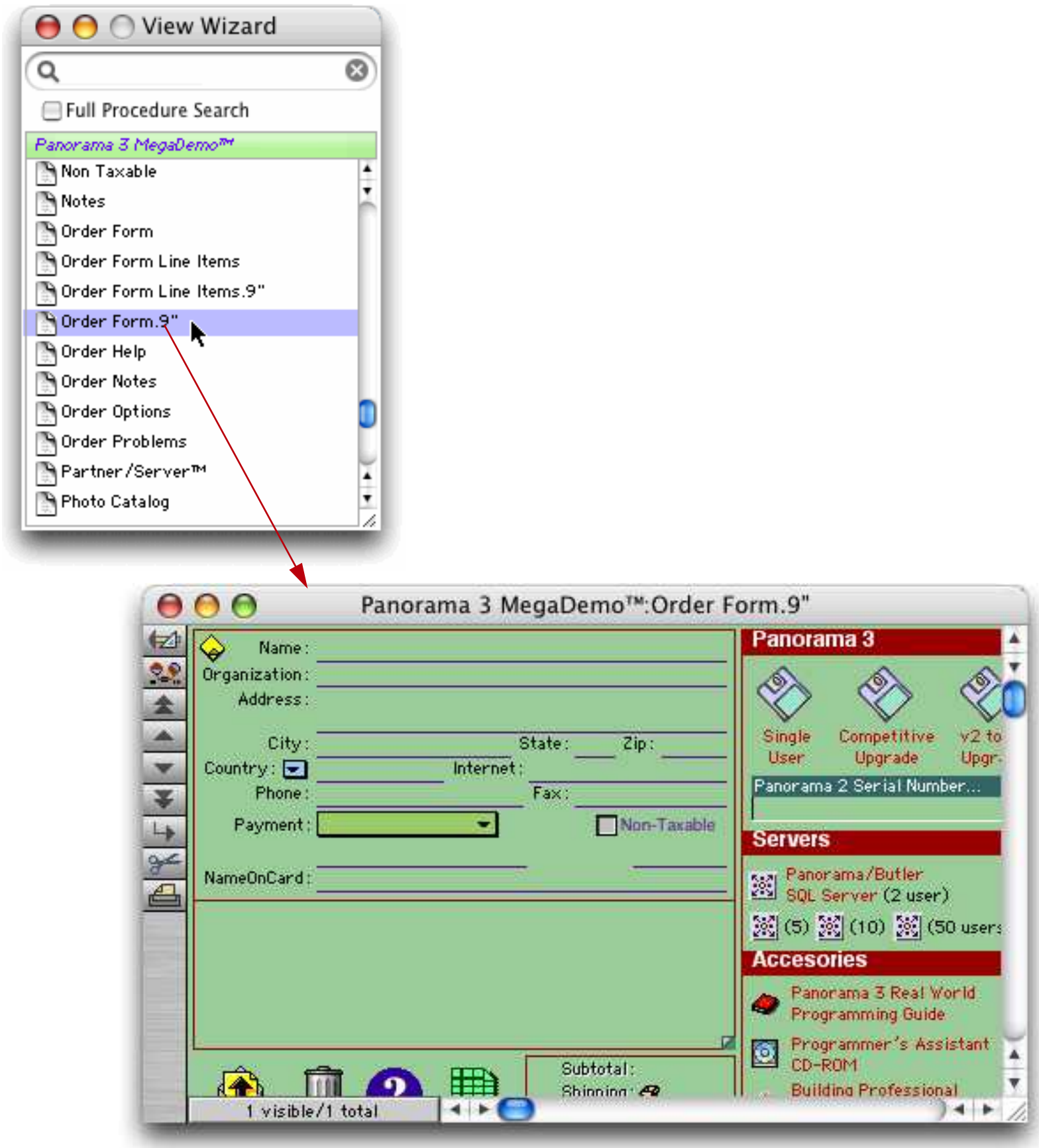
The **View** menu works well for most databases, but when a database grows to dozens of forms and hundreds of procedures it can get a bit unwieldy. For these situations the **View Wizard** comes in handy. This is a database that comes with Panorama that can help you locate and open any view. When you first open the View Wizard database it displays a list of all the forms or procedures in the currently open Panorama database. (Whether it initially displays forms or procedures depends on what type of window was open before the wizard was activated.) For example, suppose the Panorama 3 MegaDemo™ file is open.



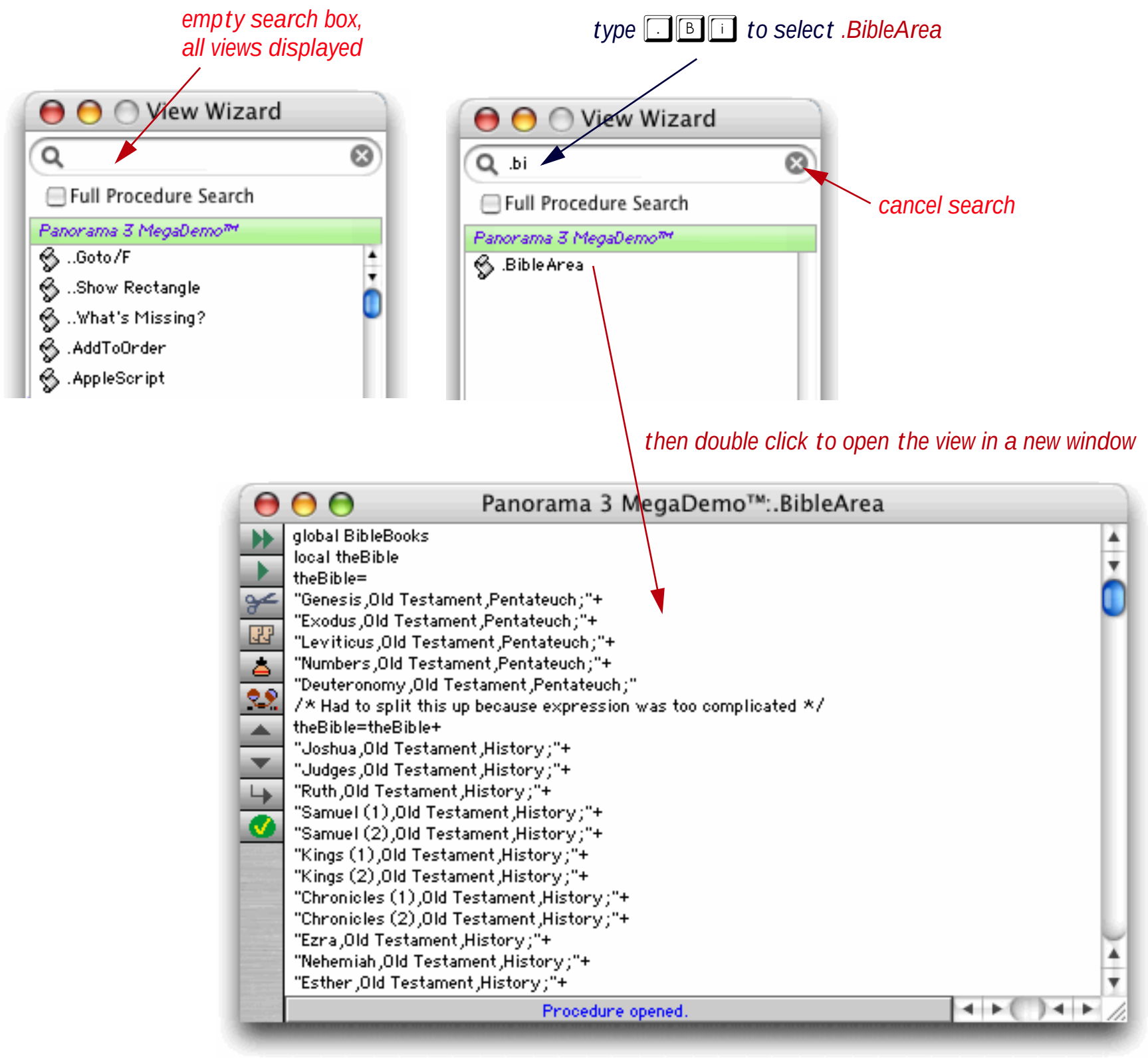
When the **View Wizard** opens it displays a list of the views in this database (Panorama 3 MegaDemo), including forms, procedures, crosstabs, the data sheet and the design sheet. In the example below the procedures and forms appear to be grouped together, but that is just a coincidence — the views are listed in alphabetical order.



Double click to open any view.



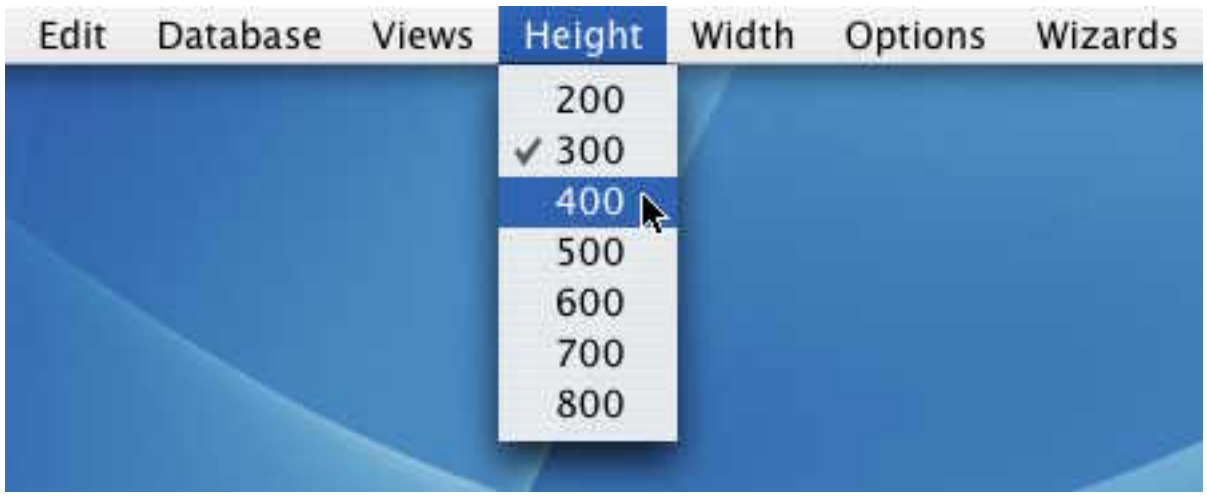
If you know the name of the view you want to open you can find it quickly by typing in the first few letters. As you type each letter, the wizard will show you the views that match that name.



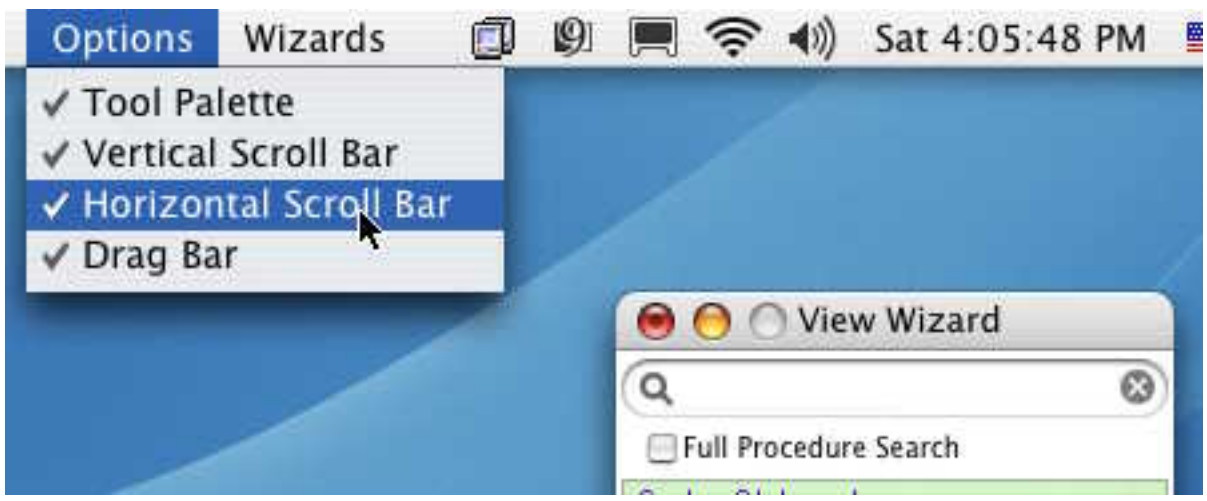
Use the **Database** menu to view forms, procedures or crosstabs in a different open database.

View Wizard Window Size and Options

The **Height**, **Width** and **Options** menus allow you to control the size and options of the new window. The **Height** and **Width** menus specify the dimensions of the new window in pixels (72 pixels per inch).



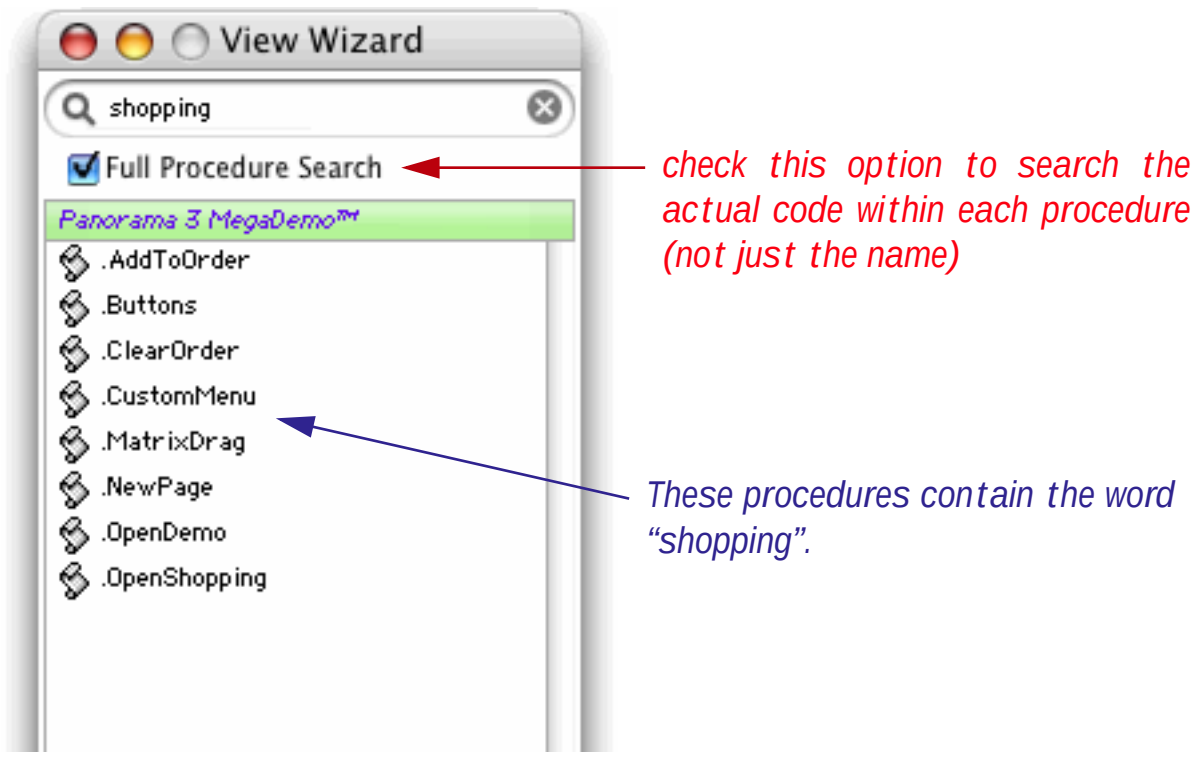
If you are opening a form view you can also use the **Options** menu to control whether or not the tool palette, scroll bars and drag bar appear in the new window. (These options are ignored when opening other kinds of windows.)



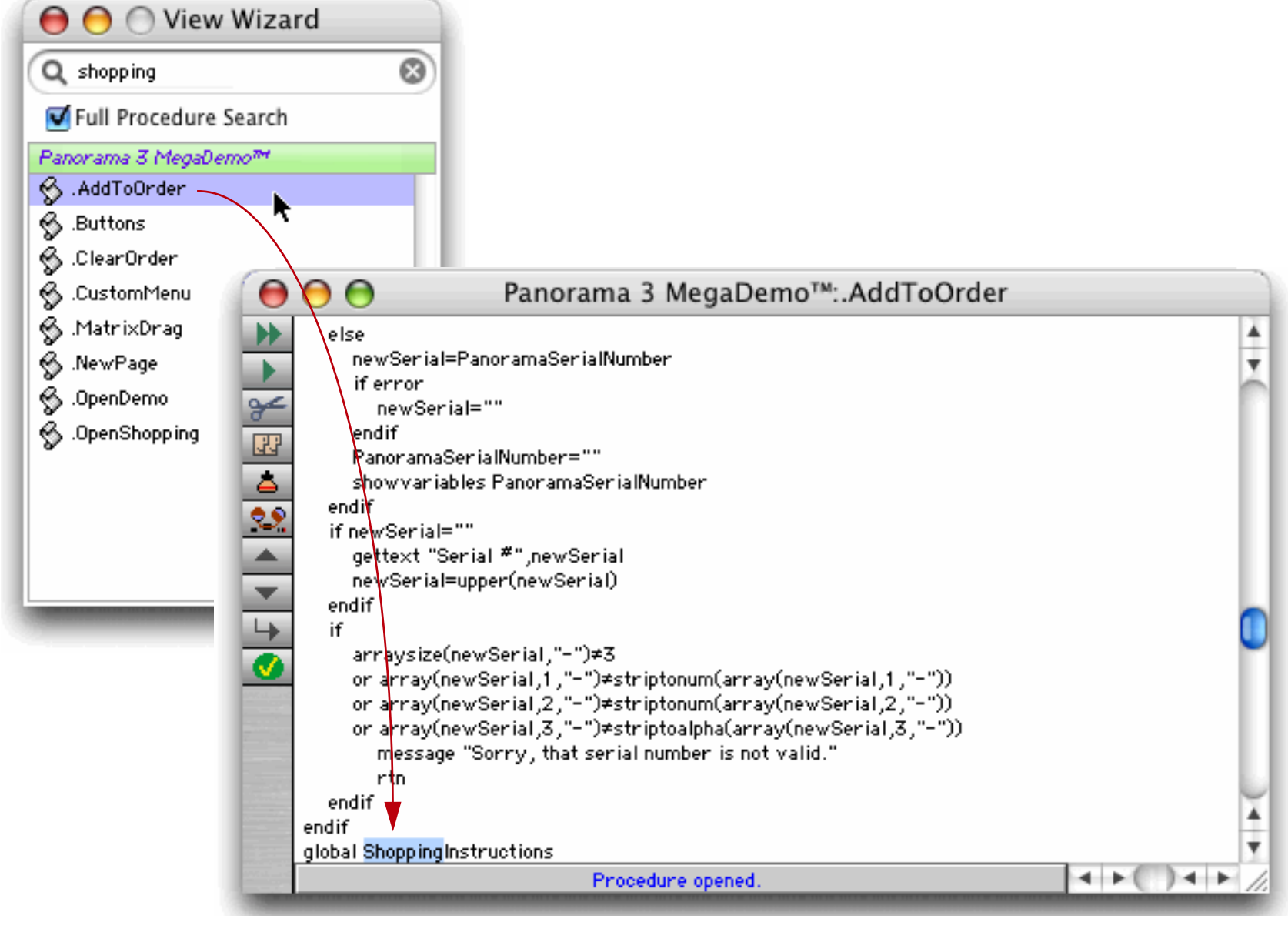
Remember that you can also adjust a form window's size and options after you have opened it with the **Window Tweak** wizard, see "Turning Window Components On and Off (Window Tweak Wizard)" on page 203.)

Searching All Procedures

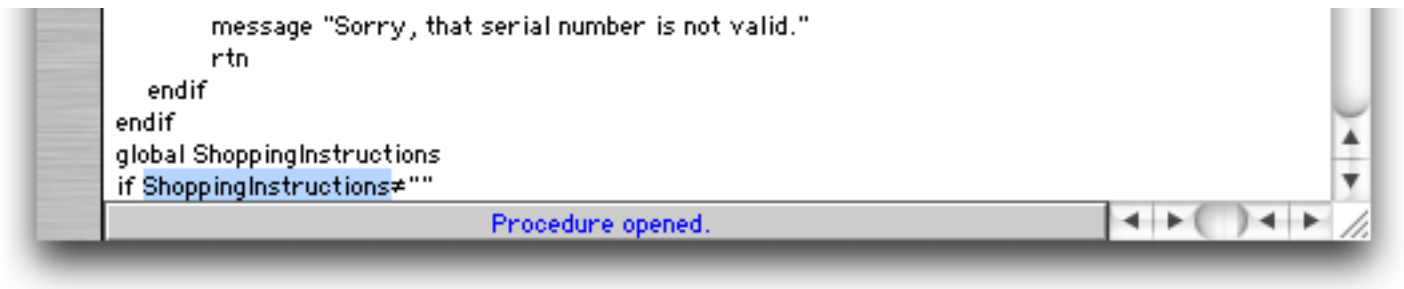
The **View Wizard** has the capability of searching the text of all procedures in a database. Simply check the Full Procedure Search option and type in the word or phrase you want to search for. The list will update as you type each key.



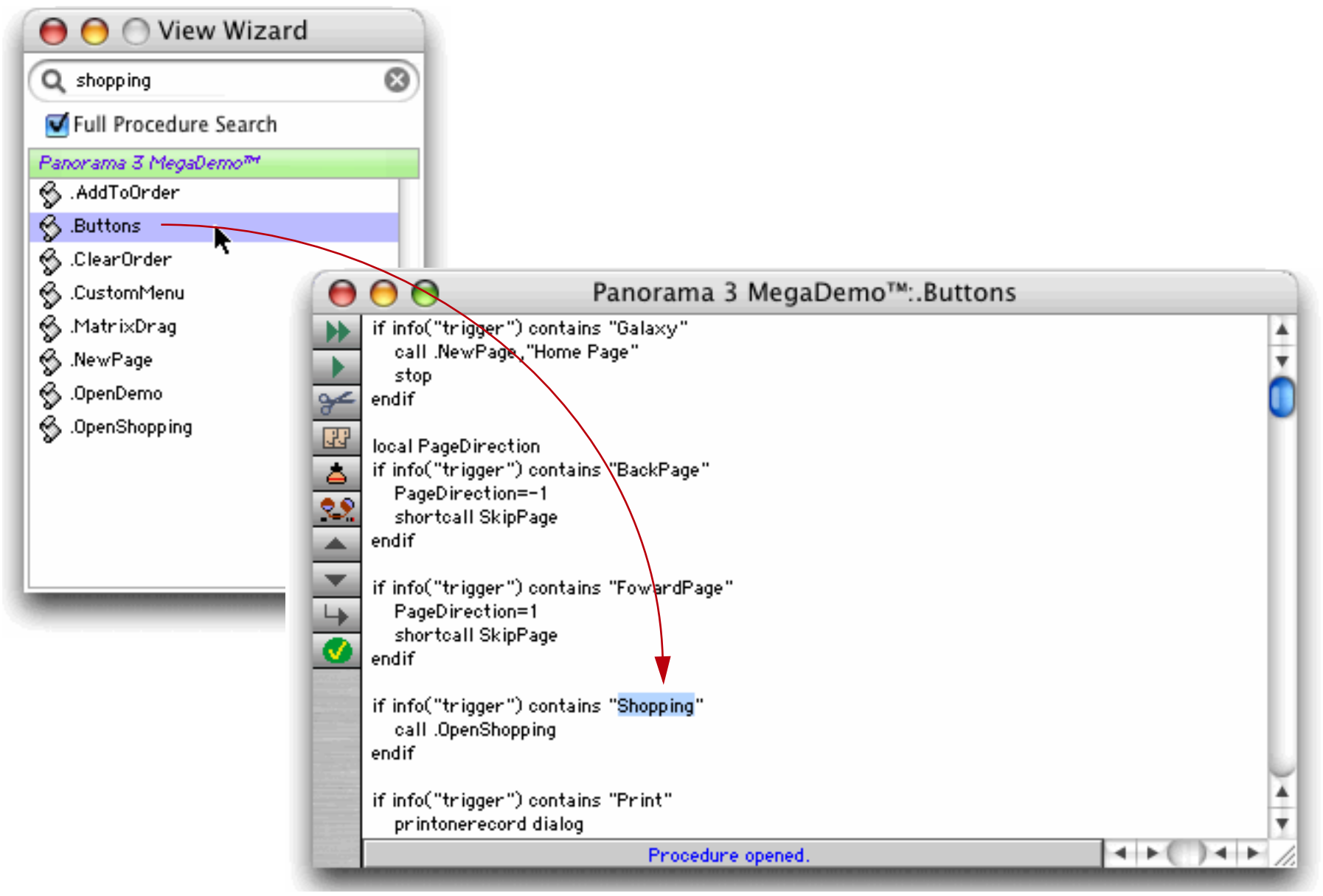
When you double click on one of these procedures the wizard will open the procedure window and automatically locate the first occurrence of the word or phrase.



Choose **Find Next** from the Search menu to find the next occurrence of this word or phrase within the procedure (if any).



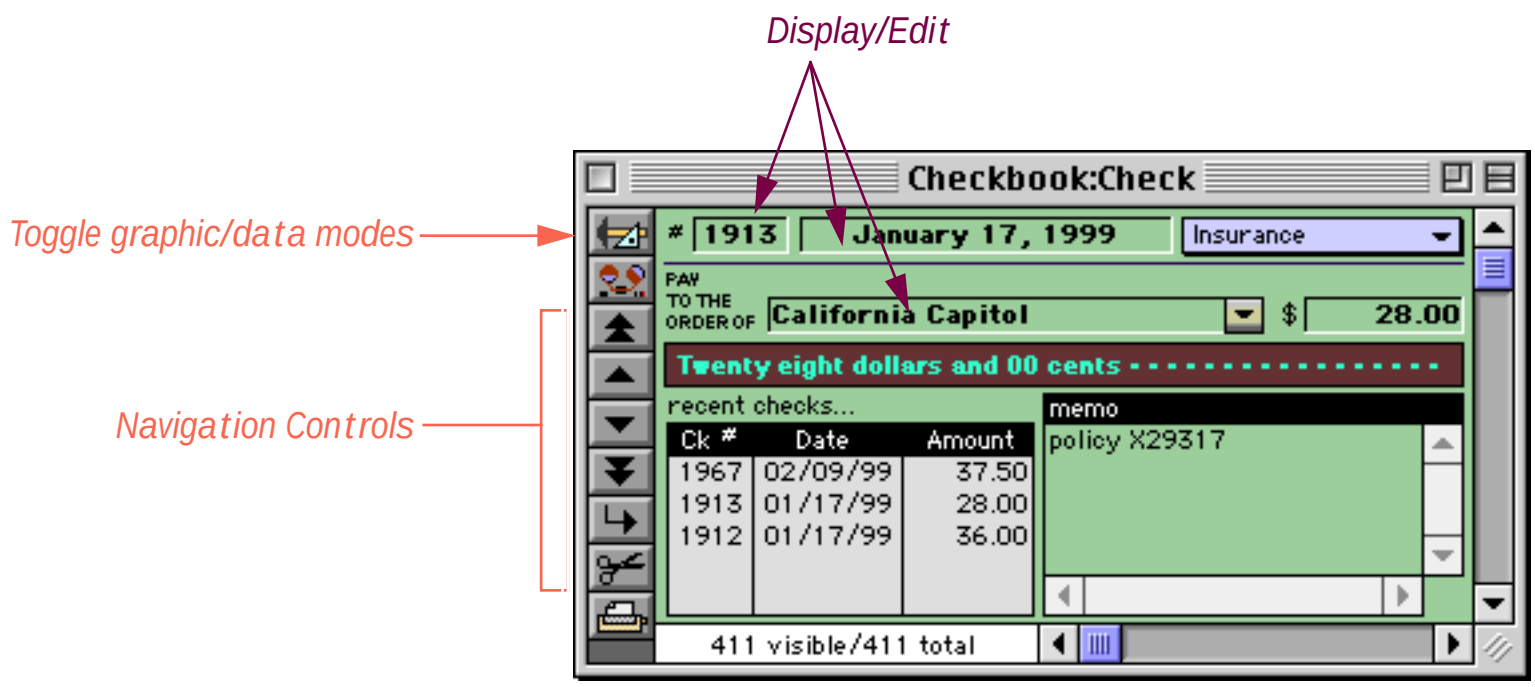
You can repeat using the **Find Next** command until you have located every occurrence of the word or phrase in this procedure. At that point you'll need to go back to the **View Wizard** to continue with the next procedure.



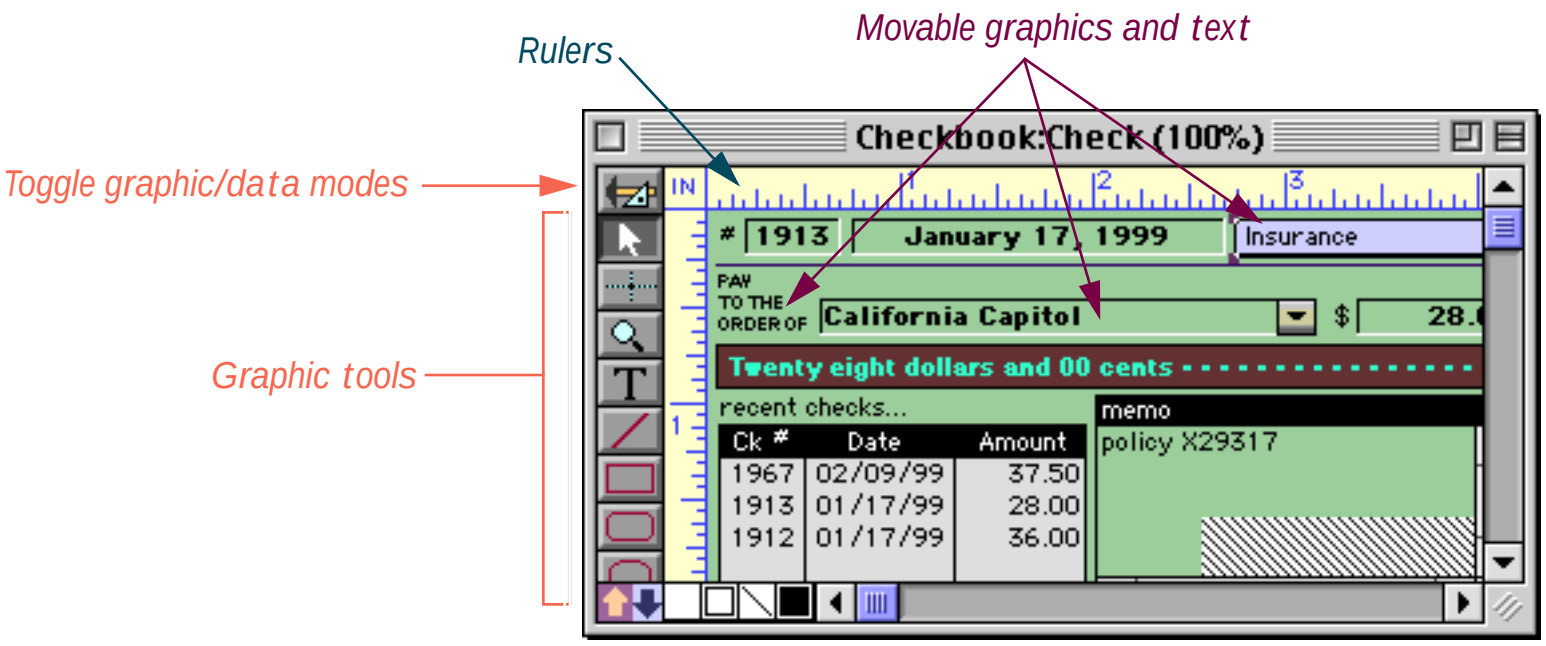
You can continue this process until you have located every occurrence of the word or phrase in the database.

Form Modes: Data Access vs. Graphic Design

Unlike other views, the Form View operates in two distinct modes—data access and graphic design. Data access mode (also called "data mode") is the default mode. In this mode you can view and display data, and navigate through the database.



Graphic design mode (also called "graphics mode") functions like an electronic drafting table. In this mode you design the form by drawing lines, boxes, and other graphic elements. This mode is very similar to many drawing and page layout programs. Graphic design mode is easily recognized by the rulers that appear at the top and left edges of the windows.

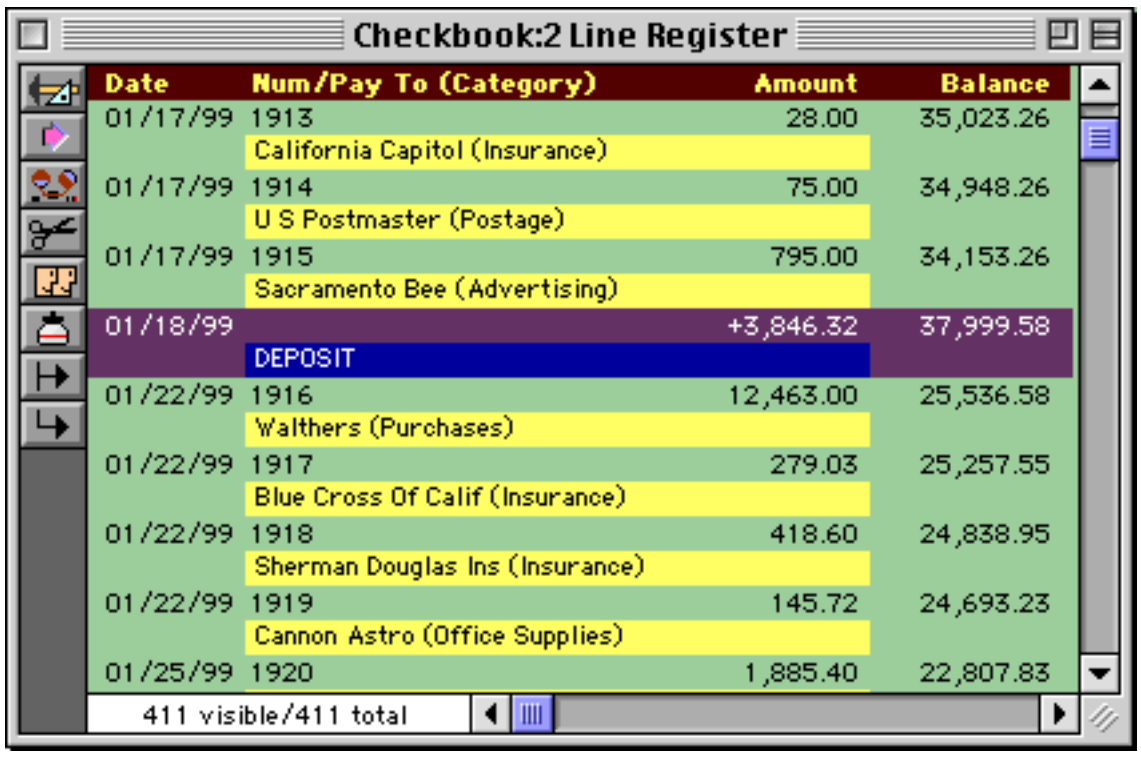


To switch between data access and graphic design modes, click on the  tool. Each click on this tool toggles the window between the two modes.

Form Operation: Individual Pages vs.View-As-List

Panorama allows you to set up blank forms as individual pages or as a continuous sheet (view-as-list). When forms are set up as individual pages you see one record at a time. You can flip through the records just as you would shuffle through a stack of paper forms. All of the examples of forms you've seen so far are individual page forms.

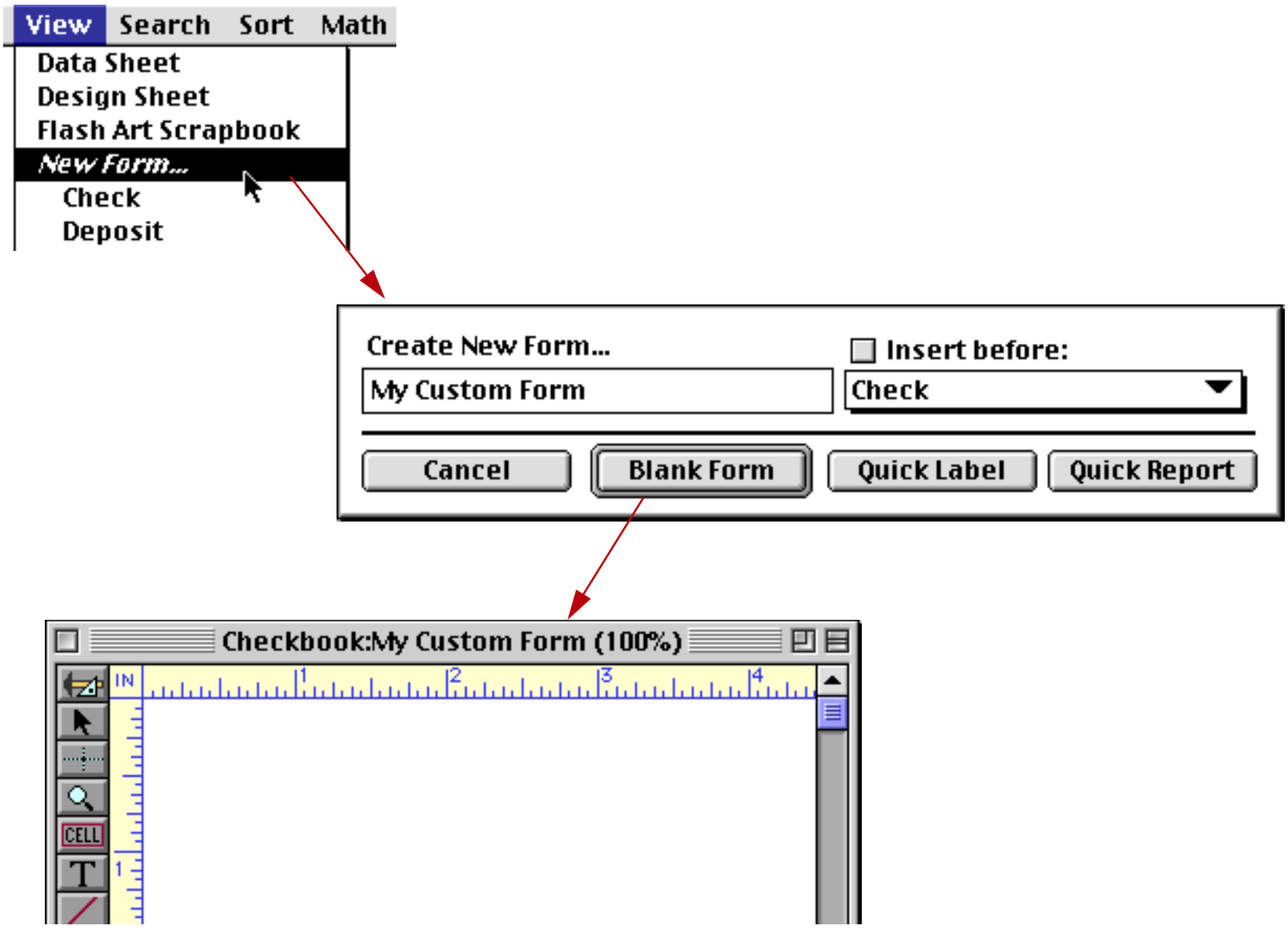
A view-as-list form displays data as a continuous sheet, as shown below.Instead of flipping from record to record, you scroll up and down through the data in a manner similar to the data sheet. However, unlike the data sheet, a view-as-list form allows you to arrange the data any way you like, and even include graphics in the display. On the other hand, view-as-list forms are slower than the data sheet (because of the overhead in displaying the graphics) and they are much more work to set up.



Unless you tell it otherwise, Panorama sets up a new form as individual pages. To convert the form to a continuous sheet you must use the **Form Preferences** command (Setup menu) to set the View-as-List option. You will also have to define the boundaries of the form by setting up a data tile (and optional header tile). For more information about setting up view-as-list forms see "View-As-List Forms" on page 889.

Creating a New Form, Crosstab or Procedure

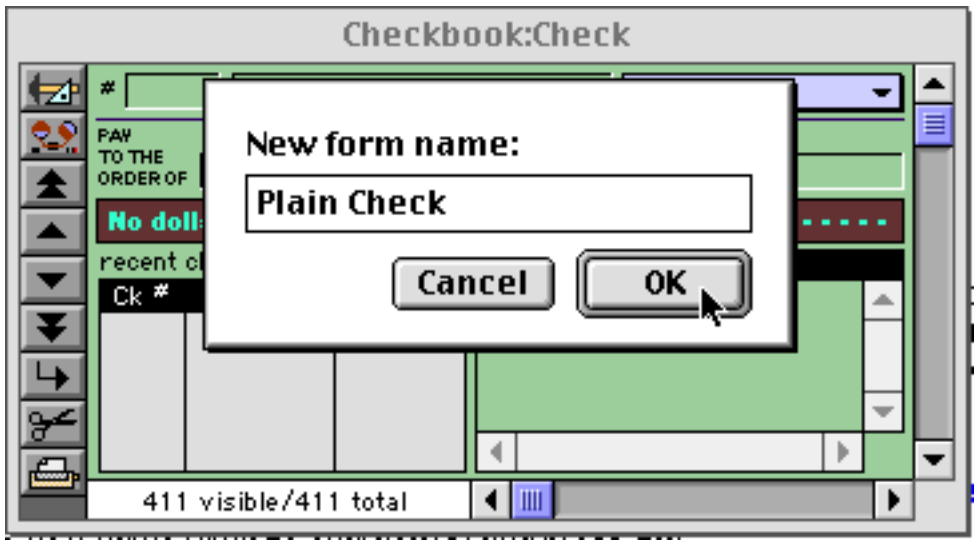
To create a new view, choose **New Form**, **New Crosstab**, or **New Procedure** from the View Menu. A dialog box will appear asking you to name the new view. A view name may be up to 25 characters long and can contain any letter, number or punctuation.



When you create a new view, it is usually added to the end of the appropriate section in the View Menu. For example, a new form usually becomes the last form in the View Menu. If you wish, you can insert the new view into the middle of the View Menu. To do this, check the Insert before button and use the pop-up menu directly below the Insert before button to specify the position of the new view. You can also re-arrange the order of the views using the **Re-Arrange** command in the Setup menu.

Renaming a Form, Crosstab or Procedure

To rename the currently visible form, crosstab or procedure choose **Rename Form**, **Rename Crosstab**, or **Rename Procedure** from the Setup Menu. Type in the new name (limit 25 characters) and press Ok.



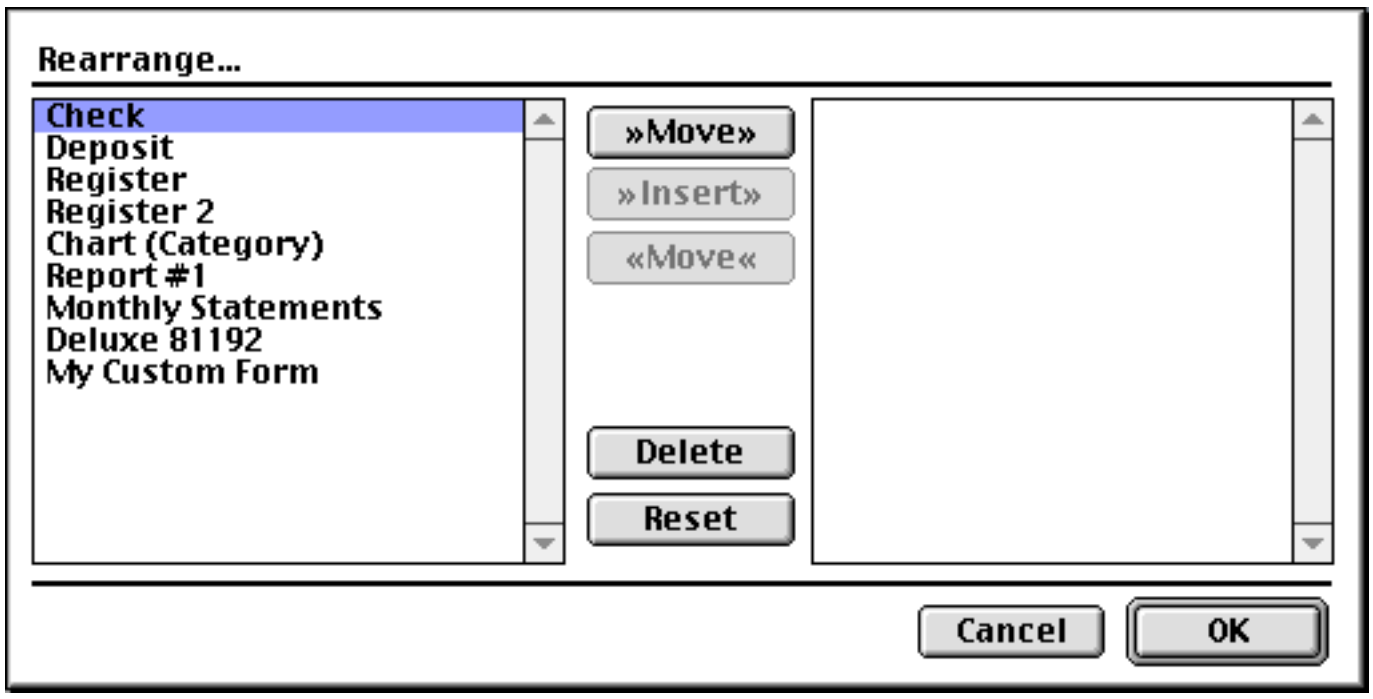
Deleting a Form, Crosstab or Procedure

To delete a form, crosstab, or procedure choose **Delete Form**, **Delete Crosstab**, or **Delete Procedure** from the Setup Menu. Since you cannot undo after you delete a view, Panorama will ask you if you are sure before it actually deletes the view. Note: If only one view is open and you remove it, Panorama will close the entire file. To avoid this, open an extra window before you delete a view.

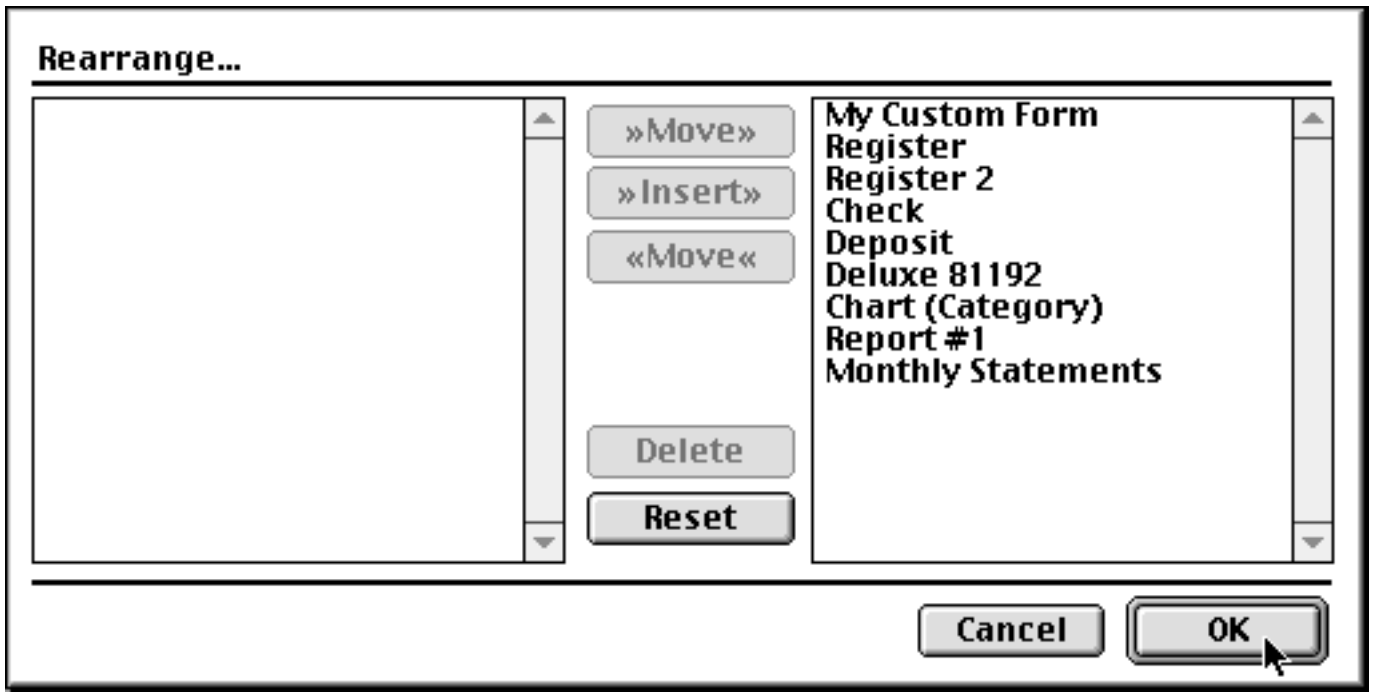
You can also delete views with the **Re-Arrange** command in the Setup menu (described in the next section). The **Re-Arrange** command is the fastest way to remove several views at once (see below).

Changing the Order of Forms, Crosstabs or Procedures

The **Re-Arrange Forms**, **Re-Arrange Crosstabs**, and **Re-Arrange Procedures** commands in the Setup menu change the order of the items listed in the View Menu. Each of these commands displays a dialog box listing the current view order on the left and the new view order on the right.



This dialog is like a puzzle—the object is to move the views from the left to the right in the order you want. To move a view to the other side, either double-click on it or press the >>Move>> button. To insert a view into the middle of the list on the right, first select the spot where you want to insert and then press the >>Insert>> button. When all the views have been moved to the right hand side, press the Ok button to rearrange the views.

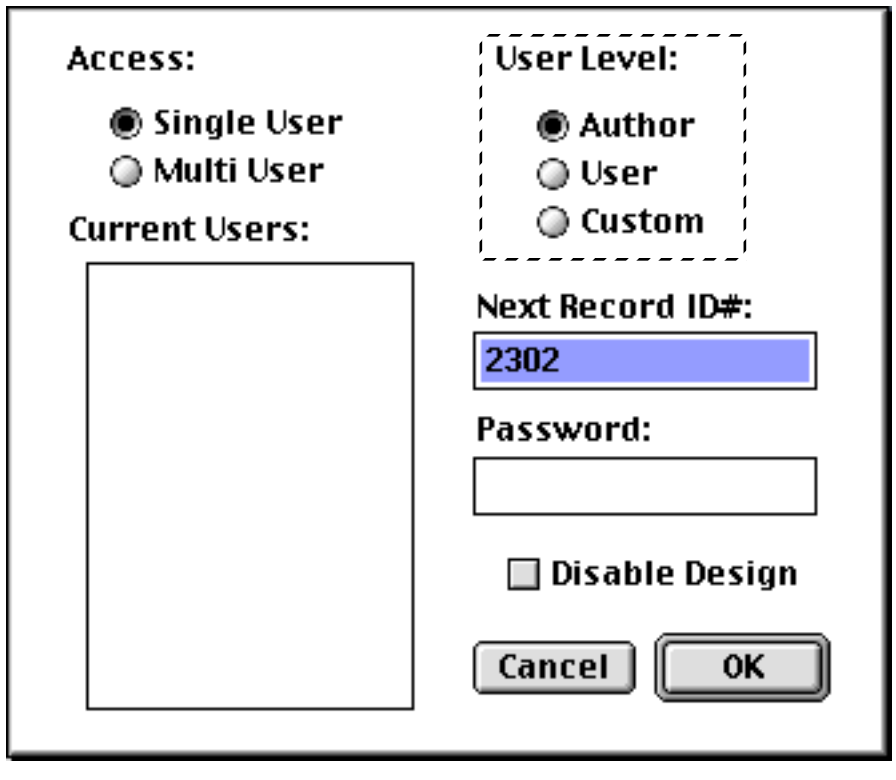


You can also use this dialog to delete views. To delete one or more views, press the Delete button instead of moving the view to the right. Warning: You cannot delete an open view using the **Re-Arrange** command. First close the view (form, crosstab or procedure) and then delete the view.

The Privilege Dialog

The **View** menu normally gives you complete freedom to access any view in your database. If other people are going to be using your database you may want to prevent them from switching to unauthorized views.

To restrict user access to the **View** menu, use the Privilege dialog (shown below). To open this dialog on a Macintosh computer, hold down the Command or Option key and choose **About Panorama** from the Apple menu. To open this dialog on a Windows system, hold down the Control or Alt key and choose **About Panorama** from the Help menu. This dialog allows you to choose one of three possible user levels for the current database: Author, User, and Custom.



The Author level places no restrictions on the user—he or she can change the database design, draw graphics, open and close windows, and generally perform any Panorama operation described in this manual.

The User level allows the user to perform data entry and analysis, but prevents the user from changing the database design or changing the graphics in a form. User level disables the **View** menu, so the user cannot choose which view he or she wants to use. They can only use views that are opened automatically when the database is opened or views that are opened with procedures set up in advance.

The Custom level is even more restrictive. All of the tools in the tool palette disappear, along with all of the menus except for the **File**, **Edit** and **Action** menus. When the database is locked to the custom level, the user can perform data entry and predefined programs (procedures)—and that's it. Everything else is forbidden.

To prevent unauthorized users from changing the user level, the Privilege dialog can be protected with a password. Once the password is set, you cannot open the Privilege dialog unless you know the password.

User Levels vs. Save Window Positions

If the user level is set to User or Custom the **Save** command will not save the new window positions—even if the Save Window Positions option is turned on. The window positions are only saved when the file is at the Author level. This allows the database author to reliably set up the initial views and window positions.

Hiding Sensitive Data

Since the User and Custom levels prevent the user from using the **View** menu, it is possible to hide fields containing sensitive information from most users of a database. To do this:

1) Create a form that shows only the non-sensitive data.
2) Close all of the other windows associated with this database.
3) Save the database with the Save Window Positions option checked.
4) Set the user level to User or Custom (you will probably want to set a password at this time also)
5) Save the file again.

The next time this database is opened, only the form displaying the non-sensitive data will appear. Since the database is locked, the user cannot open any of the views that contain sensitive information. Users who know the password can use the **Privileges** dialog to switch to Author level, or you can set up a procedure that switches views if the user knows the correct password.

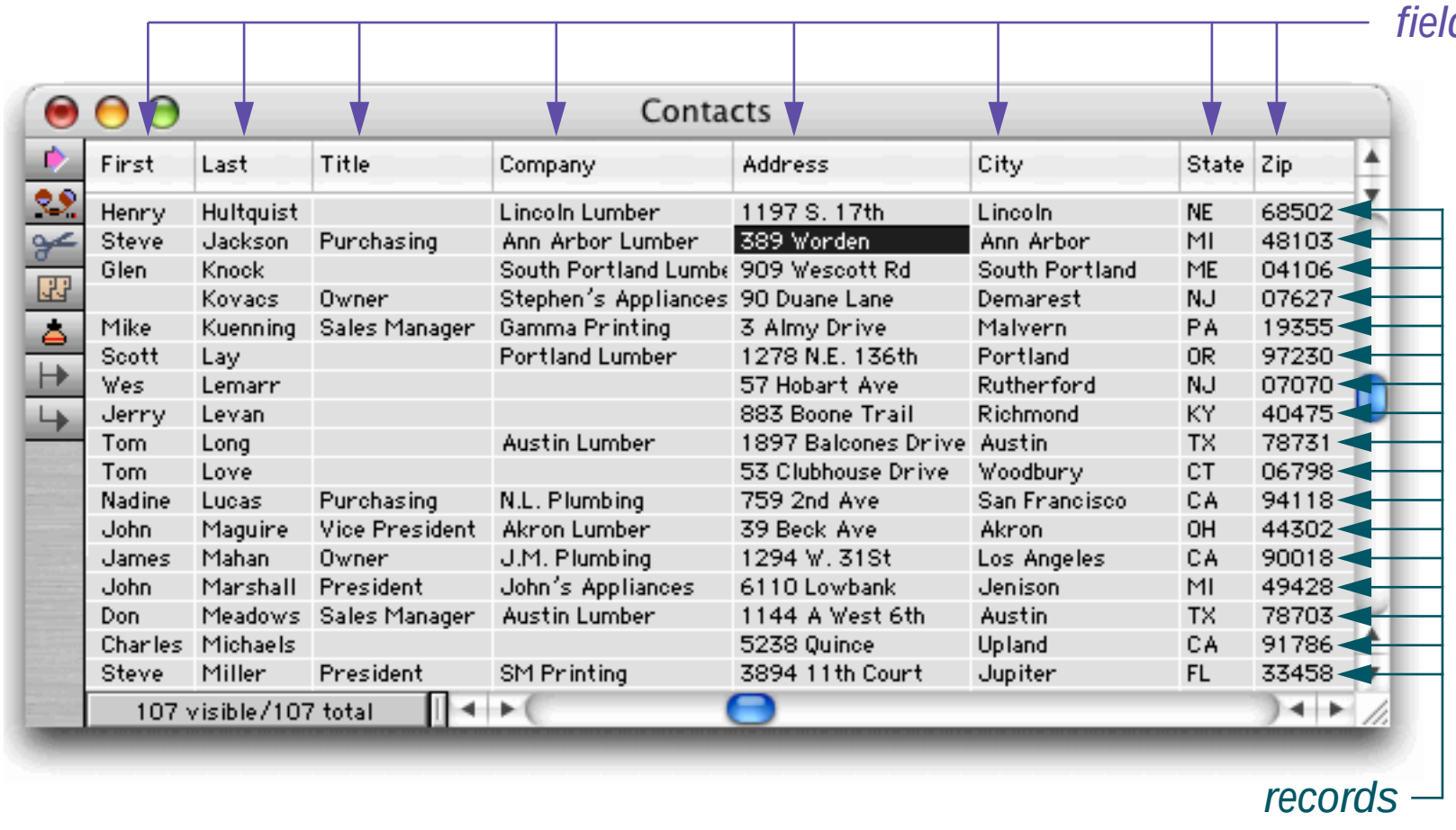
Chapter 4: Records



The heart of a database is, naturally, the data stored in it. Since storing and organizing data is Panorama's primary task, it has special rules and procedures for handling data.

Data Organization

Inside each Panorama database the information is organized into records and fields. A record consists of a group of related information. In a personnel database, for example, each record would contain all the information about a single employee. Most databases have anywhere from several dozen to several thousand individual records. The example database shown below has 102 records, 17 of which are currently visible in the window.



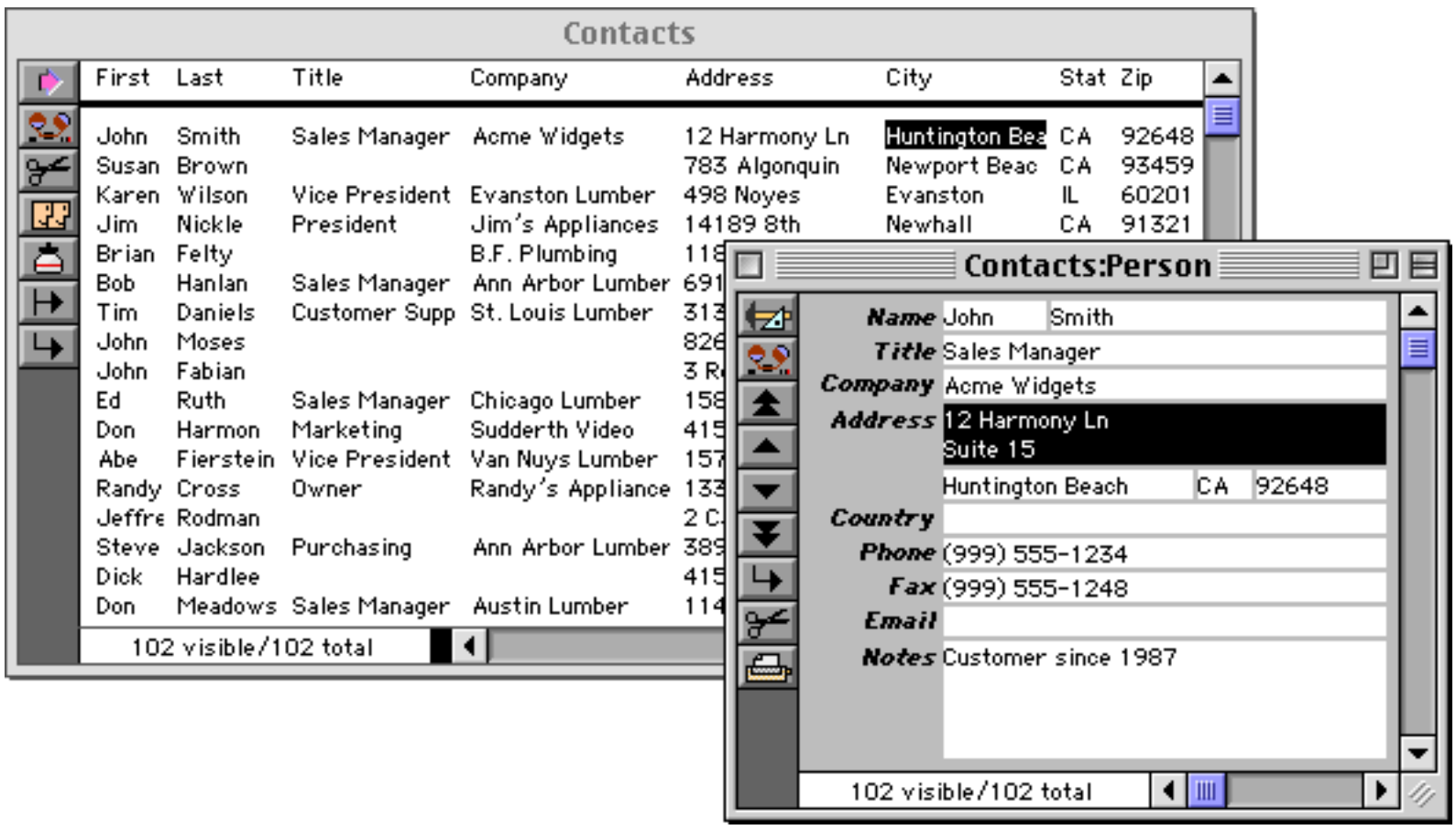
The database is also divided into fields. Each field contains a specific item of information—a street address, a phone number, a birthdate, etc. Most databases have somewhere between five and one-hundred fields (Panorama allows up to 65,000 fields per database).

Every record in a database contains exactly the same fields. If certain records don't use a particular field it can be left empty, but the field itself still exists for every record. For example, notice that in the illustration above some of the Title and Company entries are empty. (However, you can create a form that does not display all of the fields. See "Displaying and Editing Text" on page 587.)

As you work with a database, you will constantly be adding new records, revising and removing old records, and rearranging (sorting, etc.) existing records. Fields can also be added, revised, and removed, but you will do this much less often. Once the fields are set up, you will usually leave them alone.

Tables vs. Individual Pages

Panorama can display any database either as a table of records or as an individual page for each record. In the table format each line corresponds to a record, and each column corresponds to a field. Common examples include phone directories and price lists. In the individual page format each page corresponds to a single record. Each item on the page corresponds to a field. Common examples include invoices, tax returns, and report cards.



Panorama doesn't care whether the database is displayed as a table or as individual pages. As you can see, the same information is displayed and edited either way. Use the method that seems natural for the database you are working with.

Special Records

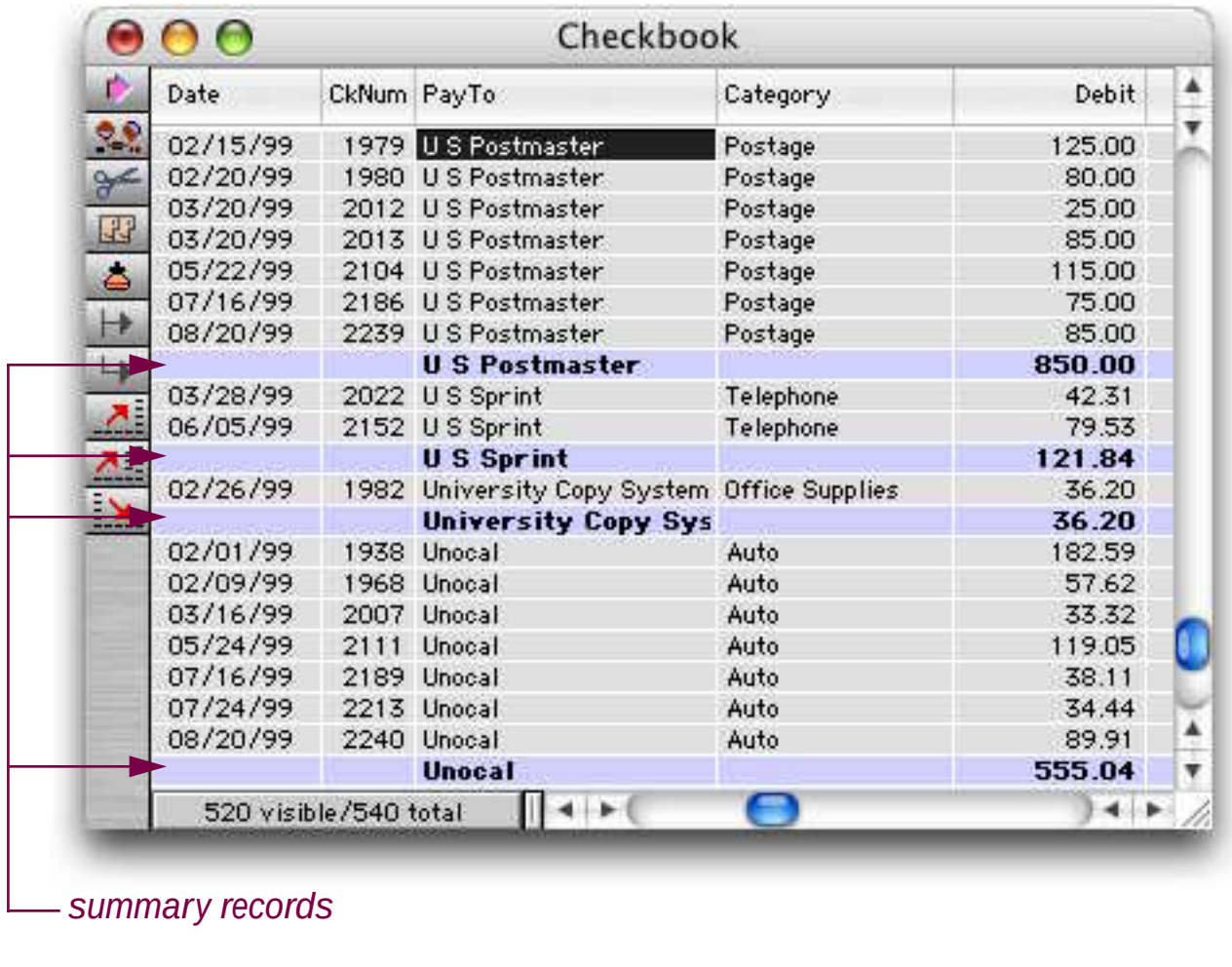
Most database programs treat every record the same way. Panorama, however, distinguishes between three different types of records: **data records**, **summary records**, and **invisible records**. Most of the work you do will be with ordinary data records. Summary and invisible records, however, are the keys to some of Panorama's unique capabilities.

Data Records

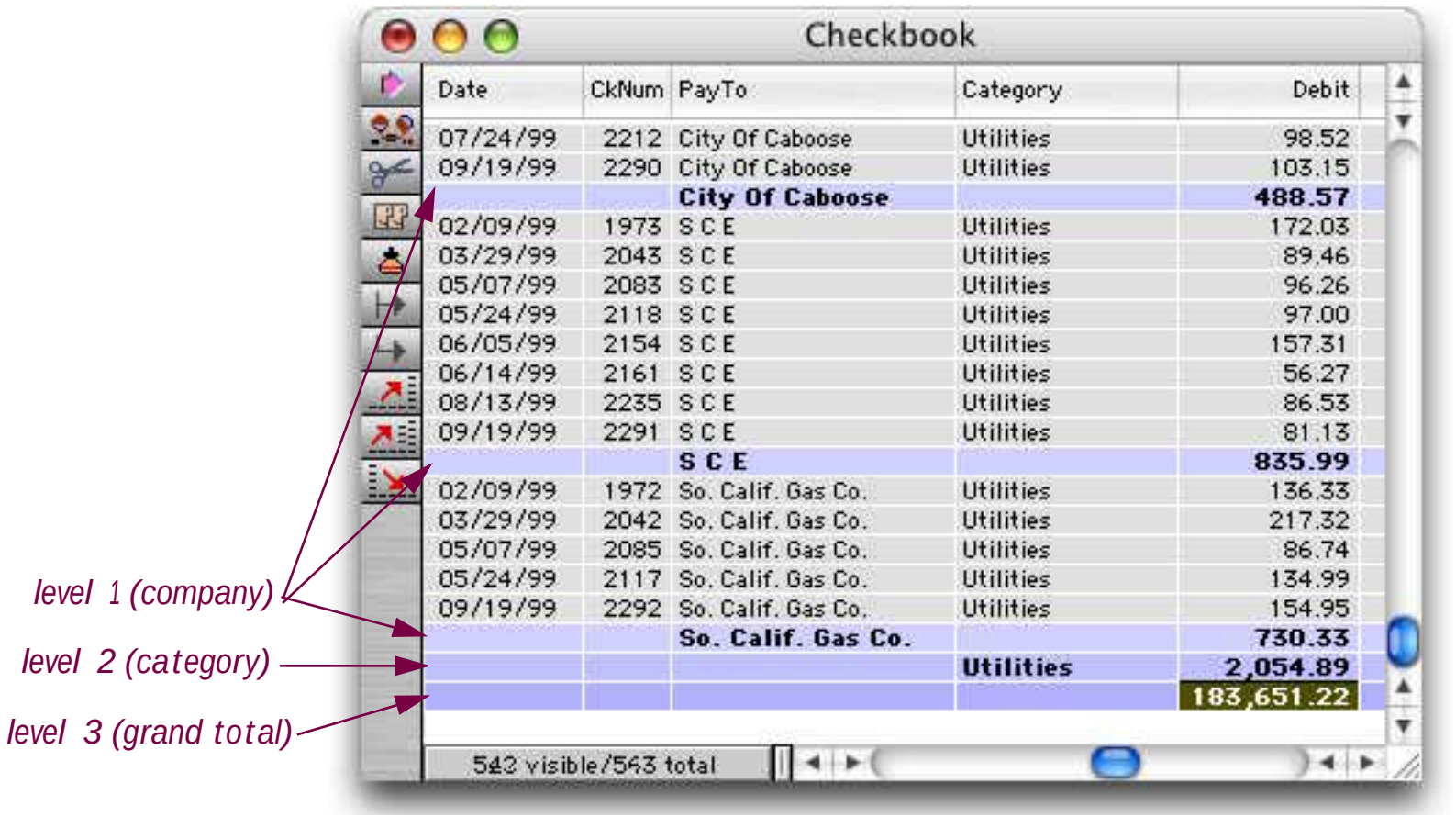
Ordinary records used for data storage are called **data records**. You can create new data records one at a time by keying in information, or you can add many new data records at once by importing data.

Summary Records

Summary records are temporary records used for calculating totals, subtotals, and other summary information. Panorama's **Group** commands automatically create summary records for you. When viewing the database as a sheet you can identify summary records by the light blue background color, and by the fact that they are usually displayed in bold.



Summary records may appear in one of seven levels, from 1 to 7. Each higher level is used for a higher level of subtotal. Each summary level can be identified by the deeper shade of blue in the background, as shown here. This database has been grouped into three summary levels.



Panorama allows you to treat summary records as a collapsible outline. You can use the **Outline Level** command to collapse the outline to show only high level summaries, then use the **Expand**, **Expand All**, and **Collapse** tools to expose the detail you need to see. For more about Panorama's outline capability, see "Summaries and Outlines" on page 393.

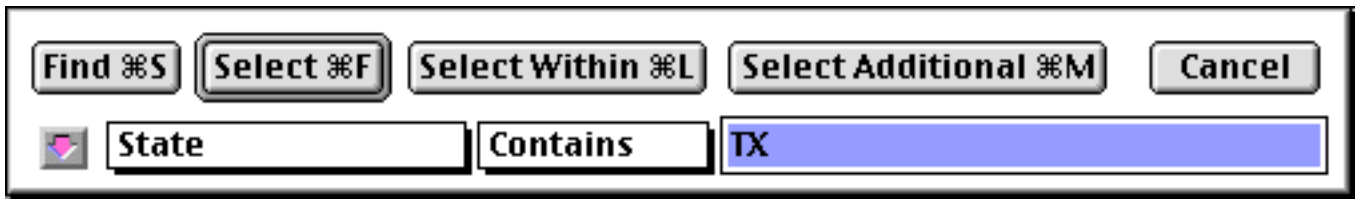


Summary records are designed to have a very short lifetime—usually only a few minutes. When you want to calculate subtotals or other summaries you'll create new summary records. After you've examined (and possibly printed) the summaries, you'll use the **Remove Summaries** command to remove them so you can get back to regular work with your database. For more information about summary records, see "3-Step Summarizing" on page 393.

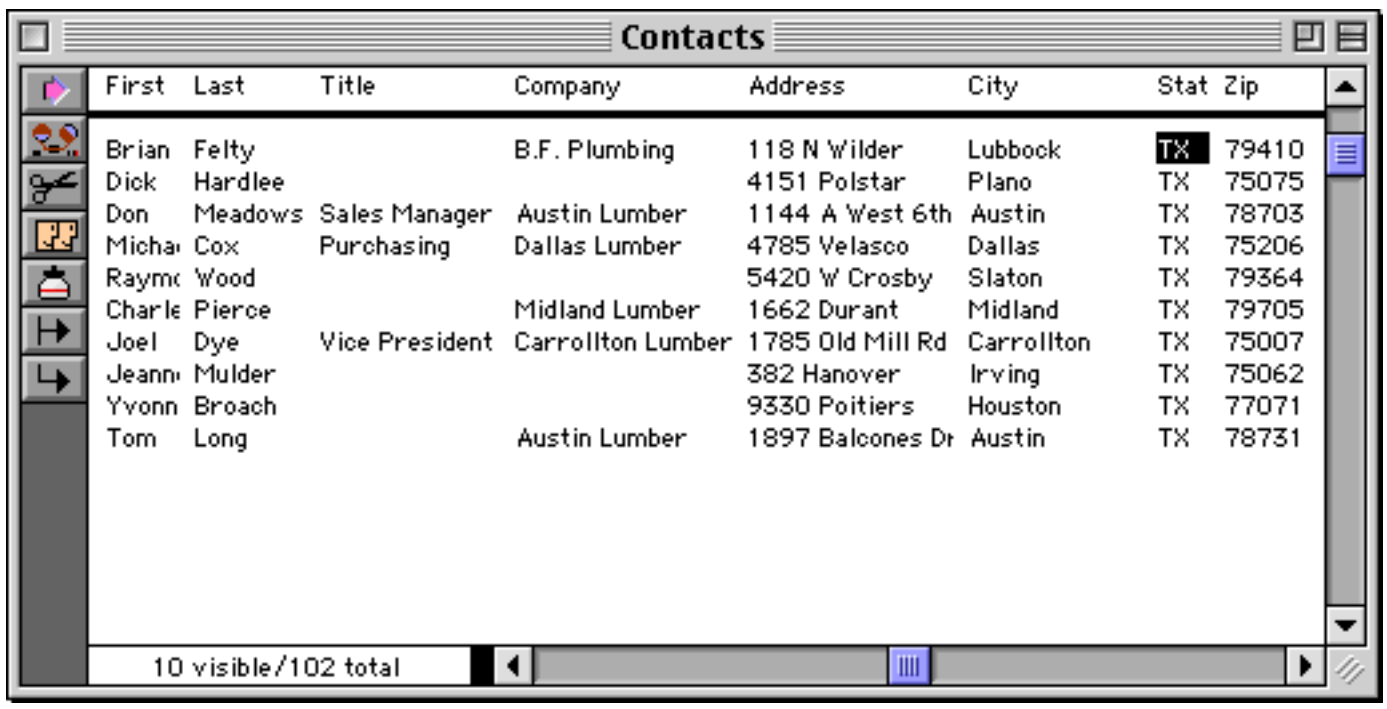
Invisible Records

Invisible records are ordinary data or summary records that have been made temporarily invisible. For instance right now you might be interested only in sales made in California, transactions over $250,000, or invoices over 45 days old. Panorama's **Find/Select** commands allow you to choose the data you want to see and make the rest temporarily invisible. This allows you to see just the information you are interested in without the other data getting in the way.

The following illustrations show a typical example. The original database started out with 102 records. We can use the **Find/Select** command to make 92 of these records temporarily invisible.



When the **Select** button is pressed, only records in Texas remain visible. All other records become temporarily invisible.



At any time you can make all records visible again with the **Select All** command. You can also make a new selection at any time. For more information about invisible records, see “Finding vs. Selecting” on page 367.

Chapter 5: Fields



The information stored in a database is organized into records and fields. Each field contains a specific category of information—names, phone numbers, birthdates, etc. Your first task after creating a new database is to decide how many and what fields are needed to get the job done. It's somewhat like designing a house—you have to decide what the best configuration is. How many bedrooms you need? How many baths? Will you need an office? And just like architecture, there are sometimes trade-offs that have to be made in designing a database.

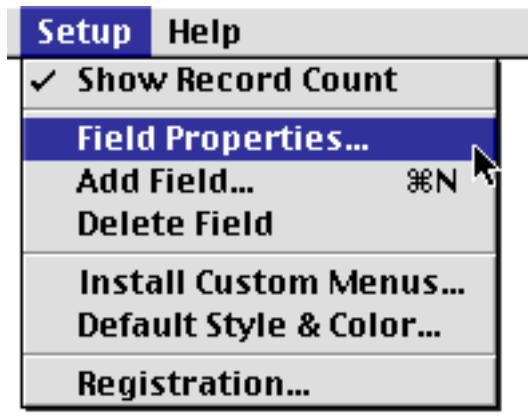
For some database jobs it is extremely obvious how the fields should be set up. But usually you have more flexibility than you might think. Take a simple name and address list, probably one of the most basic database applications. How should the name be stored? All in one field? Or should there be separate fields for first and last names? What about Mr./Ms./Mrs., should that be in a separate field? There are no hard and fast answers—it depends on how you want to use the database. For this example (names), data entry will probably be easier if you use a single field. On the other hand, you'll have more options for organizing and formatting if you split the name into several fields. The choice is up to you.

One thing to keep in mind is that if you make a mistake, it is much easier to combine two fields together than it is to split a single field into two. For example, it is quite easy to take separate first and last name fields and combine them, but if you type the names into a single field it could be quite difficult to later split them apart. If you have any doubt, it is better to err on the side of separating the data into more fields.

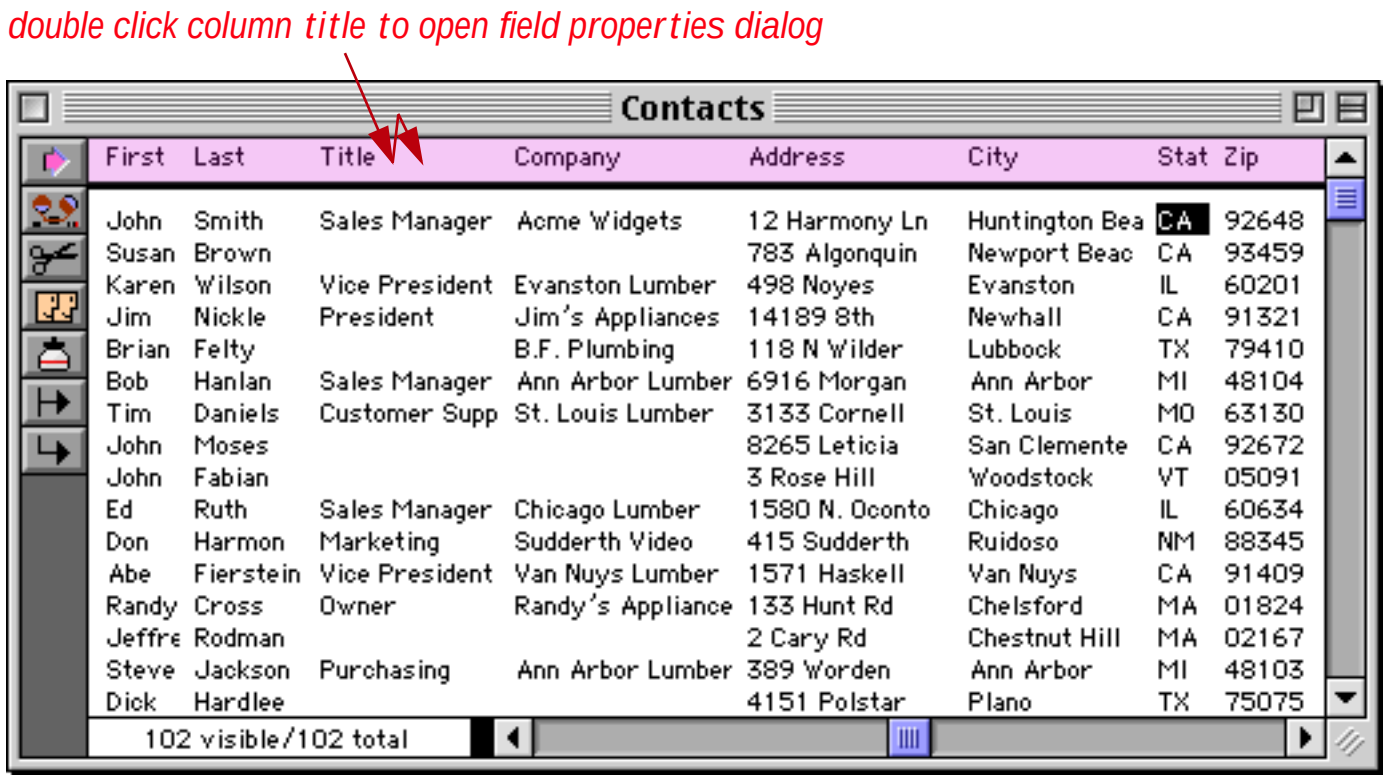
You can add new fields, remove fields, or change the properties of fields at any time, even after you've filled the database with data. Once you start entering data, however, it can be more work to re-arrange the fields. Some extra planning before you start entering data can pay big dividends in the long run.

The Setup Menu

The simple way to add or remove fields is to use the **Add Field** and **Delete Field** commands in the Setup Menu. You can also change most of the properties of a field (name, type of data, display format etc.) with the **Field Properties** command. These commands are available at any time when you are using the data sheet, and are also available when you are designing a form (graphic design mode).

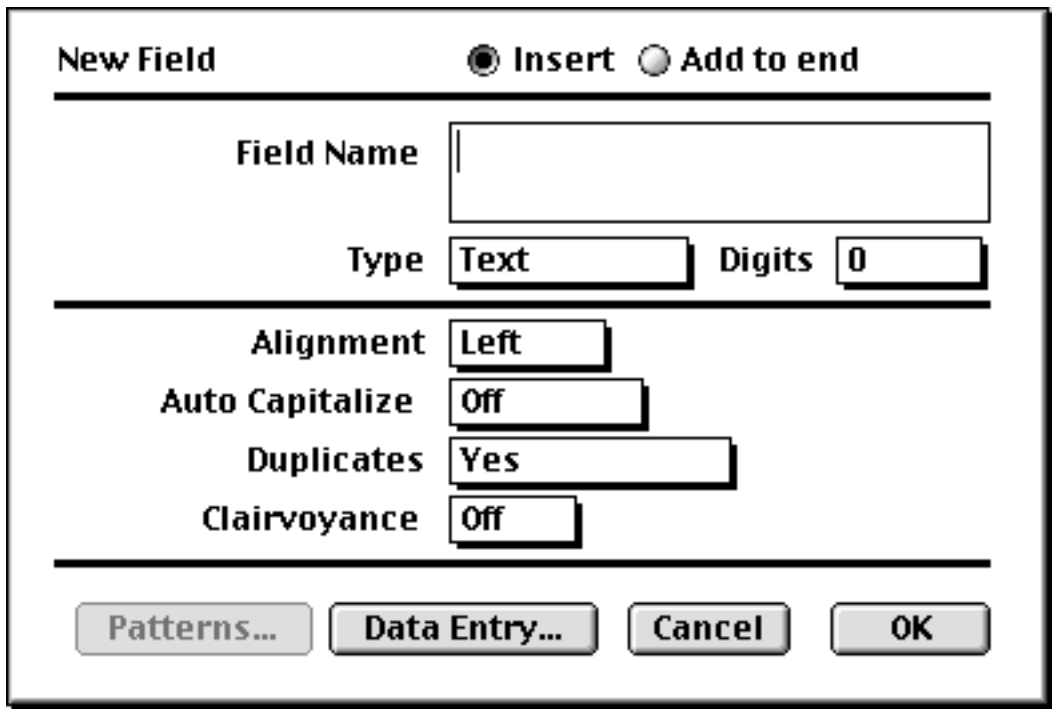


In the data sheet you can also open the **Field Properties** dialog by double clicking on a field name.



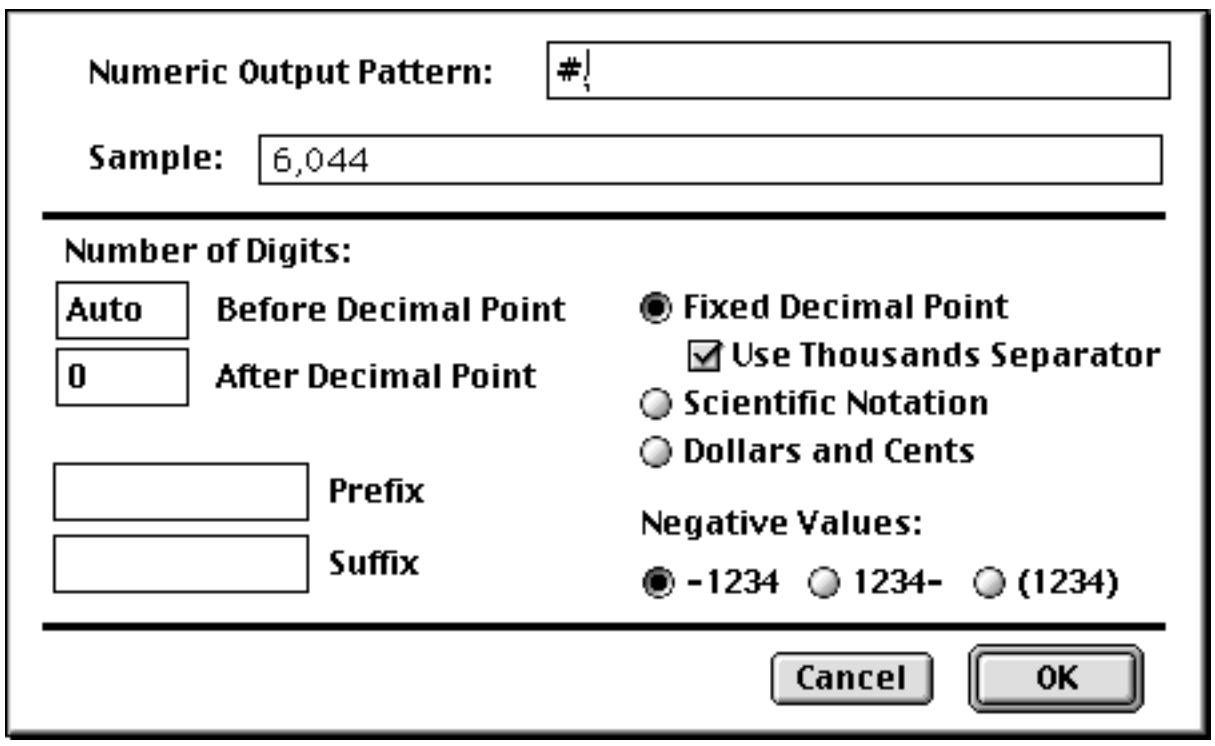
Add Field

To add a new field, use the **Add Field** command in the Setup Menu.

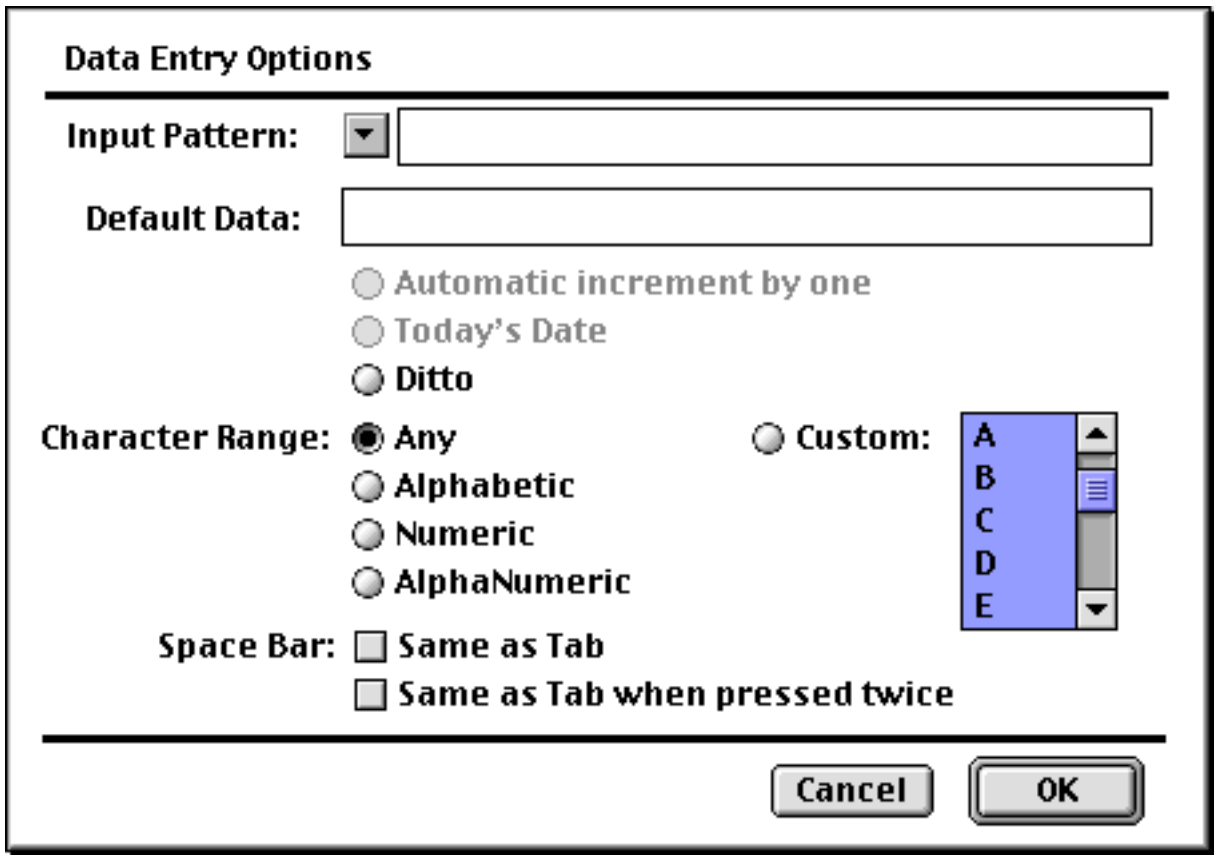


You can add the new field at the end (the right edge of the data sheet) or you can insert the new field in the middle of the database. In addition to the field name, you can use pop-up menus to set up the properties of the new field. The Type pop-up tells Panorama what kind of data you expect to store in the field—text, numeric, etc. See "Setting Up a Field's Data Type" on page 286 for a full explanation of data types and the number of digits. The Alignment pop-up controls how the field is aligned in the data sheet—left, center, or right flush. The Auto Capitalize, Duplicates, and Clairvoyance pop-up menus turn on various data entry options—see "Data Entry Accelerators" on page 318 for more information.

Use the **Patterns** dialog to set up the display format for numbers and dates. Output patterns are discussed in detail in "Numeric Output Patterns" on page 290 and "Date Output Patterns" on page 295.



Use the **Data Entry** dialog to set up the Input Pattern, Default Value, Range, and Space Bar Tab. These options are described in detail in Chapter 7, Data Entry & Editing.



Field Properties

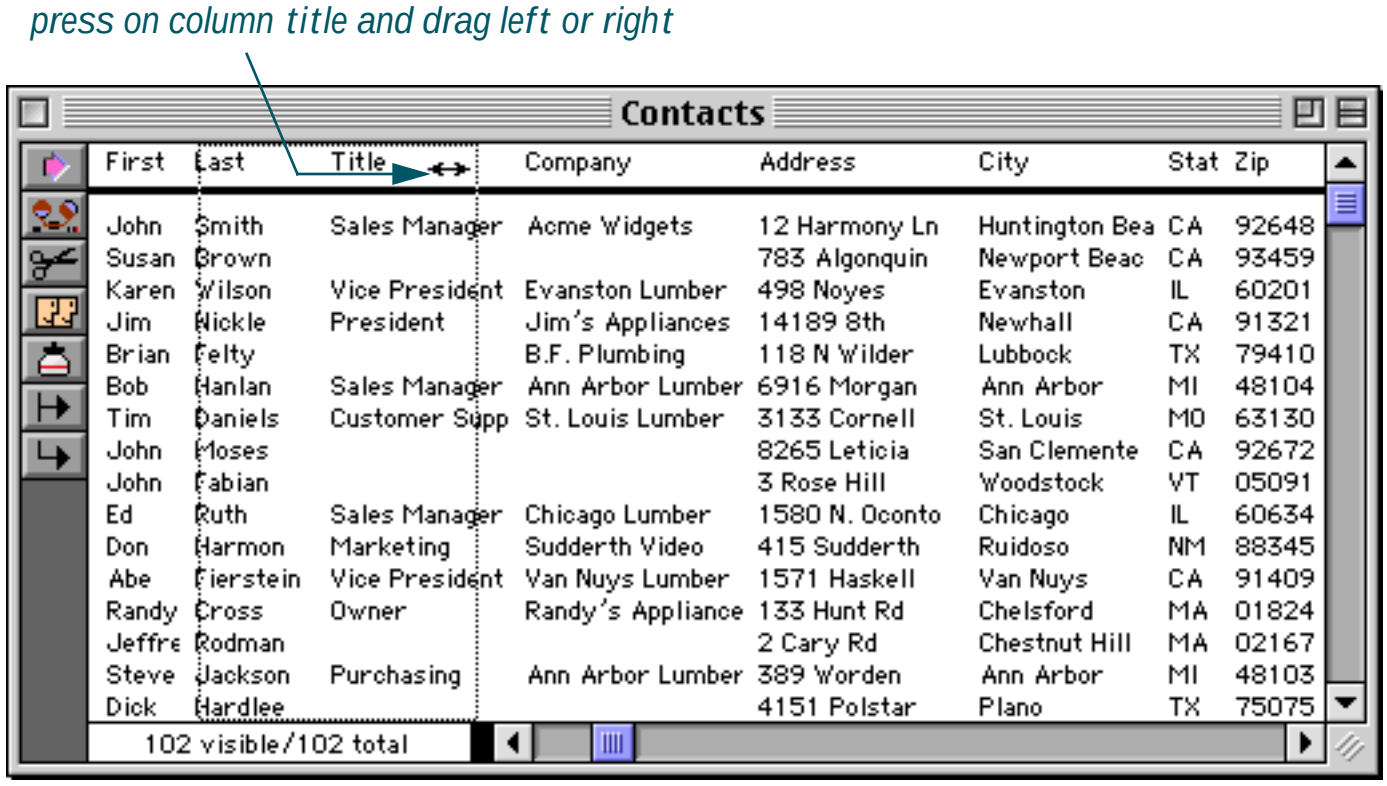
To change the properties of an existing field first select a cell in that field and then choose **Field Properties** from the Setup Menu. (If you are using the data sheet, you can also open the dialog by double clicking on a column name.) This dialog is almost identical to the **Add Field** dialog, and allows you to change the field name, data type, etc.

Delete Field

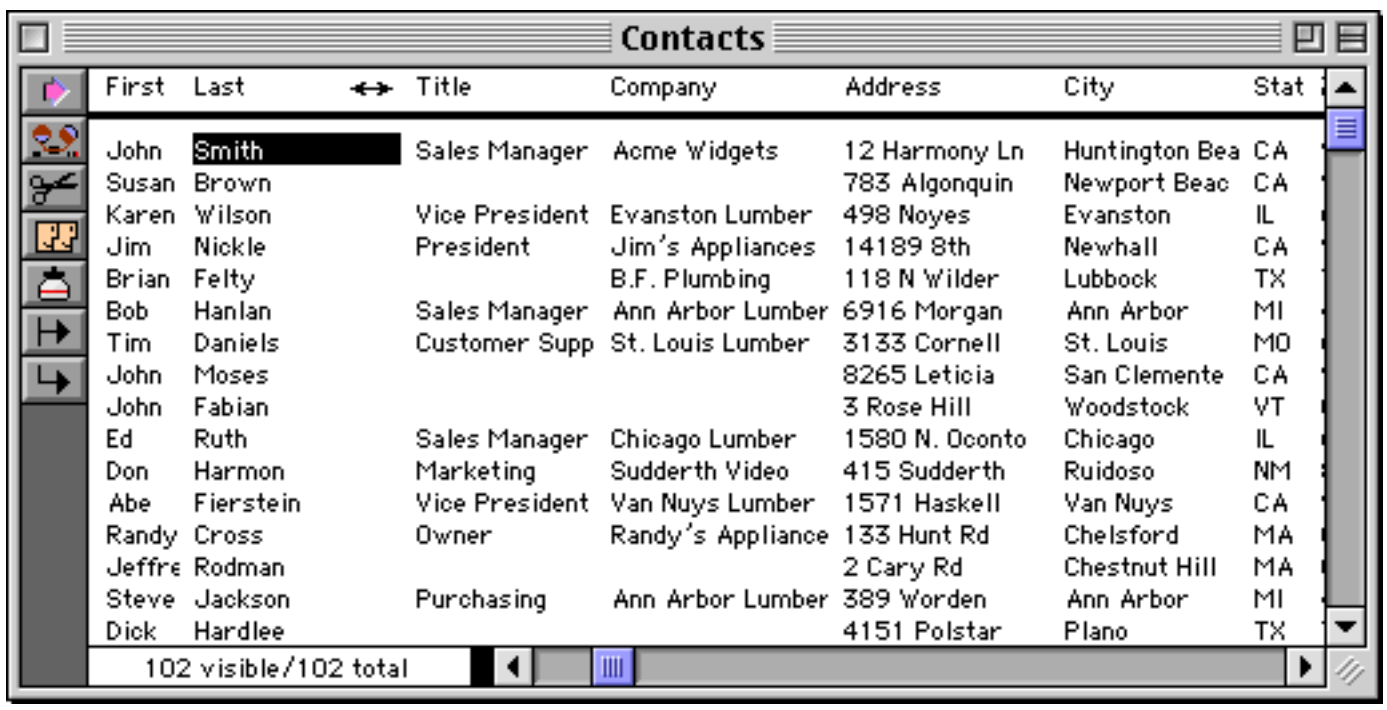
To delete a field from the database, select a cell in the field and choose **Delete Field** from the Setup Menu. (If you are using the data sheet, you can also press **Command-Delete** (Mac) or **Control-Delete** (Windows) to delete a field.) This not only deletes the field, it also deletes any data in the field. If the field contains data, Panorama will warn you that it is about to delete the field. You must confirm that you really want to delete the field before Panorama will proceed. You cannot **Undo** this operation, so be careful!

Changing the Width of a Field

To change the width of a field in the data sheet, move the mouse over the column name. When you move the mouse over the column name, the cursor will turn into a double headed arrow. When you press the mouse, a gray box appears around the column. Drag the mouse left or right to expand or shrink the column width, then release the mouse when the field is the correct width.

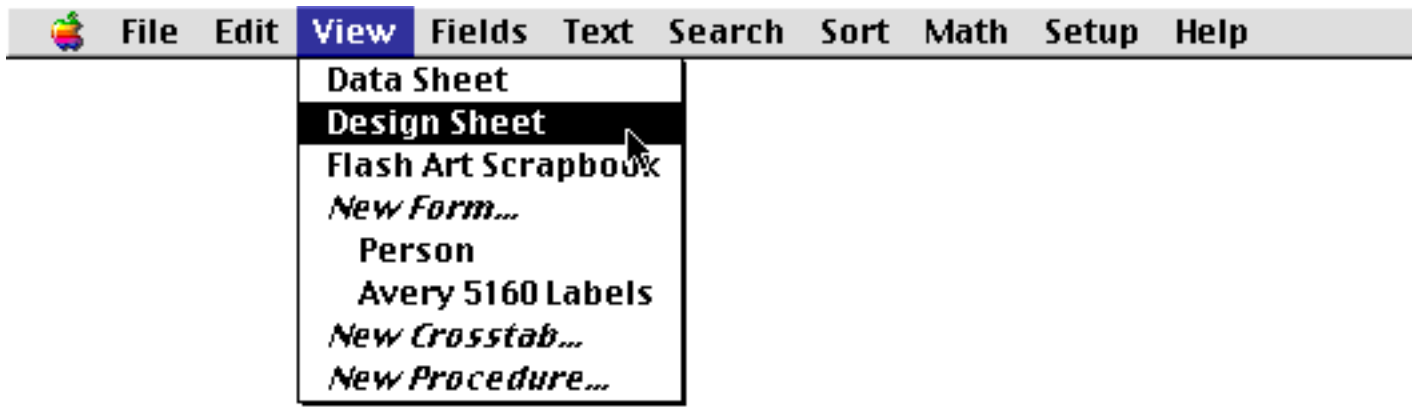


When the mouse is released the column width is adjusted.

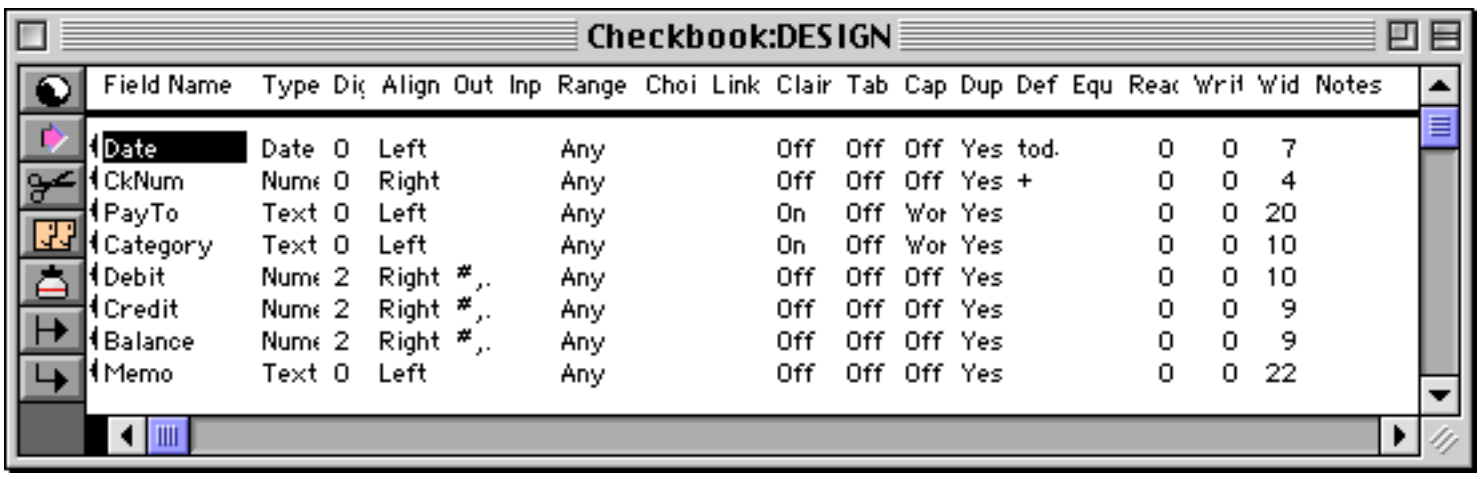


The Design Sheet

The Setup Menu provides an easy way to add or remove a few fields at a time. The disadvantage of the Setup Menu is that it only allows you to see a little piece of the database structure at a time. To get a more comprehensive view you'll need to open the actual blueprints of the database—the **design sheet**. Like other parts of the database, the design sheet is accessible from the View Menu.



The design sheet shows all the fields and their properties. Like DNA, the design sheet contains all the information about how the database is organized.



Database "Generations"

When DNA mutates, the change doesn't take effect until the next generation. Panorama's design sheet works the same way. The changes you make to the design sheet don't immediately change the database. Instead, Panorama waits for you to tell it to create a "new generation" of the database. This allows you to make multiple changes to the design sheet, check the changes for accuracy, and then apply all of the changes at once to the actual database structure.

There are three ways to tell Panorama to create a new generation.

1. Click the **New Generation** tool. 
2. Switch the window to a different view (using the View Menu).
3. Close the design sheet window.

Notice that you don't have to close the design sheet window to create a new generation. If you wish, you can even leave the design sheet open as you work with the database. This allows you to make a change to the design sheet, then quickly test the change and make further changes if necessary.

Once you have created a new generation, you cannot go back to the old generation with the **Undo** command. However, you can go back to the last generation saved on the disk with the **Revert to Saved** command (See "Revert to Saved" on page 133).

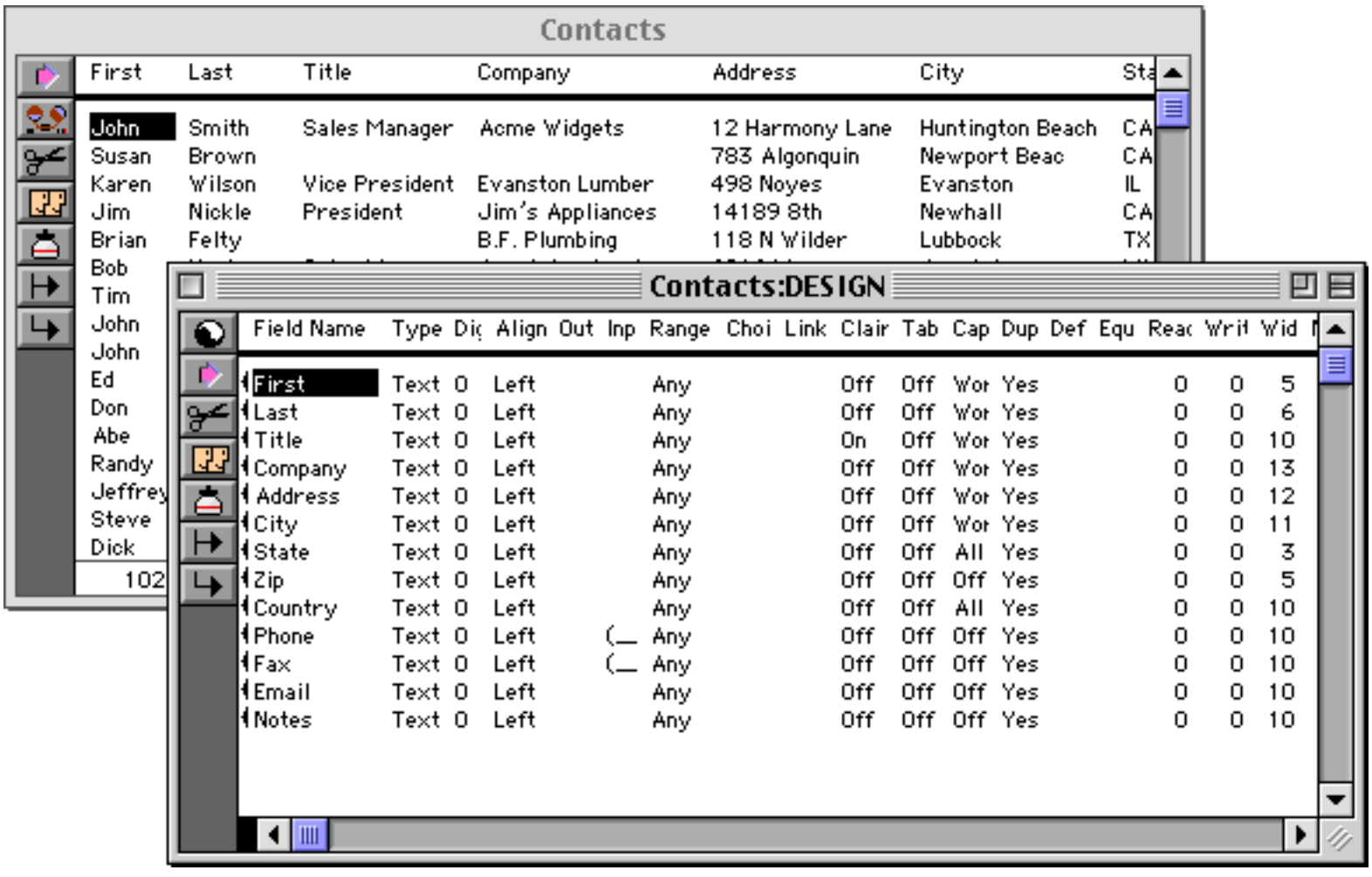
Typical Design Sheet Operation

Once you get the hang of it, the data sheet is very simple to use. As an example, we'll show how to add two new fields to a Contacts database using the design sheet. We'll add fields for a prefix (Mr./Ms./Mrs.) and for a middle name.

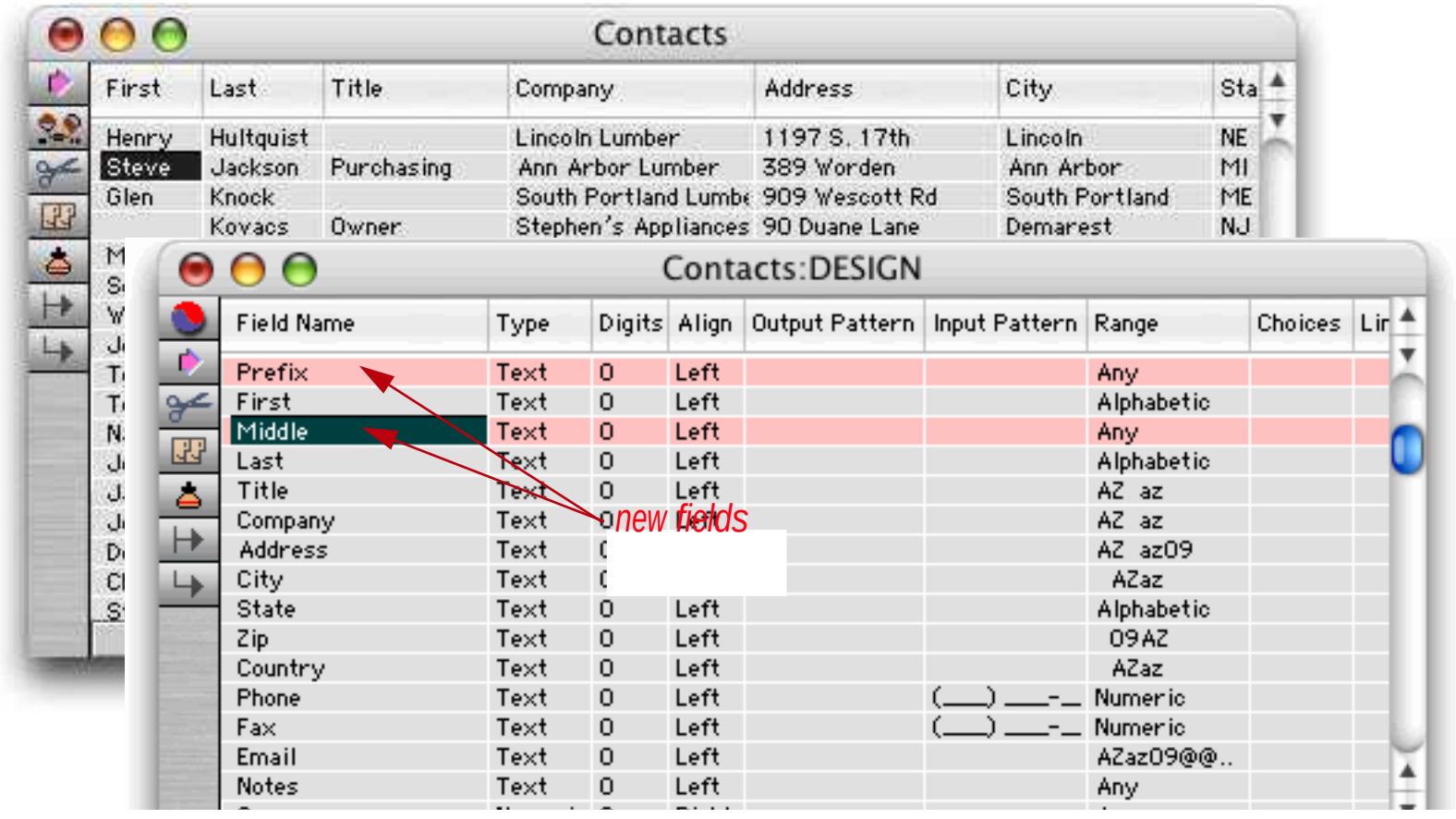
The first step is to open the design sheet using the View Menu.



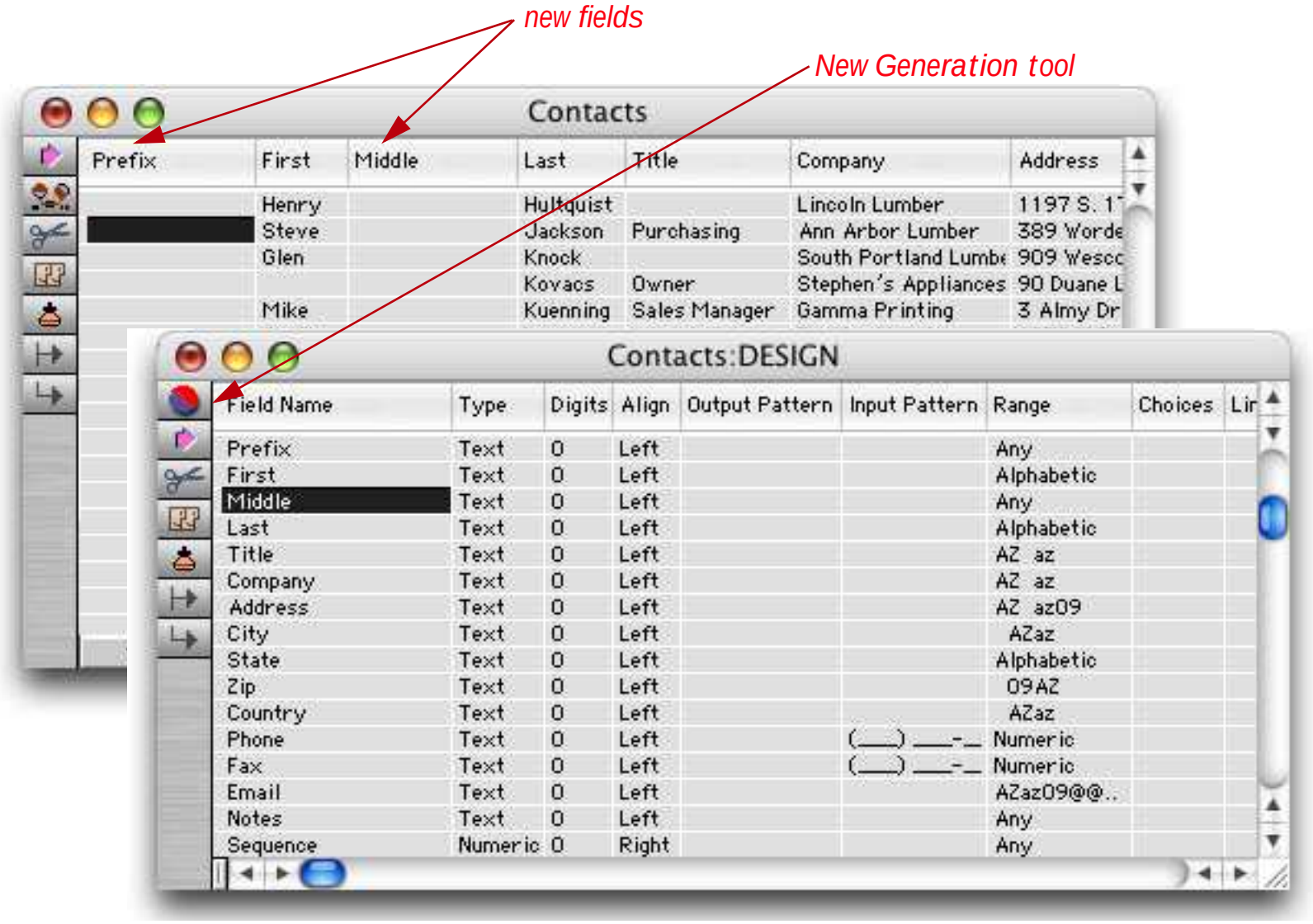
If you hold down the **Control** key (Mac) or **Alt** key (Windows) the design sheet will open in a separate window, allowing you to see both the data sheet and the design sheet at the same time (see "Opening More Than One Window Per Database" on page 223). In the illustration below we've dragged the design sheet window down and to the right so that we can se both windows at once.



Now we'll add the two new fields to the design sheet. We use the **Insert New Record** tool to insert the new lines, then type in the field names. The new fields are displayed with a pink background to show that they have not been added to the database yet.



Once the fields are entered, click on the **New Generation** tool to update the database itself.



The new fields are ready to use—you can bring the data sheet forward and start entering data in them right away. It's not necessary to close the design sheet first—you can leave it open in case you need to make further modifications to the fields. It's usually a good idea, however, to close the design sheet when you are not going to be using it again for a while.

Wait just one minute! There's one final step you don't want to forget. When you're sure you've got the structure you want, make sure you **Save** the database to make your changes permanent (See "Saving a Database" on page 131)!

Field Properties

Each row in the design sheet contains all the properties for a single field in the database. The design sheet contains 19 columns—one column for each field property.

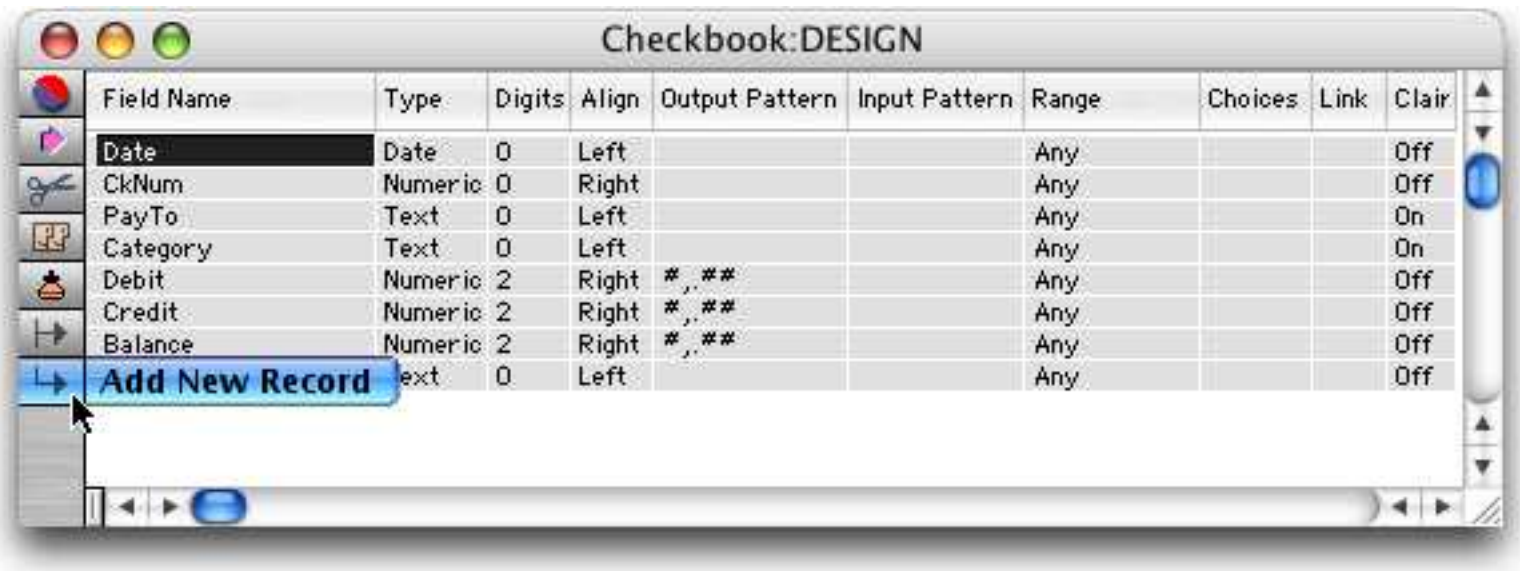
Field Name	The field name identifies the field. It helps you remember what is in the field, and is also used to identify the field in formulas. Panorama does not place any restrictions on your choice of field names, but there are some ramifications to using an unusual name (see "Rules for Field Names" on page 260).
Type	This column specifies the type of data stored in each field: text, numeric, date, choices, or picture. See "Data Types" on page 285 for more information.
Digits	This column specifies the number of digits to be used after the decimal point with numeric data: 0, 1, 2, 3, 4, Float or Money. See "Numeric Data" on page 289 for more information.
Align	This column specifies how the field should be aligned in the data sheet: left, center, or right flush. Usually left flush is used for everything except numbers, which are displayed right flush.
Output Pattern	The output pattern allows you to specify the display format for numbers or dates. See "Numeric Output Patterns" on page 290 and see "Date Output Patterns" on page 295 for more information.
Input Pattern	The input pattern forces data into a specific pattern as it is entered, for example phone numbers or social security numbers. See "Input Patterns" on page 327 for more information.
Range	This column allows you to restrict the characters that can be entered in a field. For example, you can restrict a field to only allow alphabetic or numeric entry. See "Restricting Character Types" on page 330 for more information.
Choices	This column allows you to specify a list of choices that are valid for this field; for instance Yes/No, Gold/Silver/Bronze or Regular/Unleaded. The list of choices is used by the Choices data type (see "Choices" on page 298) and is also used by the Choice Palette (see "The Choice Palette" on page 353).
Link	This column allows you to link this field with a field in another database. Only Clairvoyance® is affected by this link. You can set up this field with the Clairvoyance Link command. For more information see "Clairvoyance® Across Multiple Files" on page 323.
Clairvoyance®	This column controls Panorama's Clairvoyance feature. This feature tries to anticipate what you are about to type, then types it for you. See "Clairvoyance®" on page 321 for the straight scoop.
Tabs	This column controls the Space Bar Tab feature. This feature makes the **Space Bar** work just like the **Tab** key, saving wear and tear on your left pinky. See "Tabbing with the Space Bar" on page 317 for details.
Caps	Use this column to tell Panorama to automatically capitalize data entry in a field. Panorama can automatically capitalize everything, or just the first letter of each word or sentence. See "Automatic Capitalization" on page 319.
Dups	This column specifies whether or not you want to allow duplicate entries in this field. You can also specify that you want to require duplicate entries (no unique values). See "Checking for Duplicate Data" on page 320.
Default Value	This column allows you to specify a default value for the field when a new record is created. See "Default Values" on page 333.

Equation	This column allows you to specify one or more calculations to be performed whenever the information in this field changes or is confirmed. For example, an invoice can be set up so that all totals are calculated whenever a quantity or price is entered or changed. See “Automatic Calculations” on page 340.
Read	This attribute is used to control the security level for displaying (reading) the data in this field. The value in this field may be from 0 (anyone can see this data) to 255 (only users with the highest possible security level can see this data). For more information on security levels see the Panorama Security Handbook, available separately.
Write	This attribute is used to control the security level for modifying (writing) the data in this field. The value in this field may be from 0 (anyone can modify this data) to 255 (only users with the highest possible security level can modify this data). For more information on security levels see the Panorama Security Handbook, available separately.
Width	This contains the approximate width (in characters) of the field in the data sheet. For example, a width of 20 means that the column is about 20 characters wide (the actual width depends on the font and size). Although you can use the design sheet to set the column width, it is easier and more exact to set the width by dragging on the column name (See “Changing the Width of a Field” on page 251).
Notes	You can use this field to keep notes about the field. If a database contains dozens or hundreds of fields, it may be difficult to remember what each field is for. You can use this field to store reminders to yourself about the purpose and use of each field. Panorama ignores the contents of this column.

Adding New Fields Using the Design Sheet

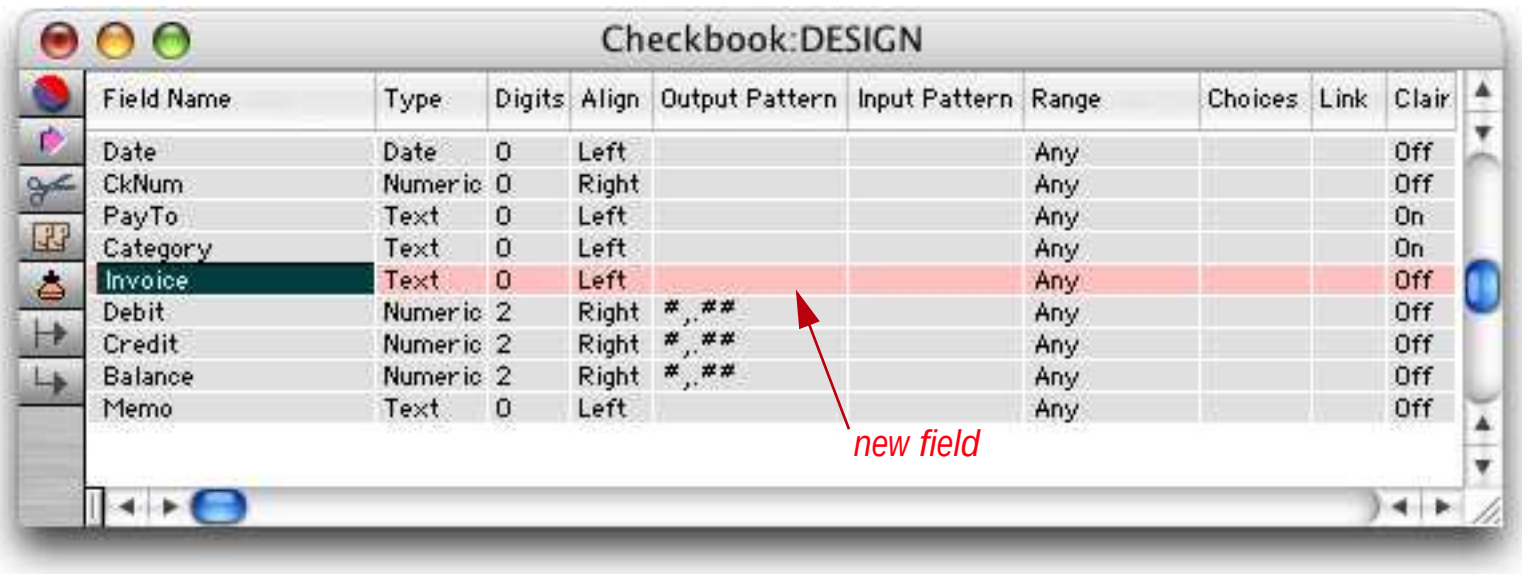
In addition to changing the properties of existing fields, the design sheet can be used to add new fields or to delete existing fields. The design sheet is especially handy when you need to add a lot of new fields at once.

Since each line in the design sheet corresponds to a field, you can add new fields simply by adding new lines to the design sheet. The design sheet works the same way as the data sheet—you can add new lines using the **Add New Record** tool, the **Insert New Record** tool, or by pressing the **Return** key. (Keep in mind that each **record** in the design sheet corresponds to a **field** in the database, so adding or inserting a record here is equivalent to adding or inserting a field in the database itself.)



Remember that adding a line to the design sheet does not immediately create the new field. When you are ready to actually add the new field(s) to the database, tell Panorama to create a new generation (See “Database “Generations”” on page 252).

When you add a new line to the design sheet, you may notice that your new line has a pink background. This pink background serves as a reminder of the new fields you have created.



As soon as you create a new generation, the background of any new fields will change to gray. On the other hand, if you close the design sheet without creating a new generation, the new fields will not be created.

Removing Fields Using the Design Sheet

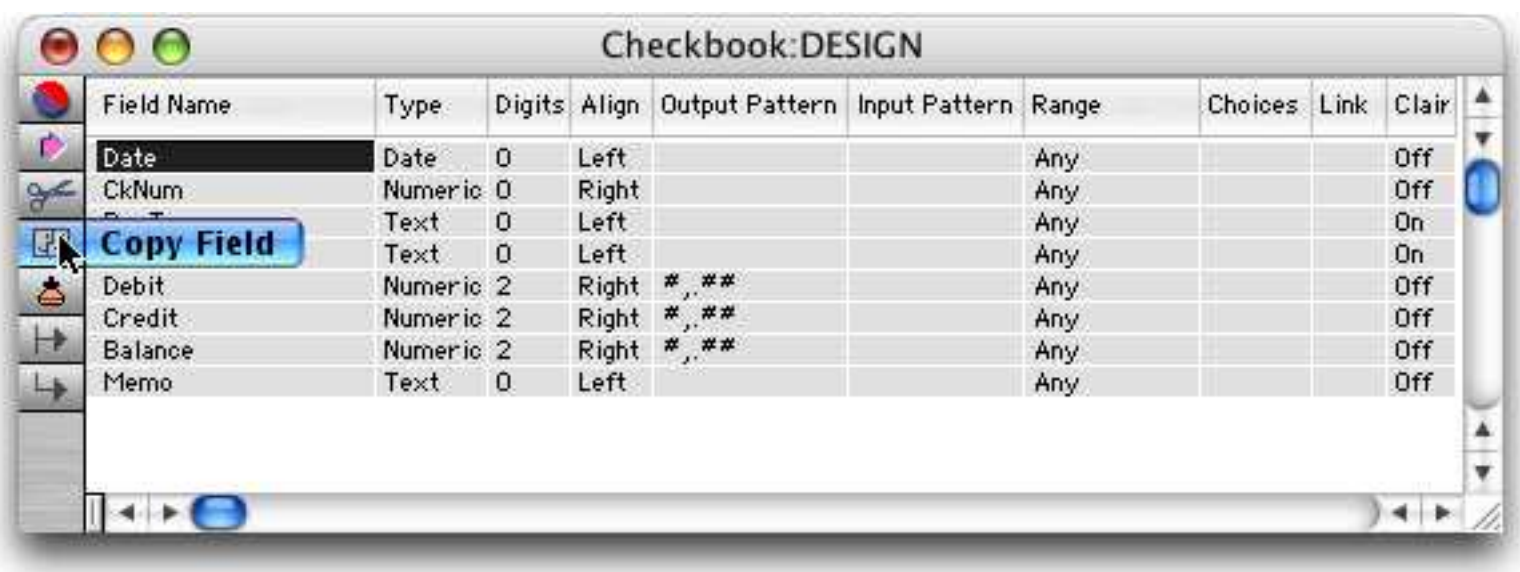
To remove a field from the database simply remove the corresponding line from the design sheet and then tell Panorama to create a new generation. You can delete a field using the **Cut Field** tool or by pressing the **Delete** key (Mac) or the **Backspace** key (PC).

When you remove a field, all the data in that field is lost forever (unless of course, you have another copy on the disk). Remember, you cannot go back to an old generation with the **Undo** command. Therefore, you'll want to be very sure that you really don't need a field anymore before you remove it.

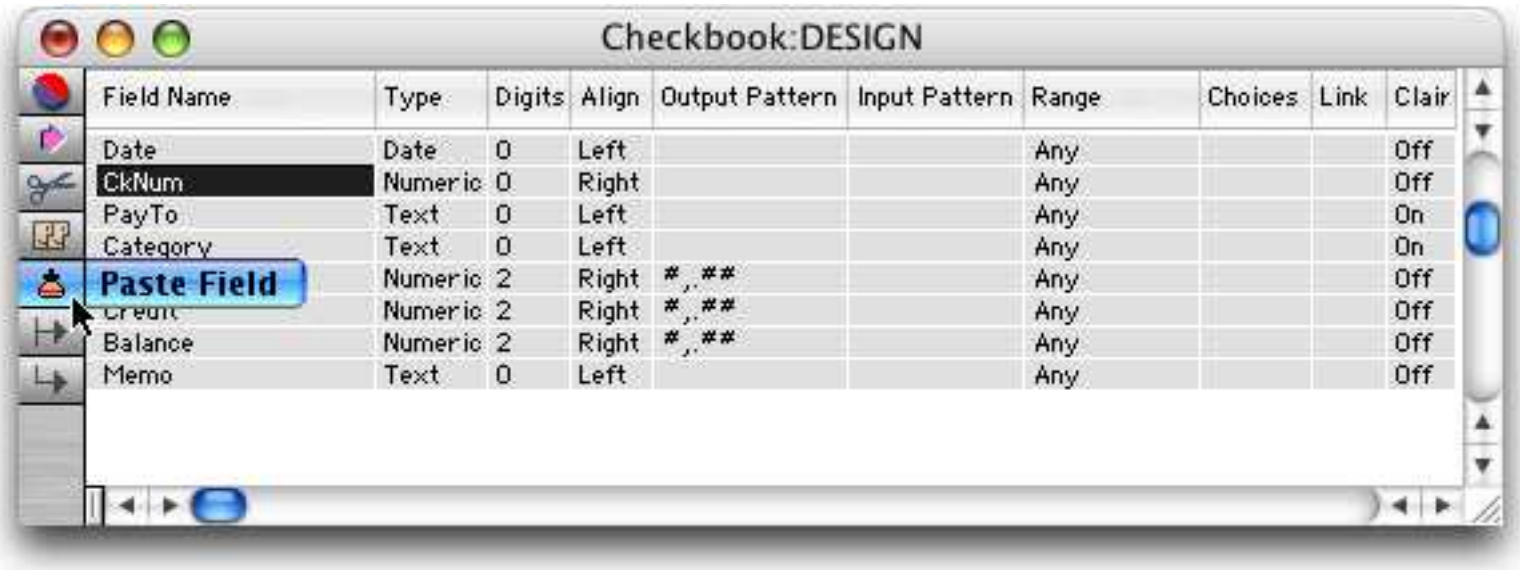
Making a Copy of a Field

You can create a copy of a field, including all the data in the field, by copying the corresponding line in the design sheet. Use the **Copy Field** tool to copy the line into the clipboard, then use the **Paste Field** tool to make a copy of the line in the design sheet. You'll probably want to rename the new field. (You must use the clipboard to create the copy—you cannot simply type in a copy of the line.) When you create a new generation the database will contain a copy of the field, including the data in the field.

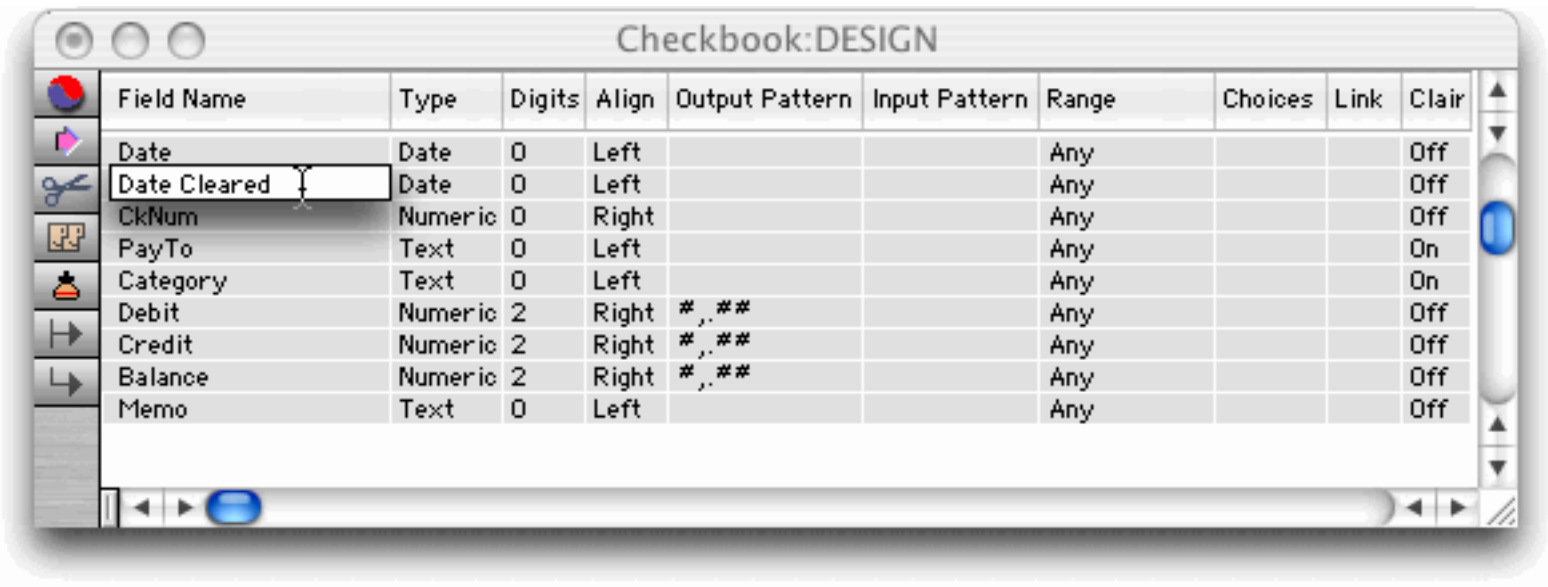
For example, let's suppose we want to create a DateCleared field for a Checkbook database, starting with the data already in the Date field. Start by selecting the Date field and copying it into the clipboard.



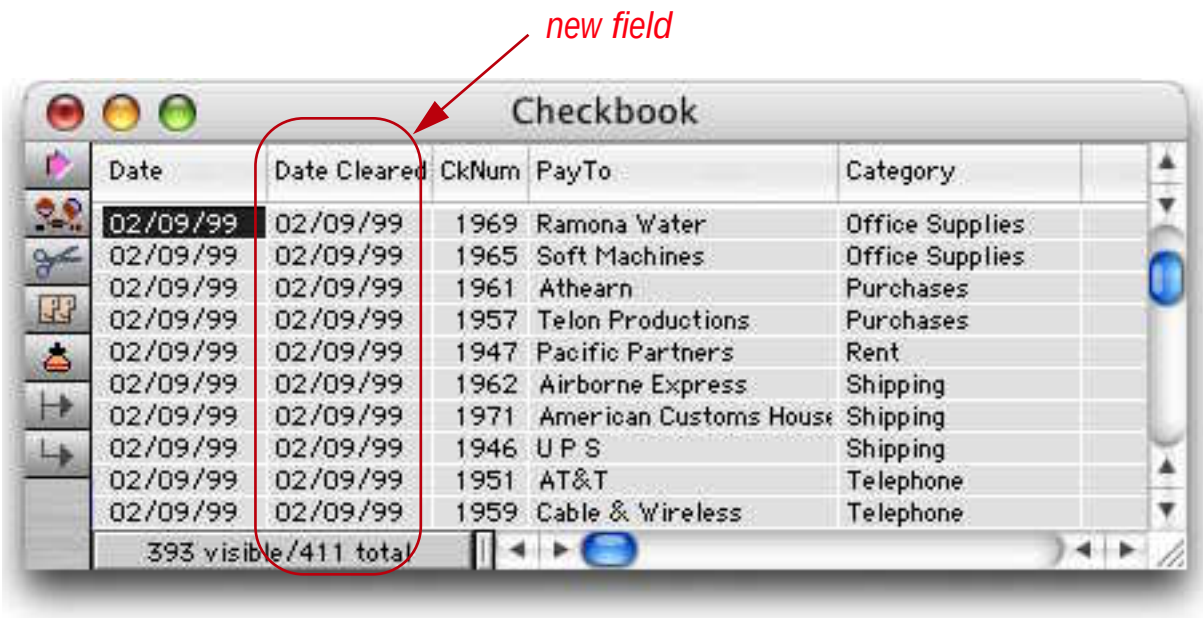
Now select the position for the new field and paste it in.



Rename the new field to DateCleared. Notice that the “new” field does not have a pink back ground, because it is not really a new field, but a copy of an existing field.



Use the View Menu to switch to the data sheet. When Panorama asks you if you want a New Generation, press the **Yes** button. As you can see, Panorama has created the new field with a copy of the data in the old field.



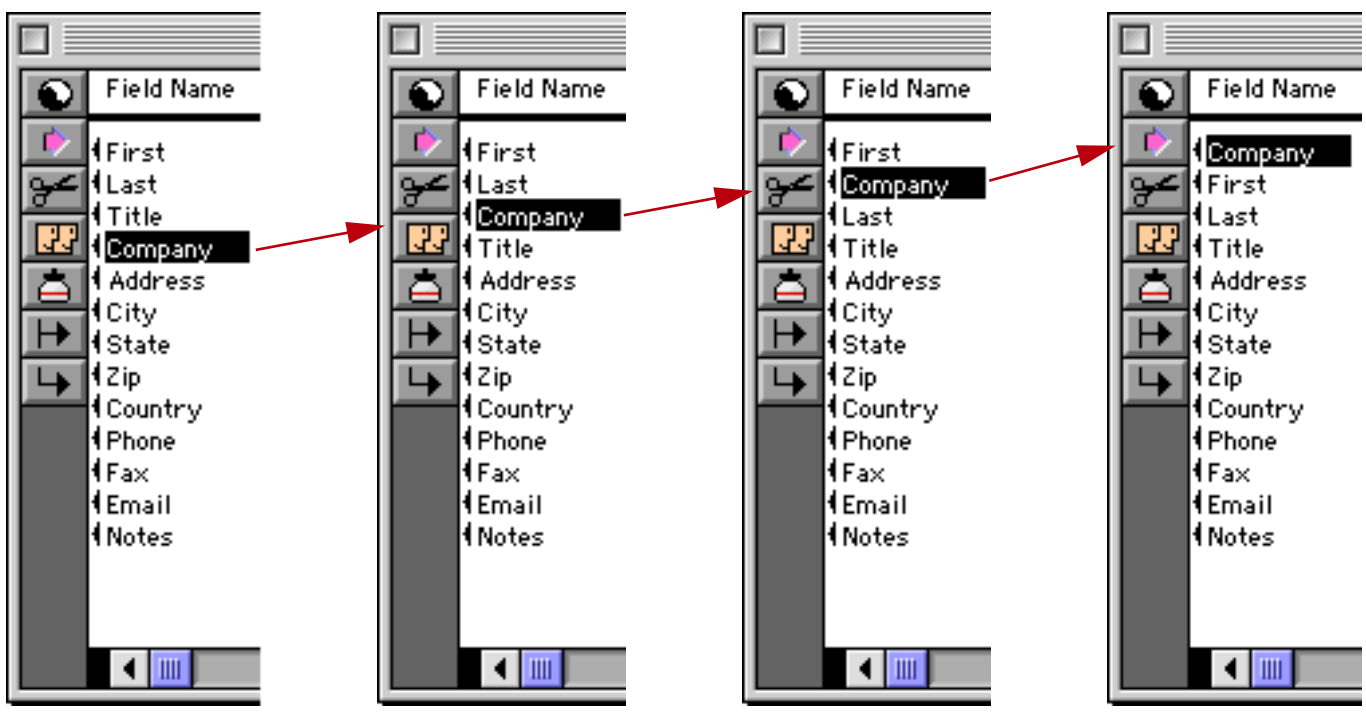
Note: You can only use this technique to copy fields within a single database. You cannot copy a field from one database and paste it into another database.

By the way, this technique is not the only way to copy a field. You can also use the **Formula Fill** command to copy a field at any time, not just when it is created (See "Filling a Field with a Formula" on page 453.)

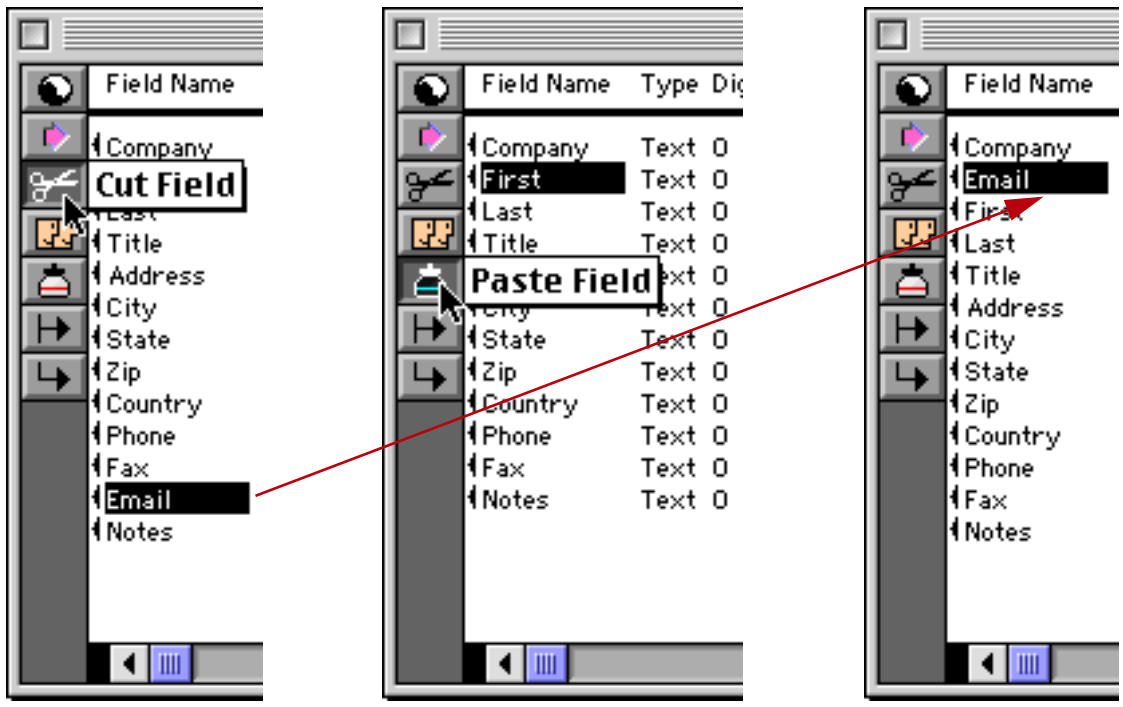
Re-Arranging Fields

The design sheet can also be used to re-arrange the order of the fields. It's a two step process—first re-arrange the lines in the design sheet, then create a new generation.

To move a field up one line in the design sheet, press **Command-Up Arrow** (Mac) or **Control-Up Arrow** (PC). To move a field down one line in the design sheet, press **Command-Down Arrow** (Mac) or **Control-Down Arrow** (PC). Keep pressing until the field reaches the desired position.



Another way to move a field is with the **Cut Field** and **Paste Field** tools. This method may be faster if you need to move a field a long distance.



When the fields are in the correct order, tell Panorama to create a new generation.

Rules for Field Names

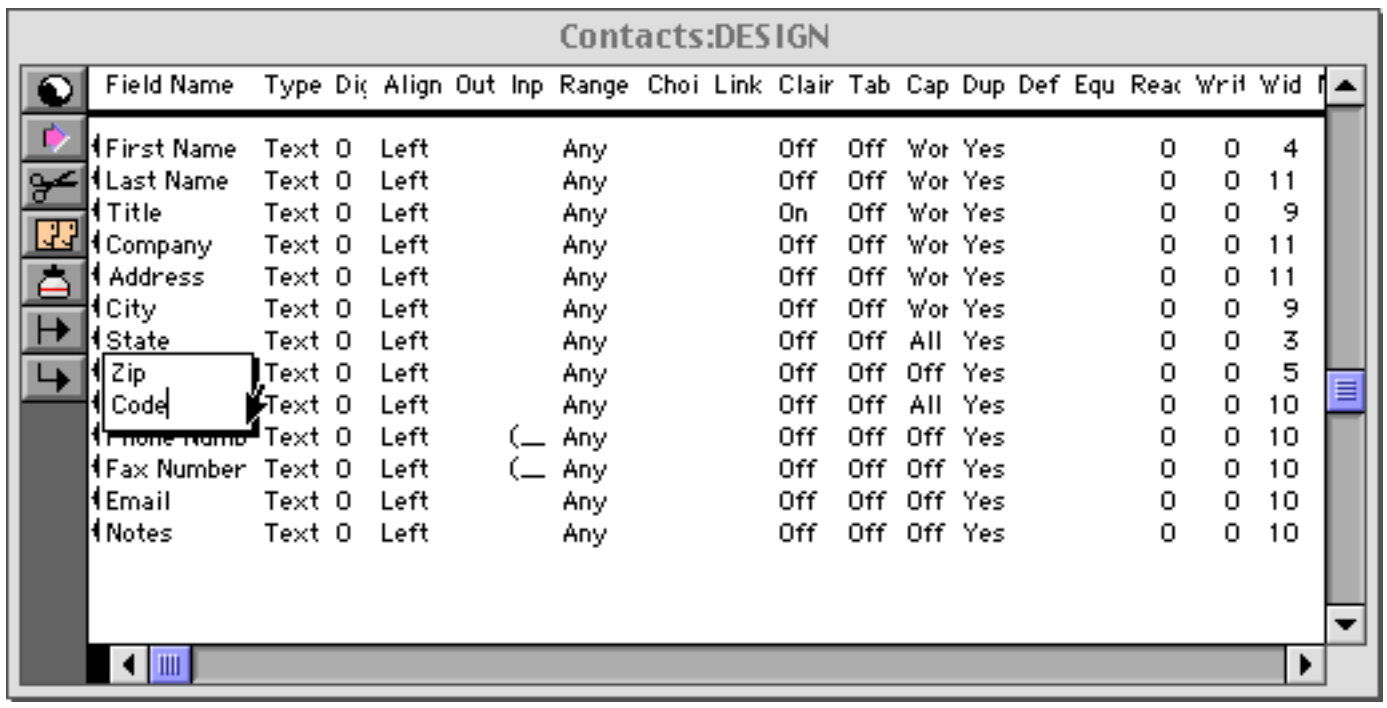
Each field in a Panorama database is identified by a field name. Field names serve several purposes: they remind you what the field is for (i.e. the Dates field probably contains dates, the Name field probably contains names, etc.), they appear at the top of each column in the data sheet, and they are used to identify fields in formulas and procedures (for example `Amount=Qty*Price`).

Panorama doesn't place any restrictions on the field names you choose. Field names may be as long as you want, and they may contain any character that can be typed from the keyboard. Field names may be split over two or more lines (see below). You can even have two or more fields with the same name (but we recommend that you avoid this, see the next paragraph).

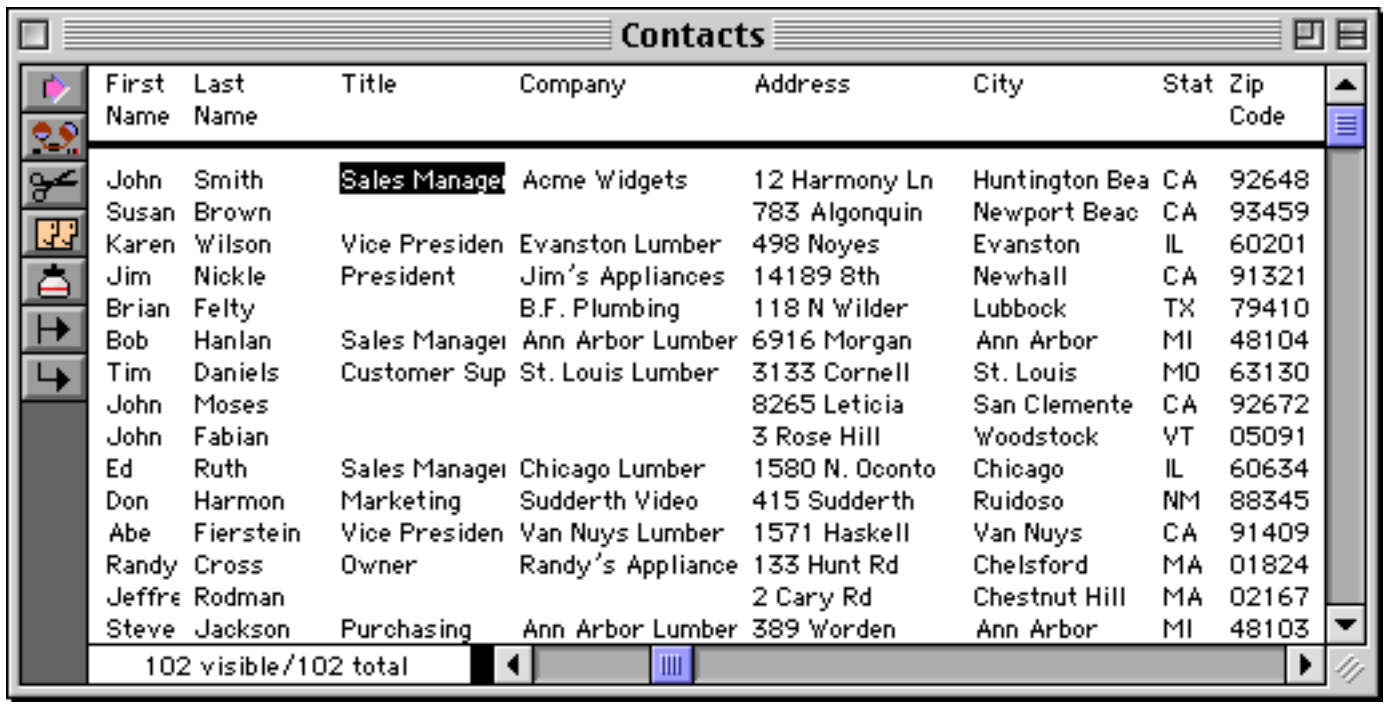
However, if you are planning to use a field in a formula or procedure, you may want to avoid some of these unusual possibilities. If you have two or more fields with the same name, only the first field will be accessible to a formula. Field names containing blanks or punctuation (for instance P/E Ratio) are more difficult to use in a formula. To use such a field in a formula, you must surround the field name with « and » (for example **`«P/E Ratio»`**. See "Fields" on page 1196). (If you left out the «», Panorama would think you were trying to divide `P` by `E`, with `Ratio` left over.) You may want to avoid field names like Date, Seconds, And, Or, and Sum. These names can be confusing when used in a formula because Panorama has functions with the same names.

Multiple Line Field Names

It is possible split a field name over two or more lines. The main reason for doing this is to create a multiple line title for the data sheet. Simply type in a **Return** between each line.



In the data sheet this field name will appear on two lines, like this.

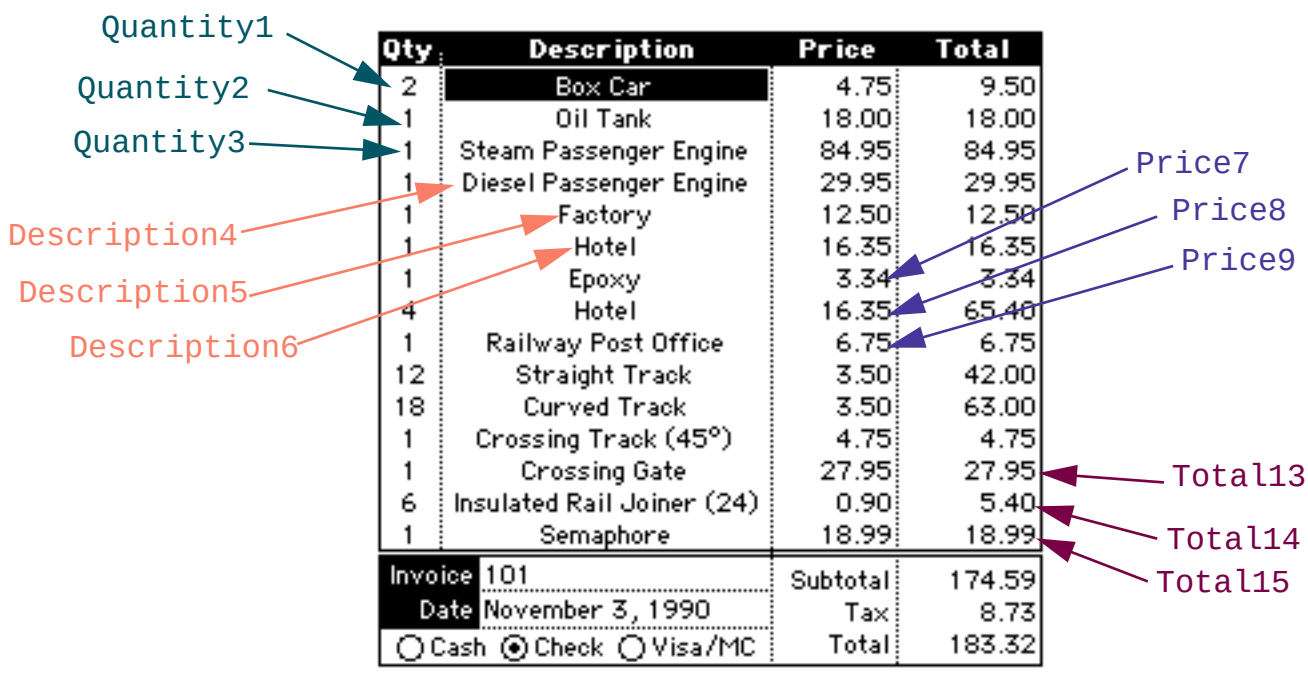


If you use one of these fields in a formula, the **Return**'s should be represented as spaces, as shown here. The formula on the left is correct, the formula on the right is wrong.

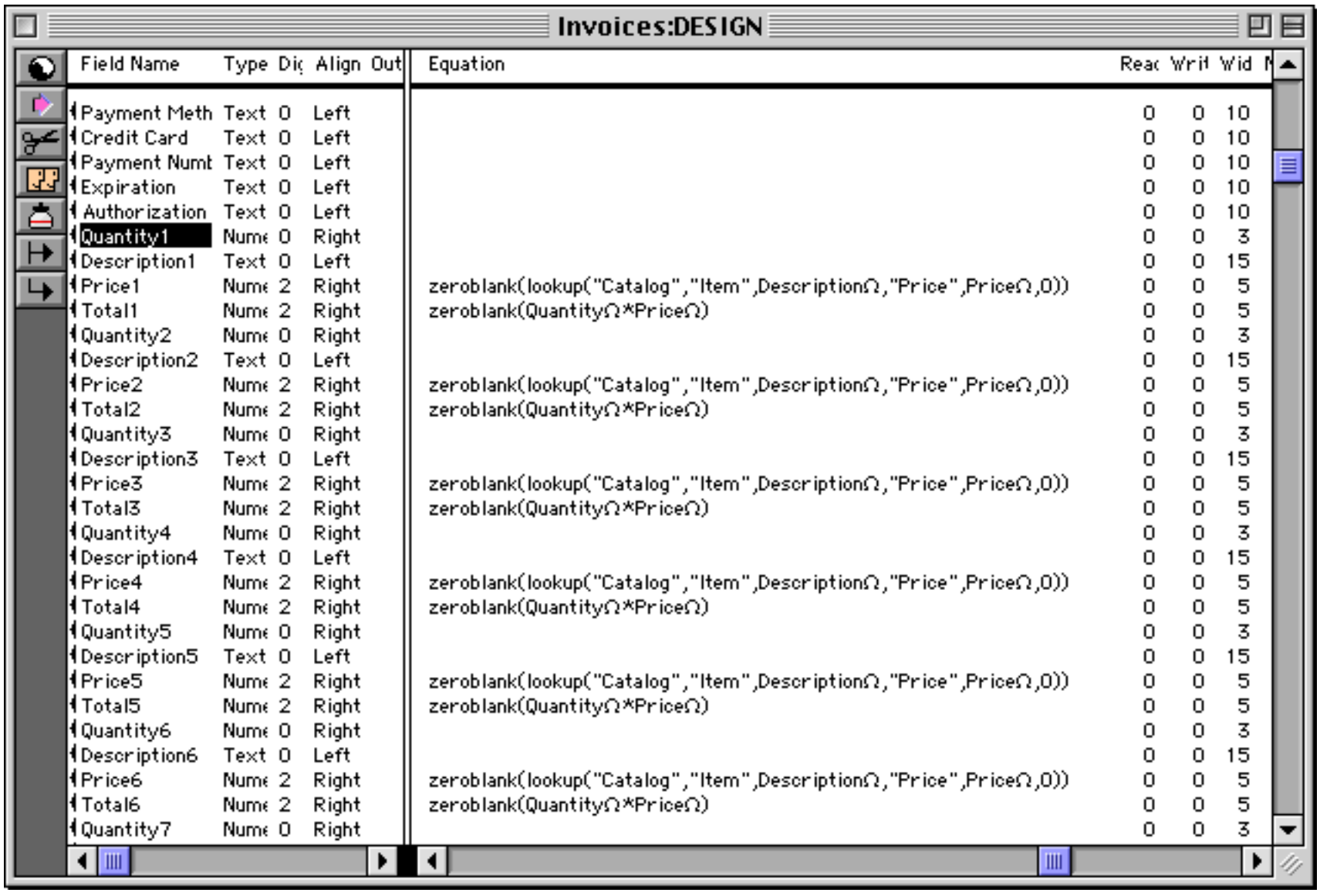
Correct	Wrong
`«Zip Code»=92365`	`«Zip` `Code»=92365`

Repeating Fields (Line Items)

Some databases contain several similar fields repeating within each record. For example, an invoice usually contains several quantities, product descriptions, product prices, etc. These fields are often called **Line Items** because they repeat for each line on the invoice. In Panorama these line item fields are created by adding a numeric suffix to the root field name, for example `Qty1`, `Qty2`, … `Qty15`. This illustration shows part of a form that contains line items arranged into 15 rows by 4 columns.



In the design sheet these line item fields are simply 60 individual fields.



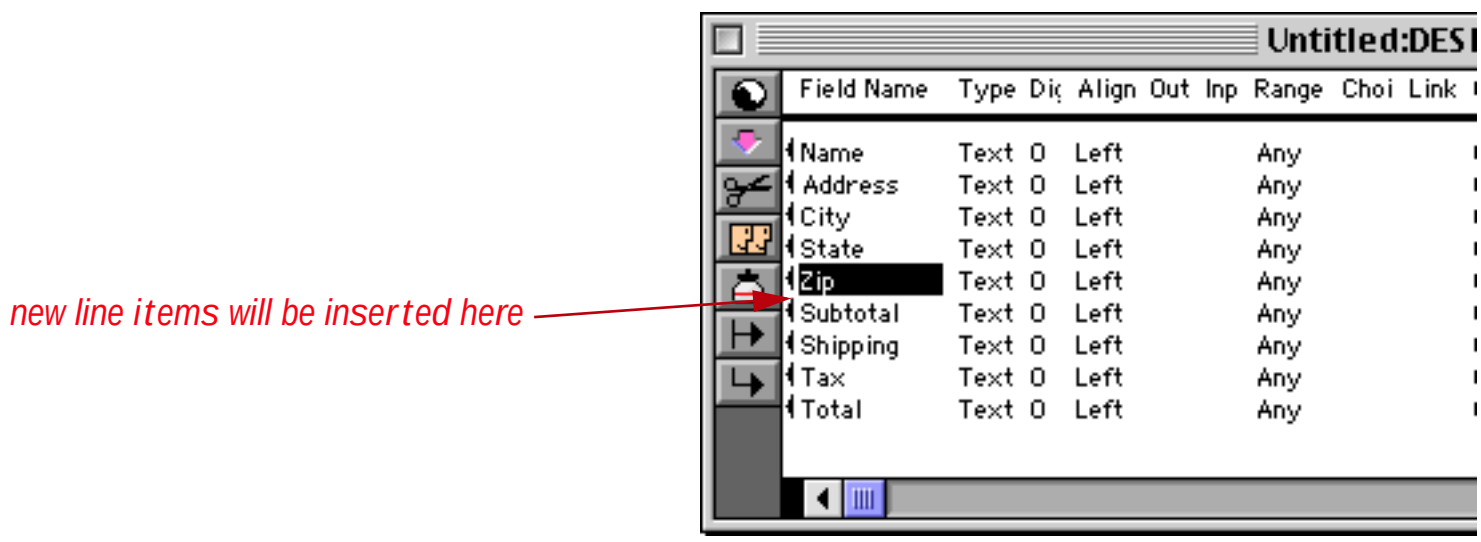
Right about now you are probably groaning at the thought of typing in all those fields. Don’t panic yet — we’ve got you covered (keep reading)!

Creating Line Item Fields

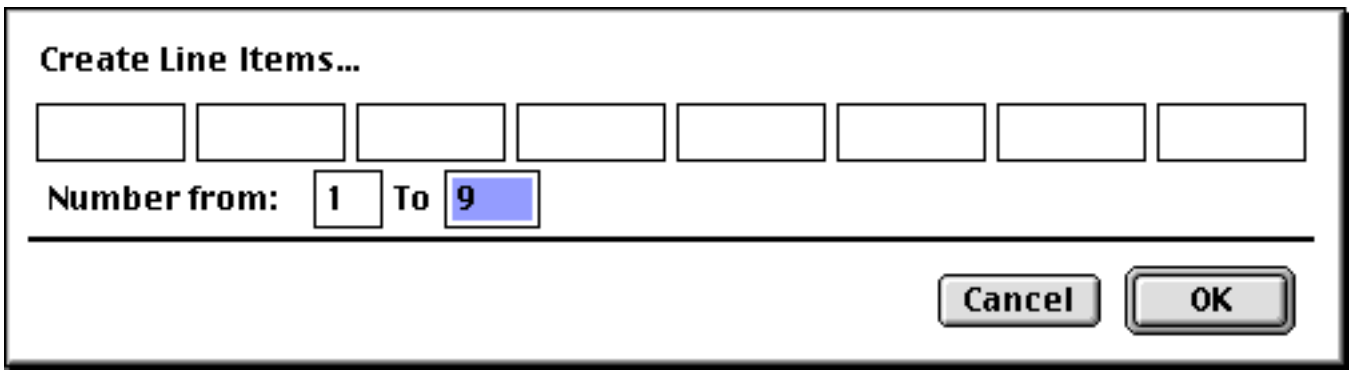
You can create line item fields the same way you create ordinary fields, using the **Add Field** command (see "Add Field" on page 249) or the design sheet (see "Adding New Fields Using the Design Sheet" on page 256). Just make sure to add the numeric suffix to the end of each name (with no space), and remember to spell the base name the same way each time, including capitalization.

OK	Not OK
Qty1	Q1
Qty2	Qty2
Qty3	QTY3
Qty4	Qty4

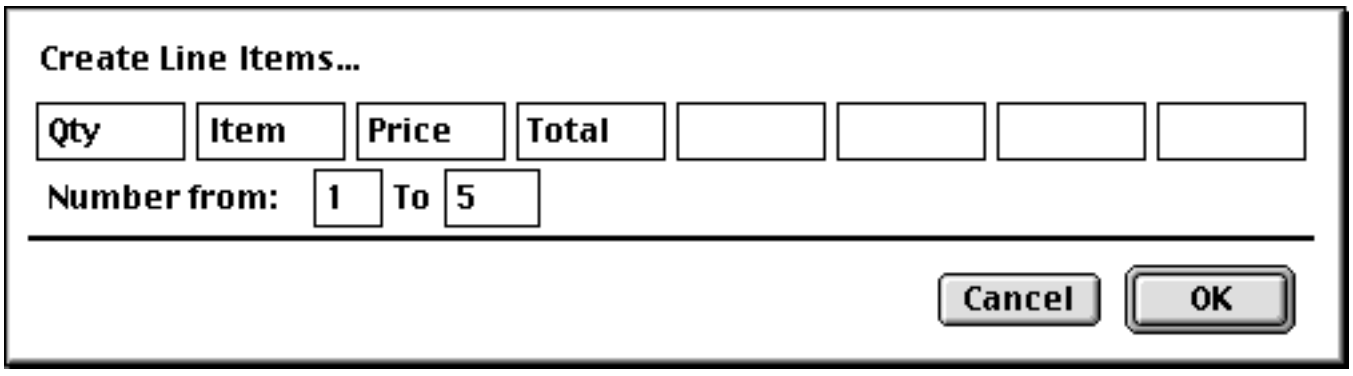
The design sheet has a shortcut for creating multiple line items in a hurry—the **Create Line Items** dialog in the Special menu. To use this dialog start by positioning the cursor just above the spot where you want the line items inserted. In the illustration below, the line items will be inserted between the Zip and Subtotal fields.



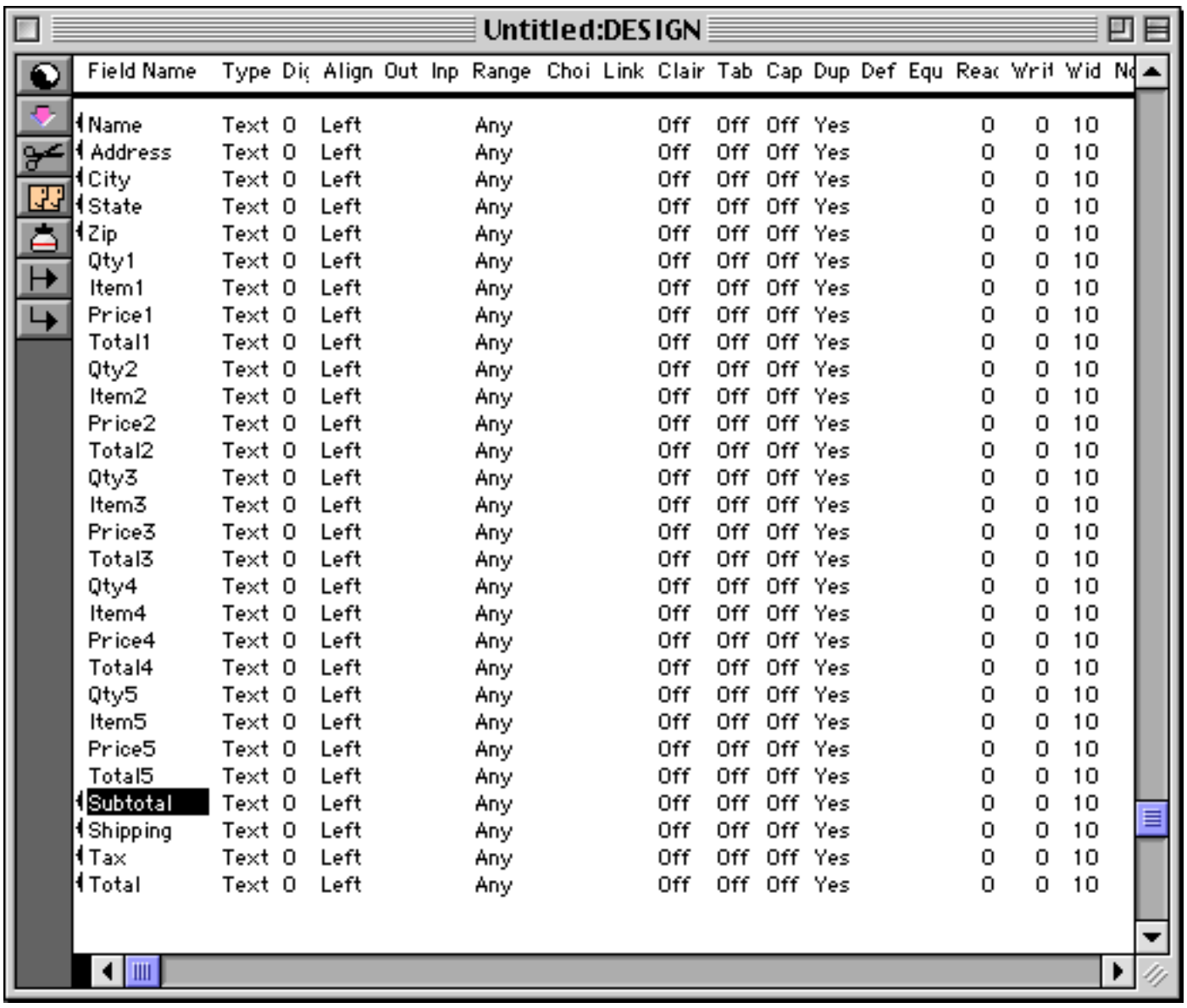
Once the cursor is in the correct spot choose **Create Line Items** from the Special menu.



The dialog allows you to enter up to eight root line item names (for example Qty, Item, Price, etc.). You can also specify the starting and ending numeric suffixes (for example 1 through 9).



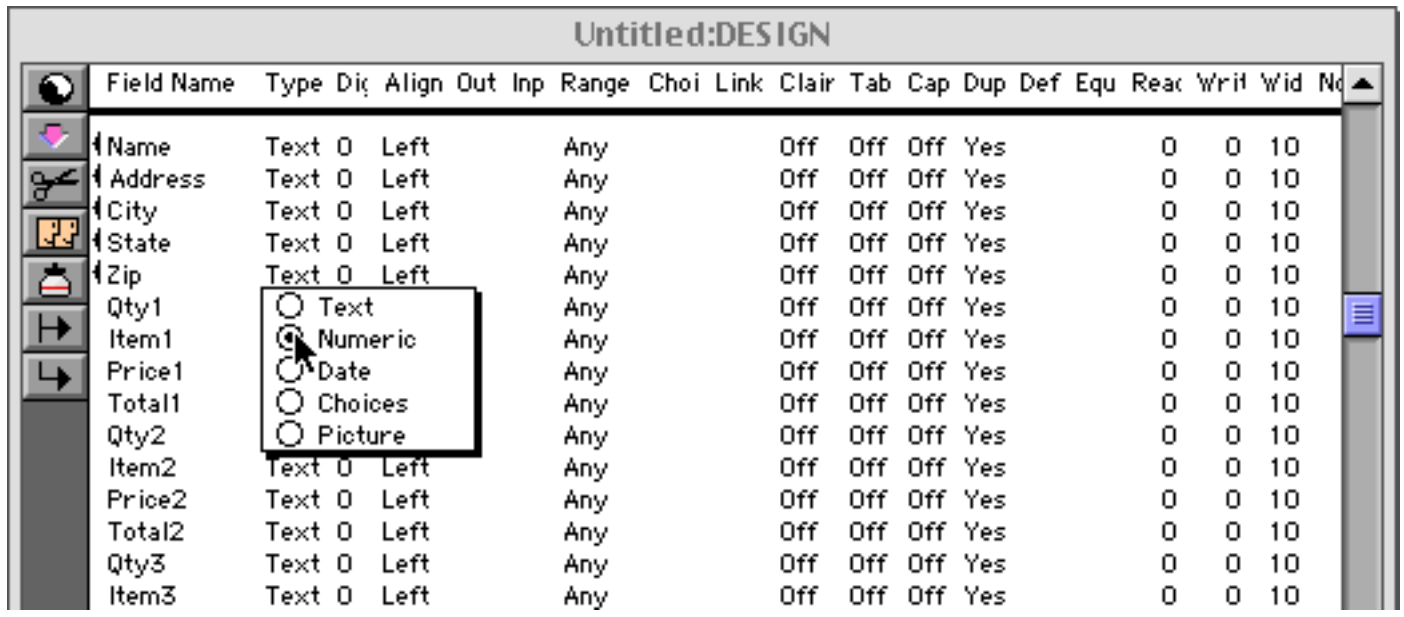
When you press **OK** Panorama will automatically insert the line items into the design sheet. The line items are inserted below the currently selected field.



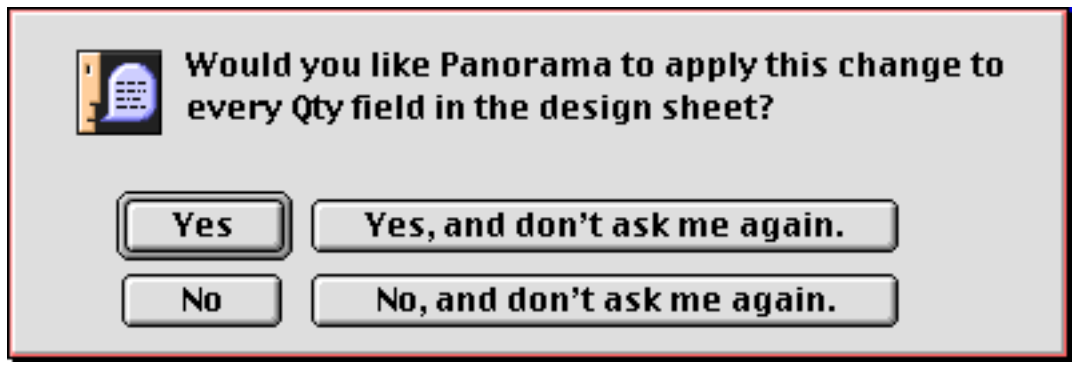
The new line item fields are actually added to the database when a new generation is created (see “Database “Generations”” on page 252).

Modifying Line Item Fields

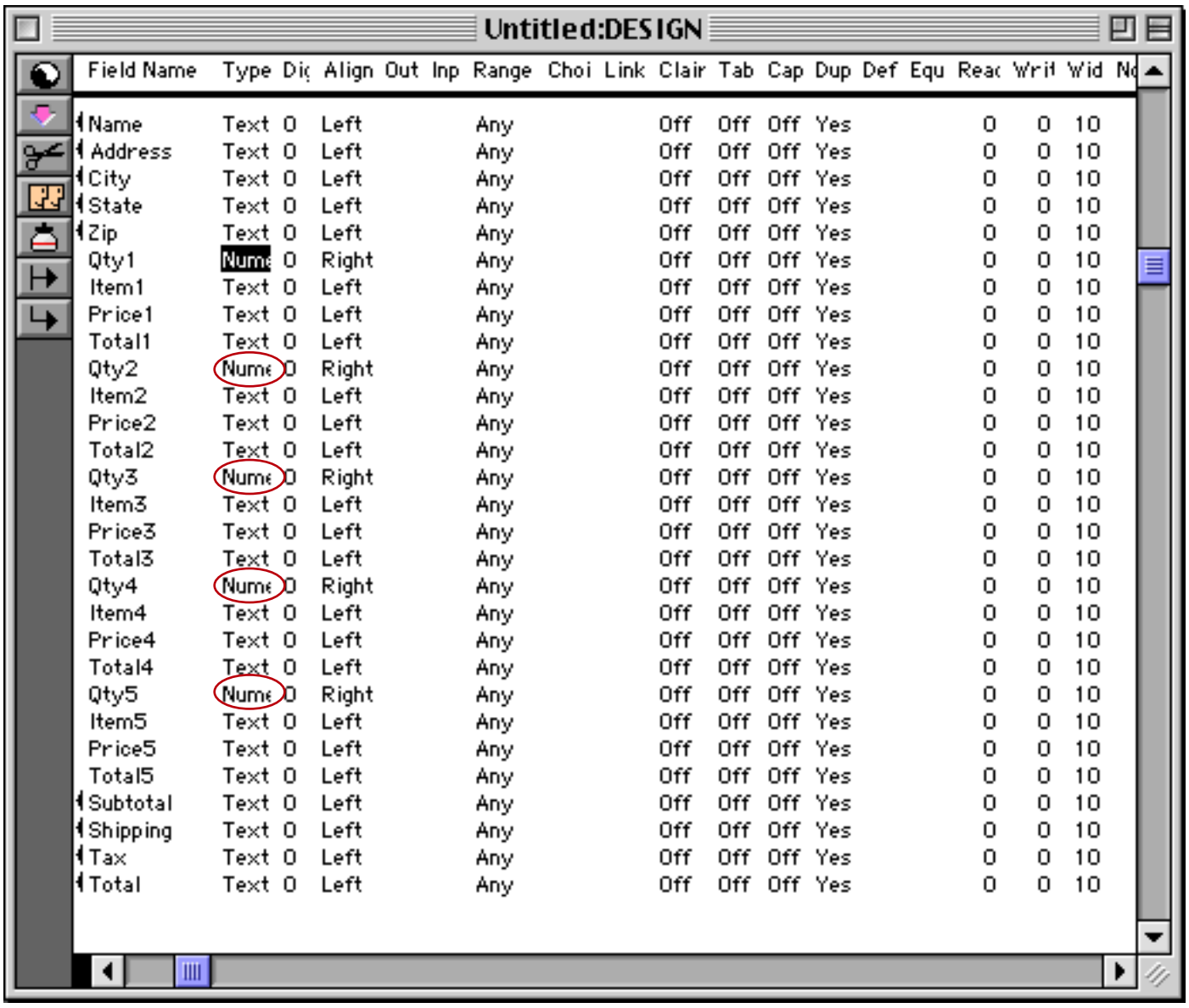
When you modify the properties of a line item field, you usually want to make the same change to all the corresponding fields. For example, if you change Qty1 to numeric, you probably also want to change Qty2, Qty3,Qty4…Qty15 to numeric also. The design sheet can do this for you. Whenever you change any of the properties of a particular line item field, Panorama will ask you if you would like to make the same change to all of the other corresponding line item fields. To illustrate this, let's go ahead and change Qty1 to numeric.



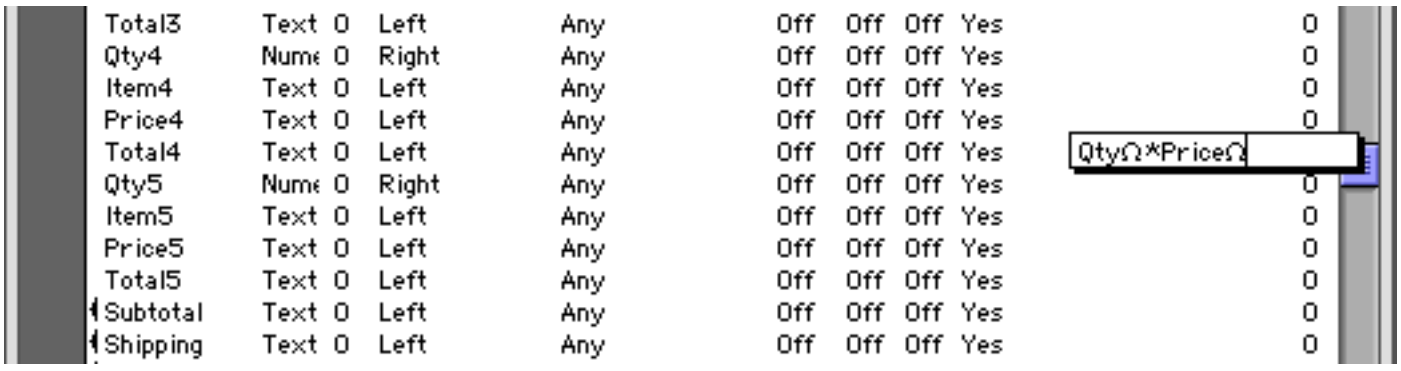
When you press the **Enter** key you will be presented with a choice.



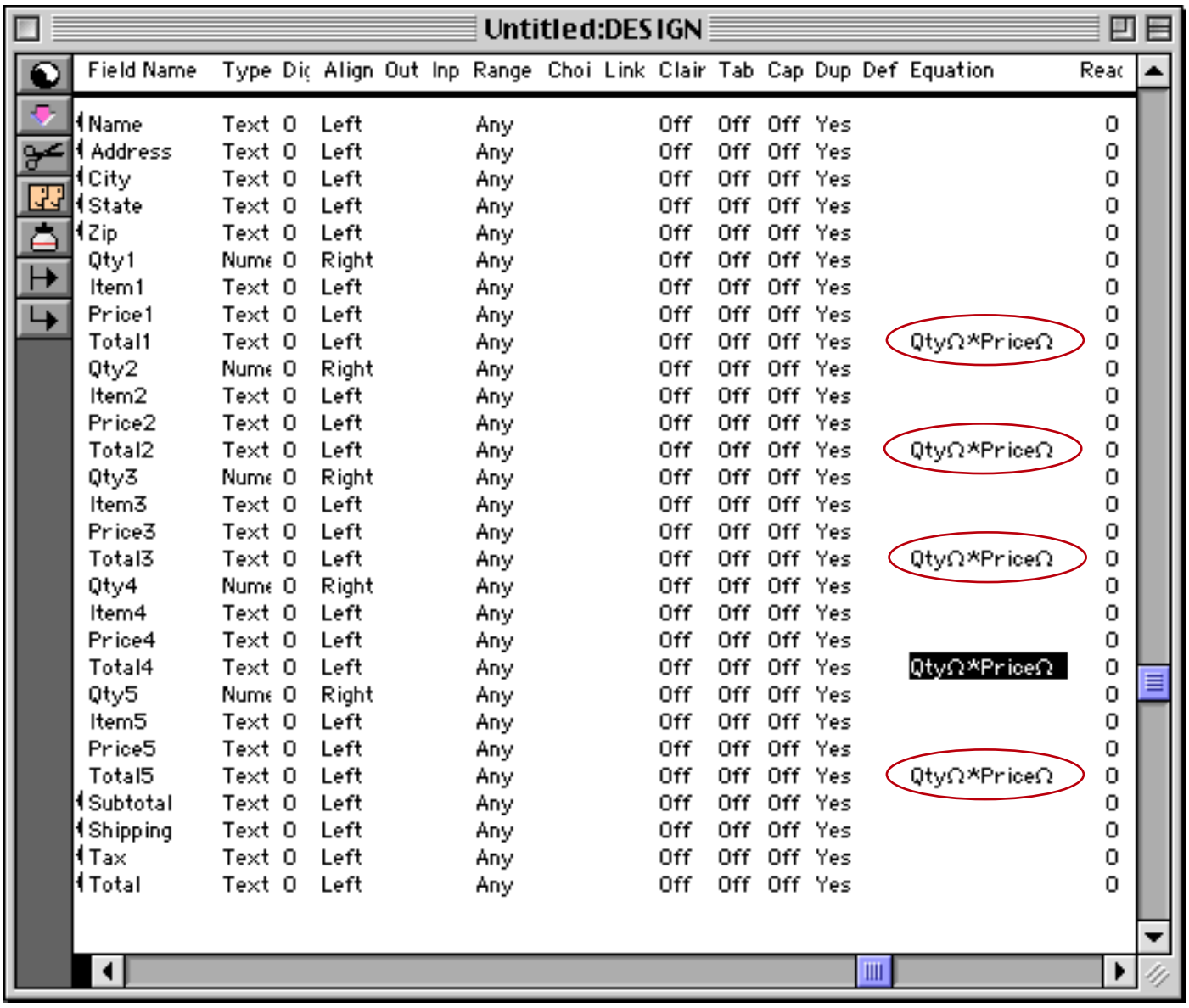
If you press **Yes** (or the **Enter** key), Panorama will go ahead and change all of the Qty fields to numeric.



This auto-repeat feature works for any column in the design sheet (except the Field Name). For example, we could type a formula into the Total4 field.



When you press **Enter** Panorama can automatically repeat this formula to all the other Total fields.

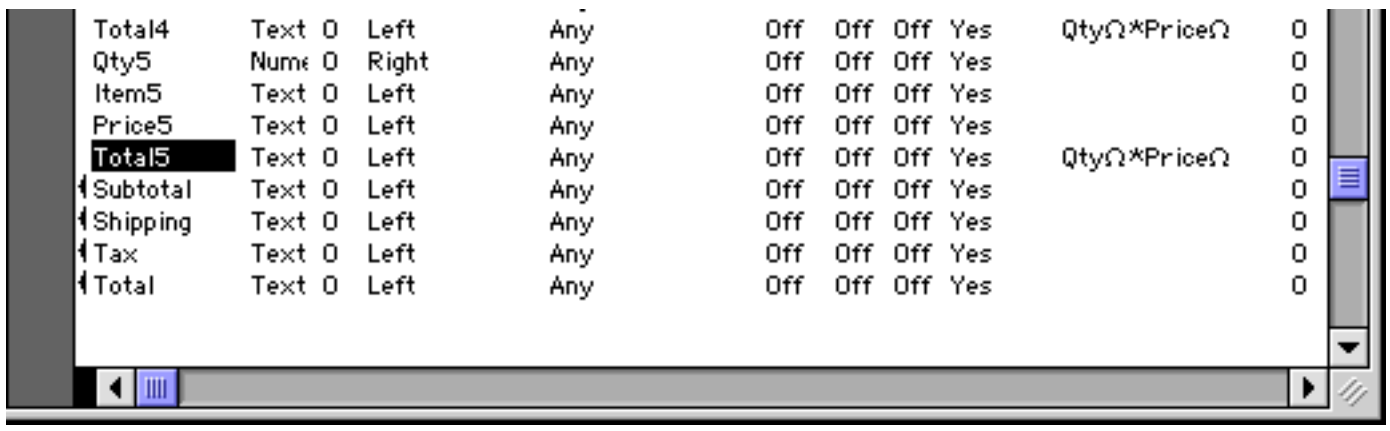


To learn more about creating formulas for line items (and those funny Ω characters) see “Line Item Fields” on page 1198.

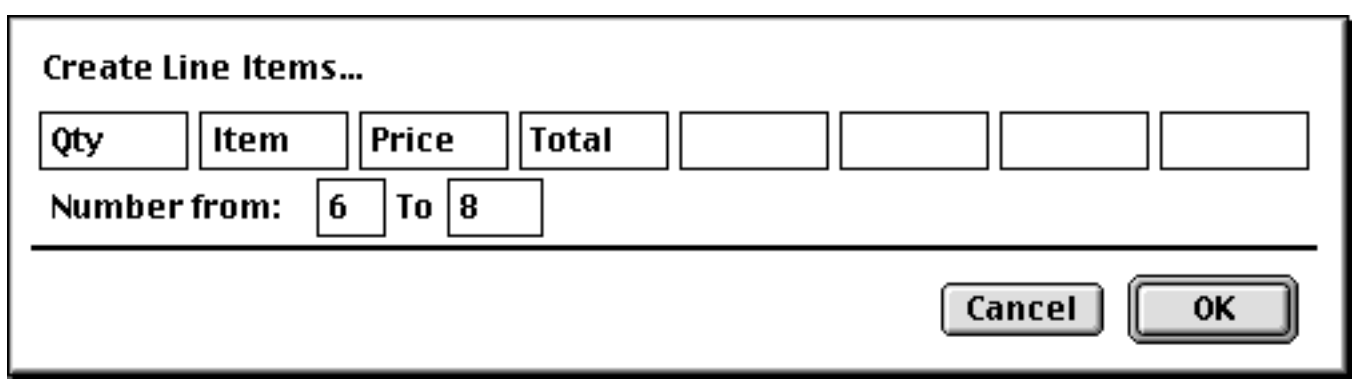
If you press **Yes, and don’t ask me again** or **No, and don’t ask me again** Panorama will remember your choice for as long as the design sheet is open. If you close the design sheet and then re-open it later, however, Panorama will start asking you again (until you tell it not to again!).

Adding More Line Item Fields

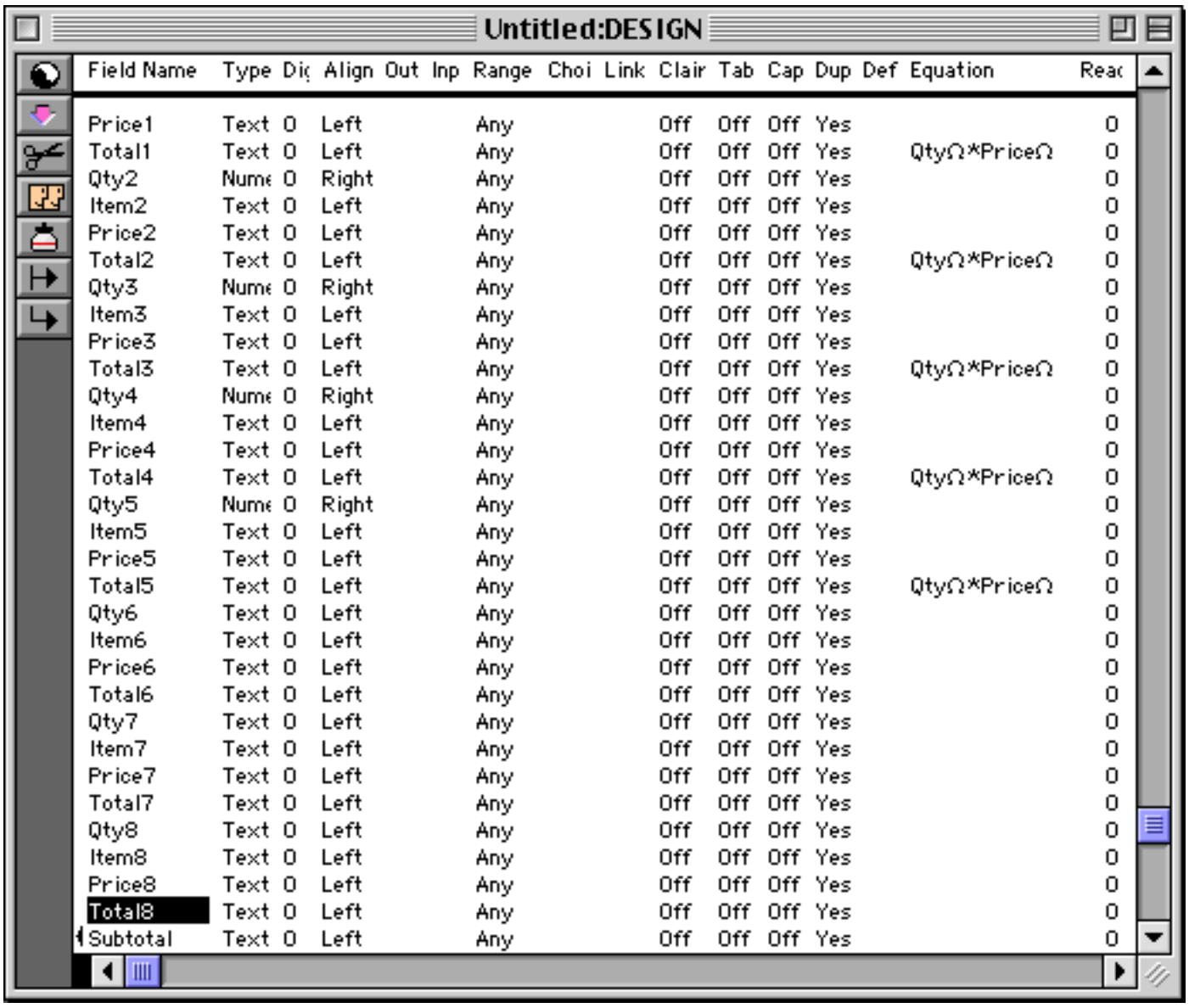
If necessary you can use the **Create Line Items** dialog to add new line items at any time. For example, suppose we wanted to add three more line items to the five we created earlier. Start by placing the cursor on the last line item.



Now open the **Create Line Items** dialog and type in the field name roots again. Set the numbering to start just past the previously created line items (in this case 6).



Panorama adds the new line items.



Field Name	Type	Dig	Align	Out	Inp	Range	Choi	Link	Clair	Tab	Cap	Dup	Def	Equation	Reac
Price1	Text	0	Left			Any			Off	Off	Off	Yes			0
Total1	Text	0	Left			Any			Off	Off	Off	Yes		Qty*Price	0
Qty2	Nume	0	Right			Any			Off	Off	Off	Yes			0
Item2	Text	0	Left			Any			Off	Off	Off	Yes			0
Price2	Text	0	Left			Any			Off	Off	Off	Yes			0
Total2	Text	0	Left			Any			Off	Off	Off	Yes		Qty*Price	0
Qty3	Nume	0	Right			Any			Off	Off	Off	Yes			0
Item3	Text	0	Left			Any			Off	Off	Off	Yes			0
Price3	Text	0	Left			Any			Off	Off	Off	Yes			0
Total3	Text	0	Left			Any			Off	Off	Off	Yes		Qty*Price	0
Qty4	Nume	0	Right			Any			Off	Off	Off	Yes			0
Item4	Text	0	Left			Any			Off	Off	Off	Yes			0
Price4	Text	0	Left			Any			Off	Off	Off	Yes			0
Total4	Text	0	Left			Any			Off	Off	Off	Yes		Qty*Price	0
Qty5	Nume	0	Right			Any			Off	Off	Off	Yes			0
Item5	Text	0	Left			Any			Off	Off	Off	Yes			0
Price5	Text	0	Left			Any			Off	Off	Off	Yes			0
Total5	Text	0	Left			Any			Off	Off	Off	Yes		Qty*Price	0
Qty6	Text	0	Left			Any			Off	Off	Off	Yes			0
Item6	Text	0	Left			Any			Off	Off	Off	Yes			0
Price6	Text	0	Left			Any			Off	Off	Off	Yes			0
Total6	Text	0	Left			Any			Off	Off	Off	Yes			0
Qty7	Text	0	Left			Any			Off	Off	Off	Yes			0
Item7	Text	0	Left			Any			Off	Off	Off	Yes			0
Price7	Text	0	Left			Any			Off	Off	Off	Yes			0
Total7	Text	0	Left			Any			Off	Off	Off	Yes			0
Qty8	Text	0	Left			Any			Off	Off	Off	Yes			0
Item8	Text	0	Left			Any			Off	Off	Off	Yes			0
Price8	Text	0	Left			Any			Off	Off	Off	Yes			0
Total8	Text	0	Left			Any			Off	Off	Off	Yes			0
Subtotal	Text	0	Left			Any			Off	Off	Off	Yes			0

These new items do not have any of the modifications you may have made to the existing line items (type, equation, etc.) To update the new items simply edit open and close the appropriate data cells in one of the line items that are already set up and let Panorama copy the changes into the new line items (see “Modifying Line Item Fields” on page 265).

Learn More About Line Items

Line items get special treatment in the design sheet, in formulas, and in form layout. To learn about set up formulas for calculating line items see “Line Item Fields” on page 1198 and “Adding Line Item Fields” on page 1207. You’ll also find some examples of line item calculations in “Automatic Calculations” on page 340. To learn how to automatically create rows and columns of line item cells in a form see “Line Items in a Form” on page 669. To learn how to adjust the width of an entire column of line item cells in a form see “Cluster Resize” on page 541. To learn how to change the font size of a table of line item cells see “Adjusting Spacing Between Multiple Objects” on page 556.

“Generic” Fields

Databases come in all sizes and shapes. Generic fields allow different databases to share information even if they have different field names or slightly different configurations. For example, one database may store company names in a field named Company, while another may have a similar field named Organization. By setting up generic fields for each database, you build a bridge so that Panorama knows that these two fields, though named differently, contain the same type of information. Once this bridge is built Panorama can exchange data between these two databases (for example by drag and drop), and between Panorama and other applications that can share information (for example Apple’s Address Book).

Unlike normal Panorama fields, generic fields don’t actually store any information themselves. Instead, a generic field simply references one or more normal Panorama fields that contain the same information. When Panorama needs to access the information in the generic field, it examines the generic field definition (see below) to extract the data from the actual database fields.

Standard Generic Fields

While ordinary Panorama fields can store an unlimited variety of information, there are only about two dozen generic fields to choose from (this list may grow in future releases). Not all databases will be able to support all of these generic fields, but a database that contains contact information (mailing lists, phone books, etc.) will be able to support many of them.

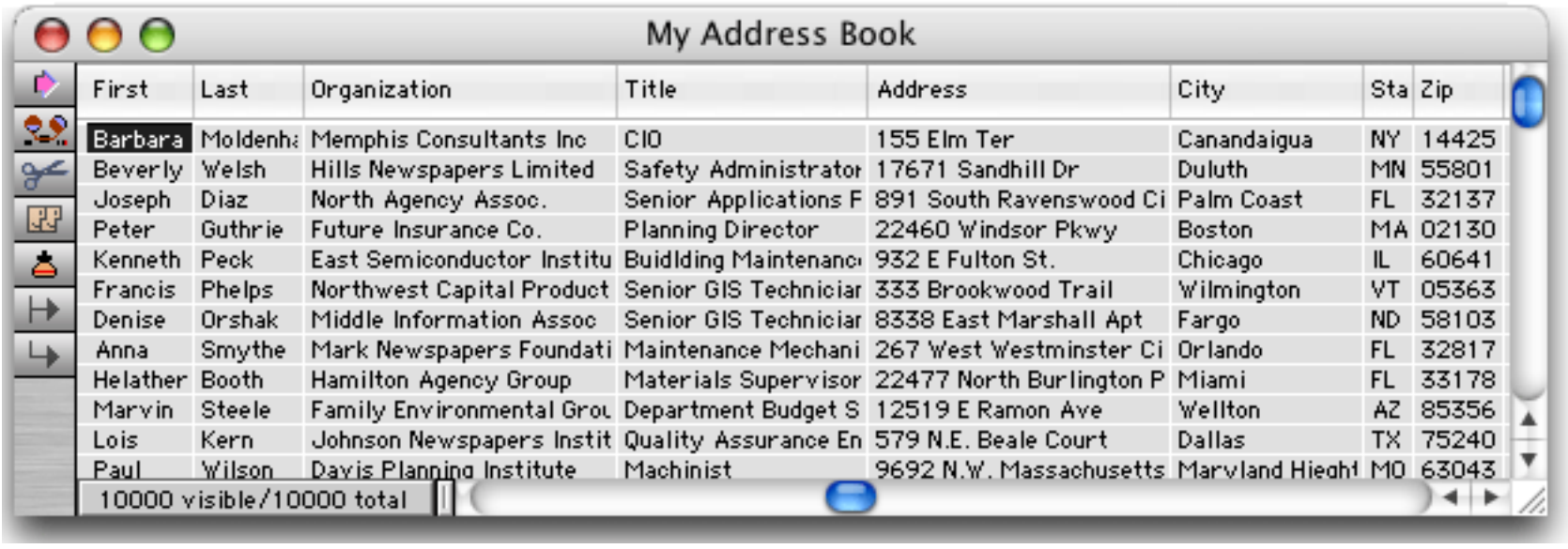
Category	Generic Field	Description	Examples
Contact	Name	A person’s first and last name.	John Wilson, Mary Furnare
	Nickname	A person’s nickname.	Sammy, Beth
	Prefix	A person’s honorific.	Mr., Ms., Dr.
	First	A person’s first name.	John, Mary
	Middle	A person’s middle name or initial.	August, Walker
	Last	A person’s last name.	Wilson, Furnare
	Suffix	A person’s suffix.	Jr., D.D.S
	Formal	A person’s honorific and last name.	Dr. Wilson, Ms. Furnare
	FullName	A person’s complete name.	Dr. John August Wilson
Affiliation	Position	A person’s title or position within an organization.	President, Sales Manager
	Organization	The company, school, agency or other organization this person is affiliated with (if any).	Acme Construction
	Division	The major division or group this person is affiliated with (if any).	Communications Products
	Department	The department this person is affiliated with (if any).	Human Resources
Address	Address	Street address.	1892 North Glendale Blvd.
	Suite	Suite, apartment or room number (if any)	Room 463
	City	City or town.	Saskatoon
	State	State or province.	SK
	Zip	Zip or postal code	4Y2 A8G
	Country	Country	Canada
Internet	Email	Email address. This field may contain multiple e-mail addresses, one per line.	maryf@acme.com
	Web	URL for web site	http://www.acme.com/cp/hr/

Category	Generic Field	Description	Examples
Telephone (separate)	Voice	Primary phone number.	(303) 442-9011
	Fax	Fax phone number.	(888) 763-1290
	Cell	Cell or mobile phone number.	(909) 487-2063
	Home	Home phone number.	(562) 491-3992
	Work	Word phone number	(324) 987-3367 ext 2384
	Pager	Pager number.	(817) 335-9832
Telephone (line items / array)	PhoneType	The PhoneType and Phone fields are composite fields designed for holding multiple phone numbers. Use these generic fields when the underlying database contains phone numbers in line items or an array instead of in separate fields. Each line in these fields corresponds to a single phone number. Each line in the PhoneType generic field contains the type of the corresponding phone number: Home, Office, Cell, Fax, etc. Each line in the Phone generic field contains the actual phone number.	Voice Cell Fax
	Phone		(303) 442-9011 (909) 487-2063 (888) 763-1290
Other	Notes	Notes, memos, comments, etc.	

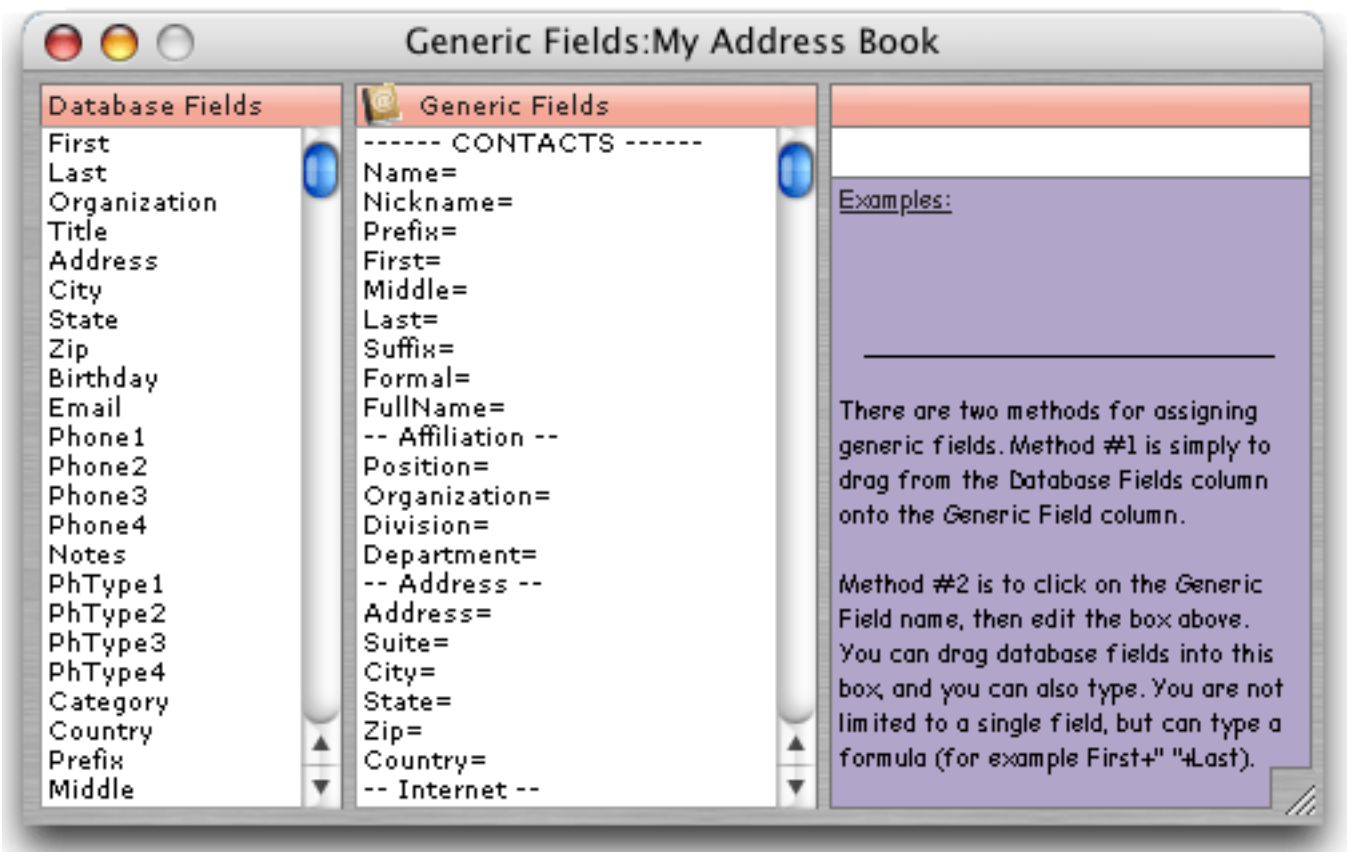
If you are familiar with the vCard specification (RFC 2426) for exchanging data between programs you'll recognize that many of these generic fields correspond to vCard fields.

Adding Generic Fields to an Existing Database

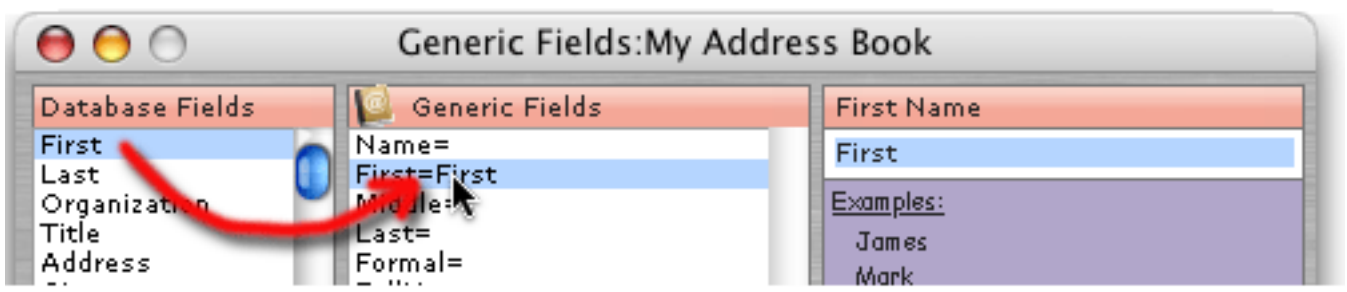
Normal Panorama fields are set up with the New Database wizard, Design Sheet, or Field Properties dialog. Generic fields are set up with the **Generic Fields** wizard, which is in the **Preferences** submenu of the Wizards menu. Generic fields can be added to an existing database at any time. The next few paragraphs will illustrate the process for setting up generic fields. This example will use a basic address book database, shown here.



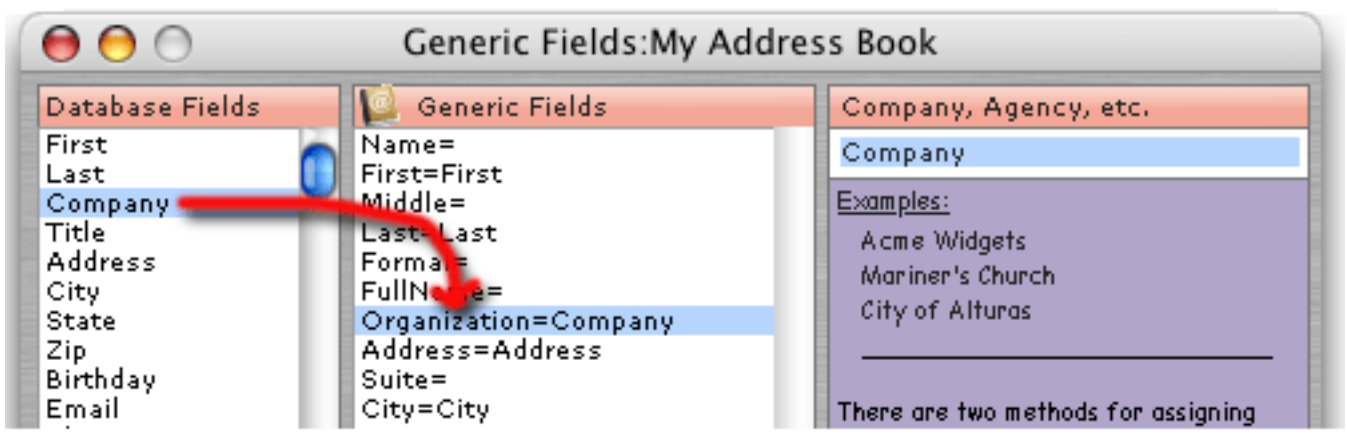
The first step is to open the **Generic Fields** wizard. The wizard is divided into three columns. The left column, Database Fields, lists the actual fields in the database. The second column, Generic Fields, lists the generic fields for this database. The generic fields column is divided into major sections (like CONTACTS) and minor sections (Affiliation, Address, etc.) Since we're just starting, none of the generic fields have been defined (everything to the right of the = sign is blank). The rightmost column is for assigning formulas to define generic fields, we'll discuss that further in a moment.



If an actual field in the database contains the exact data required by a generic field, you can simply drag the field from the Database Fields column to the appropriate generic field in the Generic Fields column.

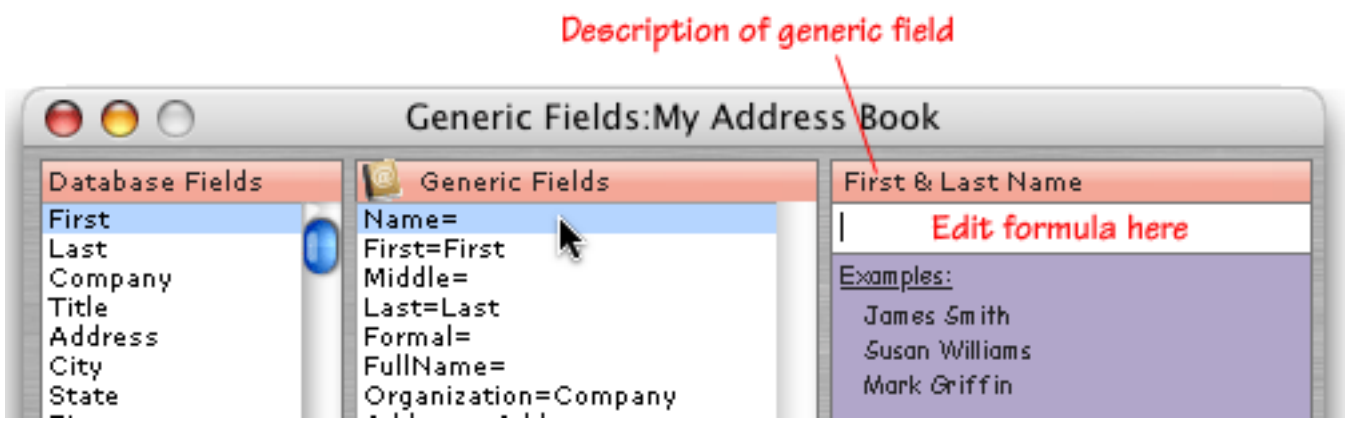


This drag and drop process must be repeated for each actual database field that corresponds to a generic field. Notice that the name of the database field does not have to be the same as the name of the generic field, as long as both contain the same data. (In this case, the actual database field that contains the organization name is called Company.)

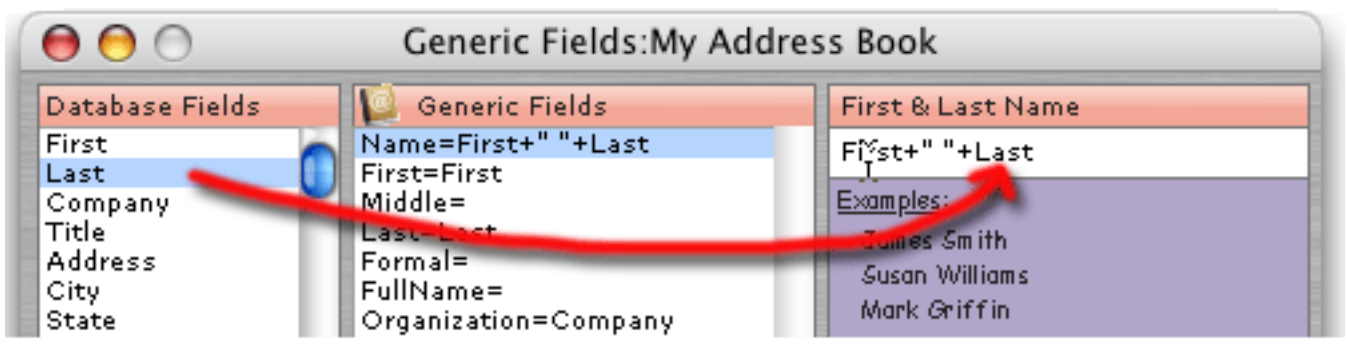


Assigning Multiple Database Fields to a Generic Field

In some cases there will be no single database field that corresponds to a generic field. In that case you may be able to create a formula for the generic field. For our example, our sample database has no combined Name field (first and last names). However, we can simulate that field with the formula **First+" "+Last** (see "Text Formulas" on page 1212 for more information on how this formula was constructed). Instead of dragging to set up the generic field, it must be typed into the right hand column of the wizard. For example, to set up a formula for the generic Name field, first click on that generic field. This enables the rightmost pane of the window, which is used for editing formulas.



You can type in the formula, and you can also drag field names from the Database Fields column.



Note: A generic field that is defined with a formula can be used to export data, but not to import it.

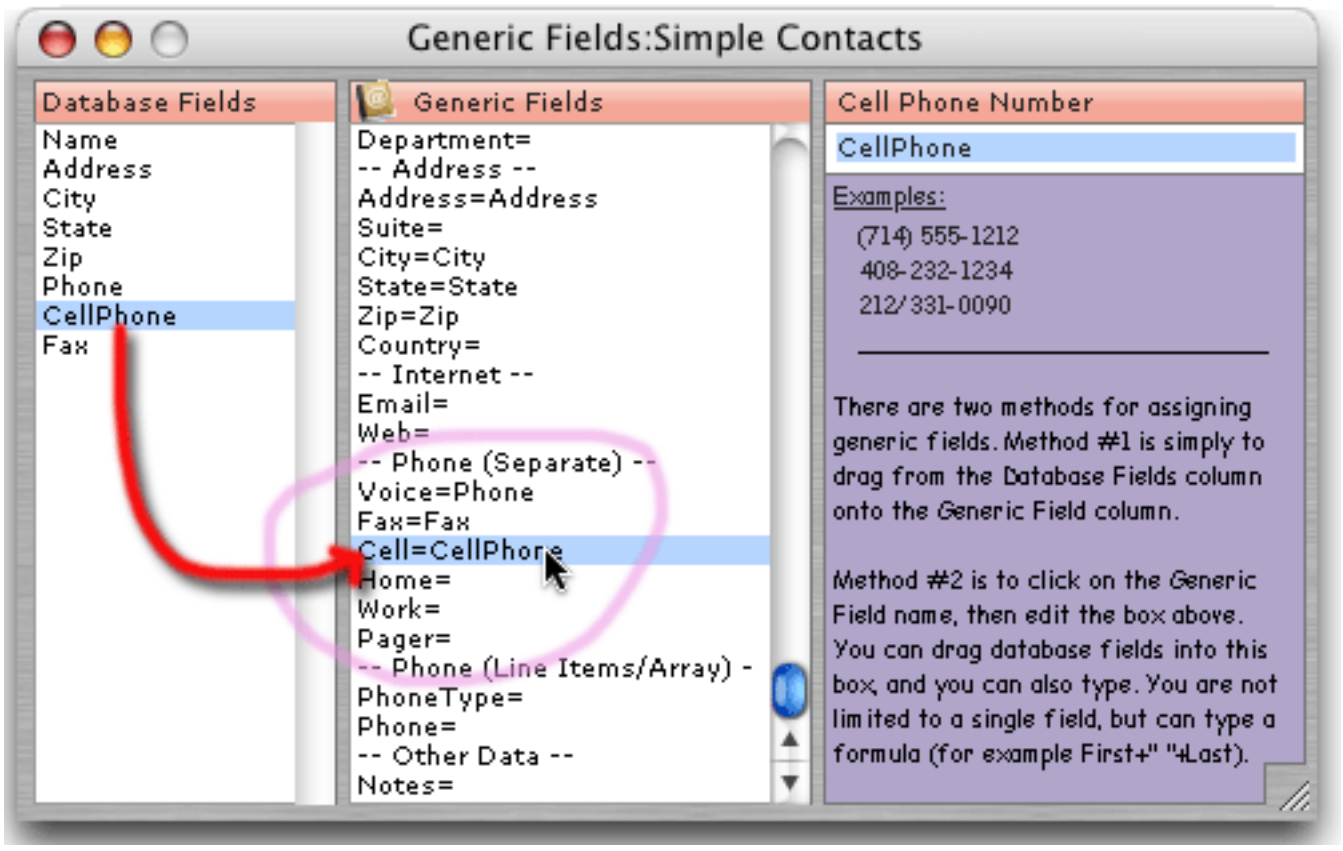
Configuring Generic Phone Number Fields

The Generic Fields wizard supports three different ways to store phone numbers: separate fields, line items, and arrays.

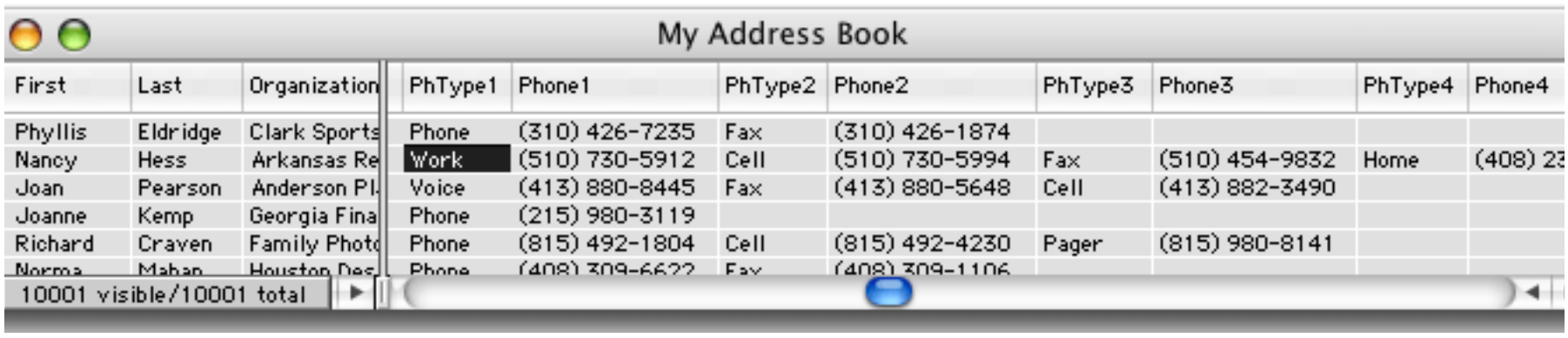
Probably the most common method for storing phone numbers is to store each number in a separate field. For example, the database might have a field for voice numbers, another field for fax numbers, another for cell phones, etc. Here is a typical example of a database with separate fields for main phone number, cell phone and fax.



To link these separate fields to generic fields, scroll down to the Phone (Separate) section of the Generic Fields column and drag the database fields to the corresponding generic fields.



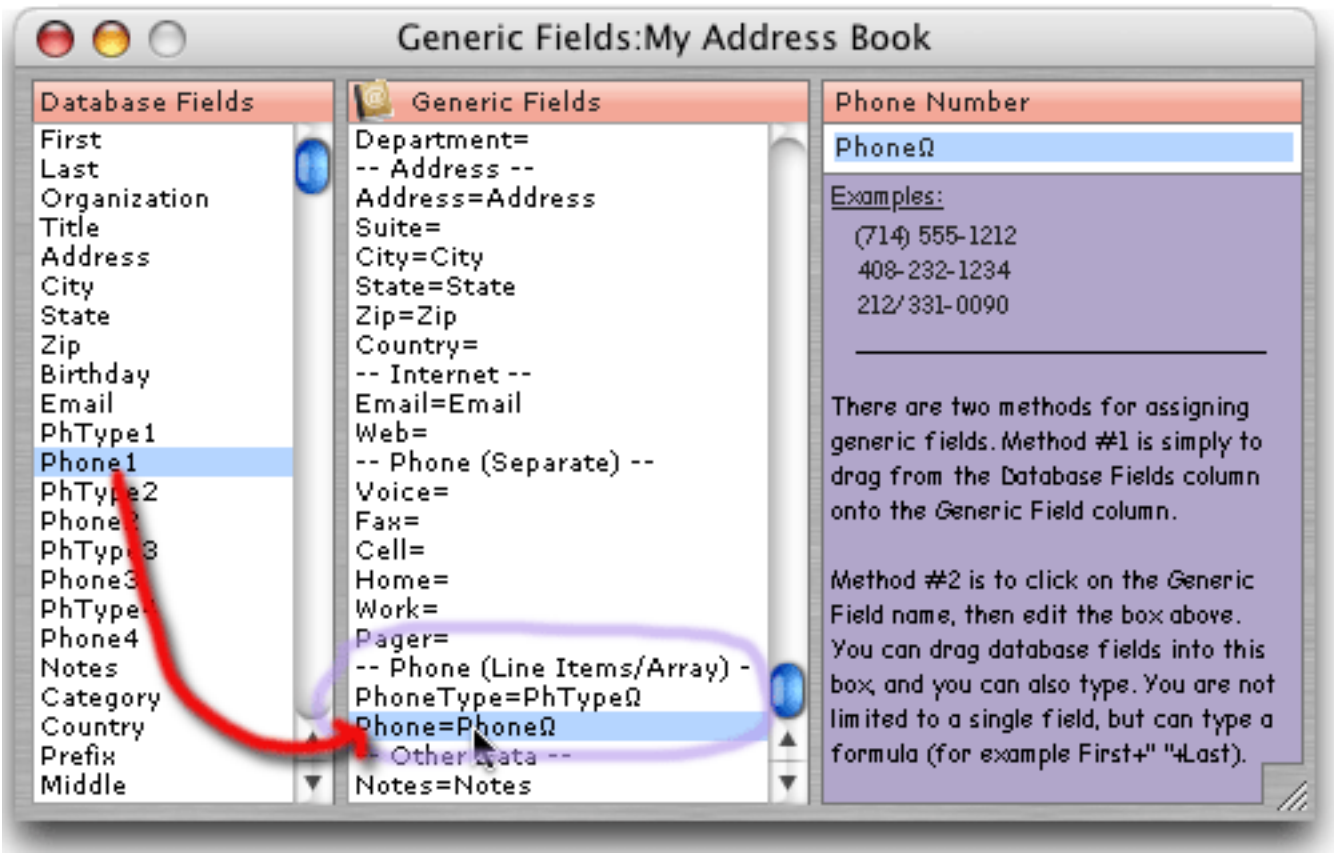
The second method for storing phone numbers is to use line item fields. Line item fields are repeating fields with a numeric suffix, for example Phone1, Phone2, Phone3, etc. (see "Repeating Fields (Line Items)" on page 262 for a general discussion of line item fields). The Generic Fields wizard assumes that phone numbers are stored in two sets of line item fields, phone types (voice, fax, cell, etc.) and the actual phone numbers themselves. Here's a typical example with four sets of line items:



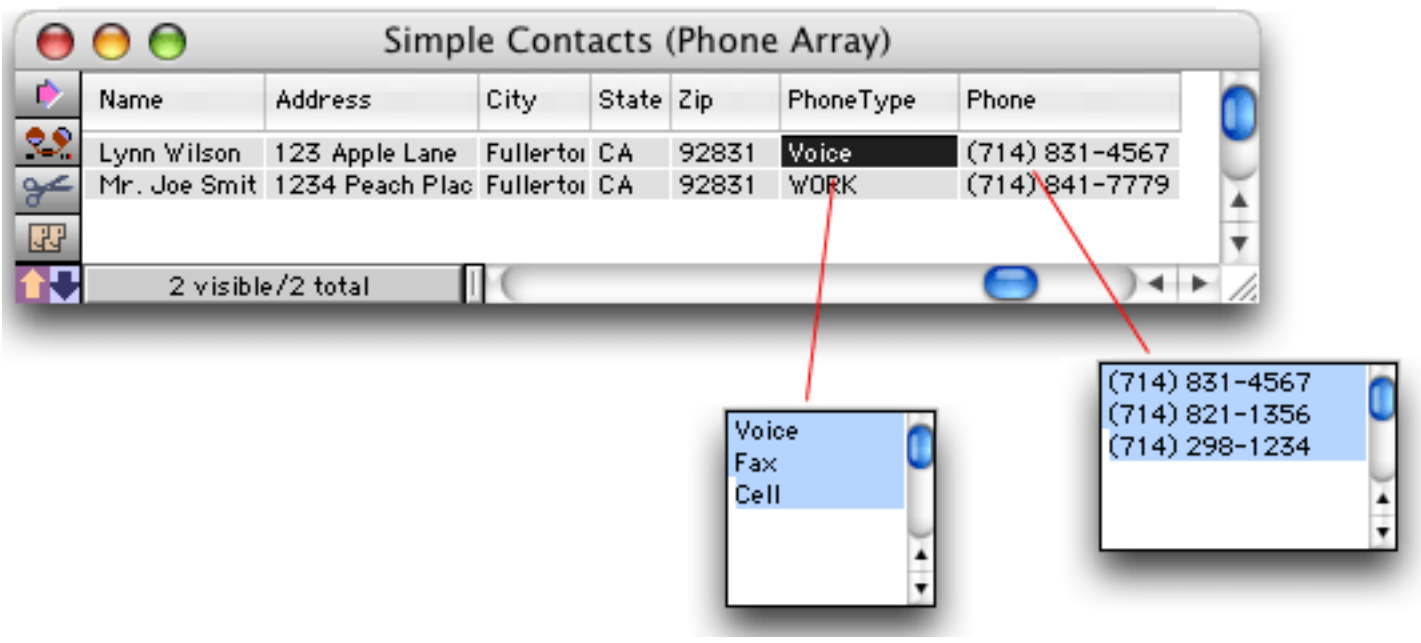
Line item fields like this are usually arranged in rows and columns on a form, like this:



To link these line item fields to generic fields, scroll down to the Phone (Line Items/Array) section of the Generic Fields column. Drag the first phone type line item field (in this case PhType1) over the PhoneType generic field. Then drag the first phone number field (in this case Phone1) over the Phone generic field. As you release the mouse you'll notice that the field number changes from 1 to Ω, as shown below. This confirms that all of the line item fields are linked to this generic field.



The final method for storing phone numbers is to use carriage return delimited arrays. In this case there are two fields, with each line corresponding to a single phone number.

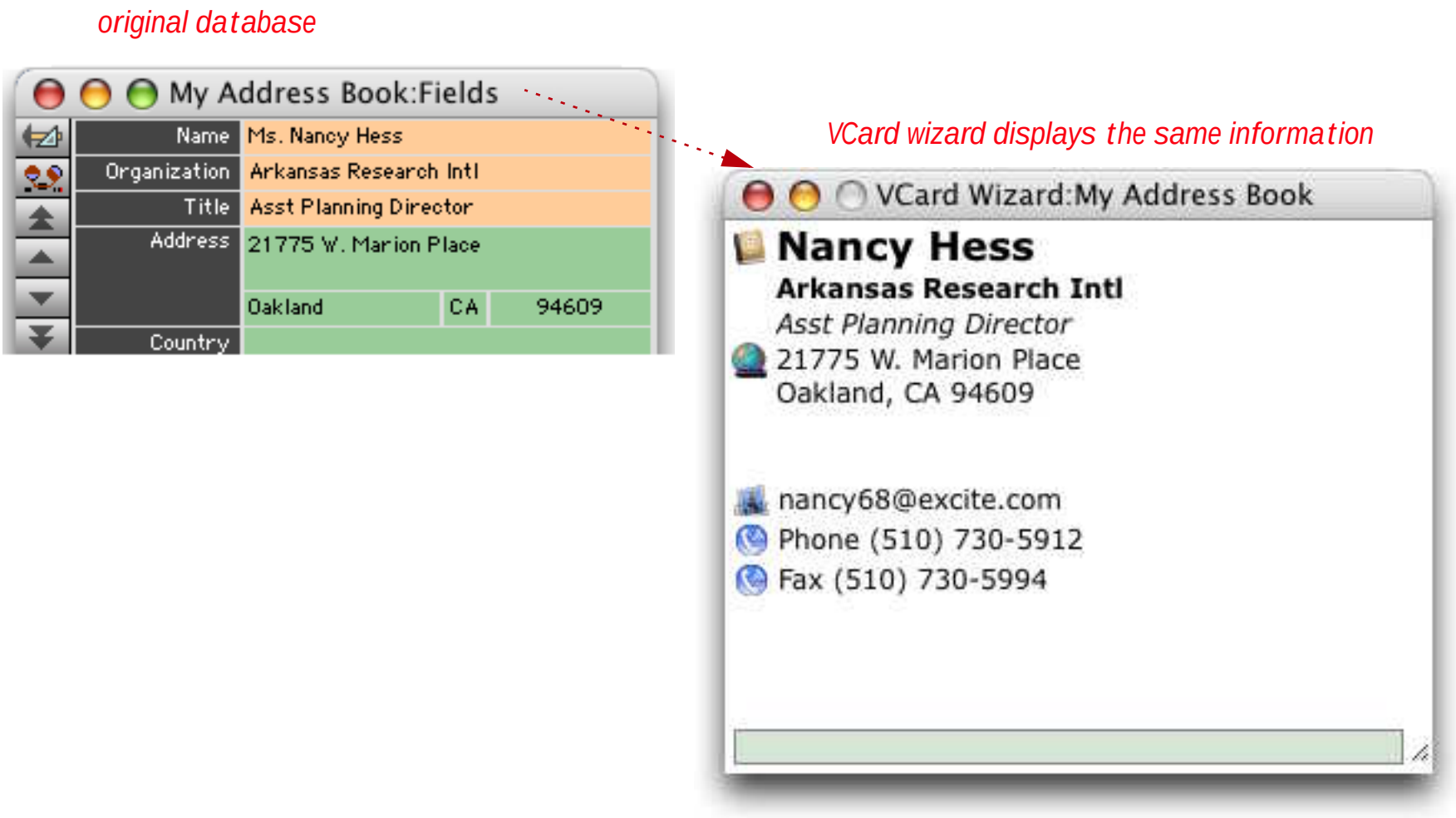


This arrangement allows you to store an unlimited number of phone numbers in only two fields. To link these two fields to generic fields, scroll down to the Phone (Line Items/Array) section of the Generic Fields column. Drag the phone type field (in this case PhoneType) over the PhoneType generic field. Then drag the phone number field (in this case Phone) over the Phone generic field.

Using Generic Fields with the VCard Wizard

Once generic fields have been set up they can be used to transfer data between the database and other databases that also have generic fields, or between the database and applications that support vCards. For example an address could be copied to Apple's address book, or used to display a map. A phone number can be used to actually dial the phone, or an e-mail address to send an e-mail.

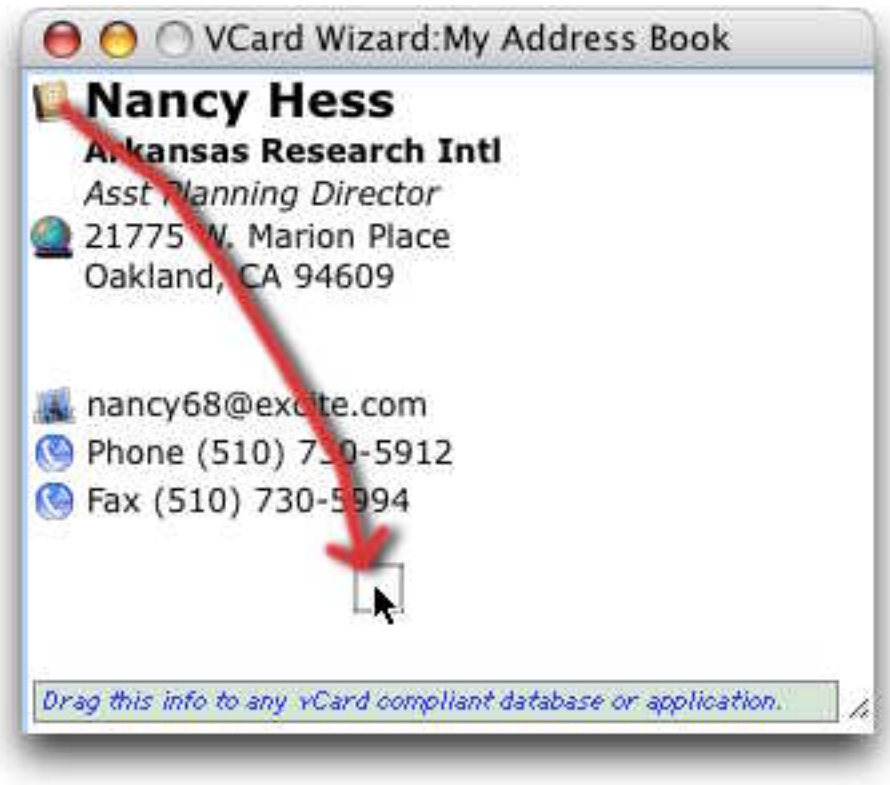
The slickest way to use generic fields is to program them into your database itself (see "VCard Drag and Drop" on page 1776). However, it's not necessary to do any programming to use generic fields. The **VCard Wizard** allows you to use generic fields without any programming at all. You'll find this wizard in the **Utilities** submenu of the Wizard menu. When you first open this wizard it will display the generic data from the current database, as shown below. (If the current database doesn't have any generic fields, it will display an error message.)



If you later move to a different record in the My Address Book database, or edit the data, the **VCard Wizard** will automatically update when you bring it back to the front.

Dragging Data to Other Applications or Databases

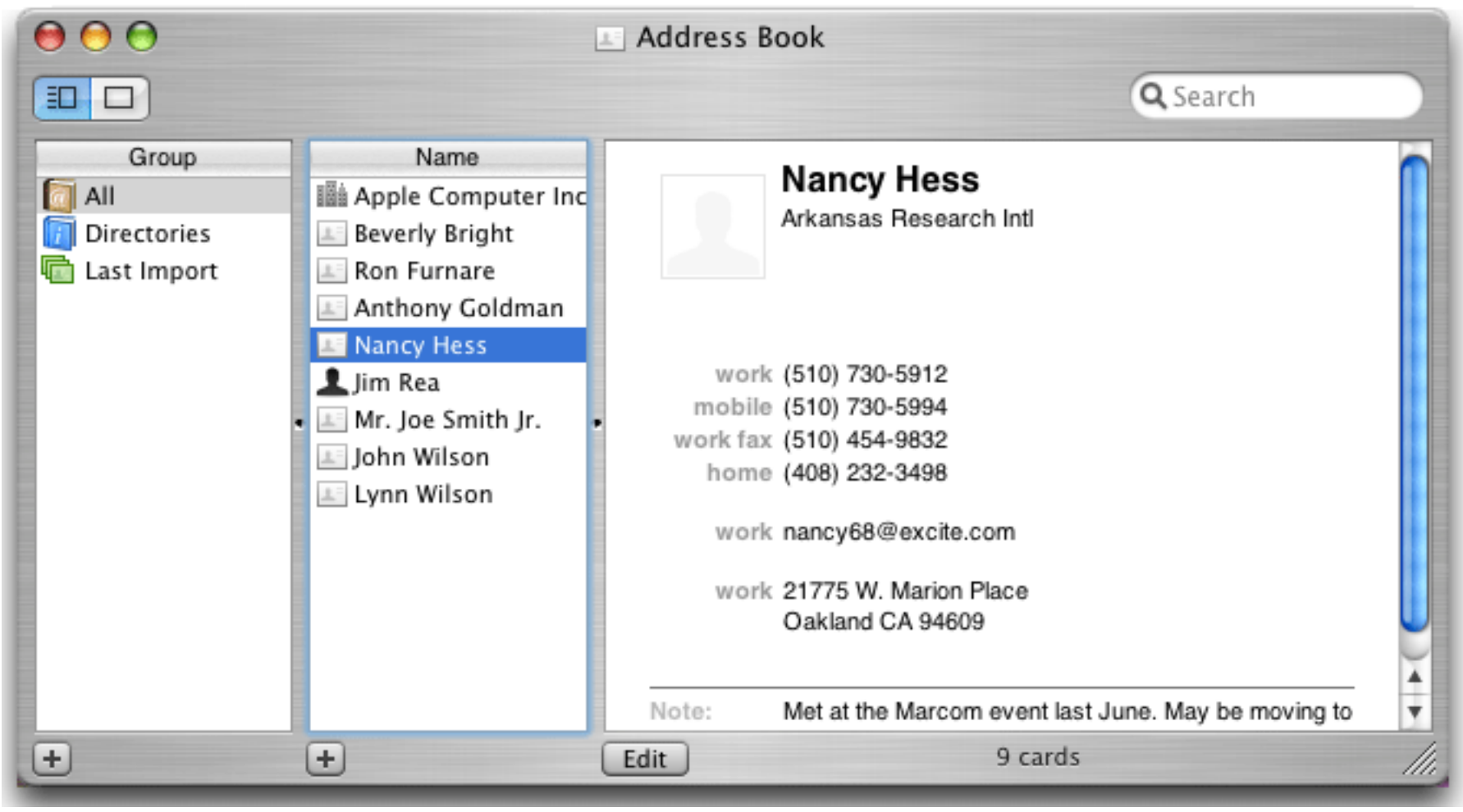
By dragging the phone book icon you can drag data to Apple's Address Book (or any other application that supports VCards) or to any database that has generic fields.



Drag onto the Address Book (or any VCard compatible application).



When you release the mouse, the data will be added to a new card in the Address Book.

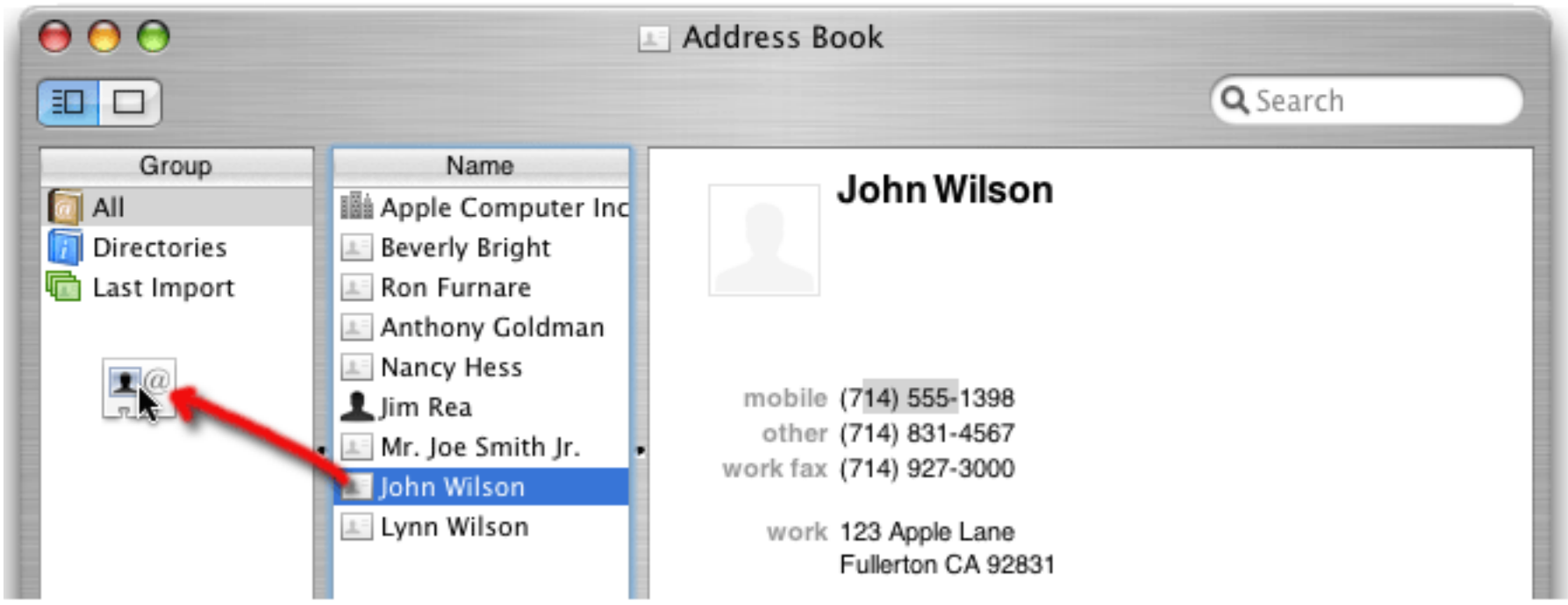


You can also drag onto other databases that support generic fields. In this example the same person has been dragged onto the **Mini Correspondence** wizard, creating a new letter to Ms. Hess.

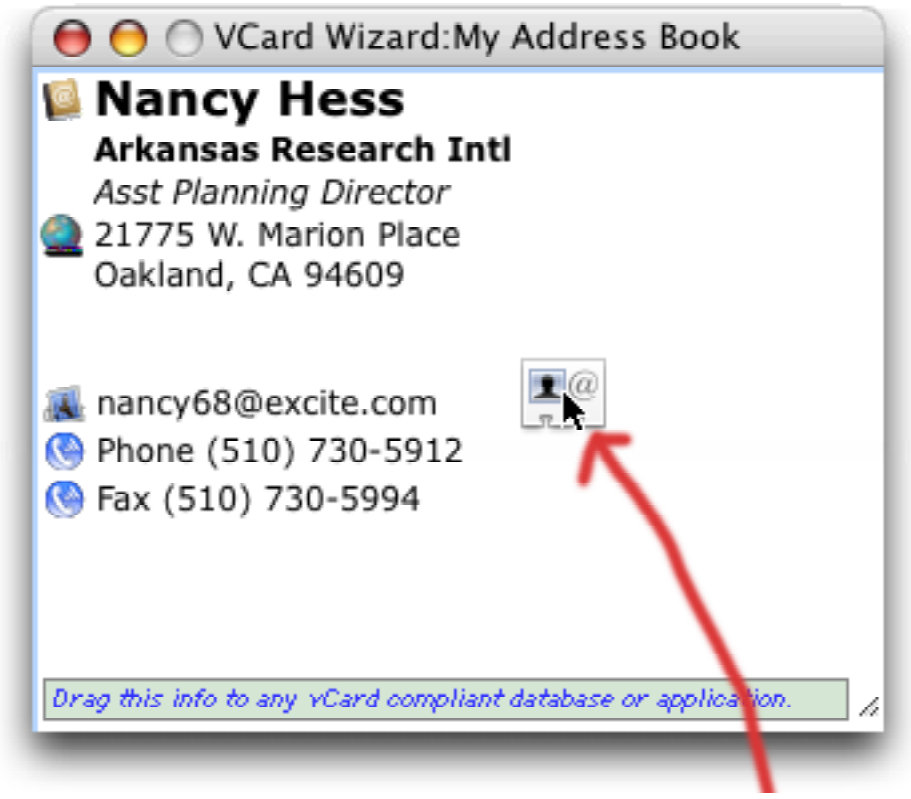


Dragging Data from other Applications or Databases

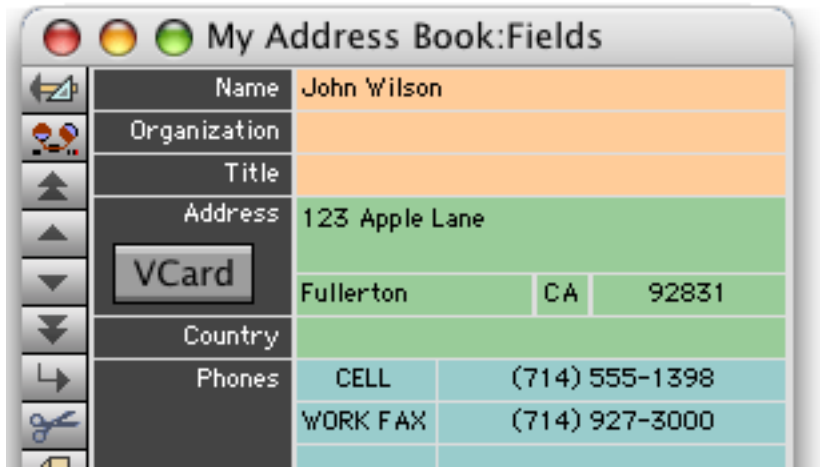
To copy data from the Address Book (or any VCard compatible application) to your database, simply drag to anywhere on the **VCard Wizard**.



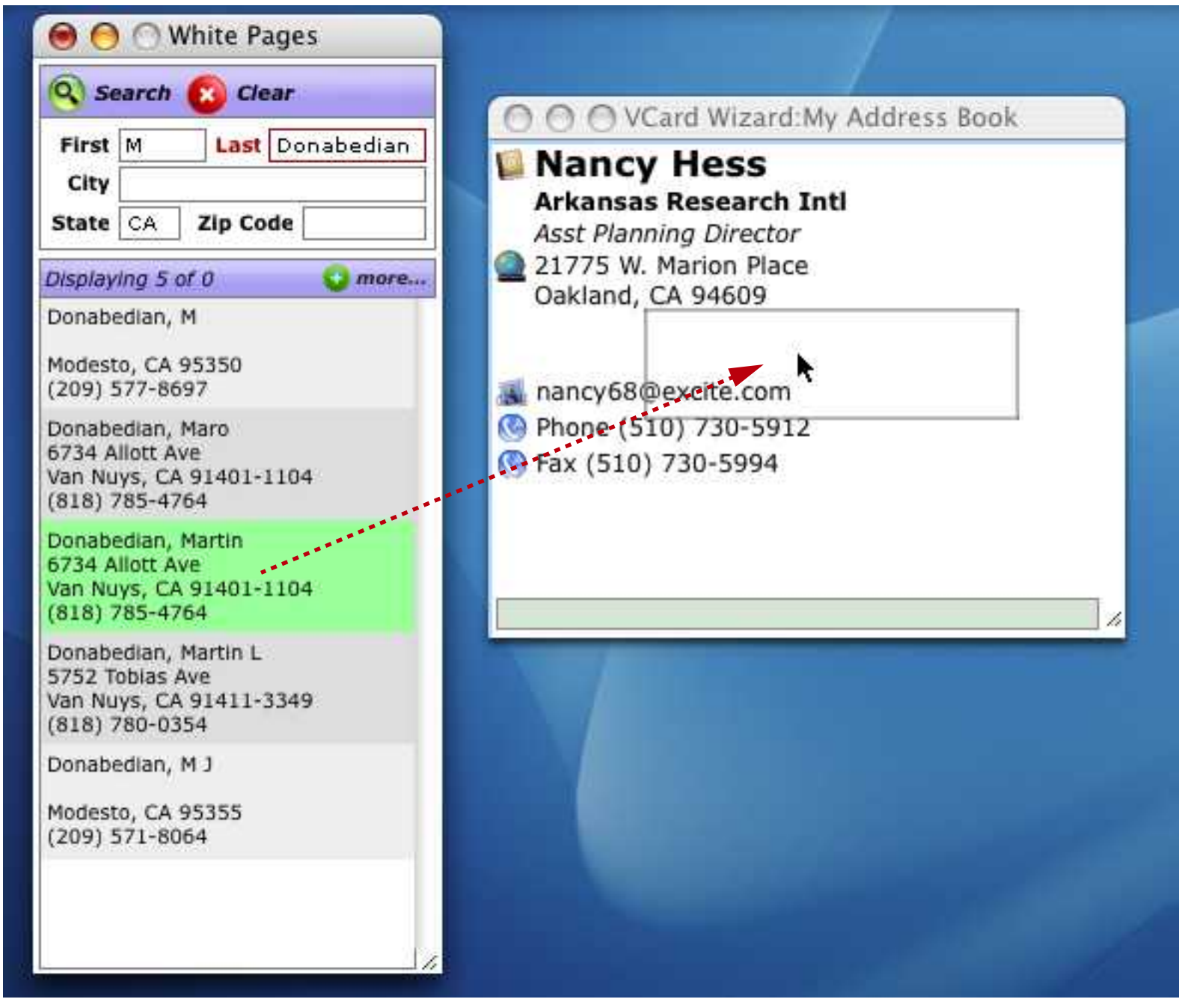
Drag the address to anywhere on the VCard Wizard.



When you release the mouse the contact will be added to the end of the database.



You can also drag from other databases that support generic fields. When you drag from the **White Pages** wizard to the **VCard Wizard**, the address is added to the database that is currently linked to the **VCard Wizard** (in this case My Address Book).



Displaying a Map

To display a map of the current address, click on the globe icon. The map will appear in whatever browser you have set up as the default browser.



Sending an Email

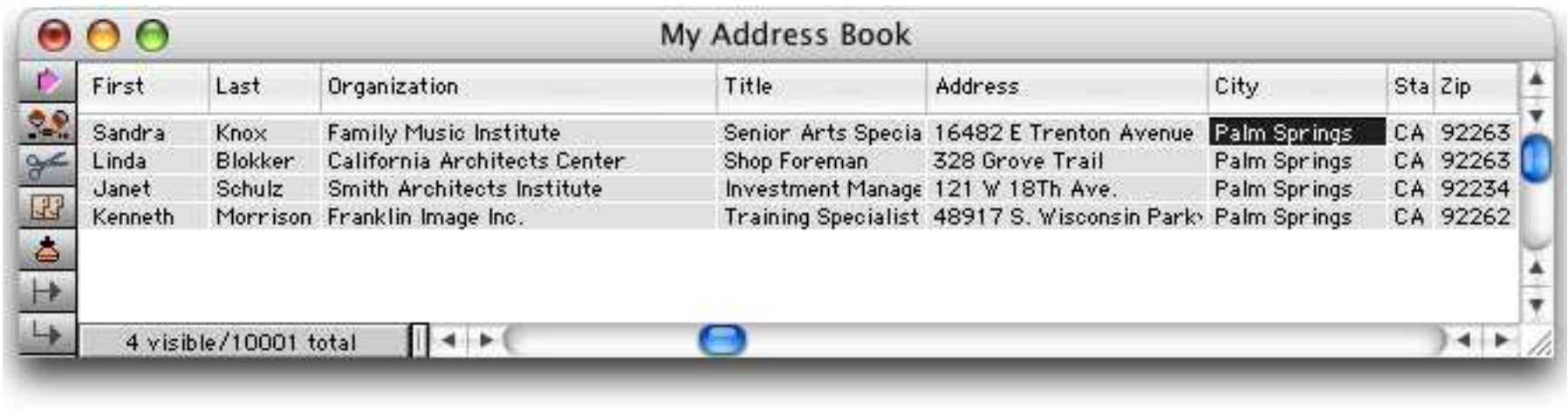
To send an e-mail, click on the postage stamp icon. The wizard will open the default e-mail program on your computer and create a new e-mail message to this person.

Dialing the Phone

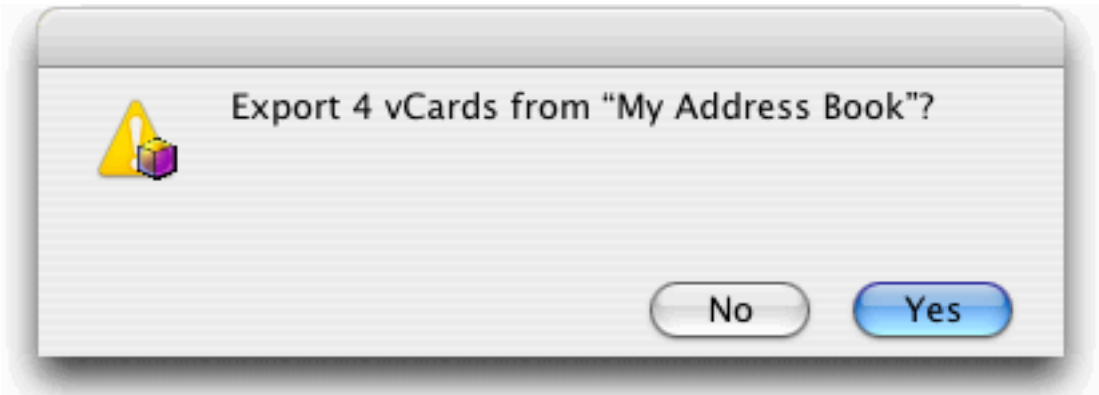
To dial the phone, click on the telephone icon. Before you do, be sure you have set up the **Channels** wizard (see “Channels” on page 54) to specify the dialing method you want to use (touch tones, modem, Vonage VOIP, etc.) and your local dialing information (area code, calling prefix and suffix, etc.).

Exporting a Batch of VCards

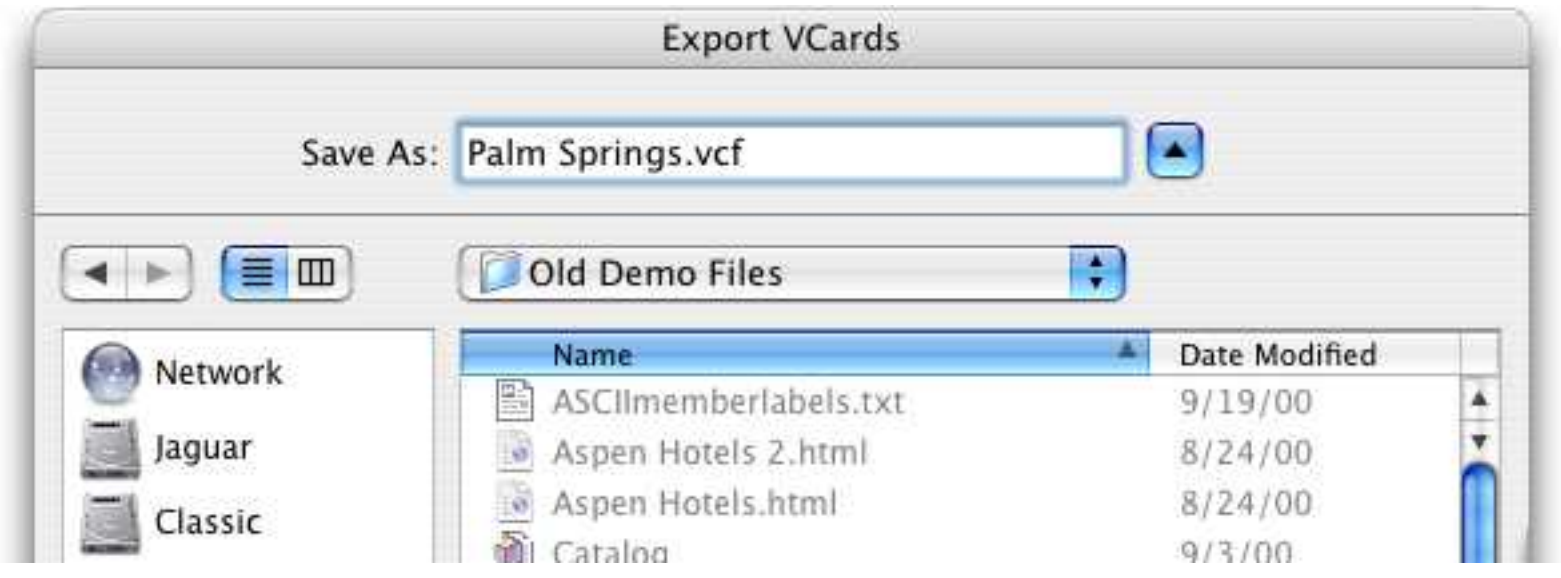
In addition to transferring records one at a time (by dragging them) you can also export a batch of vCards at once in a file. Start by selecting the records you want to export, in this case all records in Palm Springs, CA.



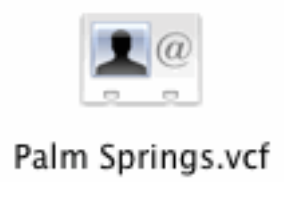
Open the **VCard Wizard** and select **Export VCards…** from the VCard menu. The wizard will ask you to confirm that you want to export multiple vCards.



Next the wizard will ask you for the name of the file you want to export. We recommend that you use the .vcf suffix for vCard files.

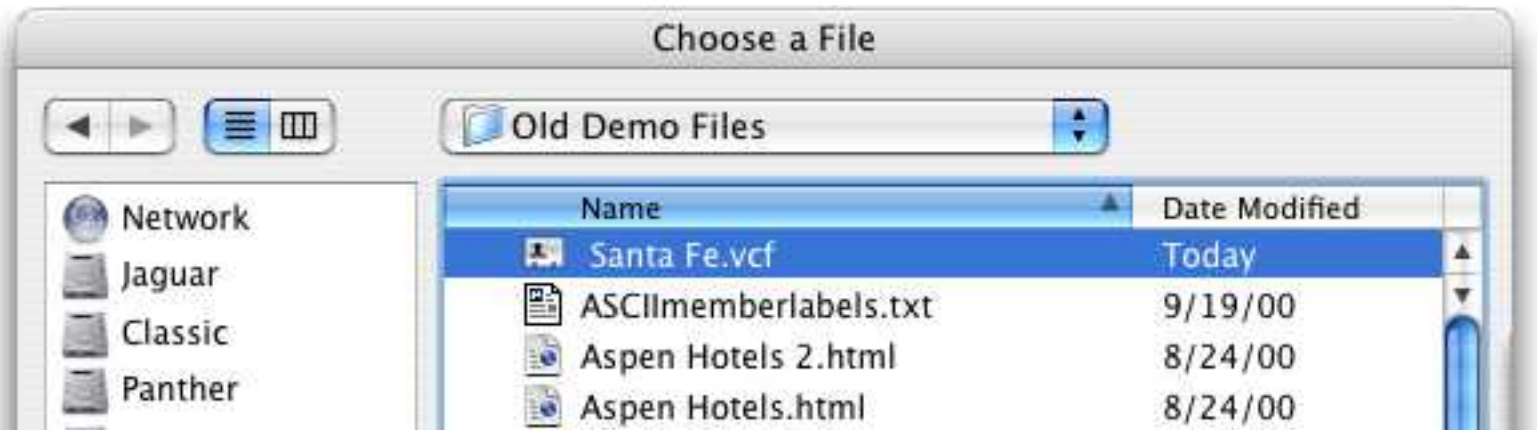


The resulting file can be imported into other applications that support vCard import (for example Apple’s Address Book).



Importing a Batch of VCards

To import a file containing a batch of VCards choose **Import VCards** from the **VCards** menu. The wizard will ask you to select the file you want to import.



Press the **Choose** button and the vCards will be imported to the end of the current database.

Chapter 6: Data Types



In Panorama, all data is not the same. Just as Eskimos distinguish between 16 types of snow, Panorama distinguishes between five types of data—**text**, **numeric**, **date**, **choices**, and **pictures**. To get the most out of a database, Panorama needs to know what type of data you intend to store in each field. This lets Panorama store the data efficiently and check for data entry errors. It also tells Panorama how to compare different values (numbers, text, and dates are all compared differently) which is important for sorting and selecting data. The data type also tells Panorama how to format some kinds of data (numbers and dates).

As mentioned above, Panorama databases can contain five different types of data. When you create the database, you specify what type of data will be stored in each field.

Data Type	Uses	Examples
Text	Names, Addresses, Descriptions, etc.	John, 234 Peach Avenue
Numeric	Prices, Quantities, etc.	4, 78.23, 4.9e-2
Date	Dates.	9/18/2002
Choices	Multiple Choice Options	Yes/No, Gold/Silver/Bronze
Picture (Obsolete)	Photographs, Drawings	n/a

The **text** data type is used for storing ordinary text—names, addresses, descriptions, etc. Panorama cannot perform mathematical calculations (add, subtract, etc.) on data that is stored as text.

The **numeric** data type is used for storing numbers—prices, quantities, etc. Use the numeric data type for any field you want to use in a calculation. The numeric data type has several variations that are discussed later in this chapter.

It's not always necessary to store numbers in numeric fields. For example, zip codes and phone numbers are usually stored in text fields, not numeric fields. This allows the use of nine digit zip codes (for example 92867-3482) and foreign postal codes and phone numbers. In general, use a numeric field if you want to perform numeric calculations (addition, multiplication, etc.) and/or if you want to select or sort the information in numeric order (1, 2, 3, … 10, 20, 30, … etc.)

The **date** data type is self explanatory—it is used for storing dates (for instance March 1, 1994). Panorama understands the properties of dates—it knows that May 1st follows April 30th and that there are six days between May 28th and June 3rd. Dates are discussed in more detail later in this chapter. Panorama can handle dates from 100 A.D. to well past the year 20,000 A.D.

The **choices** data type is used for storing data that has only a few possible values—for instance yes/no, gold/silver/bronze, or coach/first class. The choices data type saves space by storing a special code instead of the entire text. The procedure for setting up a field using the choices data type is described at the end of this chapter.

The **picture** data type is used for storing-you guessed it-pictures! This data type allows you to paste graphics from other Macintosh programs into your Panorama database. However, in general we do not recommend storing pictures within the database using the picture data type. Instead, you should store images in separate files and display them using Panorama's **Flash Art** feature (See "Flash Art™" on page 748).

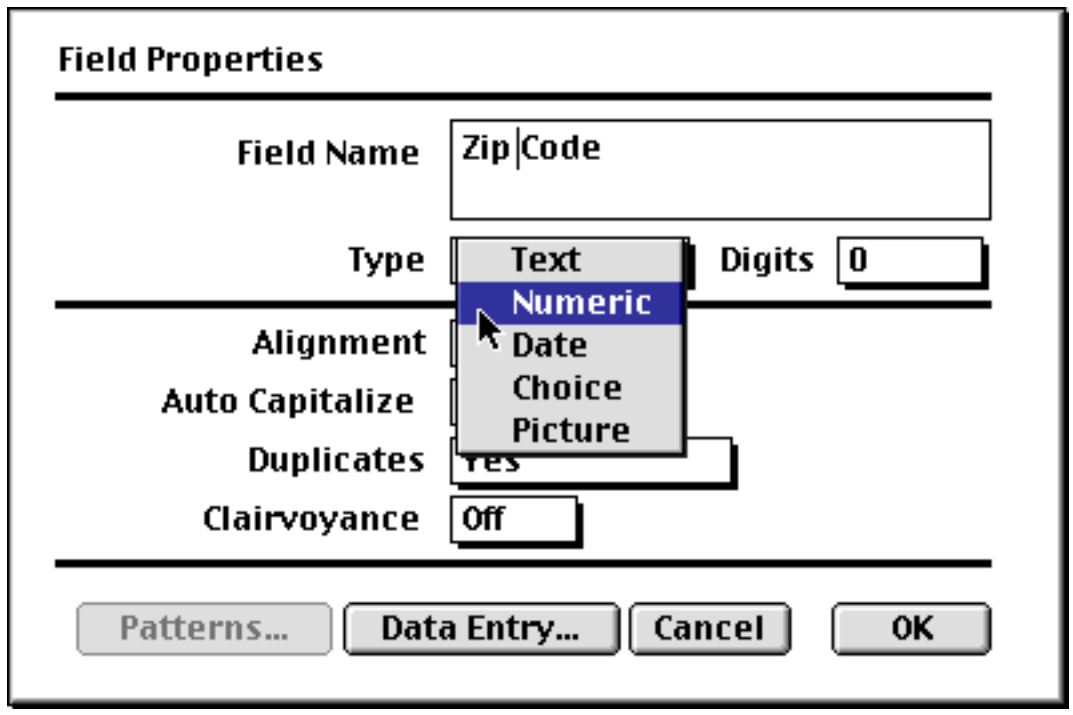
Data Types and Memory Usage

When we started writing Panorama in 1986, a typical personal computer had about 1 megabyte of memory. Because Panorama loads the entire database into RAM, it was critical to store the data as efficiently as possible. To do this we created a variety of data types, allowing you to pick the data type that was most efficient for each field (for example, there are 6 different numeric data types).

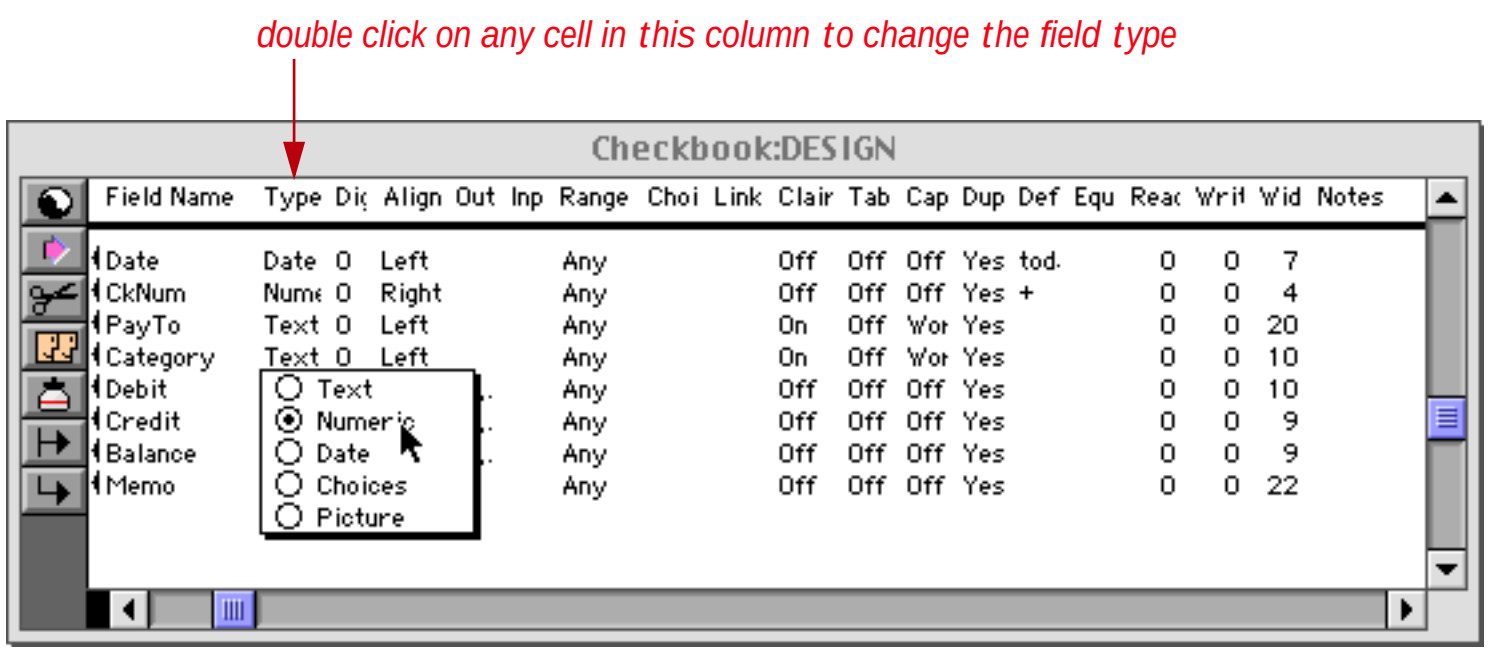
Nowadays a typical computer has 32 megabytes, 64 megabytes, or even more RAM. For most applications it is no longer necessary to be hypervigilant about choosing the most efficient data type. For example, for numbers, it's usually fine to simply use the most flexible **floating point** option, even though this consumes a little bit more memory. It's also rarely necessary to use the **choice** data type—you can simply use the **text** data type instead. Throughout the rest of this chapter you'll find a lot of discussion about picking the most efficient data type for each field. If you are working with small to moderate size databases (from 10 to 20,000 records) you probably don't need to worry about picking the most efficient data types. You can simply stick with three basic data types—text, floating point numbers, and dates. That being said, you can probably skip over most of the material in the rest of this chapter!

Setting Up a Field's Data Type

The data type of a field can be specified with either the Setup Menu or the Design Sheet. The simplest method is using the **Field Properties** command in the Setup Menu. The **Field Properties** dialog box contains a pop-up menu for selecting the data type. Tip: You can also open the Field Properties dialog box by double clicking on the name of the field on the data sheet.

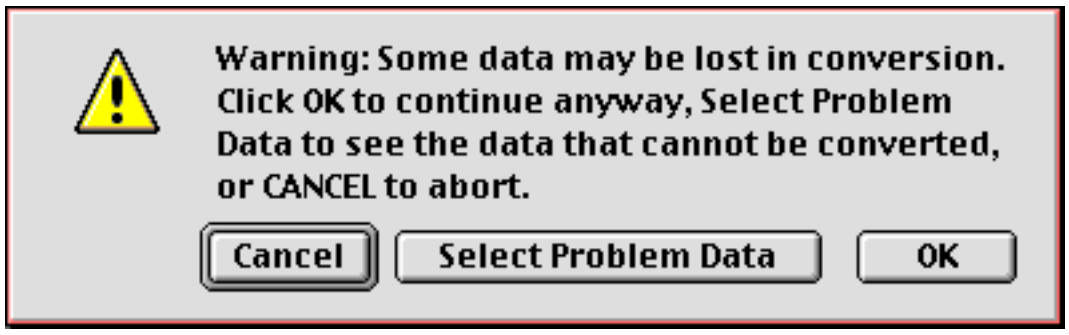


An alternate way to specify the data type is using the Design Sheet. The second column of the design sheet contains the data type of each field in the database. Using the design sheet allows you to quickly modify many fields at a time. See “The Design Sheet” on page 252 for more information on opening and using the design sheet.



Data Type Conversion Problems

When you change the data type of an existing field it is possible that you may lose some of the data in that field. For example, if you convert a field from text to numeric, any letters or punctuation will be lost in the conversion. Panorama will warn you if this situation occurs.



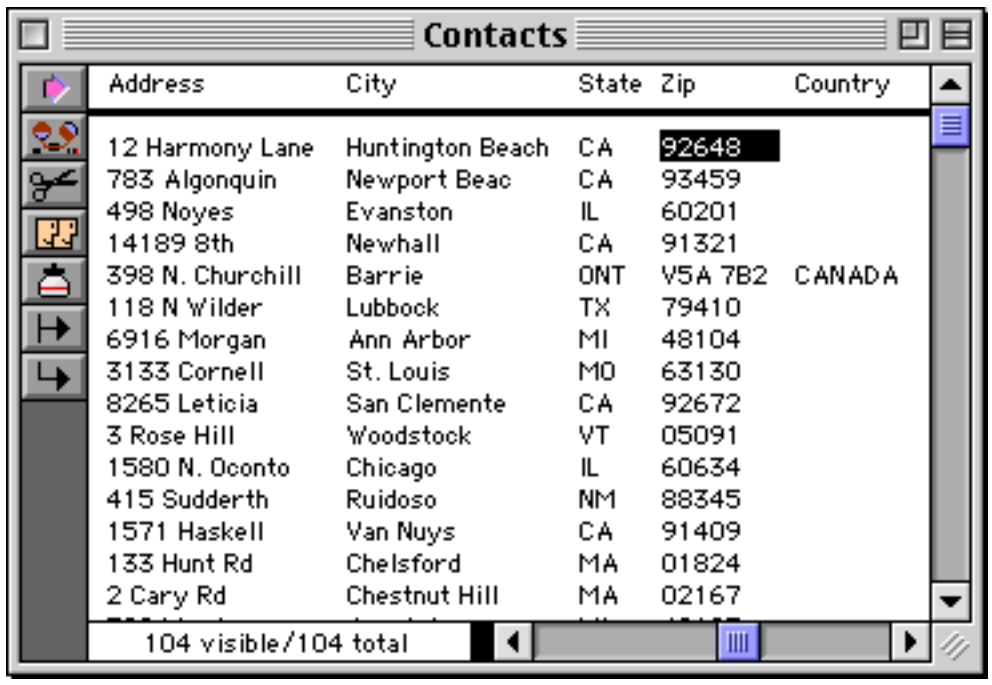
The alert gives you three options:

OK — Pressing this button tells Panorama to complete the conversion. Any data that cannot be converted will be removed. For example, if you are converting a field from text to numeric any non-numeric data will be lost in the conversion.

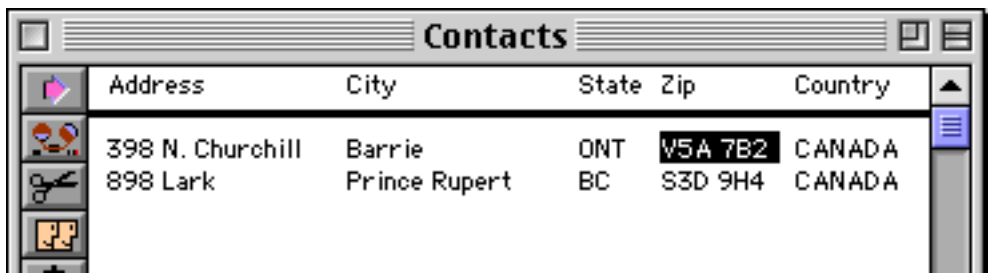
Cancel — Pressing this button cancels the conversion

Select Problem Data Pressing this button tells Panorama to cancel the conversion and show you the data that is causing the problem. This option lets you look at the data that is causing the problem and decide what to do next. If necessary, you can make manual adjustments to the data, or you may decide that you don’t want to change the data type after all.

For example, suppose you wanted to change the Zip code field in the database below to numeric.

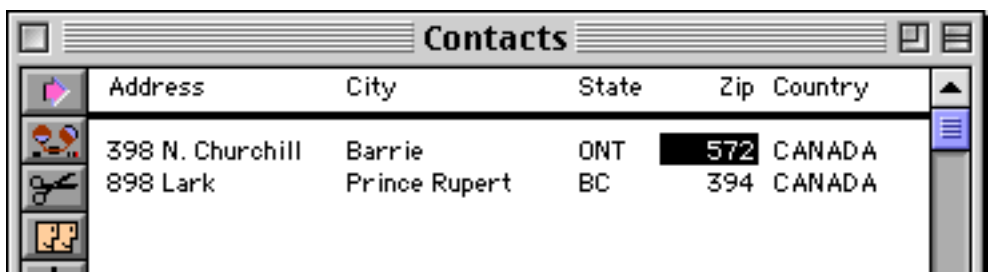


When you attempted to make the conversion, Panorama would display the conversion warning alert. Press **Select Problem Data** to see what is causing the problem.



Aha! The problem is the Canadian postal codes, which have letters instead of numbers. At this point you would probably want to rethink the idea of converting the Zip Code field to numeric.

If you did decide to go ahead with the conversion, Panorama would strip the letters from the Canadian postal codes.



When you are done looking at the problem data, choose the **Select All** command (Search Menu) to make all the data visible again. (See "Finding vs. Selecting" on page 367 for more information on selecting and the **Select All** command.)

Numeric Data

Numeric data can be stored in either **fixed point** or **floating point** format. If you choose fixed point you have a choice of 0, 1, 2, 3, or 4 digits after the decimal point.

Number of Digits After Decimal Point	Example	Largest Value	Smallest Value	Typical Uses
0	93842	2,100,000,000	1	Quantities, Part Numbers
1	73.1	210,000,000	0.1	Rarely Used
2	253.22	21,000,000	0.01	Money (Dollars, Pounds, etc.)
3	0.447	2,100,000	0.001	Rarely Used
4	929.1123	210,000	0.0001	Rarely Used
Float	1.46e-12	$1.7 \cdot 10^{308}$	$2.3 \cdot 10^{-308}$	Scientific Data

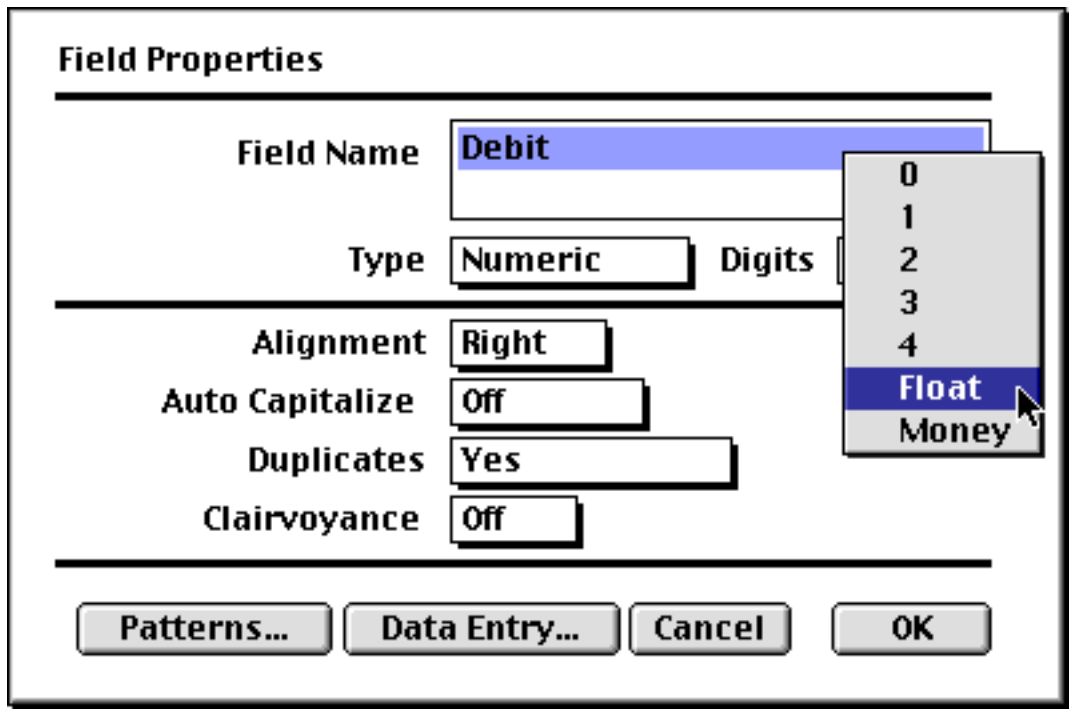
You may wonder why there are so many choices for storing numeric data. After all, a number is a number—right? Not quite. By choosing different numeric storage formats you are making a trade-off between space, speed, accuracy, and range.

Storing numbers using floating point gives you the most accuracy and numeric range. Floating point allows you to store extremely large or small values with up to 16 digits of accuracy. If you are in doubt, go ahead and pick floating point format.

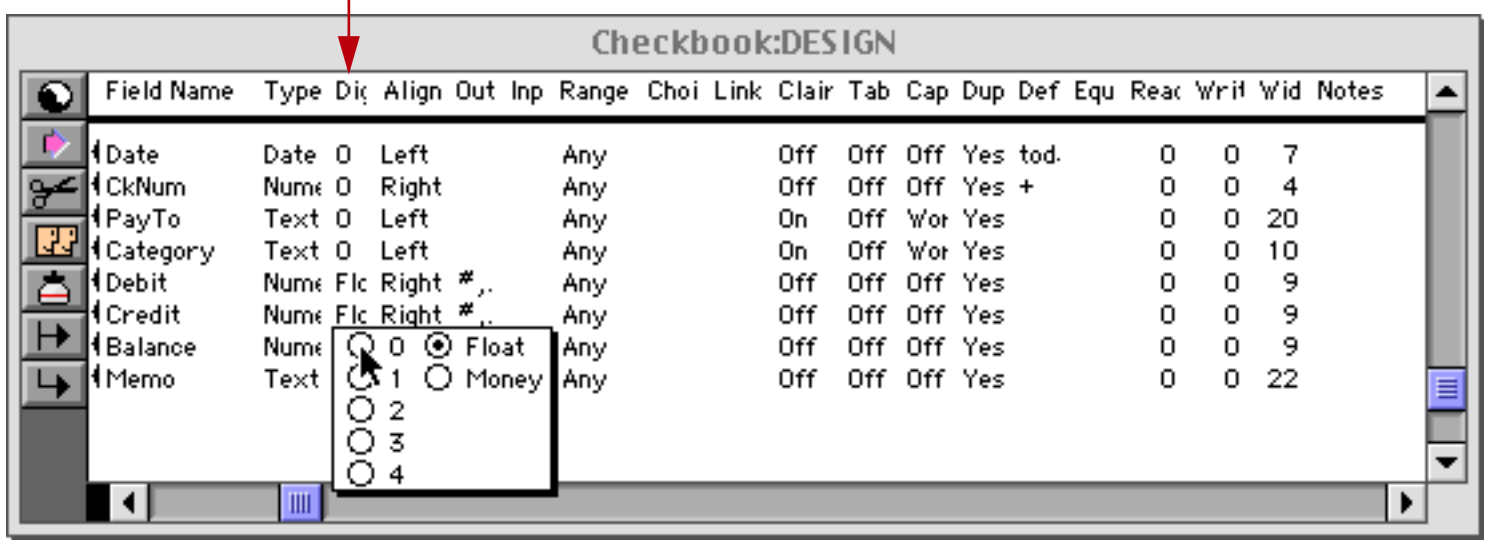
Fixed point storage is more limited. The accuracy is only about 9 digits. The largest number that can be stored is about 2 billion ($2 \cdot 10^{9}$) while the smallest fixed point number is 0.0001 (10^{-4}). Trying to store larger or smaller values using fixed point storage will result in errors.

On the other hand the space required for fixed point storage is up to 8 times smaller than floating point for the same number, and Panorama can perform fixed point arithmetic much faster than floating point. You should use fixed point numeric storage whenever possible. Check the table above to see if the numbers you will be using fit in one of the fixed point numeric ranges.

You can set the number of digits via a pop-up menu in the **Field Properties** dialog box.



You can also set the number of digits using the Digits column in the design sheet.



Money

Usually the best way to store monetary values is using either 2-digit fixed point or Panorama's special Money format. The money format is the same as 2-digit fixed point but automatically enters the decimal point for you during data entry. This table below shows how Panorama interprets data you enter into a money field.

When you enter…	it becomes
87204	872.04
3267	32.67
14	0.14
2	0.02
42.	42.00
15.4	15.40
156.78	156.78

Both the 2-digit and money formats allow you to store monetary values up to 21 million dollars, pounds, francs, etc. (If your business deals with values greater than 21 million you should use floating point numeric storage.)

Numeric Output Patterns

Output patterns allow you to control the way a number is displayed. In the design sheet, output patterns can be used to control how numbers are displayed in the data sheet. Output patterns can also be used in a formula (See "Converting Between Numbers and Strings" on page 1227).

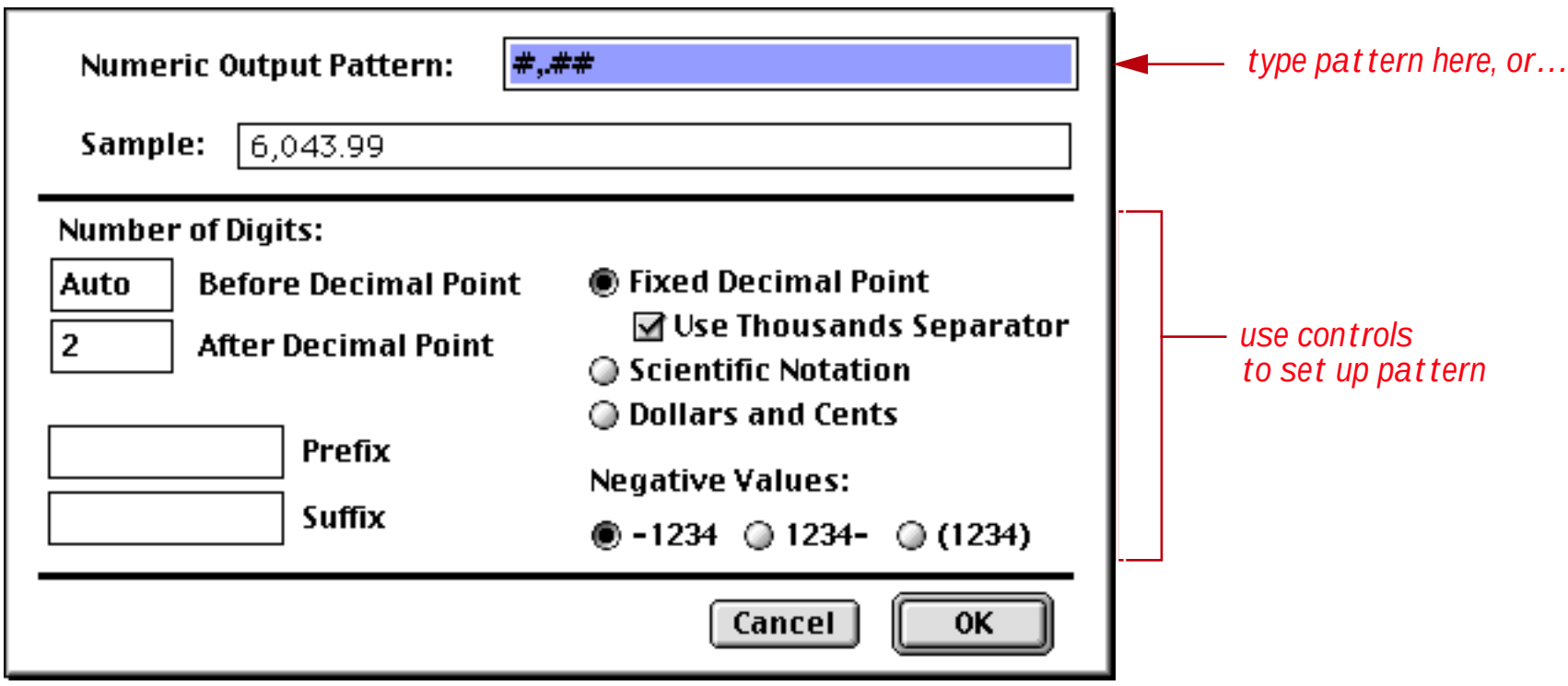
Below are some ways the same number may be displayed using different output patterns. Remember, the way a number is displayed does not change its internal value. All the numbers listed below have the value 2654.

```
2654
2,654
$2,654.00
002654
2.654e+3
26-54
Two thousand six hundred fifty four
```

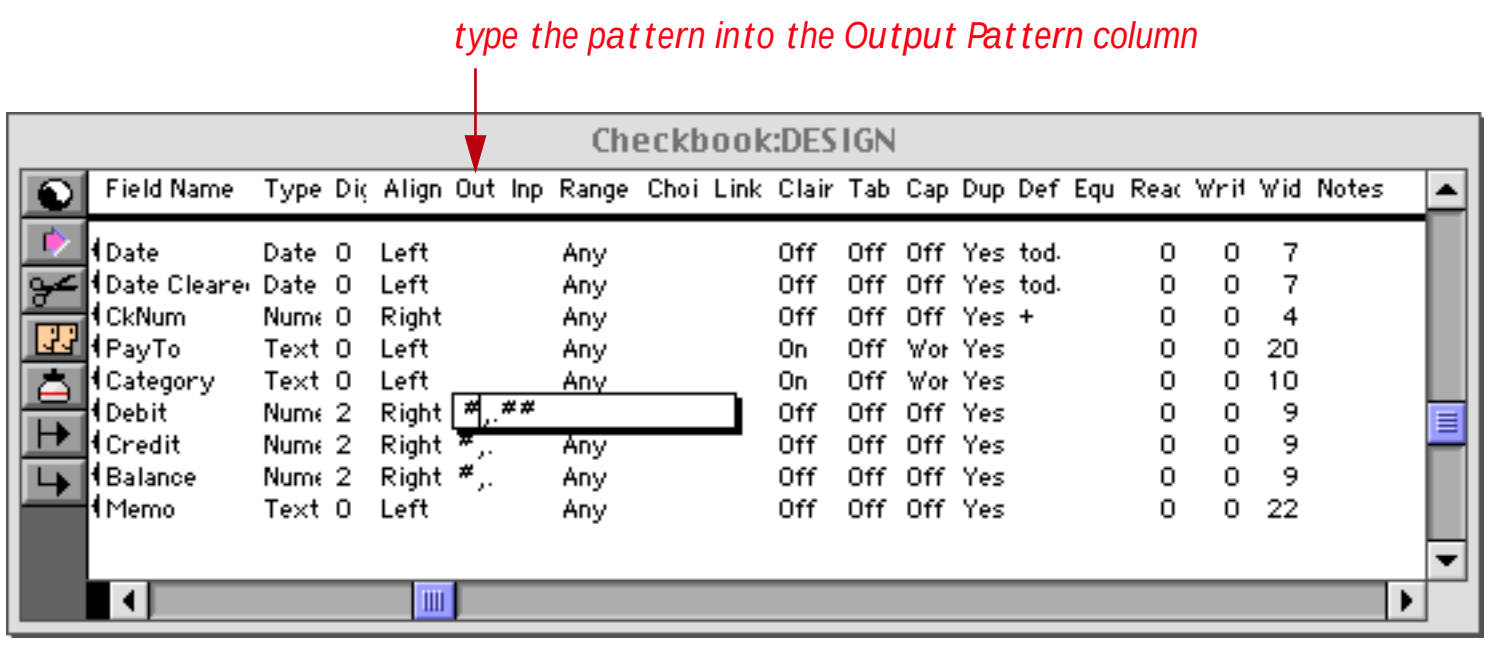
Numeric output patterns consist of a string of characters containing one or more # symbols. The # symbol tells Panorama how and where to print the number.

The overall output pattern for a field can be set using the **Pattern** button in the **Field Properties** dialog box or the Output Pattern column in the design sheet.

When you press the **Pattern** button a new dialog appears. You can either type the pattern into the box at the top or you can select the display options you want and let Panorama create the output pattern for you.



If you are using the design sheet to set up the pattern, simply type the pattern into the Output Pattern column. See "The Design Sheet" on page 252 for more information on opening and using the design sheet.



Fixed Decimal Point Patterns

The table below shows how the output pattern can be used to display numbers with a specified number of digits after the decimal point. These output patterns force a fixed point display even if you are displaying floating point numbers. They also allow you to override the natural display of fixed point numbers. For example, a money field can be set up to display only dollars, while still keeping track of cents for calculation purposes.

Number	Pattern	Display
1234.56		1234.56
1234.56	#	1235
1234.56	#.#	1234.6
1234.56	#.##	1234.56
1234.56	#.####	1234.5600

Notice that if the number of # symbols after the decimal point is less than the number of digits in the number, Panorama will round the number rather than truncating it.

Numbers with Commas, Punctuation, and Measurement Units

If a comma is added to the pattern, the number will be printed with a comma every third digit. Other characters can also be added to the beginning or end and will be displayed unchanged. For example, you can add a currency symbol or measurement unit to the pattern, as shown below:

Number	Pattern	Display
1234.56	#,.##	1,234.56
1234.56	$#,.##	$1,234.56
1234.56	#,.## kg	1,234.56 kg

Scientific Notation

If an **E** or **e** is added at the end of the output pattern, the number will be displayed using scientific notation. Any number may be displayed in scientific notation, including fixed point numbers.

Number	Pattern	Display
1234.56	#e	1e+3
1234.56	#.#E	1.2e+3
1234.56	#.##e	1.23e+3
1234.56	#.###E	1.235e+3
1234.56	#.####E	1.2346e+3
1234.56	#.#####E	1.23456e+3
1234.56	#.######E	1.234560e+3
1234.56	#.#E kg	1.2e+3 kg

Special Patterns for Negative Numbers

Negative numbers are usually displayed with a minus sign in the front of the number. This can be changed to a trailing minus sign or to enclosing parentheses.

Number	Pattern	Display
-1234.56	#.##	-1234.56
1234.56	#.##	1234.56
-1234.56	#.##-	1234.56-
1234.56	#.##-	1234.56
-1234.56	(#.##)	(1234.56)
1234.56	(#.##)	1234.56

Leading Zeros

You can use an output pattern to force Panorama to display leading zeros. To do this, put several # symbols in a row without a decimal point.

Number	Pattern	Display
123	#####	00123
1234	#####	01234
12345	#####	12345

Tip: If you are storing US Zip codes in a numeric field, use ##### as the output pattern. This pattern makes sure that all 5 zip code digits are displayed, even if the first digit is zero.

If your database also contains Canadian postal codes, the zip codes must be stored in a text field. In that case no output pattern is necessary.

Numbers with Multiple Components

An output pattern can be used to split a number into multiple components. To split up a number, spread the # characters through the output pattern, one # symbol for each digit you want to print. The examples below show how this feature can be used for social security numbers, phone numbers, and combination locks.

Number	Pattern	Display
219304349	###-##-####	219-20-4349
5293672	###-####	529-3672
241018	L## R## L##	L24 R10 L18

Using output patterns this way can save a lot of memory. For instance, storing the combination L24 R10 L18 as text requires 11 bytes per combination. Storing the same combination as a number requires only 3 bytes per combination. If your database contains 10,000 combinations this represents a savings of 80k of memory. (Sure, in the new millennium 80k isn't much! But if your database contains one million records using an output pattern would save 8 megabytes.)

Phone Numbers

We recommend storing phone numbers in text fields. However, using multiple component output patterns it is possible to store phone numbers as numbers. For local phone numbers you can use the output pattern ###-#### and store the numbers as fixed point with zero digits. This results in a savings of 5 bytes per phone number (3 bytes vs. 8 bytes). On the other hand, who ever heard of a 7 digit phone number in this day of area code splits every other week?

A long distance phone number requires more accuracy than is available in a fixed point number, so you'll have to use the floating point data type. The output pattern is `(###) ###-####`. This results in a savings of 6 bytes per phone number (8 bytes vs. 14 bytes).

If your phone numbers are stored as numbers you should only enter the digits. For example to enter the number (800) 432-4567 type `8004324567`.

We recommend storing phone numbers in a text field. Although this takes more memory, it is more flexible, allowing you to add extensions or other notes (for example 329-9583 ext 241). If you do use a text field, you can use an input pattern to enter the `(`, `)`, and `-` characters for you. See "Input Patterns" on page 327 for more information on this technique.

Plural Suffixes

If a pattern contains measurement units you may want to properly pluralize the units depending on the value being displayed. Use the `~` symbol to do this.

Number	Pattern	Display
1	`# mile~`	1 mile
5	`# mile~`	5 miles

Displaying Numbers as Words

If you wish, numbers can be displayed as words instead of digits. To do this, use the **§** symbol instead of the # symbol. Only one **§** symbol should be used. To make the **§** symbol on a Macintosh, press **Option-6**. On the PC, press **Alt-0167**. Only the integer part of the number will be displayed—any fractional part will be ignored.

If you are displaying money, you'll probably want to display the fractional part (cents) as well as the integer part. You can do this with the **¢** (cents) symbol. On the Macintosh, press **Option-4** to create the **¢** symbol. On the PC, press **Alt-0162**. Use one **¢** for each digit you want display (usually 2).

Number	Pattern	Display
312	**§**	Three hundred twelve
42.29	**§** dollar~ and **¢¢**/100	Forty two dollars and 29/100

Dates

Panorama has a special data type for storing dates. When you use the date type to store your dates, Panorama can sort your dates in the correct order, check your dates for validity as they are entered, and calculate the number of days between two dates. Dates are quite compact; almost any date in the 20th or 21st century will take only two bytes of storage.

Entering Dates

Panorama is very flexible about how you type dates. We call this feature "smart dates." You can enter dates numerically (for instance **04/09/02** or **4/9/2**) or you can spell out the date (for instance **April 9th, 1997** or **Apr 9 97**). You can use any character as a separator between numeric dates, for example **4-9-01** or even **4.9.01**.

To enter today's date, simply type **today**. You can also enter **yesterday** or **tomorrow**. Panorama will automatically convert these entries to the correct month, day and year.

If the date is in the current week, you can simply type in the name of the day, for example **saturday** or **tue**. To specify a day in the previous or upcoming weeks add the words last or next, for example **next tuesday** or **last saturday**.

When a date is edited, Panorama normally displays the date in the format `mm/dd/yy`. However, if you have set up an output pattern that Panorama understands for data entry, it will use that pattern instead. Patterns that can be used for data entry include `Month dd, yyyy`, `Mon dd yy`, and `mm/dd/yyyy`.

If you are using an international system and you enter the date as numbers you must use the format `dd/mm/yy`. Panorama does not understand the format 7-Aug-1998. (However you may use any delimiter character you want, for example 7/8/98 or 7-8-98 or even 7.8.98.)

Default Year and Century

When you enter a date, you can leave the year off and let Panorama figure it out for you. Panorama will automatically round the date to the nearest year. For instance, if today's date is 3/9/02 and you enter the date **4/1** then Panorama will assume you mean 4/1/02. But if you enter **12/1** (or **December 1**) Panorama will assume you mean 12/1/01, not 12/1/02, because 12/1/01 is closer to 3/9/02 than 12/1/02 is.

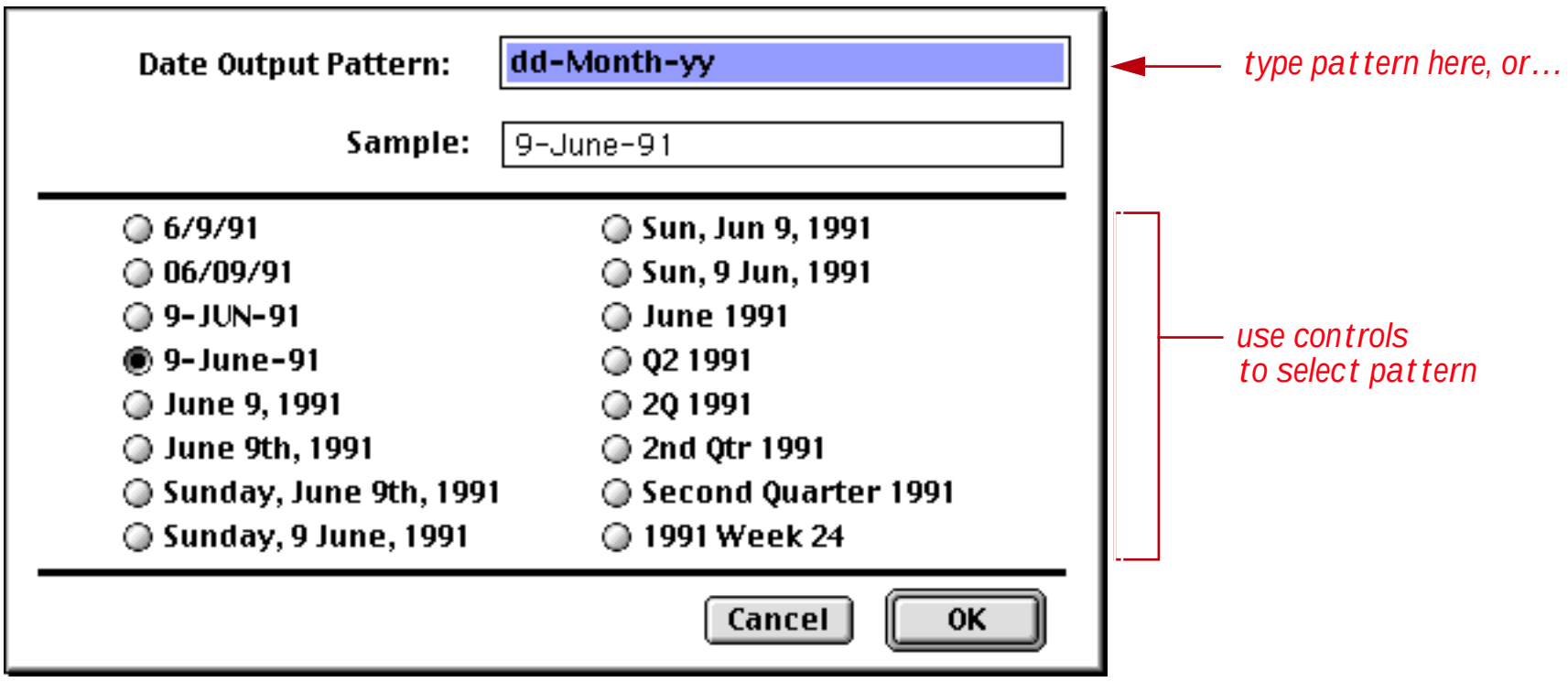
Panorama also rounds dates to the current century. If the current year is 2002 (or even 1992) and you enter the date **7/2/23** Panorama will assume you mean 7/2/2023. If you want to enter a date more than 50 years from the current date you must enter the full date, for example **7/2/1923**

Date Output Patterns

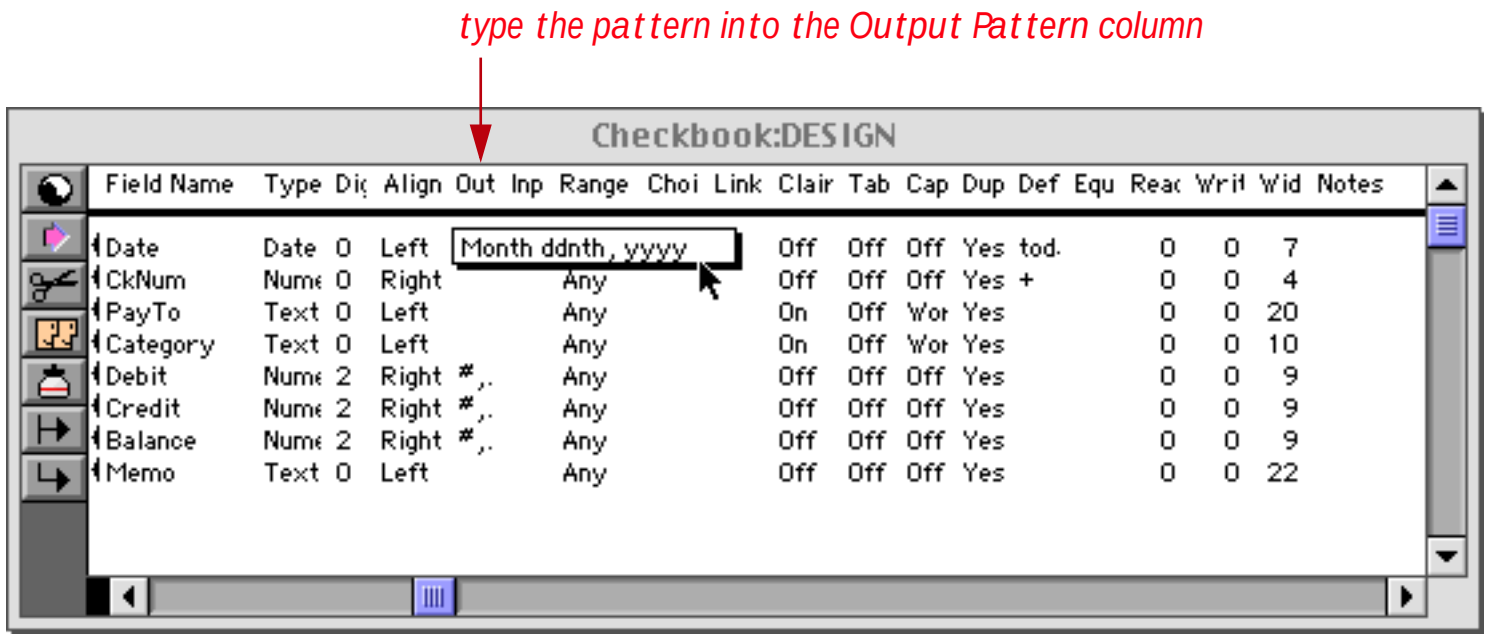
Output patterns allow you to control the format Panorama uses to display dates. A date output pattern consists of a number of individual components (month, day, year, etc.) that are strung together. For example, the pattern `mm/dd/yy` contains three components and will display in the format 3/11/04.

The overall output pattern for a field can be set using the **Pattern** button in the **Field Properties** dialog box or using the Output Pattern column in the design sheet.

When you press the **Pattern**, button a new dialog appears. You can either type the pattern into the box at the top or you can select the pattern you want using the radio buttons.



If you are using the design sheet to set up the pattern, simply type the pattern into the Output Pattern column. See "The Design Sheet" on page 252 for more information on opening and using the design sheet.



Date Pattern Components

There are 15 different basic components that can be used as part of a date pattern. A date pattern is built up by combining these components together with punctuation to build a complete pattern. (See "Common Date Output Patterns" on page 297 for examples of complete, ready-to-use date patterns.)

Component	Example	Description
yy	02	Year (within century)
yyyy	2002	Year (including century)
qq	2	Quarter (numeric)
qtr	2nd	Quarter (abbreviated)
quarter	second	Quarter (spelled out)
mm	9	Month (numeric)
MM	09	Month (with leading zero)
mon	sep	Month (abbreviated)
month	september	Month (spelled out)
ww	46	Week (within year)
dd	5	Day (numeric)
DD	05	Day (with leading zero)
day	tue	Day Of Week (abbreviated)
dayofweek	tuesday	Day Of Week (spelled out)
dow	3	Day Of Week (0[sun]-6[sat])

Some of these components (qtr, quarter, mon, month, day, and dayofweek) can be either upper or lower case. The table below shows how a component can be displayed in all lower case, initial caps, or all upper case.

Pattern	Display
month	september
Month	September
MONTH	SEPTEMBER
dayofweek	friday
DayOfWeek	Friday
DAYOFWEEK	FRIDAY

Common Date Output Patterns

A date output pattern is assembled from the basic components listed in the previous section along with any punctuation or text that is needed between the components. The table below lists several common date patterns.

Date	Pattern	Display
3/9/2002	mm/dd/yy	3/9/02
3/9/2002	MM/DD/YY	03/09/02
3/9/2002	mm-dd-yyyy	3-9-2002
3/9/2002	dd-MON-yy	9-MAR-02
3/9/2002	dd-Month-yy	9-March-02
3/9/2002	Month dd, yyyy	March 9, 2002
3/9/2002	Month ddnth, yyyy	March 9th, 2002
3/9/2002	DayOfWeek, Month ddnth, yyyy	Saturday, March 9th, 2002
3/9/2002	qqqyy	1q02
3/9/2002	Week ww of yyyy	Week 11 of 2002
5/23/2002	Quarter "Quarter" yyyy	Second Quarter 2002
7/11/2004	Qtr "Qtr" yyyy	3rd Qtr 2004
3/9/2002	"Day" dd, "Month" mm	Day 9, Month 3
3/1/2002	ddnth "day of" Month, yyyy	1st day of March, 2002
3/2/2002	ddnth "day of" Month, yyyy	2nd day of March, 2002
3/9/2002	ddnth "day of" Month, yyyy	9th day of March, 2002
3/1/1867	mmnth "month of" yyyy	3rd month of 1867
3/9/1978	wwnth week of yyyy	3rd week of 1978

If you need to include the words qtr, quarter, mon, month, or day in your date, you must quote them so that they are not treated as components, as shown in several of the examples in the table above. The quote key is just to the left of the **Return** key. Be sure to use regular quotes, not smart quotes (" not ").

As shown in several of the examples, you can add the suffix nth to the mm, ww, or dd components, Panorama automatically adds the correct suffix depending on the number displayed.

Choices

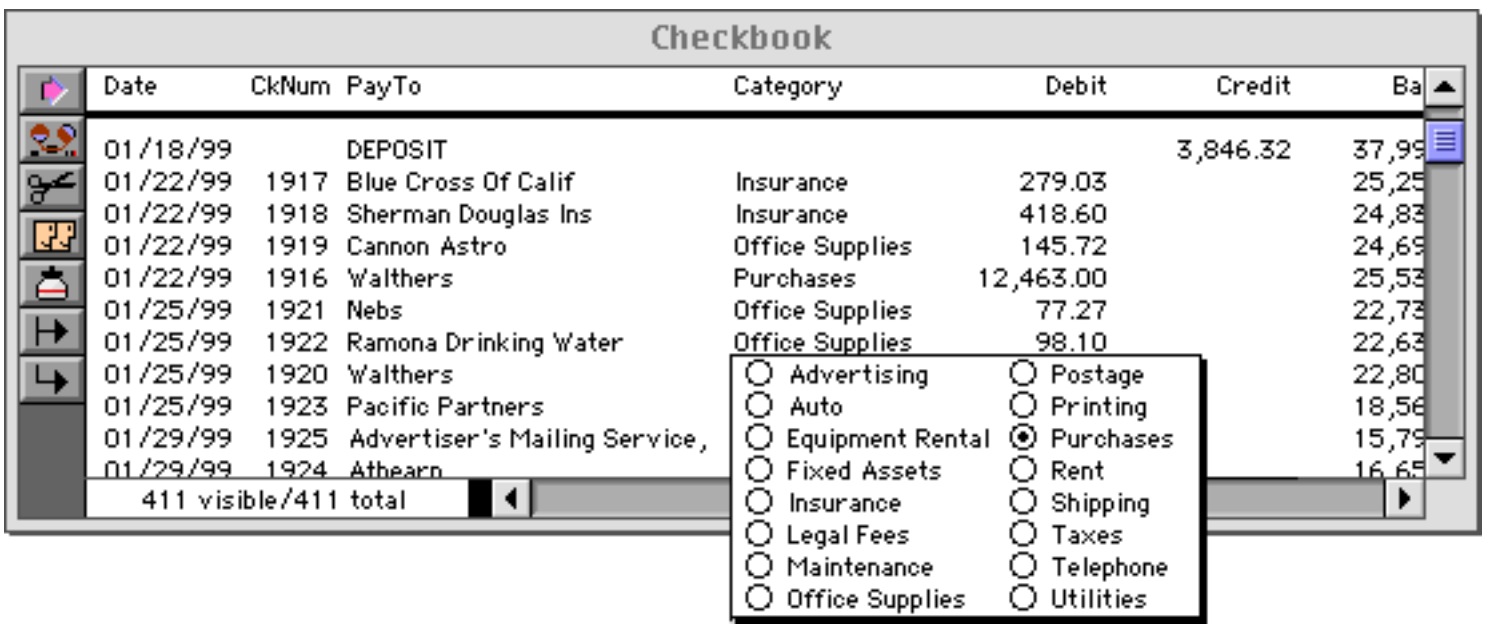
The choices data type is used for storing data that has only a few possible values. Some typical examples of this type of data are listed below. (Notice that in the case of choices that contain spaces, like US Mail and Eastman Kodak, the space is represented by an underscore, for example `US_Mail` and `Eastman_Kodak`.) Of course you can use the text data type to store this kind of information, but using the choices type can speed up data entry, reduce keying errors, and reduce storage space.

Type of Data	Typical Choices
Questionnaire	`Yes No`
Video Format	`VHS DVD`
Gasoline	`Regular Supreme Diesel`
Shipping Carrier	`UPS US_Mail FedEx Airborne DHL`
Film ASA Speeds	`25 64 100 200 400 800 1000`
Film Manufacturers	`Agfa-Gevaert Eastman_Kodak Fuji_Photo 3M`
Credit Terms	`Net_10 Net_30 COD Pre_Pay`
Operating System	`Windows MacOS Linux`
Medals	`Gold Silver Bronze`

The choices data type works by keeping a list of the possible choices for each field. In the database Panorama stores a choice number instead of storing the actual data—for example 1 for yes and 2 for no. The list of choices is used to decode the number whenever Panorama needs to display or use the data. The number itself is never visible.

Choice Data Entry (Choice Palette)

The choices data type is treated differently than other data types for data entry. Instead of entering the data from the keyboard, you pick a selection from a choice palette containing radio buttons for each choice.



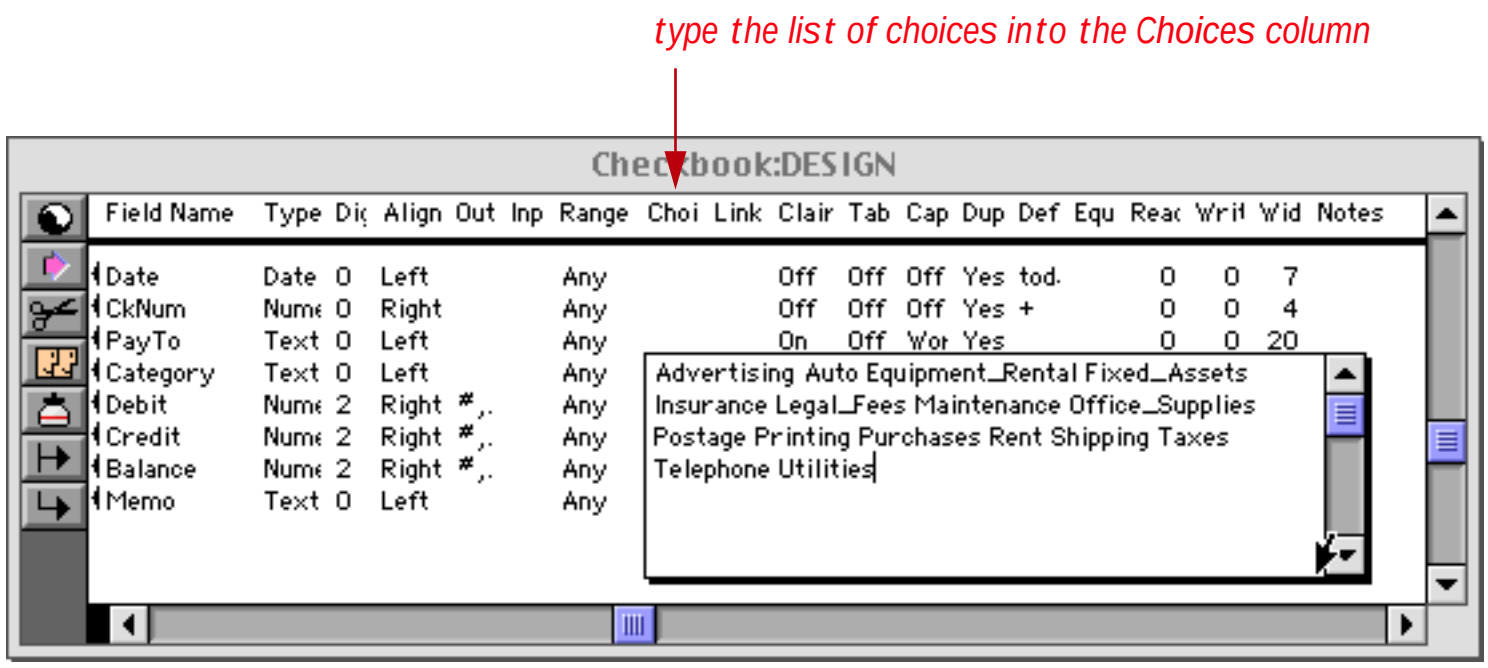
If the choice list allows for exceptions, there will also be a box allowing you to type in the exceptions. See “The Choice Palette” on page 353 for more information on using the choice palette.

In addition to picking from the list of radio buttons, you can also pick an item from the choice palette by typing in the first few letters of the item. For example, if the choice palette contains the buttons Gold, Silver and Bronze you can pick Gold by pressing the **G** key. If two or more buttons start with the same letter, keep typing until you reach a letter that is different.

If the choice list has many items, the choice palette may become too large. In that case you may want to turn off the choice palette and use normal data entry via the keyboard. To suppress the choice palette but continue using the choices data type, set the Number of Digits column in the design sheet to 1 instead of 0. The data will still be compressed via the choice list, but you will be able to enter your choice via the keyboard instead of from the choice palette.

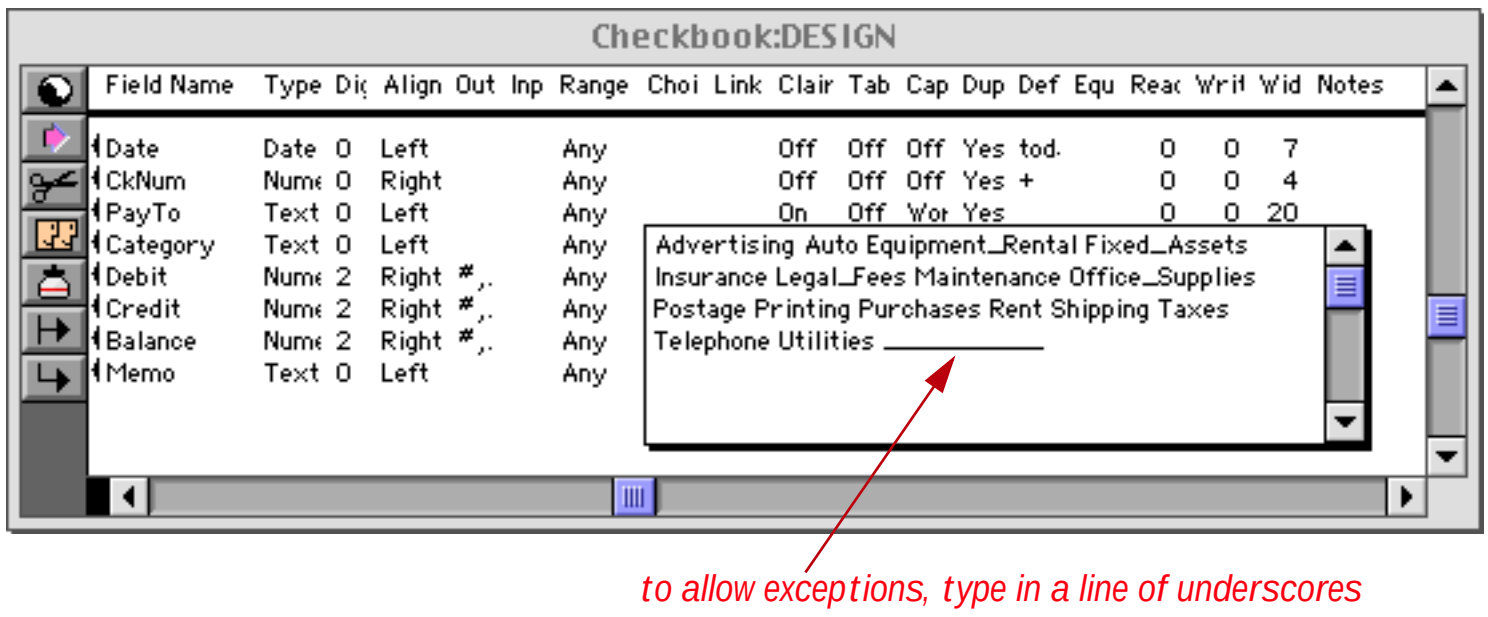
Creating the List of Choices

Panorama keeps the list of choices in the Choices column of the design sheet. (See "The Design Sheet" on page 252 if you are not familiar with the design sheet.) To create a list of choices, simply type the choices into the appropriate cell in the Choices column, separating each choice with a space. If a choice contains a space (US Mail), type an underscore instead (US_Mail). You can type an underscore by pressing **Shift-Underscore**. (The **Underscore** key is just to the right of the **0** key.)

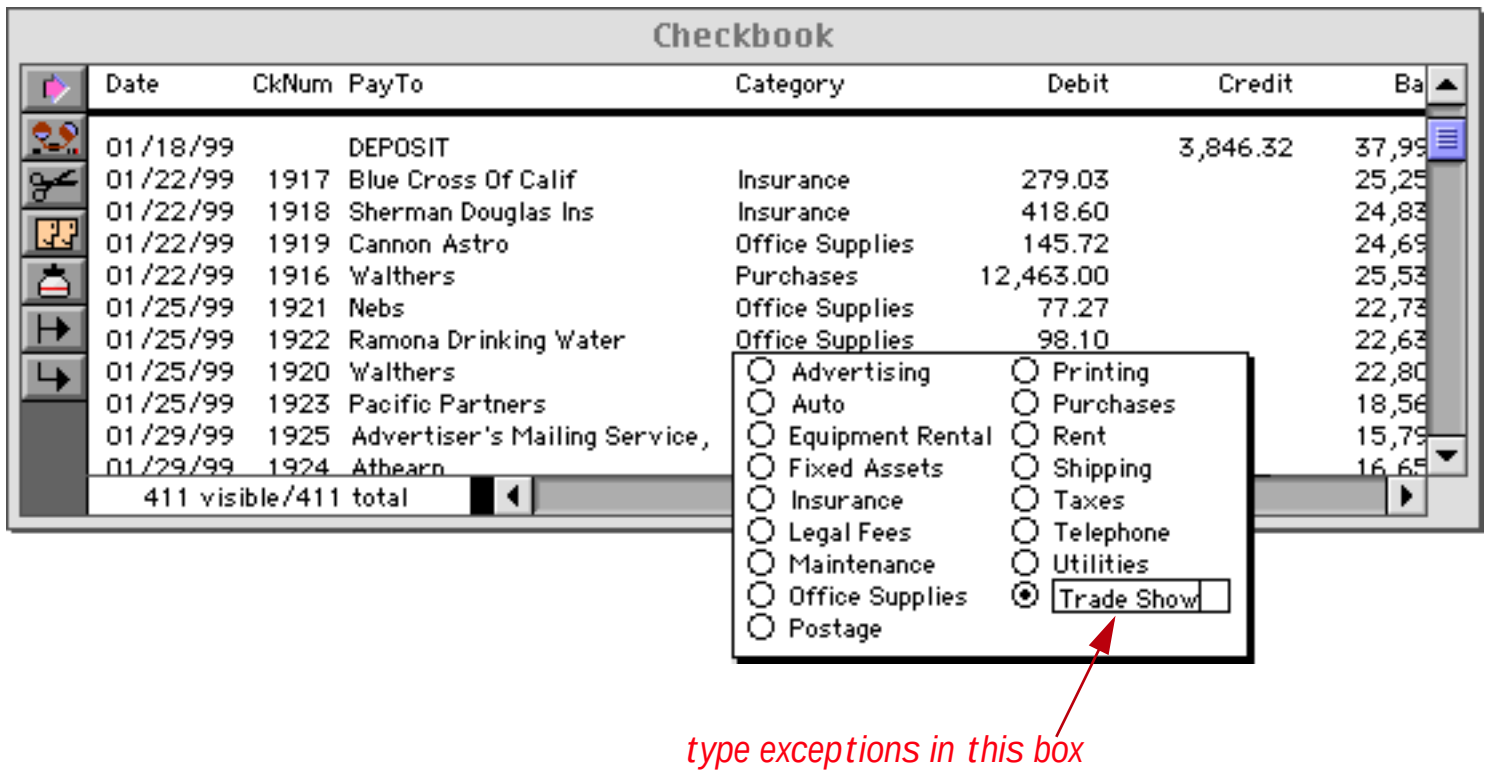


Exceptions

You can use the choices data type even if you cannot anticipate all of the possible choices in advance. For example, you may usually ship via UPS and Federal Express, but occasionally you use a variety of other shippers. In this situation you need to allow exceptions to the choice list. To allow exceptions, type in a line of underscores at the end of the list.



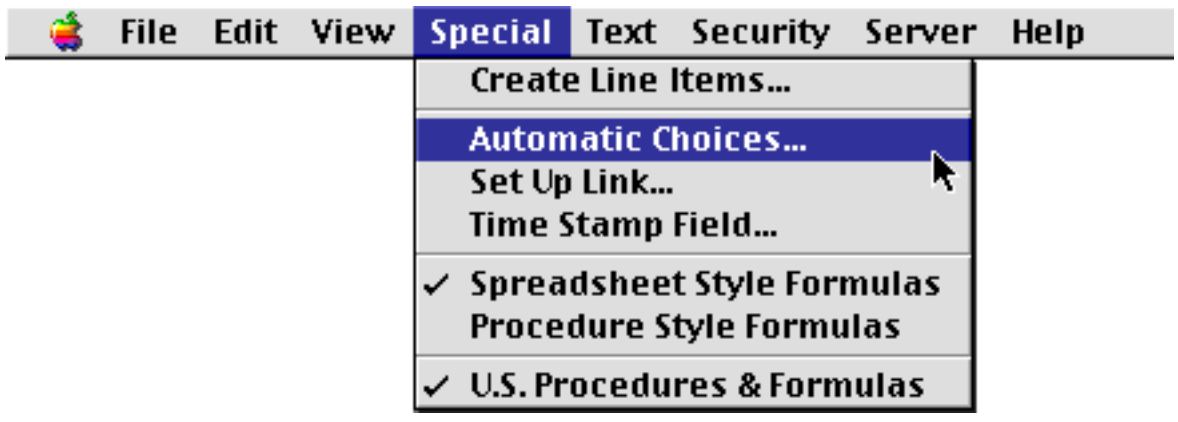
In the data sheet, this line of underscores will appear as a text entry box where any value may be typed in. The number of underscores determines the size of the exception box in the choice palette. For a larger box, add more underscores.



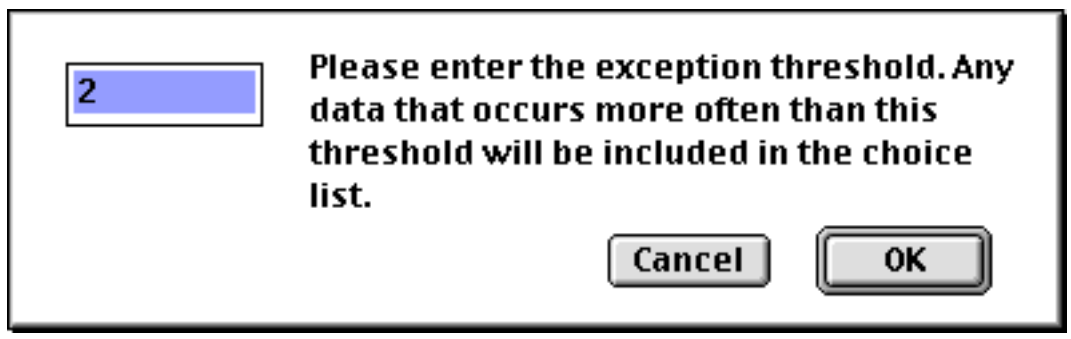
When a value is entered into the database that is not one of the choices in the list, the value is stored as an exception. Panorama must store the entire value instead of an internal number. As long as the number of exceptions is small the choices data type is still useful. However, if there are too many exceptions, you should stick to the normal text and numeric data types. In the worst case, if all the values are exceptions, the database will actually take more memory than it would if you simply stored the data as text.

Generating a List of Choices Automatically

If your database already contains data, Panorama can examine the database and automatically generate a list of choices for you. To do this you must use the **Automatic Choices** command (Special Menu) available in the design sheet. Select the field (by clicking on the corresponding row in the design sheet), then choose **Automatic Choices** from the Special Menu.



Before Panorama can generate the choice list you must specify an exception threshold.



This value tells Panorama how many times a choice must appear in the database to be included in the choice list. Any value that appears fewer times than the threshold will become an exception. Any value that appears that many times or more will be included in the choice list. When you press **Ok**, Panorama will scan the database and create a choice list containing all the choices that occur the same number or more times than the number specified by the exception threshold.

How do you decide what value to use for the exception threshold? There is no hard and fast answer. For larger databases you will probably want to use a larger value. Experiment with different values to see the effect on the choice list, then choose a value that includes the common choices on the list, while leaving out the values that only occur occasionally

Updating the Choice List

As a database grows, the choices list may need to change over time. Choices that were once exceptions can become common, while once popular choices become obsolete. If this occurs, you can update the choice list—either manually or with the **Automatic Choices** command.

Using Math Operations with Choices

Sorry, no can do. If you need to perform any kind of math calculations (**Total**, **Average**, etc.) you must store the information using the numeric data type. However, you can **Count** a choice type field.

Sorting Choices

Data stored using the choices data type is sorted according to the order of the choice list. Therefore, if you want the choices to be sorted alphabetically, you must make sure that the choice list is in alphabetical (A-Z) order. The **Automatic Choices** command does this for you.

Sometimes you may wish to sort the choices in a different order. For example, Olympic medals should be sorted in the order Gold, Silver, Bronze instead of alphabetically (Bronze, Gold, Silver). If you need the choices to sort non-alphabetically simply set up the choice list in the order you want.

Warning: If your choice list is in non-alphabetic order the sort order of exceptions is ambiguous. For example if the choice list is Gold, Silver, Bronze where does the exception Copper go? The final order is not predictable. You should only use a non-alphabetic choice list if there are no exceptions allowed.

Chapter 7: Data Entry & Editing



Before you can organize, analyze, or report anything, you have to get your data into the computer. Usually this means using the keyboard to type it in. Panorama has been designed to help make this task fast and accurate.

Of course, if the information is already stored on a computer, you may be able to access it without re-typing. Panorama can exchange data with most Macintosh software packages, and with many PC and mainframe packages as well. See “Importing a Text File” on page 145.

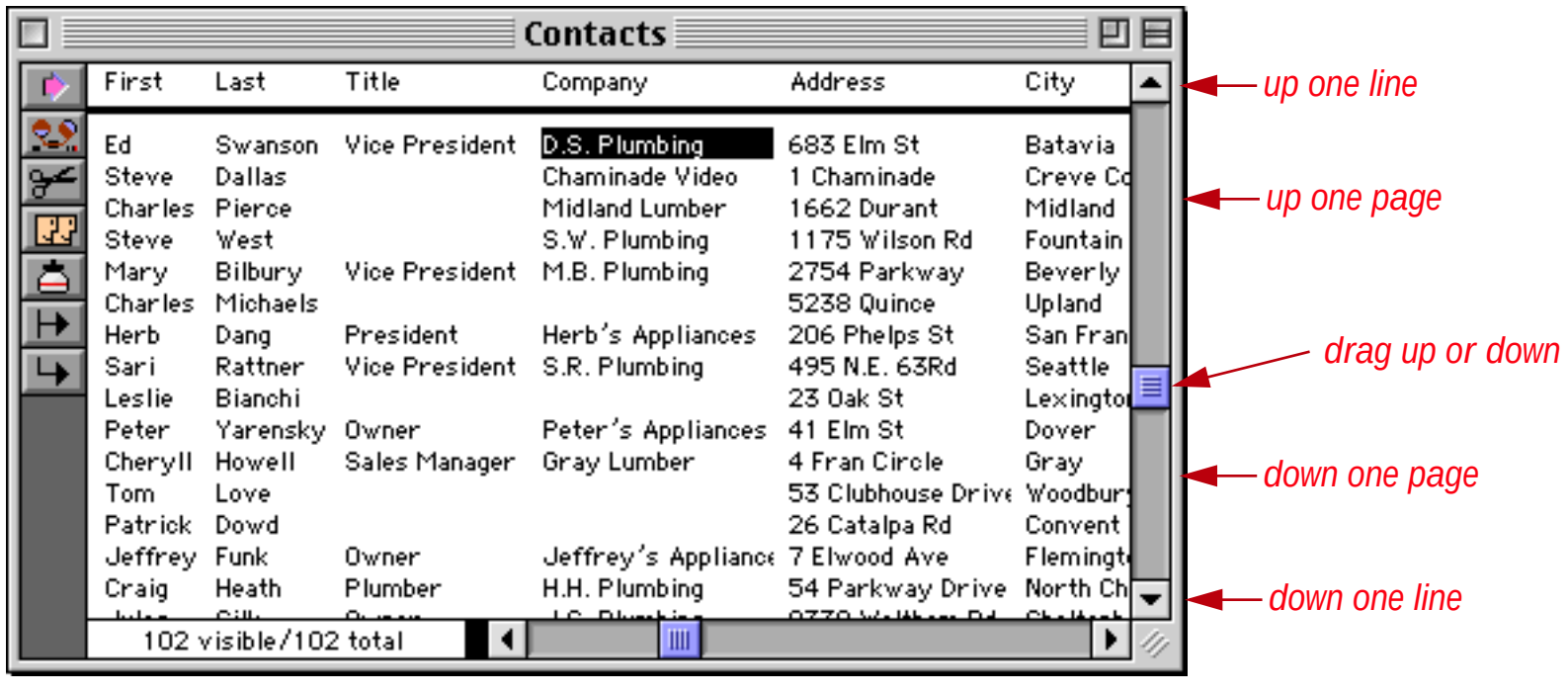
Editing Records

A new database starts out with just one lonely record. Over its lifetime hundreds or even thousands of records will be added to a typical database. In addition, many records will become obsolete and be deleted. Both inserting and deleting records are easy tasks with Panorama.

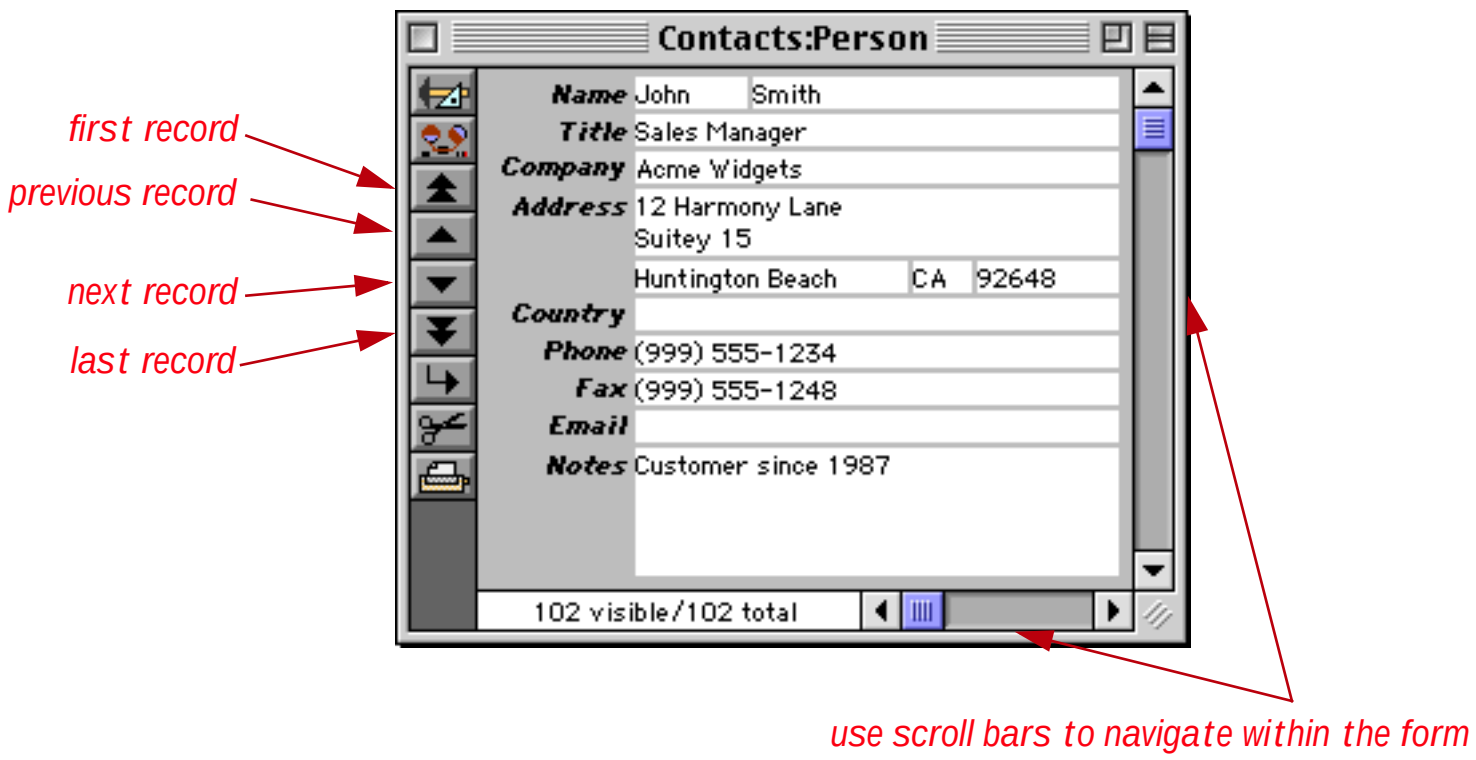
Moving From Record to Record

Unless your database is very small, only a small part of it will be visible at a time. You can shift the data within the window to work with different parts of the database. To shift the data you either scroll (data sheet) or flip from page to page (form).

Use the vertical scroll bar to scroll a data sheet to any record in the database. Click on the scroll bar arrows to move up or down one record at a time. Click in the scroll bar’s gray area to move up or down one window at a time. Drag the scroll bar thumb to move directly to any position in the database.



When you are using a form window, the vertical scroll bar works differently. Instead of scrolling to another record, the scroll bar shifts the position of the form within the window. This allows you to get at every corner of a large form even if you are using a small window. To move from record to record, use the **First Record**, **Previous Record**, **Next Record**, and **Last Record** tools in the tool palette. The **First Record** tool brings you to the very first visible record in the database—the top line of the data sheet. The **Last Record** tool takes you to the end of the database—the bottom line of the data sheet. The **Previous Record** and **Next Record** tools move one record at a time. You can also move up or down one record at a time by pressing the **Up Arrow** or **Down Arrow** keys.

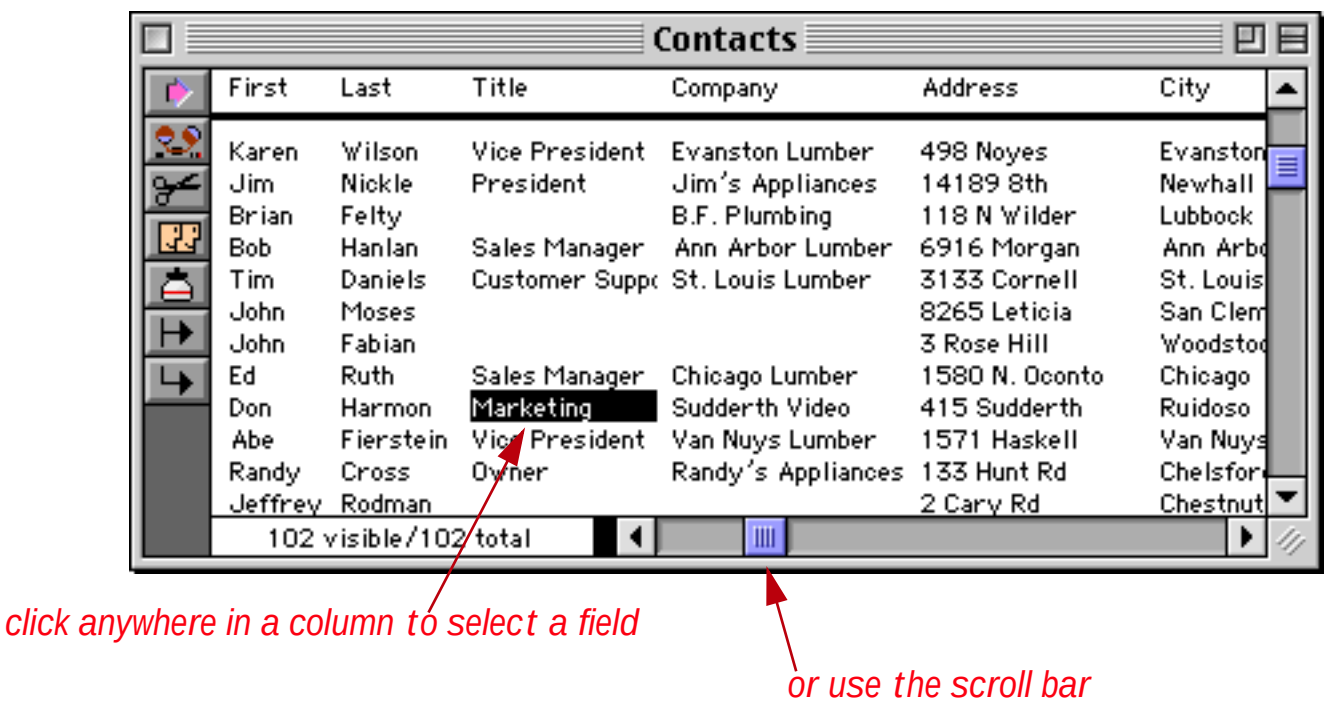


If the database has more than two open windows (for example a form and a data sheet), the position of both windows will always remain synchronized. In other words, if you scroll the data sheet the form will follow and vice versa.

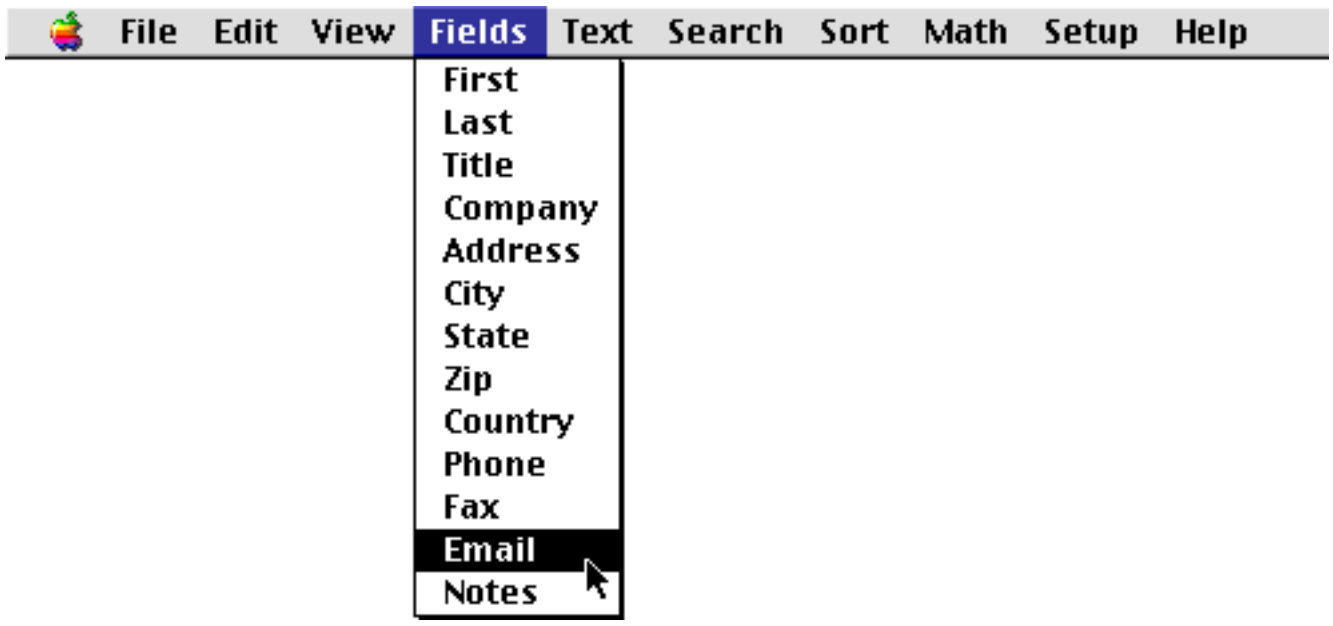
In addition to manually moving from record to record, you can let Panorama search for the information you want to look at or modify. The **Find/Select** command (Search Menu) can use a variety of criteria to locate the information you need. See “The Find / Select Dialog” on page 369, for more information on this command.

Moving from Field to Field

Within a data sheet, you can move to another field by clicking anywhere the field's column (if it is visible) or by clicking on the horizontal scroll bar. (See "Splitting a Window" on page 198 for information on how to split a window into two separately scrollable "panes.")

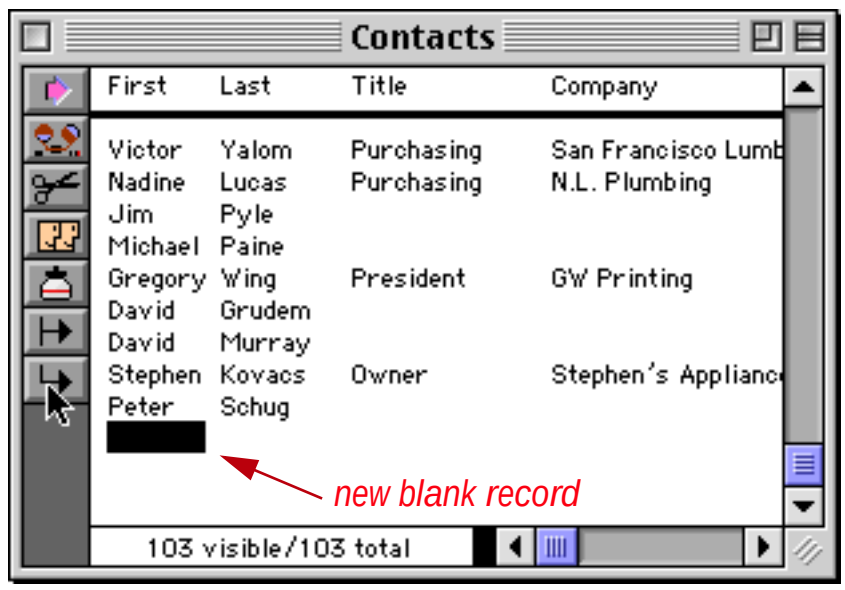


You can also use the **Fields** menu to move to a specific field. This menu contains a list of all of the fields in the current database. The fields are listed in the order that they appear in the data sheet. To move to a field, simply select it from the menu. The **Fields** menu makes it easier to select a specific field when the database has dozens (or even hundreds) of fields.



Adding a New Record

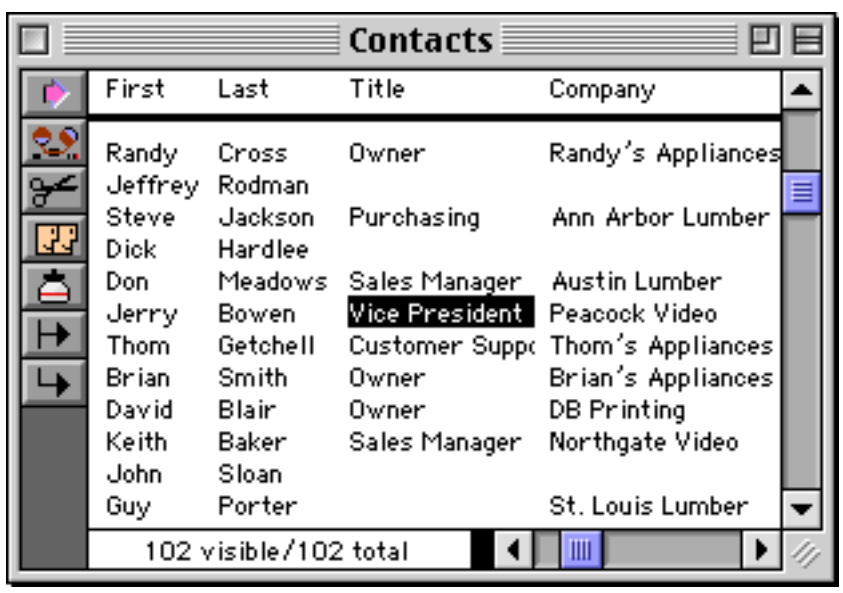
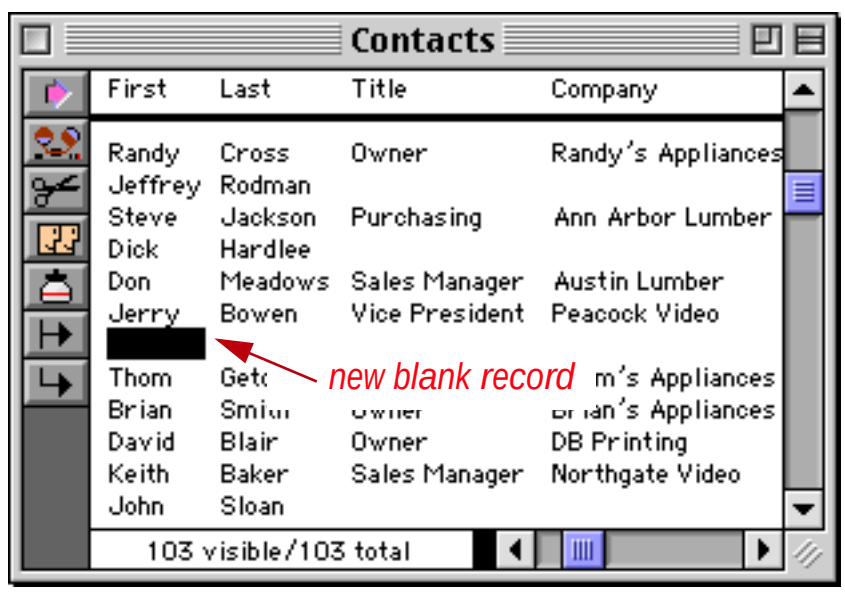
To add a new record to the end (bottom) of the database, either click on the **Add New Record** tool or choose **Add New Record** from the Edit Menu.

before **Add New Record**	after **Add New Record**
	

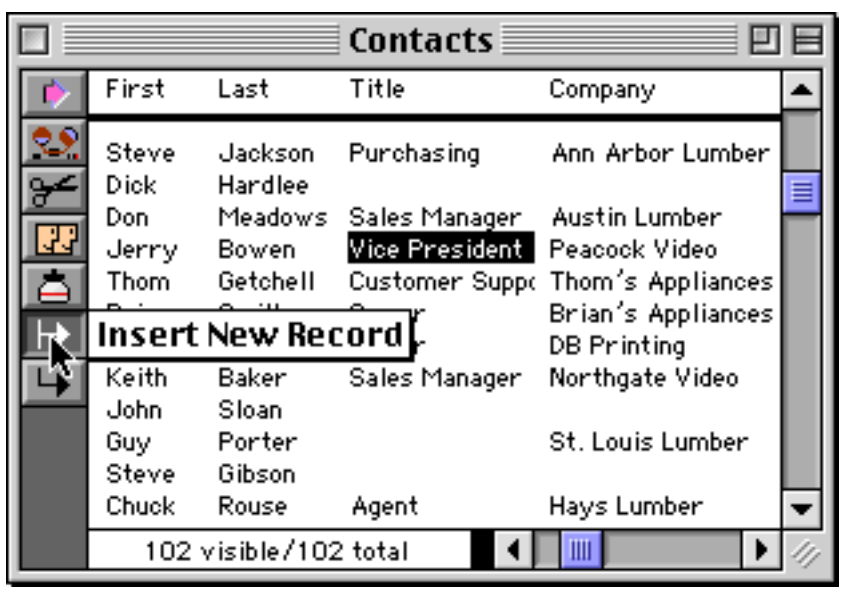
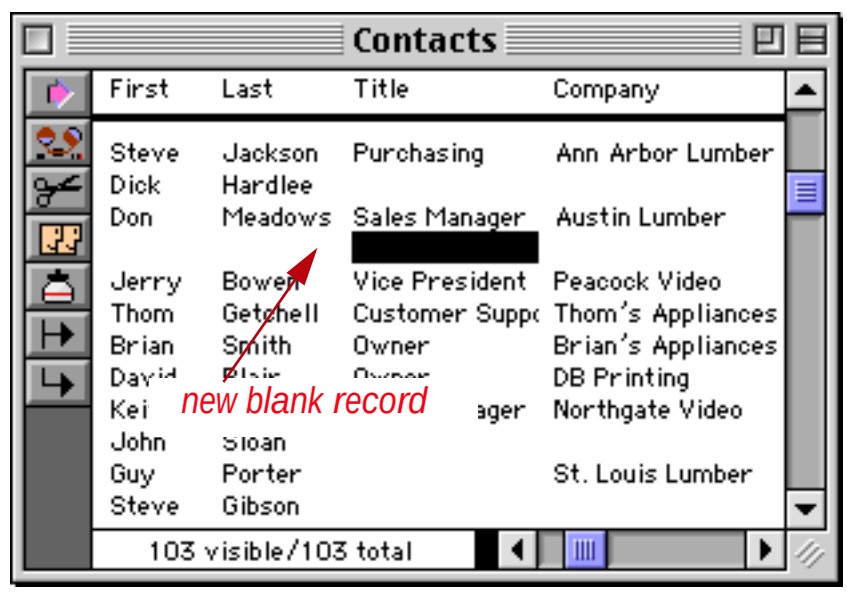
You can also add a new record by tabbing from the end of the bottom line of the data sheet.

Inserting a New Record

When you are working in the data sheet, you can insert a new record either above or below the current record. (In a form you can only add new records at the end of the database.) To insert a new record after the current record press the **Return** key.

before **Return**	after **Return**
	

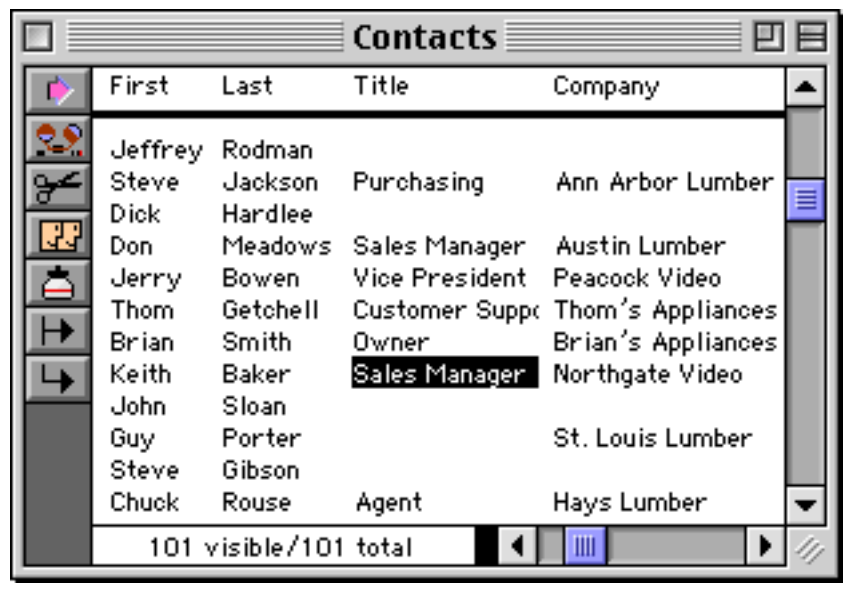
To insert a new record before the current record, click on the **Insert Record** tool.

before **Insert Record**	after **Insert Record**
	

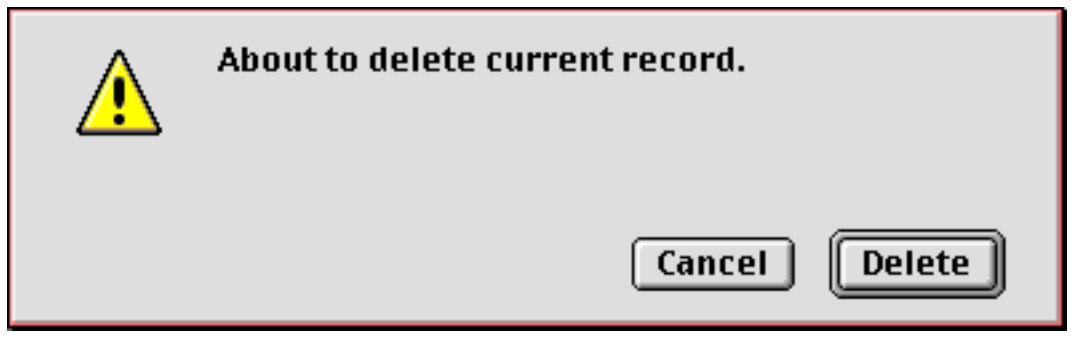
Usually new records are completely blank, ready for your input (as shown in the examples above). You can, however, ask Panorama to automatically fill in one or more cells whenever a new record is created. A field can default to a fixed value (like yes or no, or taxable, or today's date), an automatically incrementing number (1, 2, 3, …), or a copy of the data in a previous record. See "Default Values" on page 333 for more information.

Deleting a Record

To delete an entire record, click on the **Cut Record** tool.

before **Cut Record**	after **Cut Record**
	

If you are using the data sheet, you can also delete a record by pressing the **Delete** or **Backspace** key (upper right hand corner of the keyboard, above **Return**). Panorama will display an alert asking you to confirm that you really want to delete the record. (Mac only tip: If you want to skip the alert, hold down the **Option** key while you delete the record.)



If you delete a line accidentally, you can use **Undo** to restore the line. You can also get the line back with the **Paste Record** tool.

Tip: A Panorama database must have at least one visible record—it cannot have zero records. If your database has only one record, Panorama will not allow you to delete it.

Deleting Multiple Records

Sometimes you may need to systematically delete large numbers of records. For example, you might need to delete all invoices previous to 1987 or all students with below passing grades. Instead of deleting these records one-by-one you can let Panorama do most of the work for you. First use the **Find/Select** command to select the records you want to keep. Then use the **Remove Unselected** command to delete everything else. Remember, however, that you cannot delete every record—you must leave at least one record in the database at all times. See "The Find/Select Dialog" on page 369 and "Permanently Removing Unselected Data" on page 378 for more information on these commands.

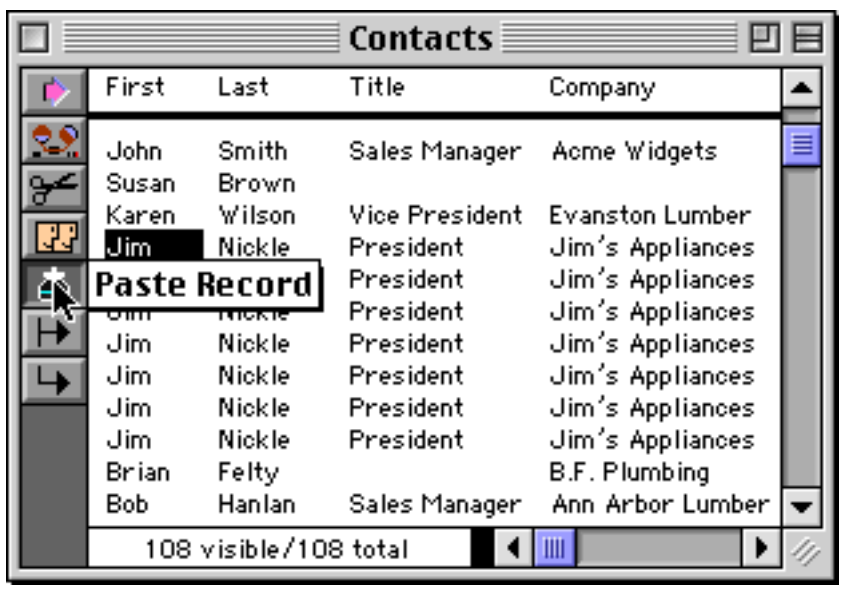
Delete All

To delete all the data in the database, use the **Delete All** command in the Edit Menu. This command deletes all the data, leaving just one blank record. Before it performs this dastardly deed, Panorama asks you to confirm that you really know what you are doing. Keep in mind that there is no **Undo** after **Delete All**. (However, if you saved a copy of your database, you can **Revert to Saved**. See "Revert to Saved" on page 133.)

You can use the **Delete All** command to set up a clone of an existing database. First open the original database, then use **Save As** to save it under a new name (See "Saving a Database" on page 131). Finally use **Delete All** to empty the new database. The new "cloned" database will contain all of the forms, crosstabs and procedures of the original database, but no data.

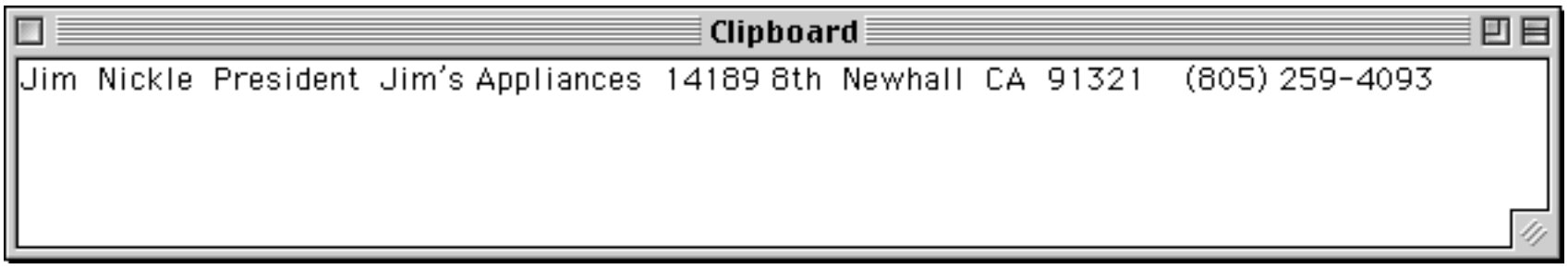
Duplicating a Record

In the data sheet you can easily make one or more copies of a record. Use the **Copy Record** tool to copy the current line into the clipboard, then use the **Paste Record** tool to paste the line back into the database. The **Paste Record** tool inserts a new line just above the current line and then pastes the contents of the clipboard into the new line. You can paste the record back into the database as many times as you like. In this illustration the **Paste Record** tool has already made six copies of the line, and is about to create a seventh.



The Clipboard Window

If you would like to see the contents of the clipboard use the **Show Clipboard** command (Edit Menu) to open the Clipboard window. You can watch the clipboard change each time you cut or copy something. Here's what the clipboard looks like as we are copying the records in the example above.



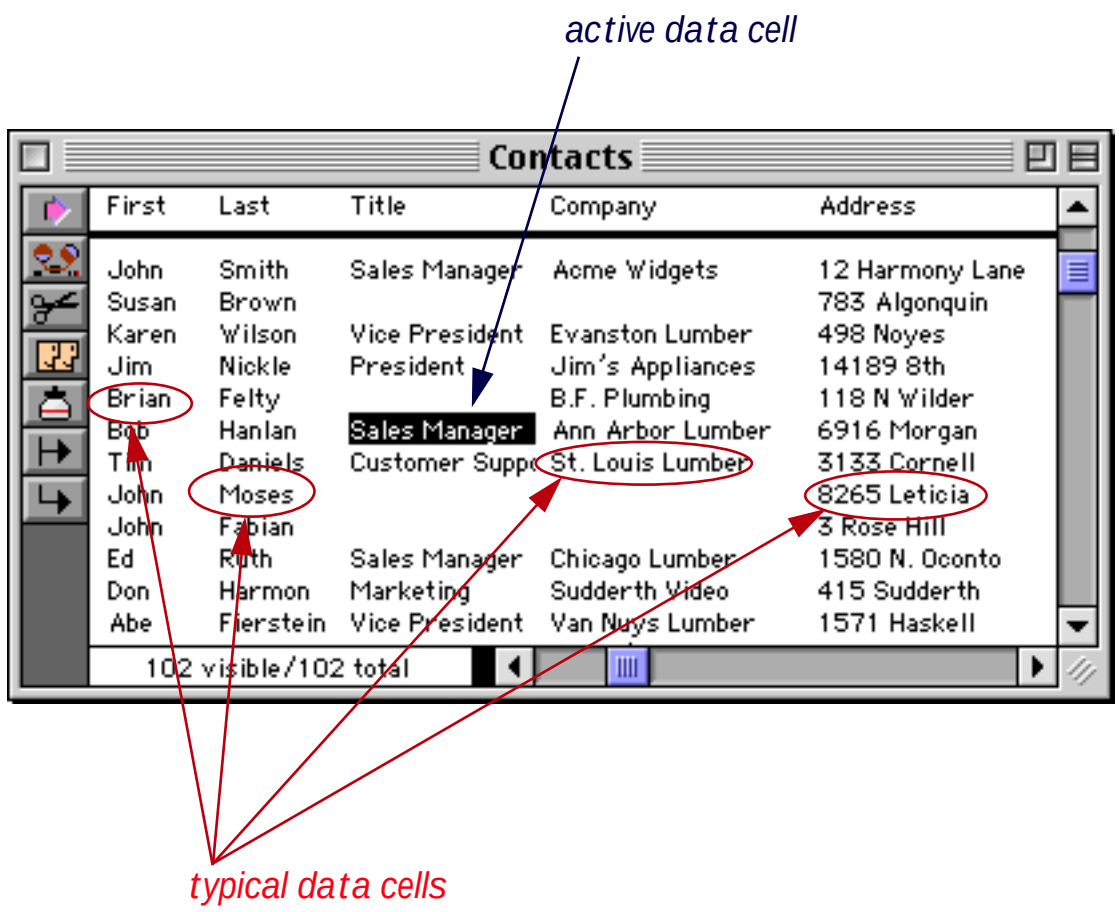
Close the clipboard window when you are finished using it.

Moving a Record

Sometimes you may wish to move a record from one spot to another. You can do this using the data sheet. To move a record use the **Delete** or **Backspace** key or the **Cut Record** tool to remove the line, then use the **Paste Record** tool to insert the line in its new position.

Editing Data Within a Cell

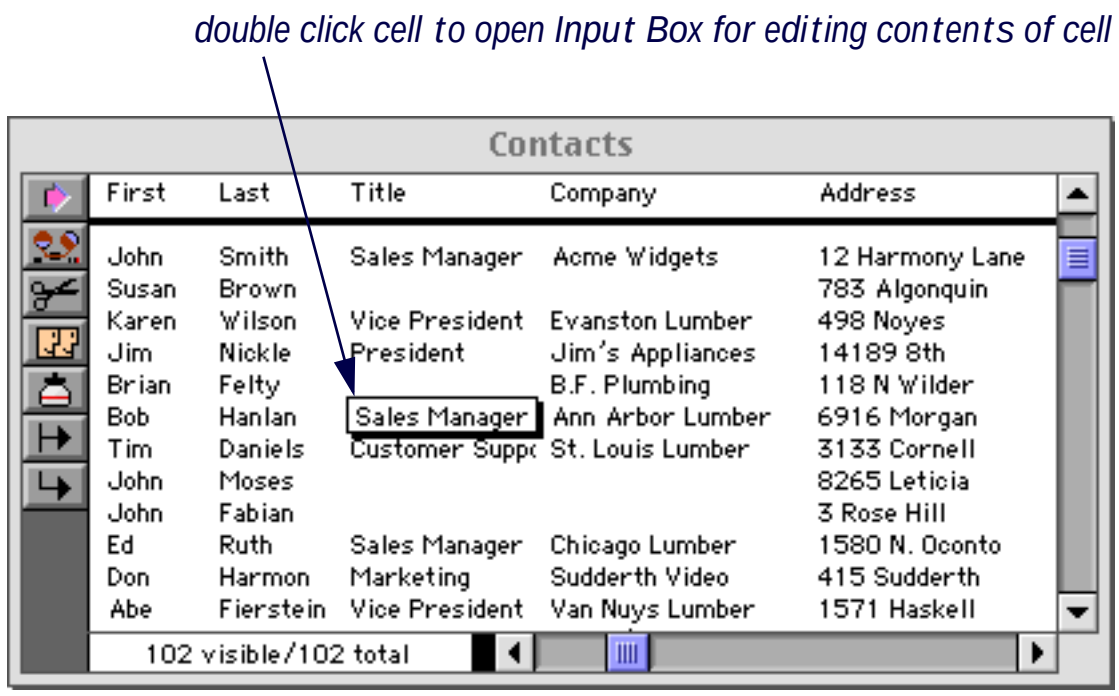
Data cells are the smallest unit of information handled by Panorama. Each data cell contains a single piece of information—a person's name, a phone number, an account balance.



The currently selected cell is called the active cell. Only one cell can be active at a time. You can activate a cell by clicking on it, or by scrolling to it with the scroll bars. You can also move the active cell with the arrow keys.

The Input Box

Every data cell has a pop-up **Input Box** that is used to edit the text within the cell. The Input Box acts like a temporary window that pops up on top of the data cell for data entry and editing.



You can open the Input Box by double clicking on the cell, or by making the cell active and starting to type. Once the Input Box is open, you can edit the text within the cell using the usual mouse editing (word processing) techniques. Specifically, you can click the mouse to select an insertion point, drag the mouse to select a range of characters, cut, copy or paste selected text using the clipboard, or use the keyboard to type characters at the insertion point. (If you are not familiar with these techniques you should review the operating system documentation that came with your computer.)

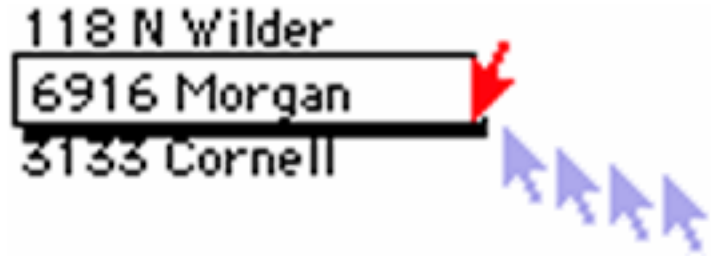
When you have finished editing the text within the cell, press the **Enter** key. This closes the Input Box and updates the data cell with your changes. The Input Box closes automatically (and updates the data) if you click on any other data cell or window. Tip: Clicking anywhere outside of the Input Box is the same as pressing the **Enter** key.

If the Input Box is only one line high pressing the **Return** key will close the Input Box and update the data cell. If the Input Box is more than one line high the **Return** key adds a new line to the data cell (see the next section).

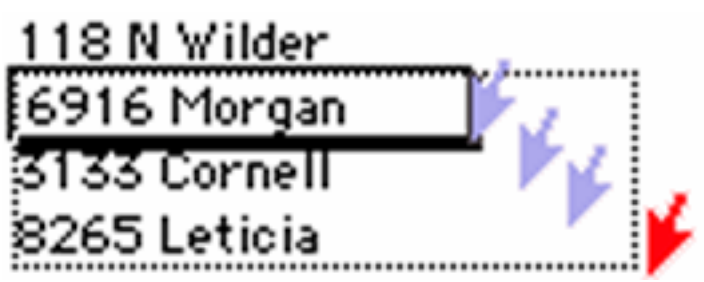
If you would like to close the Input Box without updating the data cell, press **Command-Period** (Mac) or **Control-Period** (Windows). Pressing the **Esc** key also closes the Input Box without updating the data cell. Or you can use the **Undo** command to restore the original text, then press the **Enter** key.

Expanding the Input Box

One of the most powerful features of the Input Box is that it can be expanded to accommodate large amounts of data. To expand the Input Box, move the mouse to the lower right corner of the box. When you reach the corner, the arrow will flip over. In the illustration below, you see the normal arrow cursor as you approach the corner. The arrow flips over when it reaches the corner of the Input Box. (The arrow doesn't actually change color as shown here, the color is simply to make the illustration more clear.)



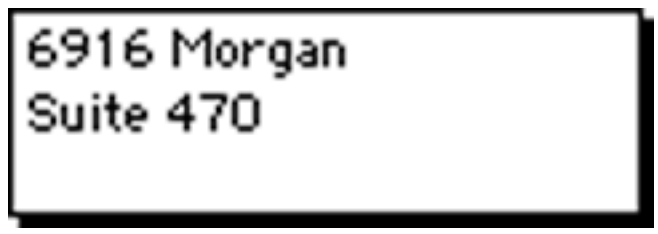
Once you see the upside down arrow, press the mouse and drag the corner of the box to its new location.



When you release the mouse Panorama will change the size of the box.

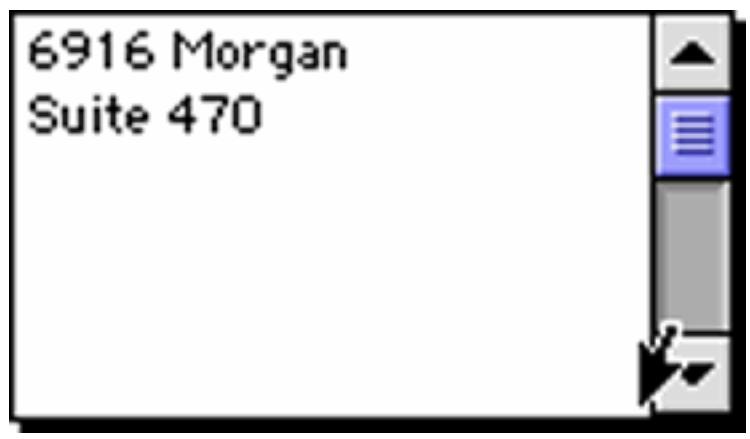


Now you can type additional lines into the Input Box. Press **Return** to start a new line.



Once you change the size of the Input Box, Panorama remembers the size permanently. In the data sheet, the Input Box size is remembered separately for each column, while in a form the size is remembered for each individual data cell object. (Of course Panorama will forget the sizes if you **Close** or **Quit** without saving the database.)

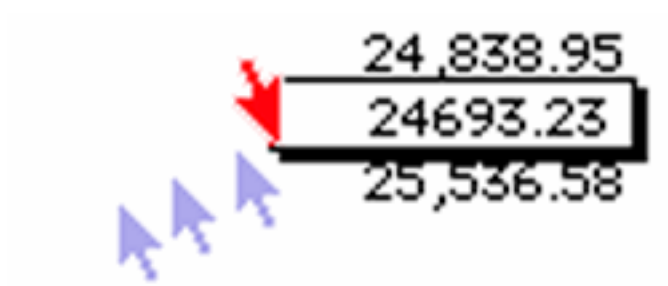
If you make the Input Box more than one inch high, a scroll bar is added on the right hand side of the box. The scroll bar allows you to enter and edit up to 32,767 characters per data cell.



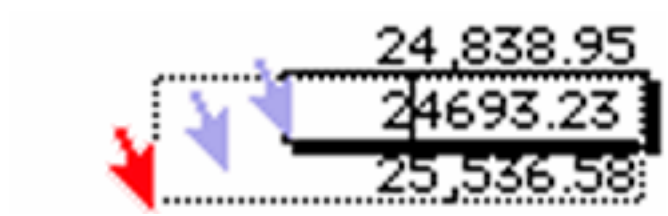
Tip: The scroll bar is actually outside the Input Box. To change the size of the Input Box you must click in the bottom corner of the box itself, just to the left of the scroll bar (as shown above). Do not try to drag the corner of the scroll bar.

Expanding a Right Justified Input Box

If you are editing a right flush data cell, move the mouse to the lower left corner of the box instead of the lower right hand corner.



Once the mouse is over the lower left hand corner you can expand down and/or to the left.



Now you can edit the text in the expanded input Box. Press **Enter** when you are finished.

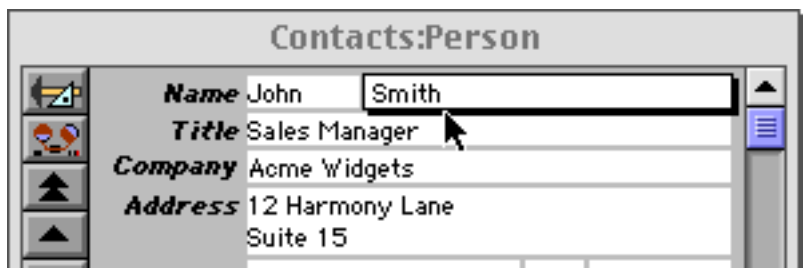


Editing Cells Within a Form

All of the previous examples have shown editing cells within the data sheet. However, a form may contain data cells also. Unlike the data sheet, the cells on a form may be arranged any way you like. They can also be more than one line high.



Just as in the data sheet, you double click on a cell to edit it.



You can also move the mouse over the corner of the cell and drag to expand the Input Box.



See “Working with Data Cell Objects” on page 635 to learn how to create a form with data cells.

As an alternative to data cells, a form may be designed with **Text Editor SuperObjects**. Text Editor SuperObjects allow you to edit text right in the form window—no double click is required. You can simply click or drag on the text to begin editing. Press **Enter** when you are finished. The illustration below shows the effect of double clicking on the word Harmony. As you can see, instead of opening an Input Box this selects the word for editing.



Since the Text Editor SuperObject doesn’t use an Input Box, you cannot expand the size of the editing area “on-the-fly” the same way you can with data cells. The editing area must be defined in advance. On the other hand, the Text Editor SuperObject doesn’t require the extra double click, and works more like other standard applications you may be used to. See “Text Editor SuperObject” on page 639 to learn how to create a form with Text Editor SuperObjects.

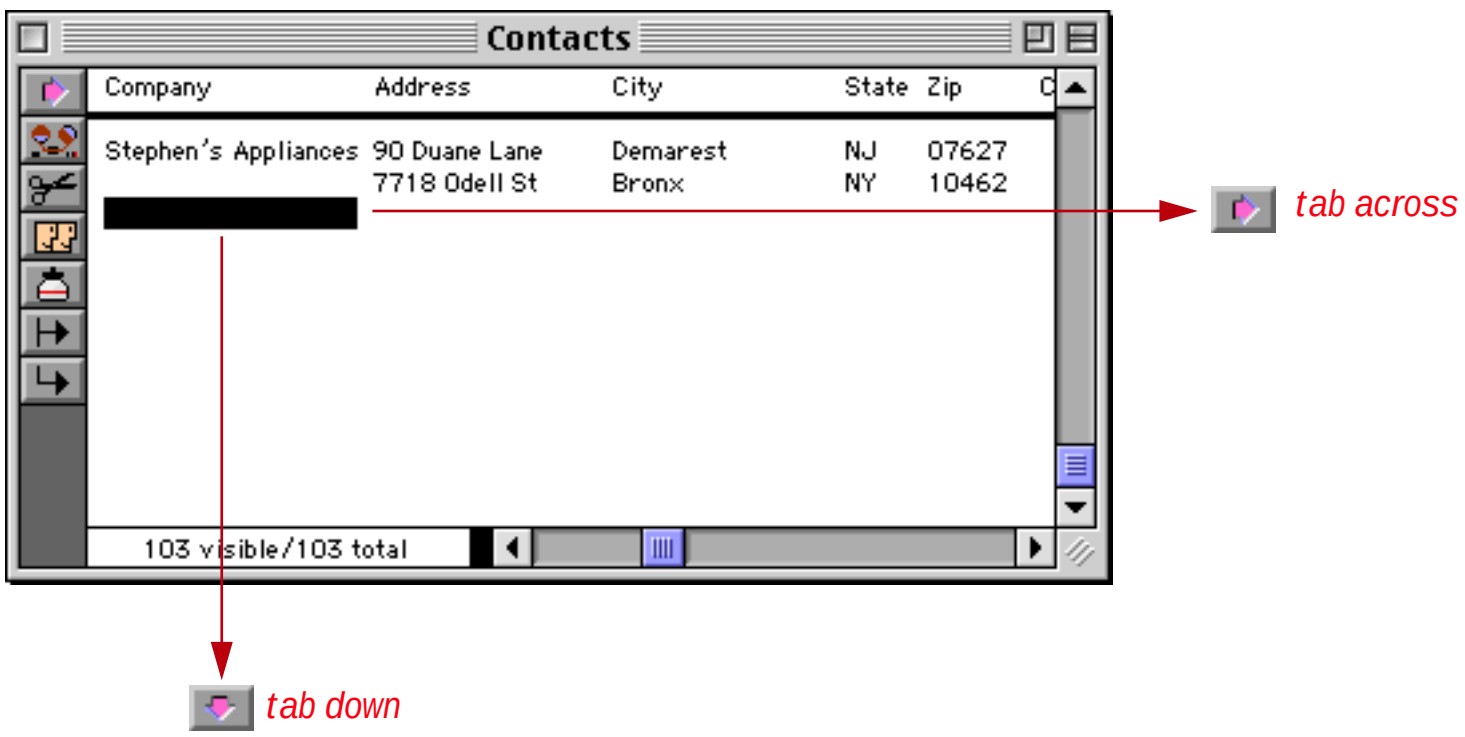
Tabbing from Cell to Cell

To make it easier to enter several cells in a row, Panorama allows you to use the **Tab** key to move from cell to cell as you enter data. In the data sheet you normally move from left to right, in a form from top to bottom. If you want to move backwards press **Shift-Tab**.

Tab Down

In the data sheet the **Tab** key normally moves from left to right. Sometimes it may be more convenient to key a column from top to bottom instead of across a row from left to right. For instance, you may find it easier to key in a dozen names, then a dozen addresses, then a dozen cities, instead of keying in each record separately.

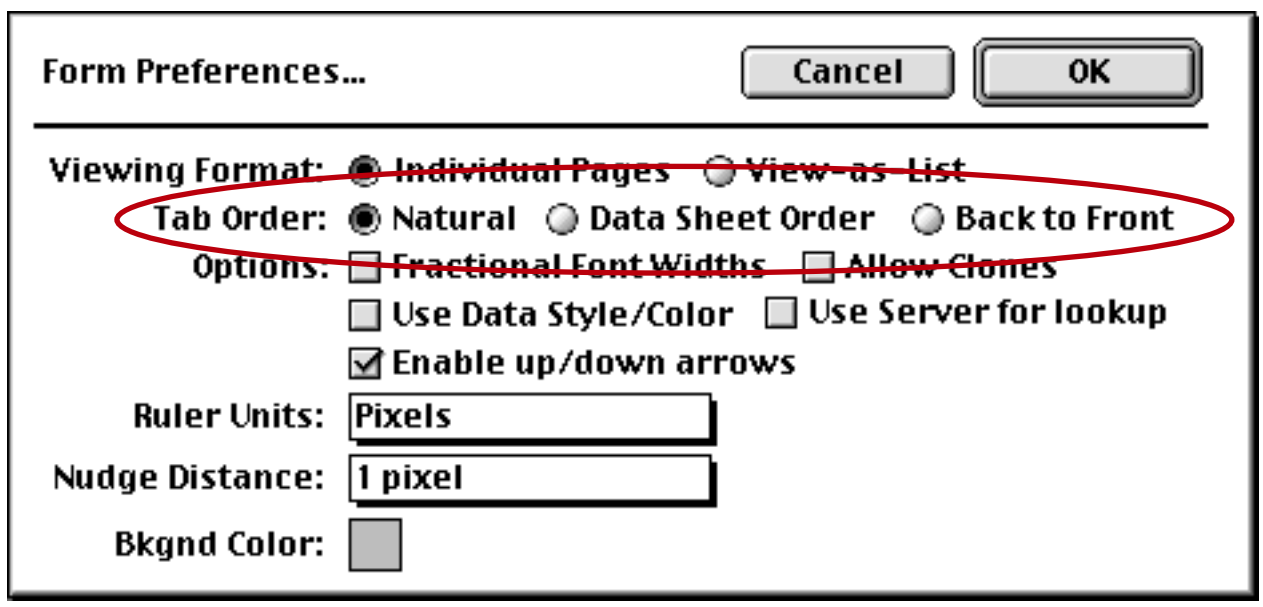
Click on the **Tab Down** tool to change the tab direction from across to down. Click the tool again to change back to normal tab operation. Panorama indicates that tab down is active by changing the arrow direction in the **Tab Down** tool.



Tab down can also be used in forms when the view-as-list option is enabled. See "View-As-List Forms" on page 889 for more information on the view-as-list option.

Tab Order in Forms

Panorama has three tab order options for forms—**data sheet order**, **back to front order**, and **natural order**. Use the **Form Preferences** command (Setup Menu) to specify the tab order option you want to use. (The form must be in graphic design mode. See "Form Modes: Data Access vs. Graphic Design" on page 485 if you are not already familiar with using graphic design mode.)

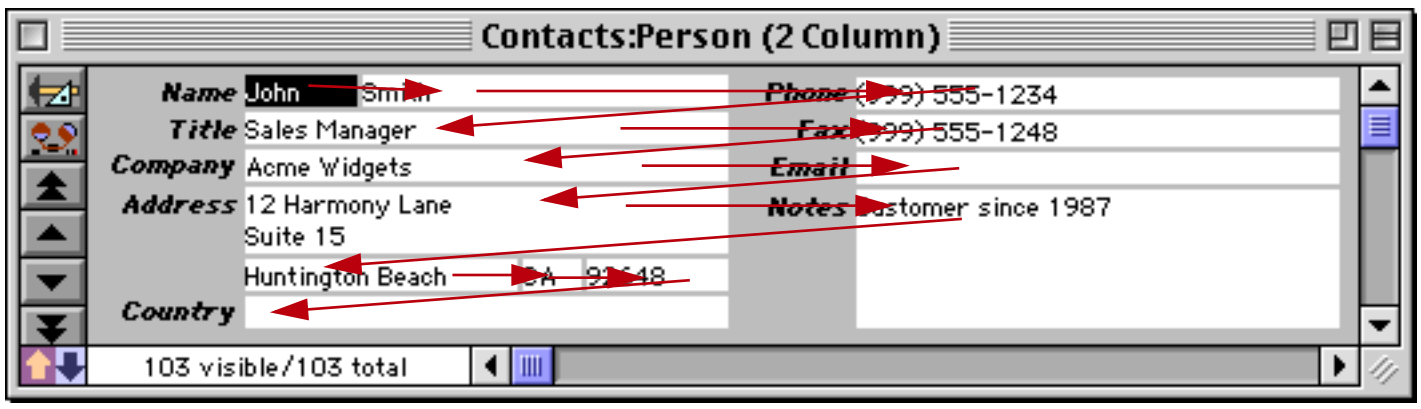


Data sheet order is exactly that—the **Tab** key moves from cell to cell in the same order as it would in the data sheet. However, data sheet order will not work if your form contains one or more variables in addition to fields to be edited (See "Text Editor Options" on page 643).

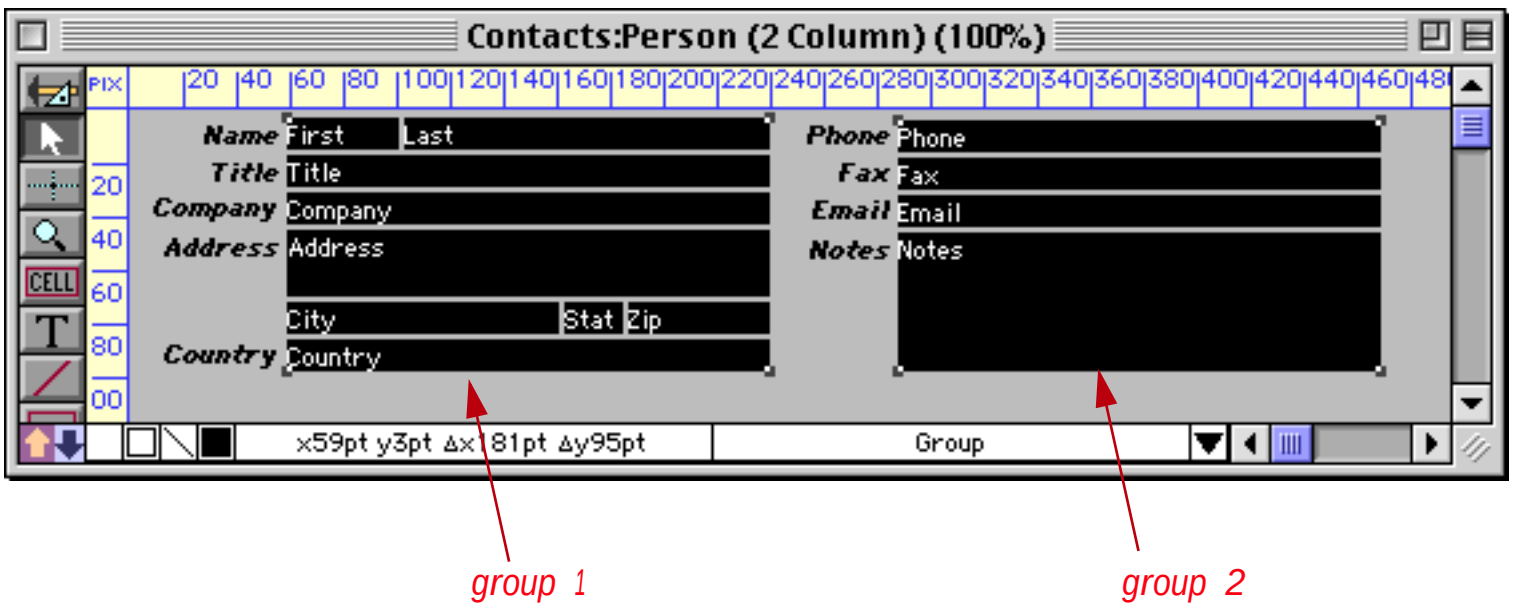
Natural order causes the **Tab** key to move from left to right, then from top to bottom.



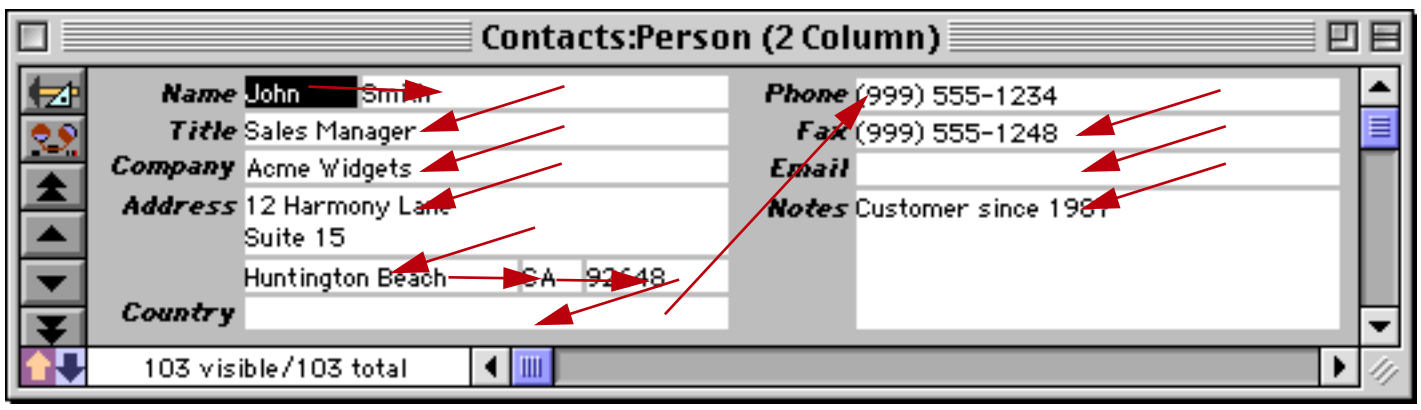
This usually works well (and is the default option), but in some cases isn't really what you want. This is especially true in forms with side by side columns of data.



To fix this you can alter the natural order by grouping data cells together–the **Tab** key will move through all the cells in the group of objects (in natural order) before it moves to the next cell. In this case the data cells need to be brought together into two groups using the **Group** command (see "Grouping Objects Together" on page 536 for more information on grouping graphic objects.)



Now that the cells have been grouped together the tab order will tab through all of the cells in the left hand column before moving to the right hand column.



Back to Front order gives you the most control, but also takes the most work to set up. When this option is enabled the tab order depends on the back to front layering of the data cell objects in the graphic design mode. Use **Send to Back** to bring a data cell to the start of the tab order, and **Bring to Front** to send it to the end of the tab order. See "Changing the Stacking Order" on page 569 for more information on these commands.

For example, suppose your form contained three fields A, B, and C and you wanted to tab from field to field in the order B › A › C. To set up this order click on field B and use **Bring to Front** (the form must be in graphic design mode). Then click on field A and use Bring **to Front**. Finally click on field C and use **Bring to Front**.

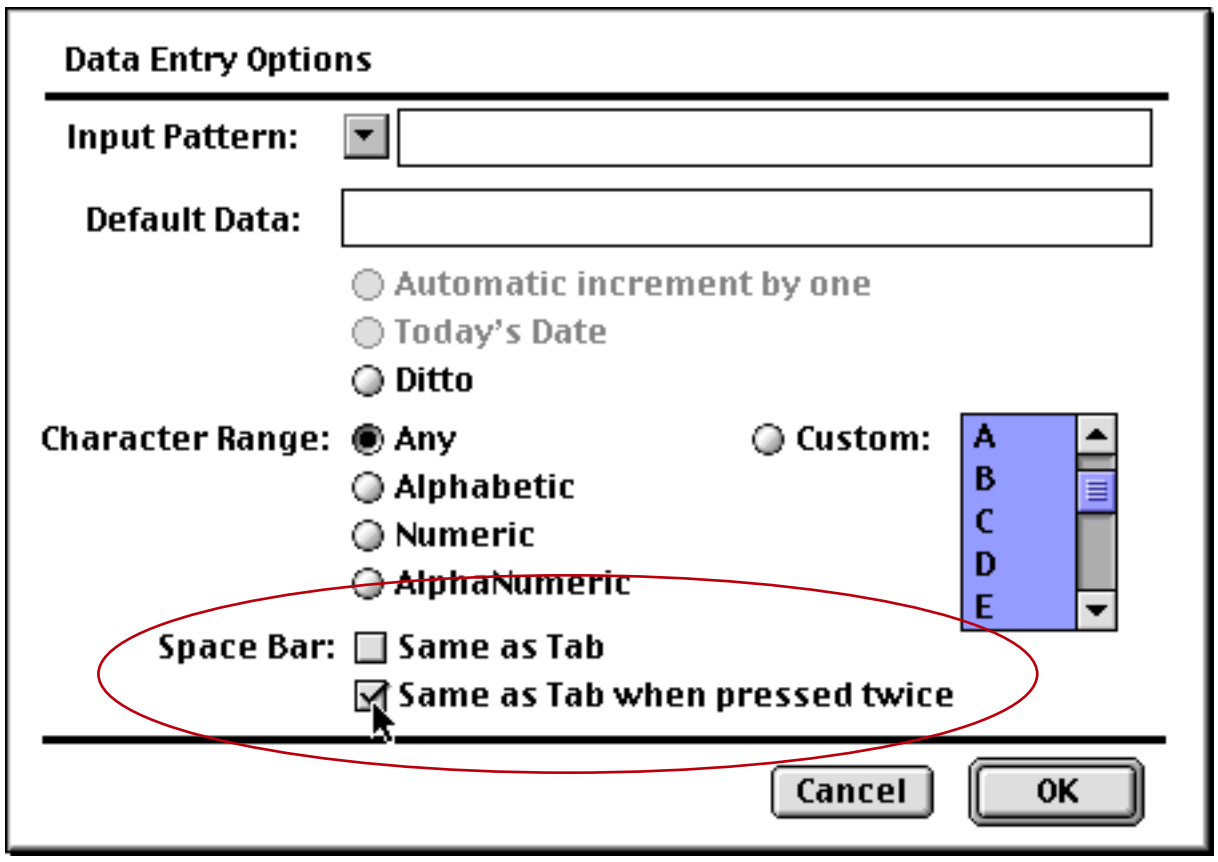
Tabbing with the Space Bar

Using the **Tab** key is a great timesaver when entering lots of data, but it sure is tough on your left pinky. You can use Panorama's **Space Bar Tab** option to give your pinky a rest. When you are using this option, pressing the **Space Bar** bar once or twice tells Panorama to skip to the next cell, just like the **Tab** key.

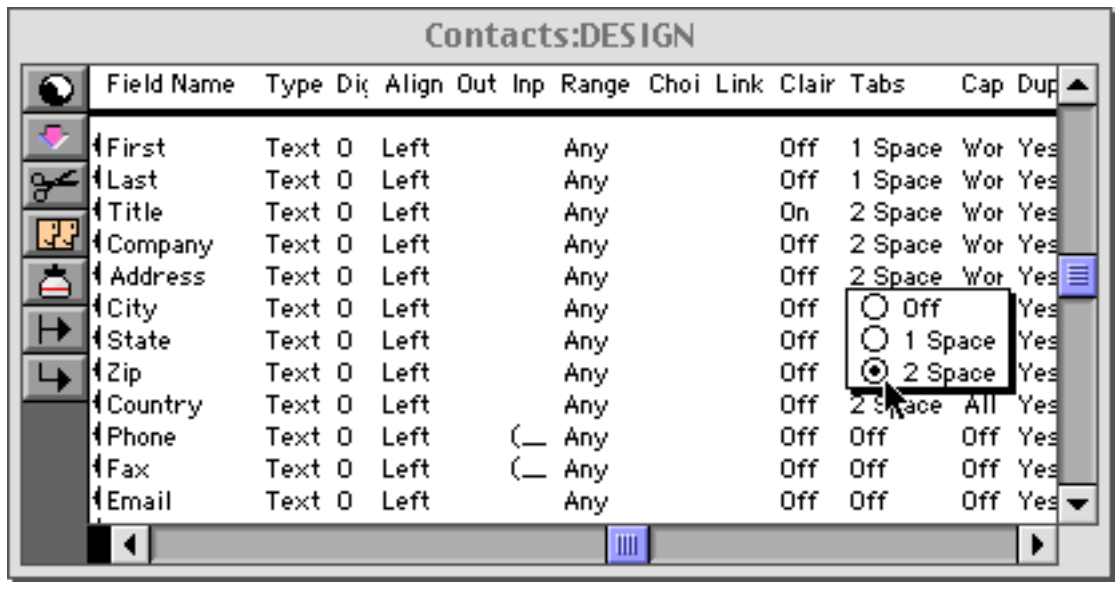
The **Space Bar Tab** option can be configured separately for each field in the database. If a field never contains a space (for example state, zip code, or price) you can use the 1 Space option. This option makes pressing the **Space Bar** skip to the next cell.

If a field sometimes contains blanks (for example name, address, or description) you can use the 2 Space option. When this option is active, you can press the **Space Bar** twice in a row to skip to the next cell. (This option does not usually work well with Clairvoyance.)

To set up the **Space Bar Tab** option, use the Field Properties Data Entry Options sub-dialog (open the **Field Properties** dialog and press the **Data Entry** button).



You can also set up the **Space Bar Tab** option with the Tabs column in the design sheet. (See "The Design Sheet" on page 252 if you are not familiar with the design sheet.)



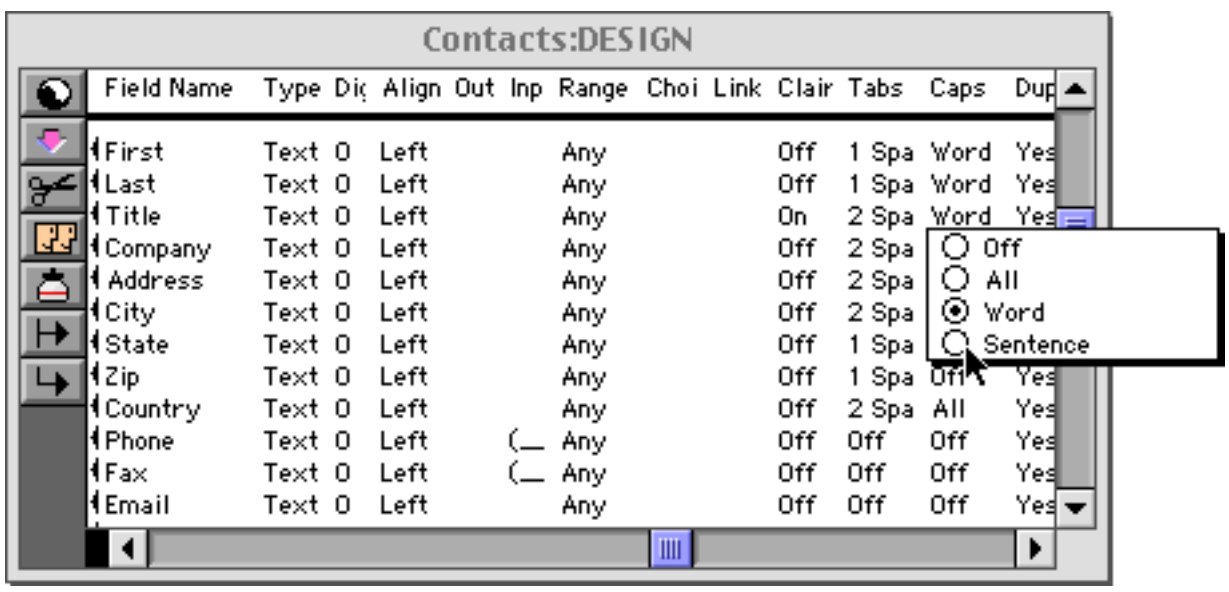
The **Space Bar Tab** option may not sound very exciting on paper. You may wonder if it is worth the bother. This option really does make it easier to enter lots of data without cramping your fingers, especially the left pinky. Give it a shot!

Data Entry Accelerators

Data entry is probably the most tedious task you'll face while using your computer. Let's face it, no one enjoys keying in data. The best way is to get someone else to do it! But since that usually isn't possible, Panorama includes a number of features that help accelerate data entry. You don't have to use any of these features, but when you do, you'll find that ugly data entry tasks are finished faster and more accurately.

Automatic Capitalization

Panorama has four options for automatic capitalization: off, all, word, and sentence. You can set up automatic capitalization with the **Field Properties** dialog or with the Caps column in the design sheet.



The **All** option tells Panorama to capitalize every letter entered into the field. Use this option for fields containing abbreviations—CA, NY, etc.

The **Word** option tells Panorama to capitalize the first letter of each word. Use this option for fields containing names, addresses, etc.

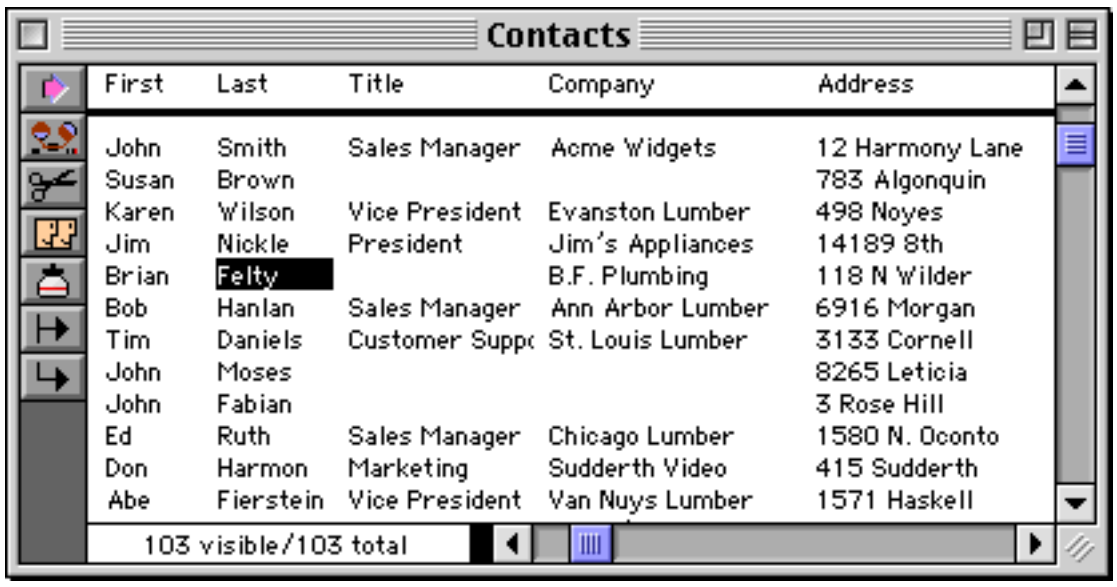
Sometimes you may need to override Panorama's automatic word capitalization. If you need an extra capital letter simply press the **Shift** key (for instance `McDonald`). If you want the first letter of a word to be lower case type the letter twice (`Bank Oof America`), then delete the upper case letter (`Bank of America`).

The **Sentence** option tells Panorama to capitalize the first letter of each sentence. Use this option for fields containing paragraphs of text—for instance catalog descriptions or correspondence.

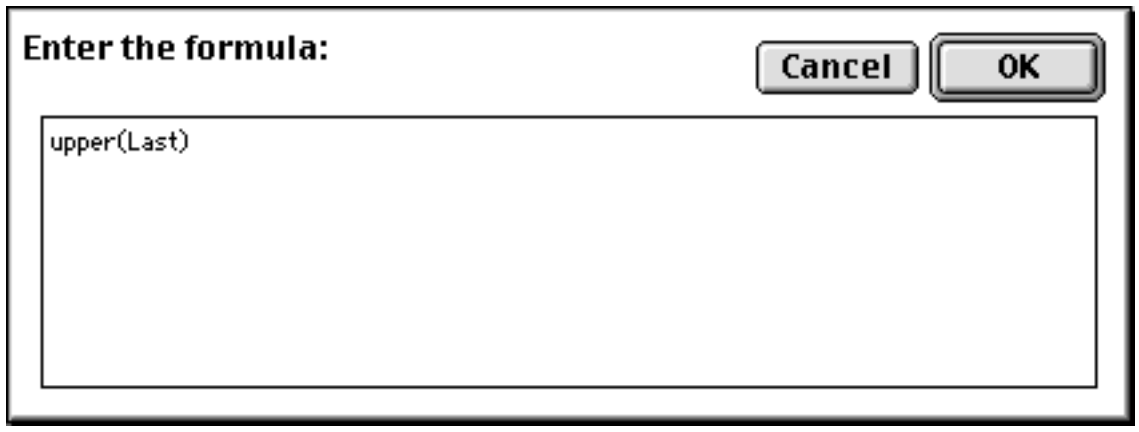
Changing Capitalization of Existing Data

Automatic capitalization only applies to new data as it is typed into the database. It does not change data that has already been entered. It also does not automatically capitalize imported data or data pasted into the database.

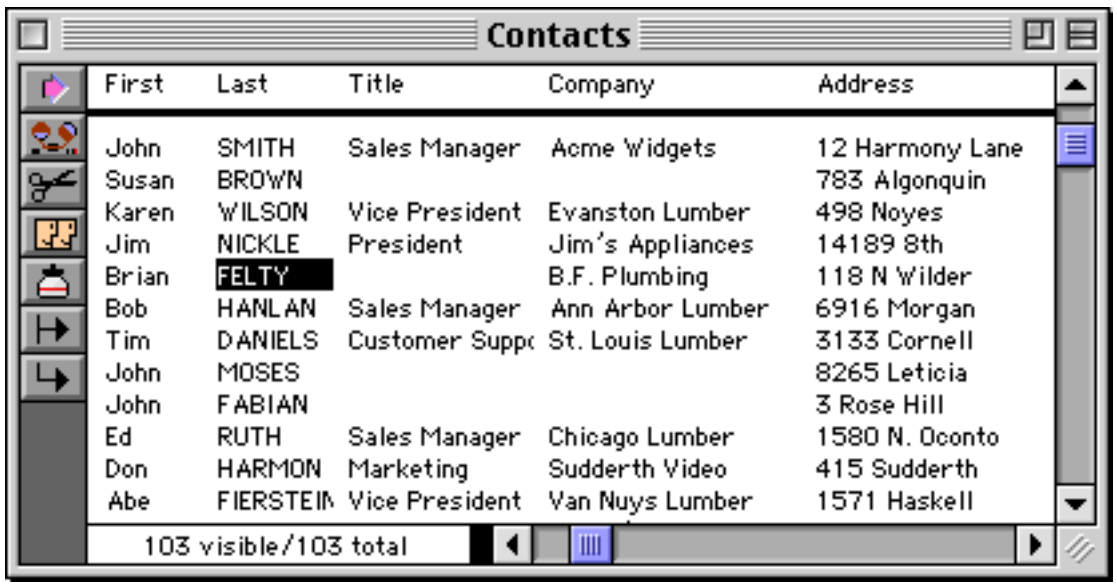
If you want to change the capitalization of existing data, you must use the **Formula Fill** command with a formula containing the **upper()**, **lower()**, or **upperword()** function. This is best explained with an example. Suppose you wanted to convert the last names in the database below to all upper case. Start by clicking anywhere in the Last column.



Now choose **Formula Fill** from the Math Menu, and type in the formula `upper(Last)`.



When you press the **OK** button, the field will be converted to all upper case.



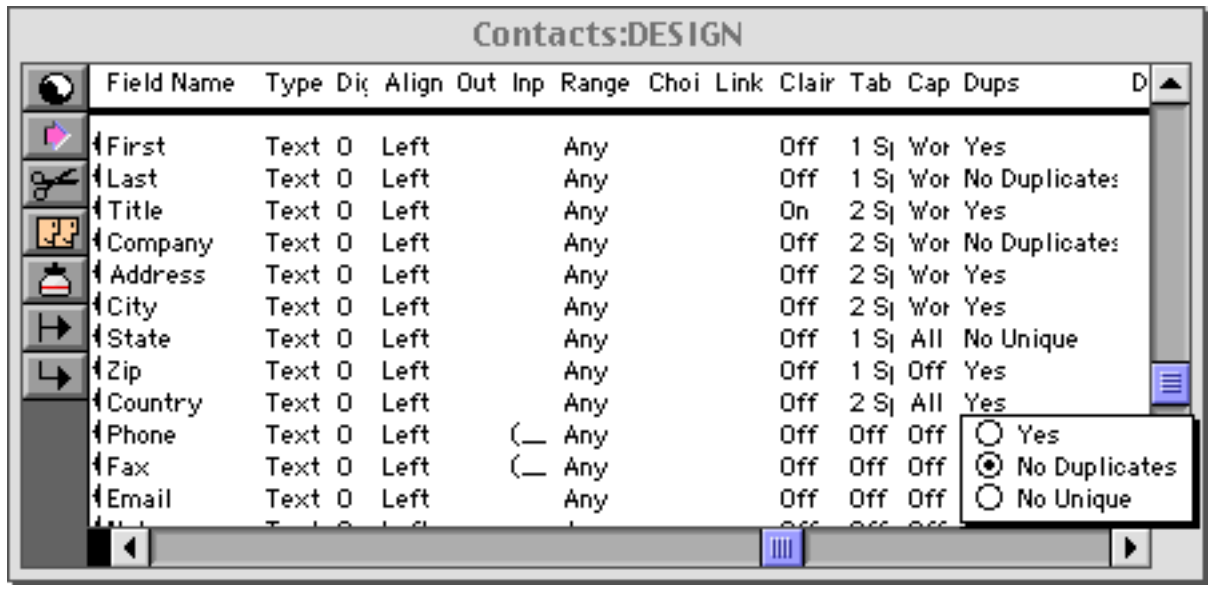
See "Filling a Field with a Formula" on page 453 for further details about the **Formula Fill** command.

Note: If the field name in the previous example had contained a space or punctuation mark, you would have needed to enclose the field name in chevrons, for example `upper(«Last Name»)`. Alternately, you can simply use the formula `upper(«»)` which will always refer to the current field. See "Formulas" on page 1165 for more information about designing formulas.

Checking for Duplicate Data

Panorama usually does not care if you enter duplicate information into a database. However, if you wish you can ask Panorama to check for duplicate data every time you enter or edit a data cell in a given field.

Panorama has three options for checking duplicate data—Yes, No Duplicates and No Unique. You can set up duplicate checking with the **Field Properties** dialog or with the Dups column in the design sheet.



The **Yes** option simply tells Panorama to allow duplicates. This is the default.

Use the **No Duplicates** option to make sure that a value is not entered more than once. For instance, a checkbook database should never have duplicate check numbers.

The **No Unique** option tells Panorama to warn you if you attempt to enter a value that is not already in the database. For instance if a field contains only Yes / No values, this option would warn you if you attempted to enter True or False.

When Panorama encounters a duplicate or unique value (depending on the option), it warns you. However, it does not prevent you from entering the value. You are given the option of entering the data even though it conflicts with the existing data—it's up to you.



Checking for Duplicates in Existing Data

Checking for duplicates only happens when new data is typed into the database. Panorama does not check data that has already been entered, and it does not check data that is imported or pasted into the database.

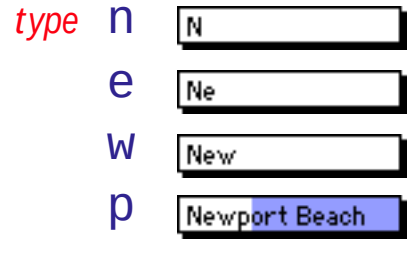
There are several techniques for checking for duplicates in existing data. See "Select Duplicates" on page 388 to learn how to use the **Select Duplicates** command. Another method is to sort the data and then use the **UnPropagate** command to identify the duplicates (by searching for blank cells). See "Using UnPropagate to Eliminate Duplicates" on page 470 for details on the **UnPropagate** command and this technique.

Clairvoyance®

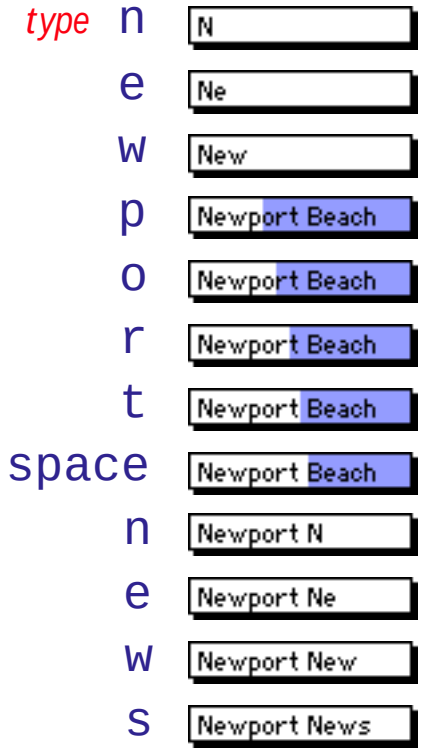
Many databases contain fields where the same information is repeated over and over. For instance, a checkbook will contain the same bills month after month—rent, phone, utilities, charge cards. Another example is an inventory database that contains many items from each vendor, with the vendor name repeated over and over. Panorama's Clairvoyance feature anticipates when you are about to enter data that has been entered before, and completes the entry for you. This can save you a lot of typing, and helps improve consistency as well.

How Clairvoyance® Works

How can Panorama anticipate what you are about to type? The secret lies in Panorama's ability to scan the database in a fraction of a second. When you are using Clairvoyance, Panorama scans the entire database each time you enter a character. As it scans the database, it checks the characters you have typed against the data already in the database. When there is only one possible match, Clairvoyance guesses that you are about to repeat yourself and completes the word or phrase for you.

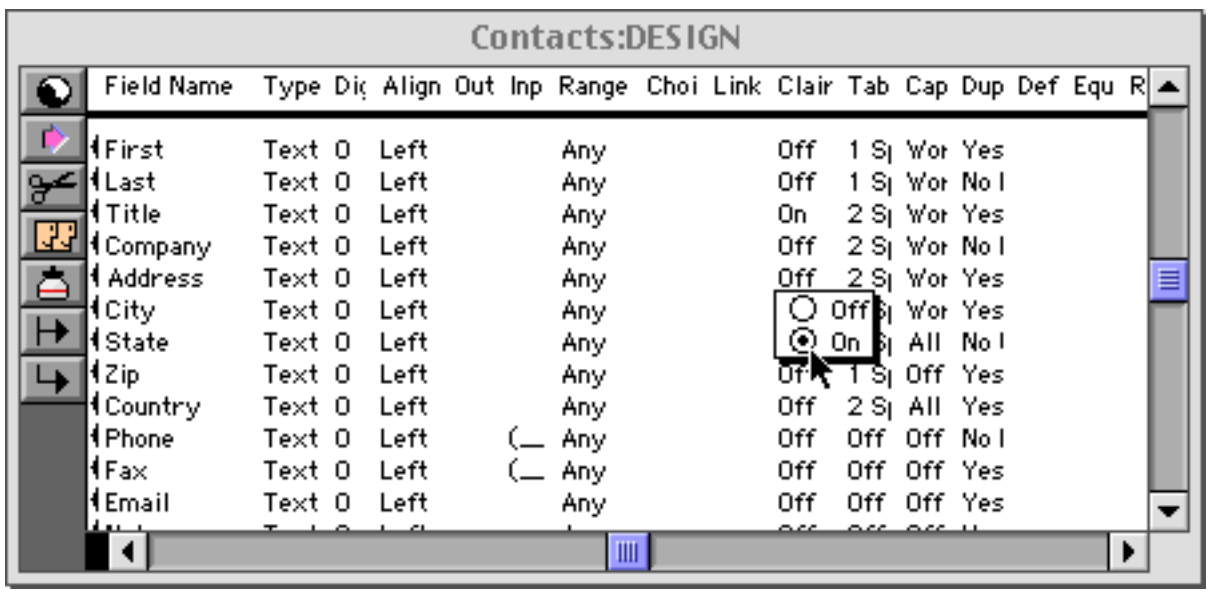


Of course, Clairvoyance can only be helpful when you are repeating a word or phrase that is already in the database. If you are entering a new word or phrase, Clairvoyance cannot help you—but it won't get in your way, either. As you type in a new word or phrase Clairvoyance may guess that you are entering an old word or phrase. Just keep typing, and Clairvoyance will automatically erase its guess when it no longer matches what you have typed.



Turning Clairvoyance® On or Off

Clairvoyance can be turned on or off with the **Field Properties** dialog box (Setup Menu) or with the Clairvoyance column in the design sheet.



Clairvoyance® Helps Insure Data Consistency

One problem when building large databases is making sure that information always gets entered the same way, especially when more than one person is keying in the data. For example, a single company could be entered in your inventory database many ways—

```
Fuji
Fuji, Inc
Fuji USA
Fuji Photo, Inc
Fuji Photo Film USA
Fuji USA, Inc.
```

Clairvoyance helps solve this problem by accurately repeating the information time after time. You may find that Clairvoyance's ability to insure data consistency is more important than the keystroke savings.

Using Clairvoyance® With Dates

We do not recommend using Clairvoyance with date fields. Although Clairvoyance will work with dates, it is not very useful since dates only contain up to six characters anyway. Using Clairvoyance with a date field can be annoying, because scanning a date field takes much longer than for regular text fields. This can make the data entry seem somewhat sluggish.

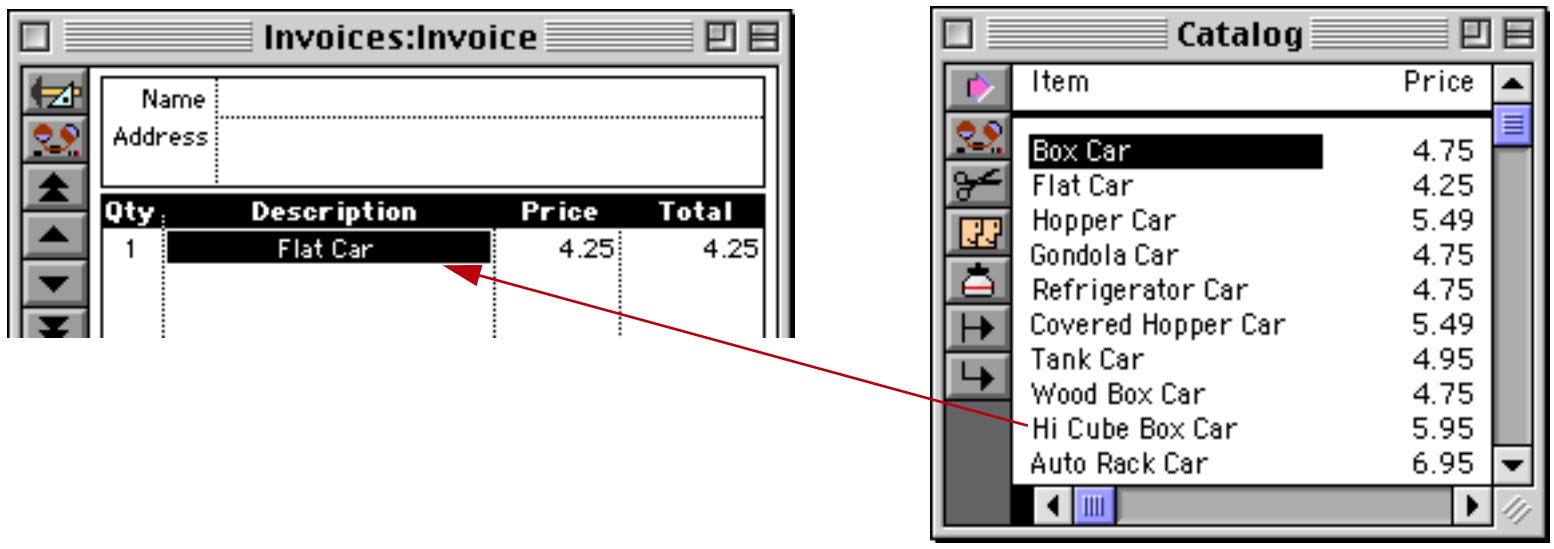
Clairvoyance® Across Multiple Files

Clairvoyance normally works only with the data you have already typed into the current database. But if you have already typed this information into another database, you can specify that Clairvoyance should be linked to that database instead of the current one. For example, you could link an invoice with your price list, allowing you to type in only a few letters to bring up a product description. This is called linking clairvoyance to another field.

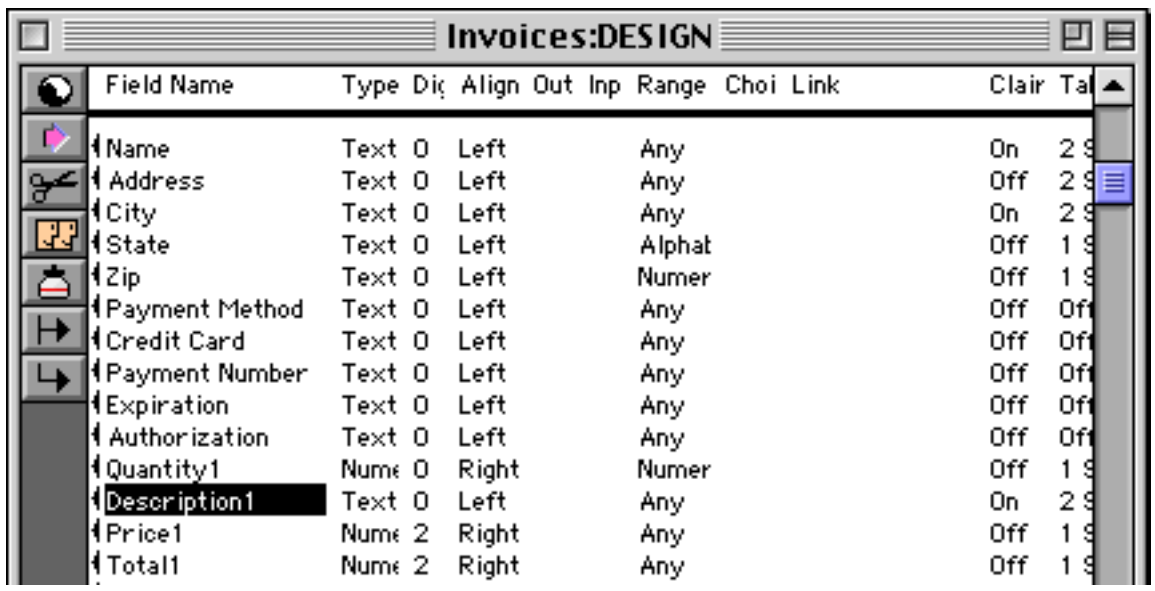
There are two reasons for linking clairvoyance to another field. Clairvoyance cannot anticipate values until they have been typed in at least once. If all the possible values have already been entered into another database, Clairvoyance can start working immediately by looking into the other database.

Another advantage is speed. If your price list contains 200 records and your invoice database contains 2000 records, Clairvoyance can scan the price list 10 times faster. As your database gets larger, this speed difference may become noticeable.

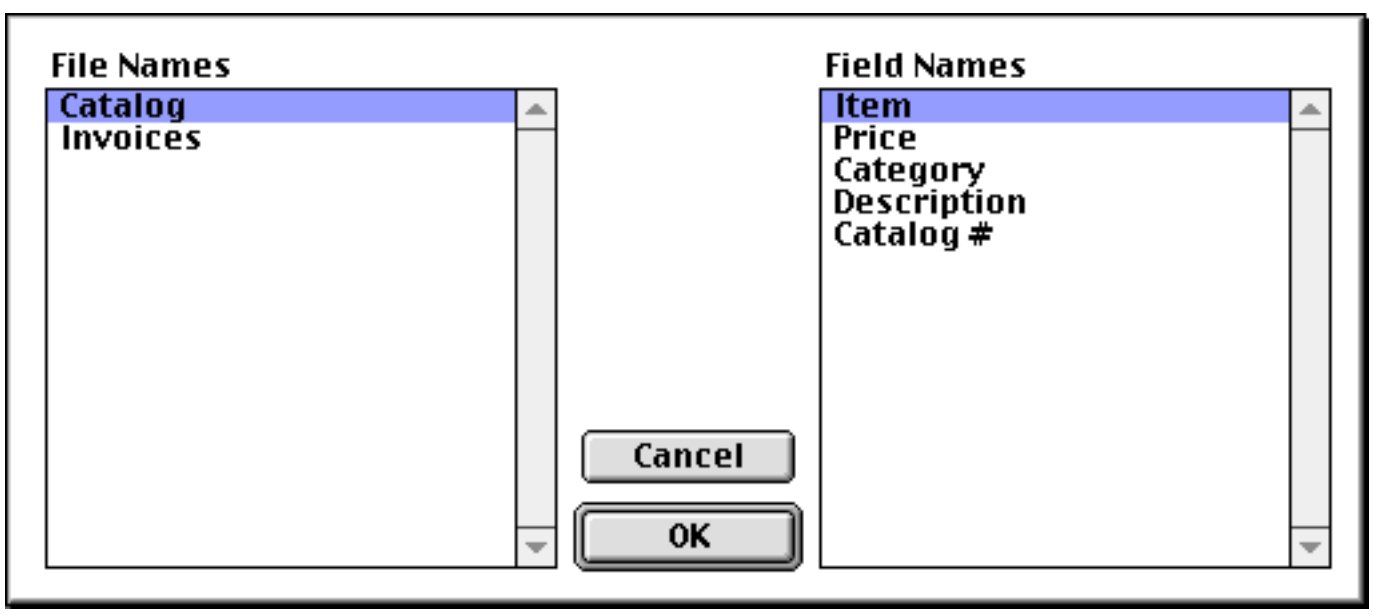
To illustrate setting up a clairvoyance link to another field we'll use the invoice and catalog databases shown below. We'll link the Item field in the Catalog database to the Description field in the Invoices database.



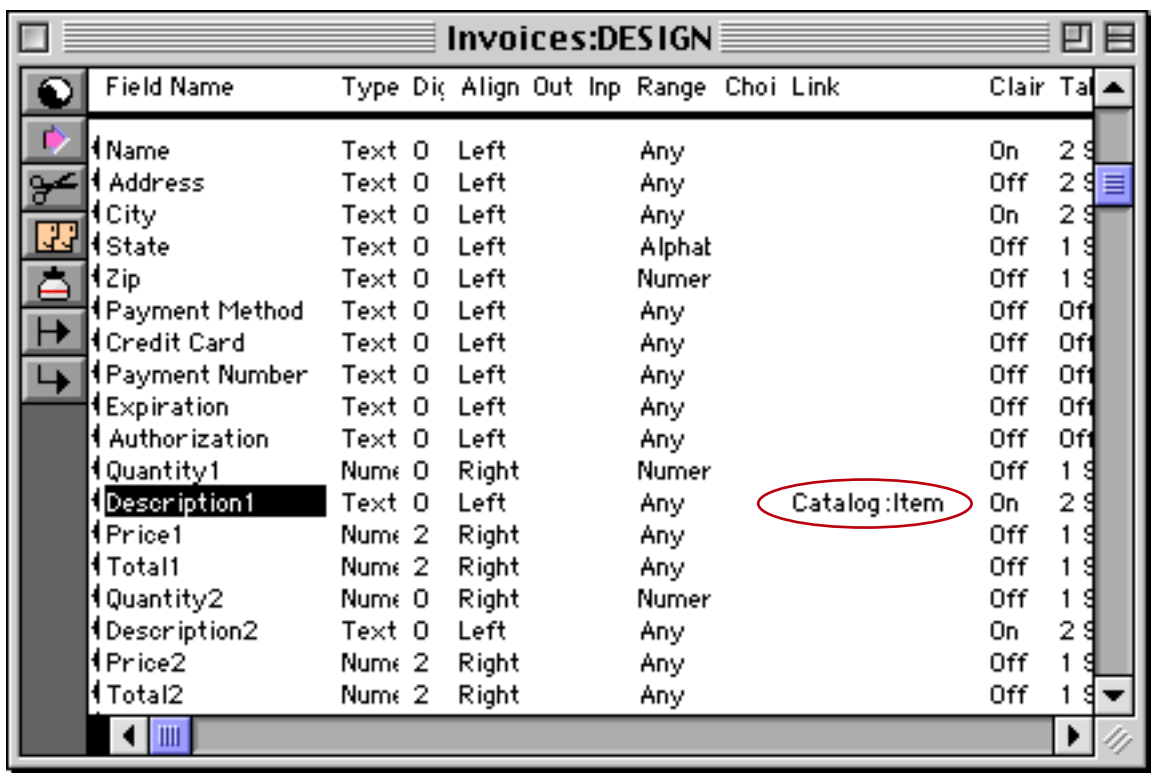
To set up the link open the Invoices design sheet. Click on the name of the field you want to set up, in this case Description1.



Once the field is selected choose **Set Up Link** from the Special menu, which opens the dialog shown below. On the left hand side of the dialog is a list of open databases. Select the database containing the field you want to link to, in this case Catalog. Once the database is selected a list of fields in that database appears on the right. Select the field you want to link to, in this case Item.

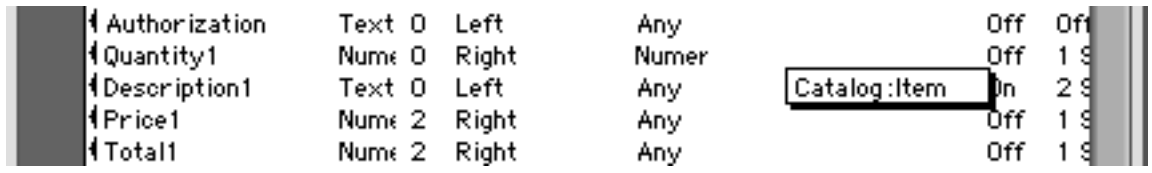


Once the database and field are selected press **OK** to enter the link into the design sheet. You can see the new link definition in the Link column.

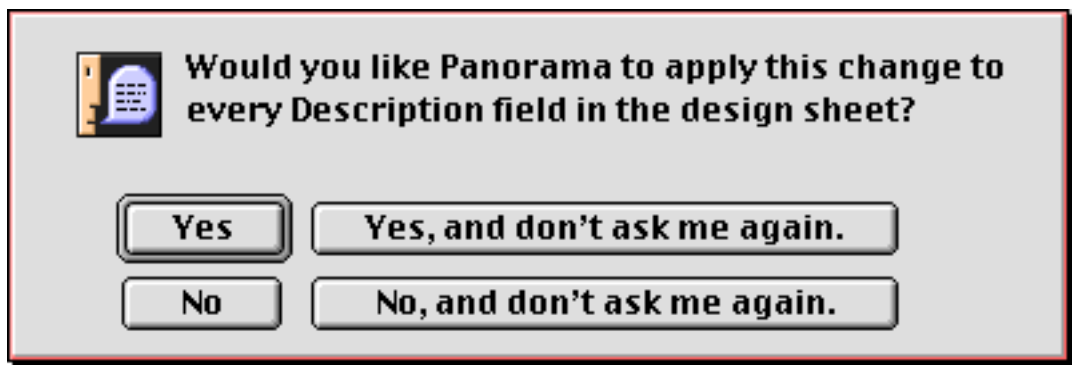


As you can see the link definition is simply the name of the database followed by a colon (:) followed by the field name. You can type in a link definition manually if you want instead of using the **Set Up Link** dialog.

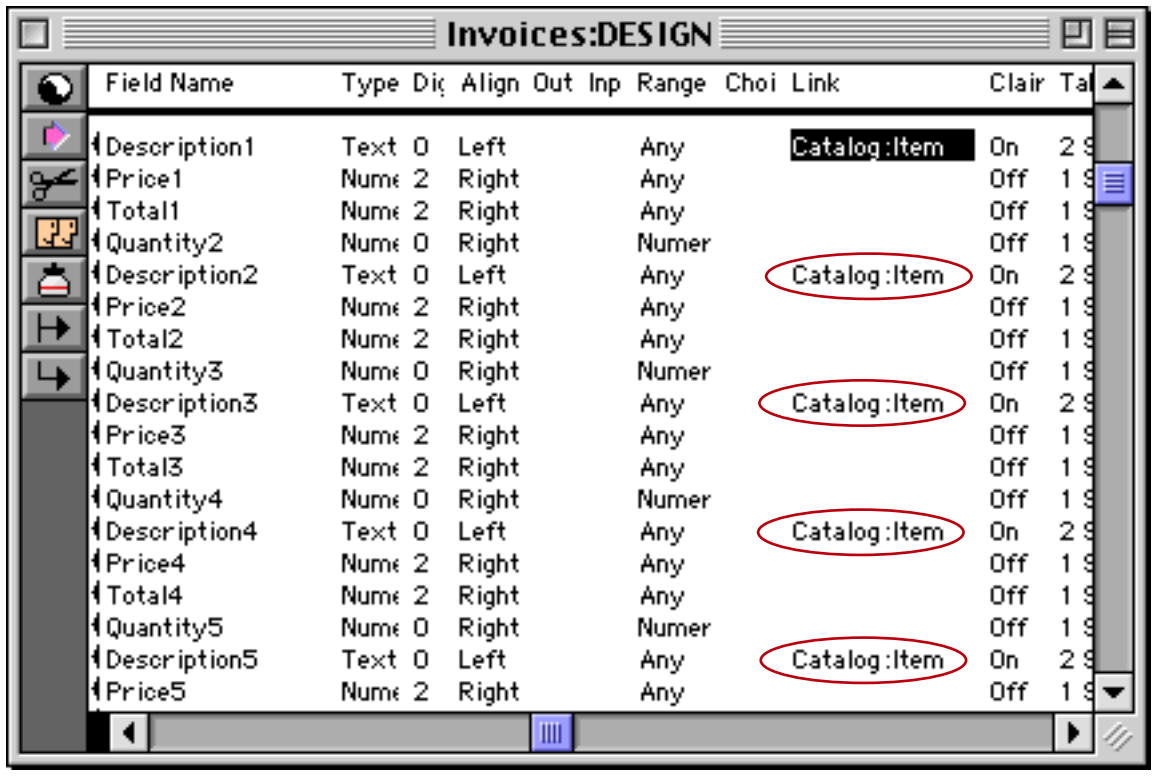
In this example Description1 is a line item field (see “Repeating Fields (Line Items)” on page 262) you will probably want to set up the same link for all of the other line items (Description2, Description3, etc.). The easy way to do this is to simply double click on the Link cell, like this.



When you press **Enter** Panorama will ask you if you want to copy this data to all of the other line items (see “Modifying Line Item Fields” on page 265).

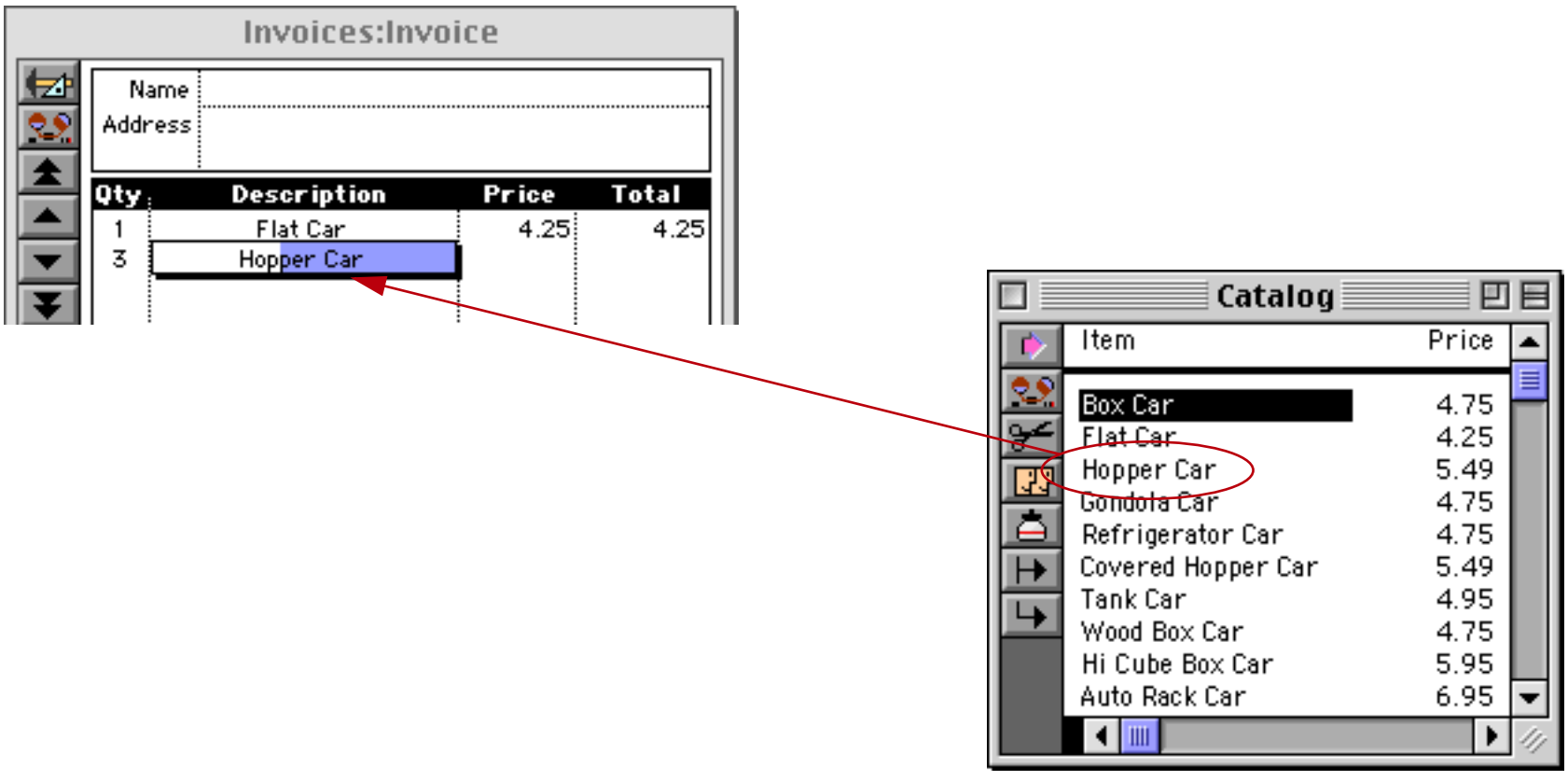


Press **Yes** and Panorama will make the changes.



Like all other design sheet options, the link does not actually take effect until you tell Panorama to create a new generation (see “Database “Generations”” on page 252).

When you are editing data within a field that has a clairvoyance link set up, Clairvoyance checks the characters you type against the data in the second database. When it finds a possible match, it enters the rest of the value for you.

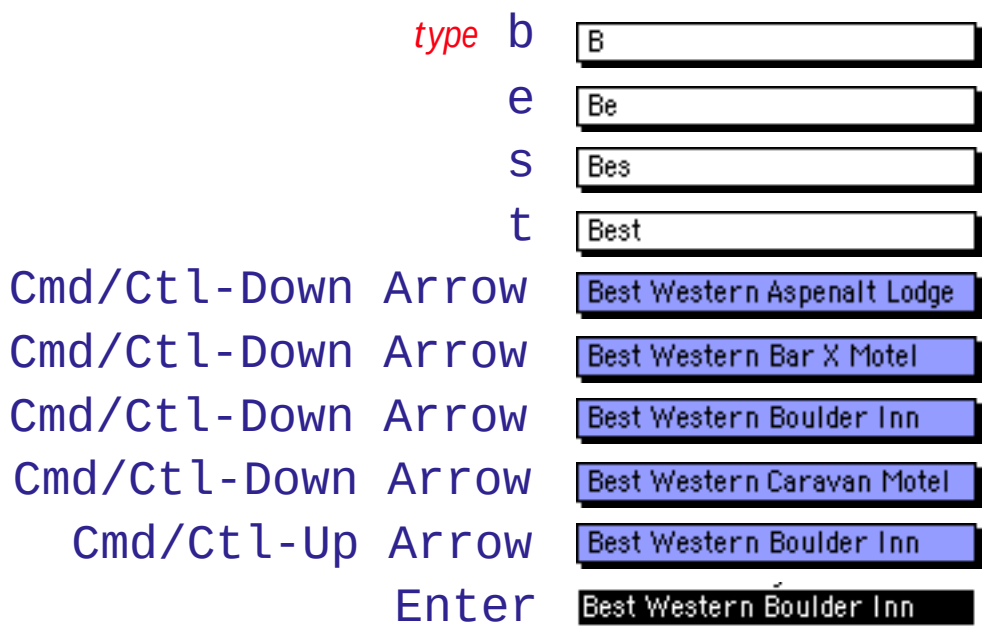


In this example Clairvoyance has been linked to a field in another database. However you can also link Clairvoyance to a field in the same database.

Clairrows

When you hold down the **Command** key (Mac) or **Control** key (Windows), the up and down arrows on the keyboard become clairvoyant arrows, or "**clairrows**." With the key held down you can use the arrows to scan through the values that are already in the database. Each time you press **Command/Control-Down Arrow** the next value appears, while each time you press **Command/Control-Up Arrow** the previous value appears. You can scan through the values until you find the information you are looking for, then press the **Enter** key to enter the value.

To give the clairrows a head start you can type in the first few letters of the information you are looking for. For example, suppose that you are looking through a travel database for a particular Best Western Hotel. Start by typing **Best**, then press **Command/Control-Down Arrow**. The first hotel with a name beginning with Best will appear. Each time you press **Command/Control-Down Arrow** the name of next hotel (alphabetically) will appear—for example Best Western Aspenalt Lodge, Best Western Bar X Motel, Best Western Boulder Inn, etc. Press **Command/Control-Up Arrow** to move backwards through the hotel names. Continue until the hotel you are looking for appears, then press **Enter**.



This technique also works when Clairvoyance has been linked to a field in another database (see "Clairvoyance® Across Multiple Files" on page 323).

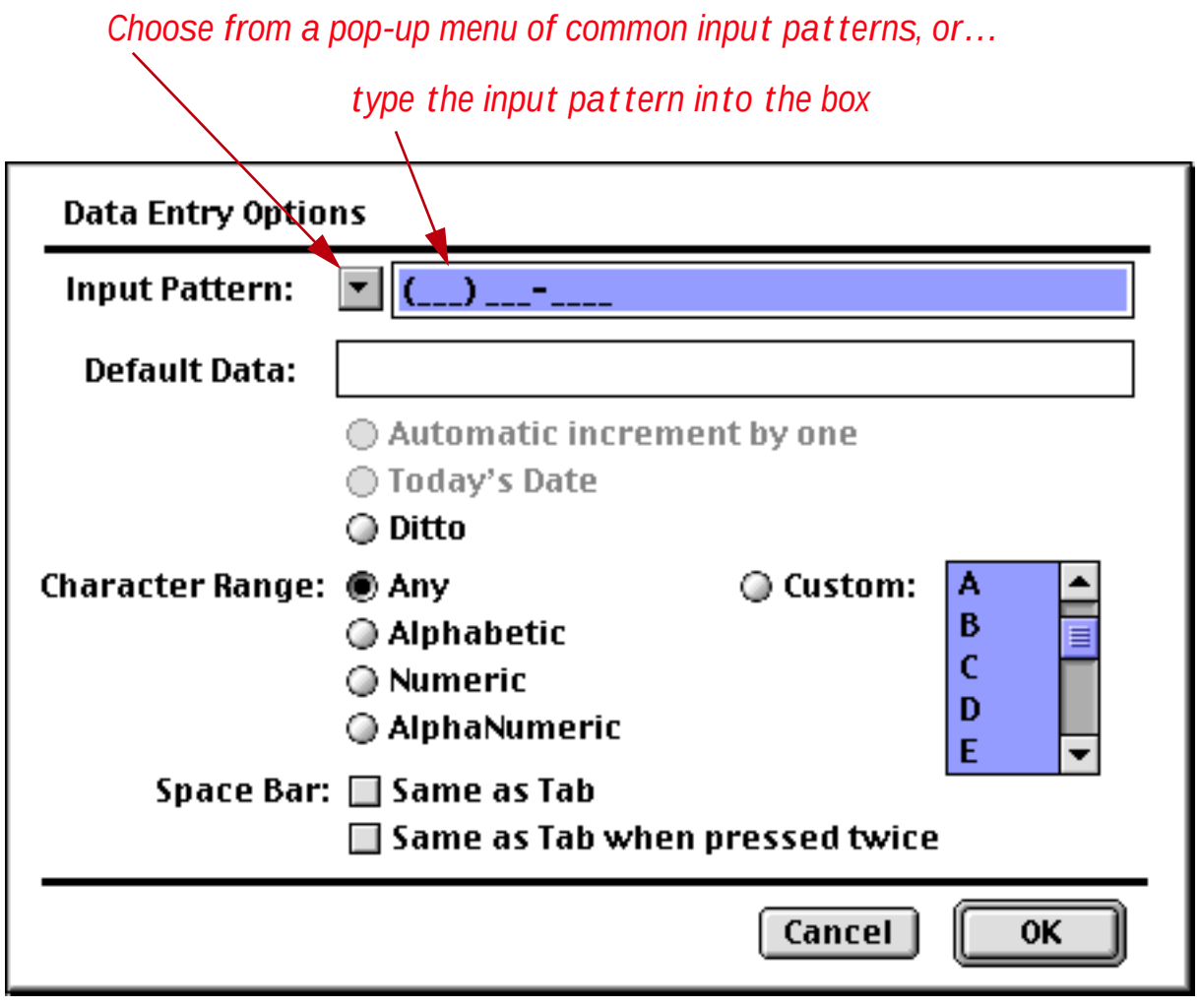
Input Patterns

Sometimes you may wish to force the data being entered into a specific pattern. For instance in the United States and Canada long distance phone numbers almost always use the pattern (999) 999-9999. Panorama's **Input Pattern** can take care of entering the pattern for you. Once the pattern is set up, you only type the actual data (in this case the digits of the phone number). Panorama combines the data you enter with the pattern to produce the actual data. For example, combining the input pattern **(_ _ _) _ _ _-_ _ _ _** with **3124562468** produces the data (312) 456-2468.

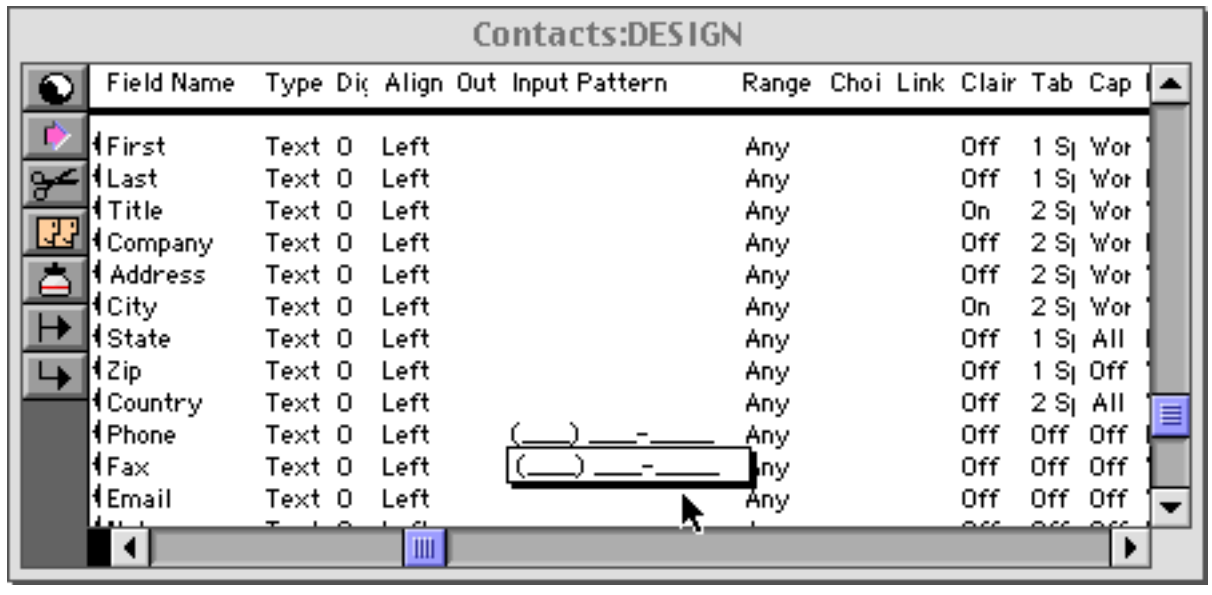
An input pattern consists of a string of characters with an underscore in each spot where actual data will be entered. The input pattern is just like fill in the blanks, but instead of filling in the blanks you fill in the underscores. (Press Shift-Dash to enter the underscore character. The dash key is in the top row of the keyboard, just to the right of the 0 key.) The table below lists some common input patterns.

Type of Data	Input Pattern	Example
Phone Number	(_ _ _) _ _ _-_ _ _	(312) 456-2469
Social Security Number	_ _ _ - _ _ - _ _ _ _	234-54-5476
License Plate	_ _ _ _ _ _	AGB 287
Date	_ _ / _ _ / _ _	03/24/05
Time	_ _ : _ _ : _ _ _ _	11:24:36 PM

You can set up the input pattern with the Field Properties Data Entry Options sub-dialog (open the **Field Properties** dialog and press the **Data Entry** button).



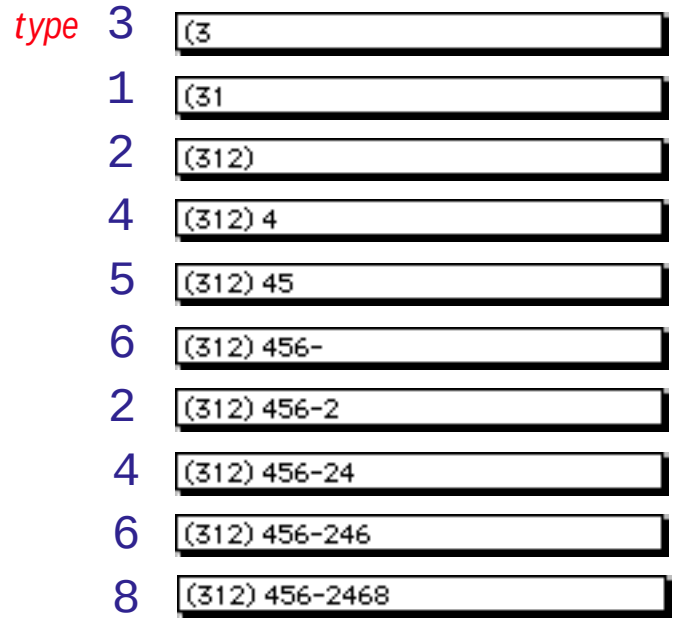
You can also set up the pattern with the Input Pattern column of the design sheet. (See "The Design Sheet" on page 252 if you are not already familiar with using the design sheet.)



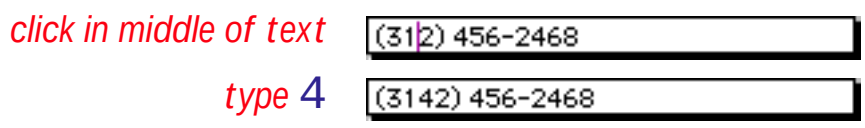
Tip: Input patterns should not be used with numeric fields. If you want to add a pattern to a numeric value, you should use an output pattern (See "Numeric Output Patterns" on page 290).

Entering Data with an Input Pattern

This illustration shows an example of entering a phone number with an input pattern.



The input pattern is only active when you are adding new characters at the end of the text. It does not adjust the data when you are inserting text in the middle of the cell. For example, it does not prevent you from creating a four digit area code, like this:



Using Input Patterns with Dates

The purpose of input patterns is to save keystrokes in data entry by inserting constantly occurring dashes, colons, parentheses, or other punctuation. When it comes to date fields, you must decide how you like to enter dates. Using the input pattern _ _ /_ _ /_ _ removes the need to type /'s (or some other separator) between the month, day and year. However, using the pattern requires that you type leading zeros in front of

single digit months and days. For instance, to enter January 1st you must type **0101**. Without the pattern you can enter a single digit, for example **1/1**. (Keep in mind that Panorama allows any non-numeric character as the separator, so you could also type **1.1** on a numeric keypad—very fast.) It's up to you which method you prefer.

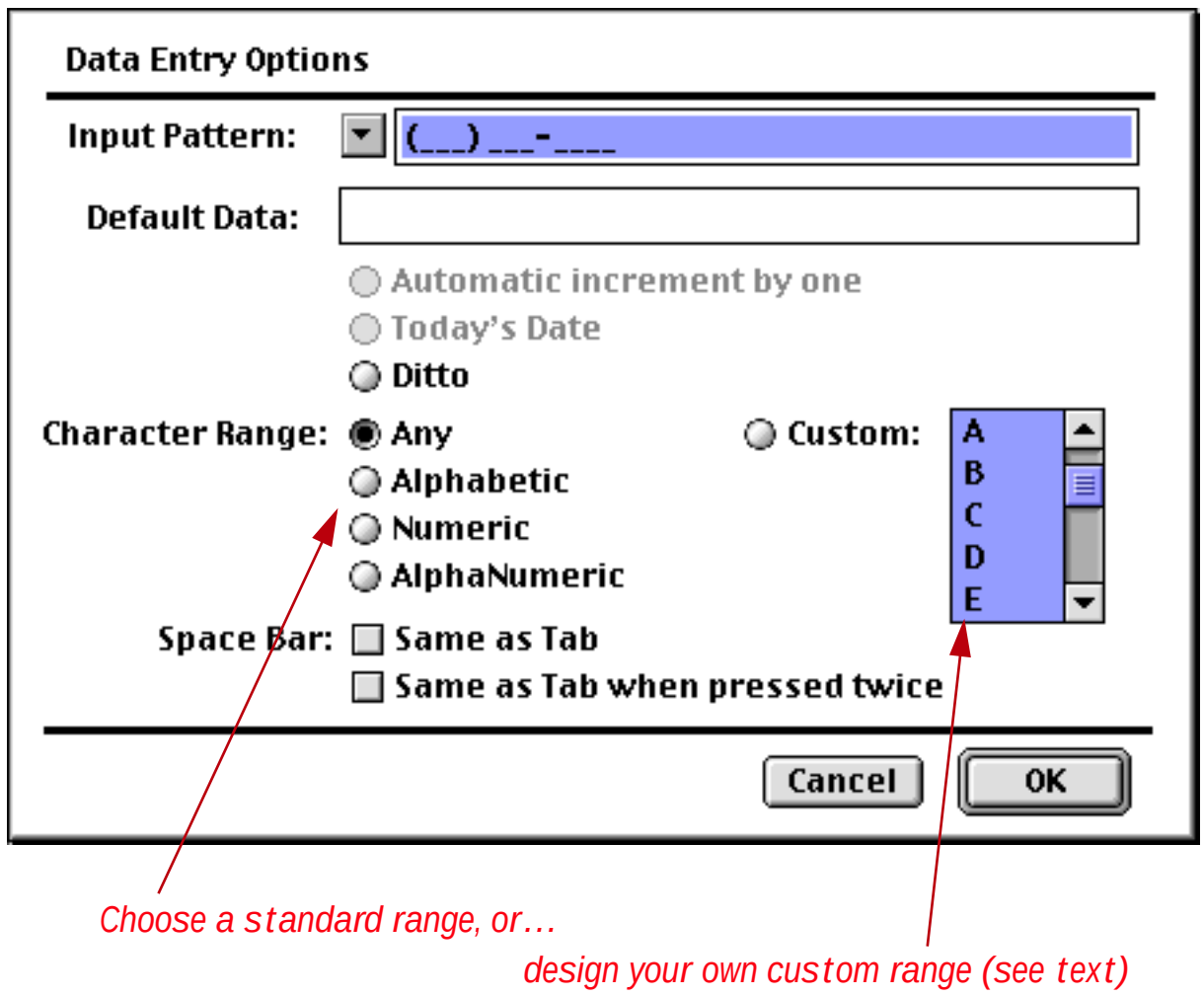
One other point to keep in mind—the input pattern can interfere with Panorama's Smart Date feature ("Entering Dates" on page 294). For example, with a pattern if you attempt to enter **yesterday** you will get ye/st/erday, and if you enter **tuesday** you will get tu/es/day. You can go back and edit out the /'s, but if you are going to use Smart Dates frequently you might want to forego the input pattern.

An input pattern can be used to override Panorama's century rounding feature. If you want to enter all dates in the 20th century you can use the pattern __/__/19__. If you are using this pattern then Panorama will treat **030423** as 3/4/1923, not 3/4/2023. (Remember, Panorama normally rounds the year to the nearest century (within 50 years) if you do not specify all four digits of the year.)

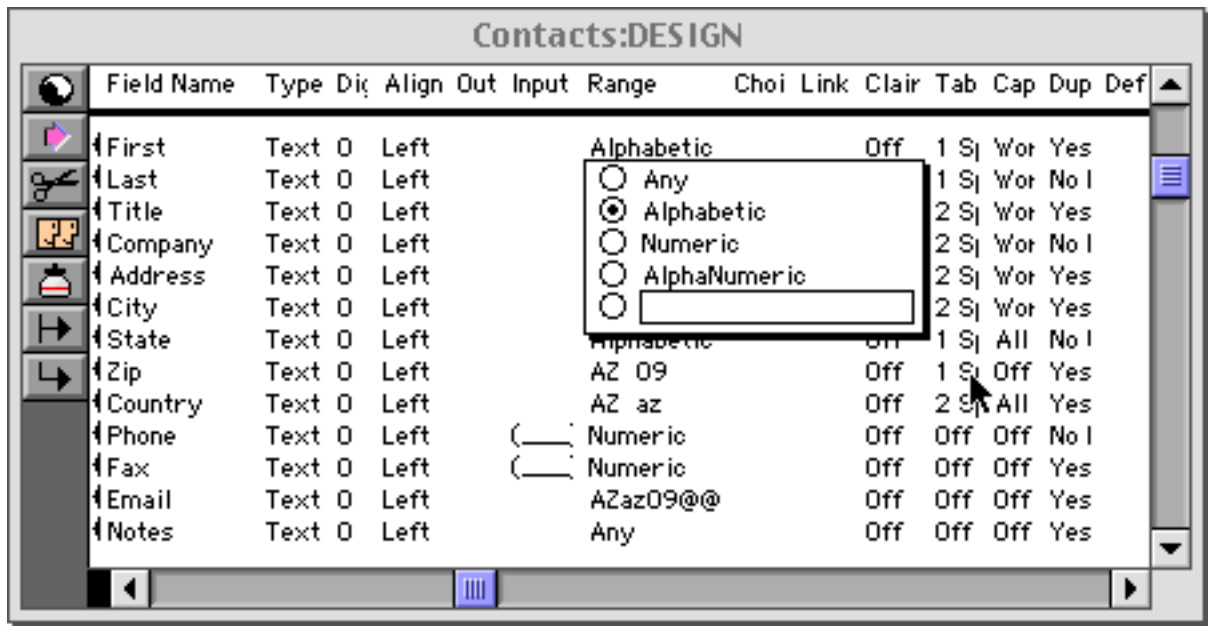
Restricting Character Types

Panorama normally allows you to enter any character that can be typed from the keyboard. If necessary you can restrict the kinds of characters that can be entered into each field. Panorama has five different character restriction levels—Any, Alphabetic, Numeric, AlphaNumeric, and Custom.

You can set up character entry restrictions with the Field Properties Data Entry Options sub-dialog (open the **Field Properties** dialog and press the **Data Entry** button).



You can also set up the character range with the Range column of the design sheet. (See "The Design Sheet" on page 252 if you are not already familiar with using the design sheet.)



The **Any** "restriction" really isn't a restriction at all—it allows any kind of text—letters, numbers, spaces, or punctuation. Panorama lets you type in anything you want with no restrictions. This is the default option.

The **Alphabetic** restriction allows only letters (A-Z and a-z) and spaces. The letters may be either upper or lower case. If you attempt to type in a non-alphabetic character (a number, for example) Panorama will beep and ignore the character.

The **Numeric** restriction allows only digits (0…9), periods, minus sign, and the letter E (for scientific notation).

The **AlphaNumeric** restriction allows both letters and numbers, as well as spaces.

Custom Character Restrictions

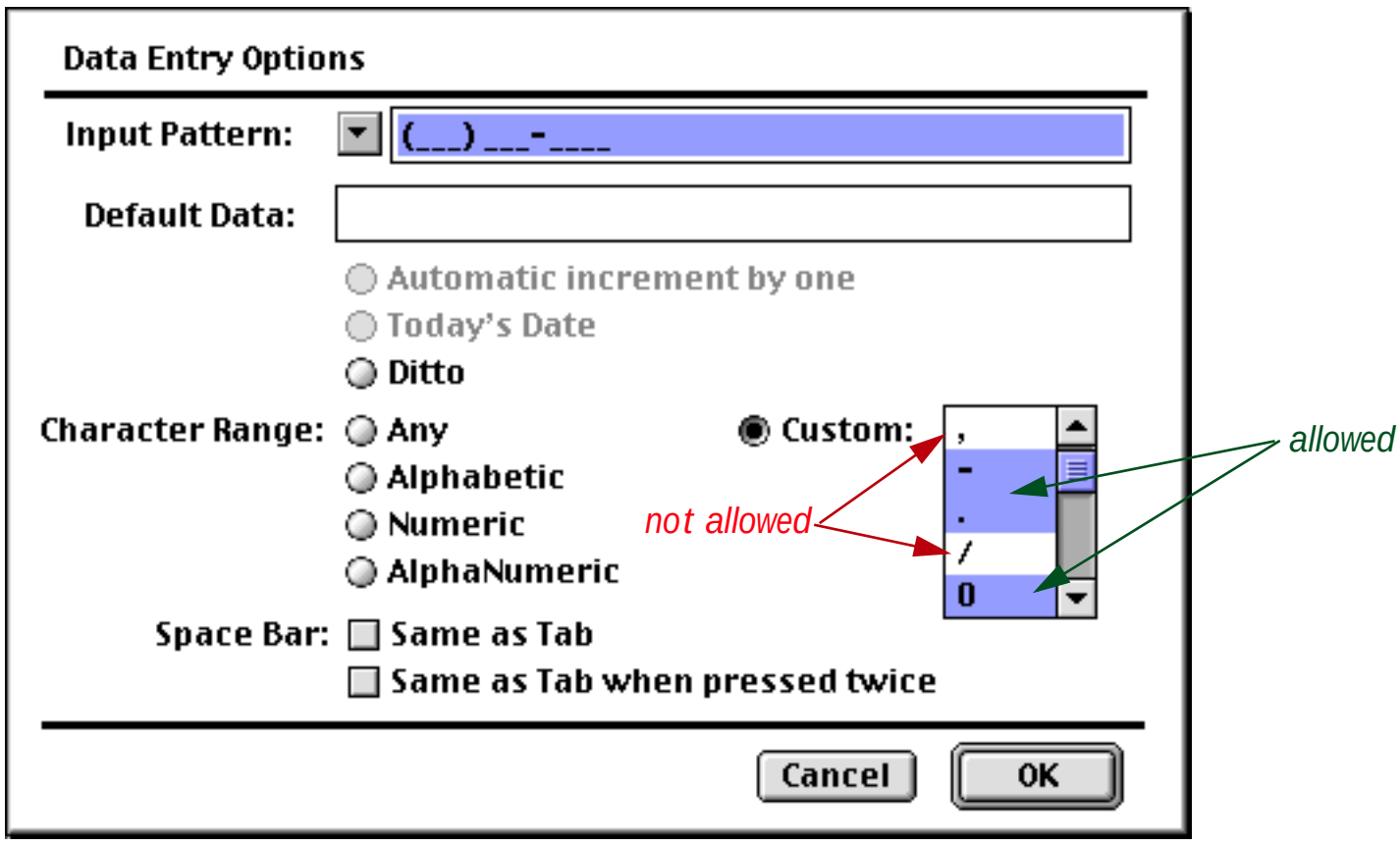
The **Custom** option allows you to exactly specify the characters you want to allow and disallow in this field. The actual characters allowed are defined by one or more pairs of characters. Each pair specifies a range of characters that are allowed. For example the pair 09 would allow all characters in the range 0…9, while the pair az would allow all lower case letters. You can combine several pairs to create a more complex range, for example az09 for all lower case letters or numbers. A pair may specify a single character as both the beginning and end of the range, for instance %% (only the percent symbol allowed), or 09%% (numerals and the percent symbol, but not the decimal point). If you wish to allow spaces, one of the pairs should be a pair of spaces, for instance AZ az09. To preview the effect of a character range you can use the ASCII Chart wizard — see "Showing Character Ranges with the ASCII Wizard" on page 1234.

The table below shows some common examples of custom character restrictions. For each range a sample of Ok data and bad data is shown, with the disallowed characters shown in red.

Custom Range	Ok Data	Bad Data	Comments
09	936	923.77	Only digits, no decimal point, spaces or other punctuation allowed.
09..	156.23	1,294.48	Basic fixed point numbers
09..%%	67.82%	67.82 %	Percentages (no spaces)
09//	5/23/02	March 1st	Numeric format dates
09::	1:24:83	1:24 PM	Time
09 ::AAaaMMmmPPpp	5:32 PM	5:32 DL	Time (am/pm)

Custom Range	Ok Data	Bad Data	Comments
AZ	SEATTLE	Seattle	Upper case letters only—no punctuation or lower case letters
Azaz	John	John Smith	Letters but no spaces
AZaz@@..	sue@my.net	sue$my.net	Handy for email addresses
09 (())--	(213) 444-1234	342-3982 ext 12	Basic US phone numbers
!~	Check#	El Niño	Everything ok except spaces, international characters and special symbols
!	Niño	El Niño	Everything ok except spaces

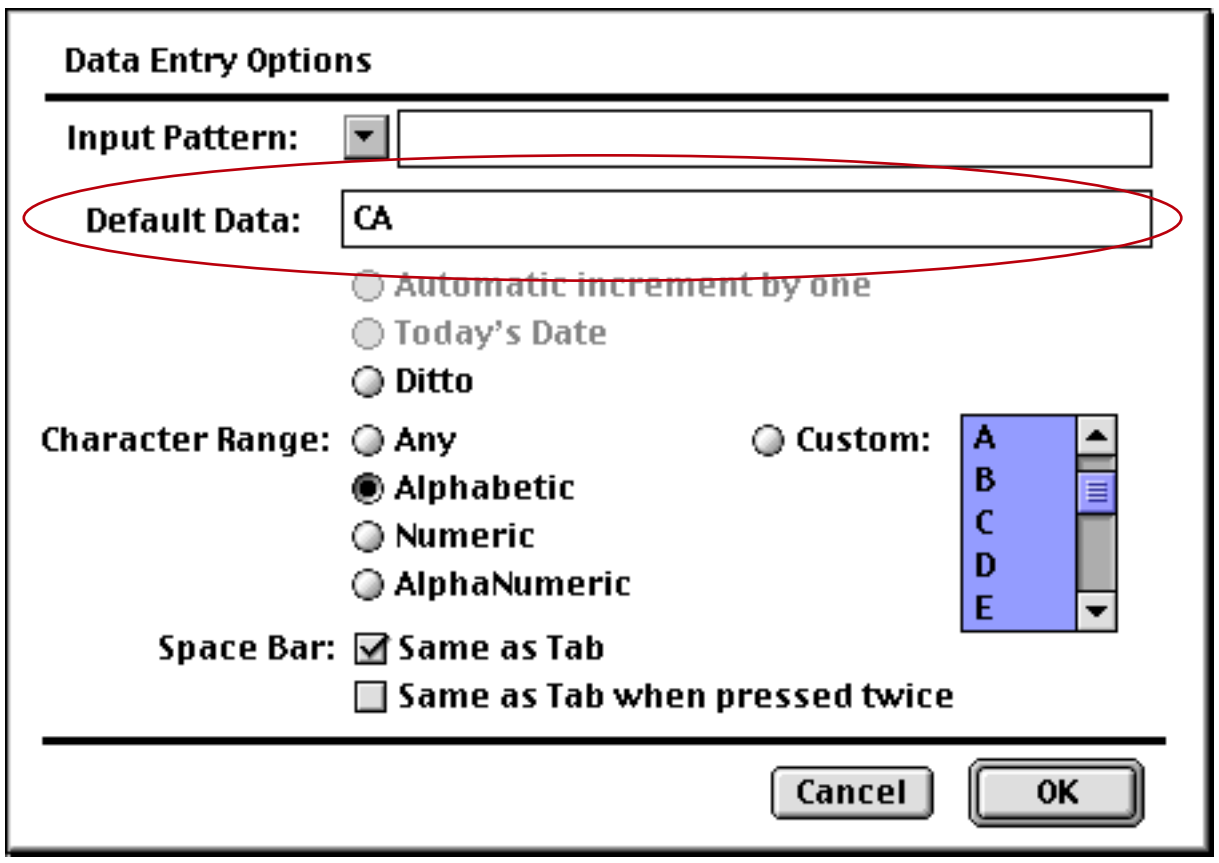
If you are using the **Field Properties** Data Entry Options sub-dialog, you can set up a custom restriction by selecting characters from the scrolling list. Only selected characters will be allowed. This dialog will automatically set up the character pairs as specified in the last paragraph.



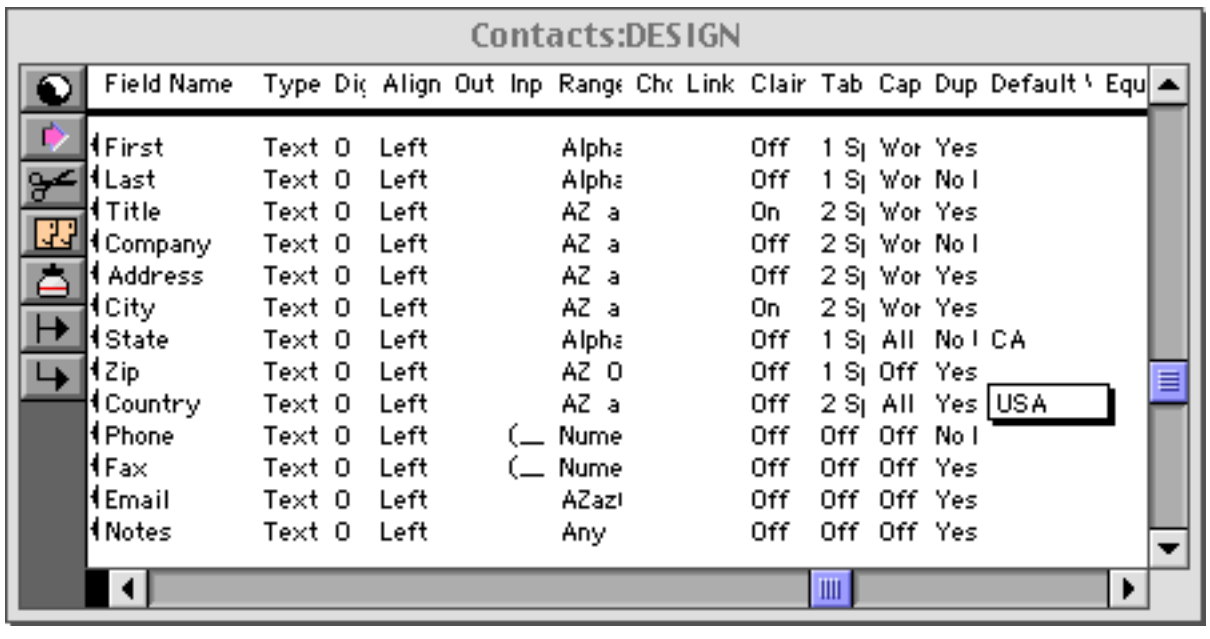
When you press the **OK** button Panorama will convert the list into a series of character ranges as described in the previous section. If you want to see what the custom range you have created looks like, open the design sheet.

Default Values

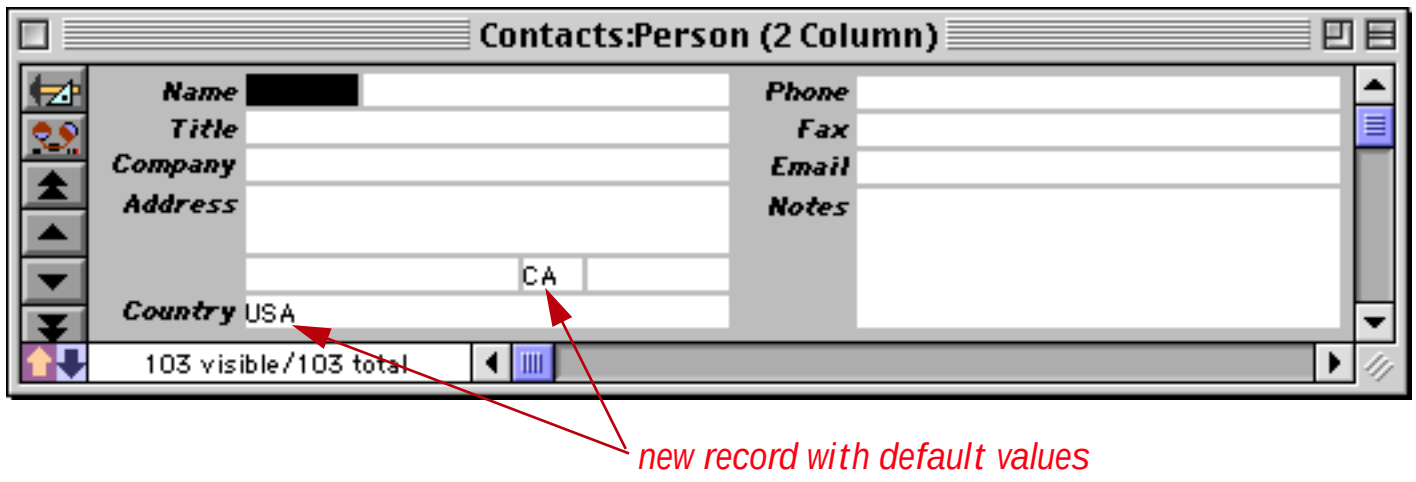
When a new record is added to a database, it is usually completely empty. You can, however, set up a default value for each field. One way to set up default values is with the Field Properties Data Entry Options sub-dialog (open the **Field Properties** dialog and press the **Data Entry** button). The dialog below is for a State field which defaults to CA (California).



You can also set up the default with the Value column of the design sheet. (See "The Design Sheet" on page 252 if you are not already familiar with using the design sheet.)

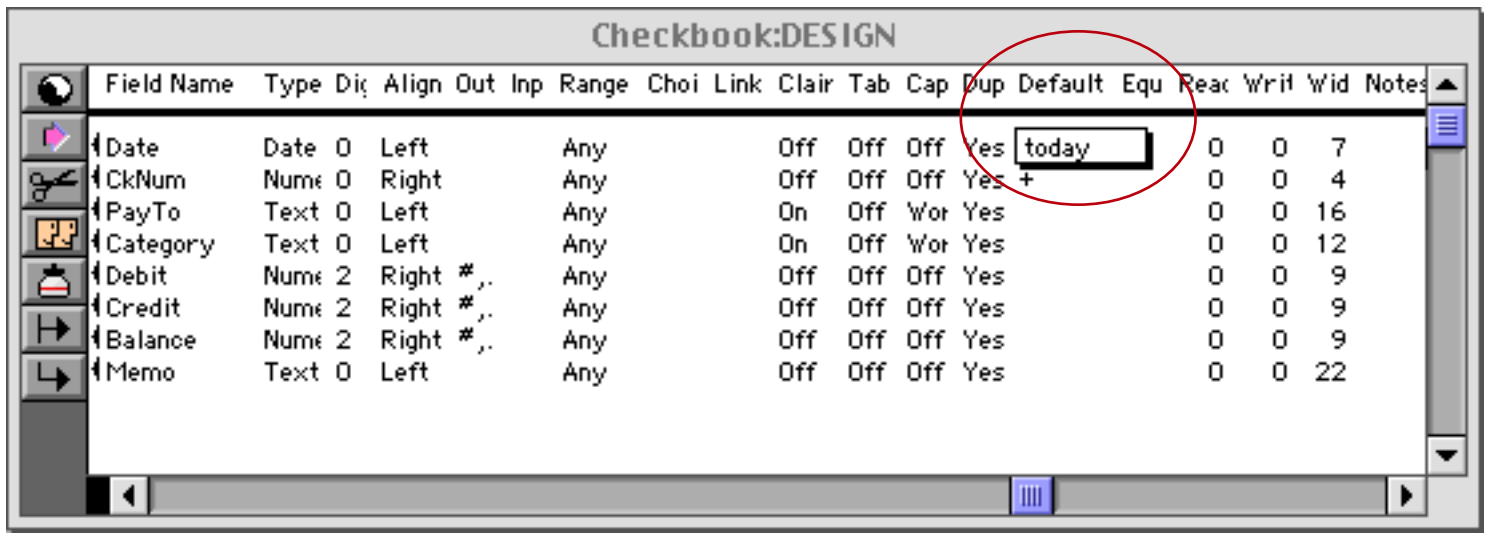


The simplest default is a fixed value, as shown in the example above. For example you might want the Country field to default to your home country, a shipping field to default to your preferred shipper. Once defaults are set up, they are automatically entered whenever a new record is created.

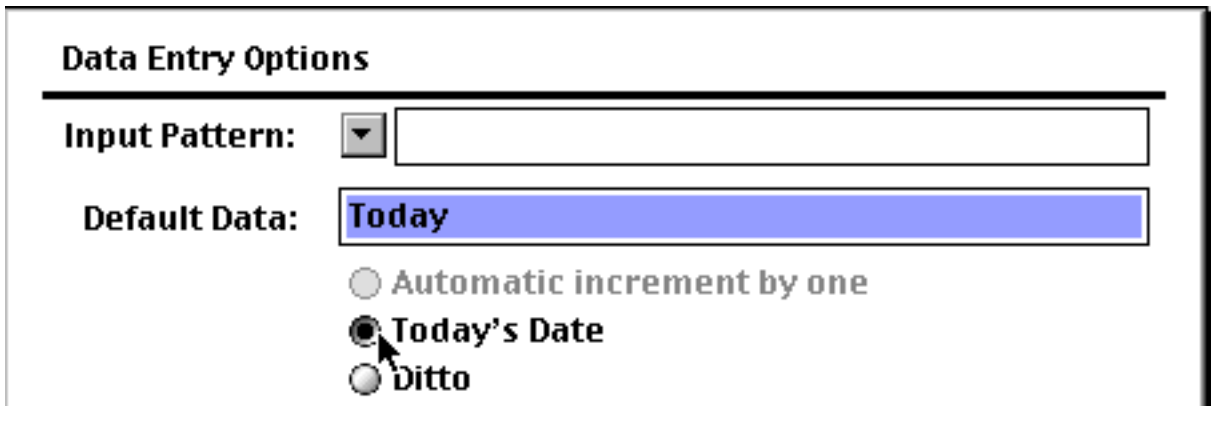


Default to Today's Date

To default to today's date use the default value today.



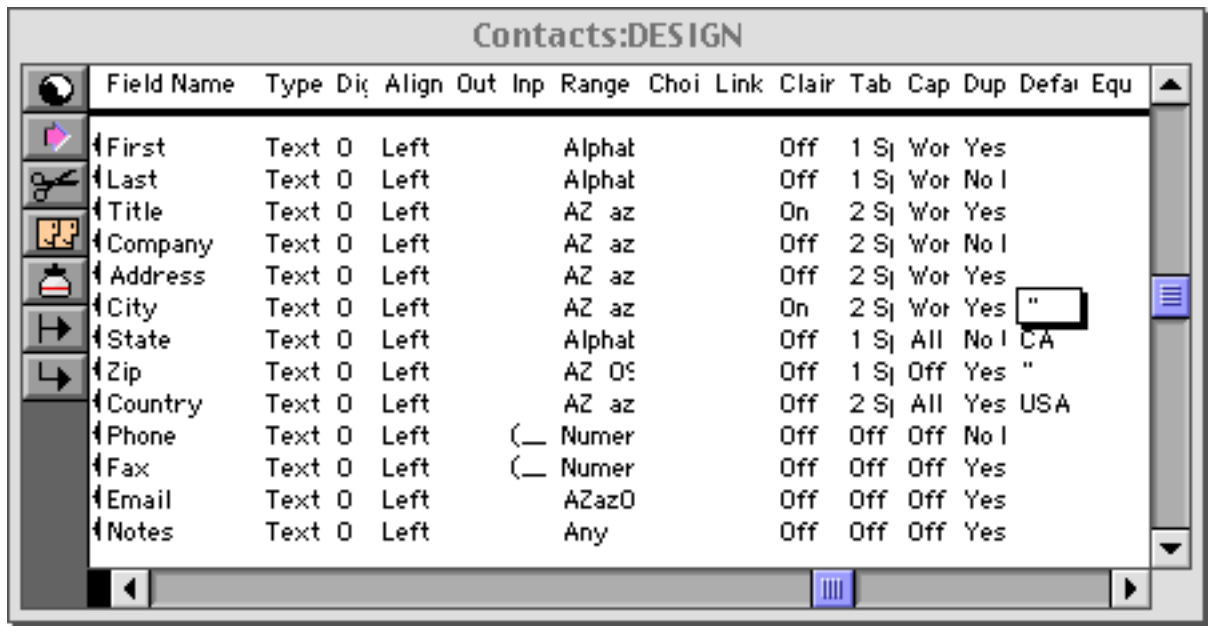
If you are using the Data Entry Options dialog, you can simply click on the Today's Date radio button.



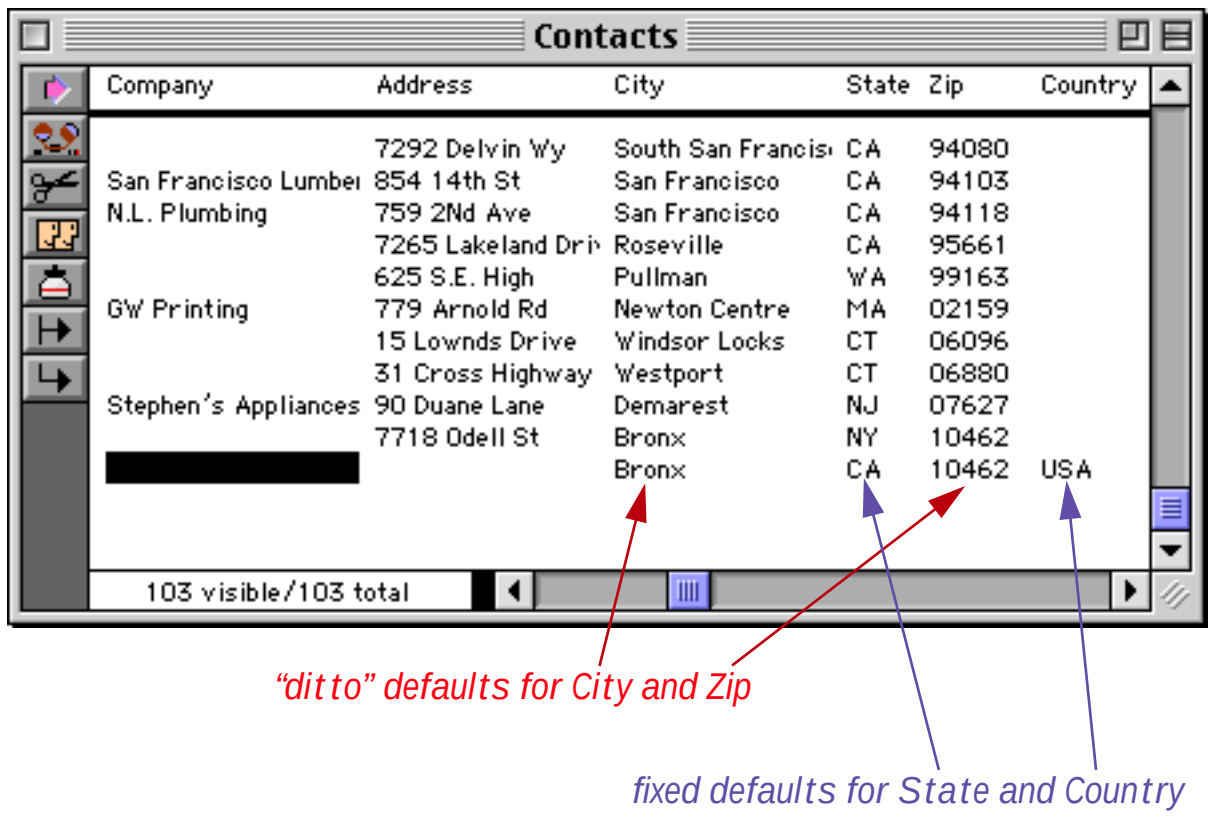
Note: The today default only works for fields that use the date data type. If you use the default value today with a text field you will simply get the word today. See "Dates" on page 294 for more information on the date data type.

"Ditto" Defaults Based on the Previous Record

Instead of being fixed, a default value can be based on the data in the previous record. You can produce this type of "**ditto**" default by using the default value ". This is the quote character, which is produced by holding down the **Shift** key and pressing the " key (just to the right of the **semicolon** key). (Some people mistakenly call this the double-quote character.)

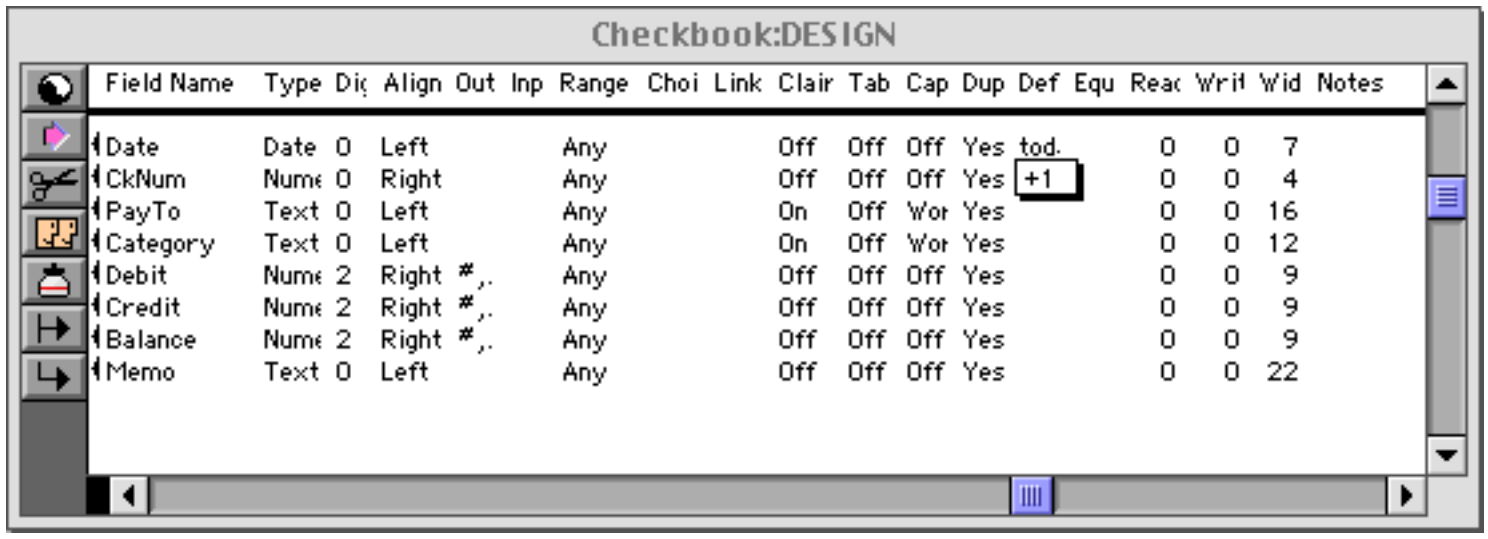


When you create a new record, the fields using the ditto default will contain the same values as the previous record. In this illustration a new record has been added with four default values, two fixed and two "ditto."



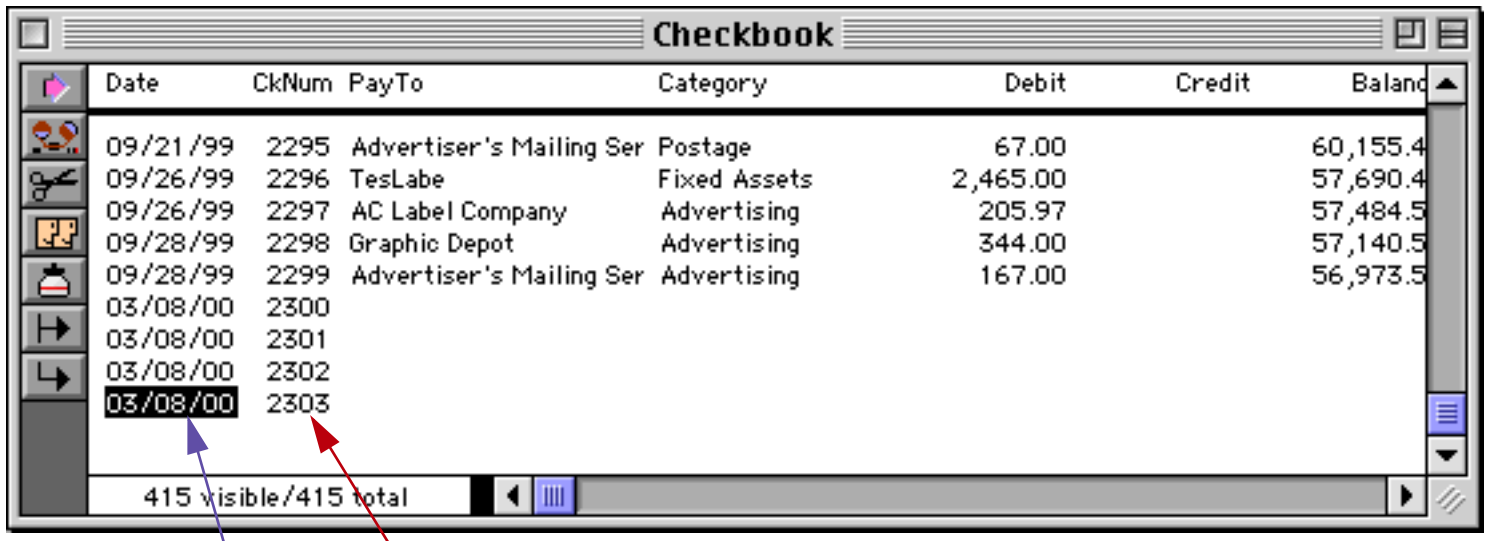
Automatically Incrementing Defaults (1, 2, 3, ...) Based on the Previous Record

For a numeric field you can specify a default that is created by adding to the previous value in the field. To do this, use a default of +nn, where nn is the amount to add to the previous value. For example +1 causes the value to increment by one for each new record.



You can use any number, even a negative number like +-5. This default would cause Panorama to add negative 5 (same as subtracting 5) to the value each time a new record is created. If the numeric type allows it, you can even use non-integer values like 2.5 or 0.1.

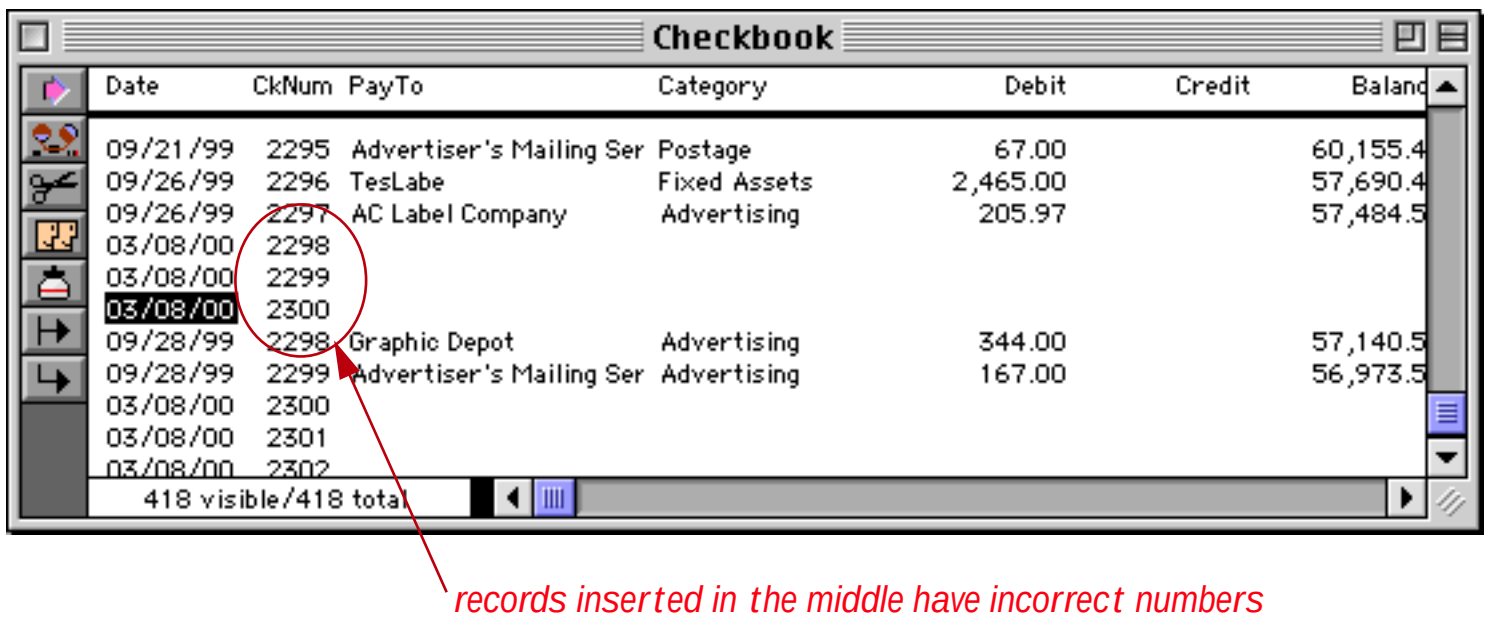
As new records are added to the database, they are numbered automatically, like this.



automatically incrementing check numbers

this field set up to default to today's date

Be sure to keep in mind that an incrementing default like +1 is based on the previous record, not on the largest value in the entire database. So if you insert a record in the middle of the database, the incremented value will be based on the value just above it, not on the value at the end of the database.

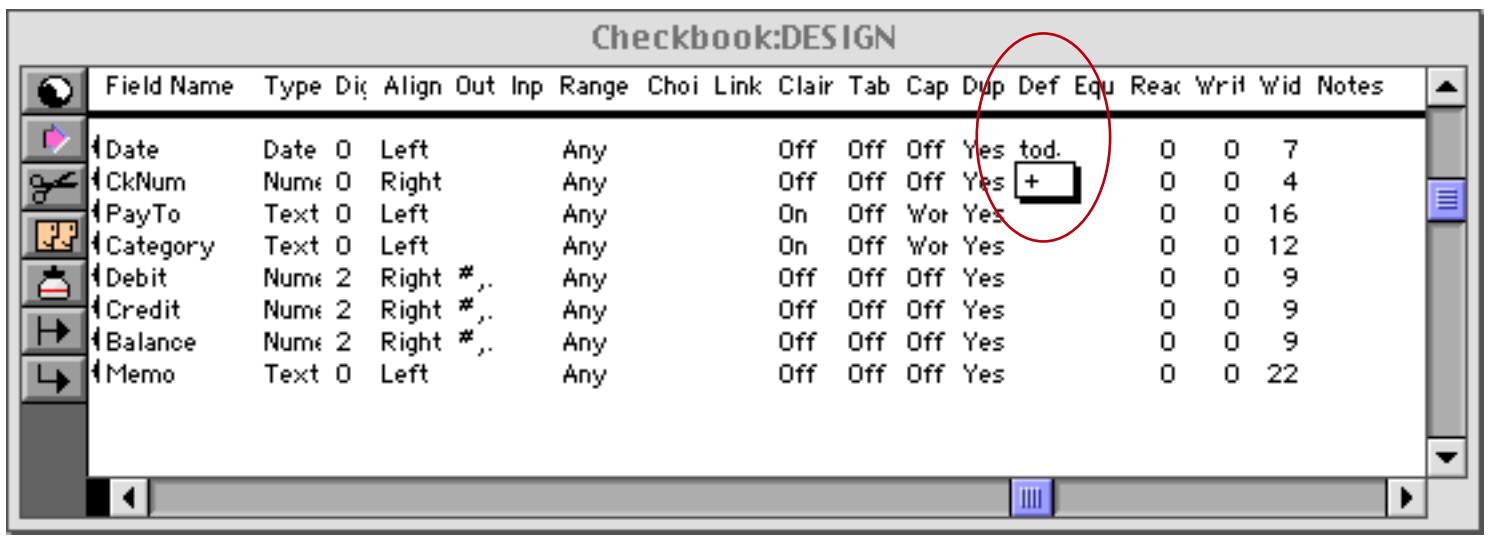


If you want to generate a unique incrementing number for use as a record ID (for instance an invoice number or check number), use the technique described in the next section.

Creating a Unique Record Number

Many databases applications require that each record contain a unique number that can be used to identify the record. Common examples include invoice numbers, batch ID's, employee numbers, etc. Panorama can automatically assign a unique number to each new record as it is created, even if several people are using the database simultaneously over a network.

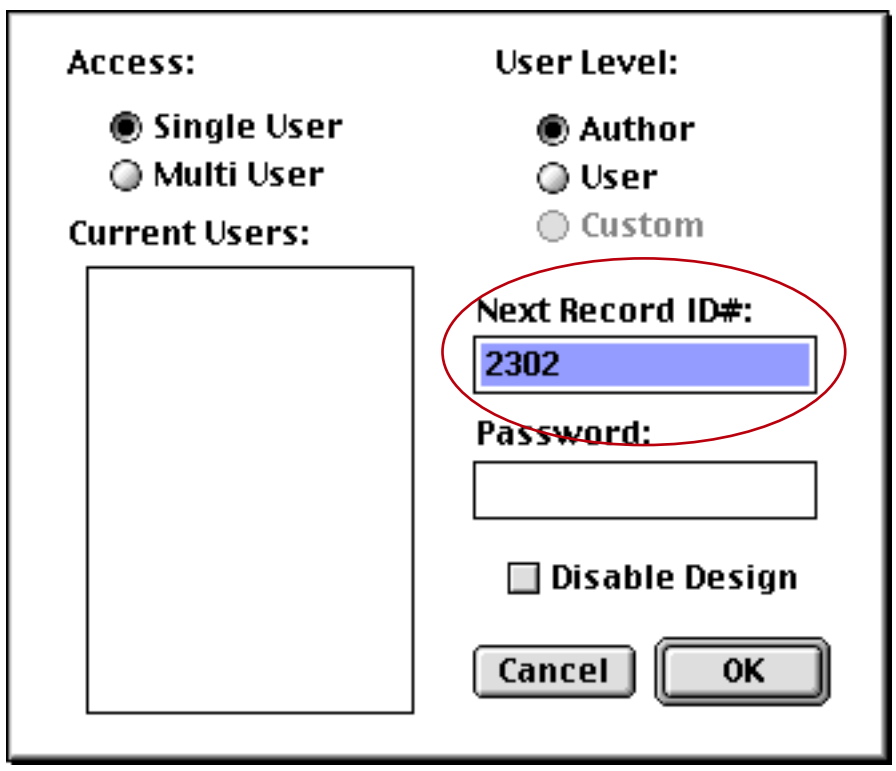
The field containing the record number must be a numeric field. To specify that this field should contain a unique record number, the default should be + . Do not specify any increment value, just use a single + character.



Each database contains a counter for keeping track of the next record number. Every time a new record is created the counter is incremented by one. Even if the record is later deleted, the number will never be re-used (unless you **Quit** Panorama or close the database without saving your changes, or unless you reset the counter manually as described below).

Manually Changing the Record Number Counter

You can manually change the record number counter using the **Privileges** dialog. (See "The Privilege Dialog" on page 238 if you don't know how to open this dialog.) Simply type in any integer value for the Next Record ID# option.



It is also possible to access and modify this ID number in a procedure. (See "GETAUTONUMBER" on page 5290 and "SETAUTONUMBER" on page 5738.) To access the next record ID # use the `GetAutoNumber` statement. Here is a simple procedure that displays the next record ID number.

```
local id
GetAutoNumber id
message "The next record number will be "+str(id)+"."
```

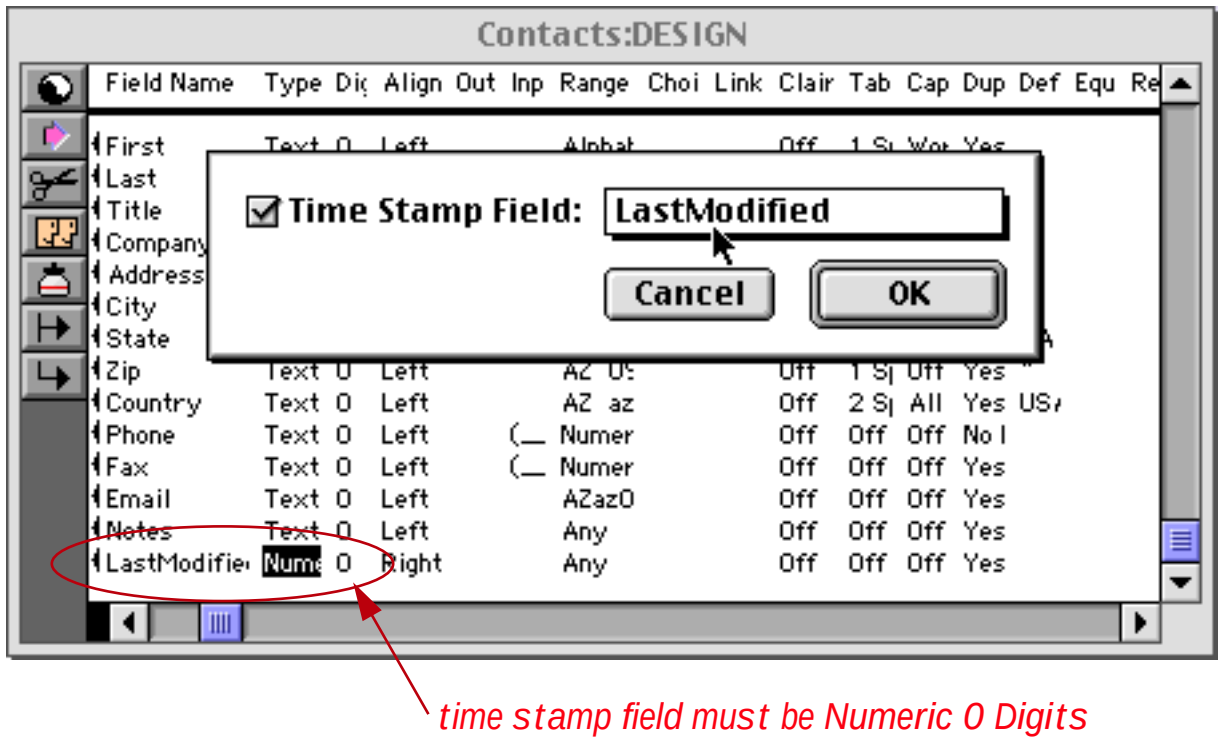
This one line procedure uses the `SetAutoNumber` statement to reset the record ID number to 1000.

```
SetAutoNumber 1000
```

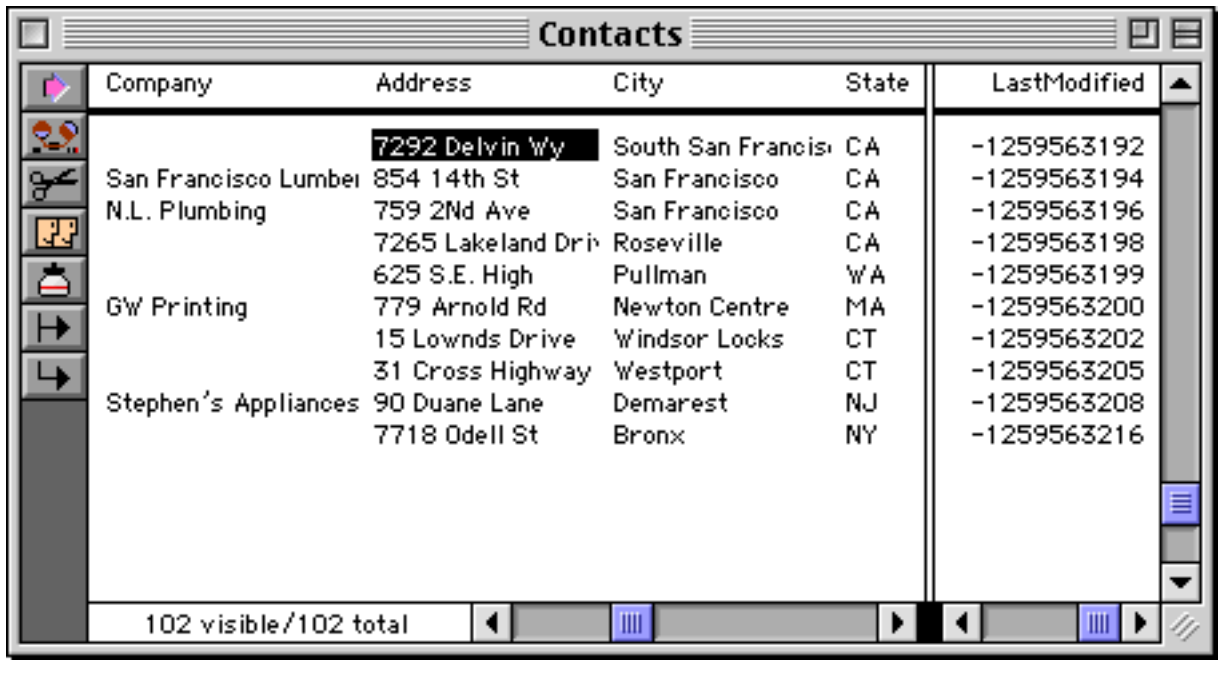
Automatic Time/Date Stamping

Using the design sheet you can designate a field as a **time stamp** field. Once a field has been designated as a time stamp field, Panorama will automatically copy the current date and time into this field every time any other field in the record is modified. Setting up a time stamp field allows you to reliably track when each record in the database was last modified.

To set up a time stamp field you must have the design sheet open. Choose the **Time Stamp Field** command from the Setup menu, then use the pop-up menu to select the field that will become the time stamp field. This field must be an integer (numeric 0 digits) field and it must already exist in the data sheet. (Just adding a new field to the design sheet isn't enough to make it appear in the Time Stamp dialog's pop-up menu. You must also make a new generation.)



Once the time stamp field is set up, Panorama will automatically update the time and date every time any cell in that database is modified. The value stored in the cell is actually the number of seconds since midnight, January 1, 1904. This combination of date and time into a single number is called a **SuperDate**. See "Super-Dates (combined date and time)" on page 1259 for more information about SuperDates.



As you can see, it's pretty difficult to look at a SuperDate and make much sense of it. Fortunately, you can convert a SuperDate into a regular Panorama date with the `regulardate(` function, and into a regular time (seconds since midnight) with the `regulartime(` function. Here is a formula that converts a SuperDate in the field LastModified into a readable date and time.

```
datepattern(regulardate(LastModified),"mm/dd/yy")+" @ "+
   timepattern(regulartime(LastModified),"hh:mm:ss am/pm")
```

This formula may be used in an auto-wrap text object (see "Displaying Formulas in Auto-Wrap Text" on page 602) or Text Display SuperObject (see "Text Display SuperObjects™" on page 608) to display the modification date in a format like this: 11/27/03 @ 4:37:22 PM.

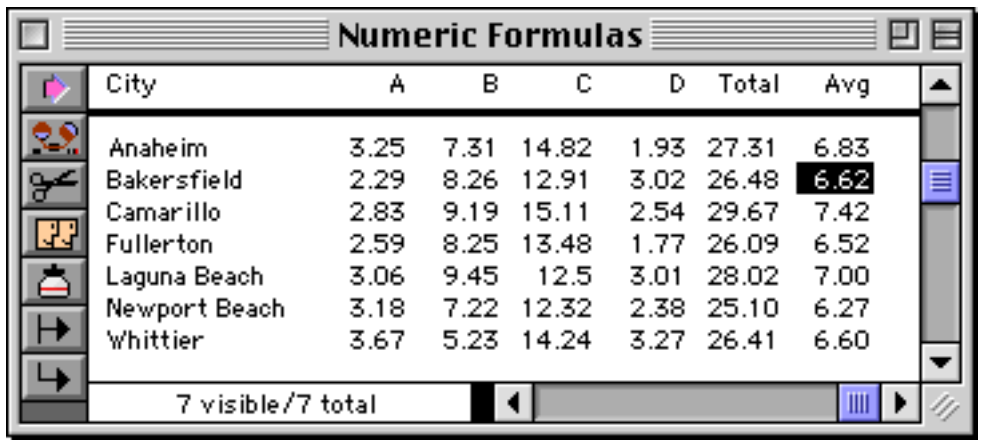
Automatic Calculations

The Equation column in the design sheet allows you to set up a formula for calculating the value of a field based on the values in other fields (see "The Design Sheet" on page 252 if you are not already familiar with using the design sheet).

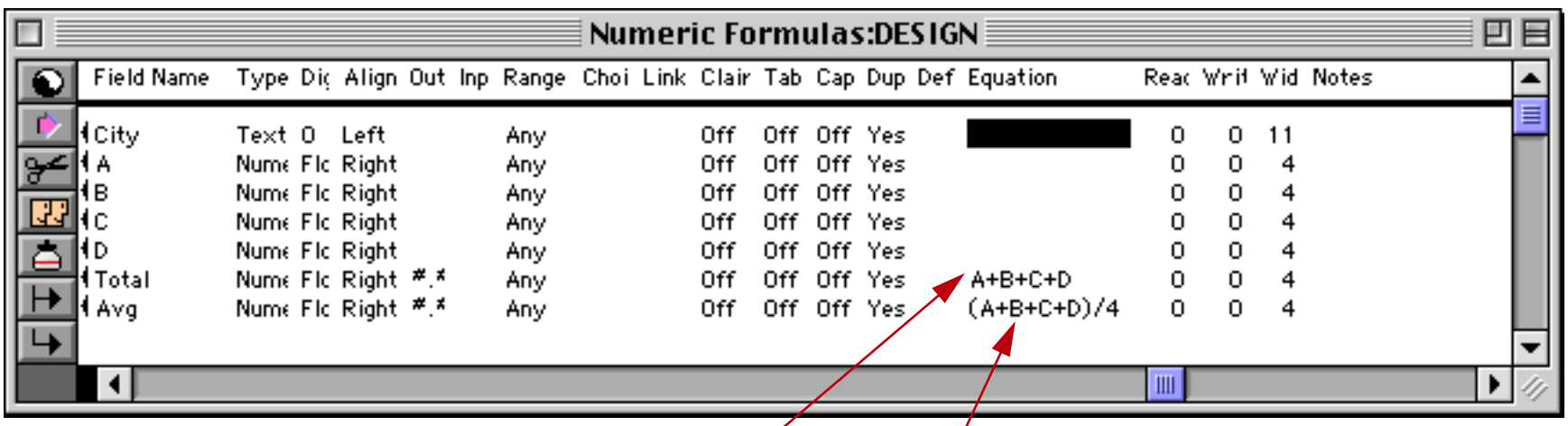
In the following sections you'll learn how to perform calculations within the current record as data is entered. To learn how to calculate an entire column of data at once see "Filling a Field with a Formula" on page 453 and "Storing Formula Results in the Database" on page 1168. To learn how to display the result of a calculation without storing it in the database see "Displaying Formulas in Auto-Wrap Text" on page 602, "Text Display SuperObjects™" on page 608 and "Displaying/Printing A Formula" on page 1166.

Spreadsheet Mode Calculations

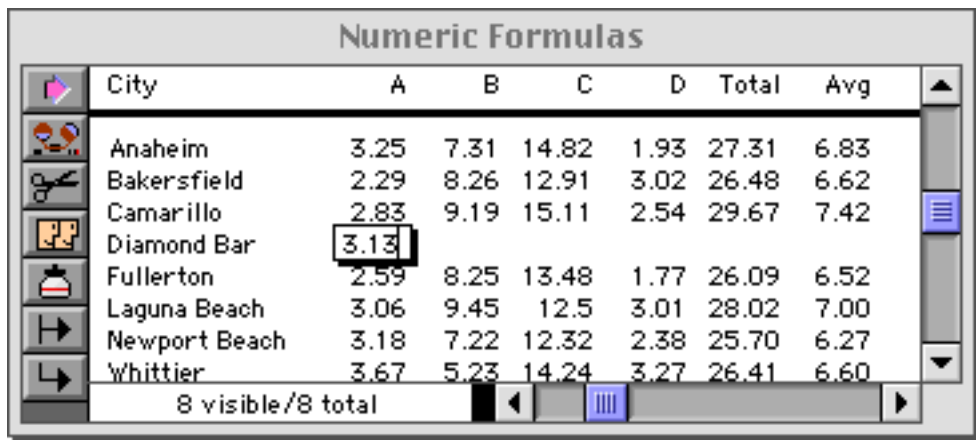
If you've ever used a spreadsheet you'll be very comfortable with Panorama's default **Spreadsheet Mode** for automatic calculations (see "Procedure Mode Calculations" on page 347 for an alternative calculation mode). In this mode you simply place the formula for calculating a field into the Equation column of the design sheet for that field. To illustrate this we'll use this simple database with four numeric fields for data entry (A, B, C and D) and two fields that we will be calculated (Total and Avg).



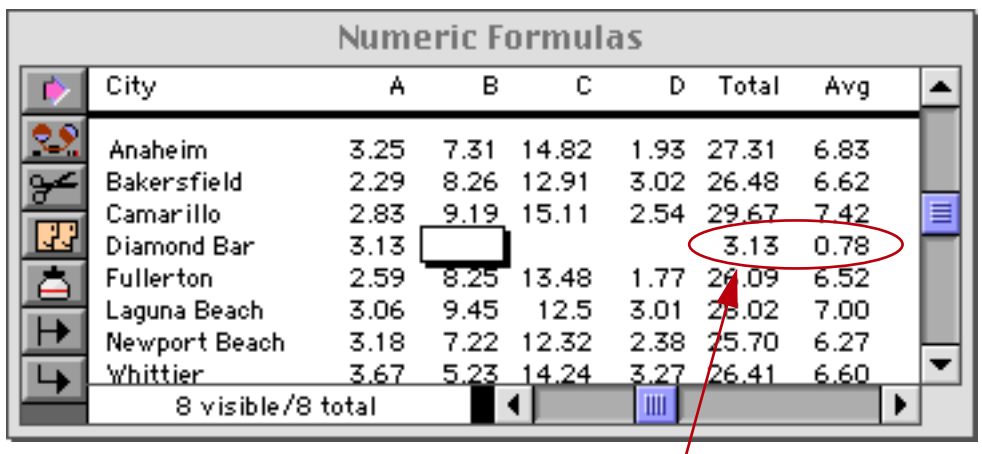
To automatically calculate the total and average simply enter the formulas in the appropriate rows of the Equation column in the design sheet.



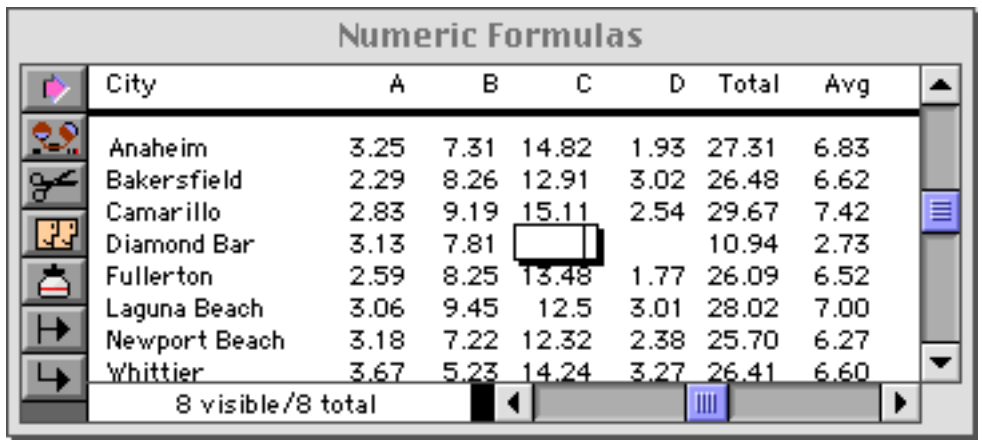
To activate these formulas you need to create a new generation for this database (see "Database "Generations"" on page 252). Once you've done this you can start entering or updating information. In this illustration a new record has been added (Diamond Bar) and the first number typed in (but not entered into the database yet).



As soon as the data is entered by pressing the **Tab** (or **Enter**) keys the formulas update the Total and Avg fields.

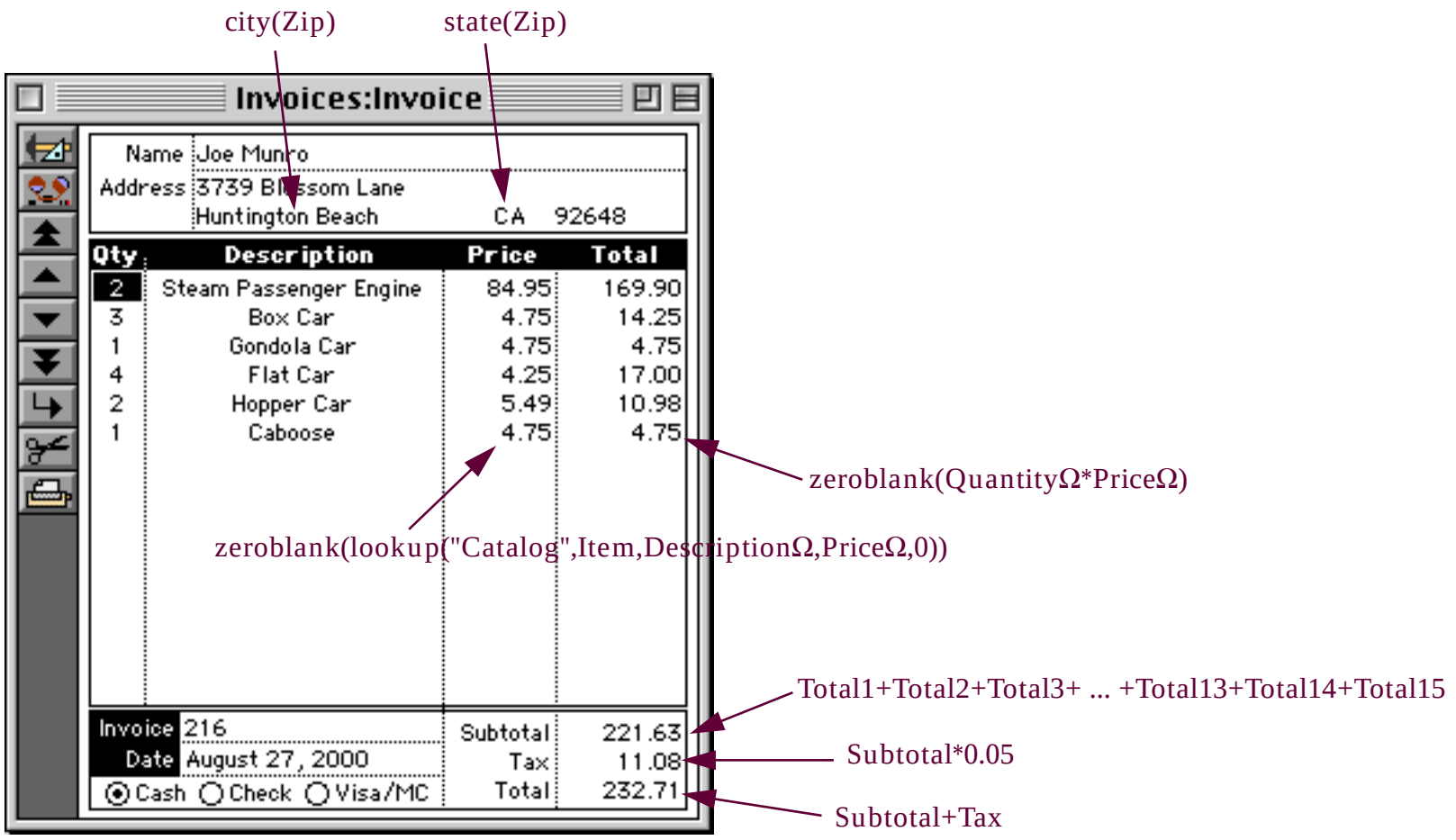


As more data is entered the Total and Avg fields are updated instantaneously.

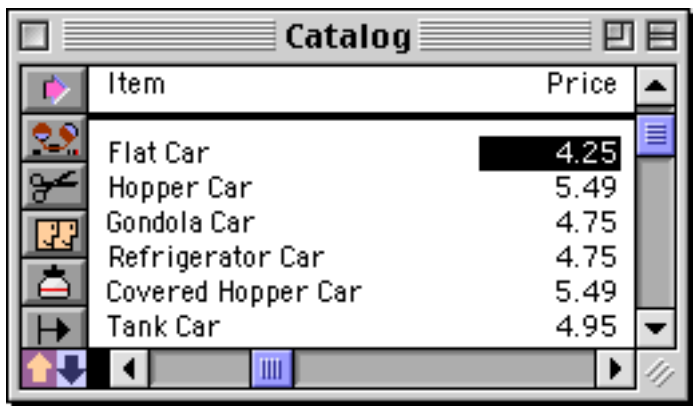


The Total and Avg fields will be updated any time the A, B, C or D fields are modified.

Next we'll look at a simple invoice that shows some more realistic examples of automatic calculations in action. Here are the calculations needed as data is entered.

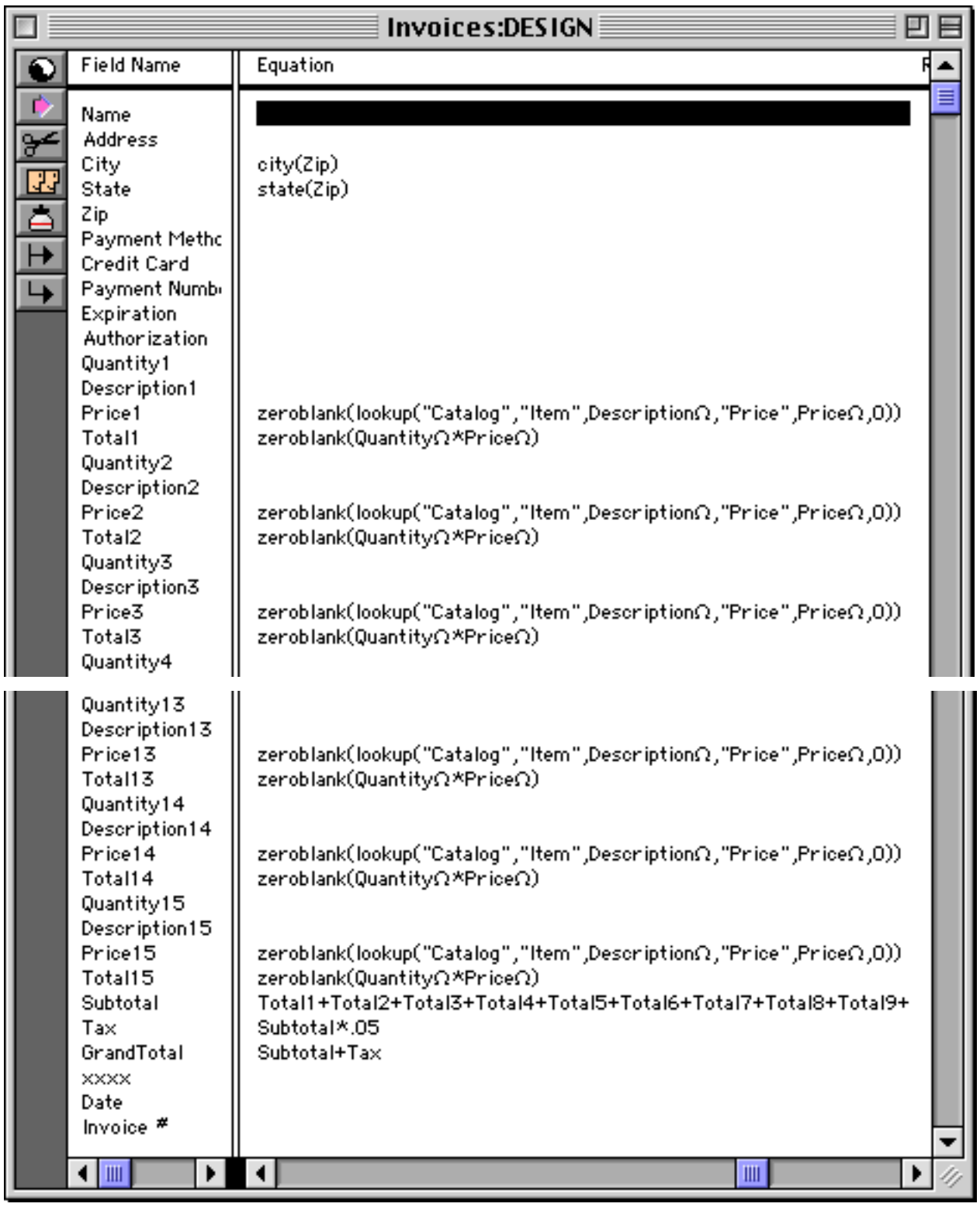


This table explains each of these formulas. Several of these formulas reference line item fields using the Ω character, see "Line Item Fields" on page 1198 to learn how these fields can be used in a formula.

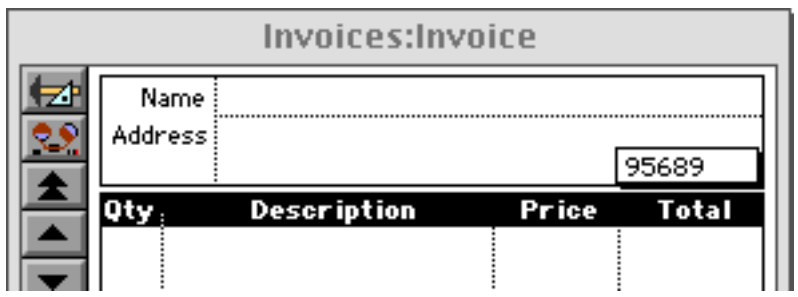
Formula	Explanation
`city(Zip)`	Using Panorama s optional zip code lookup database (see Zip Code Lookup on page 1286) this formula looks up the name of the city when the zip code is entered. For example, if 92548 is entered this formula will calculate that the city name is Huntington Beach.
`state(Zip)`	Using Panorama s optional zip code lookup database (see Zip Code Lookup on page 1286) this formula looks up the name of the state when the zip code is entered. For example, if 92548 is entered this formula will calculate that the state is CA.
`zeroblank(lookup("Catalog",Item, DescriptionΩ,Price,0)`	When an item is entered this formula looks up the price from a catalog database (see Linking With Another Database on page 1272). Here s what the catalog database, which must be open, looks like.  Panorama s Clairvoyance Link option has been used to help make sure that the Description always matches an item in the catalog database. See Clairvoyance¤ Across Multiple Files on page 323 to learn how to set up this option.

Formula	Explanation
`zeroblank(QuantityΩ*PriceΩ)`	This formula calculates the total for a each line item row (see Line Item Fields on page 1198). The zeroblank(function (also used in the previous formula) suppresses zero values to make sure that the empty items really look empty (see ZEROBLANK(on page 5915).
`Total1+Total2+Total3+ ... Total13+Total14+Total15`	This formula calculates the sum of all the line item totals. The sum(function (see Adding Line Item Fields on page 1207) cannot be used because it is not compatible with Spreadsheet Mode. If this calculation is being performed in Procedure Mode (see Procedure Mode Calculations on page 347) this formula can be replaced with the much simpler formula sum("Total‰").
`Suttotal*0.05`	This formula calculates the sales tax at 5%. To calculate the sales tax for a non-integer rate (for example 7.75%) using 2 digit numeric fields the formula must be rewritten as (Subtotal*7.75)/100 to avoid numeric overflow problems (see Overflow/Underflow Problems on page 1206). If you are using floating point fields the formula can simply be Subtotal*0.0775.
`Subtotal+Tax`	This formula calculates the grand total.

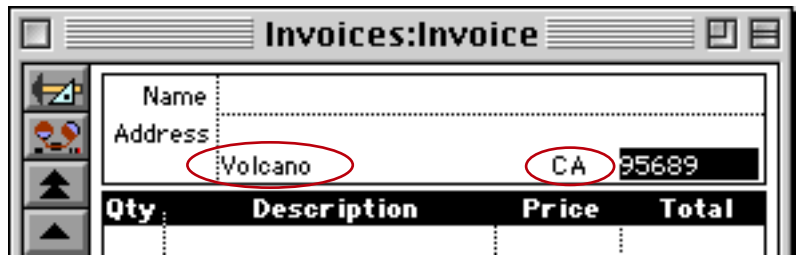
Each of these formulas must be set up in the Equation column of the design sheet. The Price and Total line item formulas only need to be entered once. Panorama will automatically copy them to the other line item fields (see "Modifying Line Item Fields" on page 265).



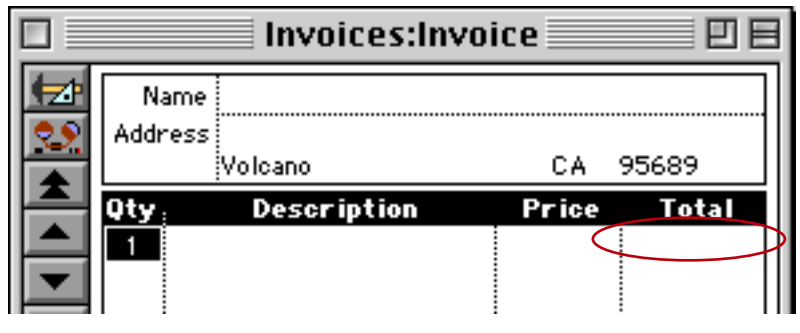
Once the formulas have been typed in you need to create a new generation for this database (see "Database "Generations"" on page 252). Once you've done this you can start entering or updating information. For example you can enter the zip code 95689.



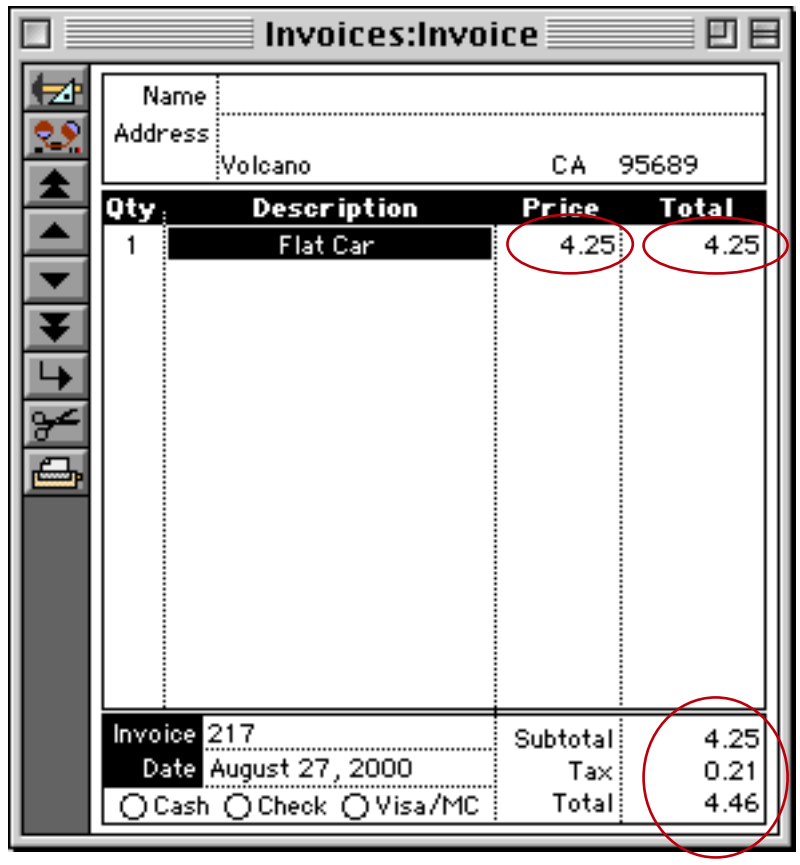
If the optional Zip Code dictionary is installed the formula will “calculate” the city and state. Yes, there really is a place named Volcano, California.



When you enter a quantity Panorama calculates the line item total. Since the price hasn’t been entered yet, the total is zero, which is suppressed by the `zeroblank(` function (see “ZEROBLANK(” on page 5915).



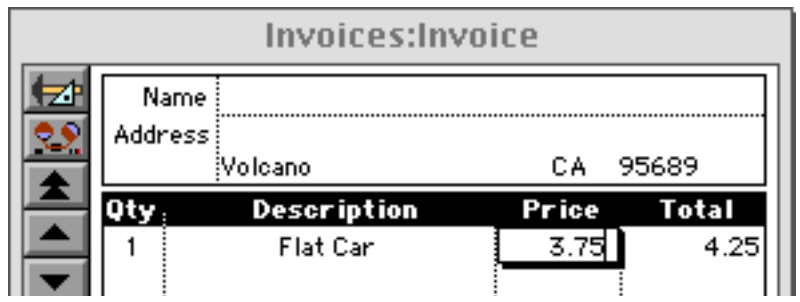
When a description is entered Panorama kicks into high gear. The automatic calculations look up the price from the catalog database, multiply the price by the quantity for the line item total, calculate the subtotal, the tax, and the grand total.



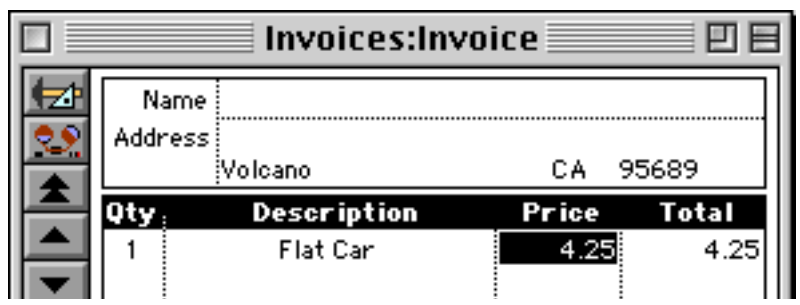
One potential problem appears if a calculation is “circular,” in other words, if the result of the calculation is part of the calculation itself. In this database the formula for calculating the Price field actually contains the Price field.

```
zeroblank(lookup("Catalog",Item,DescriptionΩ,Price,PriceΩ,0)
```

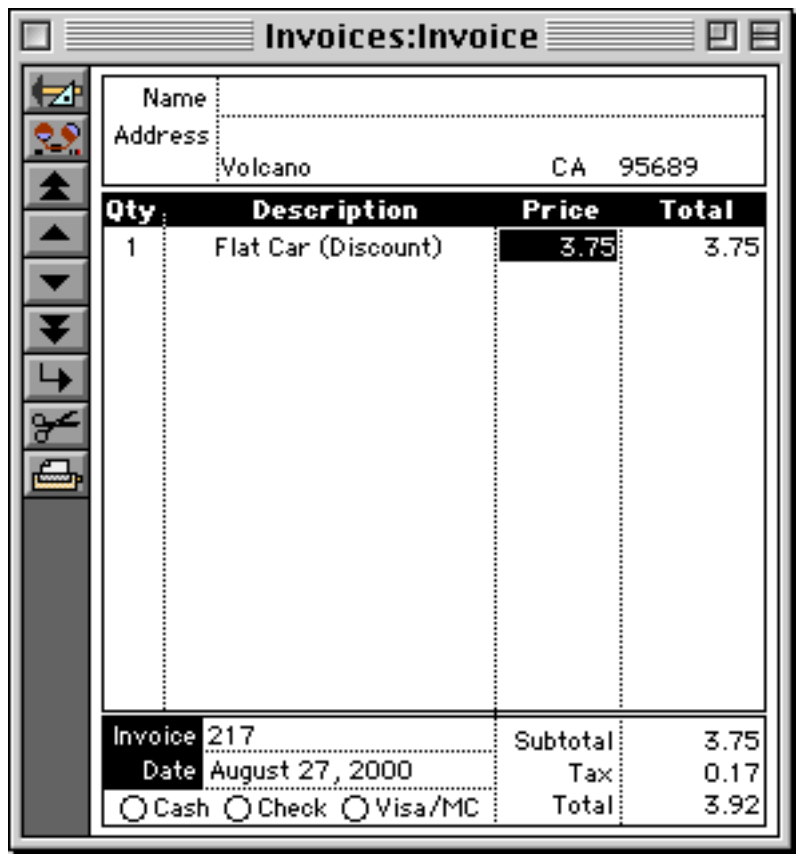
If you attempt to edit a price, this will trigger the calculation, which modifies the price. Let’s suppose you decide to give the customer a 50 cent discount on their flat car.



When you press the **Enter** key Panorama performs the calculations. But in this case part of the calculation is to look up the price from the catalog database. This resets the price back to $4.25.



The solution is to edit the description so that it no longer matches any entry in the catalog database. In this case we’ve added the word (Discount) after the item name. Once the description has been modified you can go ahead and mark down the price.

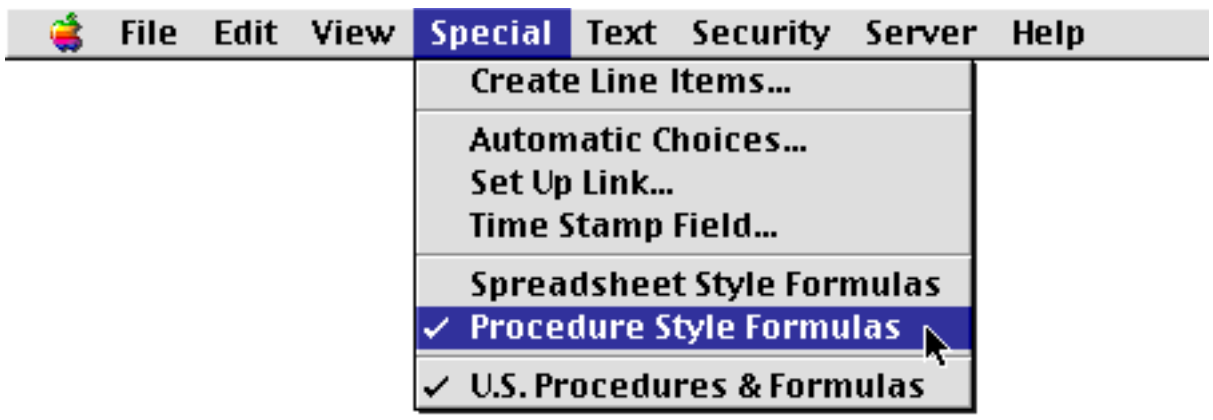


For another solution to this “circular” calculation problem see the next section.

Procedure Mode Calculations

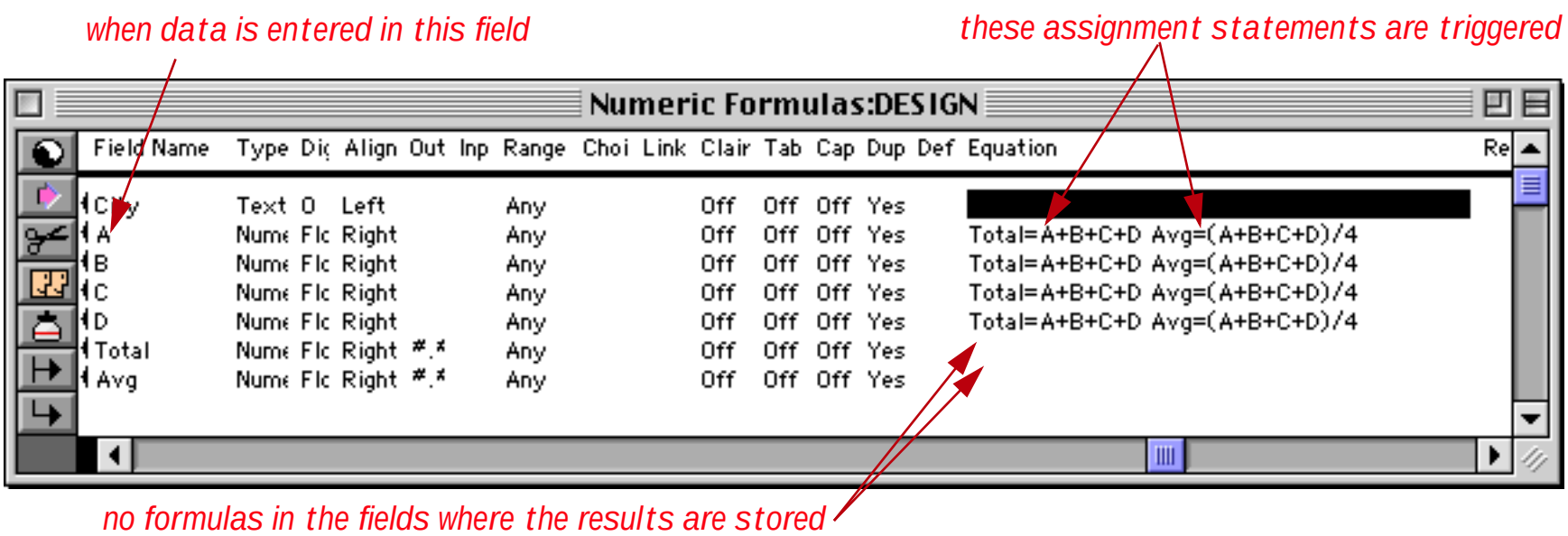
An alternate mode for performing calculations is called **Procedure Mode**. In this mode little "mini-procedures" are triggered when data is entered into a cell. Each mini-procedure can contain one or more assignment statements (see "Assignment Statements" on page 1368). A mini-procedure can also trigger a real procedure (see "Automatically Triggering a Procedure" on page 350).

To use procedure mode you must turn it on. This is done with the **Special** menu in the design sheet.

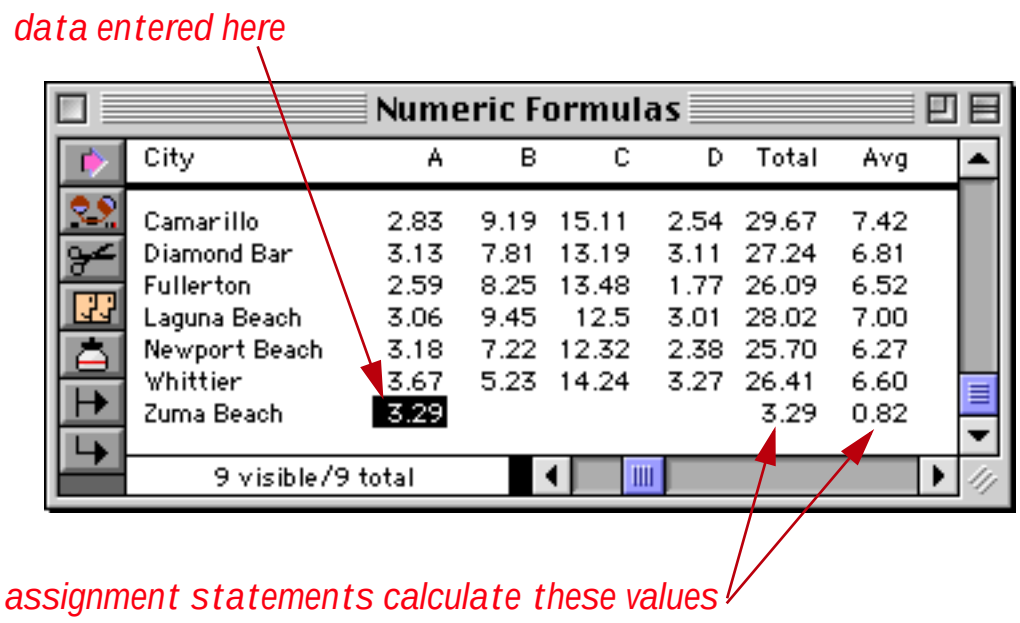


Note: New databases default to **Spreadsheet Mode**. However, if a database was created with Panorama 2.1 or earlier it will default to **Procedure Mode** for compatibility with these earlier versions of Panorama. Either way you can easily switch back and forth between the modes using the **Special** menu. However, you will have to re-create any formulas in the Equation column when you make the switch.

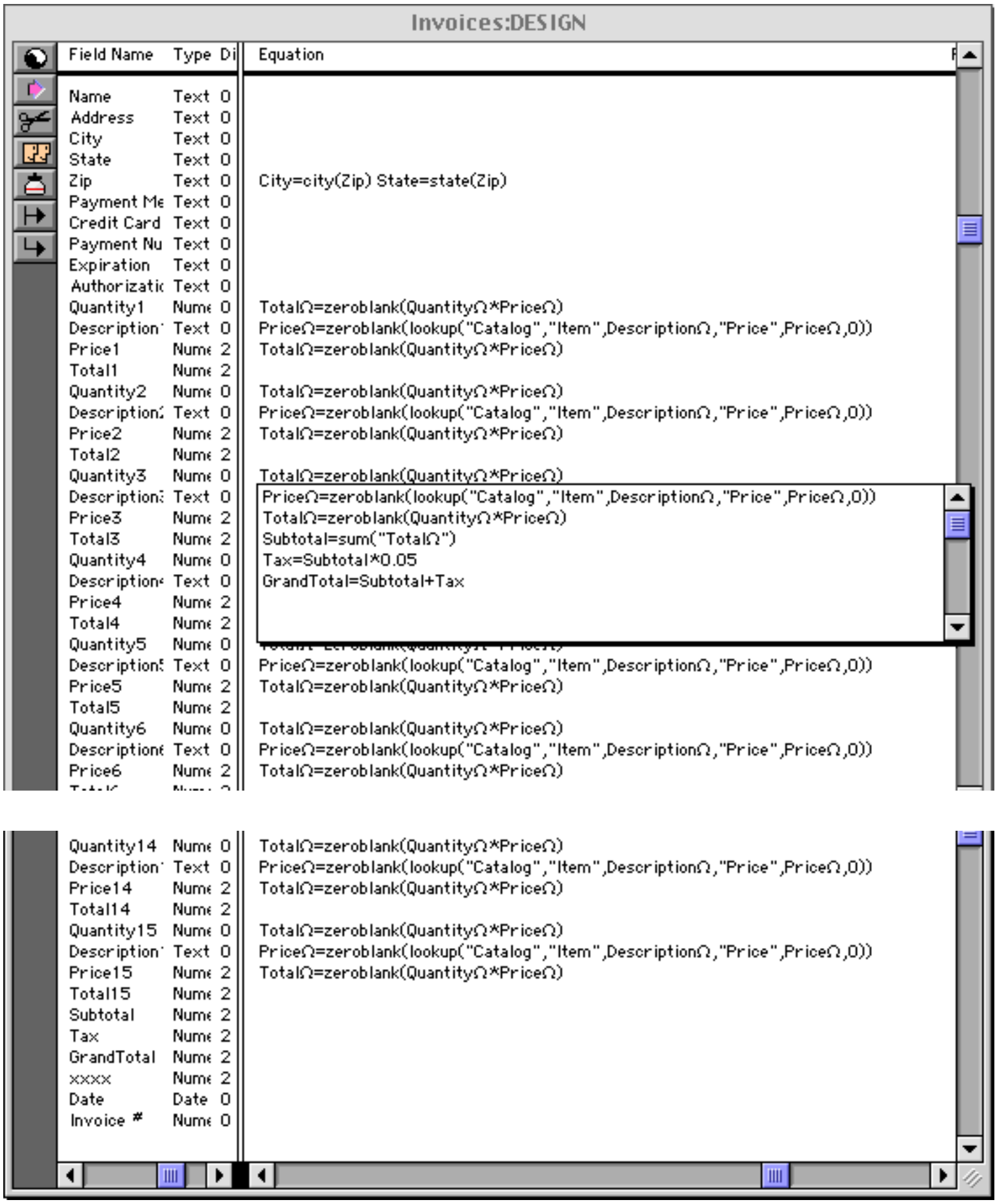
When procedure mode is used the assignment statements go along with the field(s) where the data entry is done, not where the answer is placed. This usually means that you'll have to repeat the assignment statements to place them in each column where the data is entered. Use the **Copy** and **Paste** commands to duplicate the assignment statements into each field.



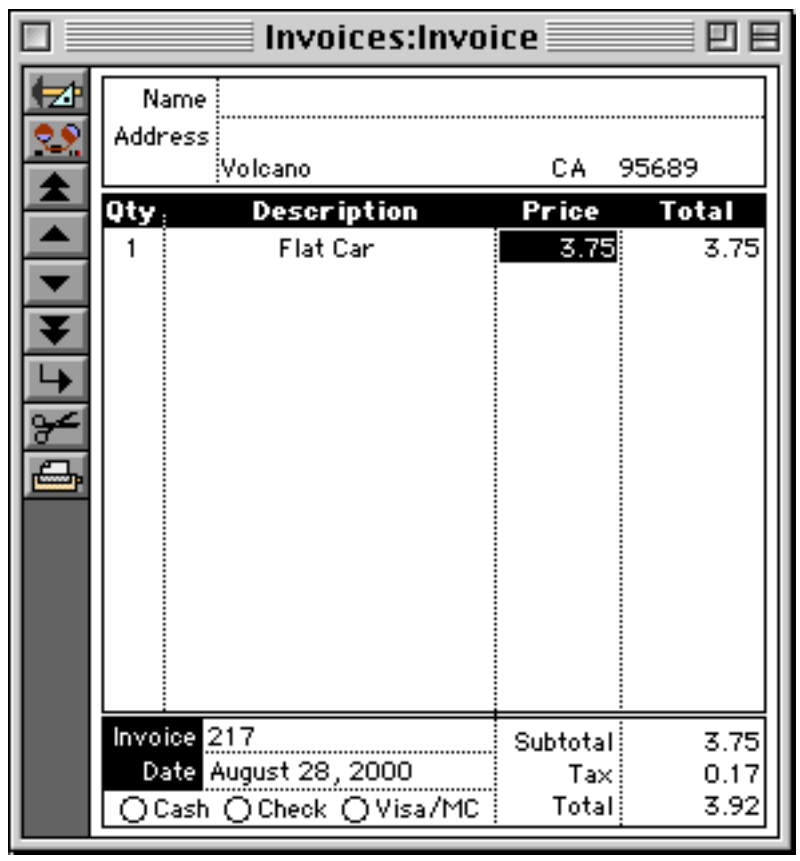
Once these assignment statements are entered and the new generation has been created(see "Database "Generations"" on page 252) the automatic calculations will be performed when data entry is performed, just as it was when Spreadsheet Mode was used.



Here is the design sheet for a revised version of our simple invoice example. This version is designed to use Procedure Mode calculations. We've opened the Equation cell for Description3 so that you can see all five assignment statements that are performed when data is triggered in this field. As you can see these are basically the same formulas that we used in Spreadsheet Mode, but re-arranged into a single cell. In addition, we also used the `sum(` function (see "SUM(" on page 5816), which works just fine in Procedure Mode.



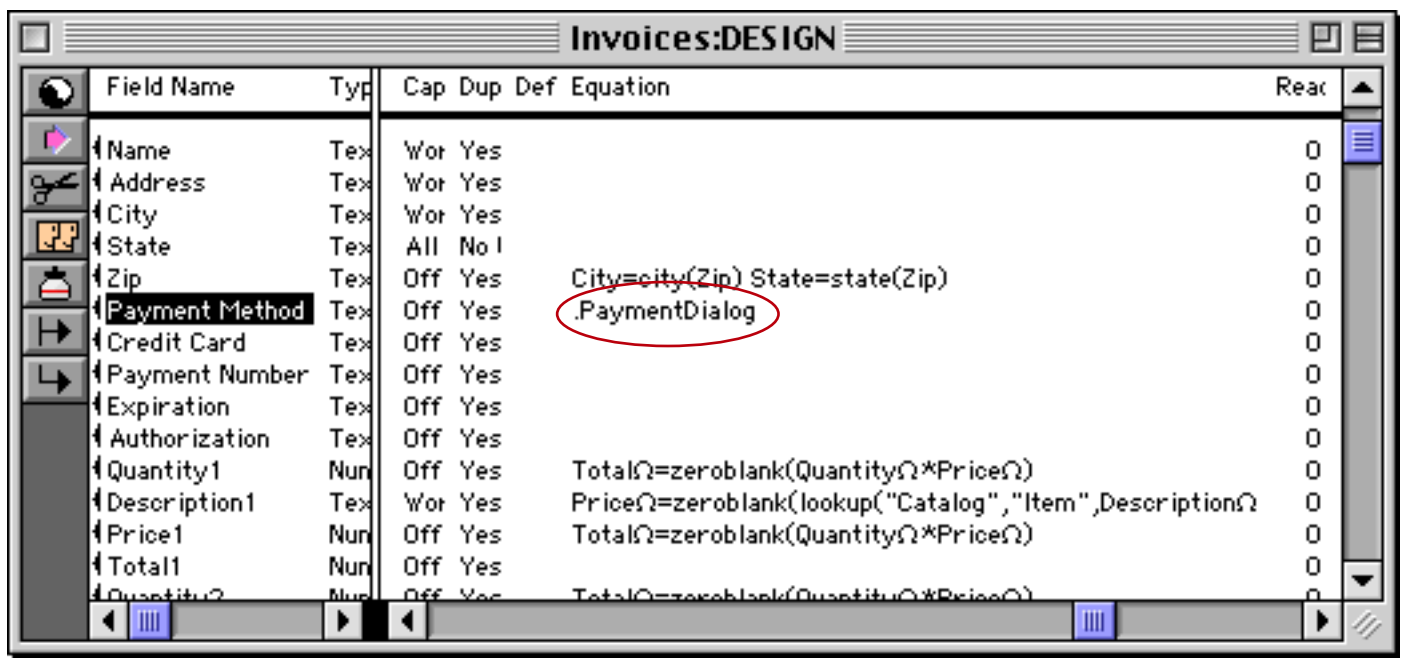
Using Procedure Mode gives you precise control over what calculations are performed. In this example that means that an item's price can be modified without Panorama trying to re-lookup the price from the catalog database (as it did in the previous section in Spreadsheet Mode).



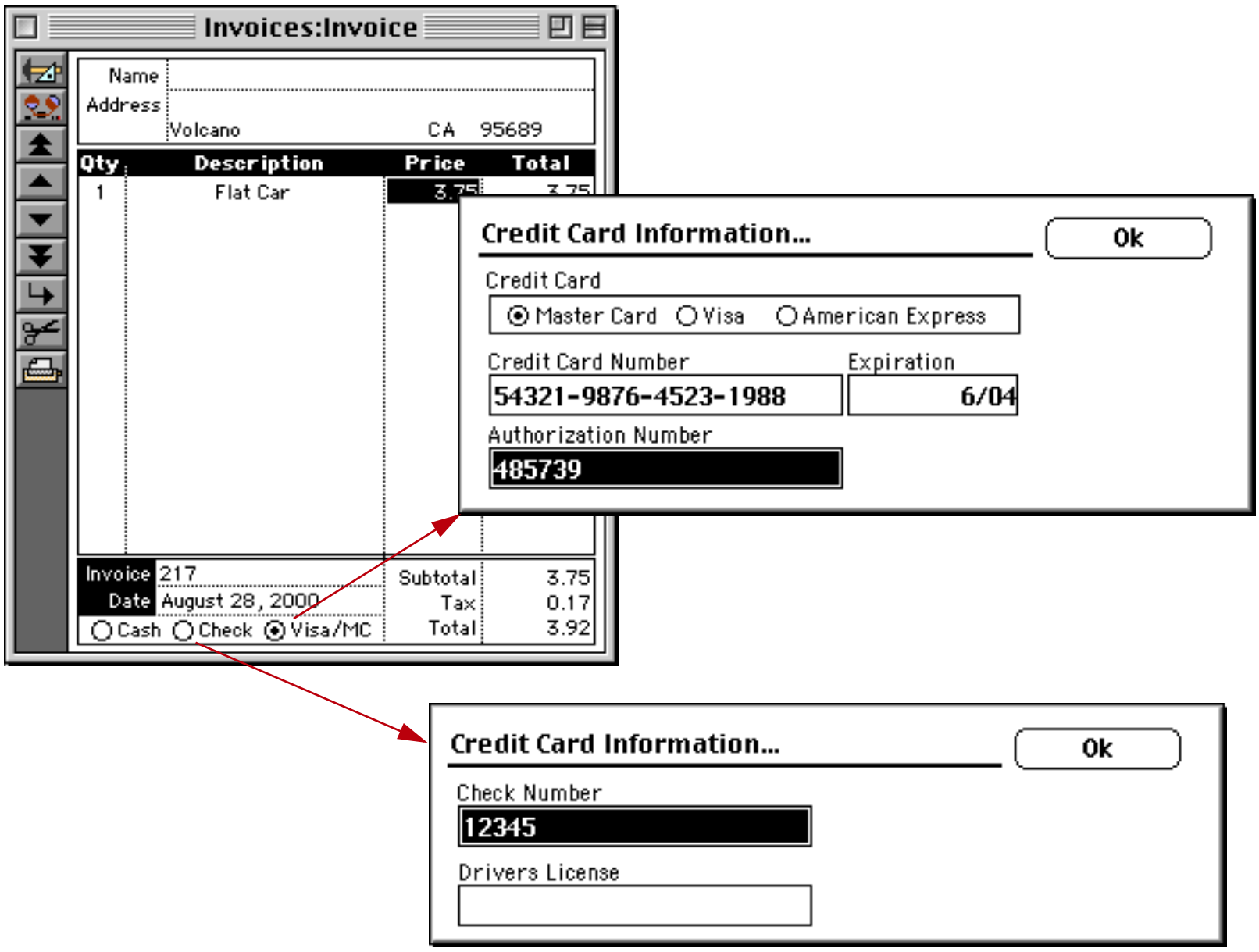
As you can see we have been able to discount the price without having to modify the description. This precise control over exactly what calculations are performed is one of the advantages of Procedure Mode, and sometimes makes it worth the extra effort.

Automatically Triggering a Procedure

When Procedure Mode is used data entry can trigger a real procedure in addition to performing calculations. To trigger a procedure simply enter the name of the procedure in the Equation column of the design sheet (the procedure name may not contain any spaces). If the procedure name is combined with any assignment statements the procedure name must come last. Here is an example that triggers the **.PaymentDialog** procedure whenever the Payment Method field is edited.



This procedure makes a dialog appear whenever the Payment Method field is modified (in this case by clicking on radio buttons). The exact dialog that appears depends on which radio button was clicked.



If you are using Spreadsheet Mode there is no way to trigger a procedure from the design sheet. However, you can still create a **.ModifyRecord** procedure that will be triggered whenever any field is modified. See ".ModifyRecord" on page 1512 to learn more about this special procedure.

Pros and Cons of Spreadsheet vs. Procedure Mode

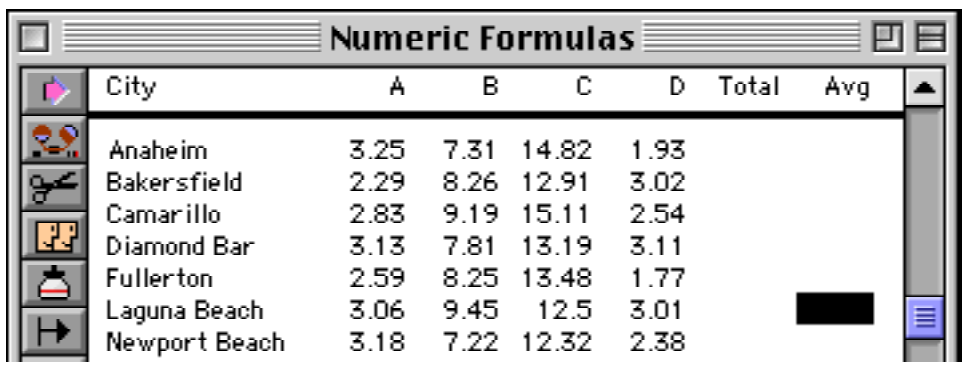
How do you decide which mode to use? The table below summarizes the advantages of each mode.

Spreadsheet Mode	Procedure Mode
• Easier to use (less typing) • Easier to modify • Less chance of errors • More familiar to spreadsheet users	• More precise control • Can trigger a procedure • Works with sum(function • Slightly faster execution

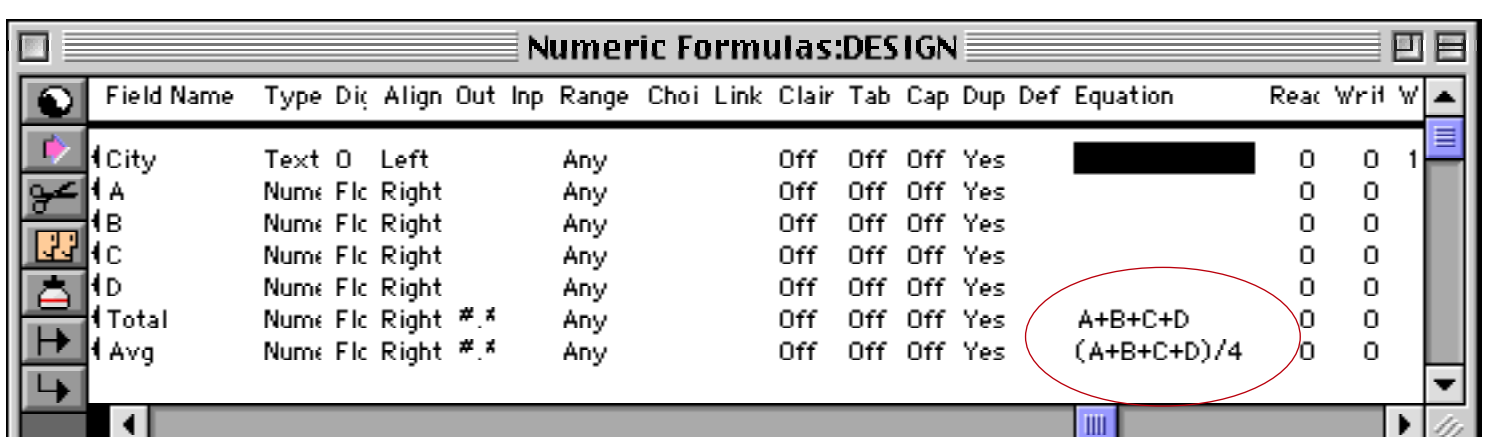
For most applications Spreadsheet Mode works fine and is easier to use.

The Run Automatic Calculations Wizard

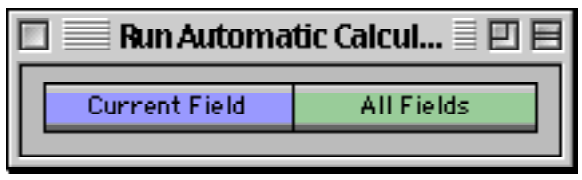
When you set up an automatic calculation that calculation is automatically applied when new data is entered or existing data is modified. The calculation is not applied to any existing data. One way to apply a calculation to existing data is to use the **Formula Fill** command in the Math menu (see "Filling a Field with a Formula" on page 453). Another method is to use the **Run Automatic Calculations** wizard. This wizard will perform calculations based on the formulas you have entered into the design sheet (see "Automatic Calculations" on page 340). For example, consider this sample database.



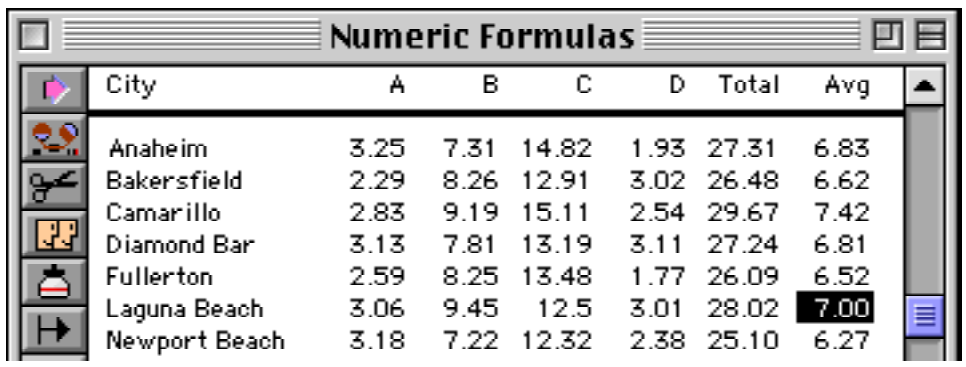
Looking at the design sheet we can see that calculations have been set up for the Total and Avg fields.



When new data is entered into this database the Total and Avg fields are calculated automatically. However, they are not calculated for the existing data. To perform this calculation, open the **Run Automatic Calculations** wizard.



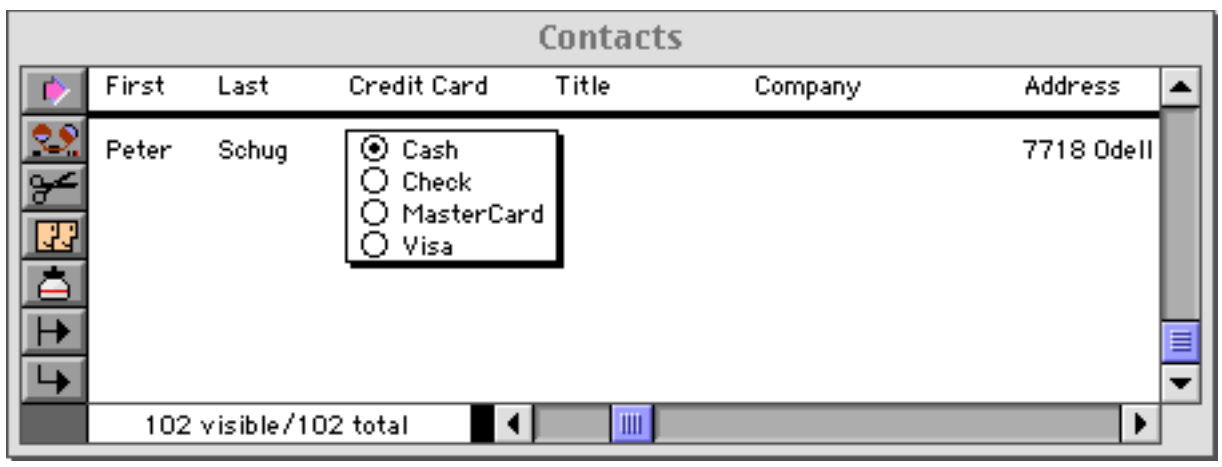
To calculate the values for all fields that have calculations set up (in this case Total and Avg) press the **All Fields** button. The wizard will perform the calculations and fill in the fields. (Note: This wizard only works with Spreadsheet mode formulas, it does not work if the database uses Procedure mode.)



If you only want to perform the calculation for the current field press the **Current Field** button. (If the current field doesn't have a calculation set up an alert message will appear.)

The Choice Palette

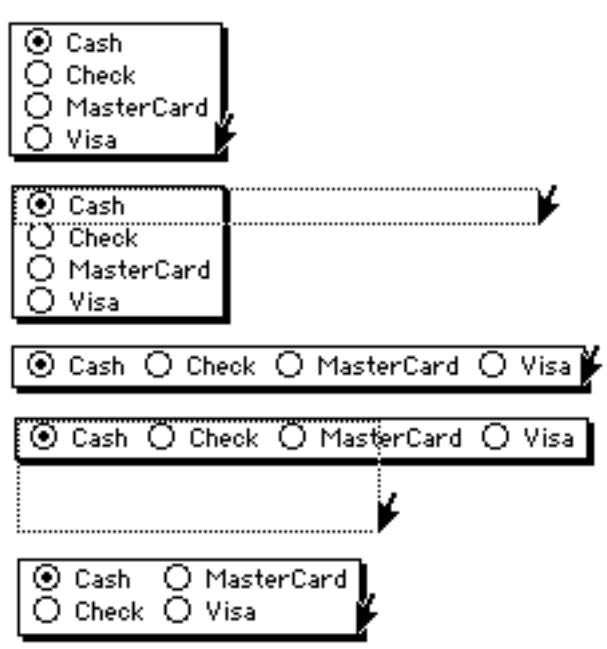
The choice palette provides a completely different way to use the Input Box. Instead of entering the data with the keyboard, you pick the value from a list of buttons.



The choice palette can be used with any data type except pictures. All you have to do is create a list of choices (see below).

Changing the Shape of the Choice Palette

Panorama automatically arranges the buttons in the choice palette for the best fit in the Input Box. By changing the size and shape of the Input Box you can arrange the buttons vertically, horizontally, or in a grid of rows and columns.



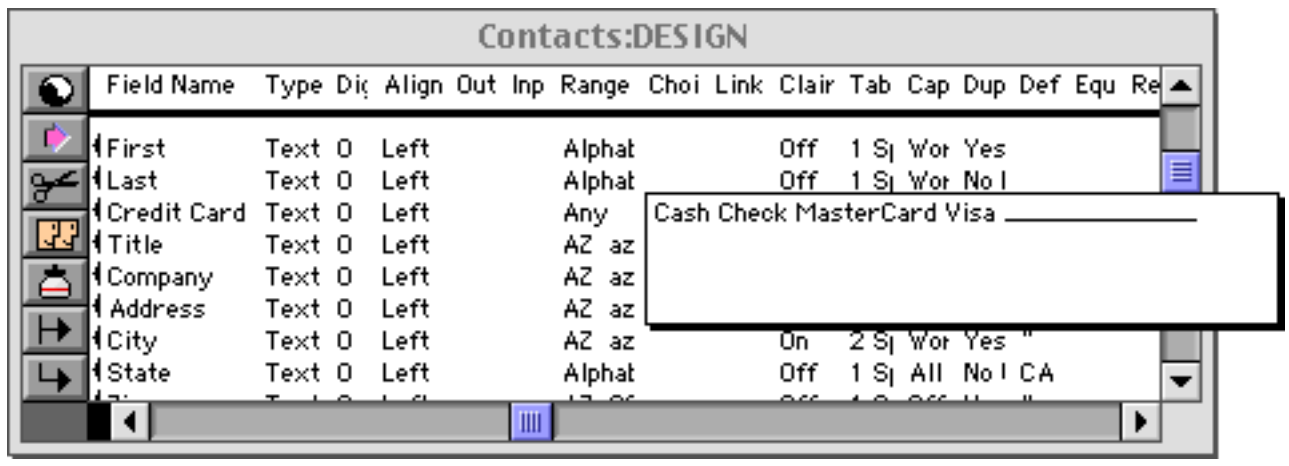
Creating the List of Choices

The list of choices (if any) for each field is kept in the Choices column of the design sheet. (See “The Design Sheet” on page 252 if you are not familiar with the design sheet.) You can key in the list manually or you can use the **Automatic Choices** command to create the list for you (See “Creating the List of Choices” on page 299).

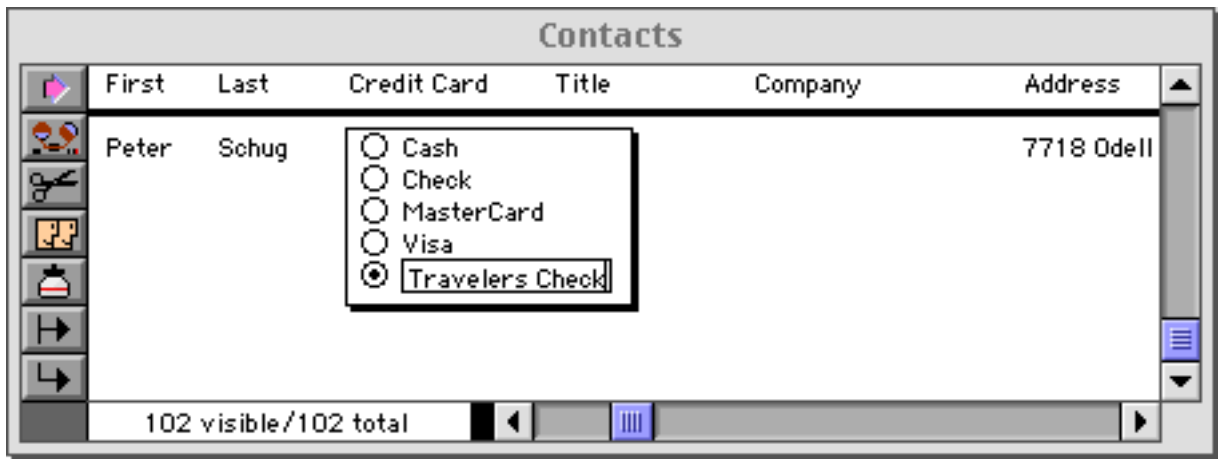
If you key in the list of choices manually, you must separate each choice with a space. If a choice contains a space (US Mail) you must represent the space with an underscore character (`US_Mail`).

Exceptions

An option for any choice palette is an exception box. The exception box lets you type any value into the data cell, even if the choice palette doesn't contain a button for it. Use the exception box when you have a few common choices, but cannot anticipate every value in advance. To create an exception box, simply type in a line of underscores at the end of the choice list.



The exception box is always one line high. The width of the exception box is controlled by the number of underscores.



To make the exception box wider, add more underscores.

The Choice Palette vs. the Choices Data Type

If you've already read about the choices data type (See "Choices" on page 298), you may be wondering how the choice palette relates to the choices data type.

The choice palette is a way to enter data. It provides a way to enter or edit data by picking from a list of choices instead of typing from the keyboard. Using the choice palette does not affect how that data is stored, however. The data can be stored as text, as a number, a date, or using the choices data type.

The choices data type is a way to store data. This data type is efficient (saves memory) for storing data that has only a few possible values.

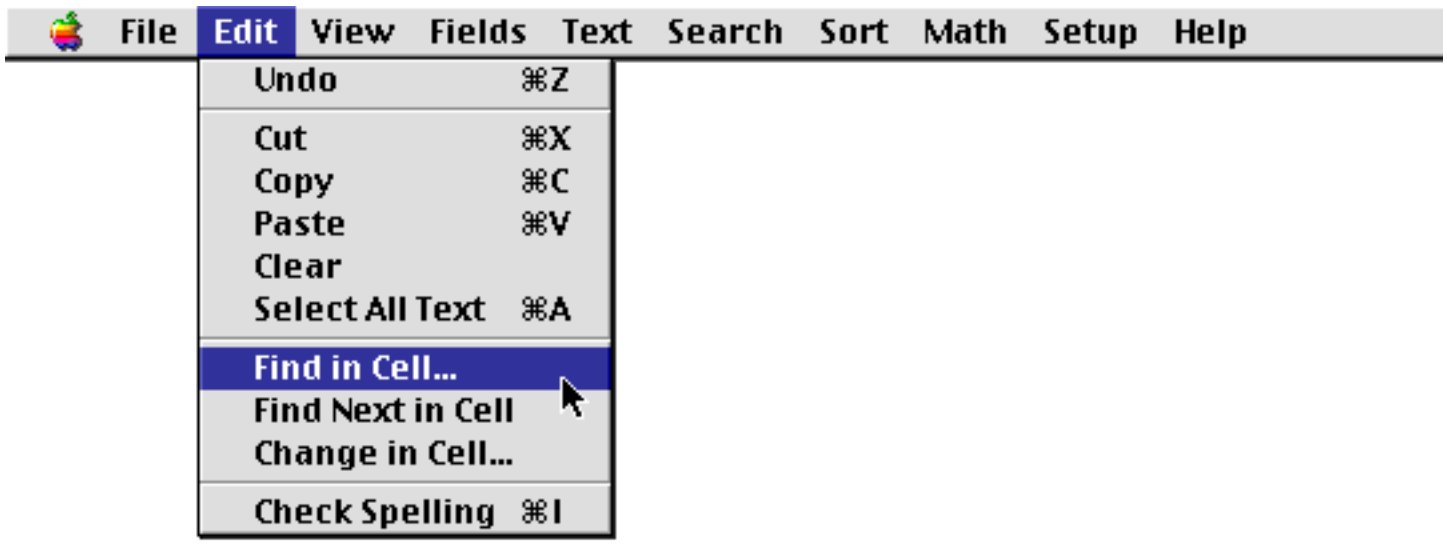
The choices data type can be used with a choice palette (and it usually is), but you can also use regular keyboard editing to enter choice values. A choice data type field will use regular keyboard editing (instead of the choice palette) if you set the number of digits to 1.

Editing Tools within a Data Cell

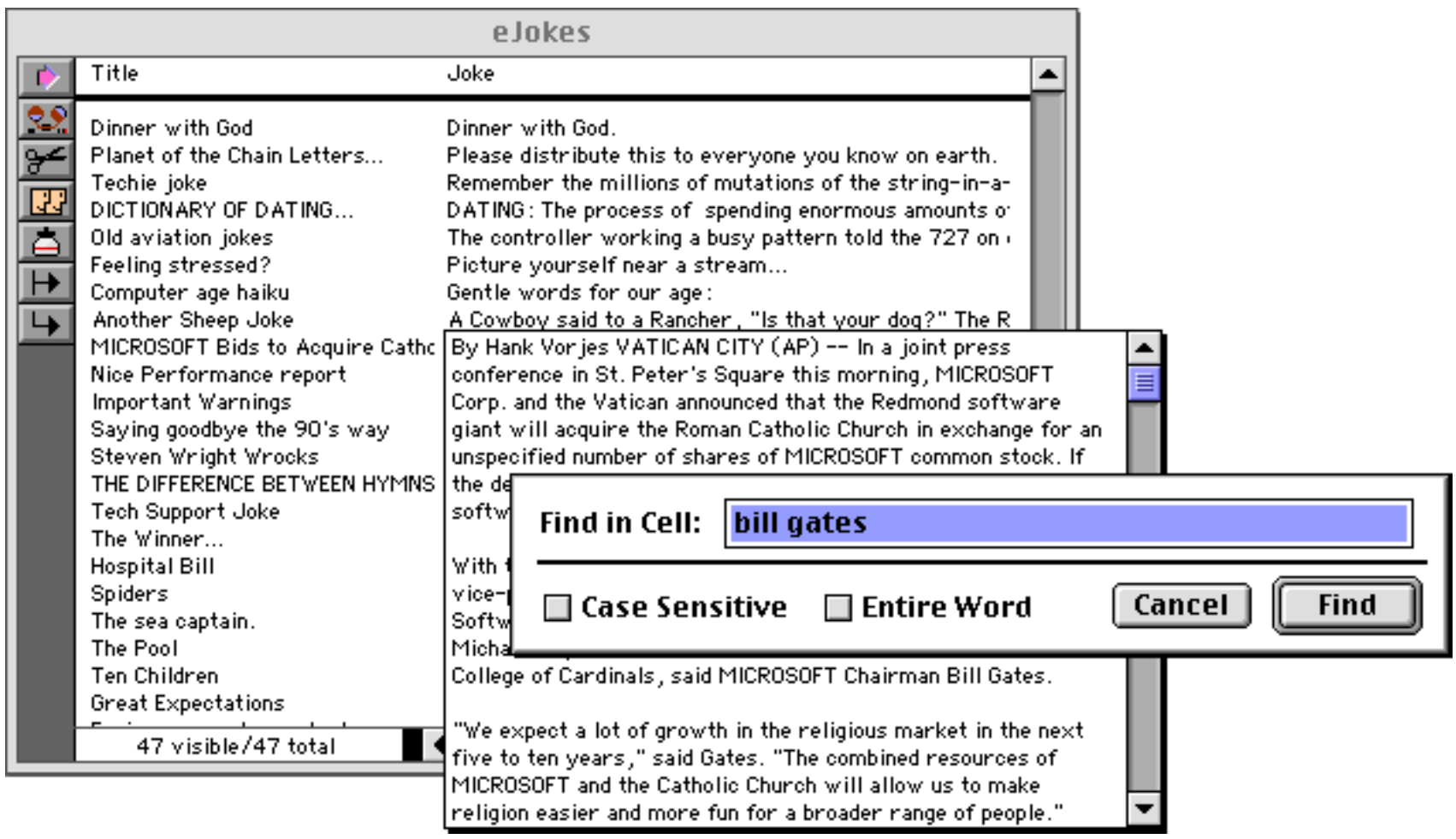
Most data cells contain no more than a few words or phrases. Some applications, however, require that each data cell contain several paragraphs or even an entire page of information. Panorama's search, replace, and spelling checker tools can help you accurately create catalogs, letters, glossaries, bibliographies, etc.

Searching for Text Within the Input Box

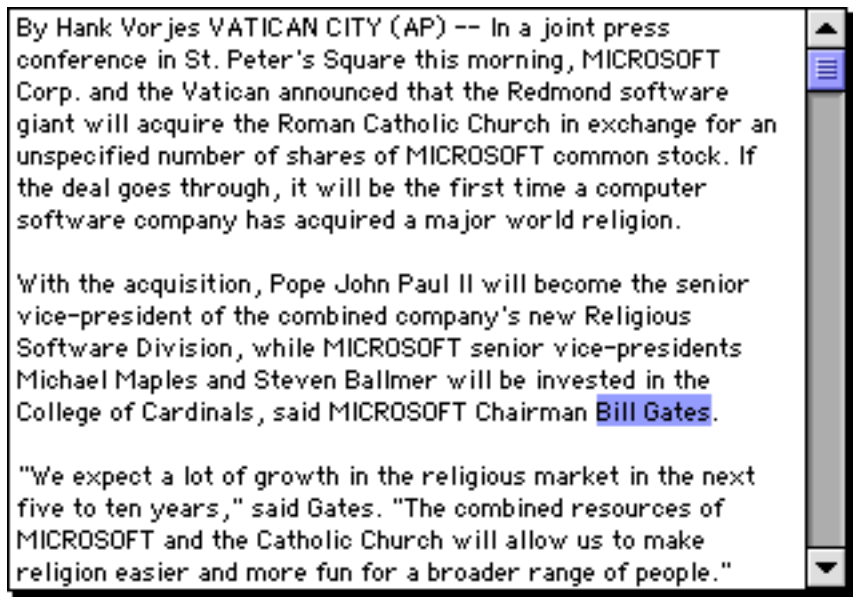
To locate a word or phrase within the Input Box use the **Find in Cell** command in the Edit Menu. Make sure you use the **Find in Cell** command in the Edit Menu, not the **Find/Select** command in the Search Menu.



Enter the word or phrase you want to search for into the dialog box, then press the **Find** button or the **Return** key.



Panorama will move the insertion point to the first occurrence of the word or phrase within the text in the cell.



If you want to look for more occurrences of the word or phrase, use the **Find Next in Cell** command in the Edit menu (not the **Find Next** command in the Search Menu).

The **Find in Cell** dialog has two options, Case Sensitive and Words Only. Use the Case Sensitive option if you want the **Find in Cell** command to only locate words or phrases that exactly match the capitalization of the text you are searching for. For example, when searching for the word Panorama, the **Find in Cell** command would normally also locate panorama and PANORAMA. However, if the Case Sensitive option is checked only Panorama will be located—the other two versions would be skipped.

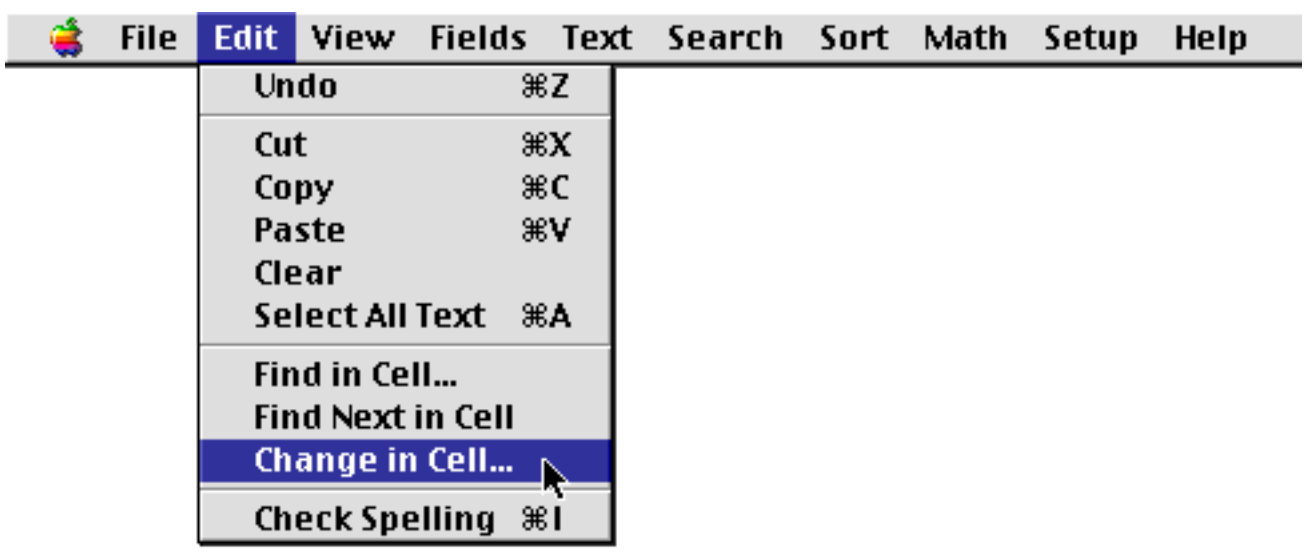
Use the Words Only option when you only want to locate occurrences of the text that are complete words, not part of a larger word. For example, if you search for the word head, the **Find in Cell** command will also locate the words header, headline, subhead, etc. However, if the Words Only option is checked, these additional variations will be skipped.

To find additional occurrences of the word or phrase within the Input Box, use the **Find Next in Cell** command in the Edit menu (not the Search menu).

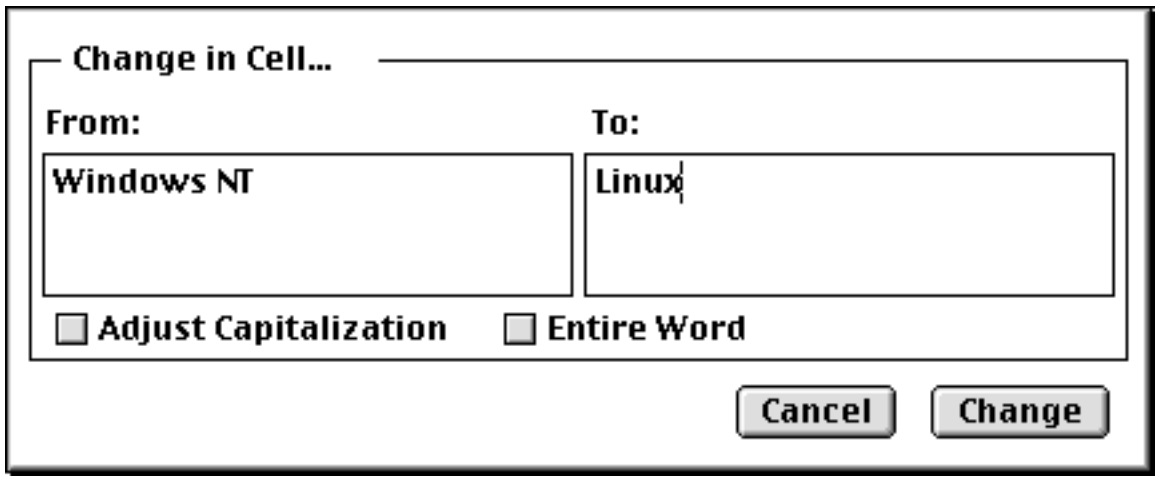
The **Find in Cell** command in the Edit menu searches only through the text in the Input Box. If you wish to search the entire database you must use the **Find/Select** command in the Search menu (see "The Find / Select Dialog" on page 369).

Replacing Words or Phrases Within a Cell

To replace a word or phrase within the Input Box use the **Change in Cell** command in the Edit Menu.



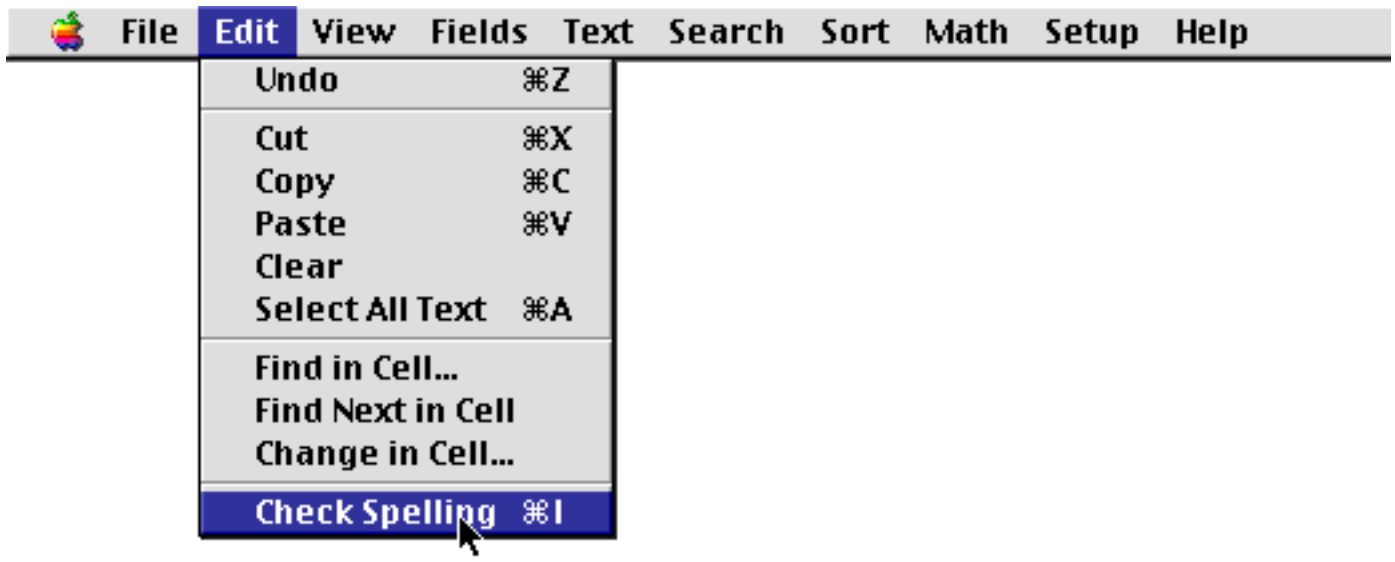
Make sure you use the **Change in Cell** command in the Edit Menu, not the **Change** command in the Search Menu. Enter the word or phrase you want to replace and the new word or phrase into the dialog box. Press the **Change** button or the **Enter** key to change all occurrences of the word or phrase within the Input Box.



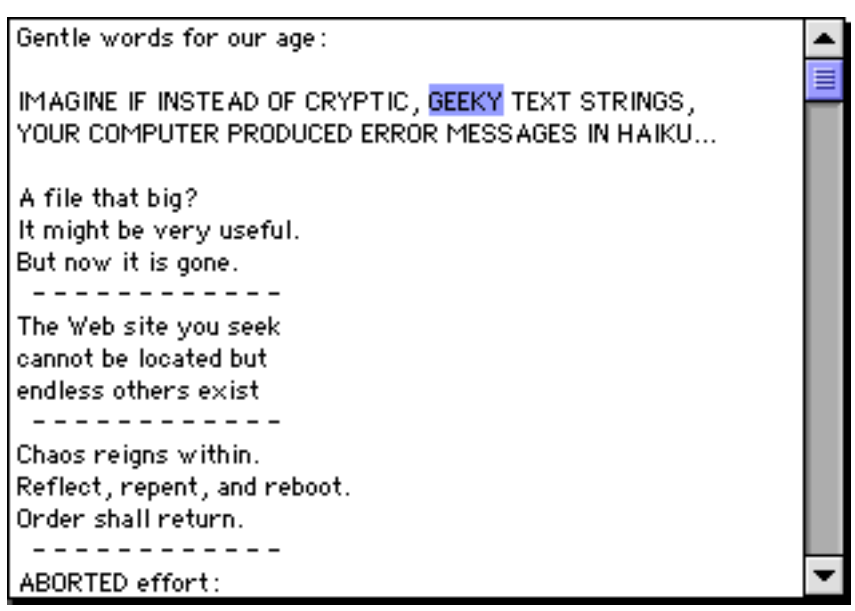
The **Change in Cell** dialog has the same Case Sensitive and Words Only options that appear in the **Find in Cell** dialog. These options work exactly the same way as described in the previous section.

Using the Spelling Checker within a Cell

If you have purchased Panorama's optional spelling dictionary, you can use the **Check Spelling** command (Edit Menu) to check the spelling of the text in the Input Box.



Starting from the currently selected spot in the text, the **Check Spelling** will scan the Input Box looking for spelling errors. If it finds a spelling error, Panorama will stop and highlight the misspelled word.



If the word is actually misspelled, you can correct the error. In the example above, geeky is probably correct, but is not in Panorama's dictionary. To resume scanning for more spelling errors, choose **Check Spelling** again from the Edit Menu.

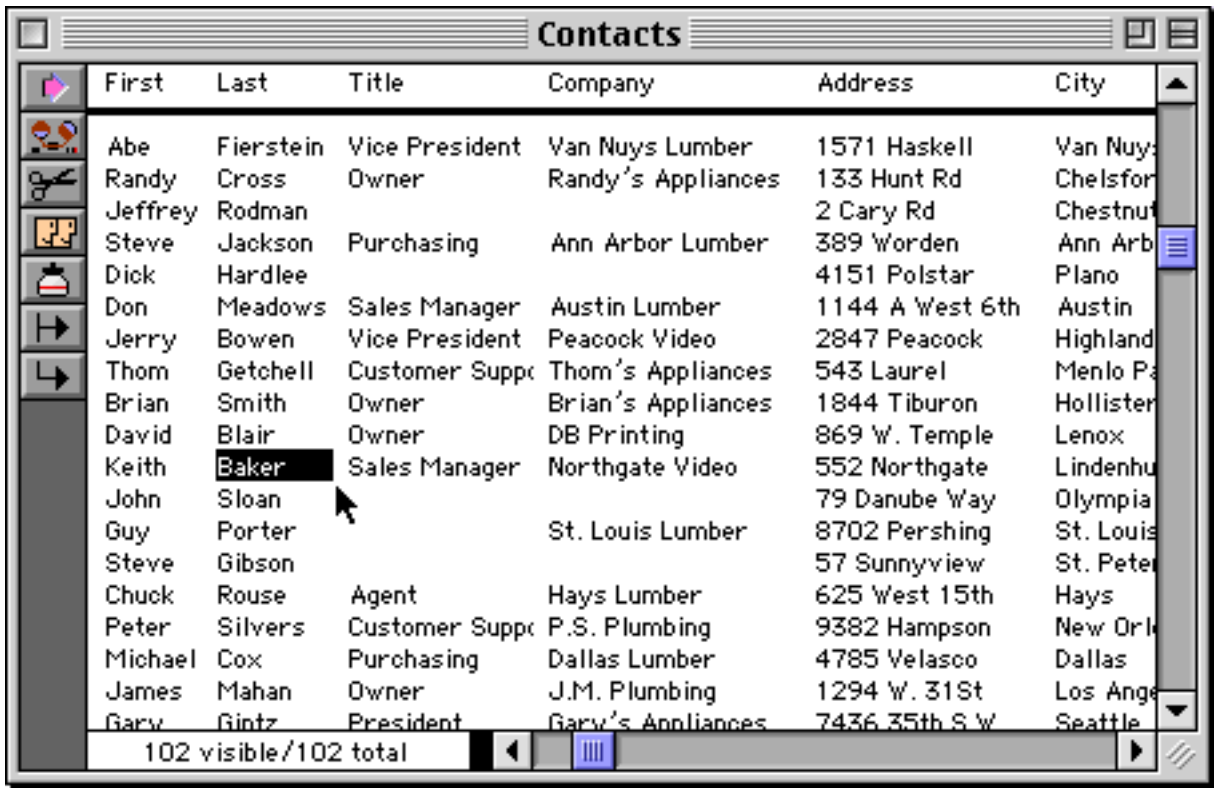
Chapter 8: Sorting



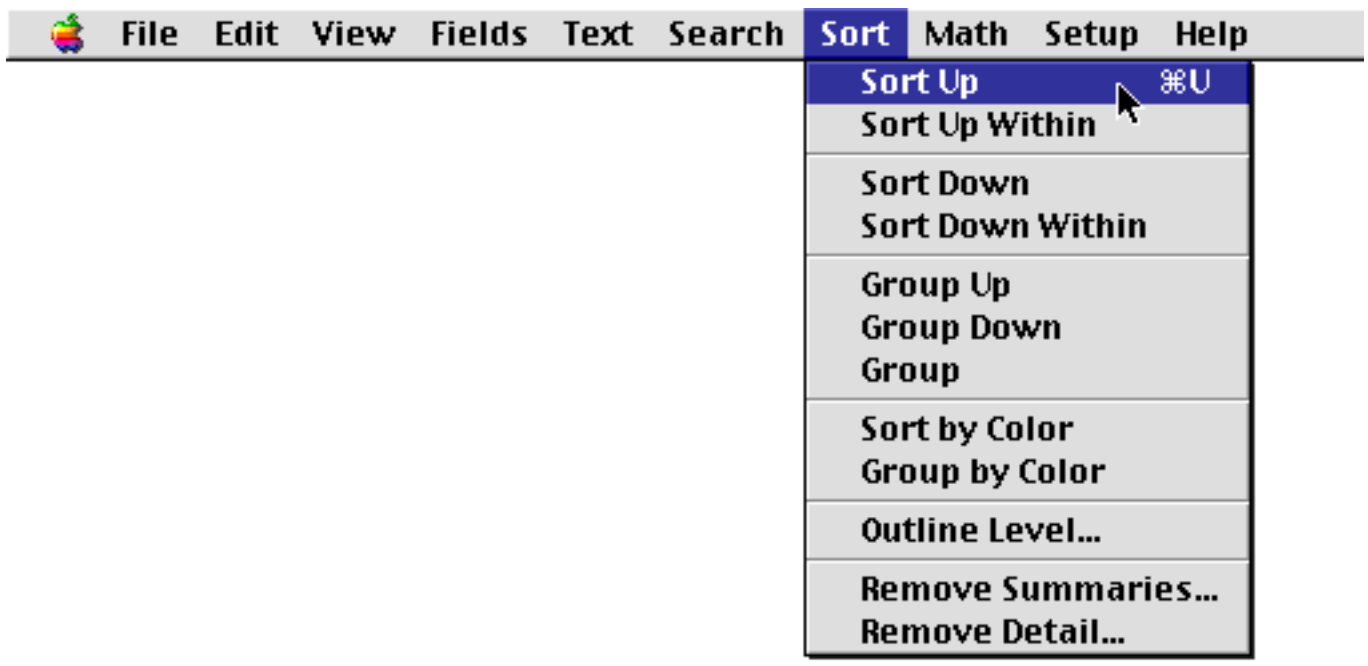
Most data is more useful when it is in some kind of order. Panorama's Sort Menu has several commands that can quickly sort your data into order.

Basic Sorting

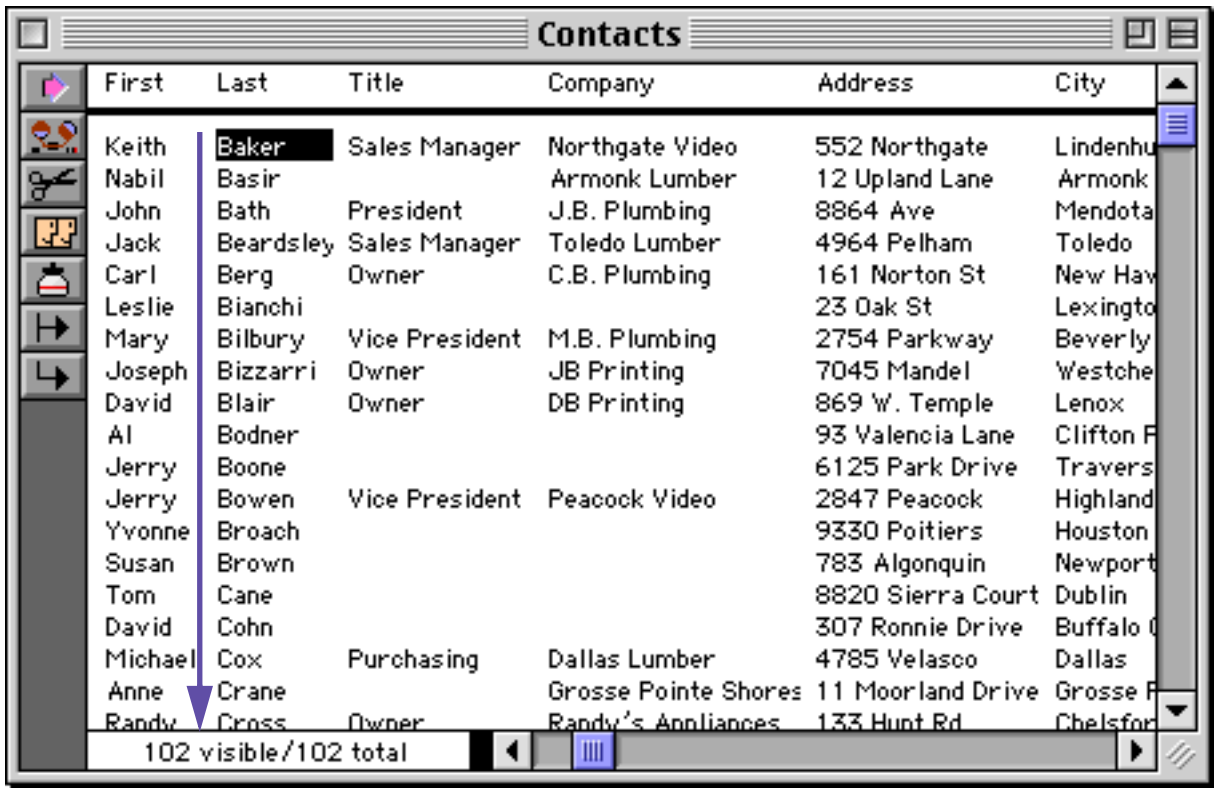
Before sorting, you must choose the field to be sorted. For example, you might want to sort a mailing list database by name, city, state, or zip code. To choose the field you want to sort, just click on it. In a data sheet or view-as-list window, you can click on any cell in the column. If you are using a regular form (individual pages), click on the field you want to sort. (You must click on a data cell, not on an auto-wrap text object containing the field.)



To sort in ascending order (A to Z) use the **Sort Up** command. To sort in descending order (Z to A) use the **Sort Down** command. The **Sort Down** command is especially useful with numeric data if you want to rank the numbers from largest to smallest.



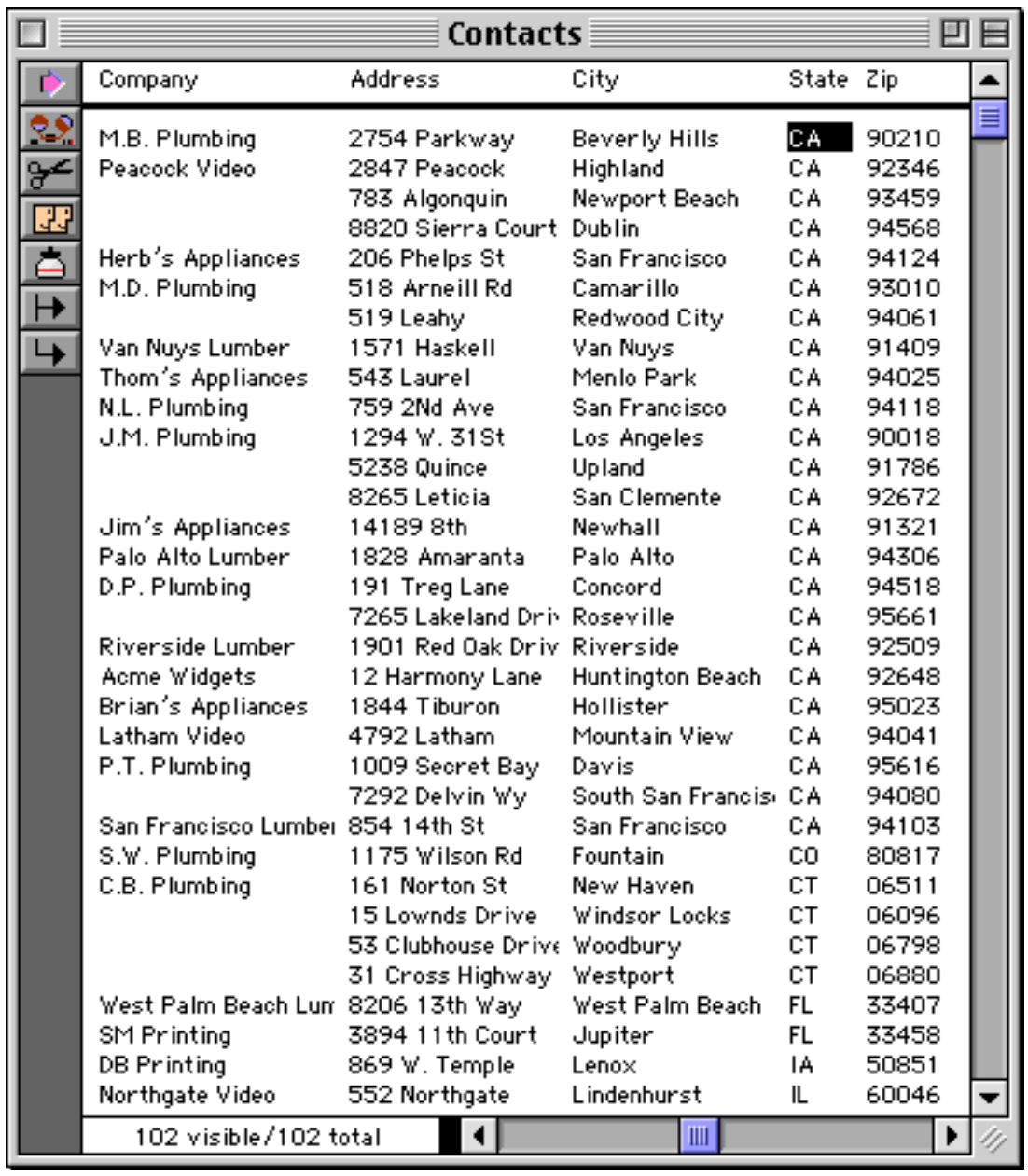
When the database is sorted, all the records are re-arranged. However the current record remains in the same spot in the window. For example, if you click on the name Zabriskie and then sort, the record containing Zabriskie will remain on the screen. The record will not be in the same spot in the file, however, because it is now near the end of the file with the other Z's. In this case the current record was Keith Baker, which is still the current record. However the database is now sorted from A to Z within the Last name field.



Sorting By More Than One Field

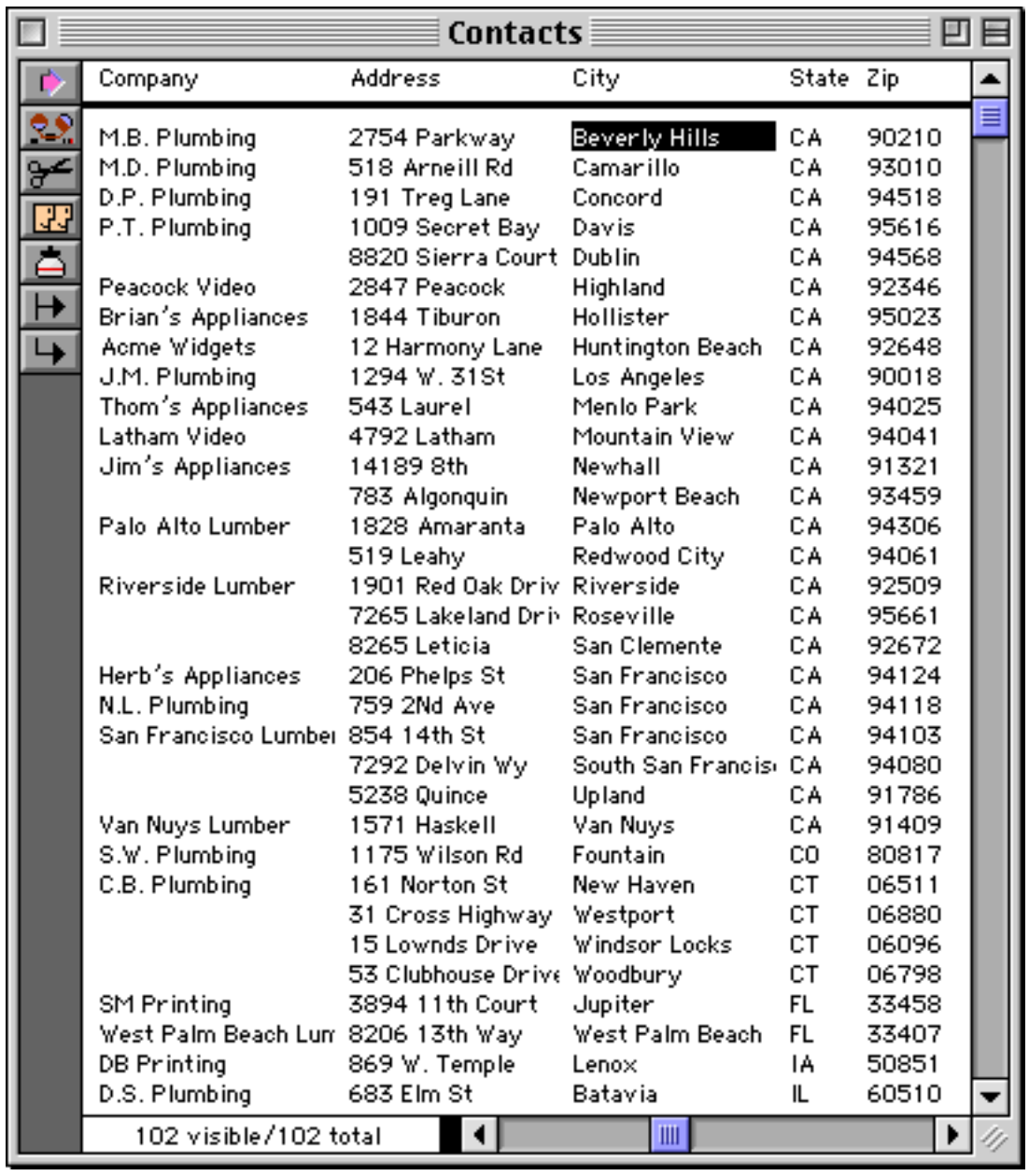
It is often necessary to sort by more than one field at a time. For example, you may need to sort both first and last names, or cities and states. After you have sorted the database once, you can use the **Sort Up Within** (or **Sort Down Within**) commands to sort by additional fields.

For example to sort by city and state, first sort by the state.



As you can see, the records are now sorted by state. However, the cities are not sorted within each state. For example, Davis, CA should appear before San Francisco, CA, but it doesn't. In fact, within each state the records aren't sorted at all, they are still in the order they were in before they were sorted.

To complete our two column sort, click on the City field and use **Sort Up Within** to sort the cities. The **Sort Up Within** command leaves the states in order but sorts the cities within each state.



As you can see, the cities are now sorted within California and Connecticut (and within all of the other states with more than one record as well).

The **Sort Within** commands can be used over and over again to sort 3, 4, or more fields within each other. Always start with the regular **Sort Up** or **Sort Down** commands, then use **Sort Up Within** or **Sort Down Within** to sort each of the additional fields. For example, if you look closely you will notice that the three San Francisco records in the window above are not sorted by zip code. If you wanted them to be, you could simply click on the Zip field and then choose **Sort Up Within** again. The records will still be sorted by state and by city within state, but now they will also be sorted by zip code within city as well.

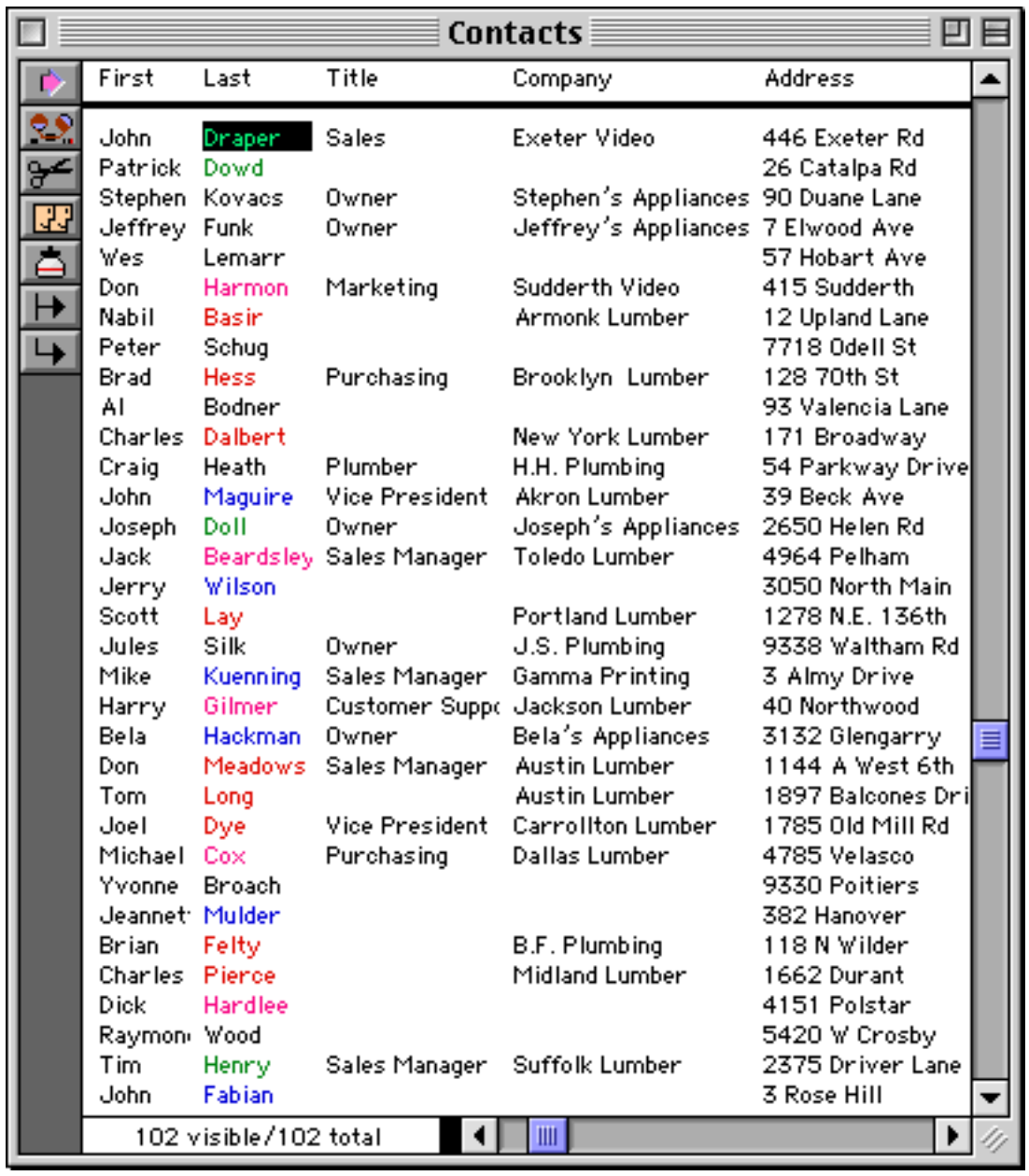


Note: There is an alternate way to sort multiple fields. This alternate method does not use the **Sort Within** command. Instead of sorting within, sort the fields in reverse order using the regular **Sort Up** or **Sort Down** commands. For example to sort by city within state, first click on the City field, then **Sort Up**, then click on the State field, and finally **Sort Up** again.

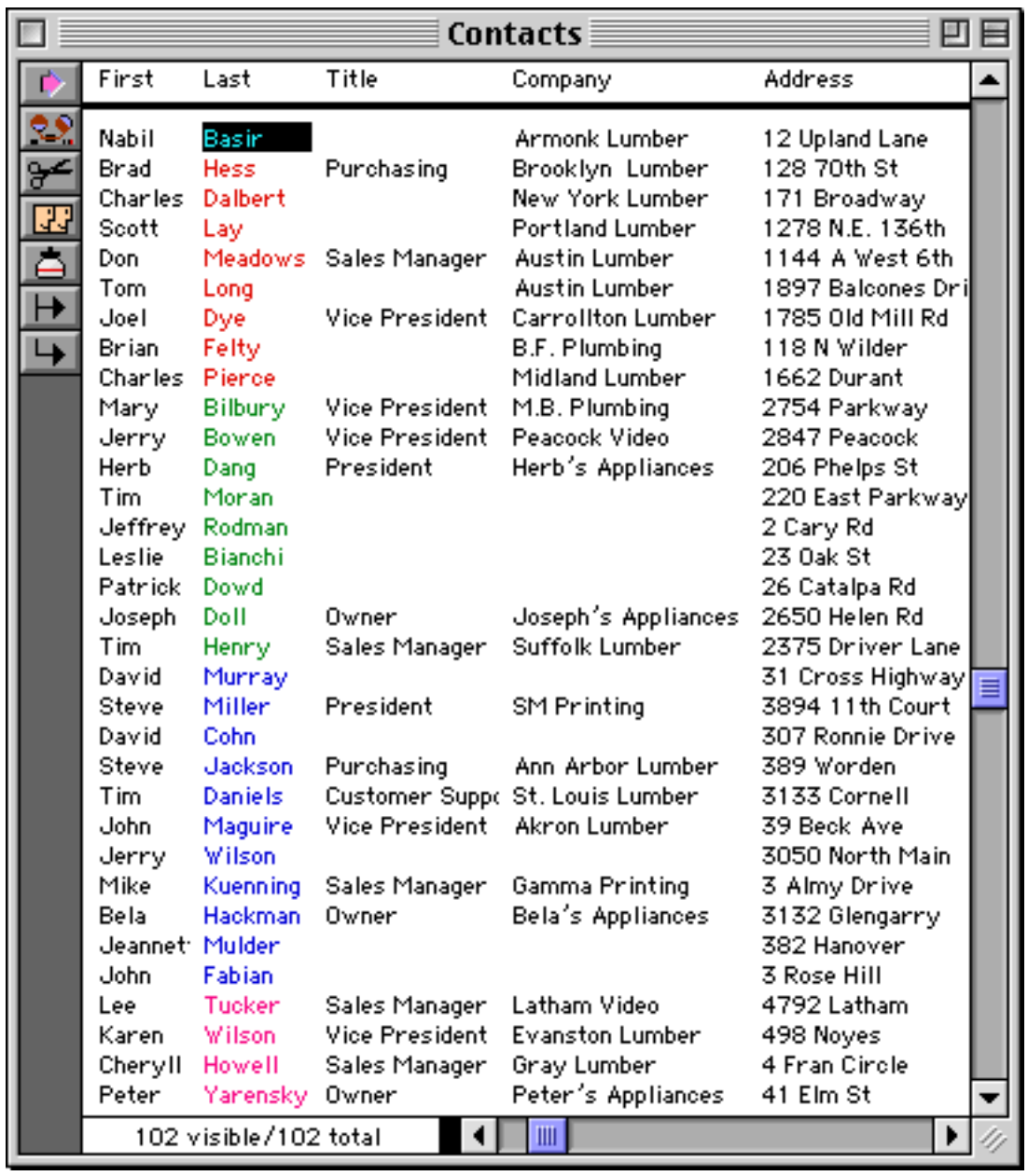
Sorting By Color

Usually the database is sorted according to the data in a field, but you can use the **Sort by Color** command to sort by the color of each cell in the field. Sorting by color can be useful if you have set up your database so that each color has a meaning. The sort order for colors is black, red, green, blue, cyan, magenta, yellow. See "Data Style and Color" on page 474 for details on how to assign a color to a cell.

Here is a database where each name has been assigned a color.



After **Sort By Color** the records are grouped together by color.



Undo Sorting

The **Undo** command will undo the effect of the last **Sort** or **Sort Within** command, putting the database back into the original order. Remember that only the last sort can be undone. Once you sort again the original order cannot be restored (unless you have saved the file on the disk in the original order, then you can use **Revert to Saved** to restore the order.)

Sorting Numbers and Dates

It is important to store numbers using the numeric data type, and dates using the date type. If you store numbers and dates using the text data type they will not sort correctly, as shown in this table.

Stored as Numeric (correct)	Stored as Text (incorrect)
9	6000
80	700
700	80
6000	9

If your numbers or dates are not sorting correctly, make sure they are stored using the correct data type. See "Numeric Data" on page 289 for more information on the numeric data type. See "Dates" on page 294 for more information on the date data type.

Sorting Right Justified Text

If your text is right justified, it will sort like a numeric field. In other words, 2 will sort before 10, and B will sort before AA. The actual sorting rules for right justified text are—1) short data sorts before longer data (B before AA) and 2) if two data items are the same length, they will be sorted in alphabetical order (AA before BA).

Sorting Selected Data

Sorting is not affected by the **Find/Select** command. The sort commands always sort the entire database—not just the selected records. When the invisible data is selected again you will see that it is sorted properly within the rest of the data.

Sorting Within Groups

Later you'll learn how Panorama can organize a database into groups, with summaries for each group. (See "3-Step Summarizing" on page 393.) If you attempt to sort your database after it has been grouped, Panorama will automatically sort the data within the groups instead of sorting the entire database. If you want to sort the entire database you must remove the groups with the **Remove Summaries** command (Sort Menu).

Sorting Choices

Data stored using the choices data type is sorted according to the order of the choice list in the design sheet. If you want the choices to be sorted alphabetically, you must make sure that the choice list is in alphabetical (A-Z) order. The **Automatic Choices** command does this for you.

Sometimes you may wish to sort the choices in a different order. For example, Olympic medals should be sorted in the order Gold, Silver, Bronze instead of alphabetically (Bronze, Gold, Silver). If you need the choices to sort non-alphabetically, simply set up the choice list in the order you want.

Warning: If you use a non-alphabetic choice order, Panorama cannot correctly sort exceptions. Panorama will attempt to place the exceptions in alphabetical order, but if the choice list is not in alphabetical order that may be impossible. The final order is not predictable. If the choice list allows exceptions it should be in alphabetical order.

See "Choices" on page 298 for more information on the choice list and the choice data type.

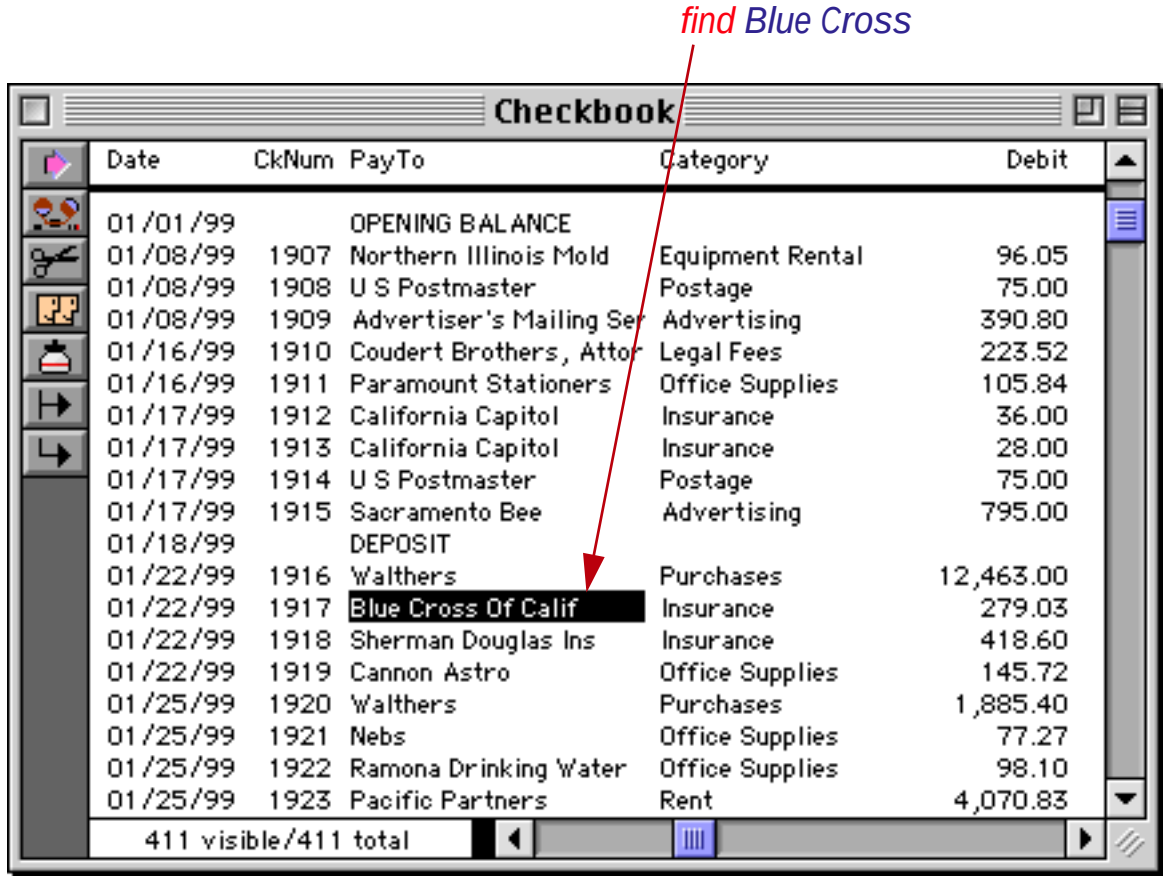
Chapter 9: Searching and Selecting



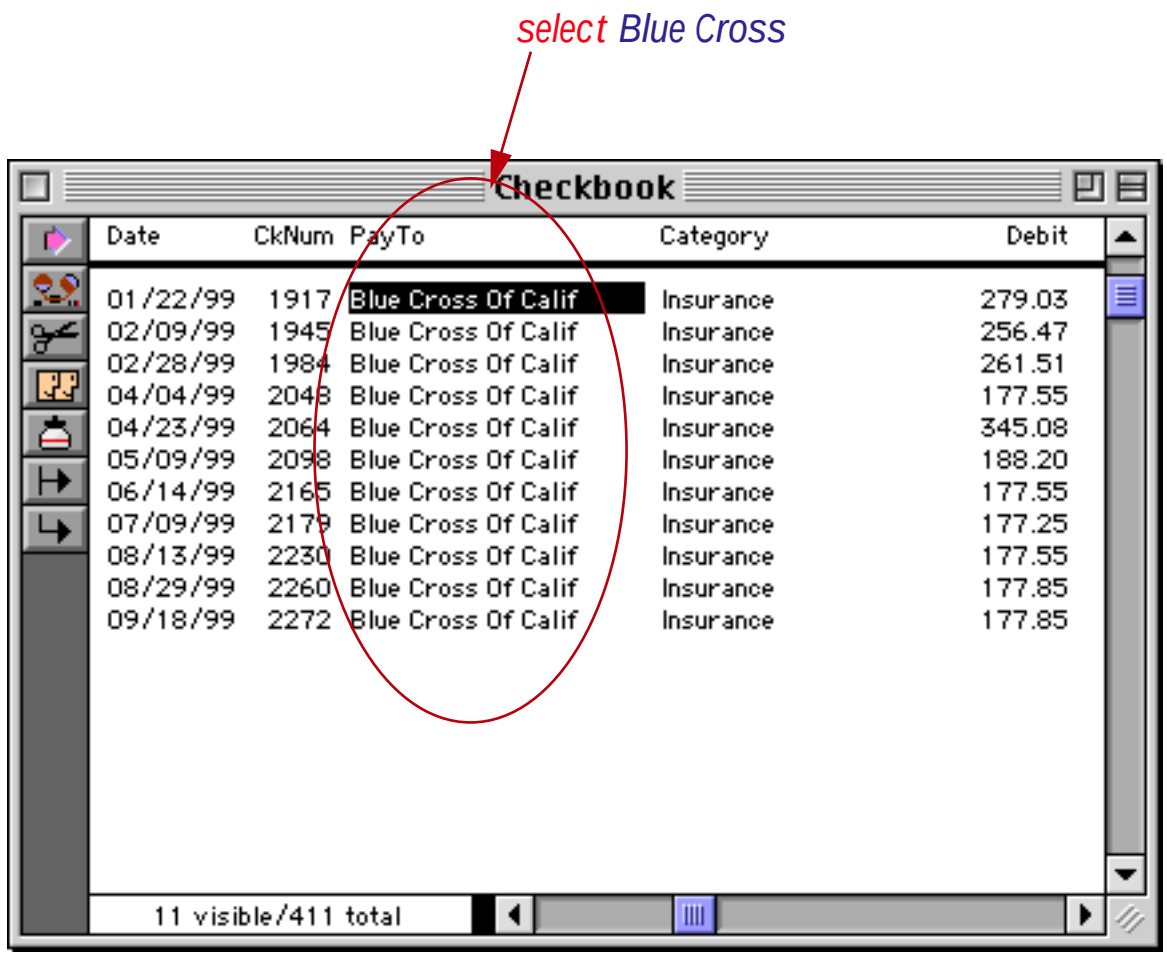
A Panorama database may contain dozens, hundreds, or even thousands of records. Finding a particular piece of information could be like locating a needle in a haystack. Fortunately Panorama can easily locate information for you.

Finding vs. Selecting

Panorama has two ways of locating information, finding and selecting. Finding is much like looking up a name in a phone book—Panorama points out the location of the information you are looking for. For example, you might ask Panorama to find a phone number or a price in a catalog. Panorama will locate the information, and position the database to that spot. In this example we've asked Panorama to find Blue Cross.



Selecting is like creating a whole new phone book containing only the information you are looking for. All the selected data remains visible, while everything else temporarily vanishes. For example, you might ask Panorama to select all customers that have purchased from you in the last six months, or all transactions over $250,000.

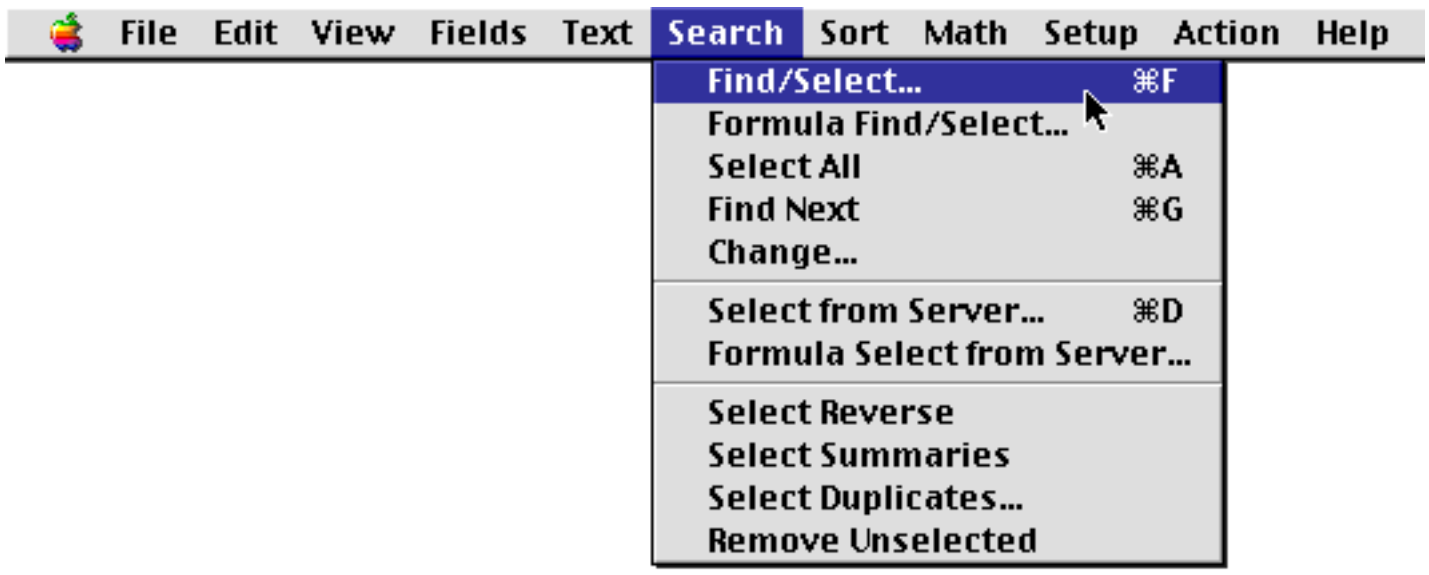


Deciding whether to find or select is your choice. Usually find is used when you want to locate a specific item like an address or price, while select is used when you want to locate a set of information. You can also combine the two techniques—for example, first select a subset of the database, then find a specific item within that subset.

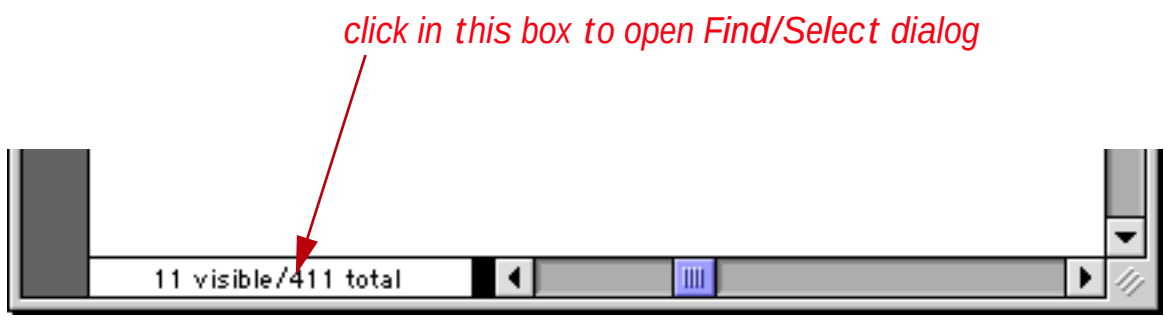
Note: Some other database programs have a “Find” command that actually does the same thing as Panorama’s Select command (for example FileMaker). These programs do not have a true find capability like Panorama.

The Find/Select Dialog

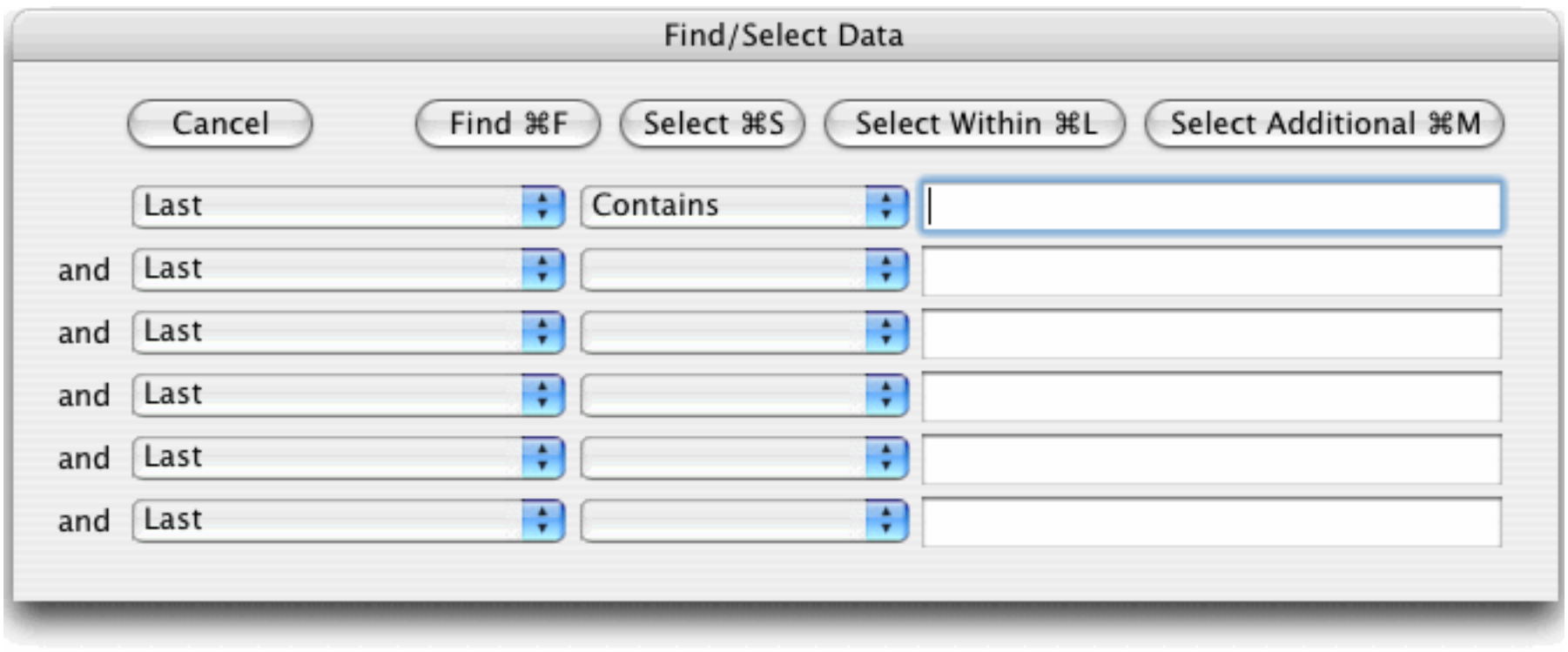
Panorama locates information by scanning through the entire database looking for data that matches your criteria. The **Find/Select** dialog (Search Menu) allows you to specify three criteria for locating information—the field containing the data, the kind of match you want (exact match, greater than, etc.) and the match value (the data you are looking for).



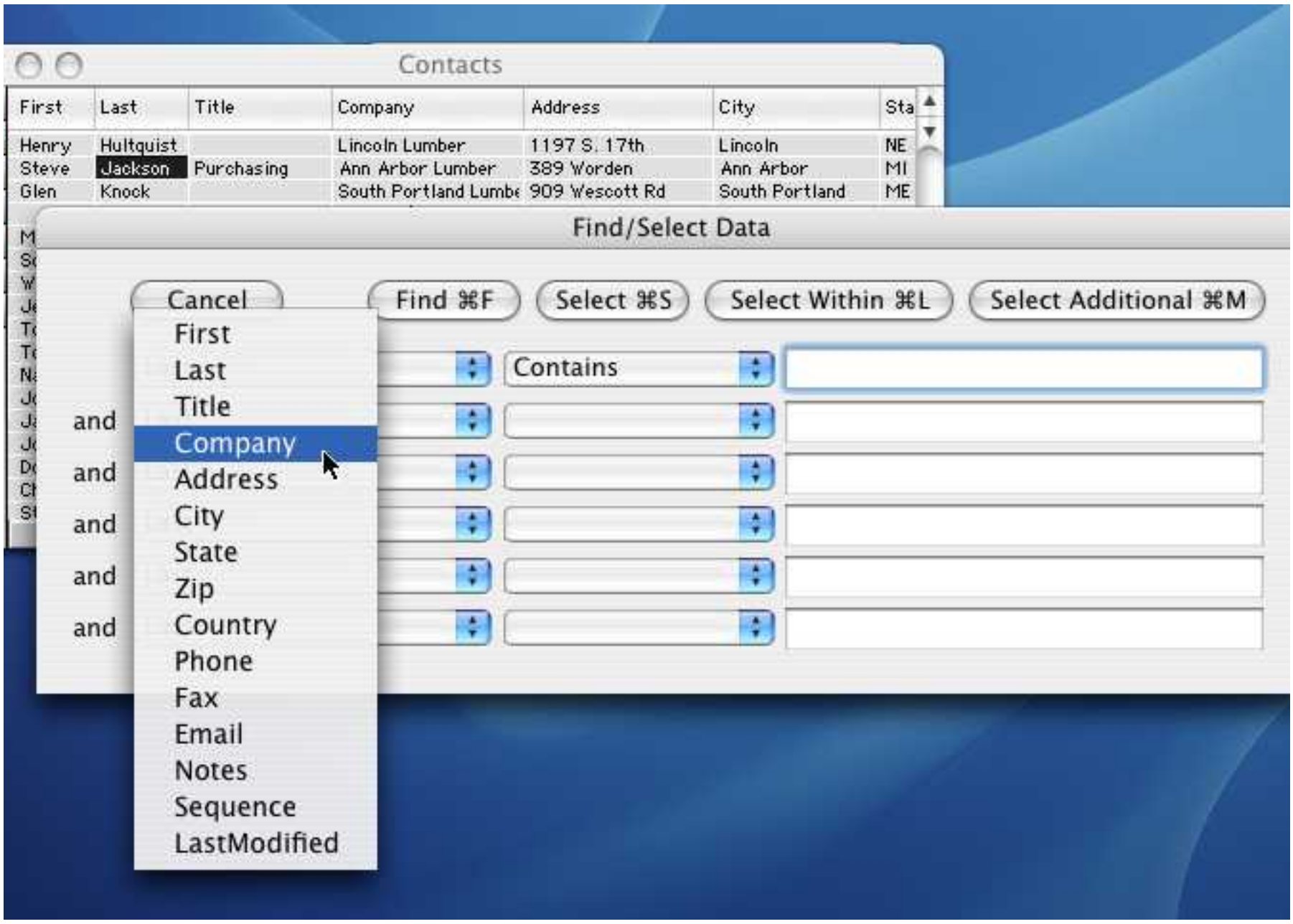
Tip: You can also open the **Find/Select** dialog by clicking on the record count displayed in the lower left hand corner of the window.



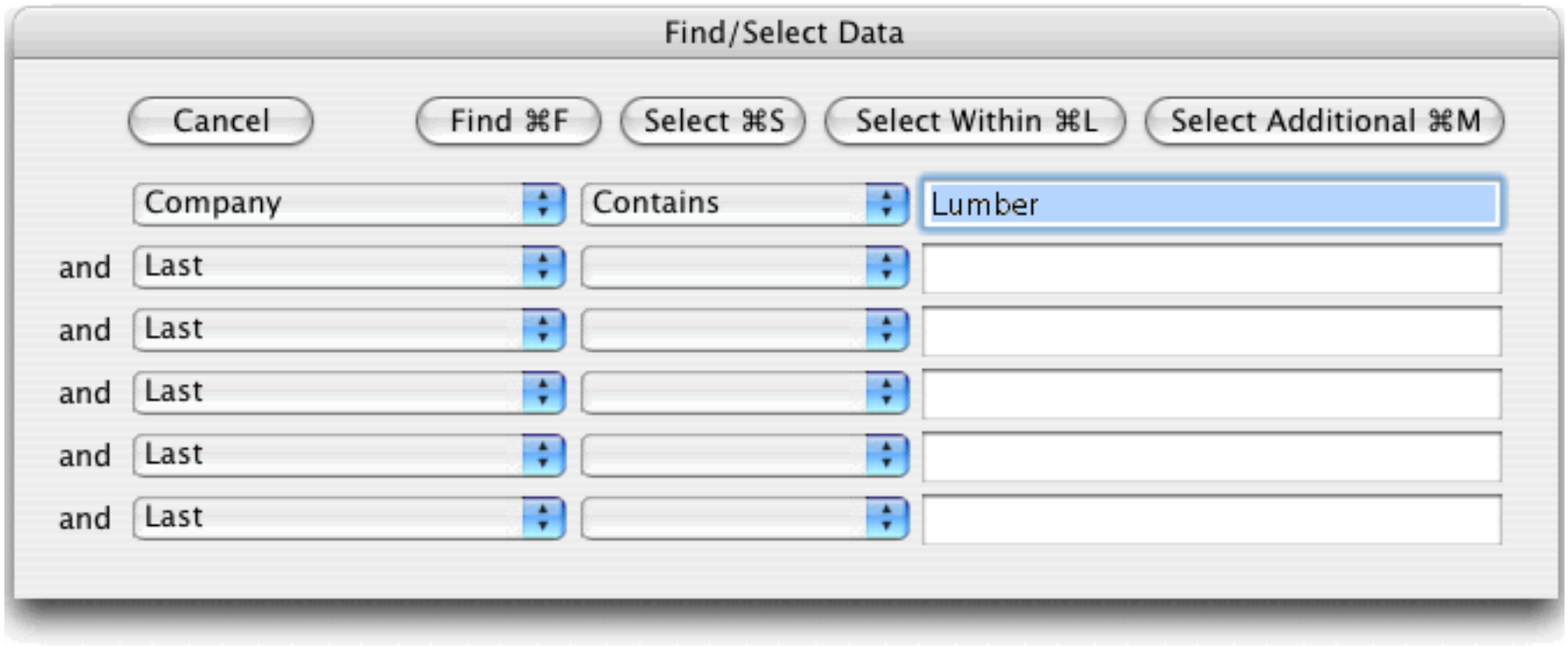
Here is what the **Find/Select** dialog looks like when you first open it.



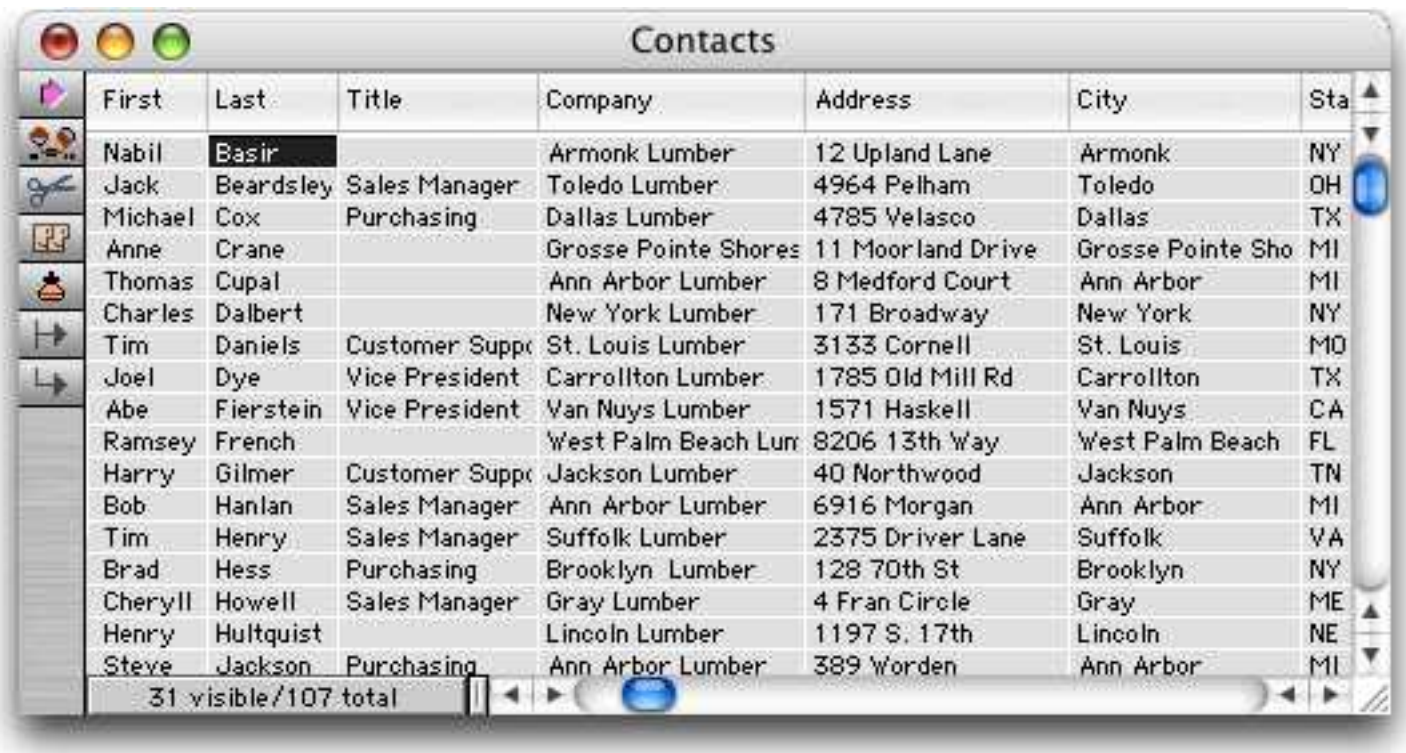
The pop-up menus on the left side of the dialog tells Panorama what fields contains the data you are looking for. For example, if you are looking for a company, you would tell Panorama to look in the Company field. The **Find/Select** dialog initially assumes that you want to locate information in the current column.



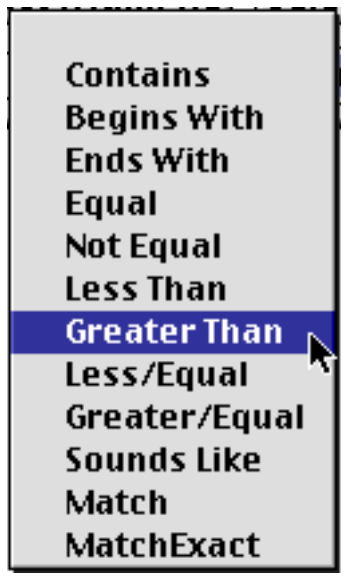
Type the data you are looking for into the box on the right side of the dialog. This data is called the match value. For example if you are looking for a person, that person's name would be the match value. If you are looking for transactions over $250,000 the match value would be 250000.



If you press the **Select** button (or the **Enter** key) Panorama will select all records with a company name containing Lumber.



The pop-up menu in the middle of the dialog controls how Panorama looks for a match. There are ten different choices for identifying a match.



Contains — Any data cell that contains the match value will be identified as a match. For example, if you ask Panorama to locate cities containing an, it will locate cities like Anaheim, Lansing, Los Angeles, and San Jose since they all contain an. Notice that capitalization is ignored, so an, An, AN, and aN are all acceptable matches.

Not Contains — Any data cell that does not contain the match value will be identified as a match. For example, if you ask Panorama to locate phone numbers not containingcontaining (714) it will locate phone numbers in other area codes. Capitalization is ignored, so an, An, AN, and aN are all equivalent as far as this option is concerned.

Begins With — Any data cell that begins with the match value will be identified as a match. For example, if you ask Panorama to locate states beginning with co, it will locate states like Colorado and Connecticut. Capitalization is ignored, so co, Co, CO, and cO are all acceptable matches.

Ends With — Any data cell that ends with the match value will be identified as a match. For example, if you ask Panorama to locate baseball teams ending with sox it will locate both Red Sox and White Sox. Capitalization is ignored, so sox, Sox, SOX, and sOx are all acceptable matches.

Equal (=) — Any data cell that exactly matches the match value will be identified as a match. An exact match means just that. The spelling, punctuation, and capitalization must be exactly the same—for example red will not match RED or Red.

Not Equal (≠) — Any data cell that does not exactly match the match value will be identified as a match.

Less Than (<) — Any data cell that is less than the value in the box on the right will be identified as a match.

Greater Than (>) — Any data cell that is greater than the match value will be identified as a match.

Less Than or Equal (≤) — Any data cell that is less than or equal to the match value will be identified as a match.

Greater Than or Equal (≥) — Any data cell that is greater than or equal to the match value will be identified as a match.

Sounds Like — Any data cell that “sounds like” the match value will be identified as a match. Panorama uses a special algorithm to determine which values sound like the match value. This algorithm is not perfect, but it does work pretty well. For example, if you are looking for someone named Luboviski but you are not sure if it is spelled with an i, ie, or y, the sounds like match will save the day.

The sounds like match can be used with more than one word at a time. For example, if you are searching through a video rental database for the movie Escape from New York, the sounds like algorithm will find it even if it is misspelled Escapade from New York. If any word in the match value sounds like any word in the data cell, the data will be identified as a match.

Note: If two words do not start with the same letter, the sounds like algorithm will not think they sound alike. For example, sounds like does not think that Chris and Kris sound alike.

Match— This option allows you to create a "pattern" for comparing data. The pattern allows you to set up very flexible "wildcard" matches where some characters must match but others don't have to. The pattern must contain one or more "normal" characters (letters, numbers, punctuation, etc.) and also may contain one or more of the wildcard characters ? (question mark) and * (asterisk). The ? wildcard character will match any character in this position. The * wildcard character will match any number of characters in this position.

A few examples should help to make the operation of the wildcard characters within the pattern clear. Suppose you want to find all records where the first three digits of the zip code are 926, and you don't care what the last three digits are. The pattern will be `926??`. This pattern will match any five digit zip code that begins with 926. It will not match if there are less than or more than 5 characters in the zip code.

If the pattern is changed to `926*`, Panorama will match with any zip code that begins with 926, no matter what the length is. It could be three digits long or thirty — Panorama doesn't care and will say that it matches as long as it starts with 926.

By changing the pattern to `926??*` we tell Panorama to match any zip code that starts with 926 and is at least five characters long. The zip code could be 5, 6, 7, or 70 characters long, but will not match if it is only 3 or 4 characters long.

If you wanted to select only 9 digit zip codes we could use the pattern `?????-????`. This will match any 10 character long string with a - (dash) in the sixth position.

Suppose that you wanted to find everyone in your database with the last name Johnson and the first initial J. Assuming that the first and last names are stored in a single field, you could use the pattern `j*johnson` to locate the person (or persons) you are looking for. The match option doesn't care about upper or lower case, so this pattern would match Jerry Johnson, jim johnson, or JOHN JOHNSON. (It will also match weird data like j346ujohnson or j@#opcjohnson, so take care to watch for unexpected matches.) If you want upper and lower case treated as different characters use the matchexact option (see below).

MatchExact— This option is the same as the match option (see above), except that any letters in the data must exactly match the pattern, including upper vs. lower case. For example, if the pattern is `J*Johnson`, names like Jerry Johnson will match, but JERRY JOHNSON will not match.

Locating Dates by Month, Quarter, or Year

The **Find/Select** dialog allows you to specify dates by day, month, quarter, or year. These options only work if the dates are stored using the date data type, not text. (See "Dates" on page 294.)

To specify an individual day, simply type the date in the format mm/dd/yy or mm/dd. If you leave off the year, Panorama will round the date to the nearest year.



To specify a month, you must enter the month in the format MONTH YY or MON YY (for example September 1998 or Aug 01). You cannot specify a month using the format mm/yy, because Panorama cannot distinguish this from mm/dd.



To specify a quarter, use the format QqYY (for example 1q99 or 3Q2006).



To specify an entire year, use the format YY or YYYY (for example 90 or 1994).



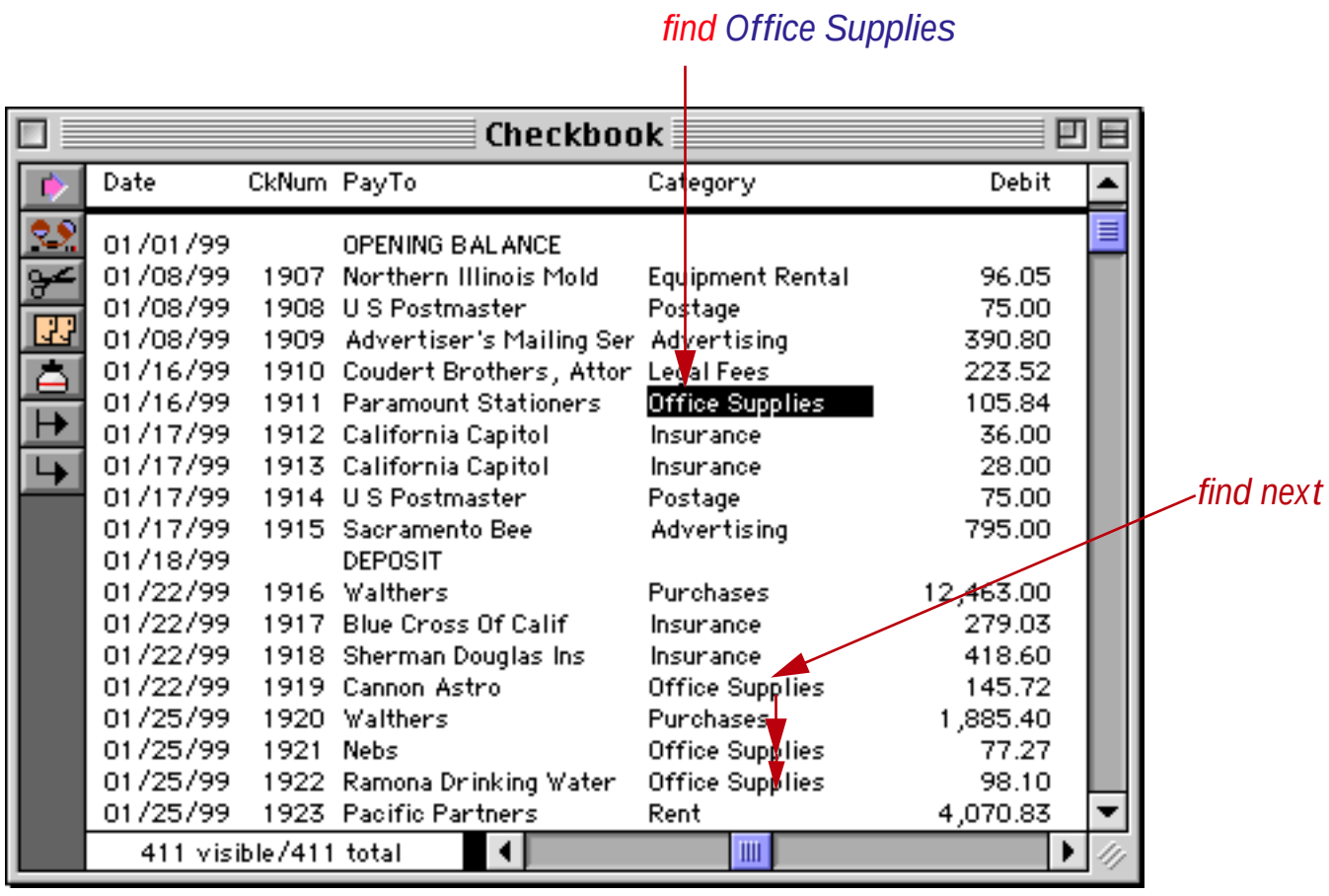
If you specify a month, quarter, or year, you must use the Equals (=) type of match.

Find and Find Next

Once you have specified what you are looking for, you can either find or select the information (See "Finding vs. Selecting" on page 367). Press the **Find** button to find the information.

When you press the **Find** button, Panorama will go to the top of the database and start scanning. When Panorama finds a data cell that matches what you are looking for, it stops scanning and displays the information it has found.

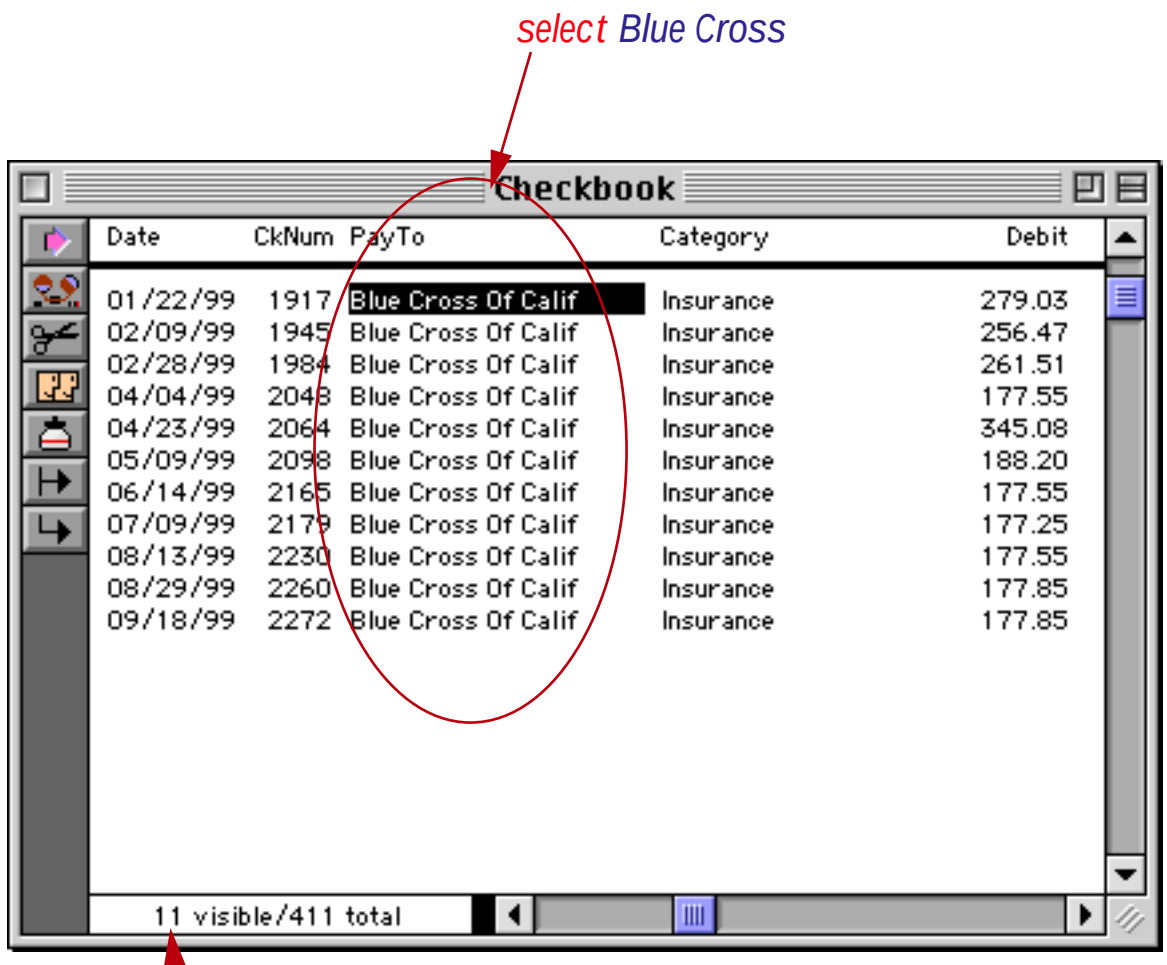
To resume scanning for additional matches, use the **Find Next** command. If there aren't any more matches, Panorama will beep.



You can use the **Undo** command to move backwards through the items you have already found. You are not limited to just one **Undo**. You can keep undoing until you reach the first matching item in the file.

Select

To select a subset of the database, press the **Select** button. Panorama will scan through the entire database and select the records that match the criteria you have specified. The selected records remain visible, while the records that do not match temporarily vanish. Panorama displays the number of selected records in the lower left hand corner of the window.



To restore the invisible records, choose the **Select All** command from the Search Menu.

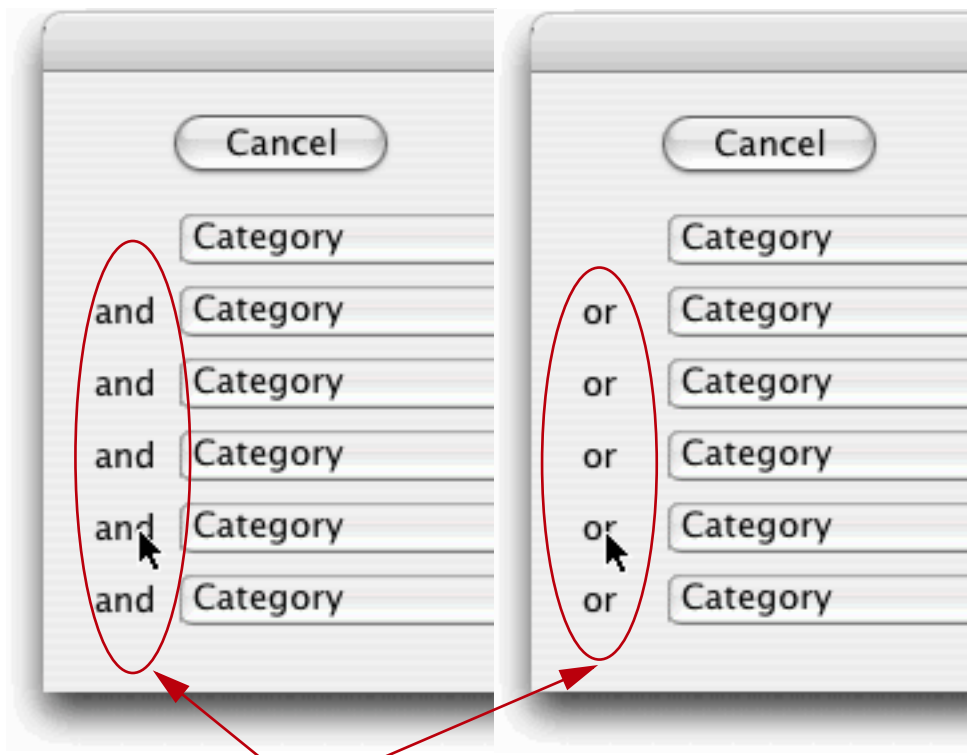
To make a different selection, simply use **Find/Select** again. The original selection will vanish and the new selection will become visible. (You do not need to choose **Select All** before selecting another subset.)

Note: If your database is large and you have selected only a few records, you may find that Panorama seems sluggish. Remember, Panorama may be skipping over hundreds of invisible records that are between the visible records on the screen. When you use **Select All**, Panorama's normal blazing speed will return.

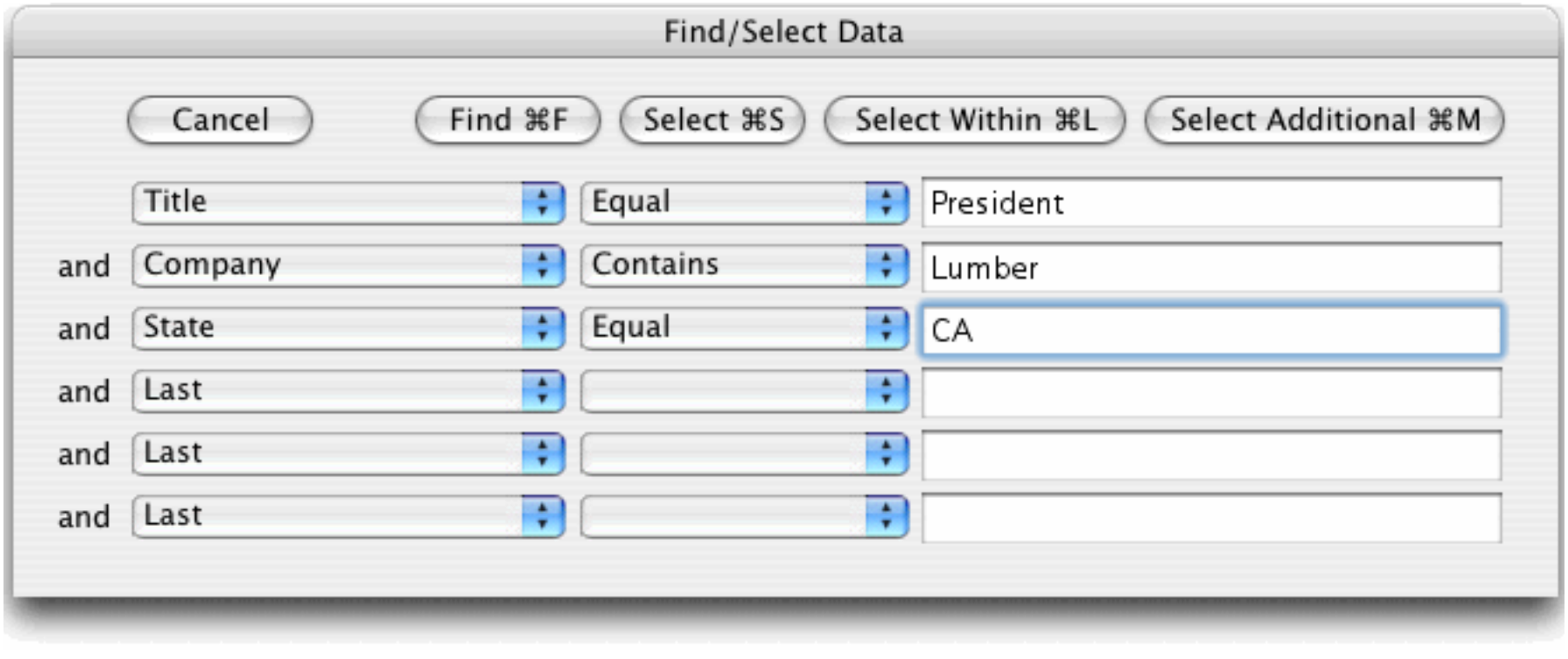
Note: After you have selected a subset of the data, you may find that you cannot move the data sheet scroll bar to the very top or very bottom. This will happen if the first or last record is not one of the selected (visible) records.

Multiple Find/Select Criteria

The **Find/Select** dialog can be expanded to allow up to six concurrent match criteria. For example, you can select all names in Oregon or Idaho or Montana, or you could select all transactions that are greater than $10,000 and less than $25,000. (However you cannot combine both **and** and **or** at the same time with the **Find/Select** dialog, to do that you must use the **Formula Find/Select** dialog — see "Formula Find/Select" on page 387). The six separate match criteria can be combined with and or or. To toggle between and and or, click on the word and or or on the left edge of the dialog.



When the criteria are combined with and, everything must match. For example, the dialog shown below could be used to select all presidents of lumber companies within California.



When the criteria are combined with or, Panorama will declare a winner if anything matches. For example, the dialog shown below tells Panorama to locate anyone with the words President, Manager, or Agent in his or her title.



The **Find/Select** dialog does not let you mix and and or within a single selection. See "Formula Find / Select" on page 387 if you need to mix and and or within a single selection.

Select Within

The **Select Within** button allows you to select a subset of the currently selected records (The normal **Select** button selects a subset of the entire database.) For example, if you have already selected transactions over $5,000, you can use **Select Within** to select only those transactions in California. The result would be the subset containing all California transactions over $5,000.

Of course, you could obtain the same effect by combining multiple criteria (and) in the first place (see the previous section). But you don't always know in advance exactly what you are looking for. The **Select Within** button allows you to whittle your data down gradually until you've extracted just the nugget of information you really need.

Select Additional

The **Select Additional** button allows you to select a superset of the currently selected records. For example, if you have already selected names in Ohio, you could use **Select Additional** to also select names in Illinois. The result would be the subset containing all names in either Ohio or Illinois.

Of course you could obtain the same effect by combining multiple criteria (or) in the first place (see above). But you don't always know in advance exactly what you are looking for. The **Select Additional** button lets you assemble the pieces of information you need piece by piece.

Select Reverse

The **Select Reverse** command reverses selected and deselected records. For instance, if you have selected all transactions greater than $600, the **Select Reverse** command will select all transactions less than or equal to $600.

Undo Select

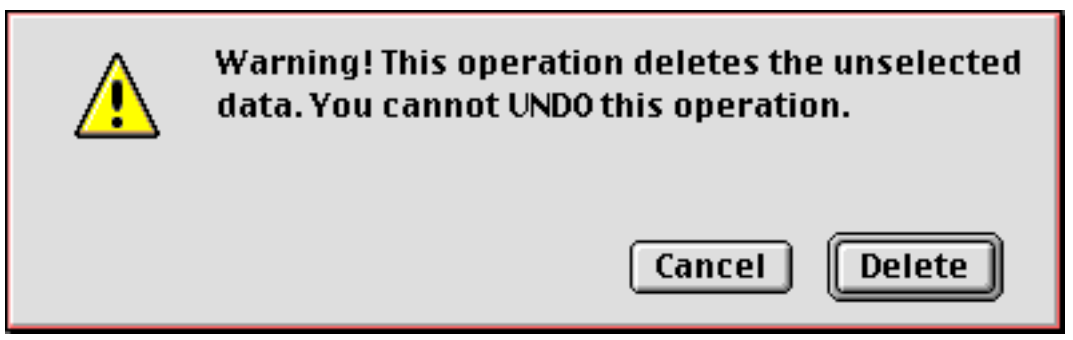
The **Undo** command can reverse the effects of the last 16 select operations, including **Select**, **Select Within**, **Select Additional**, and **Select Reverse**. As long as you do not use any other commands or tools, you can use the **Undo** command up to 16 times in a row.

The quickest way to select the entire database is the **Select All** command.

Permanently Removing Unselected Data

Unselected data is hidden, but it is still part of the database. You can restore the invisible data with the **Select All** command. Sometimes, however, you may wish to free the memory occupied by the hidden records. The **Remove Unselected** command in the Search Menu permanently removes the unselected records from the database.

Before **Remove Unselected** actually erases the unselected data, it asks you to confirm that you really want to proceed. Panorama doesn't want you to accidentally delete hundreds or thousands of records. Be careful, because the **Remove Unselected** command cannot be reversed with the **Undo** command.

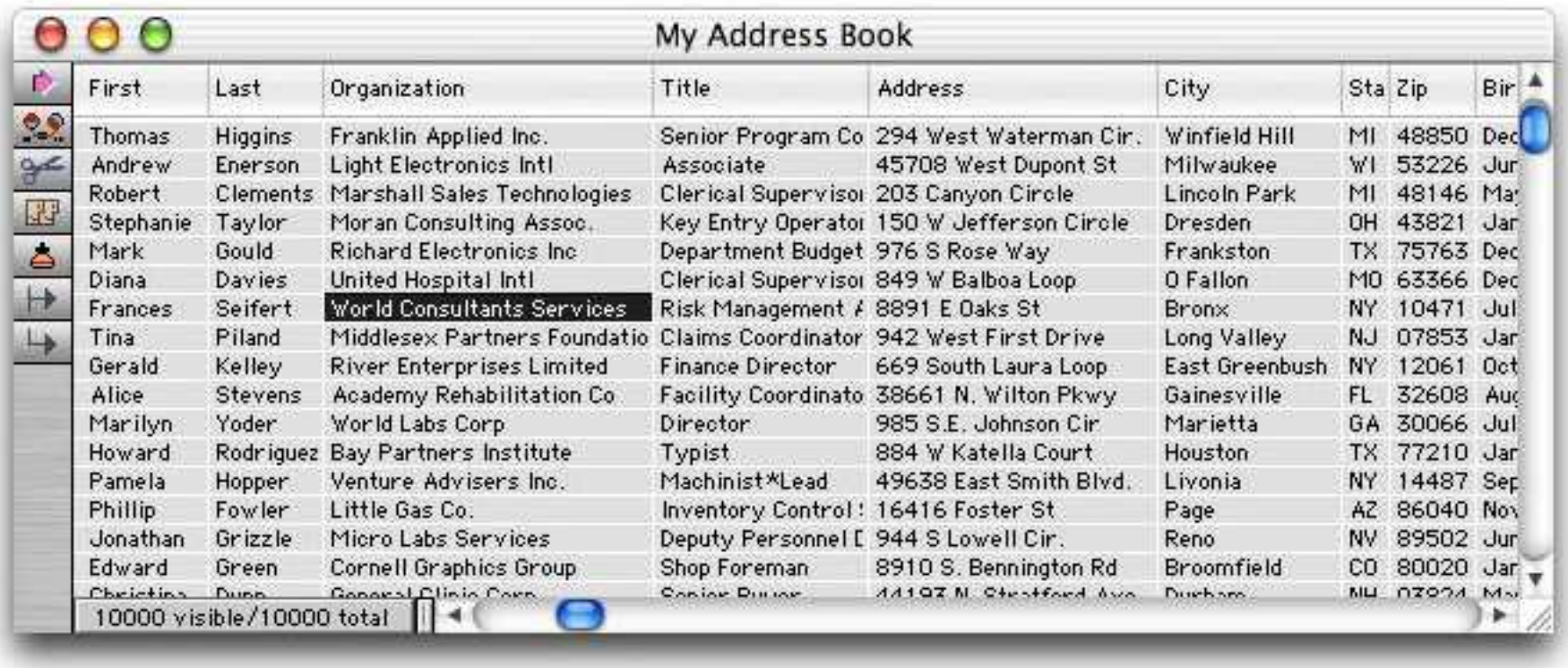


If you have saved a copy of the data on disk prior to using **Remove Unselected**, you can still recover the data with the **Revert to Saved** command (until you save again). Since saving the file after using **Remove Unselected** eliminates your ability to recover the data, you may not want to use the Auto-Save option when you are using the **Remove Unselected** command. If Auto-Save is on and you use **Remove Unselected**, Panorama will ask if you would like to temporarily disable Auto-Save. If you do disable Auto-Save, it will remain disabled until you manually save the file with the **Save** command. (See "Auto-Save" on page 133.)

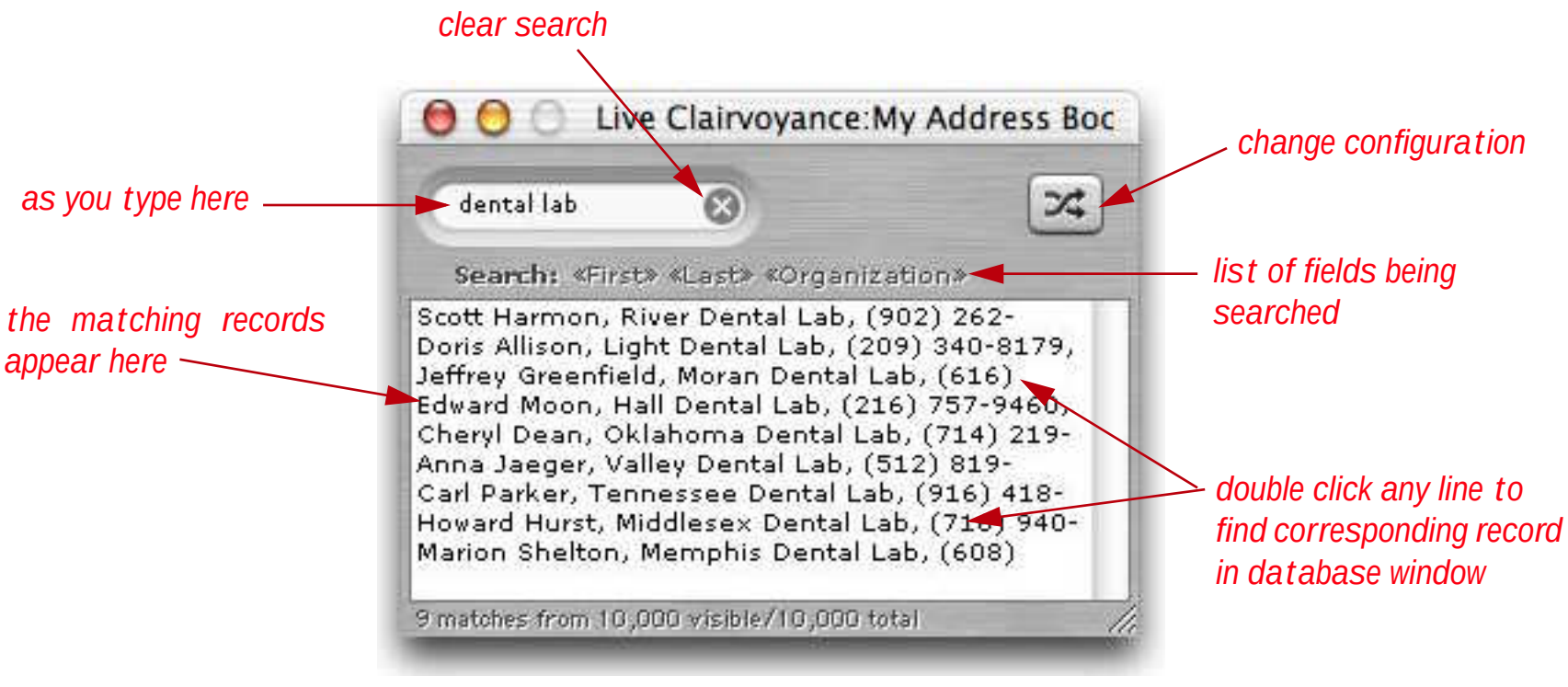
The Live Clairvoyance™ Wizard

The **Live Clairvoyance**™ wizard allows you to perform "live" searches on any Panorama database. The search results are updated dynamically as you type, allowing you to "hone in" on just the information you are looking for. The search may include multiple fields or even all fields in the database being searched. (If you've used the search box in iTunes you'll find the operation of this wizard familiar.) Using the Live Clairvoyance wizard doesn't require you to do any programming or make any modifications to your existing databases.

The easiest way to understand the Live Clairvoyance is to see it in action. For example, suppose you wanted to search the database My Address Book, which contains 10,000 names, address, phone numbers, birthdays, etc.



To search this database, choose **Live Clairvoyance** from the Wizard menu. The database that is active when you choose this command becomes the target database (the database that will be searched). Once the wizard is open simply type the word or phrase you want to search for into the search box at the top of the wizard window. As you type each character the list will update to show the records that match the word or phrase you have typed so far.



To clear the search and start over, press the button.

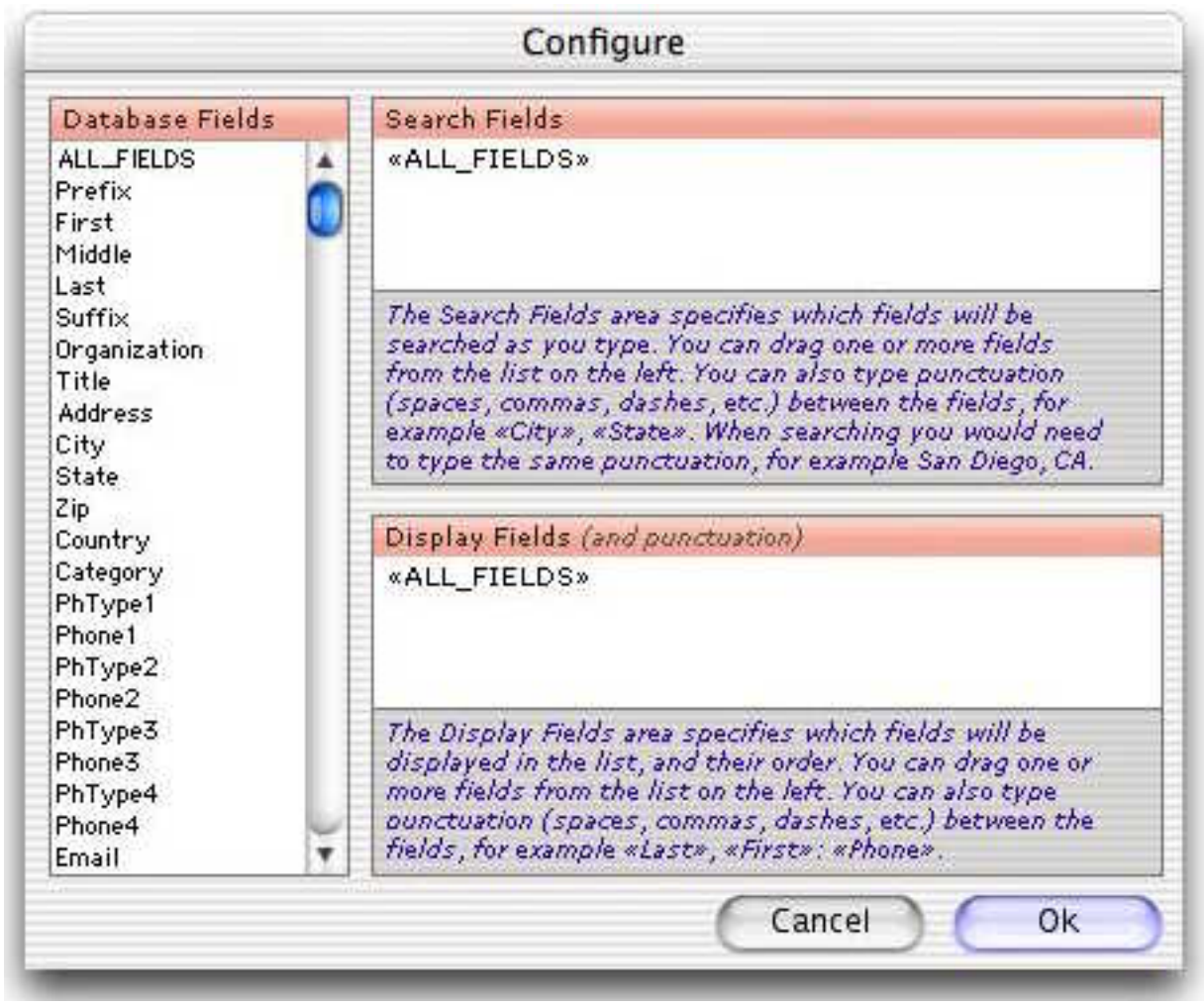
To jump to any record you have found in the original database, double click on that record in the list. Panorama will bring the target database forward and position the current record to the corresponding location.

The bottom of the Live Clairvoyance window displays the status of the current search. Three numbers are displayed: matches, visible and total. Matches is the number of records that match the current word or phrase. If there is a + symbol after this number then the wizard has not completed the search yet, and more matching records may be found. The + symbol will disappear when the entire database has been searched.

The second number is the number of selected (visible) records in the target database. The Live Clairvoyance wizard only searches within the selected records - any unselected (invisible) records will be skipped. (To make all records visible, click on the target database, choose Select All from the Search menu, then click back to the Live Clairvoyance wizard.) The final number is the total number of records in the target database (including any invisible records).

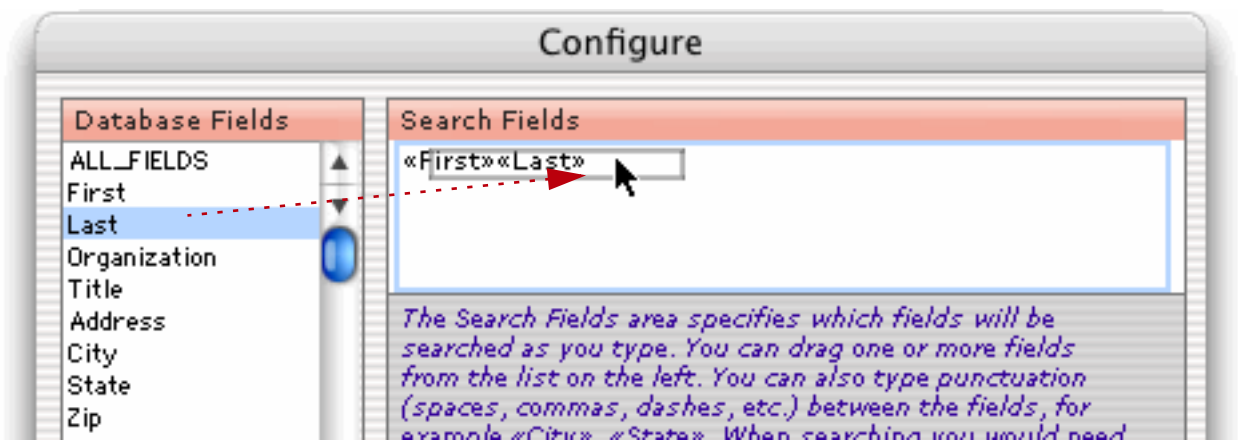
Customizing the Search and Display Fields

The Live Clairvoyance defaults to searching and displaying all fields in the target database. You can customize the wizard to search and / or display only specific fields. Customizing the search fields is especially helpful when using very large databases because searching a large number of fields can adversely affect the search speed. Customizing the display fields can help make the display easier to understand and more useful. To customize the fields, choose **Configuration** from the Setup menu or press the button. This opens the configuration dialog, which looks like this for a target database that has never been customized before.

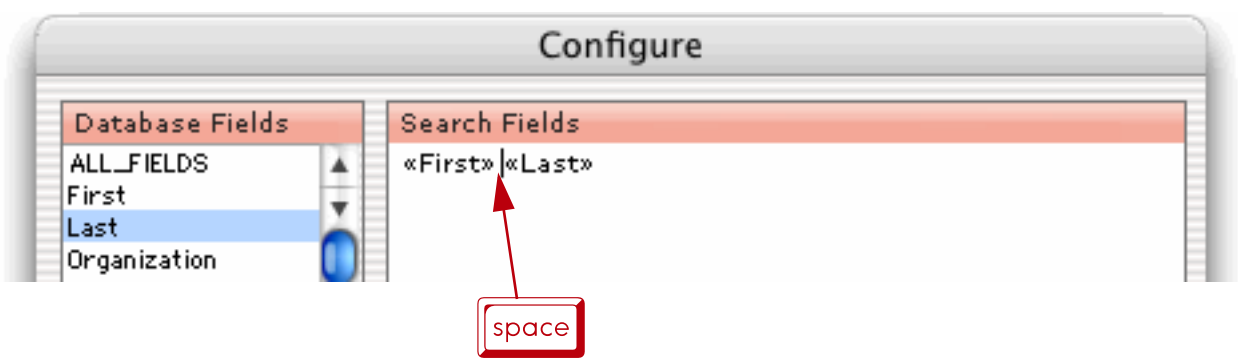


The left hand side of the dialog contains a list of all the fields in the target database. You can drag fields from this list into the two boxes on the right. The first item on the list is always ALL_FIELDS. Use this if you want to search or display all of the fields in the database.

To set up one or more search fields, drag the fields from the list on the left to the text editing area in the upper right. You'll probably want to type some punctuation in between the fields (for example spaces and / or commas). For example, suppose you had separate fields for first and last names. Now suppose you set up the search fields like this, without any punctuation between the fields.

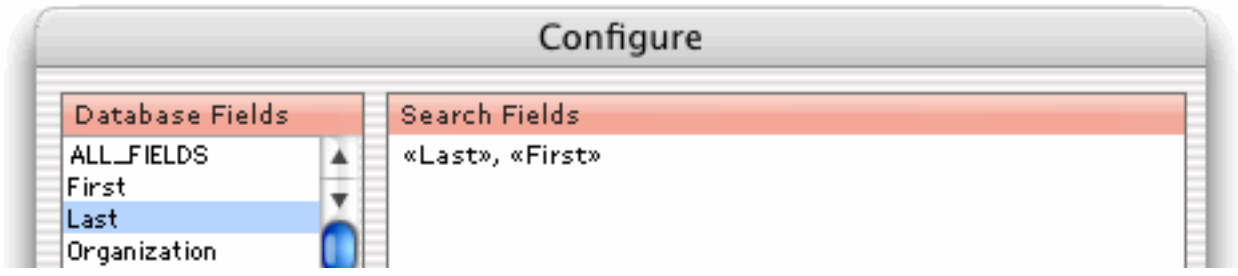


Now suppose you want to search for Bob Smith. With the search configuration set up as shown above, you would have to type **BobSmith** into the search box. A more normal way to set up this configuration would be like this:



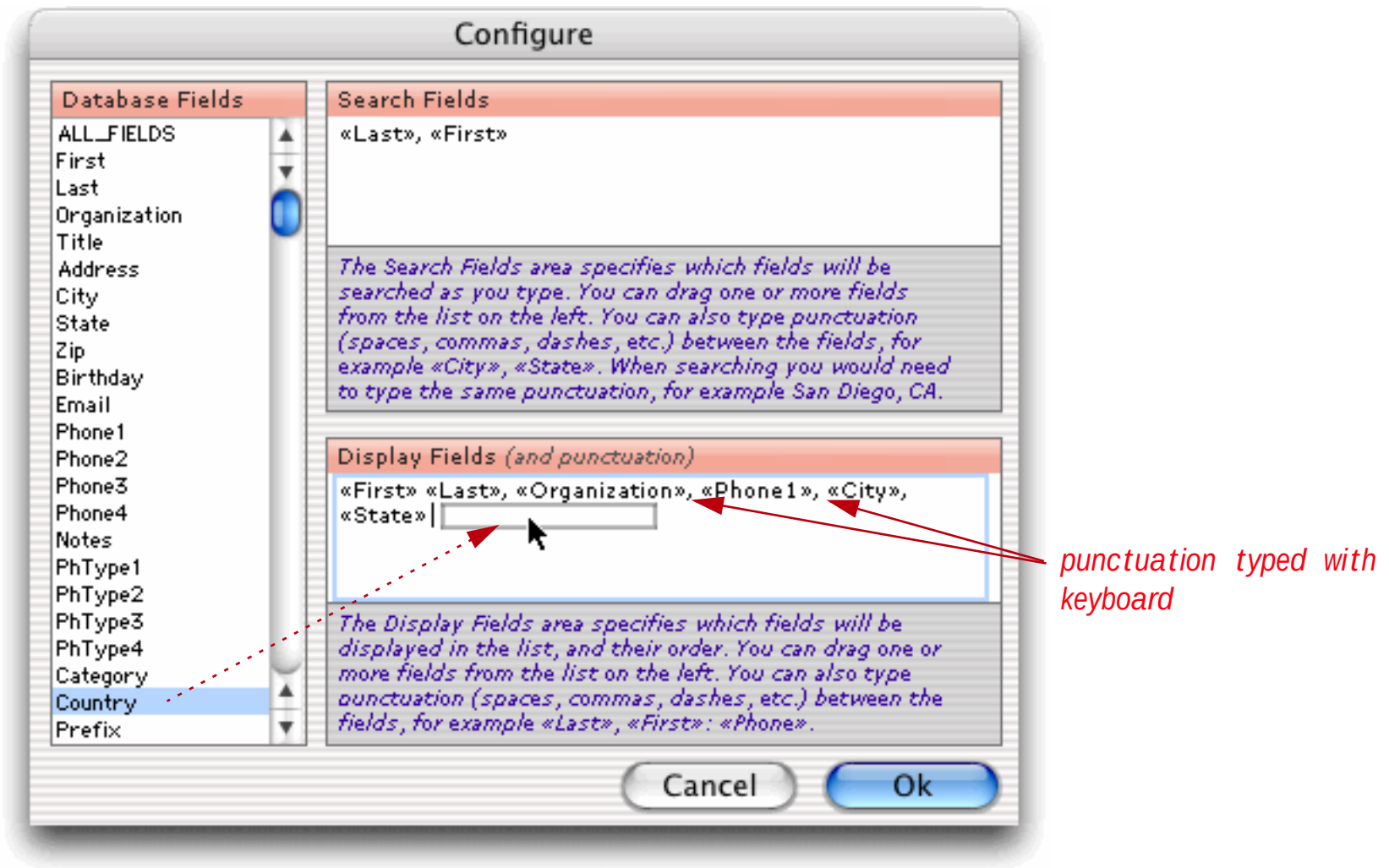
With this configuration you could search for Bob Smith by typing **Bob Smith**.

You are not limited to spaces as punctuation. Another way to set these fields up would be like this:



Now you could search for Bob by typing in **Bob, Smith** or **Smith, Bob**. Searching for **Bob Smith**, however, would not find this person. There is no right way to configure the search fields. It's basically up to you to decide what style is most comfortable for you.

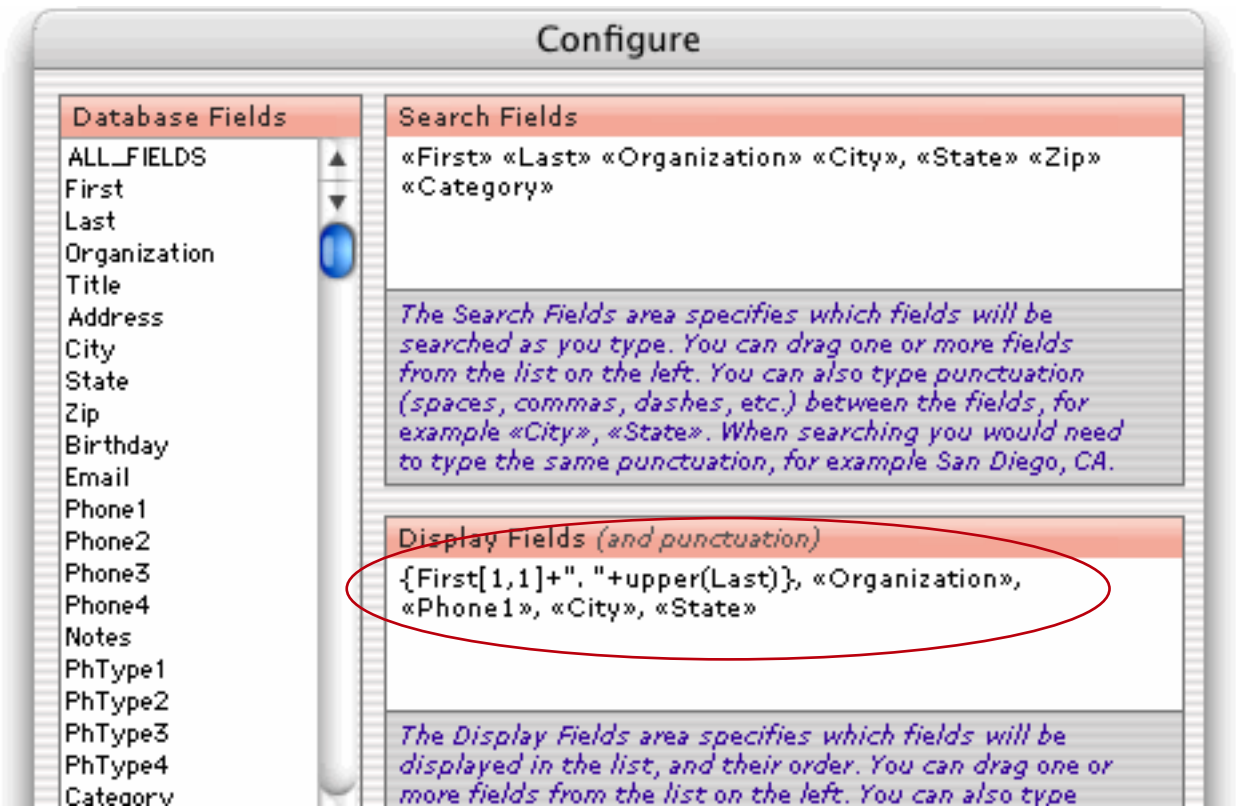
To customize the way records are displayed in the Live Clairvoyance list, drag fields from the list on the left to the text editing box on the bottom right. As with the search fields you can type in any punctuation you like between fields.



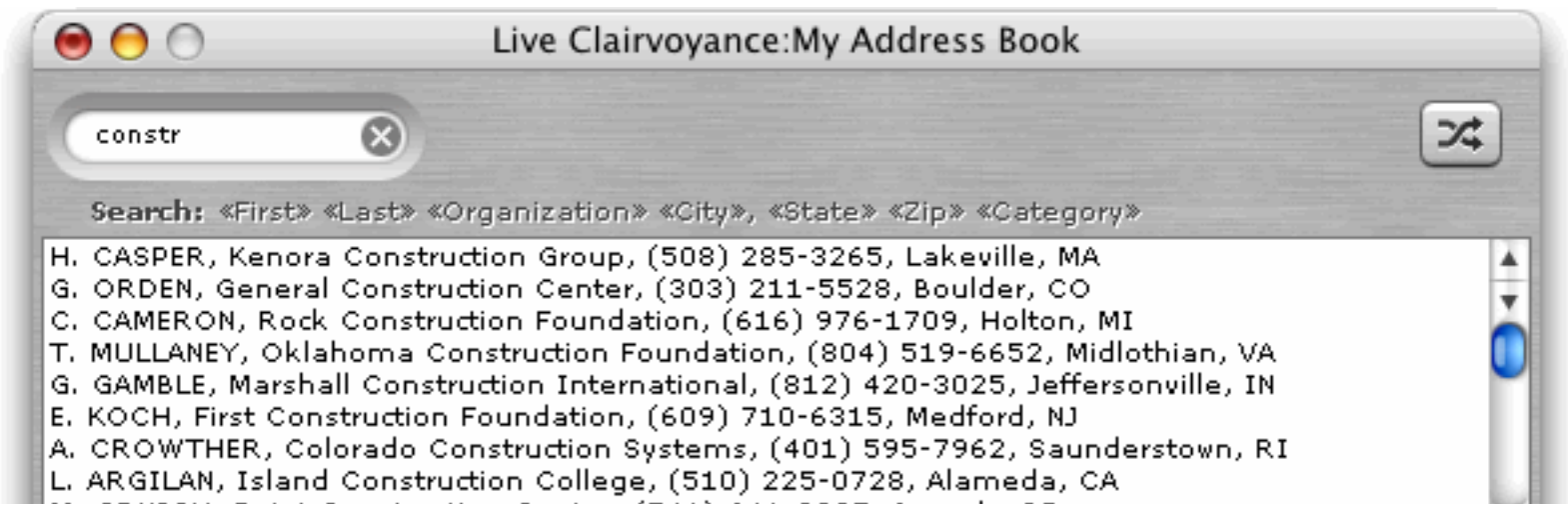
Once you press the **Ok** button the wizard will start using your new configuration. The new configuration will also be stored in a permanent variable in the target database. The next time you save the target database the Live Clairvoyance configuration for that database will be saved along with the data itself. The bottom line is that Panorama keeps a separate configuration for each of your databases.

Advanced Customization with Formulas

If you are a more advanced Panorama user you can insert a formula into either the search or display configuration by using curly braces { and }. Here is an example that will display only the first initial of the first name, with the last name capitalized.



Here is the resulting display.



You can use any Panorama formula operator or function that you want. However, one thing you should NOT do is use the « and » characters within the formula. For example, this means that the field names must be specified as First and Last, not «First» and «Last». If a field name contains spaces or other punctuation you won't be able to use it in a formula. Note: If your formula contains an error, the Live Clairvoyance wizard performs only very limited error checking. You'll get an error message, but the message may not be very helpful in determining the problem. We recommend first debugging the formula with the Formula Wizard, then using Copy and Paste to transfer the formula to the Live Clairvoyance configuration dialog.

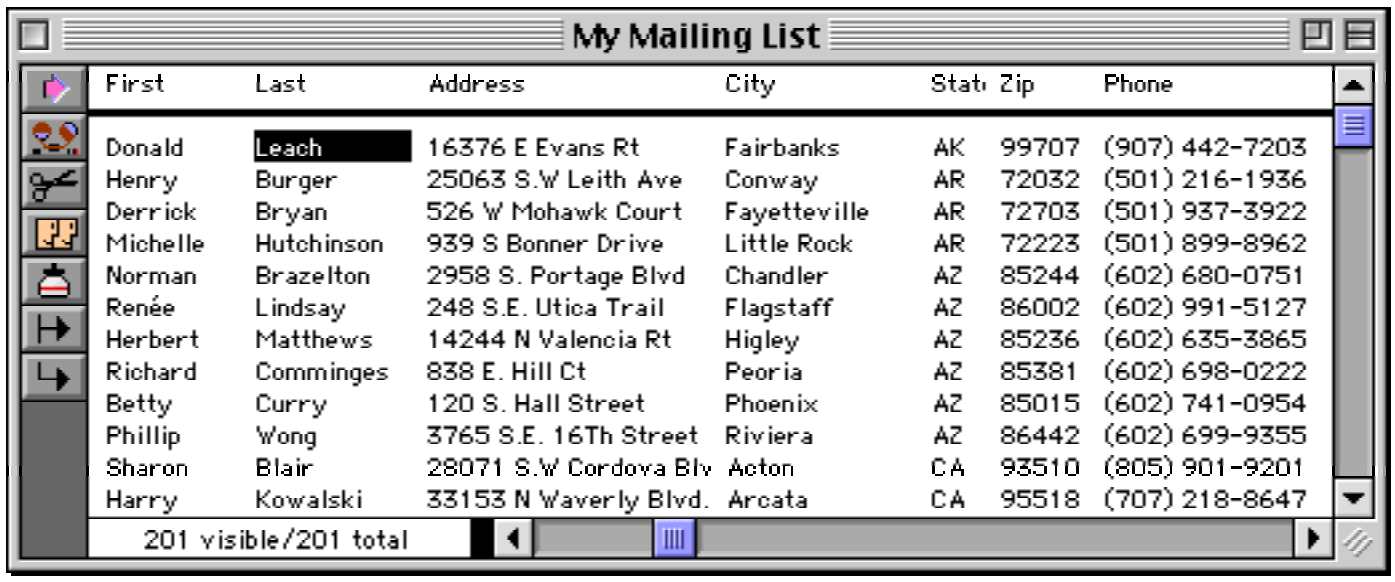
Changing the Target Database

When you open the Live Clairvoyance wizard the currently active database automatically becomes the “target” database (the database that is being searched). There are two ways to change the target database:

1) Click on another database, then choose Live Clairvoyance from the Wizard menu again. Even if the wizard is already open, this will change the target database to the newly selected database.
2) With the Live Clairvoyance window open, choose the target database from the Database menu.

The Search All Fields Wizard

If you want to search all the fields in a database an easy way to do so is to use the **Search All Fields** wizard. Start by opening the database you want to search. To illustrate the operation of this wizard we'll use a mailing list database.



Use the **Wizard** menu to open the **Search All Fields** wizard. The wizard will open in the upper right hand corner of the screen. Notice that the window name displays the name of the database that will be searched.



Enter the word or phrase you want to search for. For our example we'll search for green.



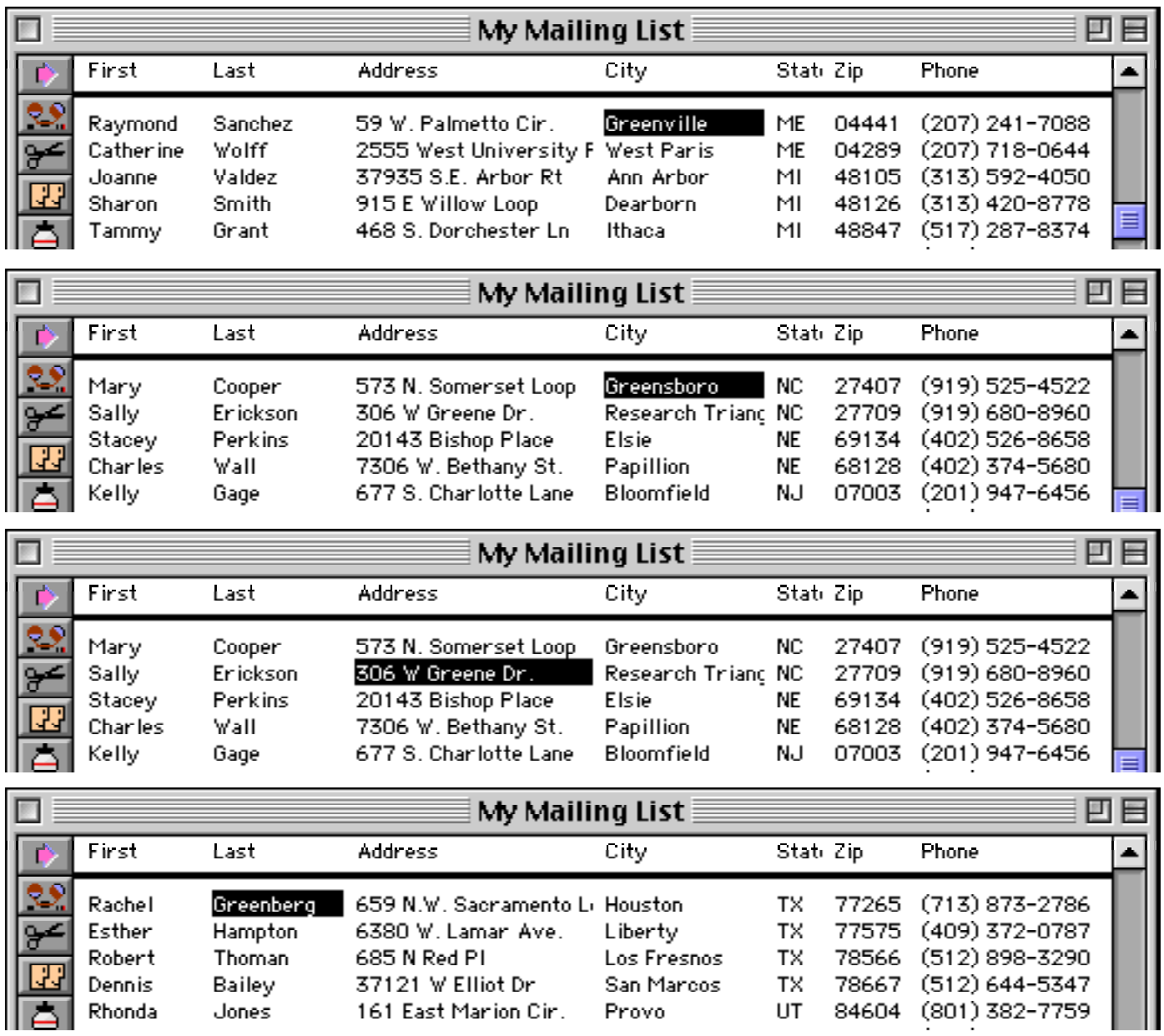
To search for the word or phrase, press the **Enter** key, the **Return** key, the **Find** button or select **Find** from the Search menu. The wizard will locate the first occurrence of the word or phrase within this database.



To find the next occurrence of the word green you can either press the **Next** button or select **Find** from the Search menu. Notice that the next occurrence may be in a different field than the first occurrence.



You can continue to skip to through the database until you have located every occurrence of the word or phrase in any field. When you have reached the last occurrence the wizard will beep.



The **Find** and **Next** commands are designed to be used with the data sheet. If the data sheet is not open the wizard will ask you if would like to open it.

Selecting From All Fields

The **Search All Fields** wizard can select in addition to finding data (see "Finding vs. Selecting" on page 367). Open the wizard, type in the word or phrase, then press the **Select** button or choose **Select** from the Search menu.



The wizard will select all records that contain the word or phrase, no matter what field.



Searching All Fields In Another Database

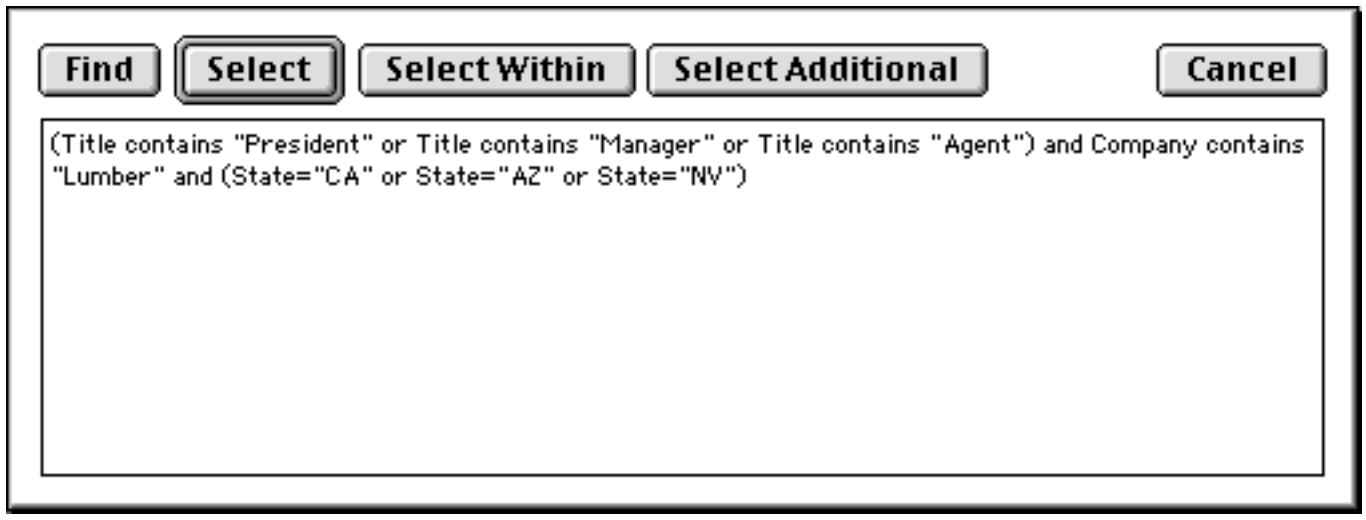
To search all fields in a different database, bring that database to the front and then choose **Search All Fields** from the Wizard menu. This resets the wizard and prepares it to search in the new database. The wizard always shows the name of the database that will be searched in the window title.



Simply clicking on the wizard window is not enough to switch the database being searched, you must choose the wizard from the Wizard menu.

Formula Find/Select

The **Find/Select** dialog is a quick and powerful way to locate data, but it does have limitations—data can only be located based on six fixed values. Another restriction is that and and or cannot be mixed. The **Formula Find/Select** dialog removes these restrictions by using a formula to locate information.



Although it is more work to set up, the **Formula Find/Select** dialog allows you to locate virtually anything that can be described by Panorama's powerful formulas.

The **Formula Find/Select** dialog relies on the ability of a formula to make comparisons and true-false decisions. See "True / False Formulas" on page 1265 for a detailed explanation of **true-false logic**.

If you locate information with the **Find/Select** dialog and then open the **Formula Find/Select** dialog, the dialog will contain a formula equivalent to the criteria specified with the **Find/Select** dialog. This can be a good way to learn how to build your own formulas.

One advantage of the **Formula Find/Select** dialog is that it allows you to mix and and or operations. For example, the formula

```
(State="CA" or State="NY") and Amount>100
```

allows you to locate all transactions over $100 in either California or New York. The parentheses allow you to force the correct order for combining the comparisons.

Another advantage of the **Formula Find/Select** dialog is that it allows you to select data based on calculations or comparisons between two fields. For example, the formula

```
Price/Cost > 2
```

allows you to quickly locate items with high profit margins.

The formula can also manipulate the data before using it. For example, this formula uses text funnels to strip off the first three characters of the name, allowing us to quickly locate all doctors in the database.

```
Name[1,3] = "Dr."
```

See "Taking Strings Apart (Text Funnels)" on page 1214 for more information on text funnels.

The SEQ Function

When combined with the **Formula Find/Select** command, the **seq(** function allows you to select records based on their position in the database. The seq(function returns a unique number for each record in the database. The sequence number starts from the top of the database: 1, 2, 3, etc.

For example, to select the first 10 records in the database use the formula—

```
seq()≤10
```

You can combine the `seq(` function with the **info("Records")** function, which returns the total number of records in the database. This formula selects the last 20% of the database—

```
seq()>=0.8*info("Records")
```

See “SEQ(” on page 5727 for more information on the `seq(` function.

The Select Summaries Command

The **Select Summaries** command (Search menu) selects all of the summary records in the database, and makes all the data records invisible. At the same time, it converts the summary records into data records.

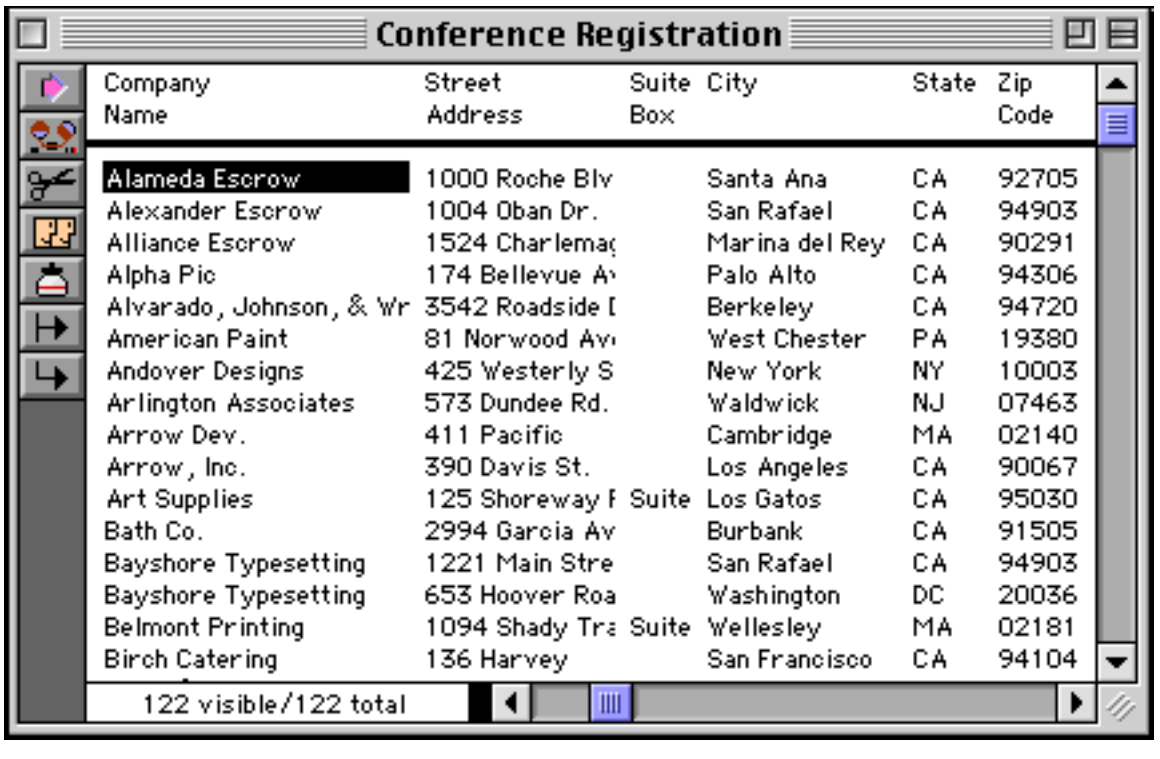
You can use the **Select Summaries** command to select random records within your database. First pick the records you want to select by turning them into summary records. (To turn a record into a summary record, click on the left edge of the record in the data sheet. See “Manually Creating and Removing Summary Records” on page 402.) To actually select the summaries, choose **Select Summaries** from the Search menu. Remember, this will convert all your summary records into data records.

Warning: If you use the **Select Summaries** command to select random records, you can’t use summary records for their regular job—calculating subtotals. If you need to use summary records for calculating subtotals, don’t use the **Select Summaries** command! Instead, create an extra field for specifying the records you want to select. See “3-Step Summarizing” on page 393 for more information on the regular uses of summary records.

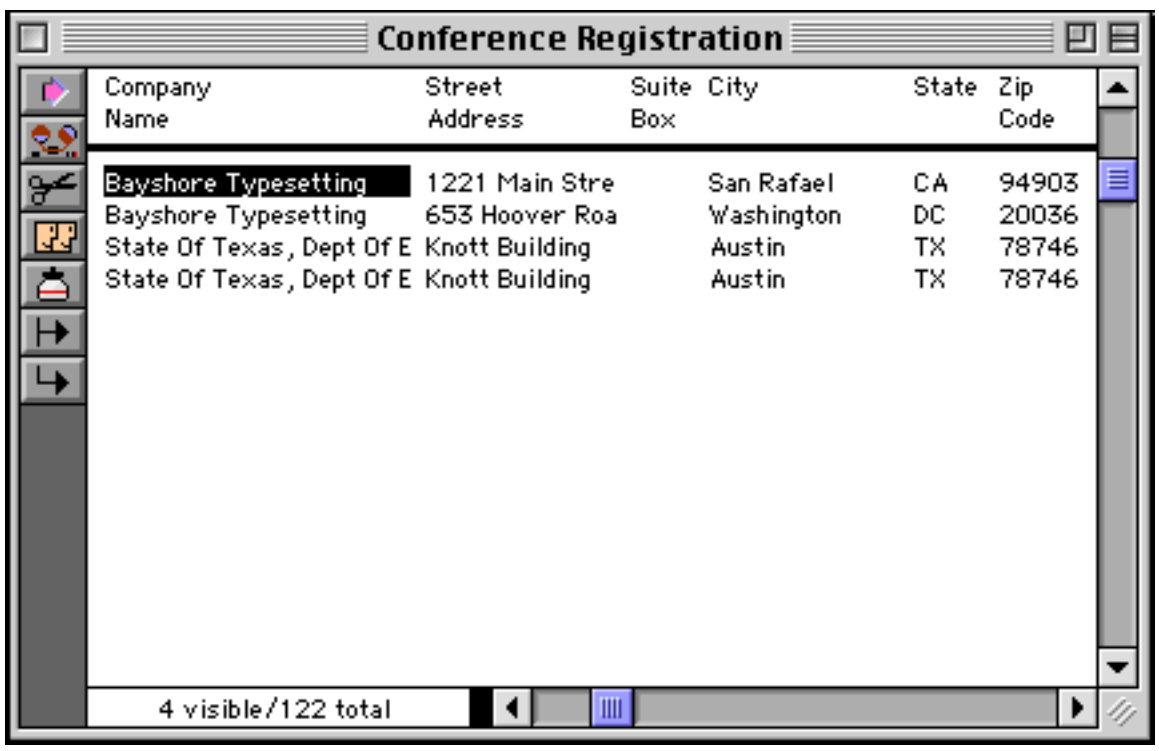
Select Duplicates

The **Select Duplicates** command (in the Search Menu) provides a fast and easy way to locate duplicate information in a database. The **Select Duplicates** command does not remove the duplicates, it simply selects them so you can examine them. You can then decide what to do about each duplicate on a case-by-case basis. You may select duplicates based on a single field (for example, all duplicate company names), on multiple fields (for example, all records with duplicate address, city, and state), or on a formula that may combine fields or use partial fields (for example, all records containing duplicate area codes).

To select duplicates based on a single field, start by using the **Sort Up** command to sort the database by that field. If the database is not sorted, the **Select Duplicates** command will warn you. For example, here is a conference registration database that may contain duplicate company information. It has been sorted into alphabetically order by company.



After the database is sorted, choose the **Select Duplicates** command from the Search Menu (Make sure you have clicked on the field you want to check for duplicates before selecting the command.) This command opens a dialog box. Leave the dialog box empty and press the **OK** button. Panorama will select the records that contain duplicate information (if any), making everything else invisible.



As you can see, there are two possible duplicates in this database.

The **Select Duplicates** command allows you to examine duplicate records. See "Using UnPropagate to Eliminate Duplicates" on page 470 if you simply want to delete all but the first duplicate entry automatically.

Select Duplicates Using a Formula

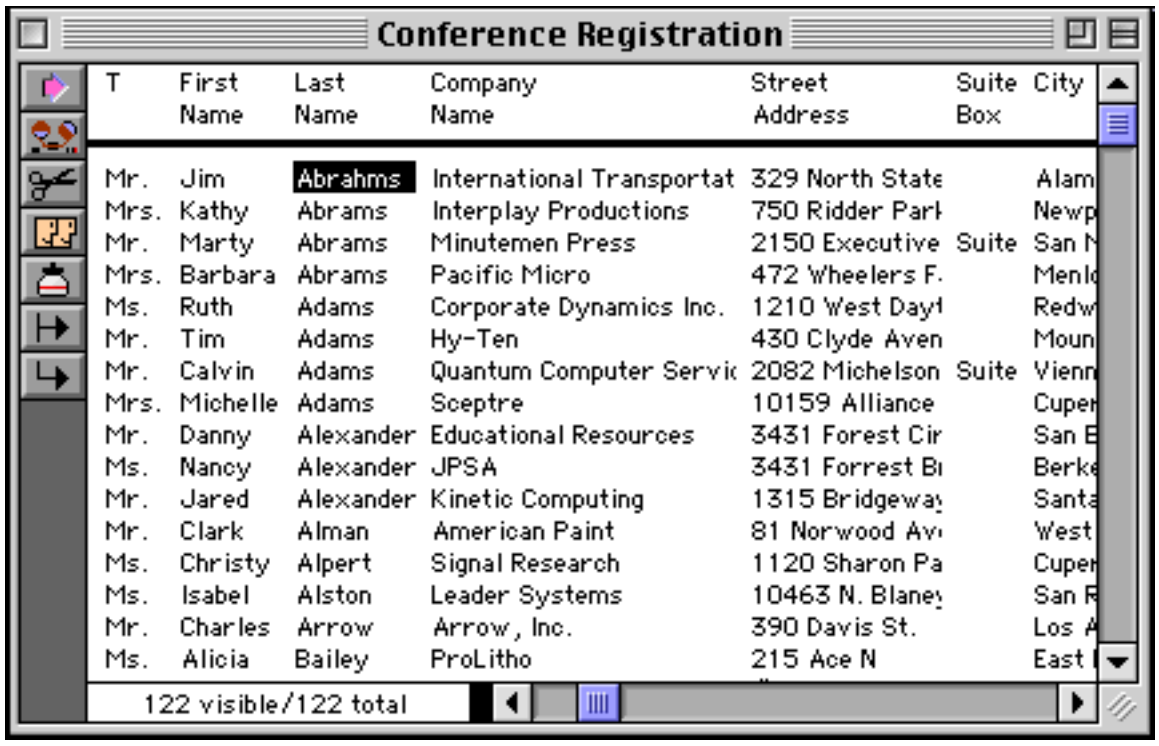
To select duplicates based on multiple and/or partial fields, you'll need to use a formula. The formula tells Panorama exactly what data should be checked for duplicates. For example, suppose a database contains separate fields for first and last names. This formula could be used to check for duplicate names:

```
FirstName+LastName
```

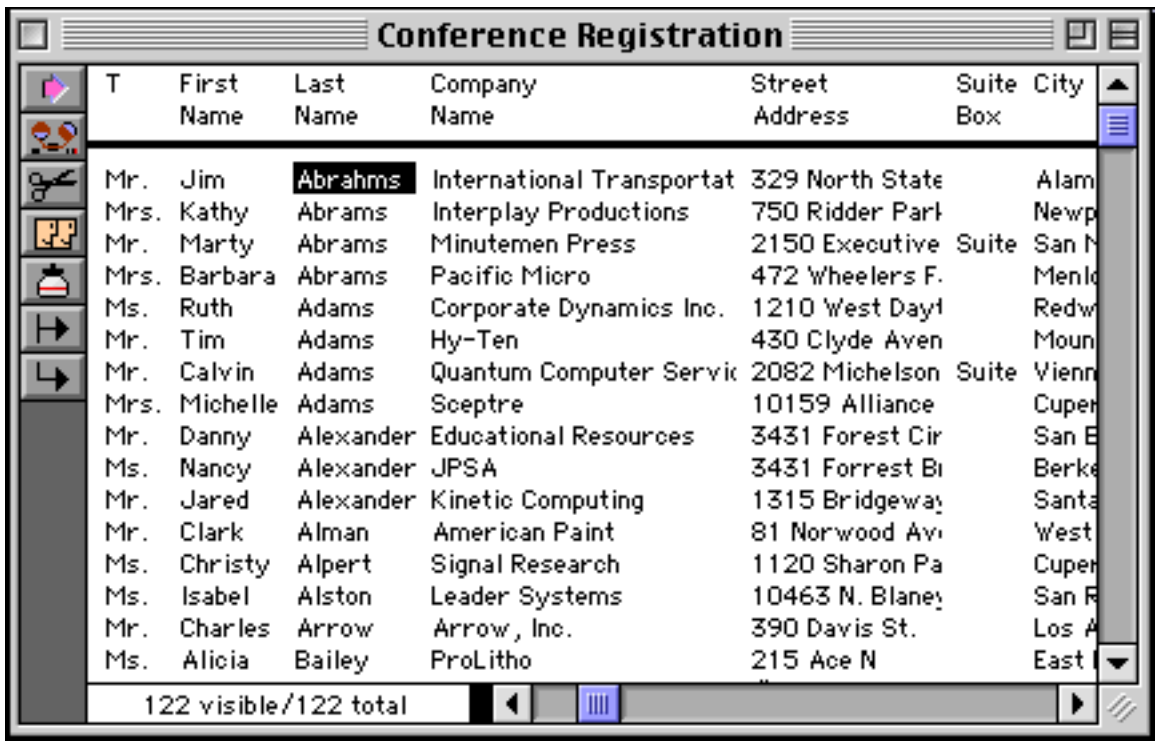
If you wanted to check for duplicates using the first initial and the last name, you would use this formula:

```
FirstName[1,1]+LastName
```

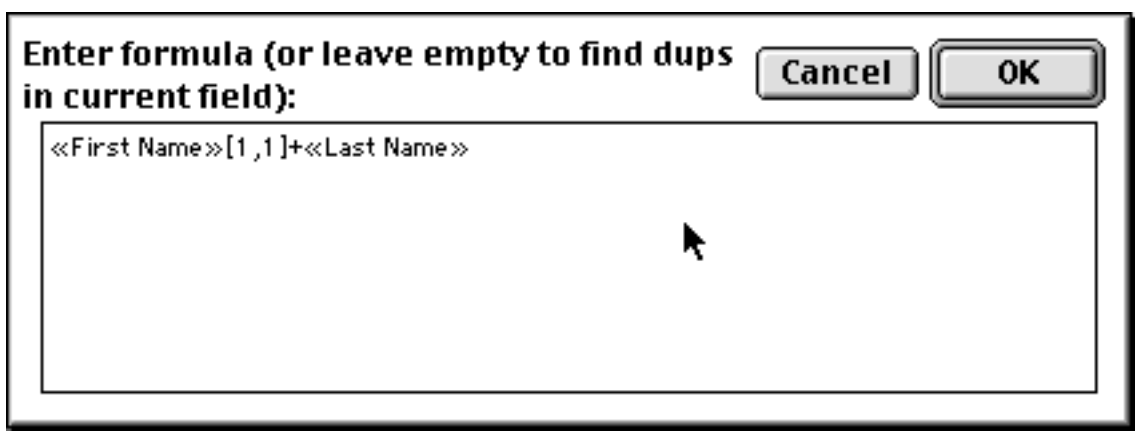
This formula would tell Panorama to treat John Doe, Joan Doe, and Jeff Doe as duplicates because they all have the same first initial and last name. Let's search for duplicates in our conference registration file. Start by sorting up by Last Name.



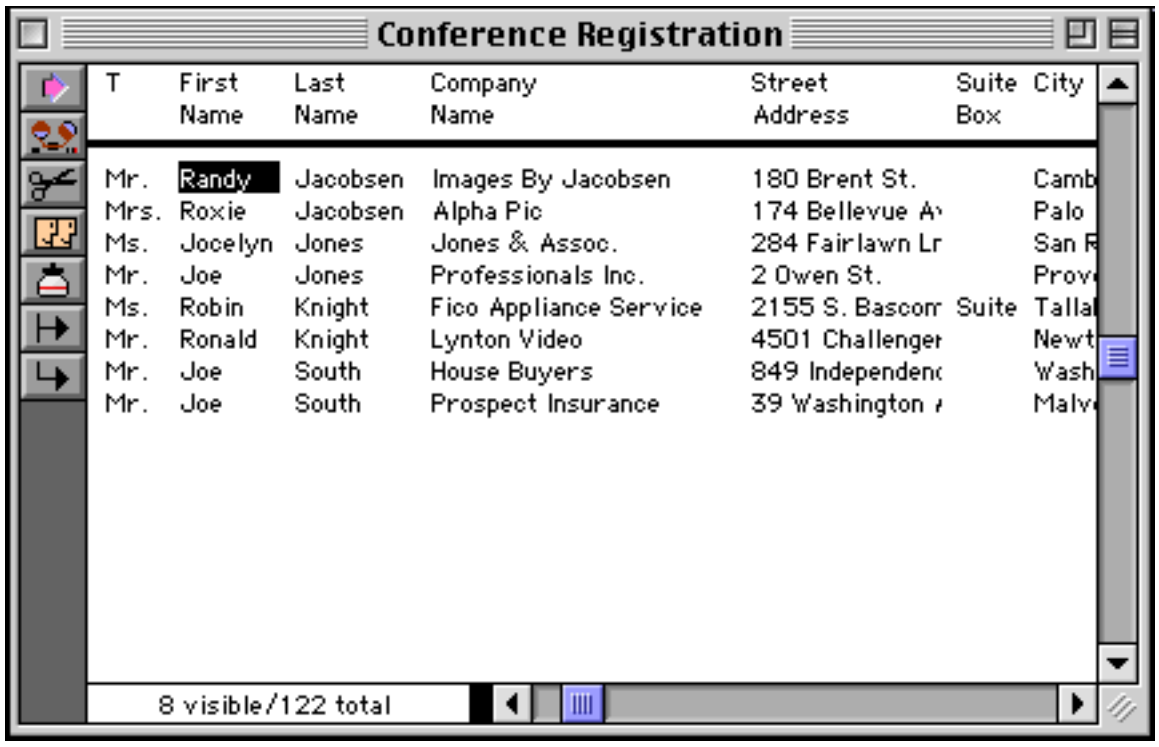
Now sort up within by first name.



Now choose the **Select Duplicates** command, and type in the formula:



Here is the final result. There are two R Jacobsen's, two J Jones, two R Knights, and two J South's.



Note: The formula must produce a text result. If you want to include numeric fields, they must be converted to text with the `str(` or `pattern(` functions (see "Converting Between Numbers and Strings" on page 1227). Date fields must be converted to text with the `datepattern(` function (see "Converting Between Dates and Text" on page 1249).

Warning: When you are using a formula to check for duplicates, only the first 300 characters from the formula result are actually used for duplicate checking. If the first 300 characters are the same, Panorama will treat these records as duplicates. Normally, this isn't a problem when you are check for duplicate names, addresses, etc., which are much shorter than 300 characters.

Chapter 10: Summaries and Outlines



It is very difficult to look at a database containing thousands of records and make much sense of it. There's simply too much information to cope with. To make the information more understandable, it needs to be summarized. Panorama can rapidly summarize a database according to the criteria you specify.

Panorama has two methods for summarizing a database. The first method, which is discussed in this chapter, is to create an outline using summary records. See "Crosstabs" on page 433 to learn about the second method, **crosstabs**.

3-Step Summarizing

Summarizing a database is a three step process—**group**, **calculate**, and **outline**. Before diving into all the details and options, let's take a look at a basic overview of these three steps.

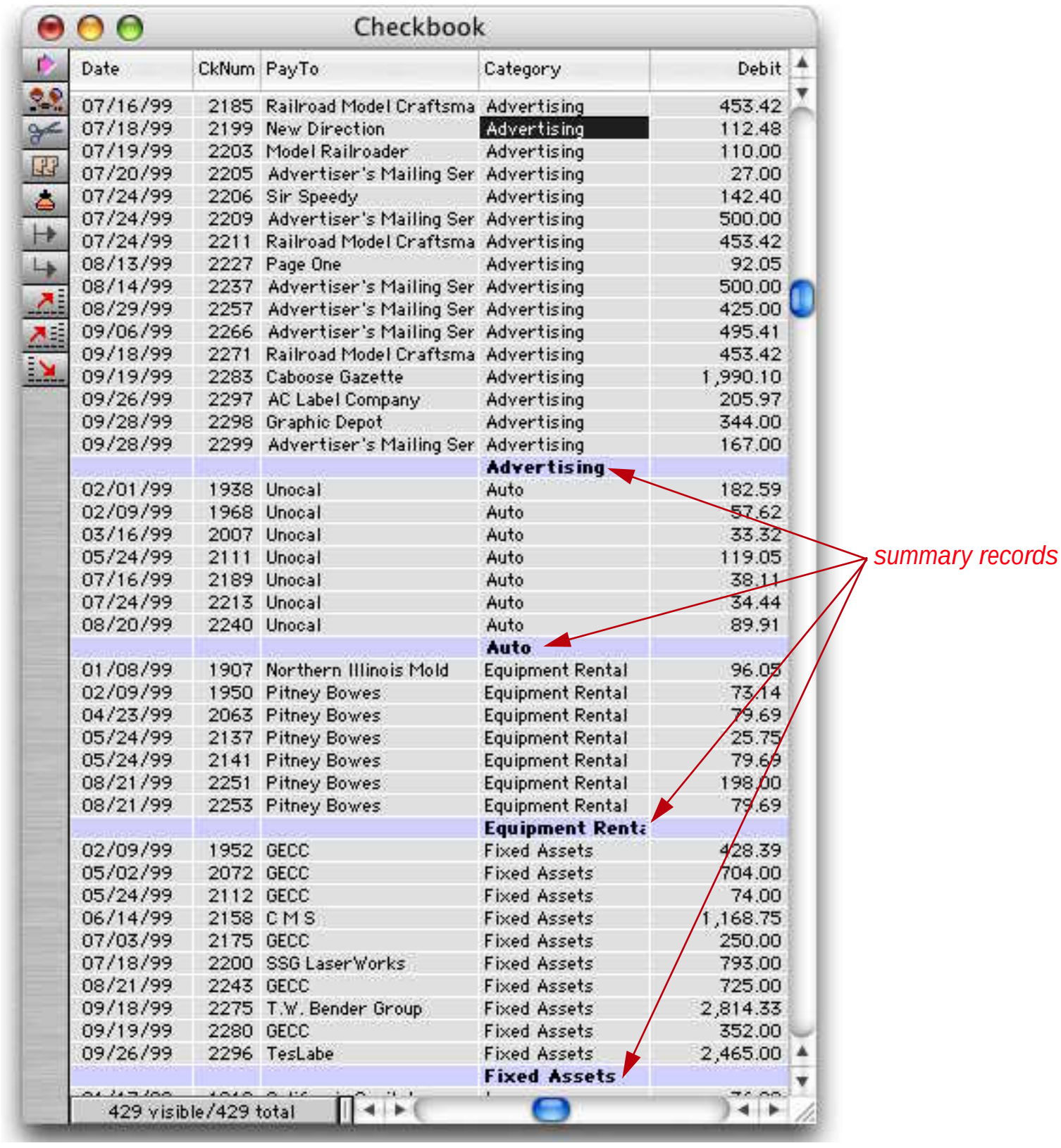
The first step is always the raw data. For this example we'll use a checkbook database, which we will summarize by category. Notice that the categories start out in more or less random order.

Checkbook

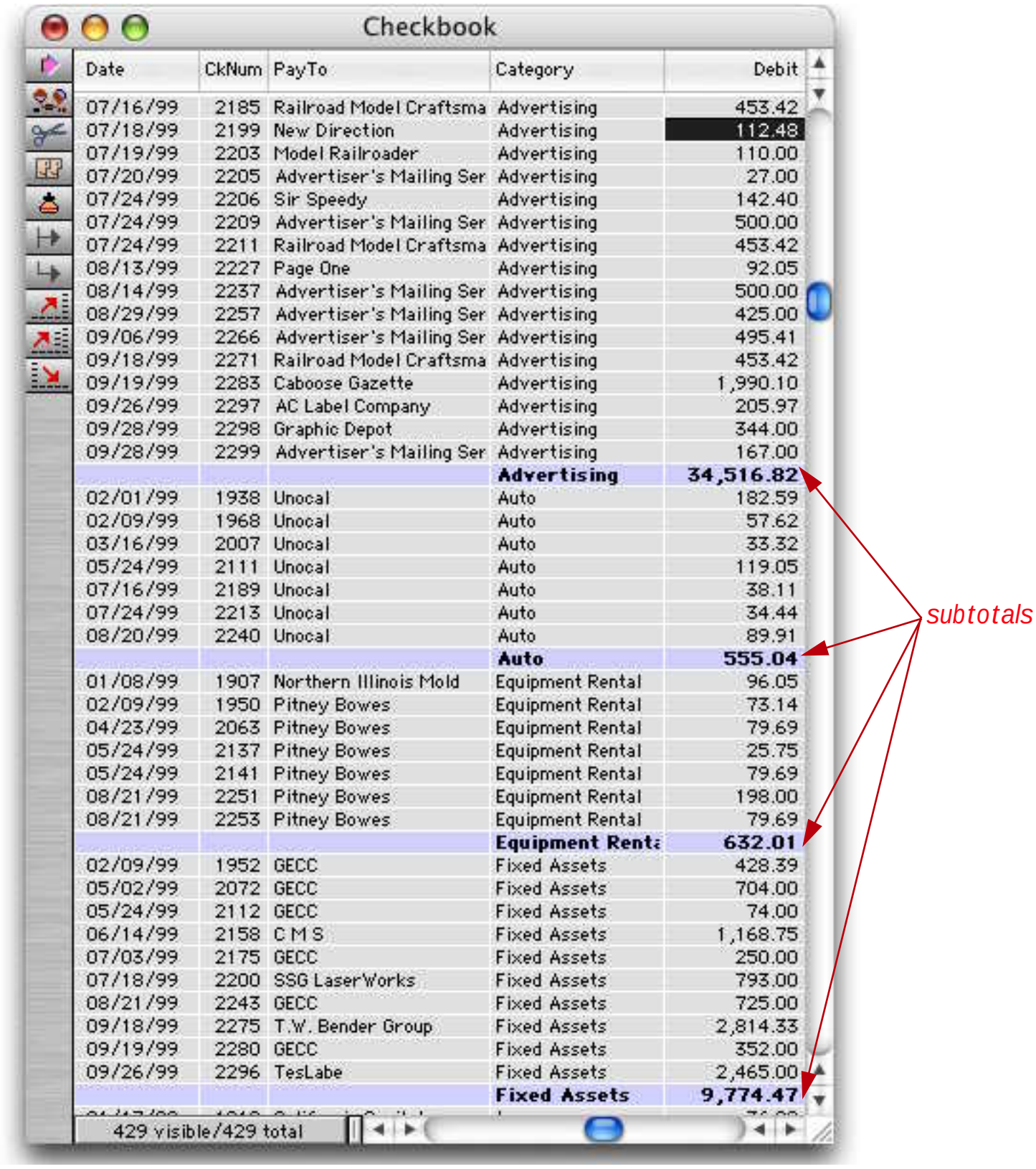
Date	CkNum	PayTo	Category	Debit
01/08/99	1907	Northern Illinois Mold	Equipment Rental	96.05
01/08/99	1908	U S Postmaster	Postage	75.00
01/08/99	1909	Advertiser's Mailing Ser	Advertising	390.80
01/16/99	1910	Coudert Brothers, Attor	Legal Fees	223.52
01/16/99	1911	Paramount Stationers	Office Supplies	105.84
01/17/99	1912	California Capitol	Insurance	36.00
01/17/99	1913	California Capitol	Insurance	28.00
01/17/99	1914	U S Postmaster	Postage	75.00
01/17/99	1915	Sacramento Bee	Advertising	795.00
01/22/99	1916	Walthers	Purchases	12,463.00
01/22/99	1917	Blue Cross Of Calif	Insurance	279.03
01/22/99	1918	Sherman Douglas Ins	Insurance	418.60
01/22/99	1919	Cannon Astro	Office Supplies	145.72
01/25/99	1920	Walthers	Purchases	1,885.40
01/25/99	1921	Nebs	Office Supplies	77.27
01/25/99	1922	Ramona Drinking Water	Office Supplies	98.10
01/25/99	1923	Pacific Partners	Rent	4,070.83
01/29/99	1924	Athearn	Purchases	1,906.32
01/29/99	1925	Advertiser's Mailing Ser	Advertising	860.22
01/29/99	1926	PacTel Cellular	Telephone	141.09
01/30/99	1927	State Board Of Equalizat	Taxes	549.00
01/30/99	1928	Walthers	Purchases	828.70
01/30/99	1929	Federal Express	Shipping	178.75
01/31/99	1930	U P S	Shipping	52.97
01/31/99	1931	Sacramento Bee	Advertising	795.00
02/07/99	1932	Cable & Wireless	Telephone	99.25
02/01/99	1933	Ed Burnett Consultants	Advertising	327.64
02/01/99	1934	US Sprint	Telephone	17.14
02/01/99	1935	Copierland	Office Supplies	137.04
02/01/99	1936	Alpha Graphics	Advertising	195.74
02/01/99	1937	Walthers	Purchases	536.00
02/01/99	1938	Unocal	Auto	182.59
02/01/99	1939	FastFacts	Office Supplies	108.25
02/01/99	1940	Yuba Register	Advertising	316.66
02/07/99	1941	City Of Caboose	Utilities	84.78
02/07/99	1942	U S Postmaster	Postage	75.00
02/07/99	1943	U S Postmaster	Postage	35.00
02/09/99	1944	Page One	Advertising	23.68
02/09/99	1945	Blue Cross Of Calif	Insurance	256.47
02/09/99	1946	U P S	Shipping	122.60
02/09/99	1947	Pacific Partners	Rent	3,862.63
02/09/99	1948	California Secretary Of :	Legal Fees	5.00
02/09/99	1949	City Of Caboose	Legal Fees	90.00
02/09/99	1950	Pitney Bowes	Equipment Rental	73.14

411 visible/411 total

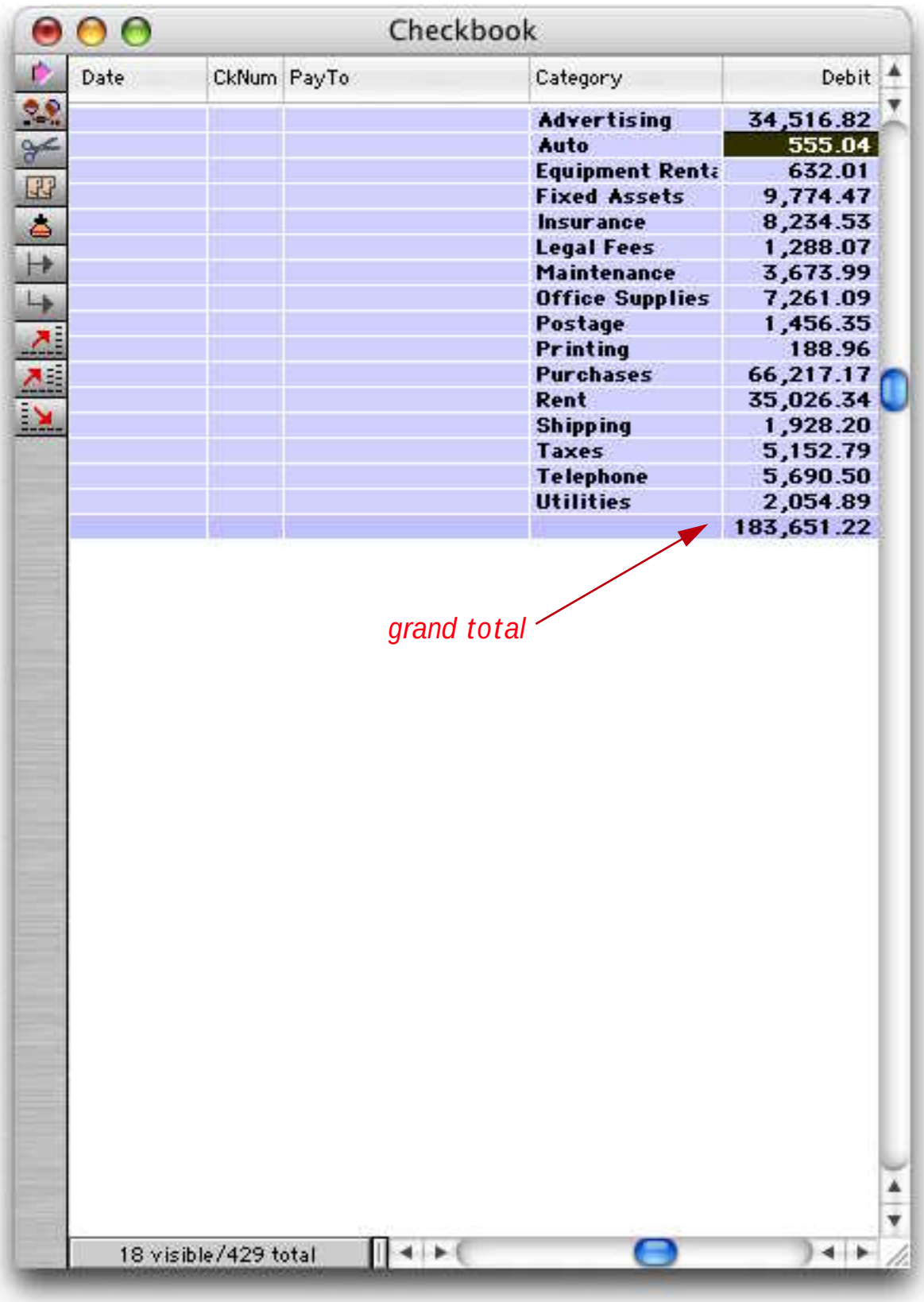
Step 1 is **group**, so we'll start with the **Group Up** command (Sort menu) to divide the database into groups by spending category (advertising, purchases, rent, etc.). As you can see, the database is now sorted in order by category. In addition, Panorama has added several new records to the database. These are called **summary records** and can be identified by their blue background color and by the fact that they are displayed in bold.



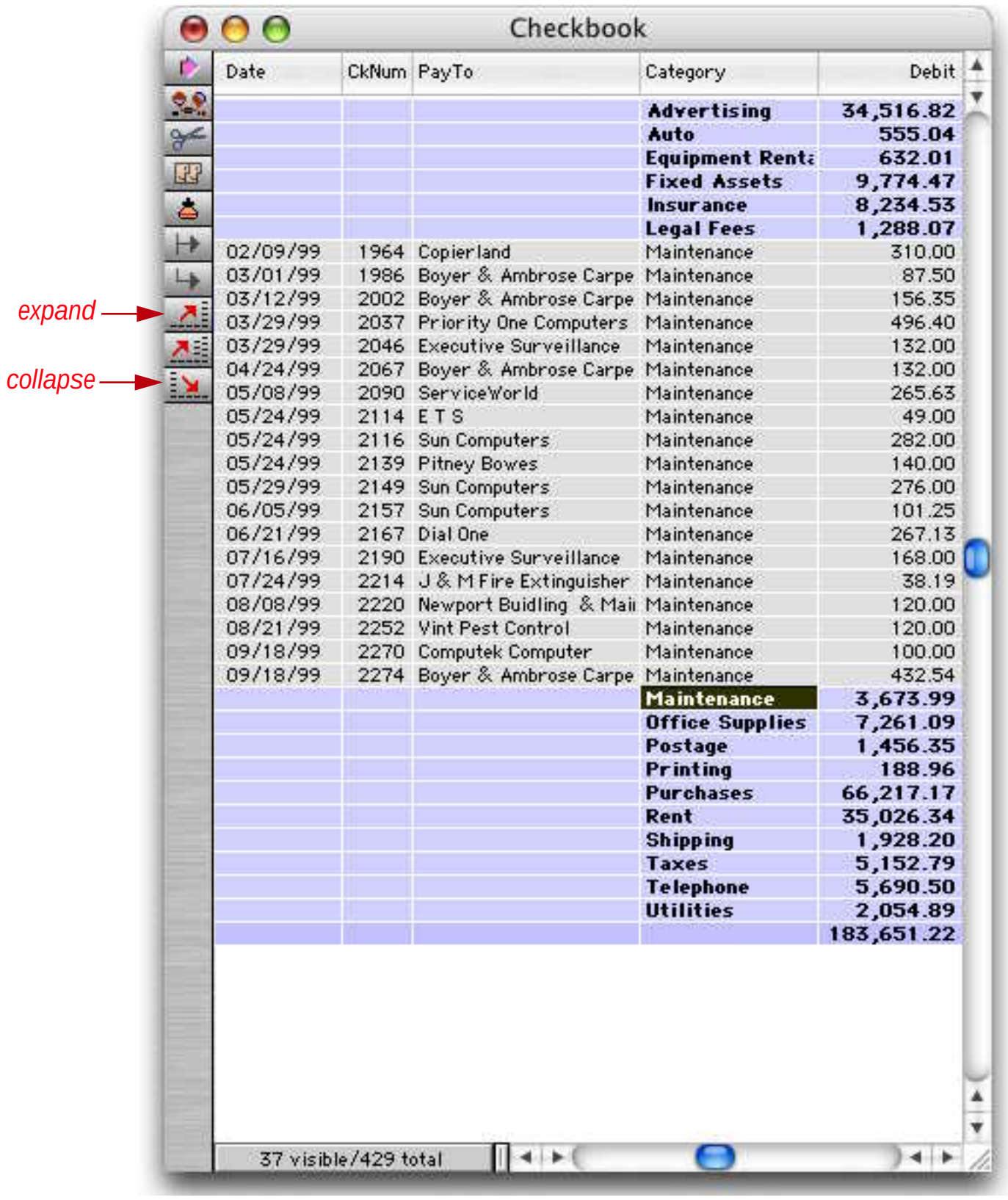
Step 2 is **calculate**, which we'll do with the **Total** command (Math menu). This command scans the database and calculates the subtotal for each category, as well as an overall total at the bottom of the database (not visible in this illustration).



Step 3 is **outline**, which allows us to hide unnecessary detail so that we can focus on just the numbers that are important to us. For our example we'll use the **Outline Level** dialog (Sort menu) to collapse the database so that only the summary information is visible. In addition to the subtotals you can also now see the grand total at the bottom. Notice that the background blue color for the grand total is slightly darker than the other summary records.



Using the **Expand** and **Collapse** tools (in the tool palette) the summaries can be expanded or collapsed to show more or less detail. Here we've used the **Expand** tool to examine the maintenance detail.



When you are finished with the summary, use the **Remove Summaries** command (Sort menu) to remove the subtotals and totals, leaving only the original data. You can then continue with normal operations on your database (data entry, sorting, searching, etc.).

Tip: The 3-Step summary process is an ideal candidate for automation with procedures. See "Procedures" on page 1339 for information on recording and using a procedure. Panorama also comes with a special wizard to help automate the summary process, see "The Summaries & Outlines Wizard" on page 423.

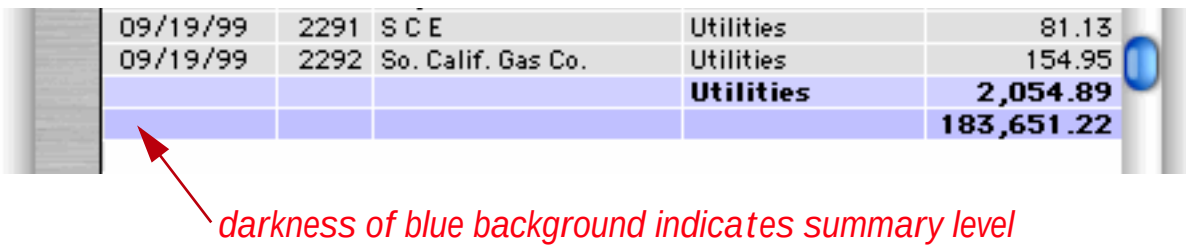
Now that you've seen the basic three step summary process, let's look at each step in detail.

STEP 1 - GROUP

The first step in summarizing a database is to divide the database into groups or categories. For example, a checkbook database could be arranged into groups by month (Jan, Feb, Mar, etc.), by budget category (rent, food, transportation, etc.), or by payee (Evian Apartments, Lakeman's Market, Unocal, etc.).

To divide a database into groups, first click anywhere in the field you want to group. Then use either the **Group Up** or **Group Down** command (Sort menu) to divide the database. The **Group Up** command arranges the data into ascending order—A's first, Z's last. The **Group Down** command arranges the data in descending order, Z to A.

The **Group Up** and **Group Down** commands add a special summary record at the end of each group. Summary records are temporary records used for calculating and displaying summary information. On the data sheet you can easily identify summary records by their blue background color, and by the fact that they are usually displayed in bold.



Subgroups

Groups can be sub-divided into even smaller subgroups. For example if you had arranged a mailing list into groups by state, you could further divide each state into subgroups by city. You can continue subdividing the groups up to six times (up to seven levels of groups within groups).

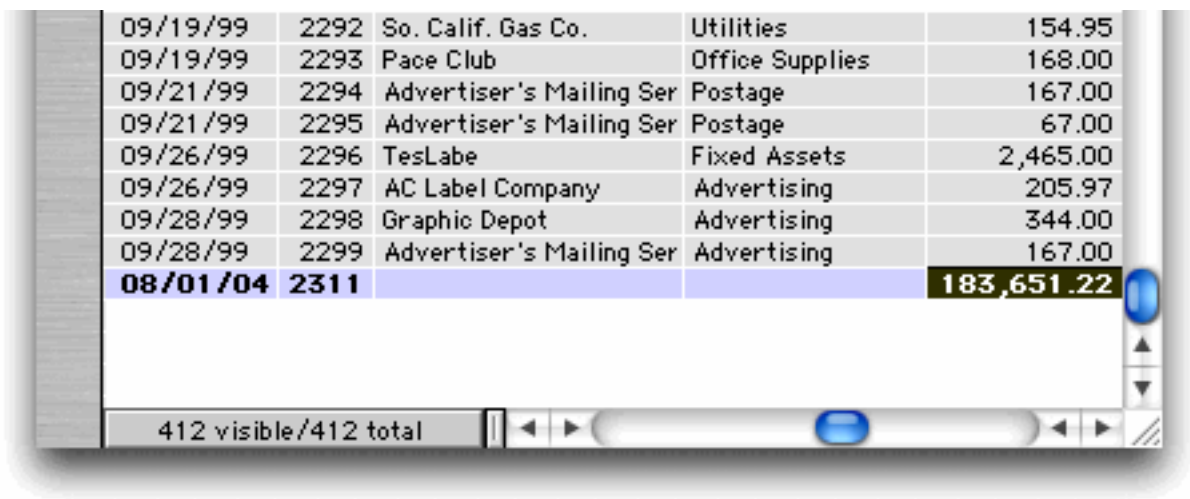
To subdivide groups into smaller groups, first click on the field you want to sub-group, then use the **Group Up** or **Group Down** command again.

When you are looking at the data sheet, you can identify the subgroup level by the size of the plus sign to the left of the summary record. The lowest level subgroups have the smallest plus signs; the higher level groups have larger plus signs. The grand-total record has the largest plus sign (see illustration above).

Grand Total

When you arrange a database into groups, Panorama automatically creates an additional summary record at the bottom of the database. This summary record is for the largest group of all, the entire database. When totals or other summary calculations are performed (see "STEP 2 - CALCULATE" on page 403), this summary record holds the overall grand total (or average, count, etc.) for all of the selected records in the entire database.

If all you need is a grand total (or average, count, etc.), you can skip Step 1 and go directly to Step 2, Calculate. When a summary calculation is performed on a database that doesn't have any summary records, Panorama automatically appends a single summary record to the end of the database. It then calculates the grand total (or average, count, whatever) in this summary record. The illustration below shows the result after the **Total** command has been used without first grouping the database.



09/19/99	2292	So. Calif. Gas Co.	Utilities	154.95
09/19/99	2293	Pace Club	Office Supplies	168.00
09/21/99	2294	Advertiser's Mailing Ser	Postage	167.00
09/21/99	2295	Advertiser's Mailing Ser	Postage	67.00
09/26/99	2296	TesLabe	Fixed Assets	2,465.00
09/26/99	2297	AC Label Company	Advertising	205.97
09/28/99	2298	Graphic Depot	Advertising	344.00
09/28/99	2299	Advertiser's Mailing Ser	Advertising	167.00
08/01/04	**2311**			**183,651.22**

412 visible/412 total

The Group Command

Unlike **Group Up** and **Group Down**, the **Group** command does not sort the database. The **Group** command is useful when you want to add summary records to a database that is already arranged in the proper order.

Grouping by Week, Month, Quarter, or Year

When a group command is used on a field containing dates, Panorama will ask you how long each group should be—a week, month, quarter, or year.



Select the period and press **Group** to arrange the data into groups. Note: The dates must be stored using the date data type, not text. See "Dates" on page 294, for more information about the date data type.

You can group dates more than once—for instance first by year, then by month. This produces subgroups (in this case by month) within the larger groups (by year).

Grouping a database by month, quarter, or year does not change the way the dates are displayed. You may want to change the output pattern for the date field so that only the month, quarter, or year is displayed, instead of the entire date. The output pattern is specified as part of the design sheet. To display only the month and year, use an output pattern like Mon-yy or mm-yy. To display only the quarter and year, the pattern could be `qqqyy` or `Qtr Qtr yy`. To display the year only, the pattern could simply be `yy` or `yyyy`. See "Date Output Patterns" on page 295 for more information about date output patterns and how to set them up.

If you print a database that is grouped by month, quarter, or year, you can use different summary tiles to format the dates properly. See "Printing Data Grouped by Month, Quarter or Year" on page 1136.

Group by Color

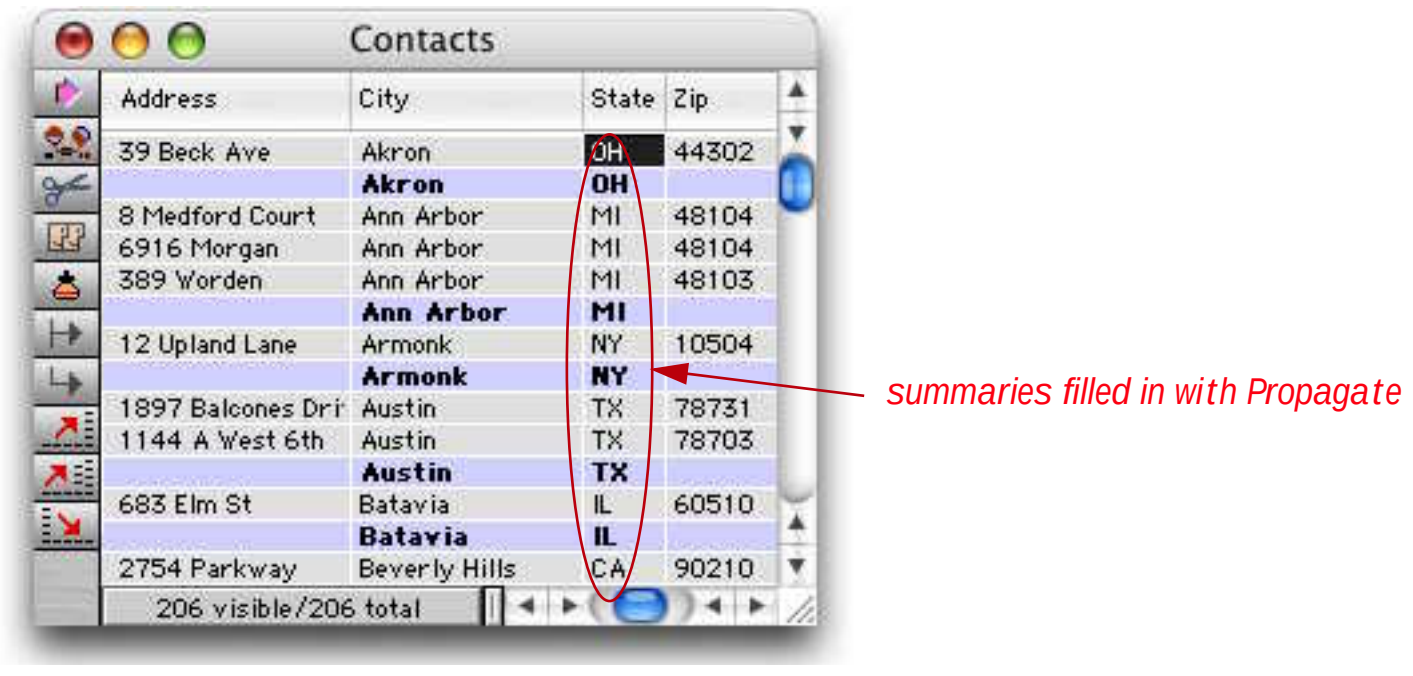
The **Group by Color** command groups the database by the color of the data cells in the field. The database can be divided into as many as seven groups—black, red, green, blue, cyan, magenta, and yellow. See "Data Style and Color" on page 474 for details on how to assign colors to data cells.

Propagating Data into Summary Records

The Group commands create summary records but leaves most fields blank. The **Propagate** command can be used to copy additional information into the newly created summary records. In the illustration below the database has been **Group**ed by City.



Next we'll fill in the State field for each summary record by choosing **Propagate** from the Math menu.



See "Propagate" on page 465 for more information on the **Propagate** command.

Manually Creating and Removing Summary Records

Summary records are normally created automatically with one of the three **Group** commands, and removed with the **Remove Summaries** command (described later in this chapter). You can also manually turn any normal data record into a summary record and vice versa, although this should rarely be necessary. In fact, we actively discourage the use of this feature.

To turn a normal data record into a summary record, click anywhere in the record in the data sheet, then choose the **Toggle Summary Level** command in the Sort menu. Each time you choose this command will toggle between normal and summary.



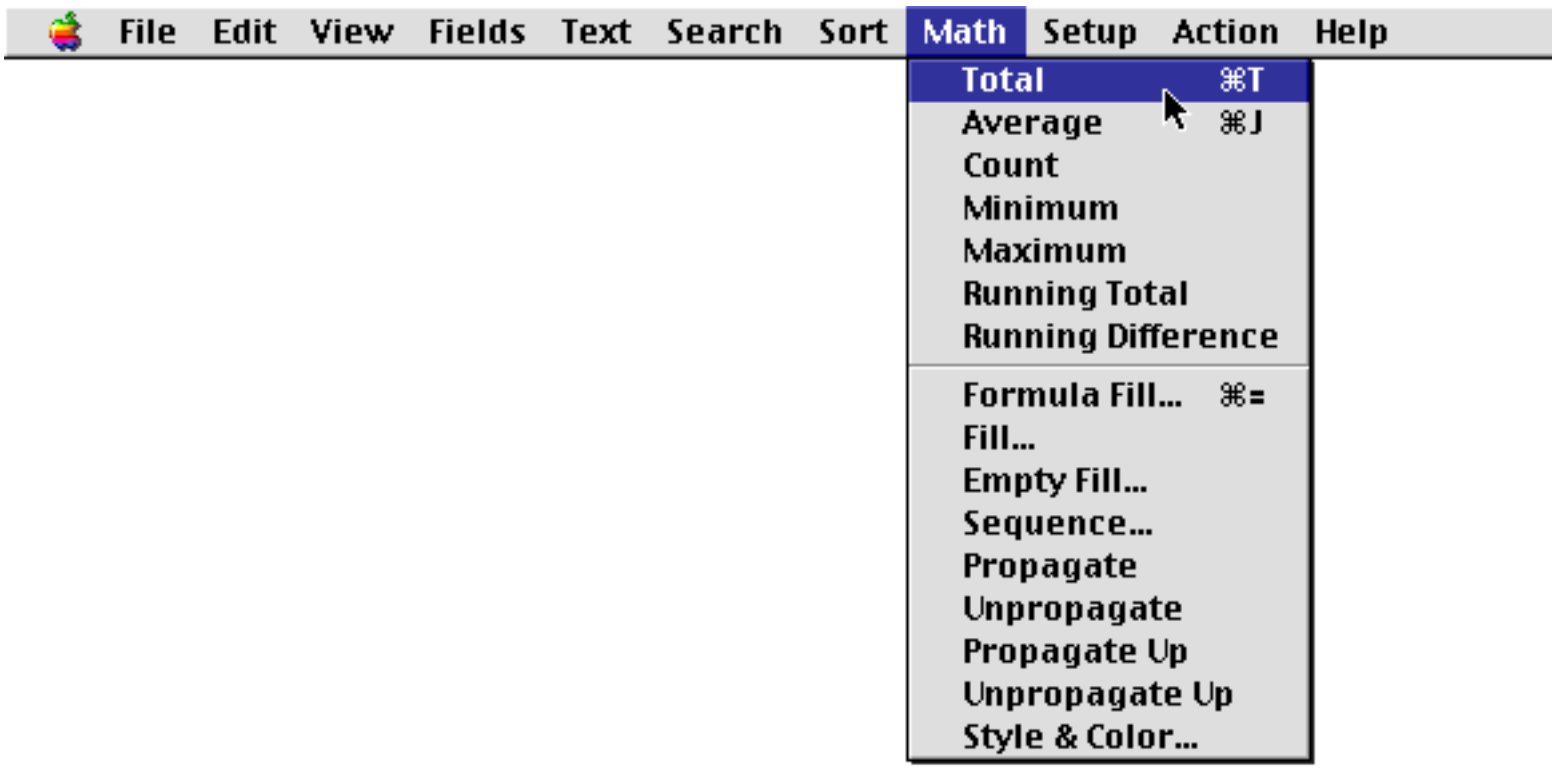
The **Toggle Summary Level** command allows you to create the lowest level of summary records, corresponding to the smallest subgroup. If you want to create higher level summary records (for instance for grand totals) hold down the **Option** key (Mac) or **Alt** key (PC) while you choose this command. As long as you hold down the **Option** / **Alt** key, each click will increase the summary level of the record. You can continue increasing the level until you get to the maximum level (seven). You can see the blue background get darker as the level increases.

There is no way to go down one level at a time, but by releasing the **Option** / **Alt** key and choosing the **Toggle Summary Level** command the line will turn back into an ordinary data record and you can start over.

If you want to select the summary records you have manually created, use the **Select Summaries** command. See "The Select Summaries Command" on page 388 for details on this command.

STEP 2 - CALCULATE

Once the database has been arranged into groups, the next step is to calculate the summary information. Panorama's Math menu has 7 different kinds of summary calculations).



To perform a calculation, first pick the field you want to calculate by clicking on it. Then choose the command (**Total**, **Average**, etc.) from the Math menu. Panorama will calculate the summaries for each sub-group, group, and for the entire database.

Total

The **Total** command adds up the data in the current field. It calculates subtotals for each group and the grand total for the entire database. The **Total** command can only be used with numeric fields. If you attempt to total a text, date, or choice field, Panorama will display a warning message.

Count

The **Count** command counts the number of non-empty data cells in the current field. If the database is arranged into groups, it will also count the number of non-empty data cells in each group. Empty data cells will not be counted. You can count any field containing either text or numbers, but dates cannot be counted. (Date fields cannot be counted because Panorama would be unable to correctly display the result.)

Average

The **Average** command averages the data in the current field, calculating sub averages for each group and the overall average for the entire database. Averages can only be computed for numeric and date fields. If you attempt to average a non-numeric field Panorama will display a warning message.

Minimum

The **Minimum** command finds the smallest value in the current field. If the database is arranged into groups, it will find the smallest value in each group and sub-group. The **Minimum** command can be used with text, numeric or date fields.

Maximum

The **Maximum** command finds the largest value in the current field. If the database is arranged into groups, it will find the largest value in each group and sub-group. The **Maximum** command can be used with text, numeric or date fields.

Recalculating Summaries

Summaries are not re-calculated automatically when the database changes. If the information in the database changes, you must go back and use the Math menu to re-calculate. If the summary records have been deleted, or if the categories have changed, you must remove the summary records and re-group the database before you can re-calculate the new summary values.

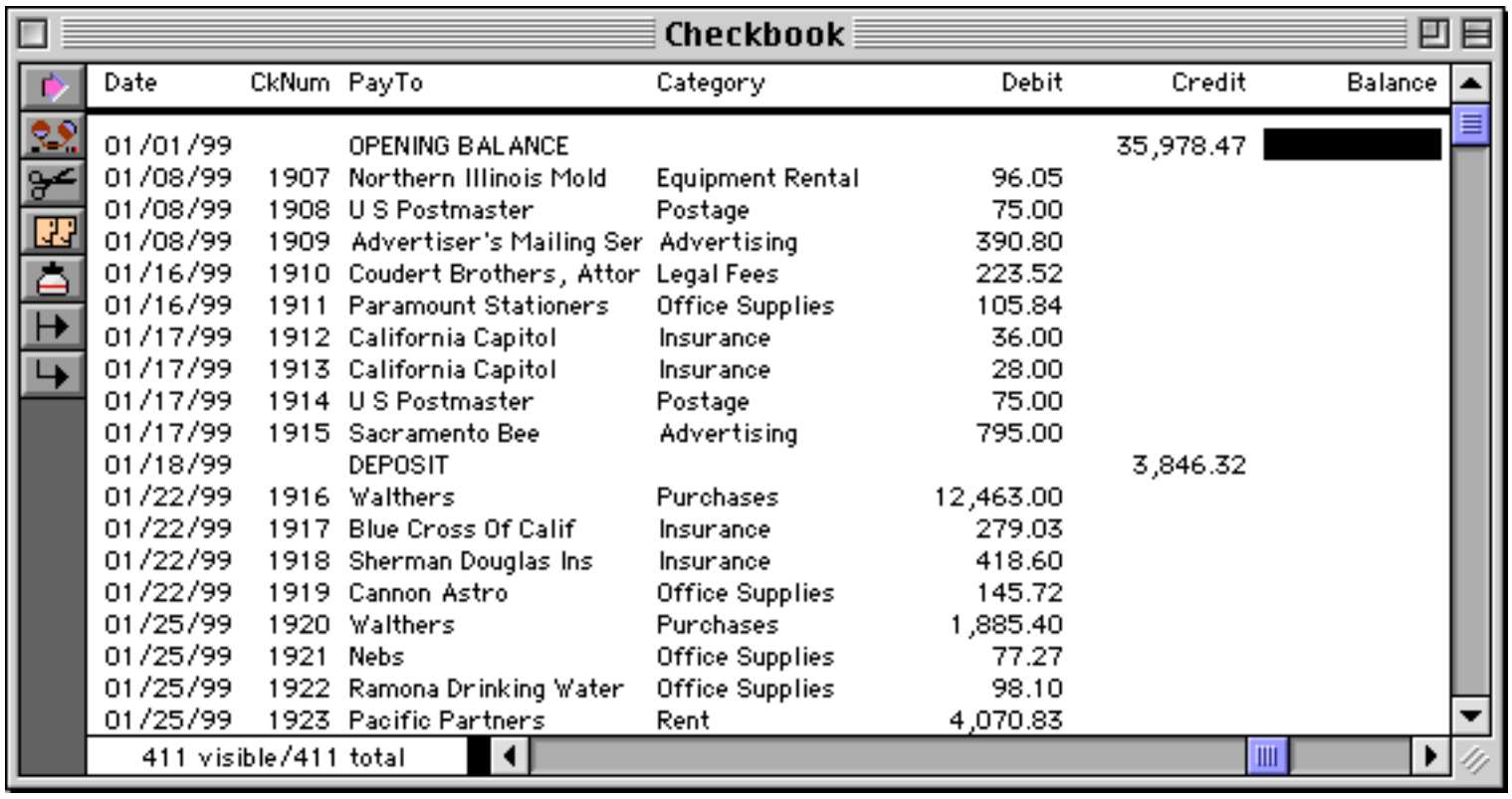
Running Total

Running Total is a special computation. Unlike the other summary calculations, **Running Total** modifies every data cell in the active field, not just the summary records. Like the **Total** computation, **Running Total** starts at the top of the database and adds up each data cell as it moves down the column. **Running Total**, however, replaces each data cell with the current total. The result is a field which contains the cumulative total at each point in the database. This is very useful for computing checking account balances, sales year to date, and other cumulative statistics.

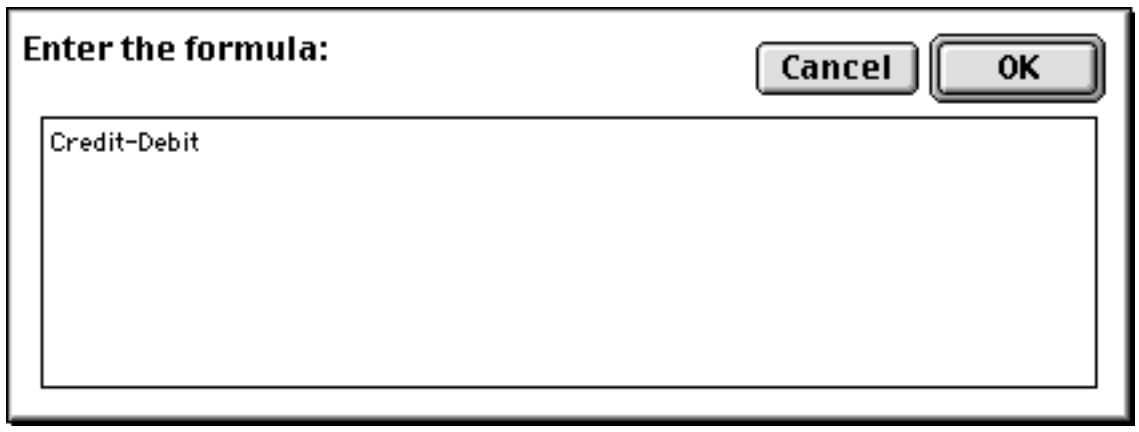
If you use the **Running Total** command on your raw data, the raw data will be destroyed in the process of calculating the running total. We recommend that you avoid this problem by creating an extra field to hold the running total. You can use the **Formula Fill** command to copy the data into the field, and then use the **Running Total** command without disturbing the original raw data.

Using Running Total to Balance a Checkbook

Let's take a look at how to balance a checkbook using the **Running Total** command. Start with an empty Balance field.



Use the **Formula Fill** command to calculate the how much the balance changes for each line. The formula is `Credit-Debit`.



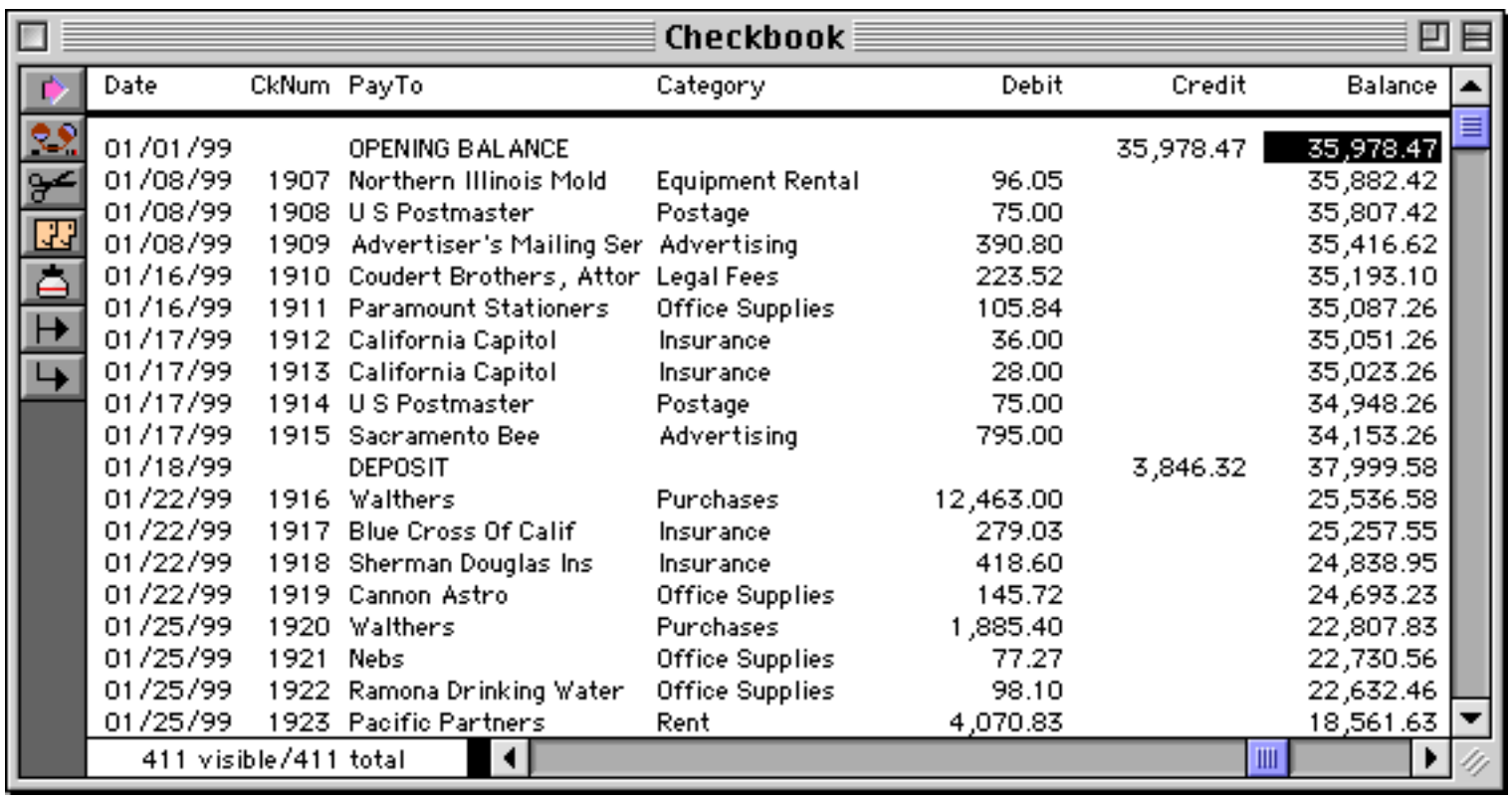
Press the **OK** button to calculate the formula.

Checkbook

Date	CkNum	PayTo	Category	Debit	Credit	Balance
01/01/99		OPENING BALANCE			35,978.47	35,978.47
01/08/99	1907	Northern Illinois Mold	Equipment Rental	96.05		-96.05
01/08/99	1908	U S Postmaster	Postage	75.00		-75.00
01/08/99	1909	Advertiser's Mailing Ser	Advertising	390.80		-390.80
01/16/99	1910	Coudert Brothers, Attor	Legal Fees	223.52		-223.52
01/16/99	1911	Paramount Stationers	Office Supplies	105.84		-105.84
01/17/99	1912	California Capitol	Insurance	36.00		-36.00
01/17/99	1913	California Capitol	Insurance	28.00		-28.00
01/17/99	1914	U S Postmaster	Postage	75.00		-75.00
01/17/99	1915	Sacramento Bee	Advertising	795.00		-795.00
01/18/99		DEPOSIT			3,846.32	3,846.32
01/22/99	1916	Walthers	Purchases	12,463.00		-12,463.00
01/22/99	1917	Blue Cross Of Calif	Insurance	279.03		-279.03
01/22/99	1918	Sherman Douglas Ins	Insurance	418.60		-418.60
01/22/99	1919	Cannon Astro	Office Supplies	145.72		-145.72
01/25/99	1920	Walthers	Purchases	1,885.40		-1,885.40
01/25/99	1921	Nebs	Office Supplies	77.27		-77.27
01/25/99	1922	Ramona Drinking Water	Office Supplies	98.10		-98.10
01/25/99	1923	Pacific Partners	Rent	4,070.83		-4,070.83

411 visible/411 total

Now choose the **Running Total** command from the Math menu to calculate the balance after each transaction.



You'll need to repeat these steps when new transactions are added, or existing transactions changed. This process can be automated with a procedure.

```
field Balance
formulafill Credit-Debit
runningtotal
```

See "Procedures" on page 1339 for more information on creating procedures.

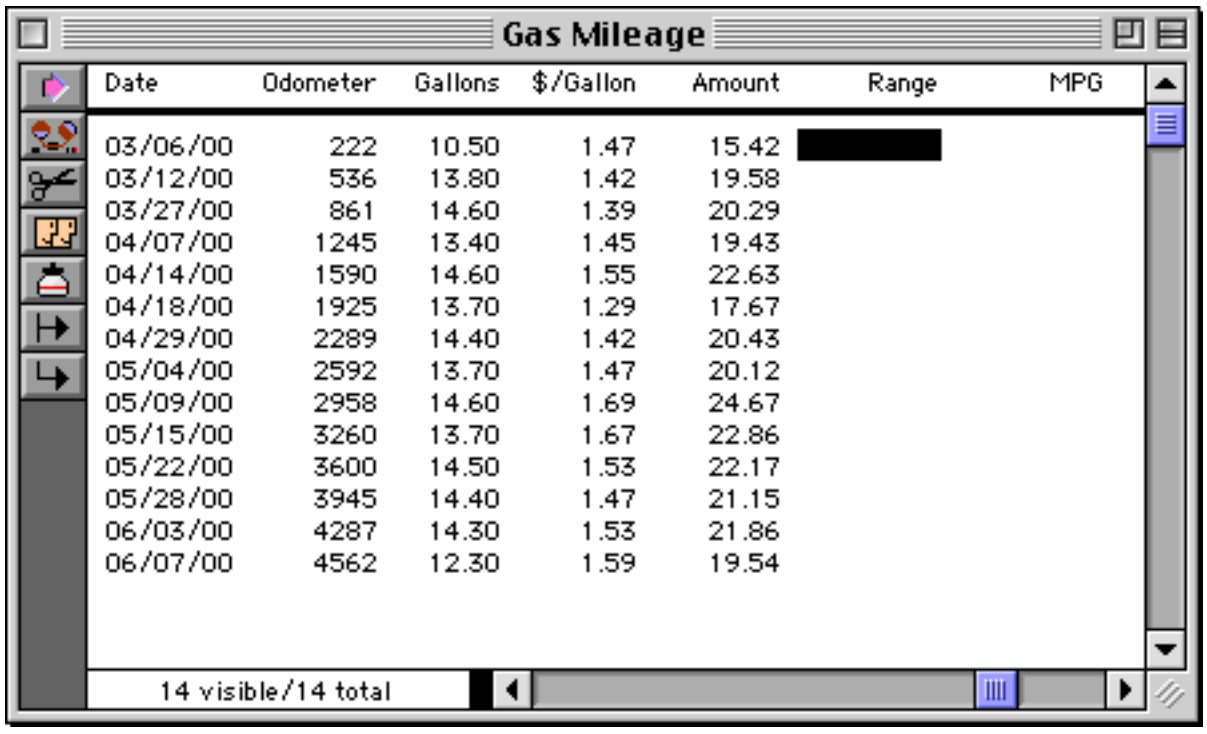
Running Difference

Running Difference is the opposite of **Running Total**. **Running Difference** fills each data cell with the difference between the cell and the cell above it. Use **Running Difference** when you want to compute the spread or interval between consecutive values, for example odometer readings or dates.

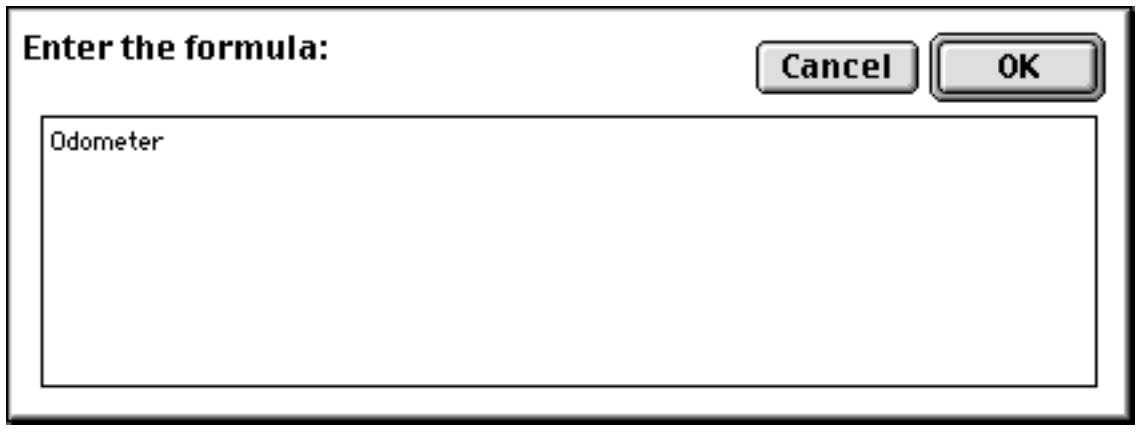
If you use the **Running Difference** command on your raw data, the raw data will be destroyed in the process of calculating the running difference. We recommend that you avoid this problem by creating an extra field to hold the running difference. You can use the **Formula Fill** command to copy the data into the field, and then use the **Running Difference** command without disturbing the original raw data.

Using Running Difference to Calculate Gas Mileage

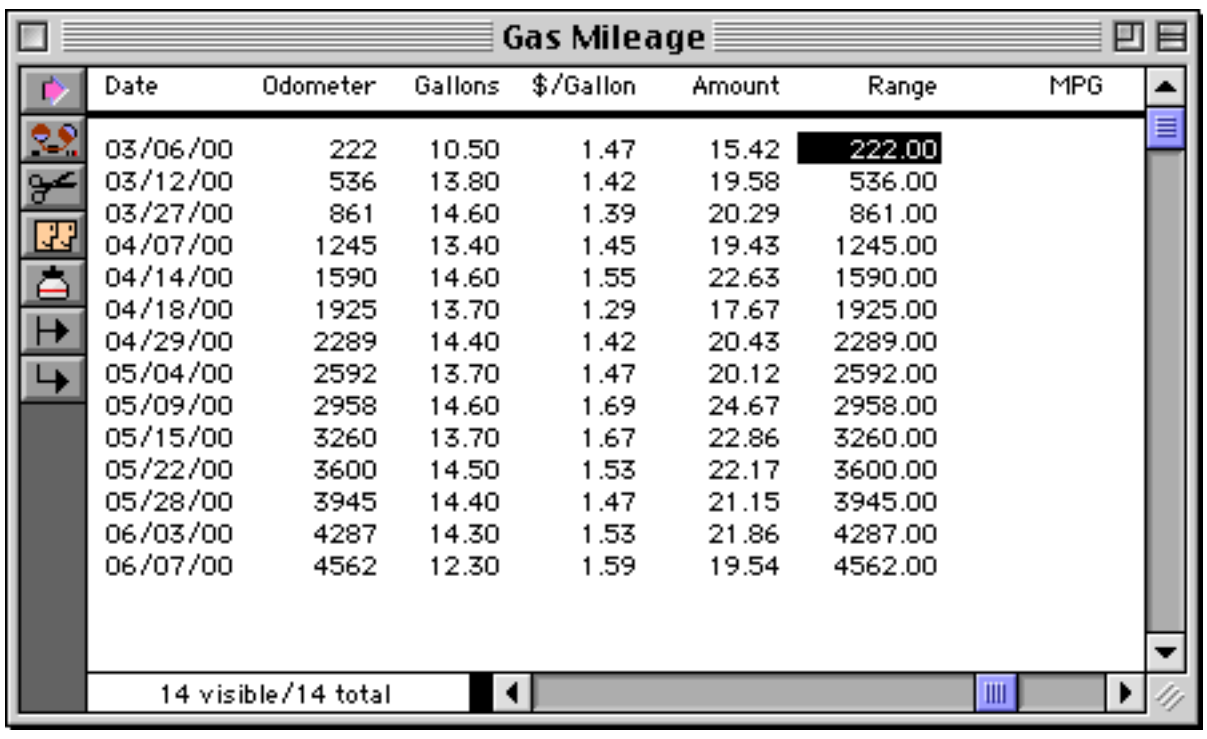
Let's take a look at how to balance a checkbook using the **Running Difference** command. Start with empty Range and MPG fields.



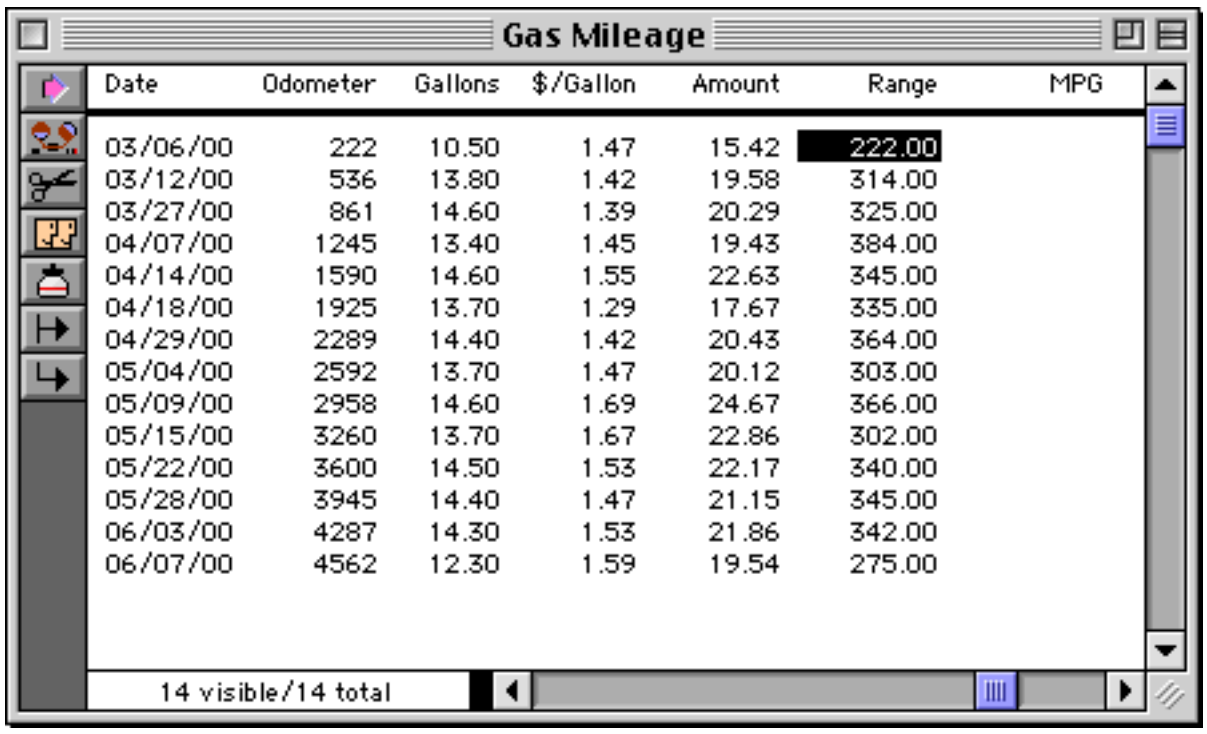
Use the **Formula Fill** command to copy the Odometer field into the Range field. The formula is simply `Odometer`.



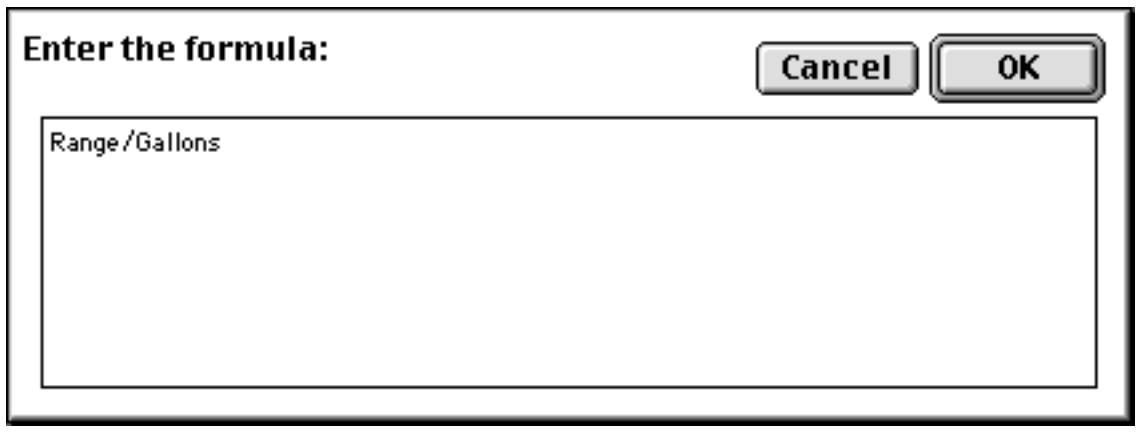
Press **OK** to copy the field.



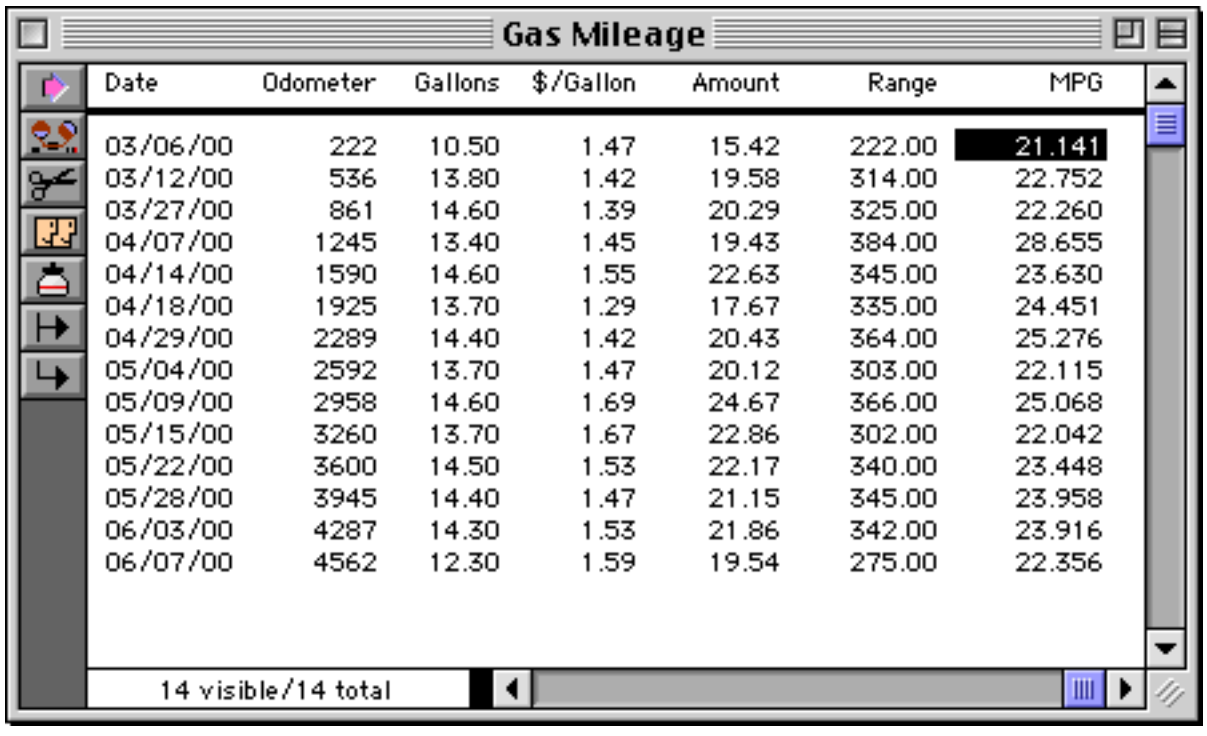
Now use the **Running Difference** command to convert the odometer readings into the distance between fill-lups.



Now we'll calculate the actual miles per gallon. Move to the MPG field and use the **Formula Fill** command to calculate the formula `Range/Gallons`.



Press OK and presto, the miles per gallon between each fill-up is calculated.



You'll need to repeat these steps periodically as you continue to drive. This process can be automated with a procedure.

```
field Range
formulafill Odometer
runningdifference
field MPG
formulafill Range/Gallons
```

See "Procedures" on page 1339 for more information on creating procedures.

STEP 3 - OUTLINE

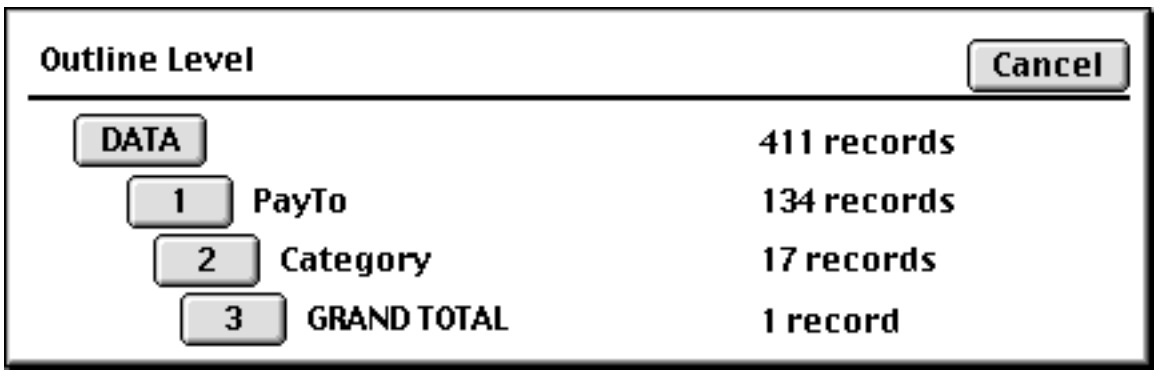
Outlines are a way of organizing information into groups within groups. Panorama's group commands rearrange your database into an outline structure. Tip: Unlike most outlines which start from the top, Panorama's outline is upside-down…it starts from the bottom.

Unlike a paper outline, a Panorama outline can be expanded or collapsed to show more or less detail. This makes it easy to spot overall trends in your data, and then zero in on the details behind those trends.

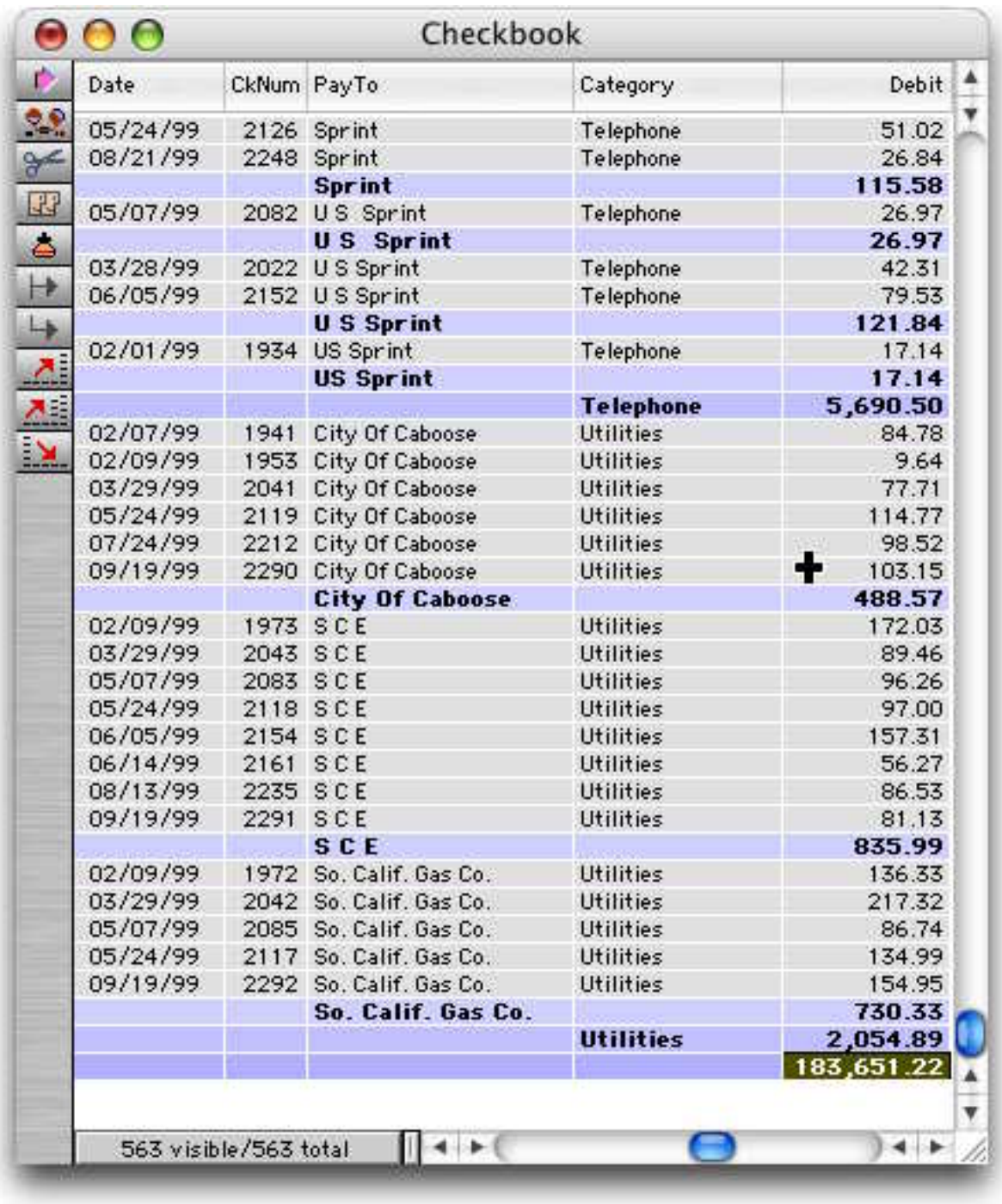
The Outline Level

The **Outline Level** dialog (in the Sort menu) can be used to hide the database detail, leaving only the summary information visible. The **Outline Level** dialog displays a diagram of the database outline structure as it is currently grouped. The diagram shows each level of the database, where it came from, and how much data it contains.

Like the **Find/Select** command (see "The Find / Select Dialog" on page 369), the **Outline Level** dialog makes part of the database temporarily vanish. The **Outline Level** dialog allows you to hide all data below a certain summary level, leaving only the higher summary levels. The numbered buttons on the left side of the dialog allow you to choose how much detail you want to see. Low numbers display more detail, higher numbers display less detail. Press the **Data** button to make everything visible.

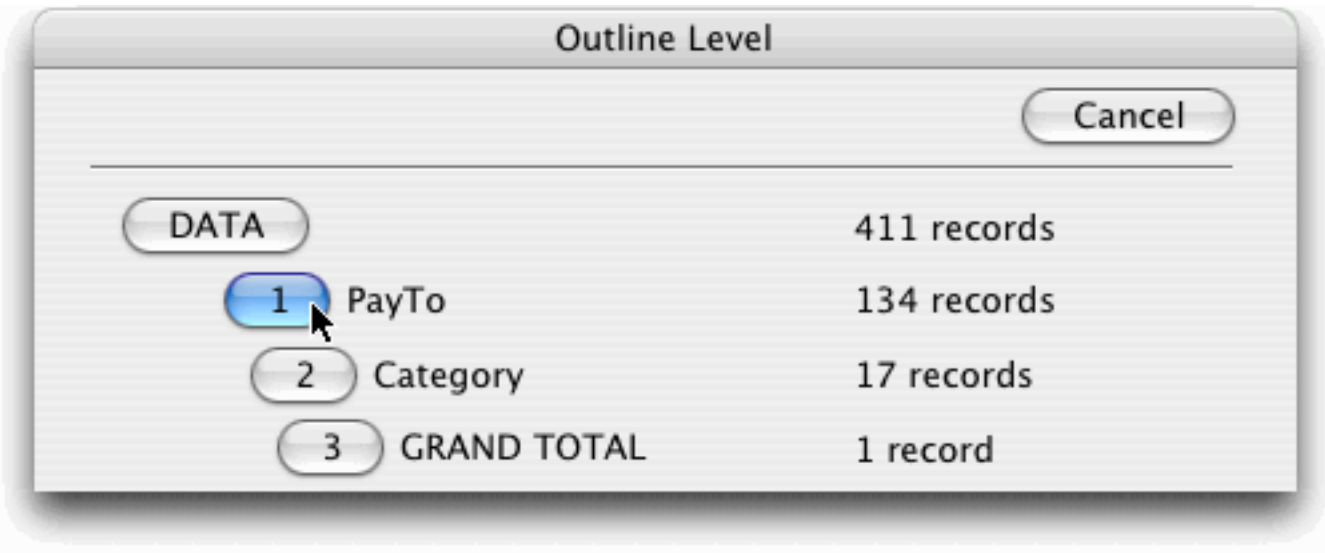


Some examples will help make the operation of this dialog clear. We'll start with a checkbook database that has been grouped by Category and by PayTo name.

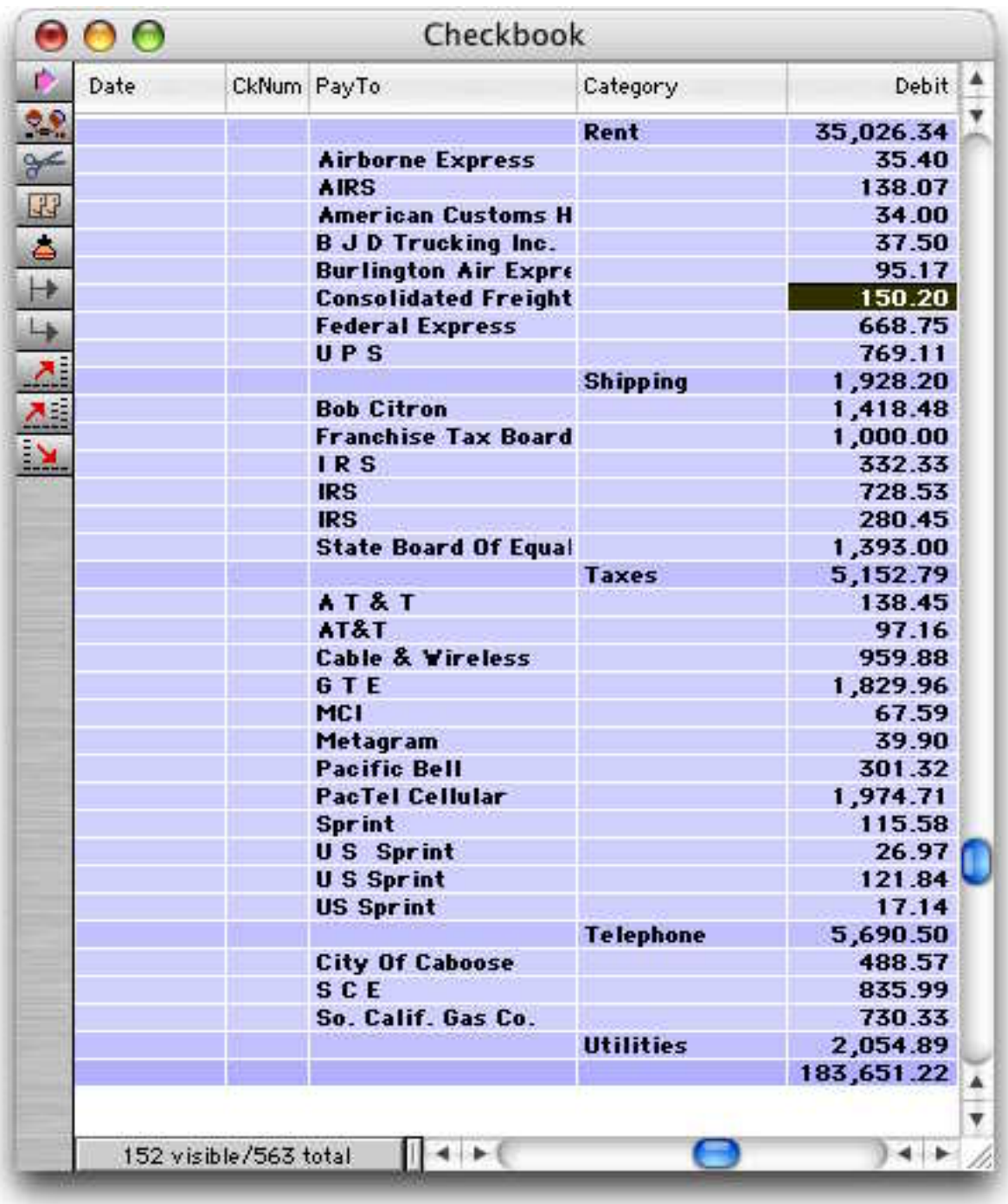


Date	CkNum	PayTo	Category	Debit
05/24/99	2126	Sprint	Telephone	51.02
08/21/99	2248	Sprint	Telephone	26.84
		Sprint		**115.58**
05/07/99	2082	U S Sprint	Telephone	26.97
		U S Sprint		**26.97**
03/28/99	2022	U S Sprint	Telephone	42.31
06/05/99	2152	U S Sprint	Telephone	79.53
		U S Sprint		**121.84**
02/01/99	1934	US Sprint	Telephone	17.14
		US Sprint		**17.14**
			Telephone	**5,690.50**
02/07/99	1941	City Of Caboose	Utilities	84.78
02/09/99	1953	City Of Caboose	Utilities	9.64
03/29/99	2041	City Of Caboose	Utilities	77.71
05/24/99	2119	City Of Caboose	Utilities	114.77
07/24/99	2212	City Of Caboose	Utilities	98.52
09/19/99	2290	City Of Caboose	Utilities	103.15
		City Of Caboose		**488.57**
02/09/99	1973	S C E	Utilities	172.03
03/29/99	2043	S C E	Utilities	89.46
05/07/99	2083	S C E	Utilities	96.26
05/24/99	2118	S C E	Utilities	97.00
06/05/99	2154	S C E	Utilities	157.31
06/14/99	2161	S C E	Utilities	56.27
08/13/99	2235	S C E	Utilities	86.53
09/19/99	2291	S C E	Utilities	81.13
		S C E		**835.99**
02/09/99	1972	So. Calif. Gas Co.	Utilities	136.33
03/29/99	2042	So. Calif. Gas Co.	Utilities	217.32
05/07/99	2085	So. Calif. Gas Co.	Utilities	86.74
05/24/99	2117	So. Calif. Gas Co.	Utilities	134.99
09/19/99	2292	So. Calif. Gas Co.	Utilities	154.95
		So. Calif. Gas Co.		**730.33**
			Utilities	**2,054.89**
				183,651.22

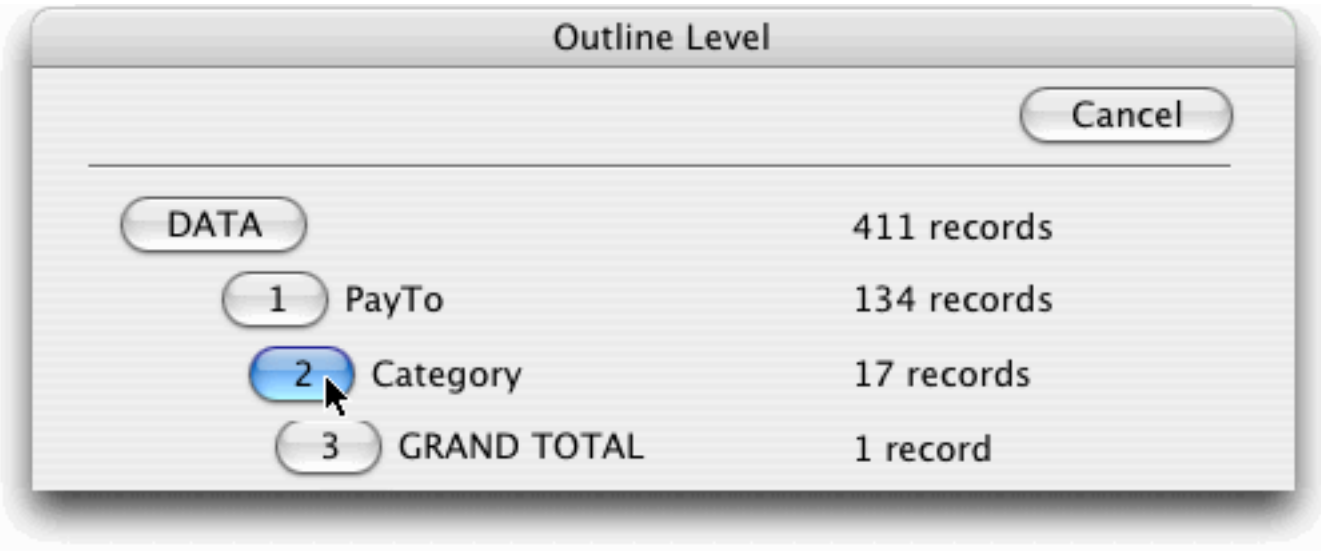
Now open the **Outline Level** dialog…



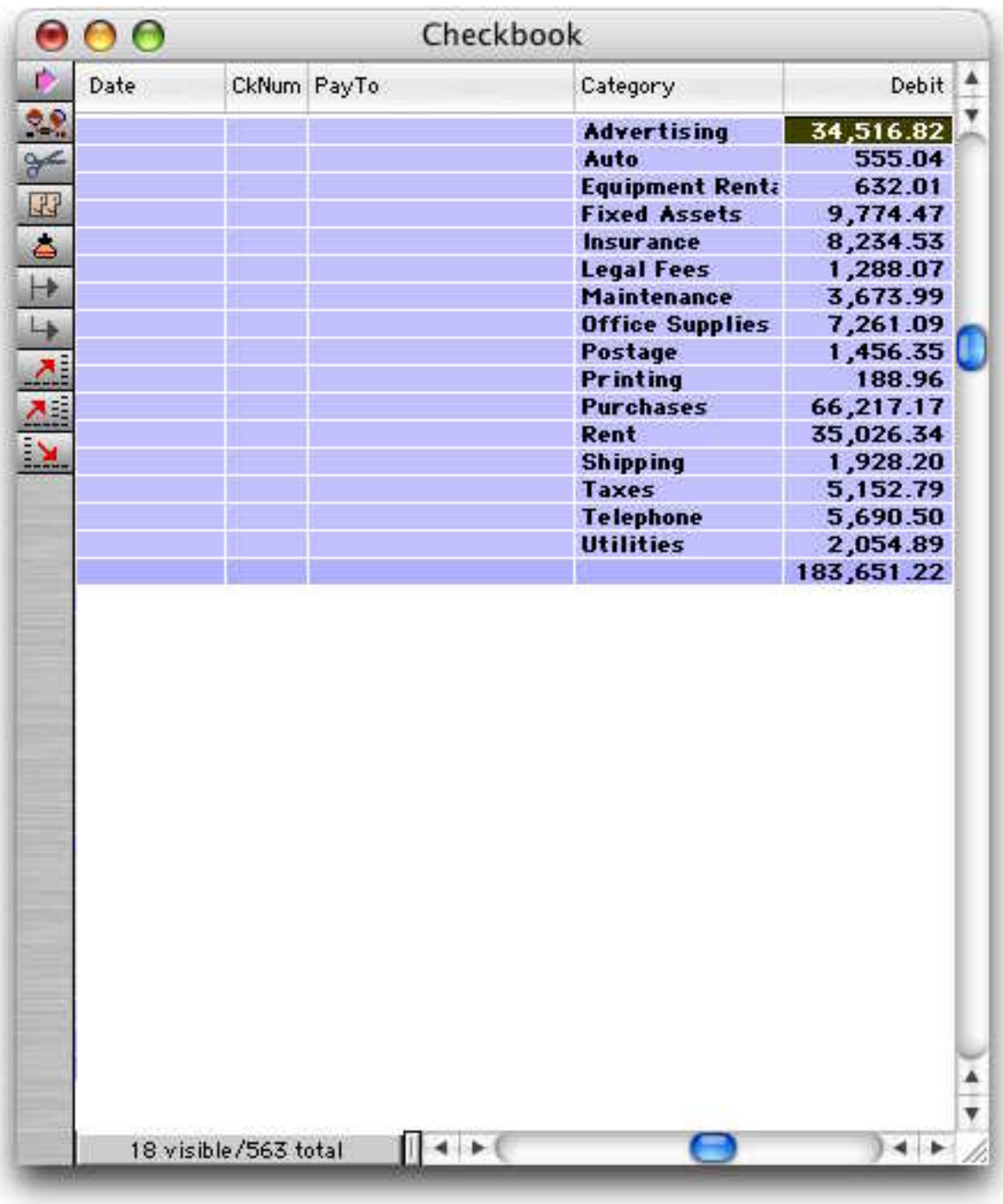
Press **1** to collapse the database so that none of the raw data is visible, only summary records.



Now open the **Outline Level** dialog again to collapse further…



Press **2** to collapse the database so that only the category subtotals and grand totals are visible.



If you want to collapse the database completely so that only the Grand Total is visible, press **3**. Or you can go back to showing all of the data by pressing the **Data** button.

Collapsing vs. Selecting

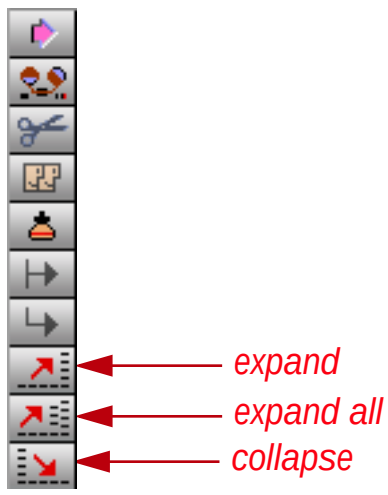
Collapsing and selecting are very similar in one way—both hide data from view. In spite of this similarity, collapsing and selecting are independent of each other. Data that is selected can be invisible because it is collapsed, and data that is expanded can be invisible because it is not selected. Data is only visible when it is both selected and expanded.

This distinction can sometimes be confusing, since you can't always tell at a glance why data is invisible. Suppose you select a subset of the database, and then collapse the database outline. Some of the data is invisible because it is de-selected. Some of the data is invisible because it is collapsed.

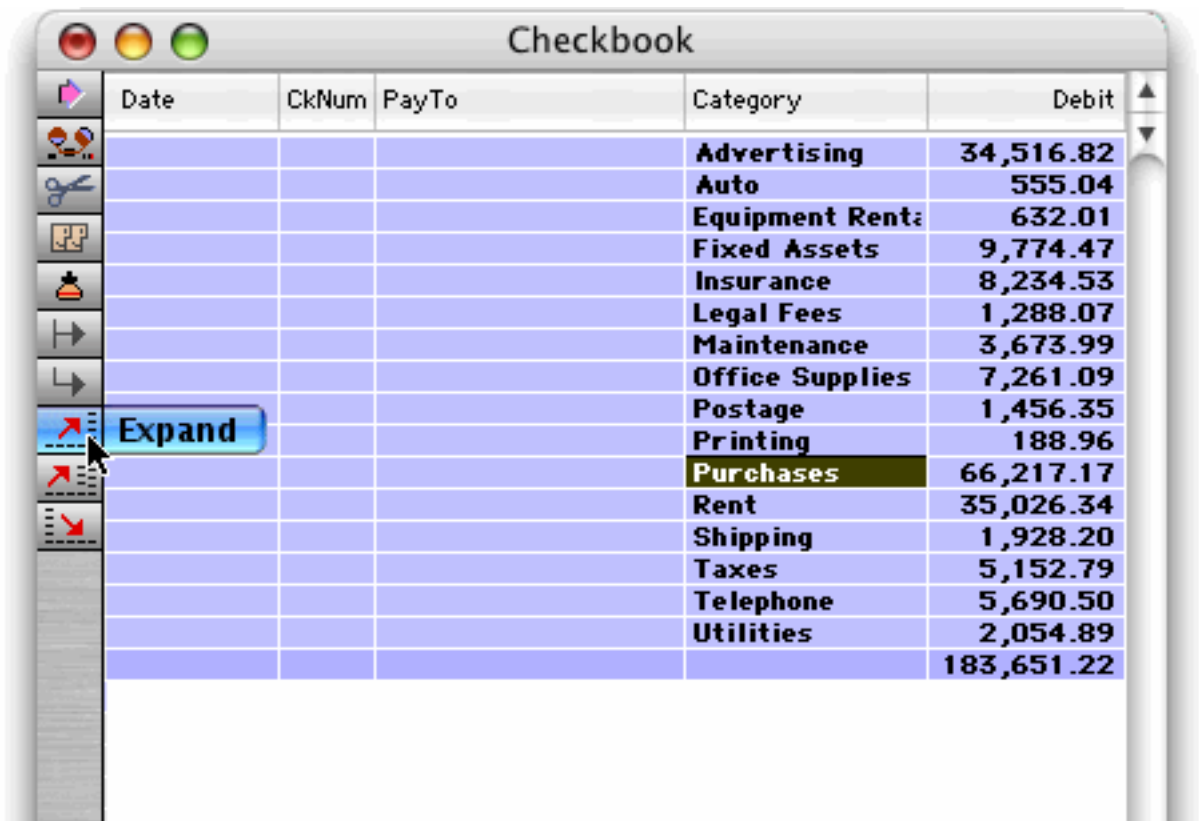
To make sure that all the data is visible, you must use both the **Outline Level** command to expand all the data and the **Select All** command to select all the data.

Expanding and Collapsing Specific Details

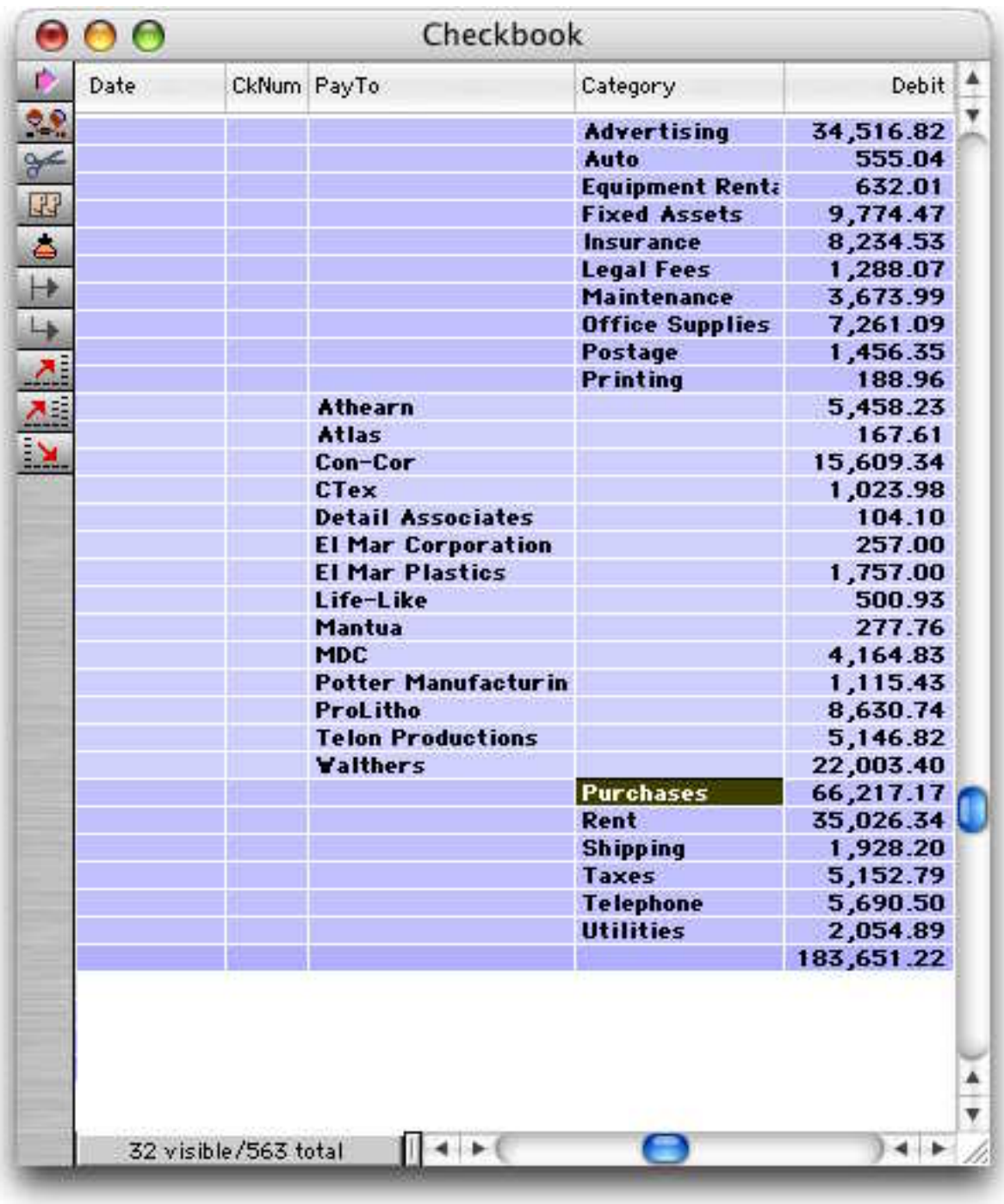
The **Outline Level** dialog expands and collapses the entire database. You can use the **Expand**, **Expand All**, and **Collapse** tools to expand or collapse a specific area of the database. These tools automatically appear when summary records are added to your database, and disappear when they are removed.



To expand a specific summary, click on the summary record you want to expand and then press the **Expand** tool. This tool expands the next level of detail for that summary record only. To illustrate this we'll use our checkbook database which has been grouped by Category and PayTo (see "STEP 1 - GROUP" on page 399), and collapsed to show only the Category subtotals (see "The Outline Level" on page 409). To expand a specific category, click on the line and press the **Expand** tool.



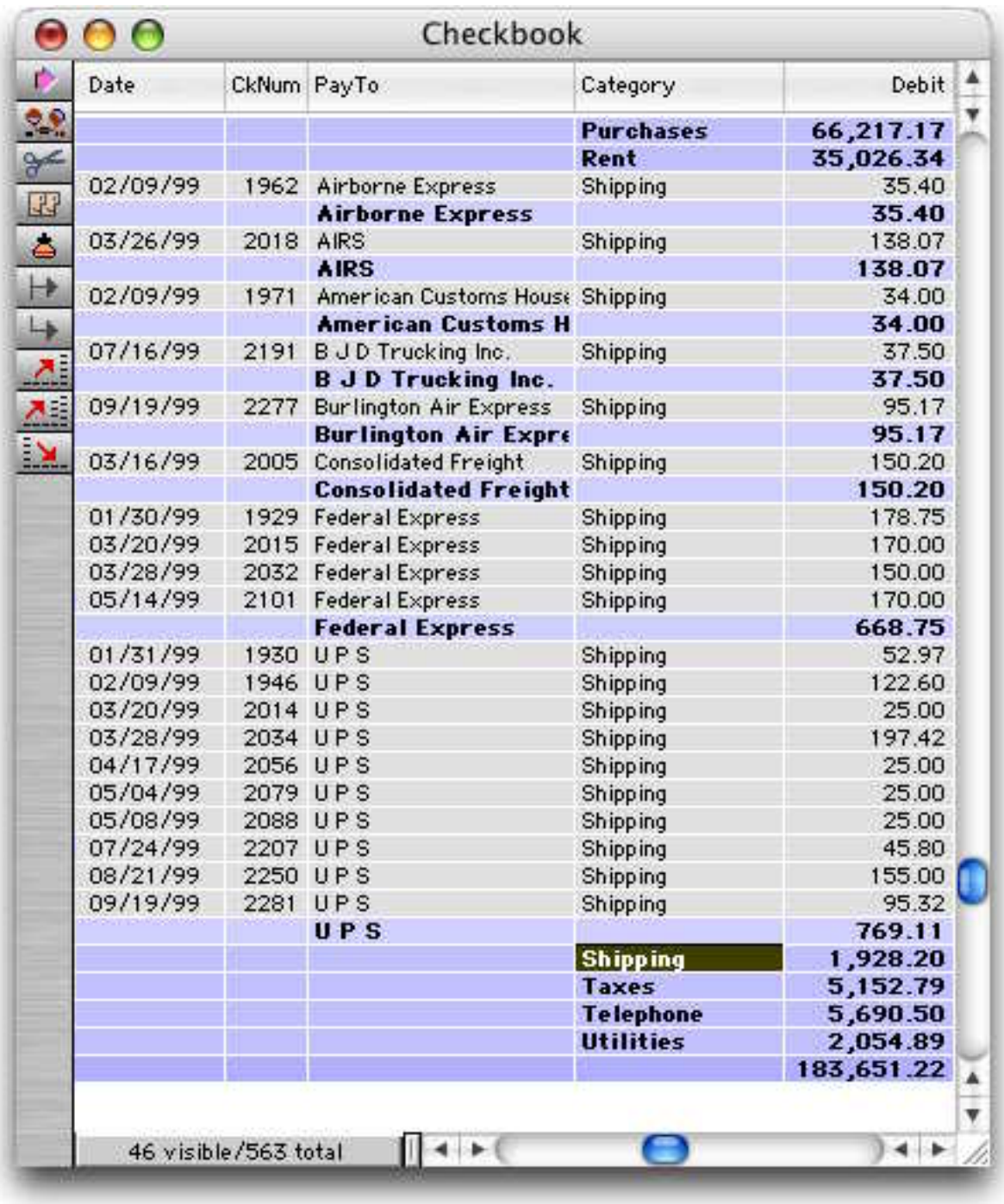
Now you can see all of the subtotals for each PayTo within the Purchases category.



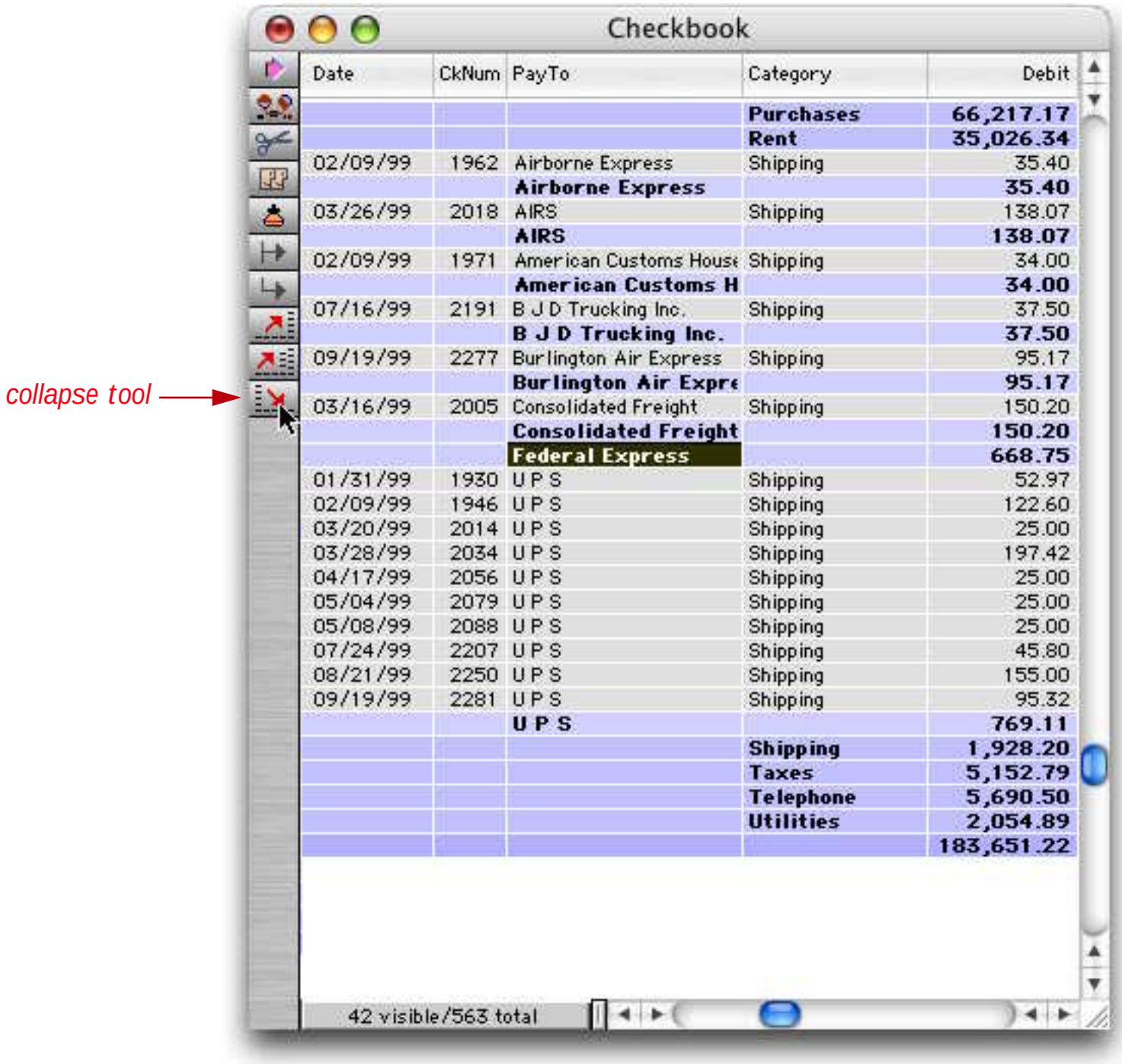
In this example you can expand further. You can always expand until you get to the raw data. Simply click on the subtotal line and press the **Expand** tool.



The **Expand All** tool expands a single summary record all the way to the raw data. It shows you all of the data that went into calculating that summary. In our checkbook example, using **Expand All** on Shipping would reveal every check written for shipping, no matter what carrier was used.



The **Collapse** tool hides the detail associated with a specific summary record. For example, if you collapsed Federal Express, the four checks written to Federal Express would disappear.



Sorting by Summary Value

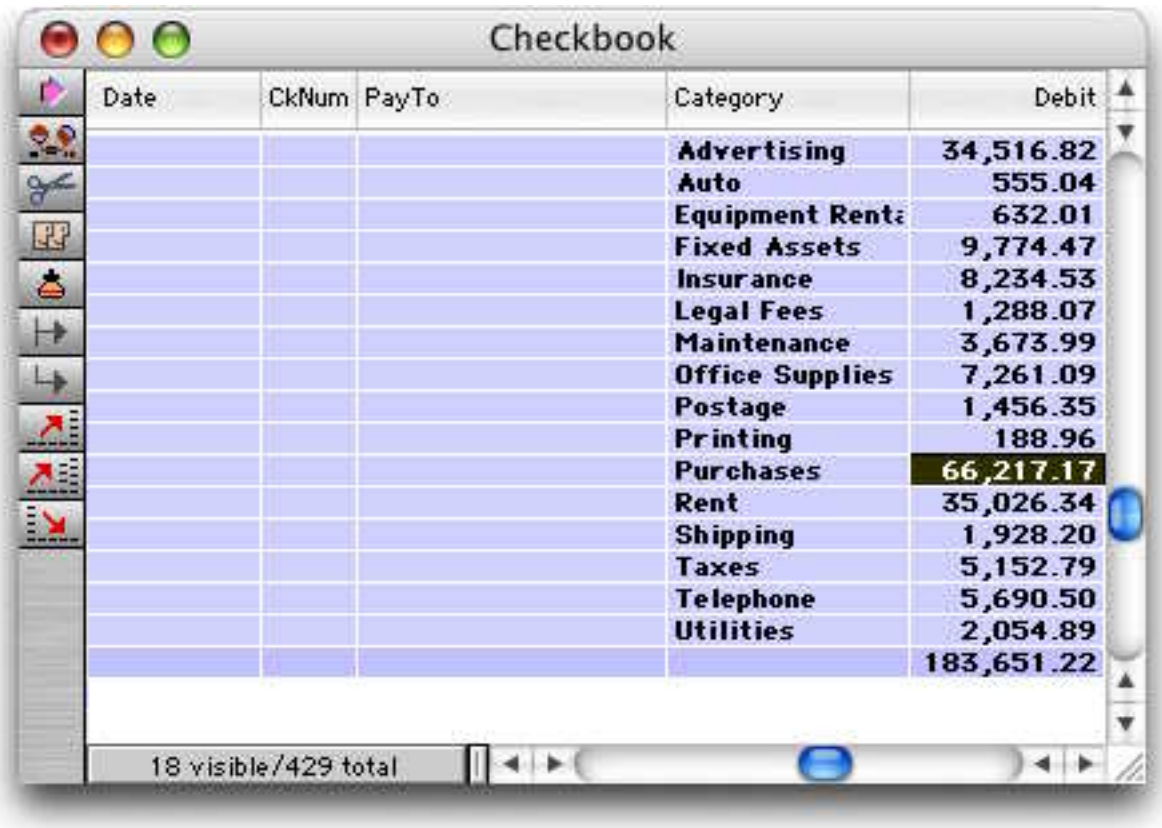
Once summary information has been calculated, you can sort the database so that the data is ranked in order by the summary values. For example, the information in an invoice database can be summarized by customer and then rearranged to rank each customer by sales volume. Once the data is ranked you can see who your top customers are at a glance. The ability to rank summary information is a unique feature of Panorama.

The first step in sorting summary information is to calculate the summary information. To do this use the group (see “STEP 1 - GROUP” on page 399) and summary calculation (see “STEP 2 - CALCULATE” on page 403) commands already covered in this chapter.

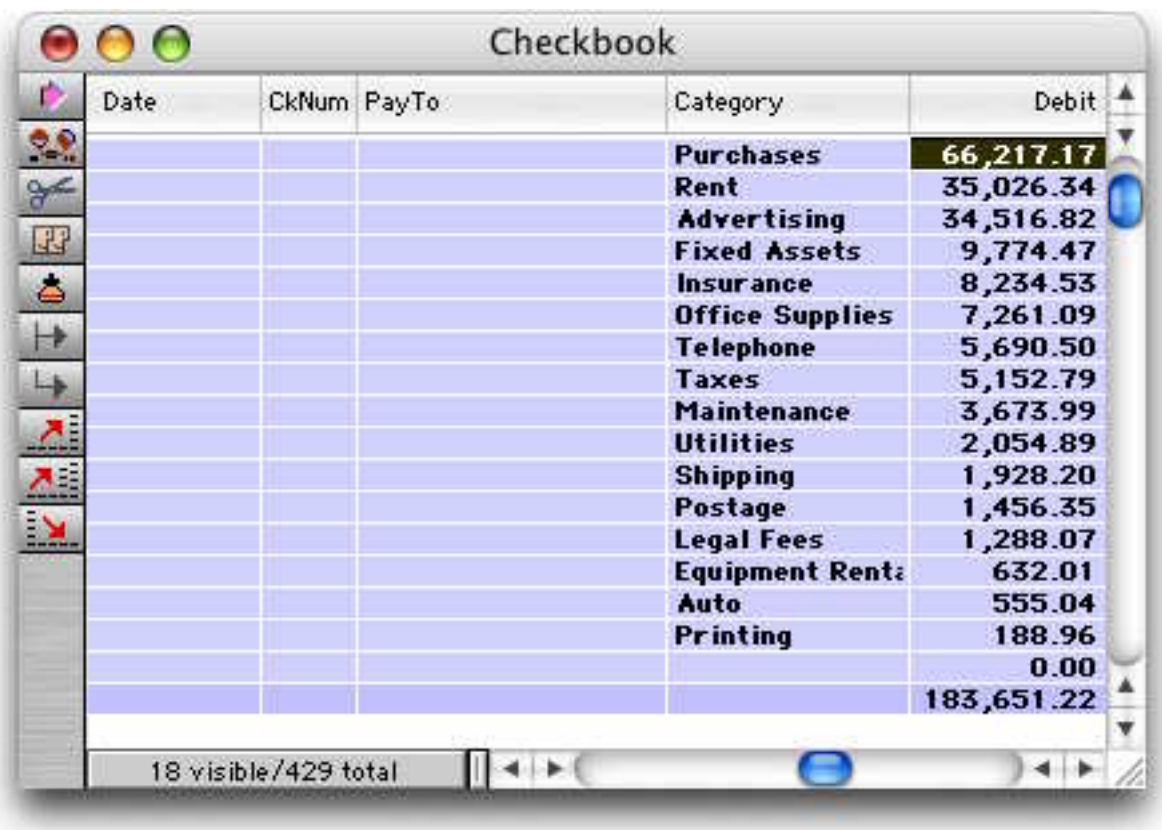
The second step is to collapse the outline. Use the **Outline Level** command to collapse the database to the level you want to sort (see “The Outline Level” on page 409).

Once the database is collapsed, the final step is to sort the database by summary value. Click on the field containing the summary values and then choose **Sort Up** or **Sort Down**. **Sort Down** is the usual choice because it ranks the summaries from highest to lowest.

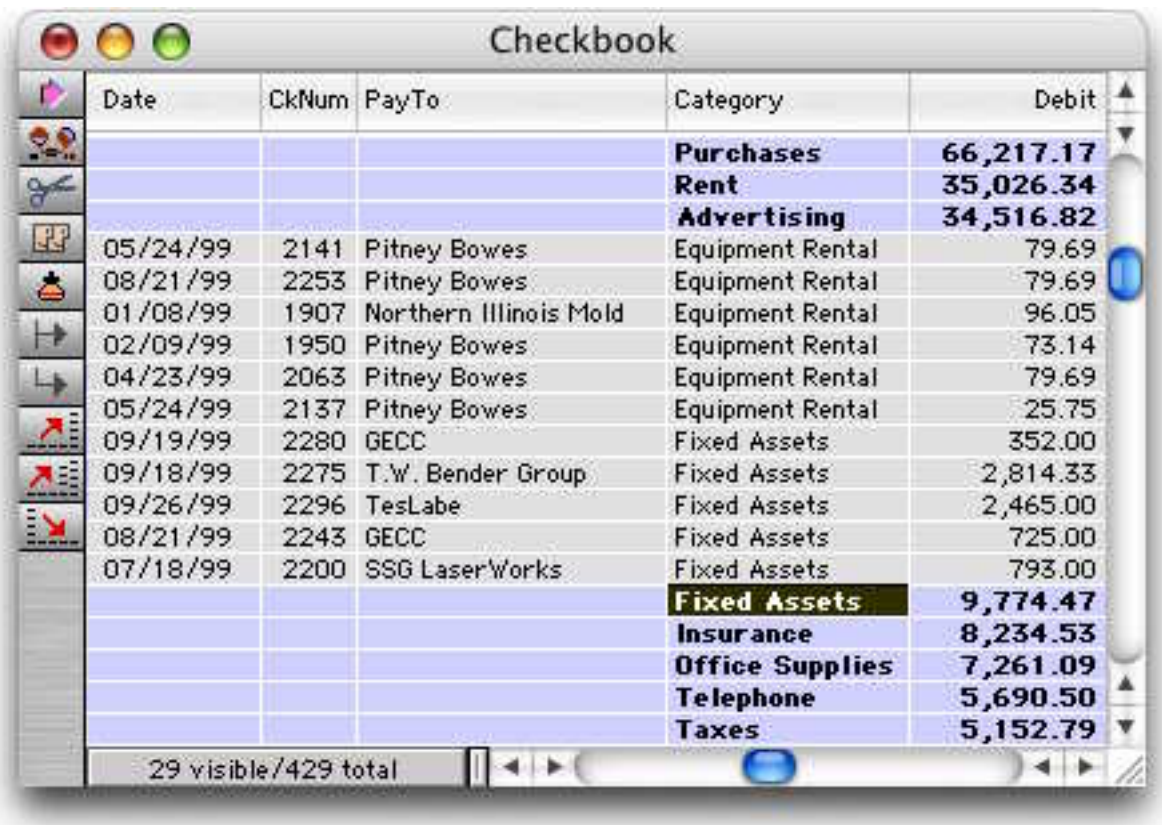
To illustrate this technique, we started by grouping and totalling the checkbook database by category.



As you can see, the subtotals are listed in alphabetical order. Use the **Sort Down** command to sort the subtotals so that they are listed in order from highest to lowest amount.



After the database has been sorted, you can re-expand some or all of the outline if you wish. The detail information for each summary follows the summary as it is sorted. (In other words, when the database is grouped the sort commands sort the groups, not the individual records.)



The real power of this technique appears when the database is grouped by multiple levels. For example, suppose the checkbook database is grouped by month and by category, then collapsed to show the monthly totals for each category. This is shown in the window on the left. On the right is the same database, but now the **Sort Down** command has been used to sort the subtotals. The subtotals have been sorted within each month. Now we can easily see that Purchases was the top spending category in January, March, and April, but Advertising was tops in February (with Purchases dropping to number 3).

before Sort Down...

Checkbook

Date	Category	Debit
	Advertising	2,841.02
	Equipment Rent:	96.05
	Insurance	761.63
	Legal Fees	223.52
	Office Supplies	426.93
	Postage	150.00
	Purchases	17,083.42
	Rent	4,070.83
	Shipping	231.72
	Taxes	549.00
	Telephone	141.09
01/31/99		26,575.21
	Advertising	8,134.13
	Auto	240.21
	Equipment Rent:	73.14
	Fixed Assets	428.39
	Insurance	601.48
	Legal Fees	95.00
	Maintenance	310.00
	Office Supplies	915.50
	Postage	315.00
	Purchases	3,386.78
	Rent	7,742.19
	Shipping	192.00
	Telephone	736.66
	Utilities	402.78
02/28/99		23,573.26
	Advertising	7,501.90
	Auto	33.32
	Insurance	1,539.14
	Maintenance	872.25
	Office Supplies	1,579.74
	Postage	110.00
	Purchases	19,557.48
	Rent	3,566.30
	Shipping	830.69
	Taxes	2,008.98
	Telephone	1,027.01
	Utilities	384.49
03/30/99		39,011.30
	Advertising	1,024.52
	Equipment Rent:	79.69
	Insurance	522.63
	Maintenance	132.00
	Office Supplies	339.46
	Postage	156.35
	Purchases	2,268.19
	Rent	265.00
	Shipping	25.00
	Taxes	734.33
	Telephone	305.56
04/27/99		5,852.73

after Sort Down

Checkbook

Date	Category	Debit
	Purchases	17,083.42
	Rent	4,070.83
	Advertising	2,841.02
	Insurance	761.63
	Taxes	549.00
	Office Supplies	426.93
	Shipping	231.72
	Legal Fees	223.52
	Postage	150.00
	Telephone	141.09
	Equipment Rent:	96.05
01/31/99		26,575.21
	Advertising	8,134.13
	Rent	7,742.19
	Purchases	3,386.78
	Office Supplies	915.50
	Telephone	736.66
	Insurance	601.48
	Fixed Assets	428.39
	Utilities	402.78
	Postage	315.00
	Maintenance	310.00
	Auto	240.21
	Shipping	192.00
	Legal Fees	95.00
	Equipment Rent:	73.14
02/28/99		23,573.26
	Purchases	19,557.48
	Advertising	7,501.90
	Rent	3,566.30
	Taxes	2,008.98
	Office Supplies	1,579.74
	Insurance	1,539.14
	Telephone	1,027.01
	Maintenance	872.25
	Shipping	830.69
	Utilities	384.49
	Postage	110.00
	Auto	33.32
03/30/99		39,011.30
	Purchases	2,268.19
	Advertising	1,024.52
	Taxes	734.33
	Insurance	522.63
	Office Supplies	339.46
	Telephone	305.56
	Rent	265.00
	Postage	156.35
	Maintenance	132.00
	Equipment Rent:	79.69
	Shipping	25.00
04/27/99		5,852.73

This technique is very powerful any time you need to rank summary information. You can quickly answer questions like “Who are our top customers?” “What products had the most service problems last year?” or “Which SKU’s are the best sellers in different seasons of the year?”

Sorting Within Groups

If you sort your database without collapsing it, Panorama will sort the data within each group instead of sorting the entire database. If you want to sort the summary values themselves, you must use the **Outline Level** dialog to collapse the outline, as described in the previous section.

Getting Rid of Summary Records

When you are done with summary records, you can get rid of them with the **Remove Summaries** dialog (Sort menu). This command can either remove all the summaries, or it can selectively remove certain summary levels.

The **Remove Summaries** dialog displays a diagram of the database outline structure. The diagram shows each summary level, how it was created, and the number of records in the level. If you want to get rid of all the summary records, press the button with the highest number or the **Remove All Summaries** button.



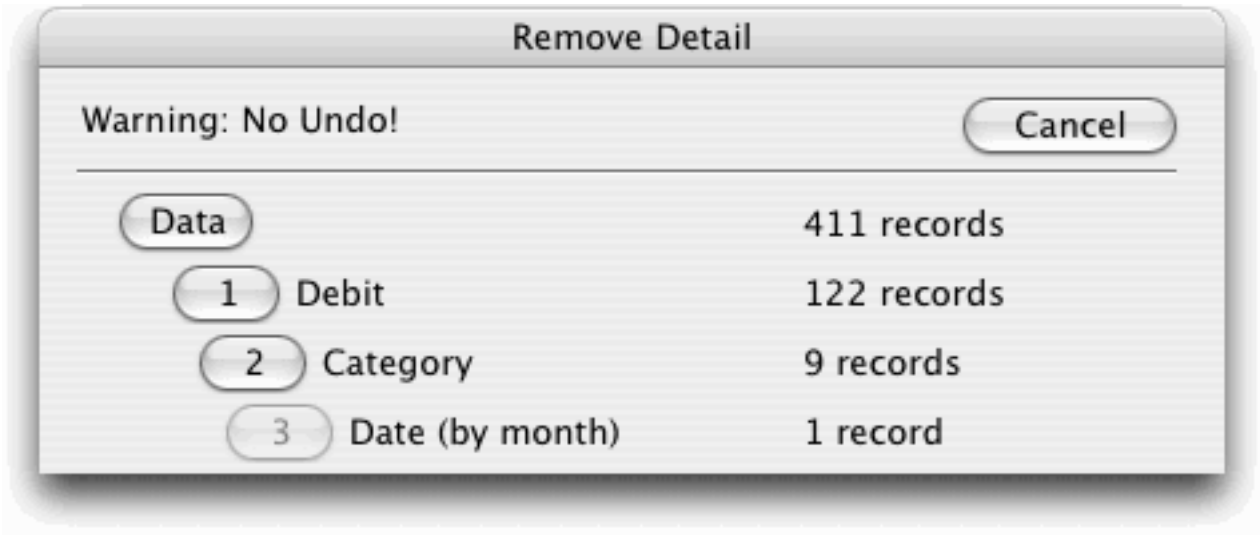
If you press one of the lower numbered buttons, only some of the summaries will be removed. For example if you press **1**, only the lowest summary level will be removed (the smallest subgroups). The remaining summary records will each drop down to the next lowest level.

You can also remove individual summary records with the **Cut Record** tool or the **Delete** or **Backspace** key (see “Deleting a Record” on page 307).

Getting Rid of Detail

Occasionally you may want to completely remove the raw data, leaving only the summary information. The **Remove Detail** dialog (Sort menu) does this for you. You can remove just the data, or you can remove the data along with some of the lower levels of summary information. Warning: This command is extremely dangerous, since it effectively deletes most of your database. Handle with care!

The **Remove Detail** dialog displays a diagram of the database outline structure. The diagram shows each summary level, how it was created, and the number of records in the level. Press one of the buttons on the left to choose the level of detail to remove. If you press the **Data** button, only the raw data will be removed. The level 1 summary records will be converted into data records, and all other summary records will drop down to the next lower level.



If you press one of the numbered buttons (**1** through **6**) Panorama will remove the summary records up to and including that level as well as the data records. The remaining summary records will be adjusted so that the lowest remaining summary level becomes the new data records, etc.

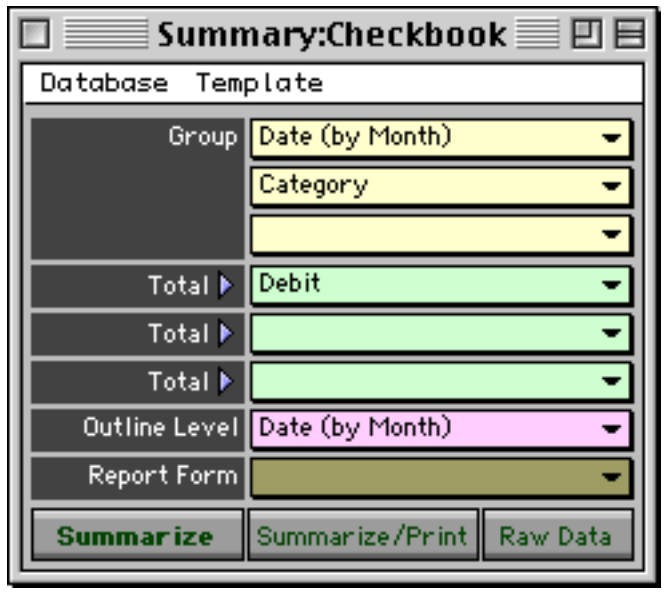
Printing Reports with Summary Information

You'll often want to print reports with the summary information you have generated. In Panorama reports are printed using forms and report tiles (see "Custom Reports" on page 1049). In addition to the standard report tools Panorama also has special features for printing summary information. You can print special headers and footers for each group, and can control how groups break across columns and pages. See "Printing Summary Information" on page 1129 to learn more about how to use these features. You can also use the Summaries & Outline Wizard to help print summaries — see "Printing the Summary Results" on page 427.

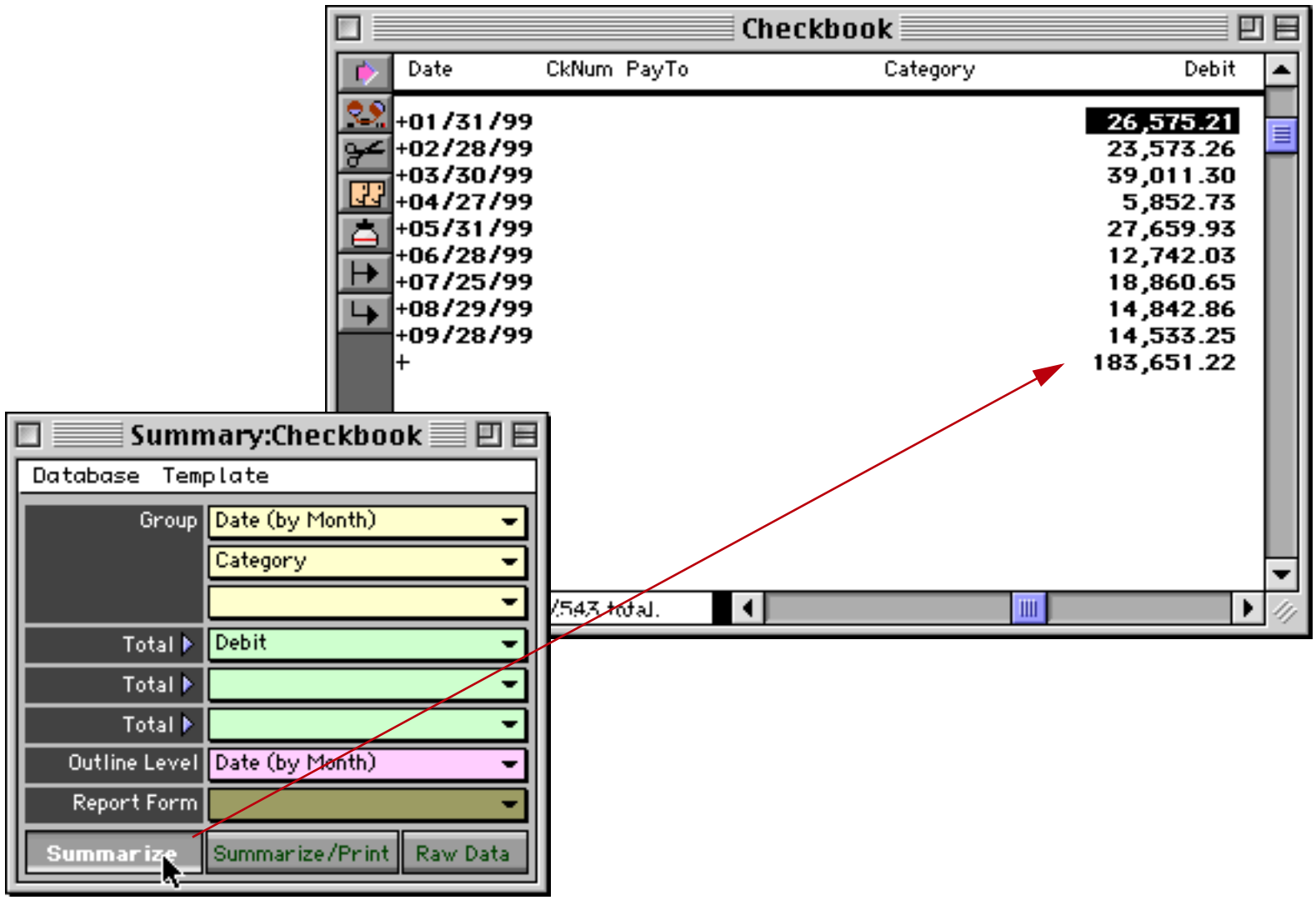
The Summaries & Outlines Wizard

The **Summaries & Outlines Wizard** can automate the process of calculating summaries. In fact, once a template has been set up you can perform all three summary steps (Group / Calculate / Outline) with a single mouse click. To use the **Summaries & Outlines Wizard** start with the database you want to summarize and select the wizard from the **Utilities** submenu of the **Wizard** menu.

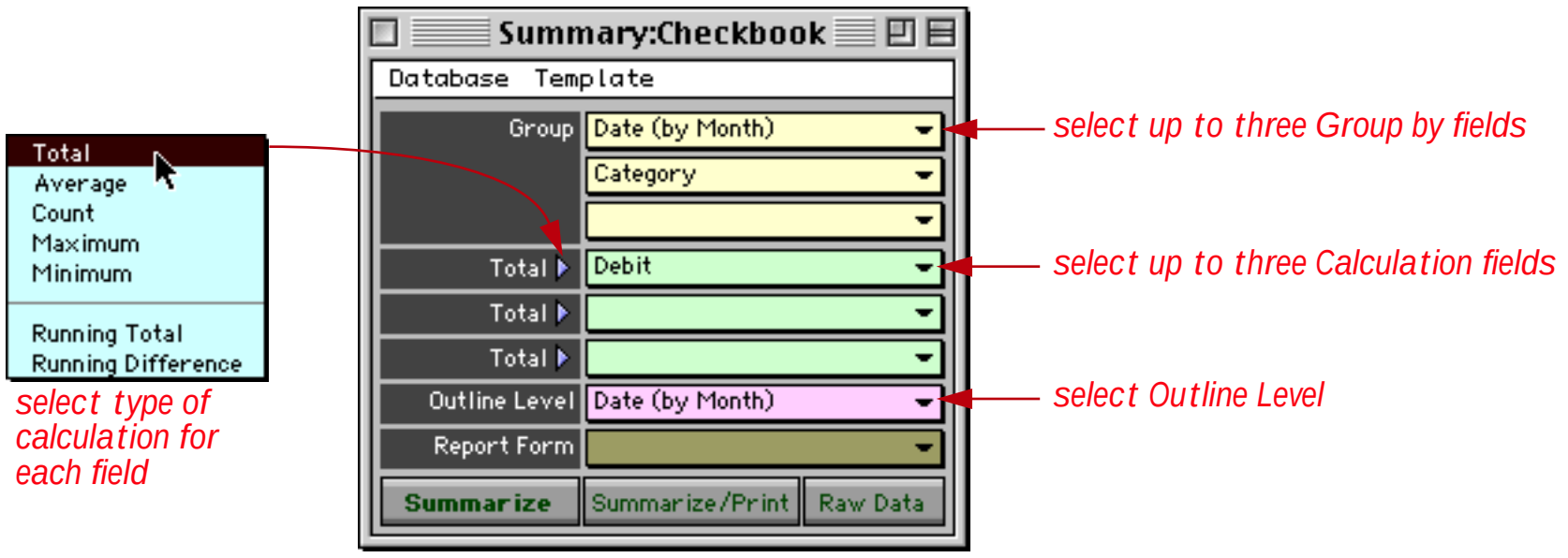
When the wizard first opens it automatically remembers the settings used to summarize this database the last time the wizard was used with this database. (Of course if this is the first time the wizard has been used with this database all of the settings will be blank).



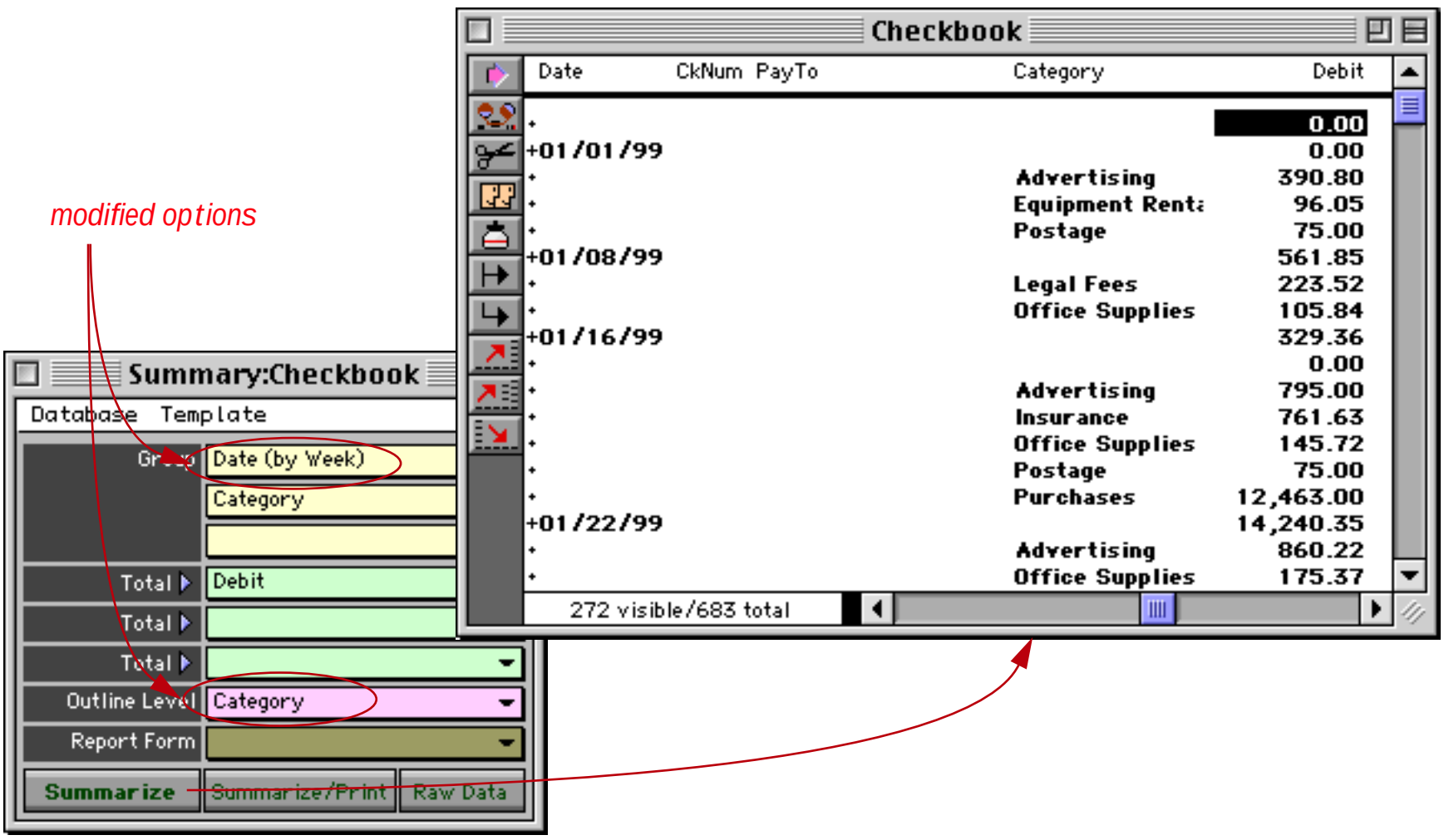
To repeat the previous summarization all you have to do is click on the **Summarize** button.



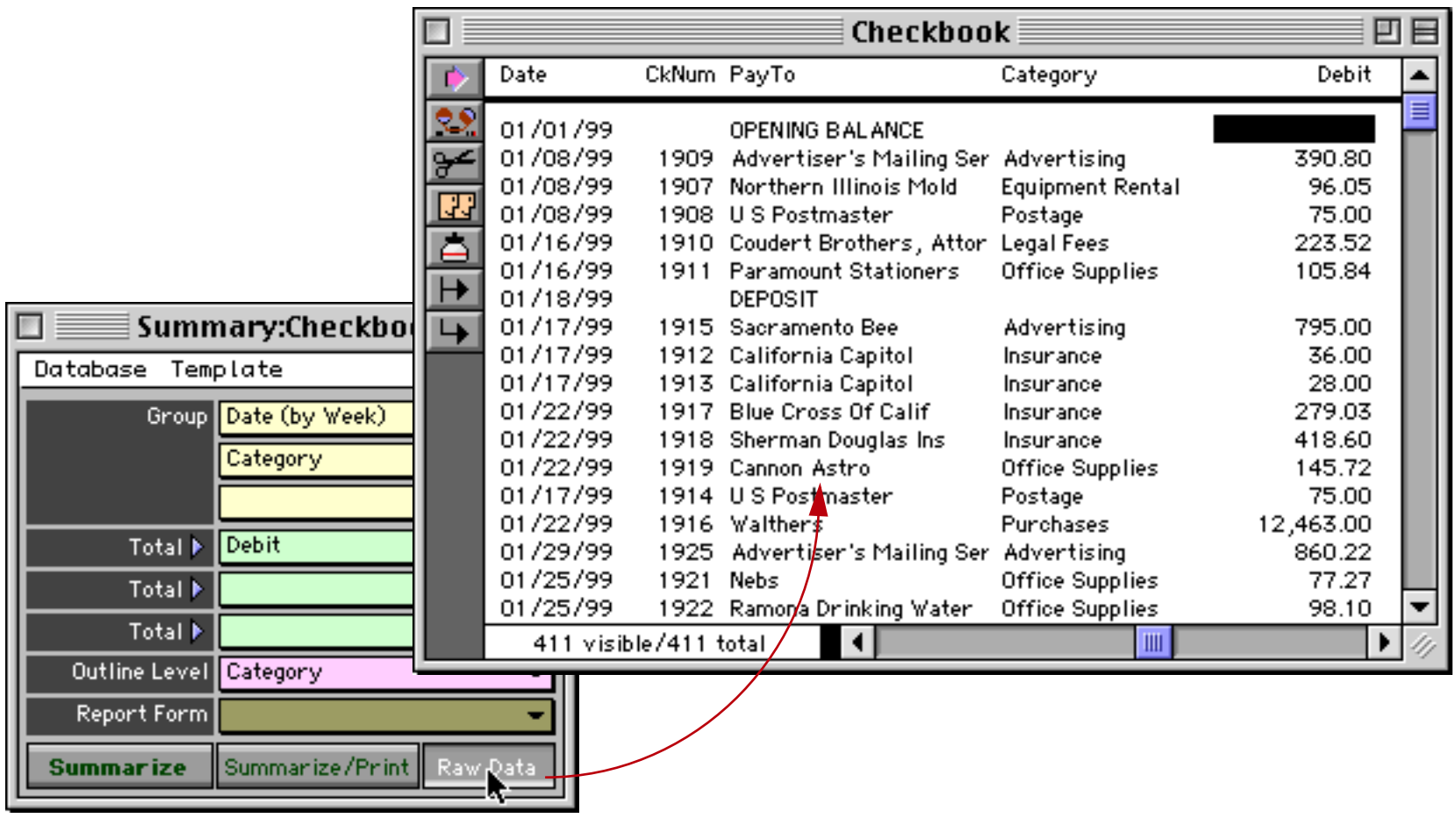
The wizard window is divided into three sections that correspond to the three steps in the summary process. The top section allows you to group the database by up to three fields (see "STEP 1 - GROUP" on page 399). The middle section allows you to calculate totals, averages, counts, etc. for up to three fields (see "STEP 2 - CALCULATE" on page 403). The bottom section allows you to decide what summary level to show after the calculations are complete (see "STEP 3 - OUTLINE" on page 409).



After you change one or more options press the **Summarize** button to see the new results. It's not necessary to remove the old summaries before calculating the new summaries — the wizard will do it for you. In the illustration below two options have been changed.

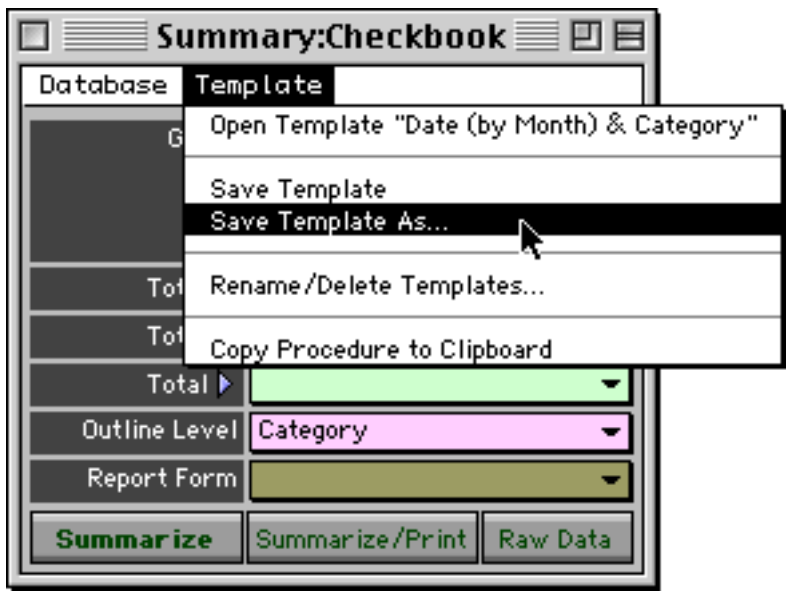


When you're done with the summaries press the **Raw Data** button to remove the summary records and get the original raw data back (see "Getting Rid of Summary Records" on page 421).

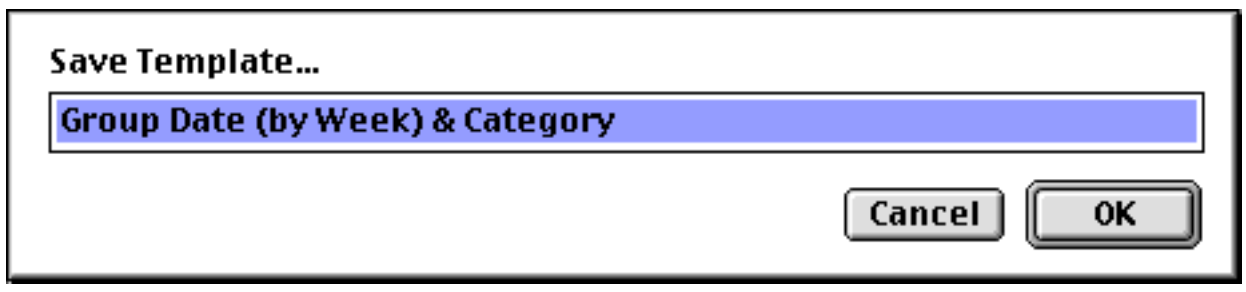


Using Summary/Outline Templates

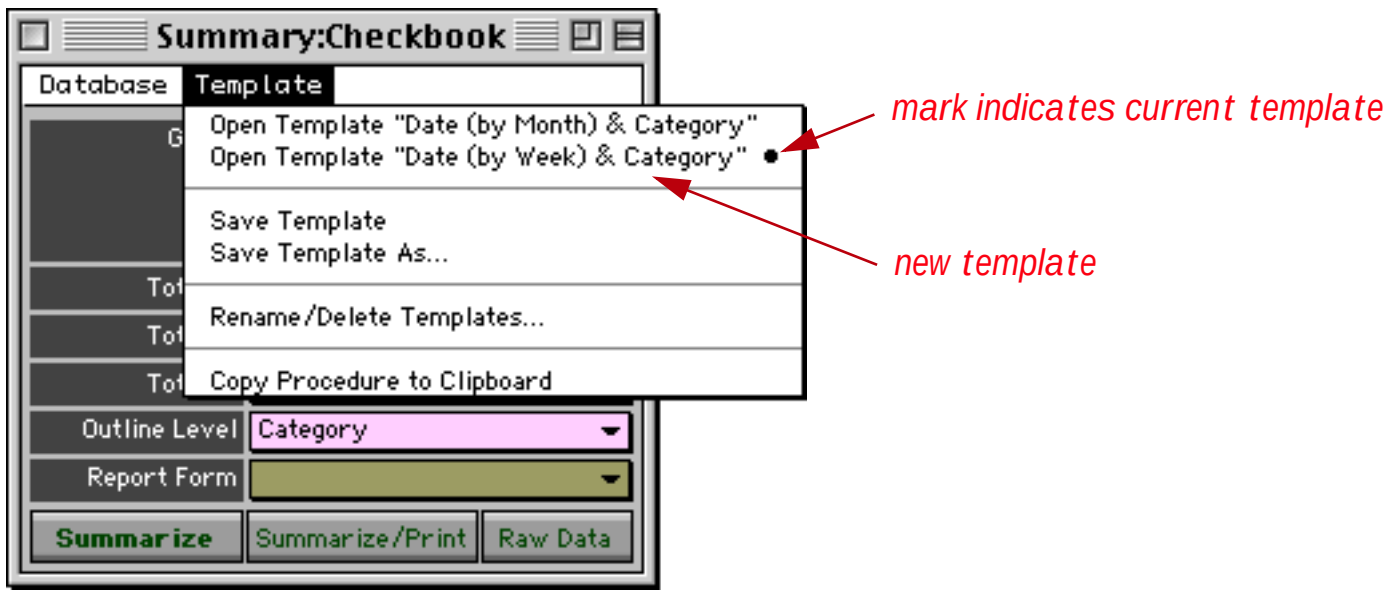
Once you've set up a configuration for summarizing a database you can save that configuration as a template so that it can be re-used at any time. To save a new template use the **Save As** command in the Template menu.



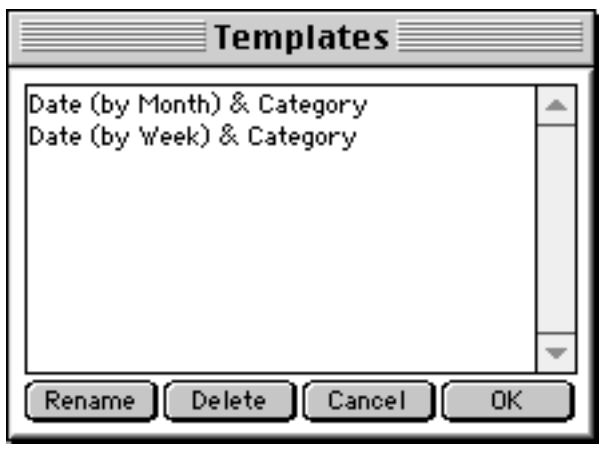
The wizard will prompt you to type in a name for your new template (and will make a suggestion based on the fields being grouped).



Type in the name and press **OK**. The new template is added to the **Template** menu and may be selected any time this database is open.



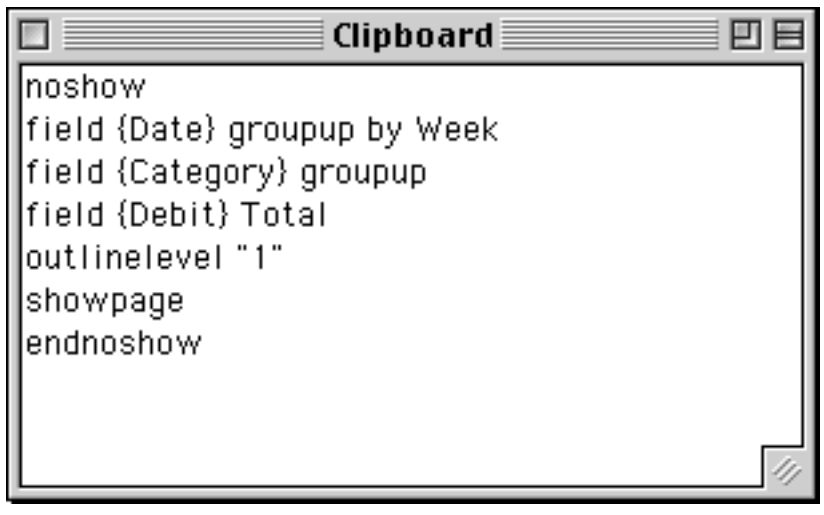
If you want to rename or delete a template choose **Rename/Delete Templates...** from the Template menu. This opens a dialog listing all the templates in the current database.



To rename a template click on it and then click the **Rename** button. The wizard will prompt you to type in the new name for the template. To delete a template click on the name and then press the **Delete** button. In either case the changes won't take effect until you press the **OK** button, giving you one last chance to back out with the **Cancel** button.

Converting a Template into a Procedure

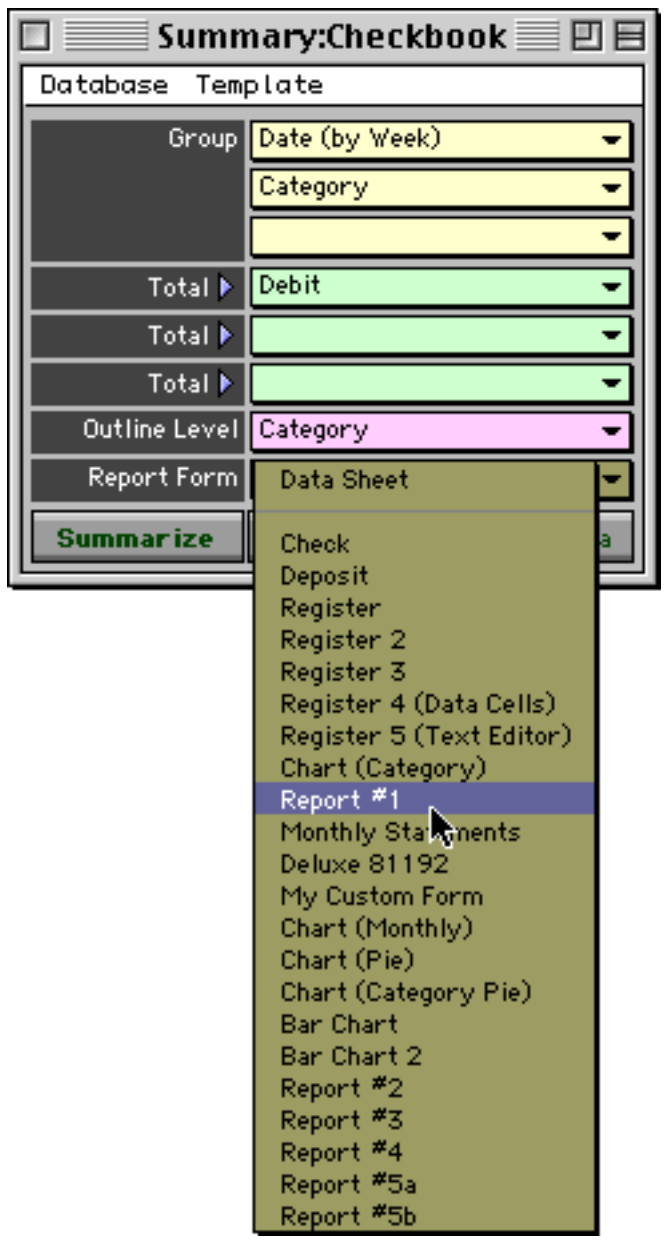
If you wish you can convert a template into a procedure (see "Procedures" on page 1339) using the **Copy Procedure to Clipboard** command in the Template menu. This allows you to calculate this summary without needing to open the **Summary & Outline Wizard**. This command will take the current settings and writes a procedure for you, placing the text of the procedure on the clipboard.



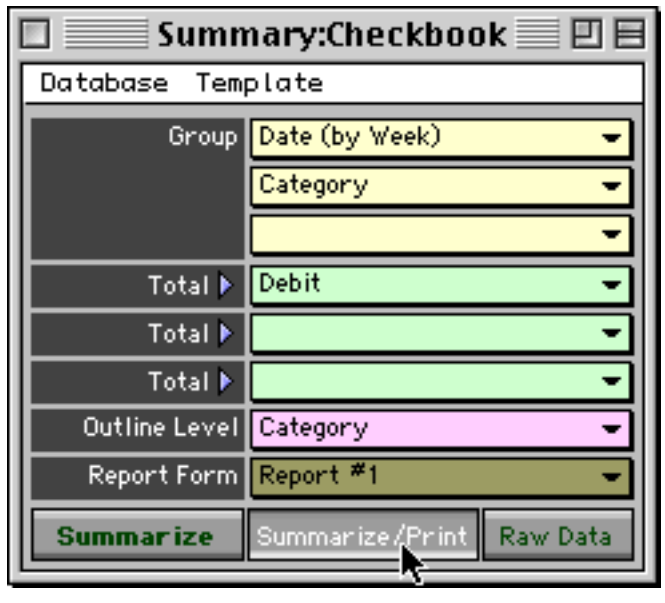
See "Writing a Procedure from Scratch" on page 1352 to learn how to create a new procedure within the database. Once the procedure is created you can simply use the **Paste** command to copy the text created by the **Copy Procedure to Clipboard** command into the new procedure.

Printing the Summary Results

The **Summary & Outline Wizard** can automatically open a form and print the results. The form can be set up with report tiles to allow you to customize the way the report is printed (see "Printing Summary Information" on page 1129). To set this up choose the form using the pop-up menu.



Once the form is selected you can press the **Summarize/Print** button.

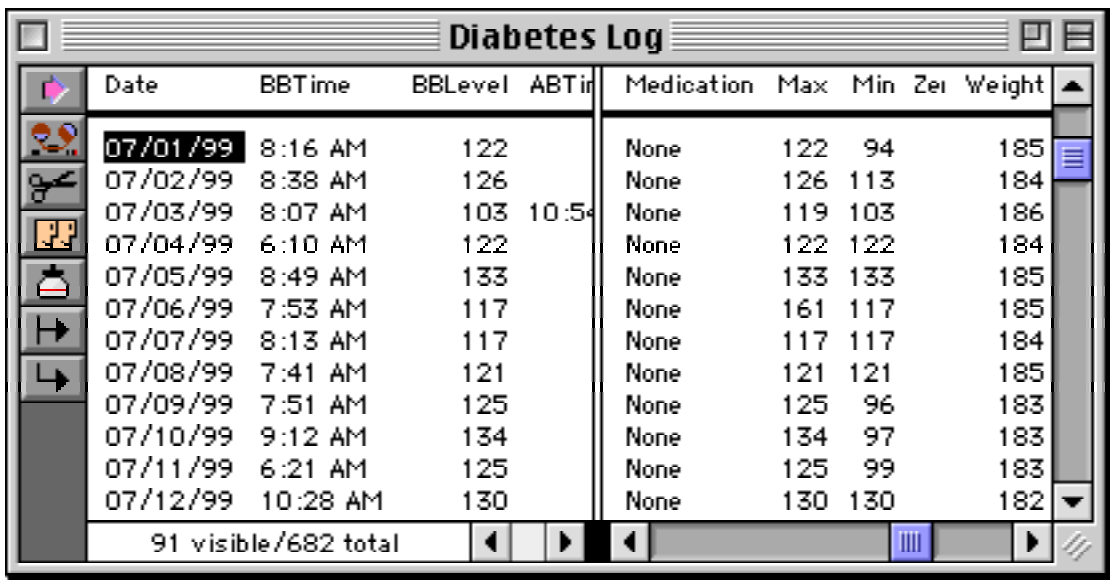


The wizard will summarize the database, open the form, print the database (stopping to let you set print options) and then close the form.

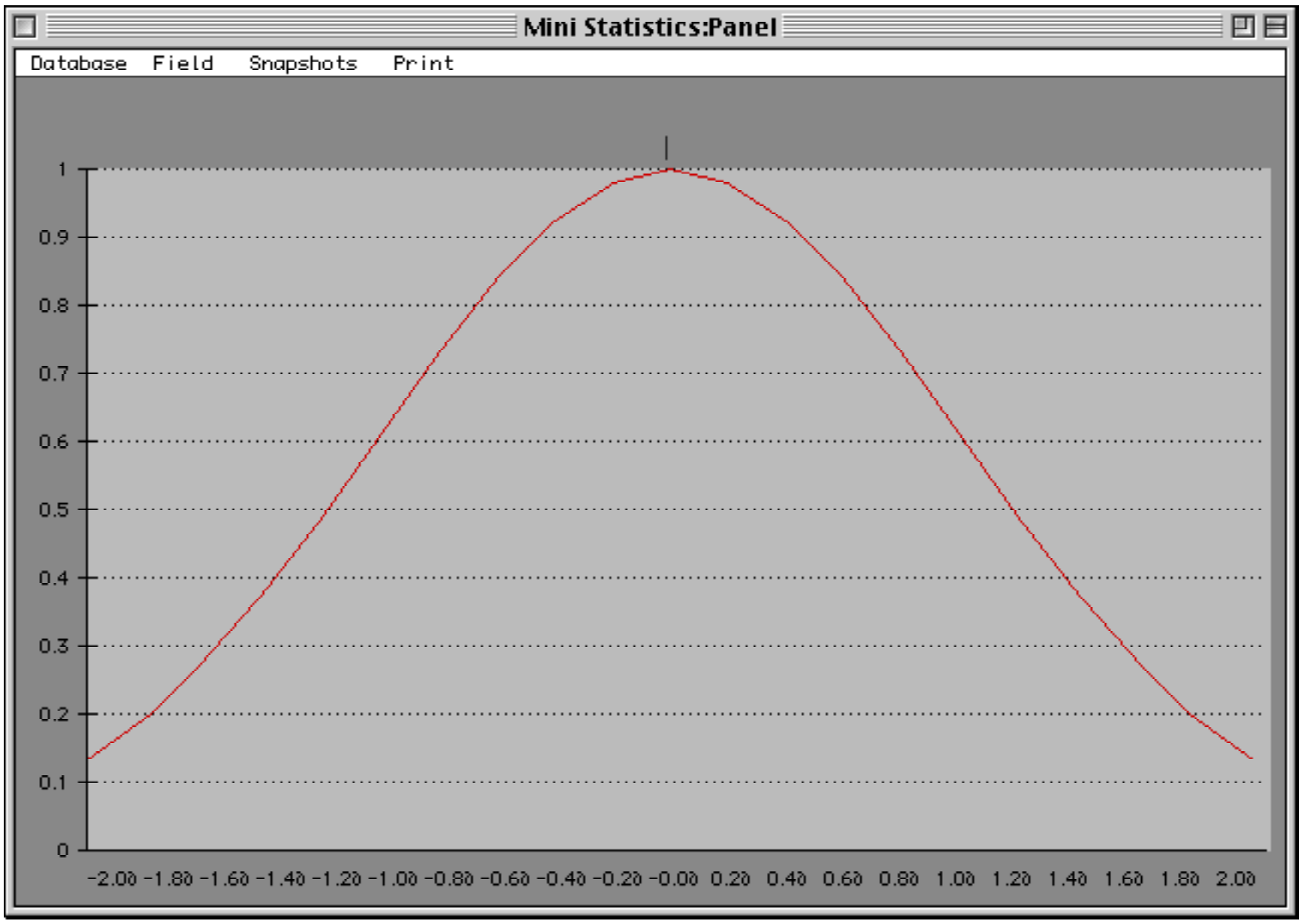
The Mini Statistics Wizard

In addition to the summary & outline tools described previously in this chapter Panorama also has a special wizard to help with statistical analysis. This wizard can calculate the mean (average), median, and standard deviation of a data set. In addition the wizard can plot a normalized chart showing how the data is distributed around the mean. You can easily see how this distribution compares with the standard gaussian distribution (the famous bell shaped curve).

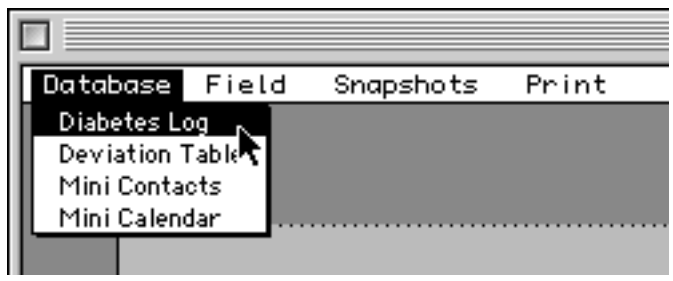
To use this wizard you'll need a database that contains one or more numeric fields. To illustrate we'll use a database that contains medical data.



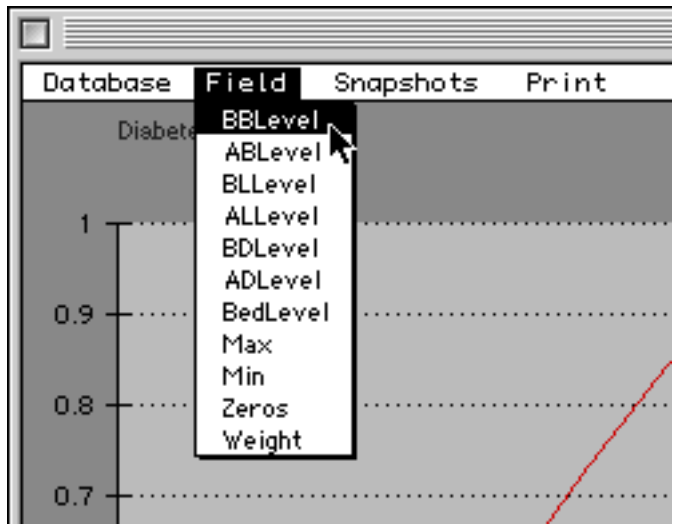
To analyze the data, open the **Mini Statistics** wizard.



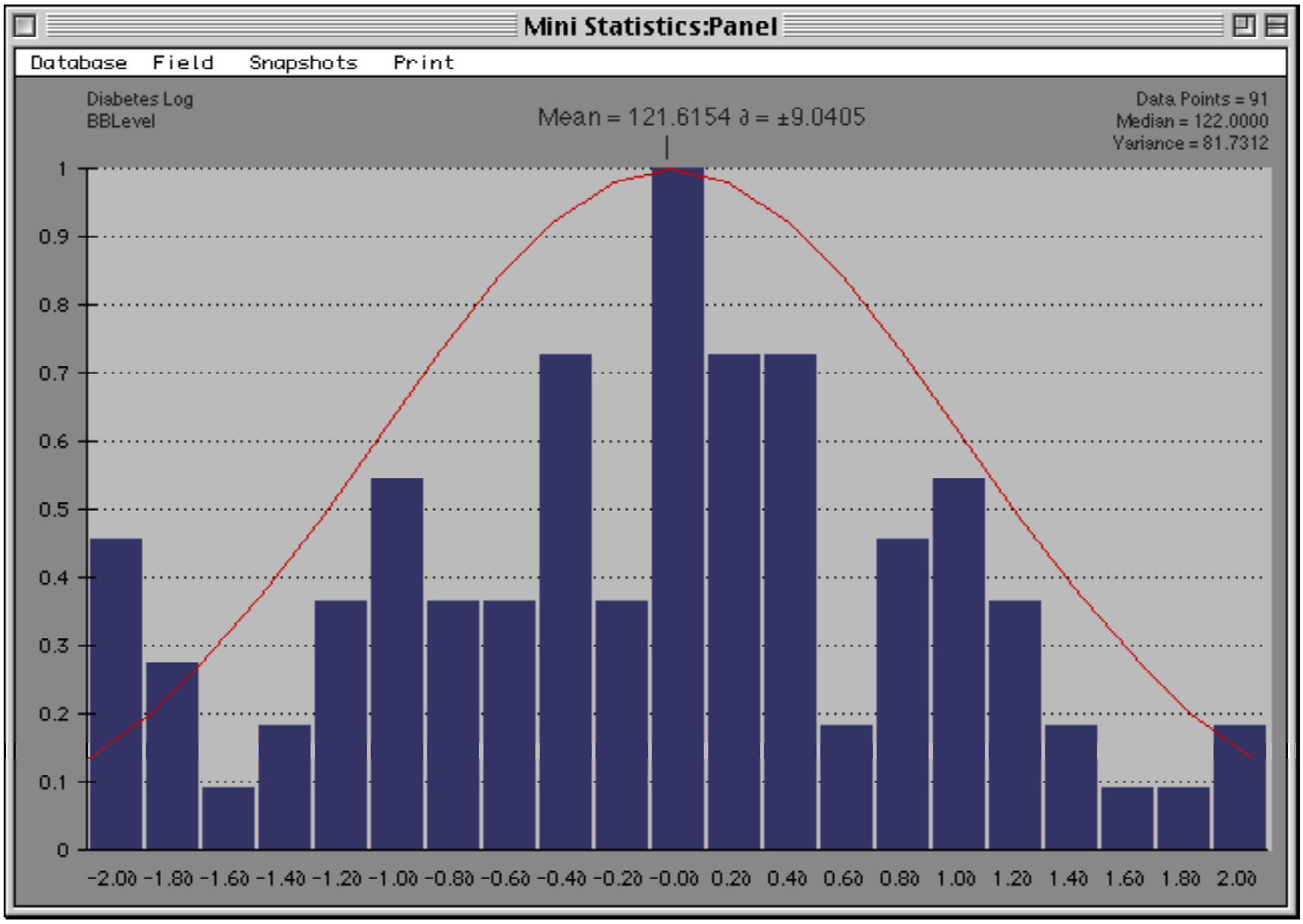
To set up the statistical analysis, start by selecting the database you want to use, in this case Diabetes Log.



Now choose the field that contains the data you want to analyze, in this case BBLevel. (Note: The **Field** menu only lists numeric fields — text, date and picture fields are not listed.)



After a couple of seconds the analysis will appear.

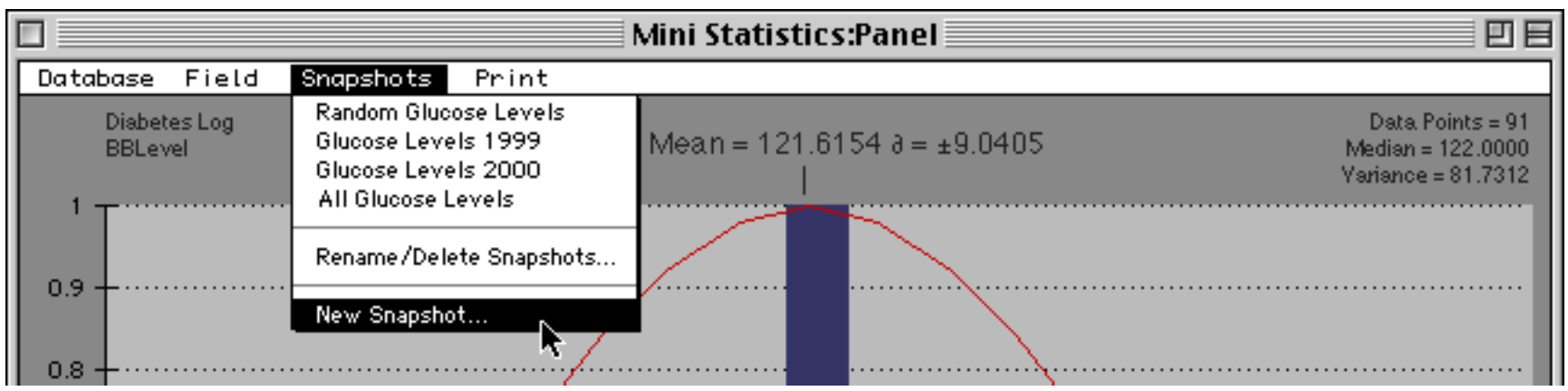


For these data points the mean (average) value is 121.6154. The standard deviation (∂) is 9.0405. The data set contains 91 values, the median is 122, and the variance is 81.7312. The blue bars show how the data is clustered around the mean, while the red line shows the normal gaussian “bell shaped curve” distribution.

If the original data changes you may want to update the analysis. You can either select the field again from the **Fields** menu, or simply click on the chart.

Saving a Statistical Snapshot

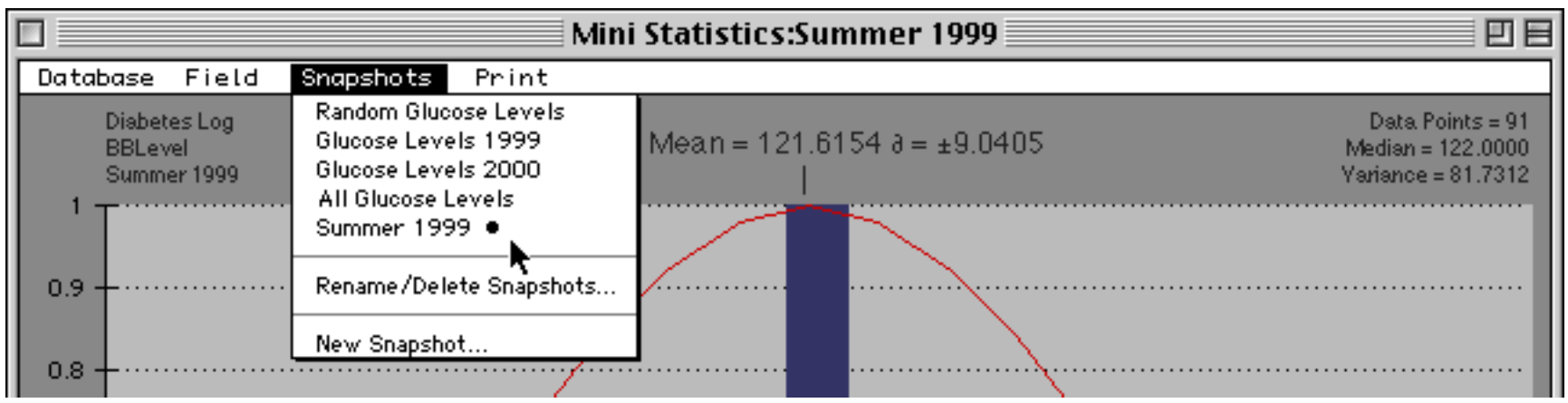
Once you've performed a statistical analysis on a set of data you can save a snapshot of that analysis for later review. To save a snapshot of the current analysis choose **New Snapshot** from the Snapshot menu.



Enter a name for the new snapshot. (If you enter the name of an existing snapshot it will overwrite the old snapshot.



When you press the **OK** button the new snapshot is listed in the **Snapshot** menu.

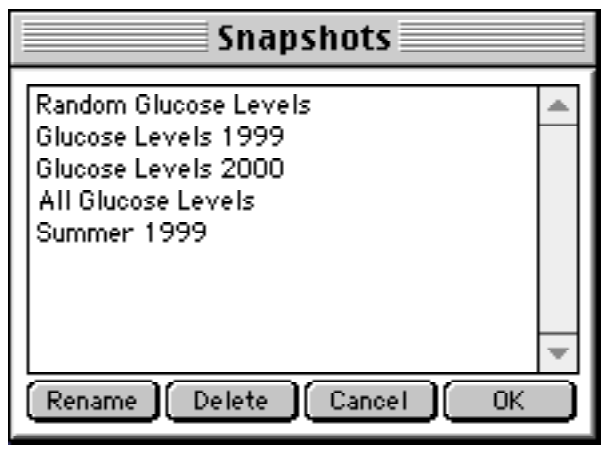


To see a snapshot you have recorded before simply select it from the menu.

Note: Snapshots are actually stored in a permanent variable in the database that contains the original data (in this case Diabetes Log). This means that you will only see the snapshots for the currently selected database. It also means that the snapshots are not permanently saved to disk until you save the original database to disk.

Renaming and Deleting Snapshots

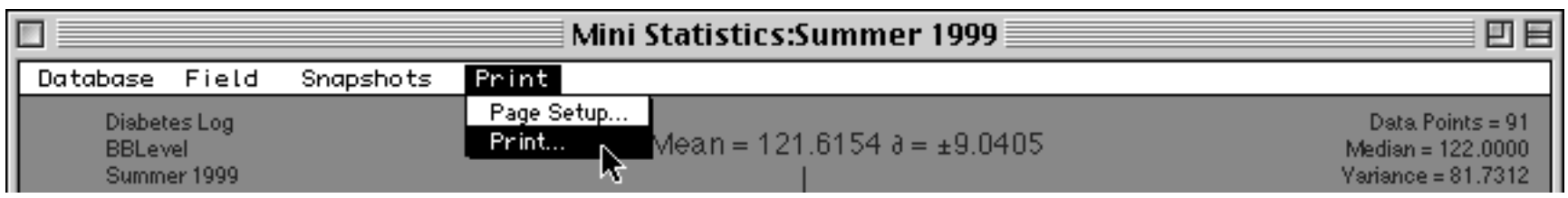
The Rename / Delete Snapshots command opens a dialog.



To rename a snapshot, click on it, press the **Rename** button, then type in the new name. To delete a snapshot, click on it and then press the **Delete** button.

Printing a Statistical Analysis

To print a statistical analysis use the **Print** menu inside the window.



Chapter 11: Crosstabs



This chapter describes Panorama's most powerful tool for analyzing and summarizing data—**crosstabs**. A crosstab is simply a table with categories across the top and down the left, with numbers in the middle and totals across the bottom and down the right.

Checkbook:XTABS:Spending by Month

xtab	Jan 99	Feb 99	Mar 99	Apr 99	May 99	Jun 99	TOTAL
Advertising	2,841.02	8,134.13	7,501.90	1,024.52	7,541.18	828.00	34,516.82
Auto		240.21	33.32		119.05		555.04
Equipment Rental	96.05	73.14		79.69	105.44		632.01
Fixed Assets		428.39			778.00	1,168.75	9,774.47
Insurance	761.63	601.48	1,539.14	522.63	1,220.45	461.99	8,234.53
Legal Fees	223.52	95.00			799.55	15.00	1,288.07
Maintenance		310.00	872.25	132.00	1,012.63	368.38	3,673.99
Office Supplies	426.93	915.50	1,579.74	339.46	1,265.34	281.52	7,261.09
Postage	150.00	315.00	110.00	156.35	115.00		1,456.35
Printing					96.68		188.96
Purchases	17,083.42	3,386.78	19,557.48	2,268.19	4,796.82	5,172.74	66,217.17
Rent	4,070.83	7,742.19	3,566.30	265.00	7,630.56	3,874.00	35,026.34
Shipping	231.72	192.00	830.69	25.00	220.00		1,928.20
Taxes	549.00		2,008.98	734.33	513.51	155.76	5,152.79
Telephone	141.09	736.66	1,027.01	305.56	915.96	202.31	5,690.50
Utilities		402.78	384.49		529.76	213.58	2,054.89
TOTAL	**26,575.21**	**23,573.26**	**39,011.30**	**5,852.73**	**27,659.93**	**12,742.03**	**183,651.2**

17 visible/17 total

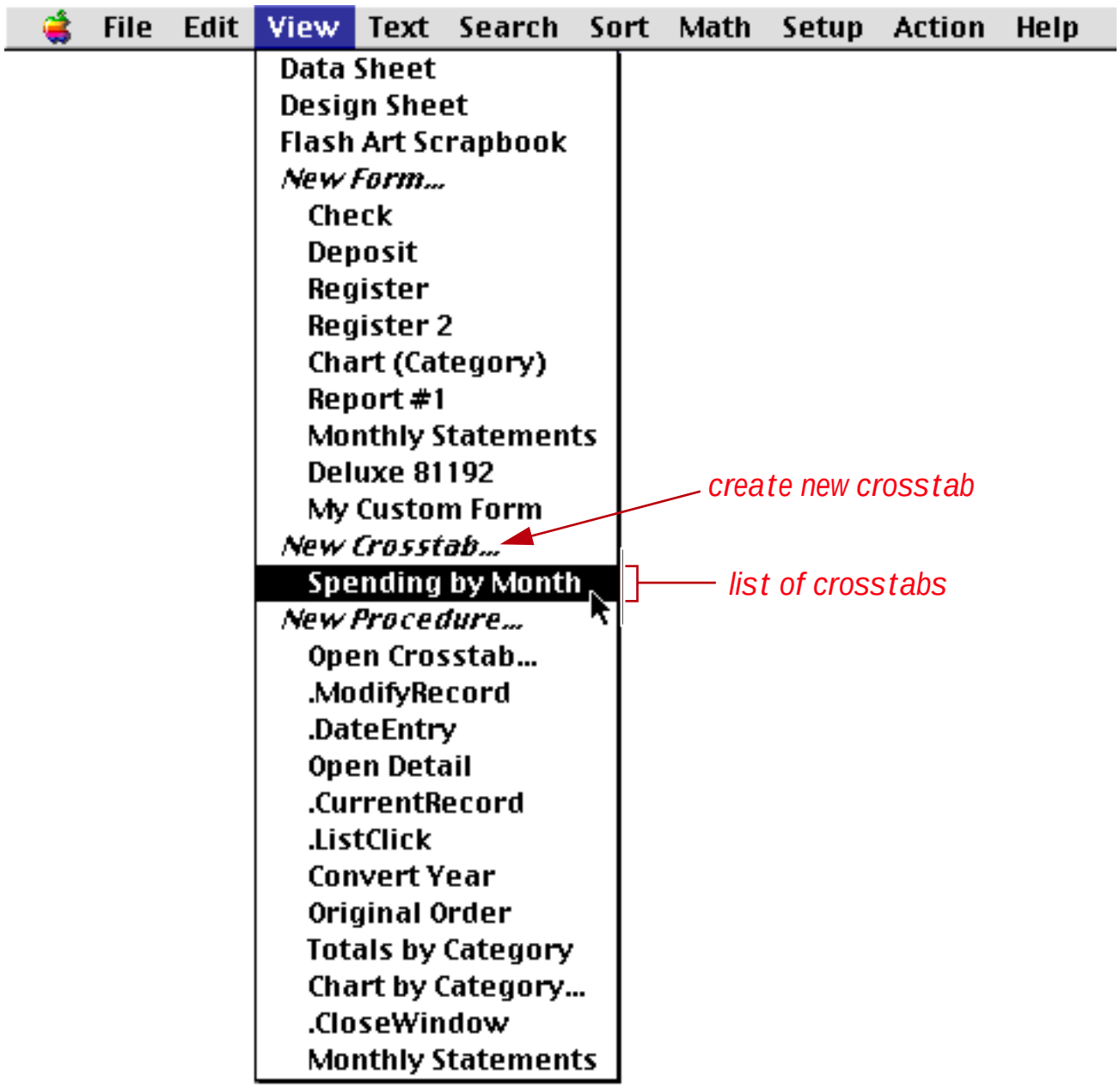
The word crosstab is short for cross tabulation, referring to the criss-cross way that totals are tabulated both across and down. Probably the most common example of a crosstab is a budget, with months or years across the top, and spending categories down the left.

Before Panorama became available, crosstabs were usually created using a spreadsheet. Spreadsheets are perfect for totalling the rows and columns in the crosstab table. Unfortunately, spreadsheets cannot help with the really tedious part of creating a crosstab table—taking the raw data, categorizing it, and converting it into the crosstab table format. With a spreadsheet, this tedious number crunching must be done by hand.

Panorama automates the entire crosstab process from start to finish. Starting with raw data (a checkbook database, for example), Panorama divides the data into categories and automatically creates and calculates the entire crosstab table. When the raw data changes, the entire process can be repeated with a single mouse click. A simple dialog sets up the whole process.

Panorama can also work a crosstab backwards, allowing you to locate the raw data associated with any crosstab value. For example, if the crosstab table shows that July's advertising expenditures seem a bit high, simply click on that value and press the **Select Original Data** tool. The individual data records for July advertising will appear.

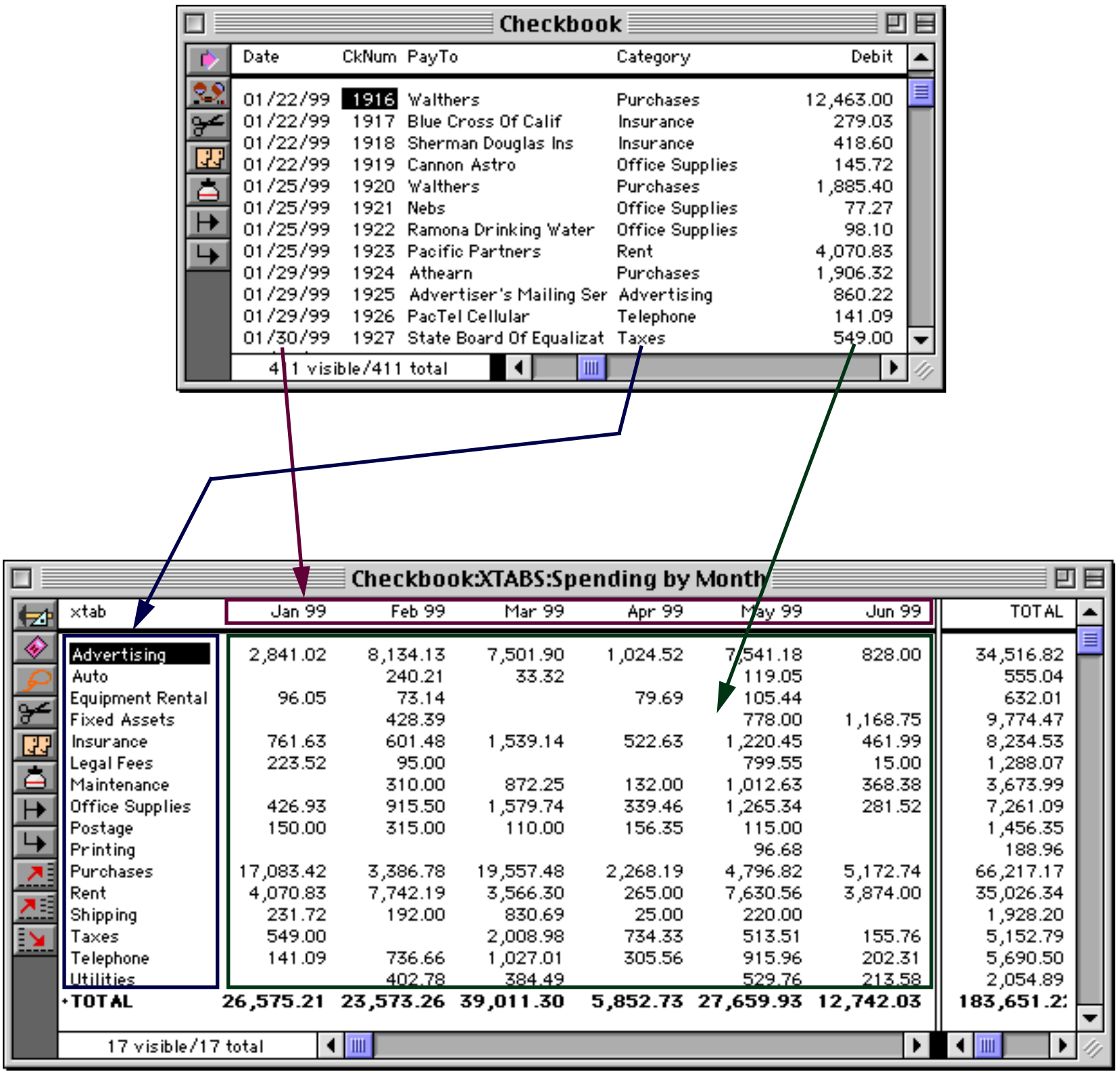
Panorama does not limit you to one crosstab table per database. Each crosstab table appears in its own window, and you can have as many different crosstabs as you need. Crosstab tables are created and opened with the **View** menu.



Although each crosstab table gets its raw data from the main database, it is otherwise independent. Setting up and calculating a crosstab table does not change the main database in any way.

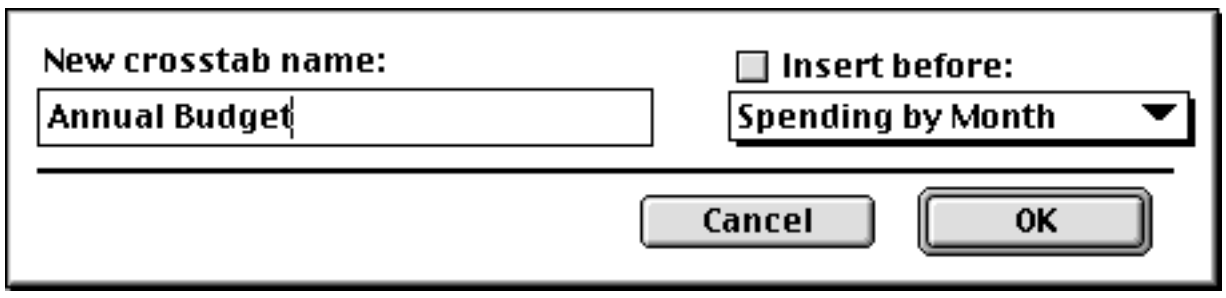
Category and Tabulation Fields

A crosstab is based on three fields in the main database. Two of these fields are called **category fields**, and the third is called the **tabulation field**. The two category fields are the fields that criss-cross across the top and left sides of the crosstab table. The tabulation field holds the raw data that is counted or totalled in the center of the crosstab table. In the example crosstab shown below, Date and Category are the category fields while Debit is the tabulation field.

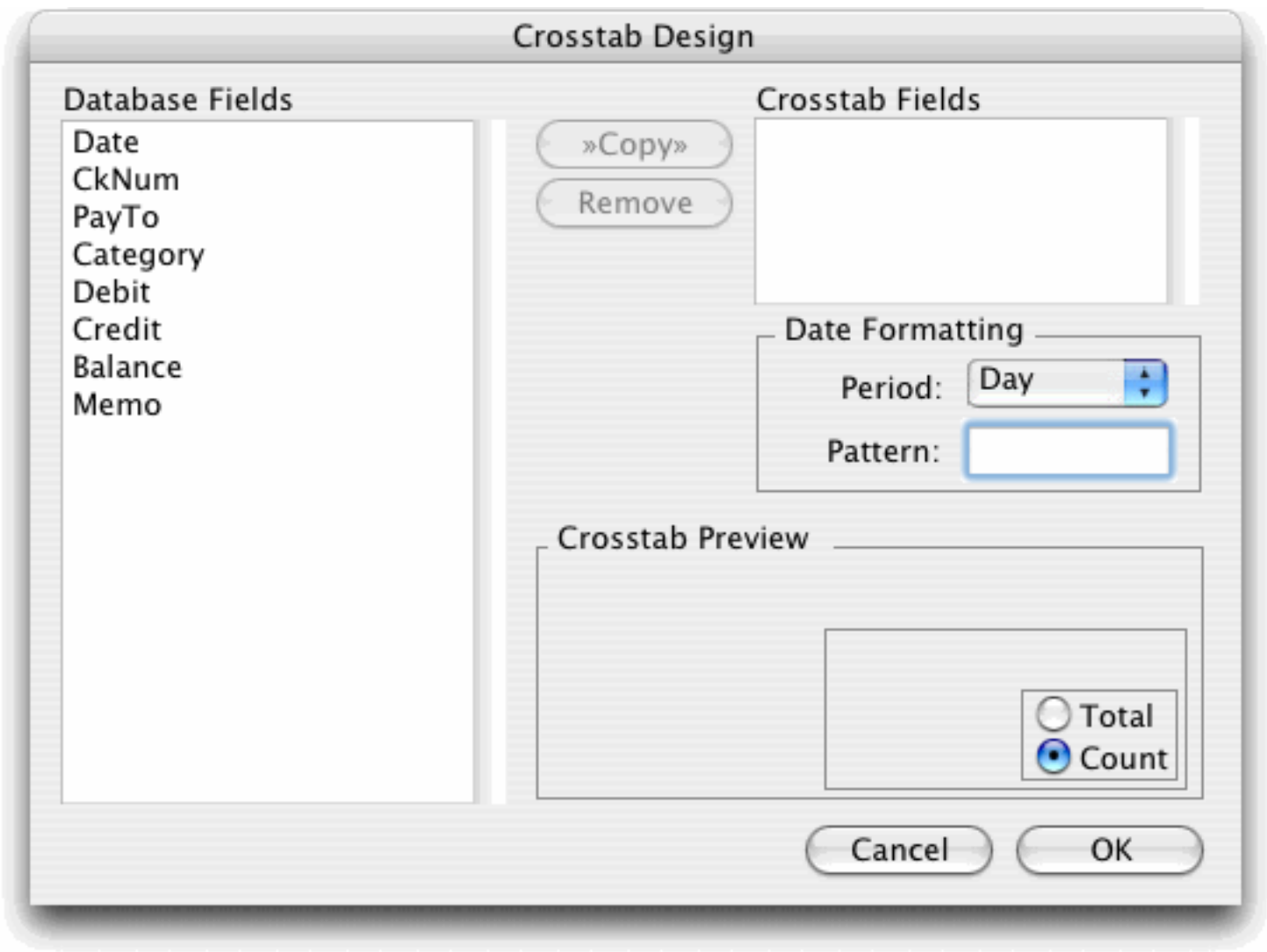


Creating and Setting Up a New Crosstab View

New crosstab views are created using the View menu. Choose **New Crosstab** from the menu. Then you must give the new crosstab view a name (up to 25 characters) and press **Ok** to create the new view.

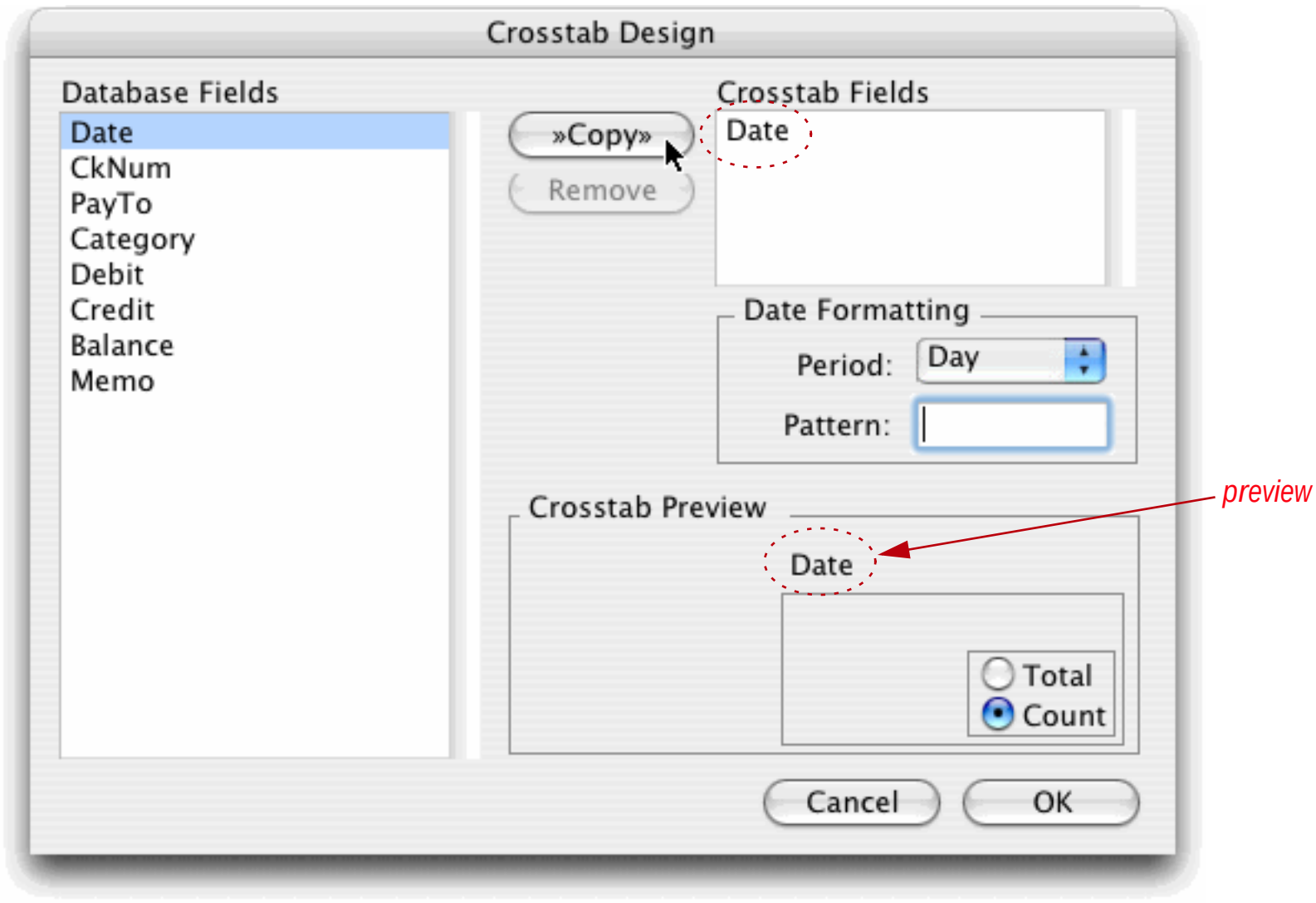


When a new crosstab view is created, the **Crosstab Design** dialog box automatically appears. This dialog allows you to specify the category and tabulation fields (see previous section) and to specify what type of calculation (total or count) to use.



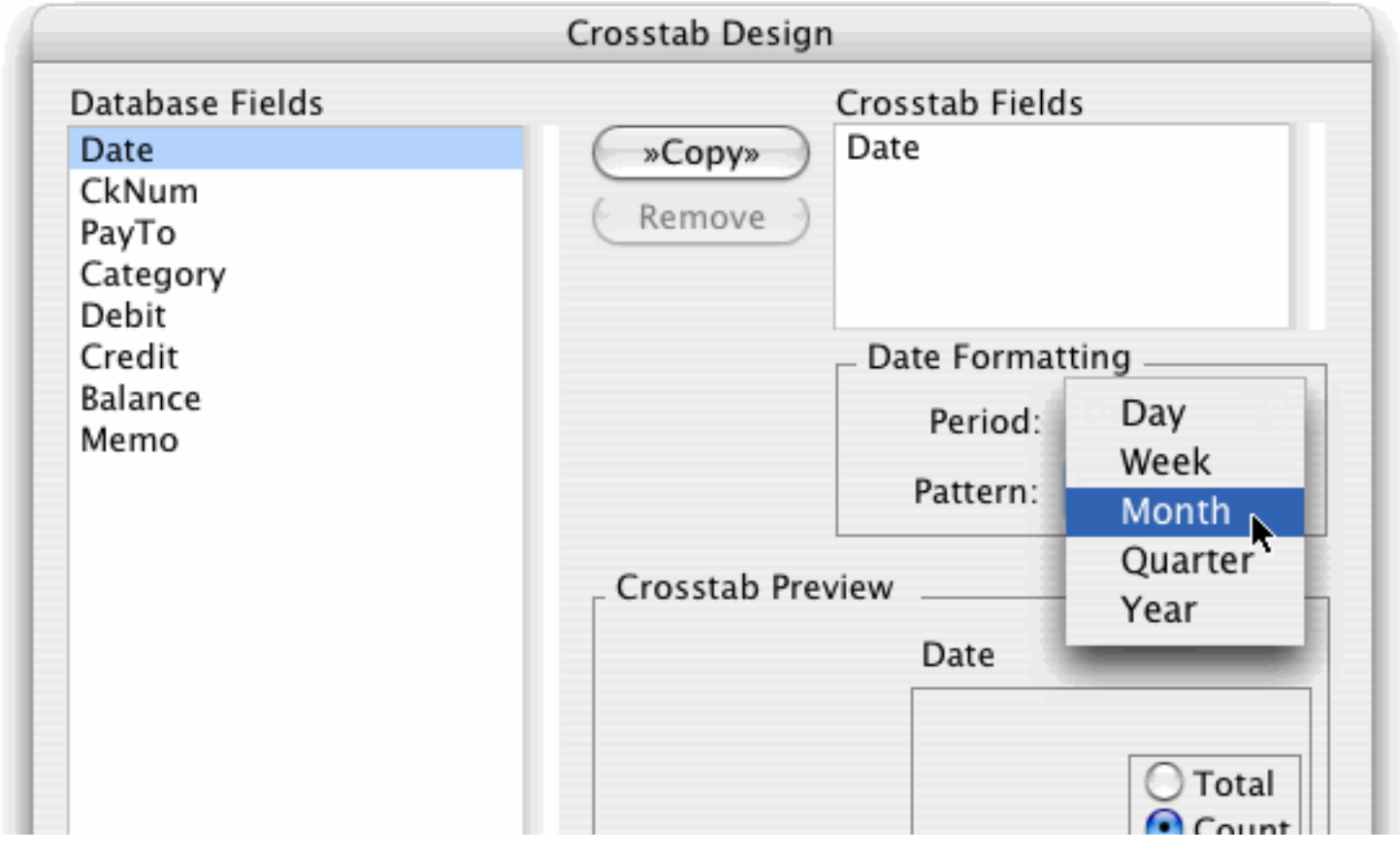
To use the Crosstab Design dialog you copy fields from the list on the left over to the list on the right. The left side lists all the fields in the database. The right side lists the fields in the crosstab. The first two fields copied to the right become the two category fields. The third field becomes the tabulation field. As you build the crosstab, a miniature schematic diagram of the crosstab appears in the lower right hand corner of the dialog.

Let's walk through the creation of a crosstab like the one shown at the beginning of this chapter. First, click on the Date field and copy it to the right. (Hint: To copy the field either press the **»Copy»** button or double click on the field name.

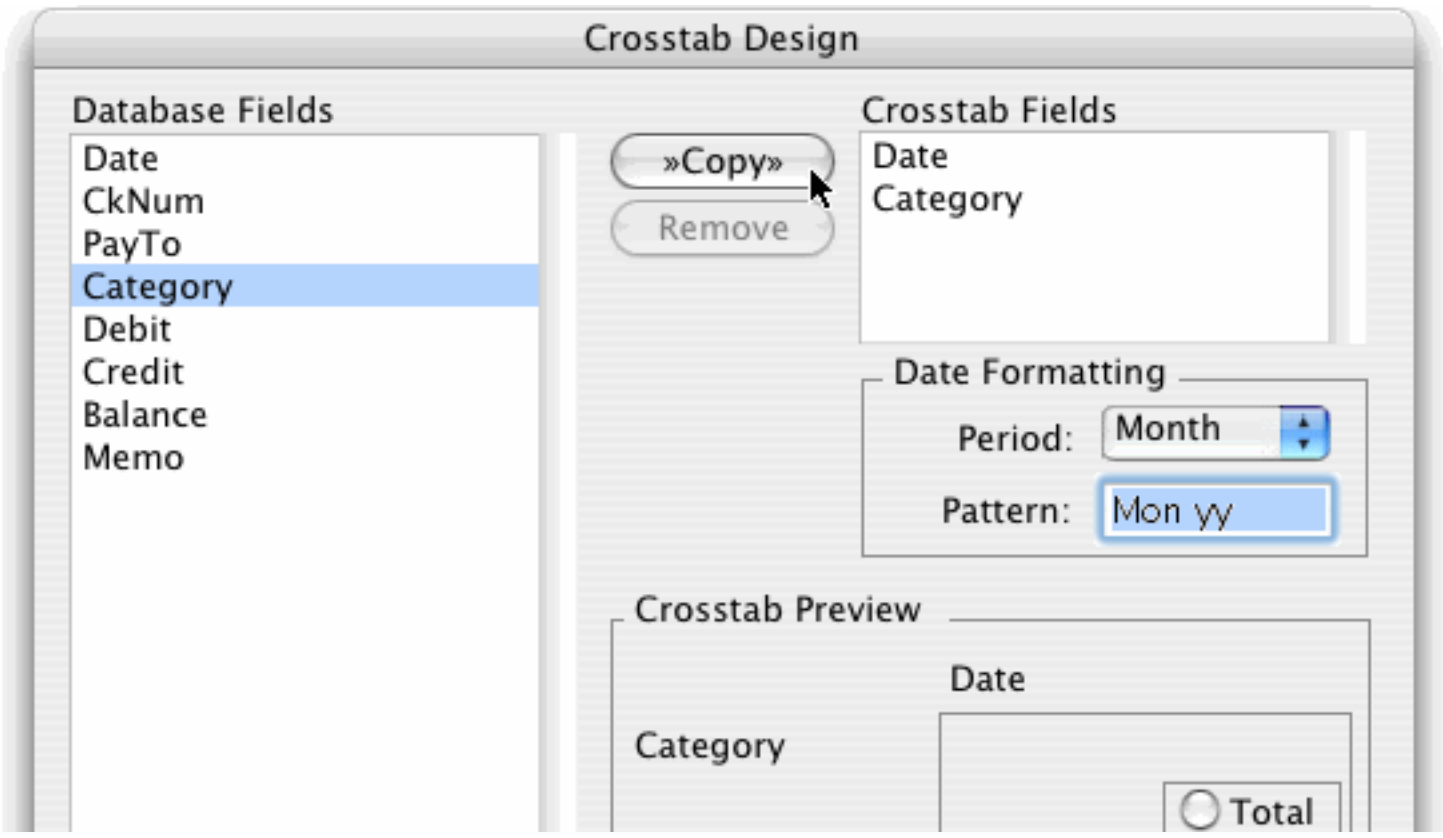


As you can see, the Date field now appears in the list of Crosstab fields. It also appears in the preview of the final crosstab, showing that the Date will appear across the top of the crosstab.

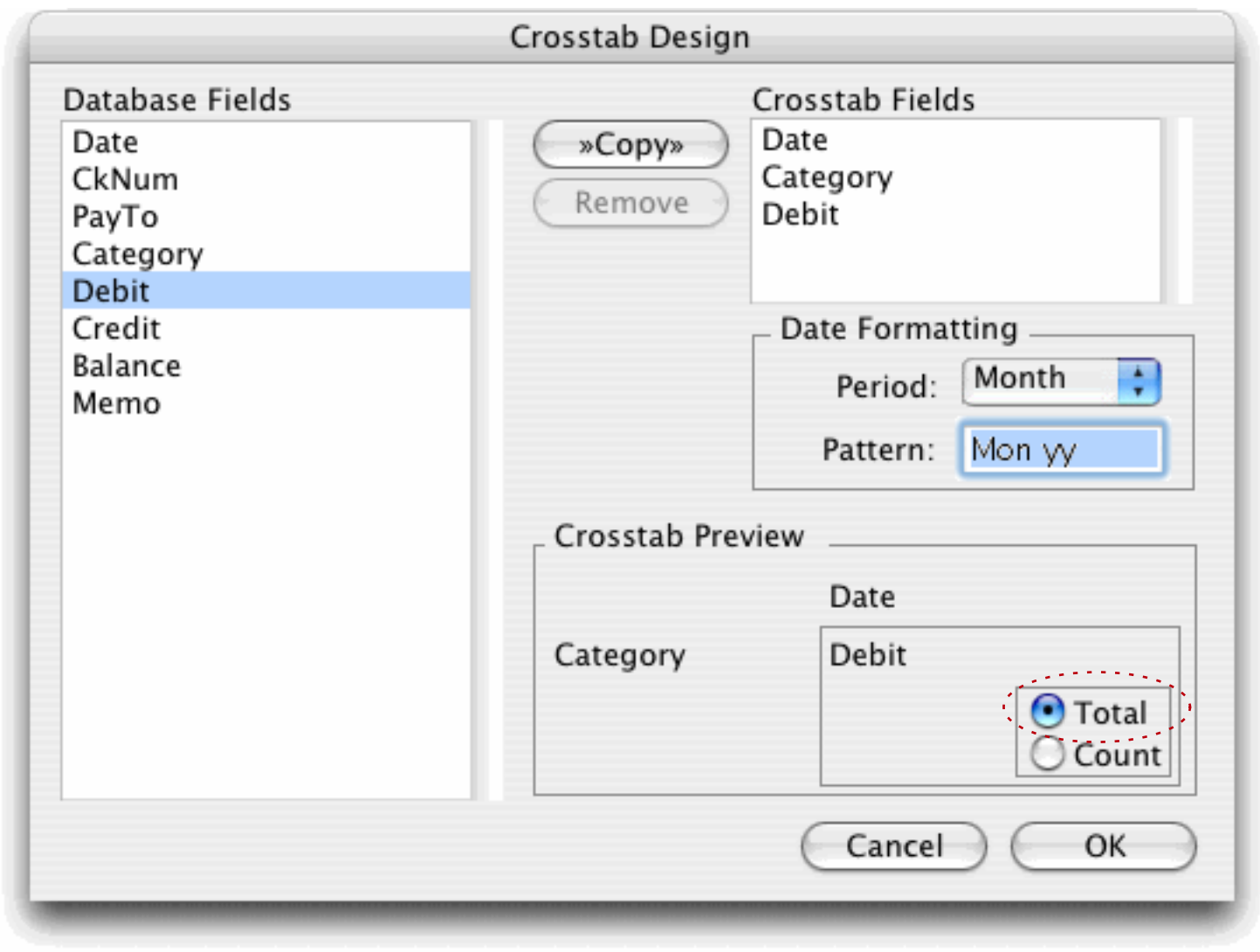
Since we want the date to be grouped by month, select Month from the Date Period pop-up menu. (If you forget to do this now you can always go back and change it later.)



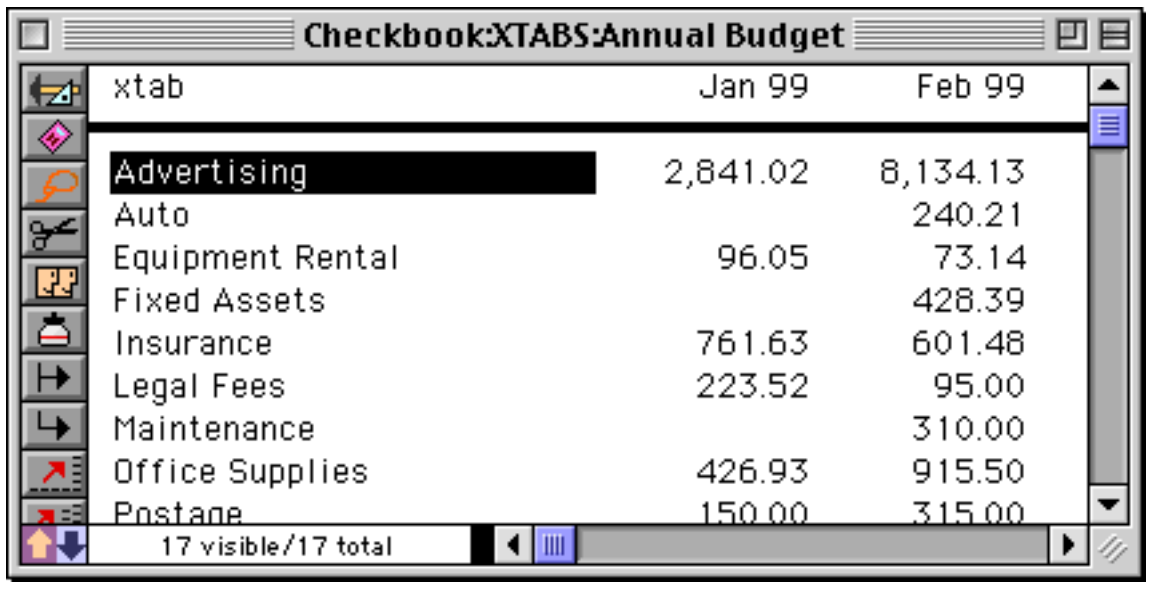
Next, copy the Category field into the crosstab. It will appear in the list and on the left side of the mini-diagram, showing that the category will appear on the left side of the crosstab.



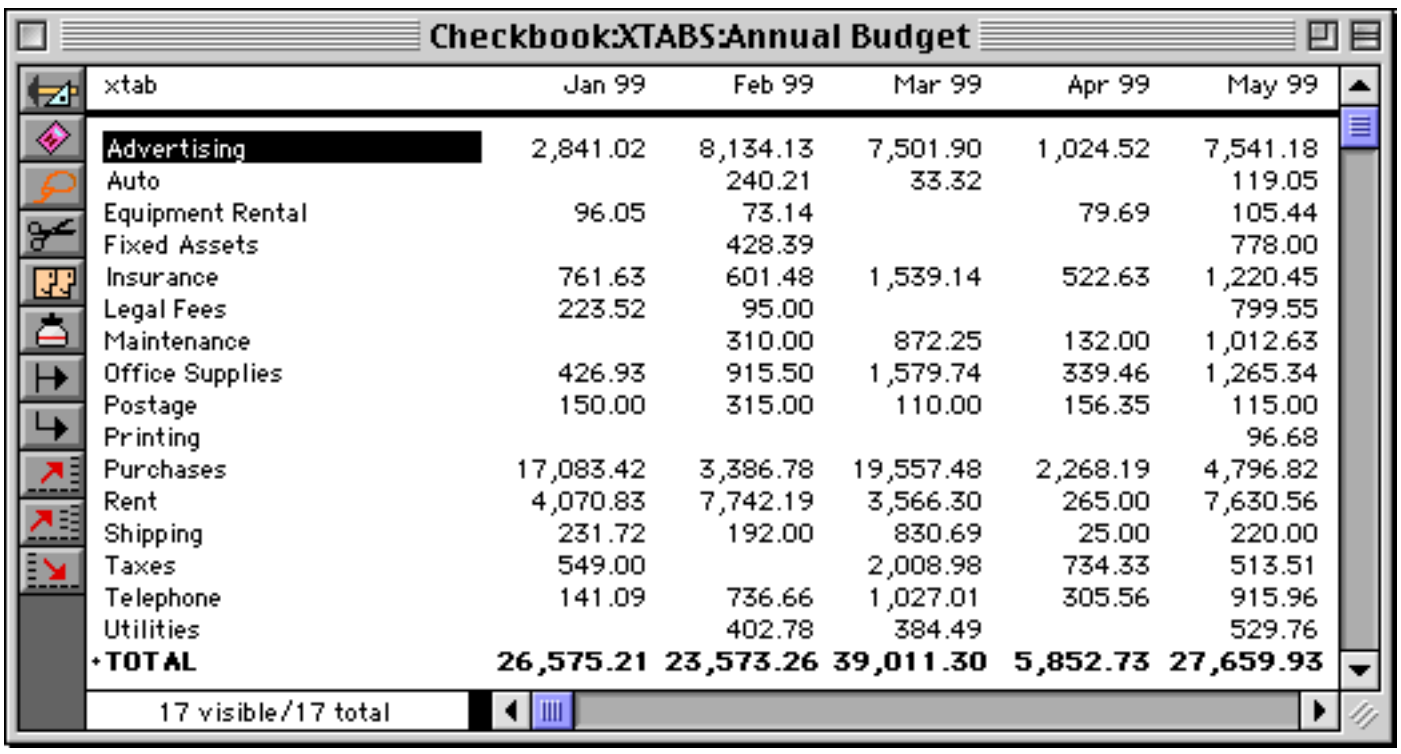
Now copy the Debit field into the crosstab. You'll also need to click on the Total radio button, since we want to calculate sums of the checks, not counts of the checks.



Once you've specified the category and tabulation fields, press **Ok** to actually calculate the crosstab. Depending on the complexity of the database, there may be a delay of seconds or even minutes as the crosstab is calculated. When the calculations are finished, the new crosstab table will appear. It will look something like this.

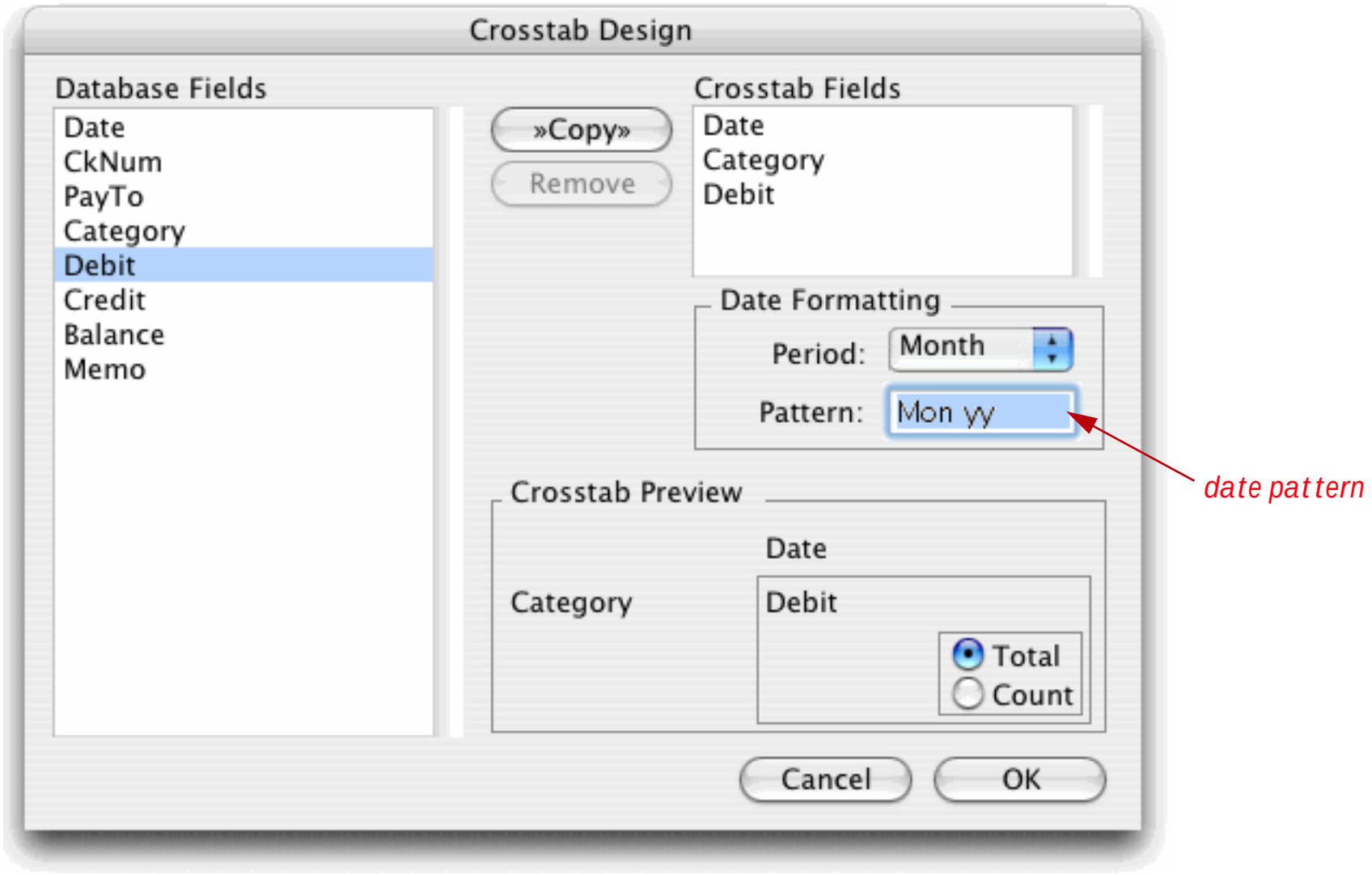


You can use now adjust the window size, font, and column widths as you like.



Crosstabs by Day, Month, Quarter or Year

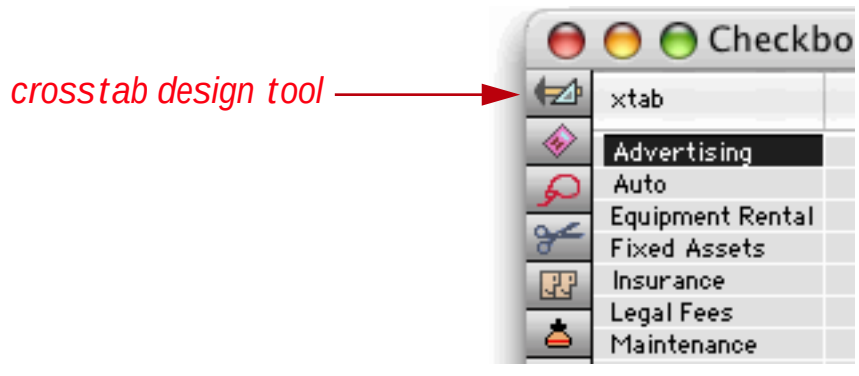
If one of the category fields contains dates, you must tell Panorama what period to group by. The Date Period pop-up menu has five choices: day, week, month, quarter, and year. You can also specify a pattern for displaying the date in the crosstab. For example, months can be displayed as 1-04, Jan-04, or January 2004.



Panorama supplies a default pattern when you choose from the Date Period pop-up menu. You can use this predefined pattern, or you can type in any pattern you want. See “Date Output Patterns” on page 295 for more information about date patterns.

Changing the Crosstab Design

The crosstab design can be changed at any time by pressing the **Crosstab Design** tool. This brings up the same dialog box you used to originally set up the crosstab.

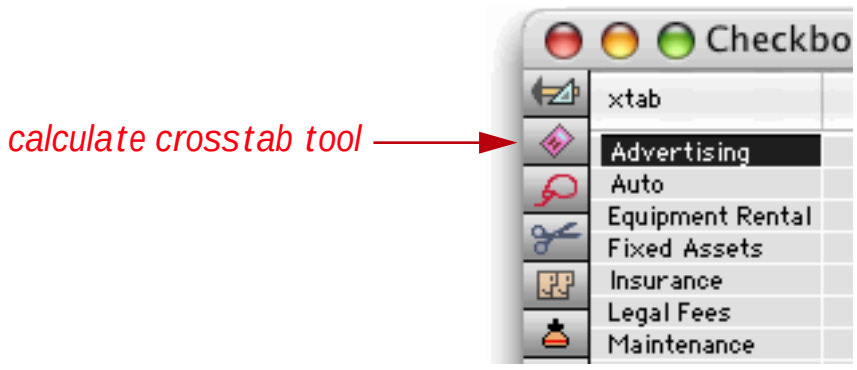


To erase the entire crosstab design and start over, select all the fields in the CrossTab Fields list and press the **Remove** button. (You can select all the fields by dragging the mouse over the list.)

If you want to change just one field, select both the old field from the CrossTab Fields list and the new field from the Database Fields list. Then press the **»Copy»** button to change the field.

Re-Calculating a Crosstab

Crosstabs do not automatically update when the main database changes. This is because of the time it takes to recalculate the crosstab. If you change the main database and want to re-calculate a crosstab, press the **Calculate Crosstab** tool.



Adjusting Crosstab Column Widths

When a new crosstab is created, Panorama tries to assign an appropriate width for each column. You can adjust these column widths the same way you would adjust the column widths in the data sheet. Move the cursor over the column titles and drag left or right to adjust the width.

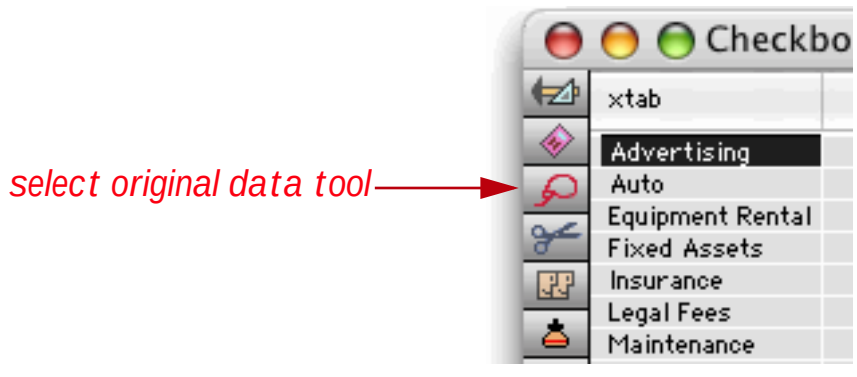
Whenever the crosstab is recalculated, Panorama automatically resets the width of every column except for the first two. The third, fourth, fifth and all additional columns are all set to the same width as the second column.

Crosstab Font and Size

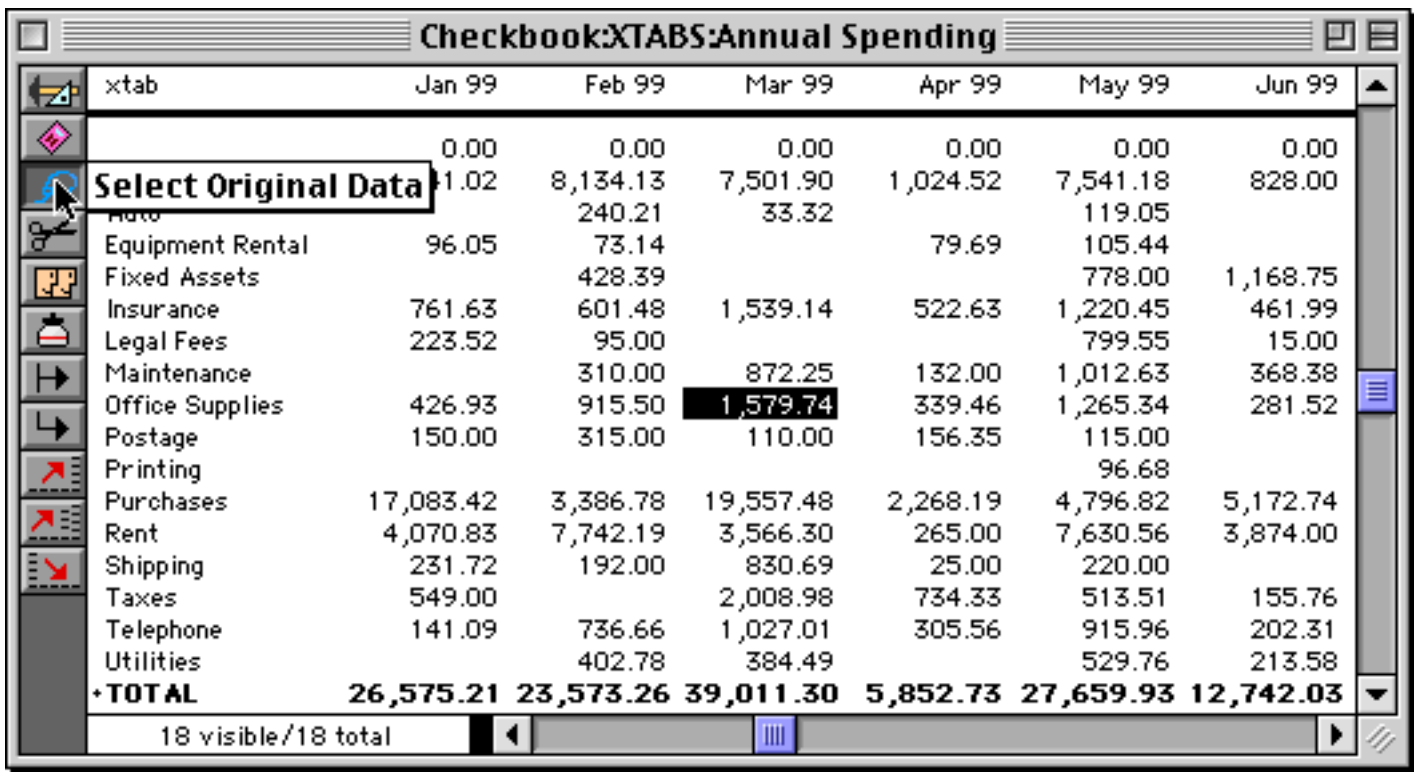
You can change the font and size of the crosstab with the Text Menu.

Selecting Original Data

Using the **Select Original Data** tool, Panorama can locate the raw data behind any value in the crosstab table. To do this you must have a regular database window open in addition to the crosstab window, usually the data sheet window.

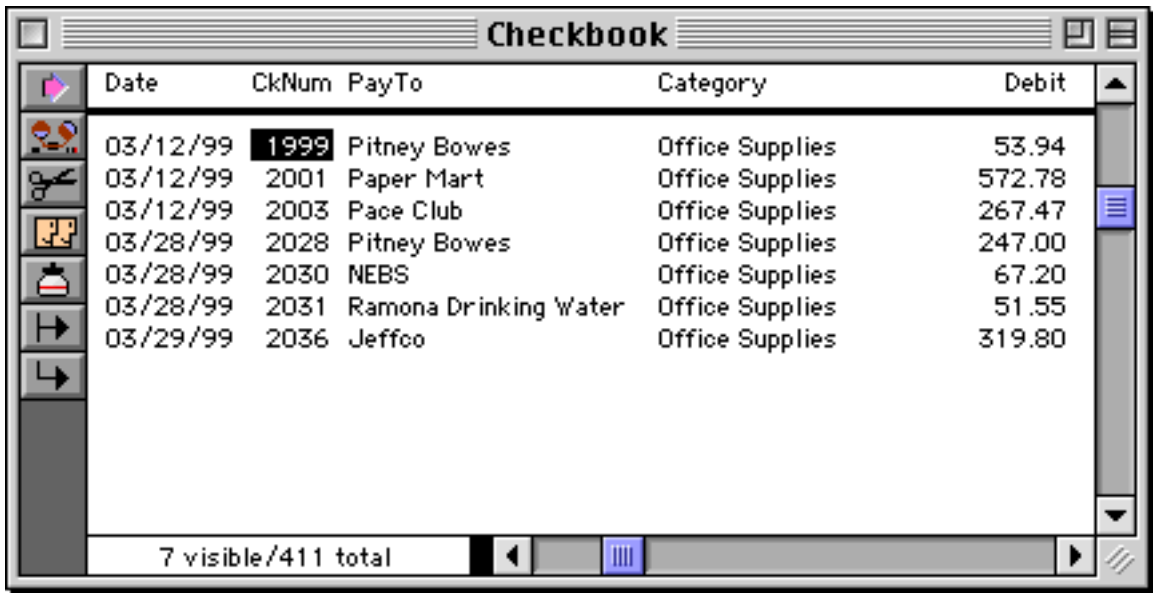


To select the original data, first click on the crosstab value you are interested in. Then click on the **Select Original Data** tool, The original data is selected and will appear in the data sheet or other database window. For example, you could click on the Mar 99 Office Supplies cell, then choose **Select Original Data**.



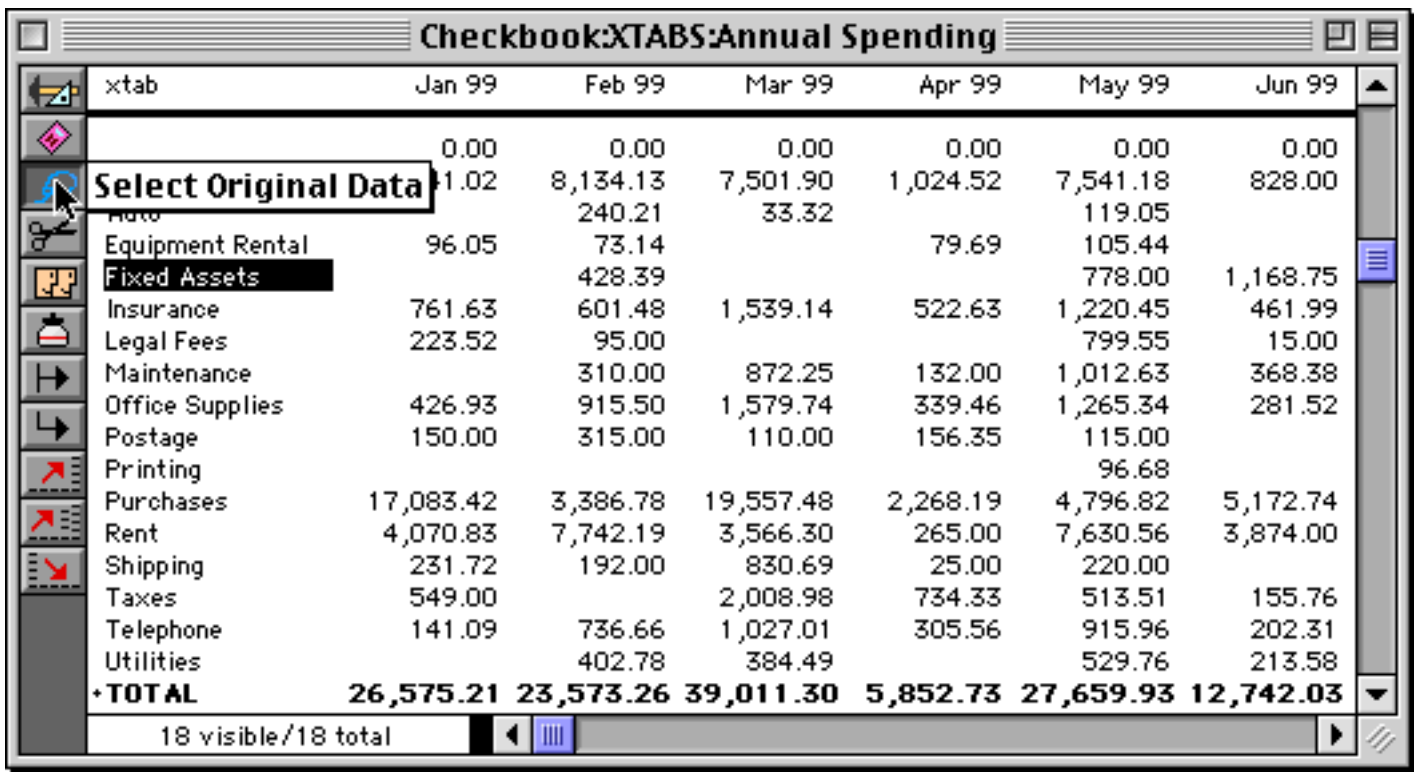
xtab	Jan 99	Feb 99	Mar 99	Apr 99	May 99	Jun 99
	0.00	0.00	0.00	0.00	0.00	0.00
	1.02	8,134.13	7,501.90	1,024.52	7,541.18	828.00
		240.21	33.32		119.05	
Equipment Rental	96.05	73.14		79.69	105.44	
Fixed Assets		428.39			778.00	1,168.75
Insurance	761.63	601.48	1,539.14	522.63	1,220.45	461.99
Legal Fees	223.52	95.00			799.55	15.00
Maintenance		310.00	872.25	132.00	1,012.63	368.38
Office Supplies	426.93	915.50	1,579.74	339.46	1,265.34	281.52
Postage	150.00	315.00	110.00	156.35	115.00	
Printing					96.68	
Purchases	17,083.42	3,386.78	19,557.48	2,268.19	4,796.82	5,172.74
Rent	4,070.83	7,742.19	3,566.30	265.00	7,630.56	3,874.00
Shipping	231.72	192.00	830.69	25.00	220.00	
Taxes	549.00		2,008.98	734.33	513.51	155.76
Telephone	141.09	736.66	1,027.01	305.56	915.96	202.31
Utilities		402.78	384.49		529.76	213.58
TOTAL	**26,575.21**	**23,573.26**	**39,011.30**	**5,852.73**	**27,659.93**	**12,742.03**

In the data sheet the raw data backing up this crosstab cell will be selected - in this case the seven checks written for Office Supplies in March of 1999.



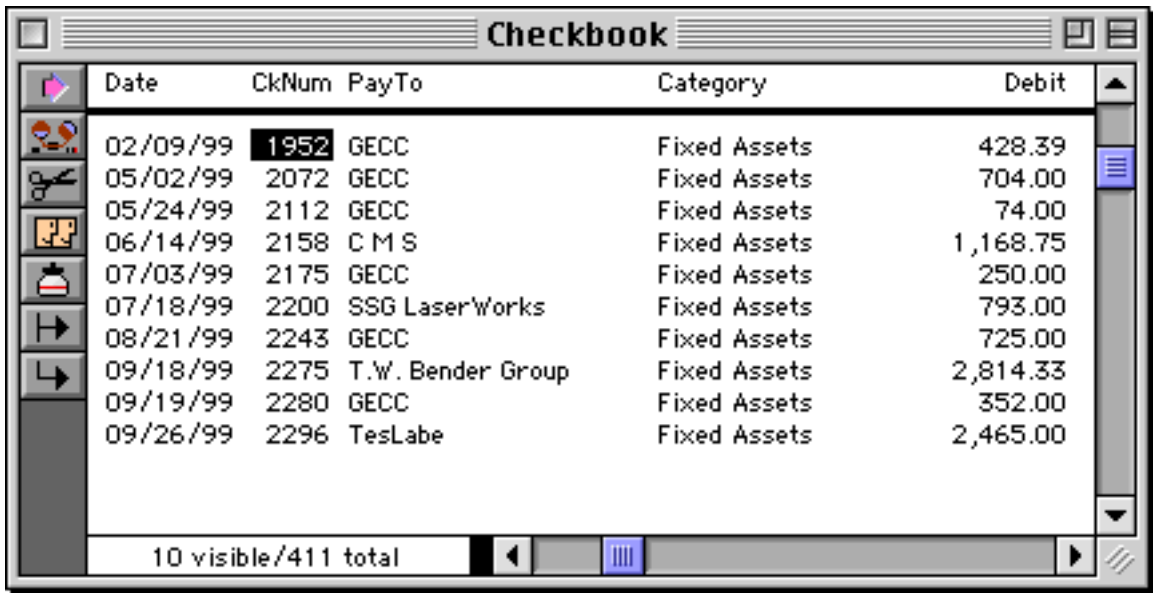
Date	CkNum	PayTo	Category	Debit
03/12/99	1999	Pitney Bowes	Office Supplies	53.94
03/12/99	2001	Paper Mart	Office Supplies	572.78
03/12/99	2003	Pace Club	Office Supplies	267.47
03/28/99	2028	Pitney Bowes	Office Supplies	247.00
03/28/99	2030	NEBS	Office Supplies	67.20
03/28/99	2031	Ramona Drinking Water	Office Supplies	51.55
03/29/99	2036	Jeffco	Office Supplies	319.80

You can select the original data for any value in the crosstab. If you click on a cell in the first or last column of the crosstab, the **Select Original Data** tool will select all the data associated with the entire row. For example, you could click on the Fixed Assets cell.



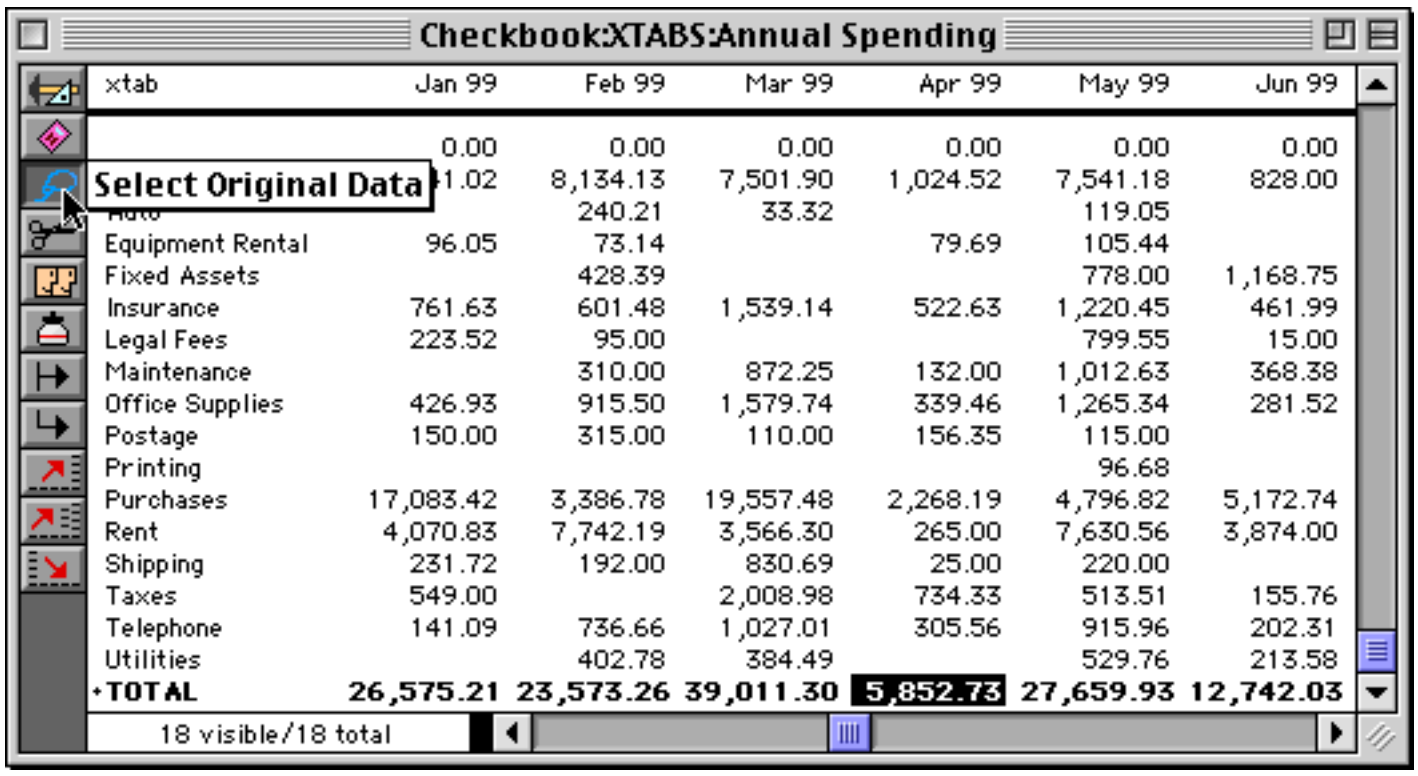
xtab	Jan 99	Feb 99	Mar 99	Apr 99	May 99	Jun 99
	0.00	0.00	0.00	0.00	0.00	0.00
	1.02	8,134.13	7,501.90	1,024.52	7,541.18	828.00
Auto		240.21	33.32		119.05	
Equipment Rental	96.05	73.14		79.69	105.44	
Fixed Assets		428.39			778.00	1,168.75
Insurance	761.63	601.48	1,539.14	522.63	1,220.45	461.99
Legal Fees	223.52	95.00			799.55	15.00
Maintenance		310.00	872.25	132.00	1,012.63	368.38
Office Supplies	426.93	915.50	1,579.74	339.46	1,265.34	281.52
Postage	150.00	315.00	110.00	156.35	115.00	
Printing					96.68	
Purchases	17,083.42	3,386.78	19,557.48	2,268.19	4,796.82	5,172.74
Rent	4,070.83	7,742.19	3,566.30	265.00	7,630.56	3,874.00
Shipping	231.72	192.00	830.69	25.00	220.00	
Taxes	549.00		2,008.98	734.33	513.51	155.76
Telephone	141.09	736.66	1,027.01	305.56	915.96	202.31
Utilities		402.78	384.49		529.76	213.58
TOTAL	**26,575.21**	**23,573.26**	**39,011.30**	**5,852.73**	**27,659.93**	**12,742.03**

When you choose the **Select Original Data** tool, the data sheet will show all of the checks written for Fixed Assets in every month.



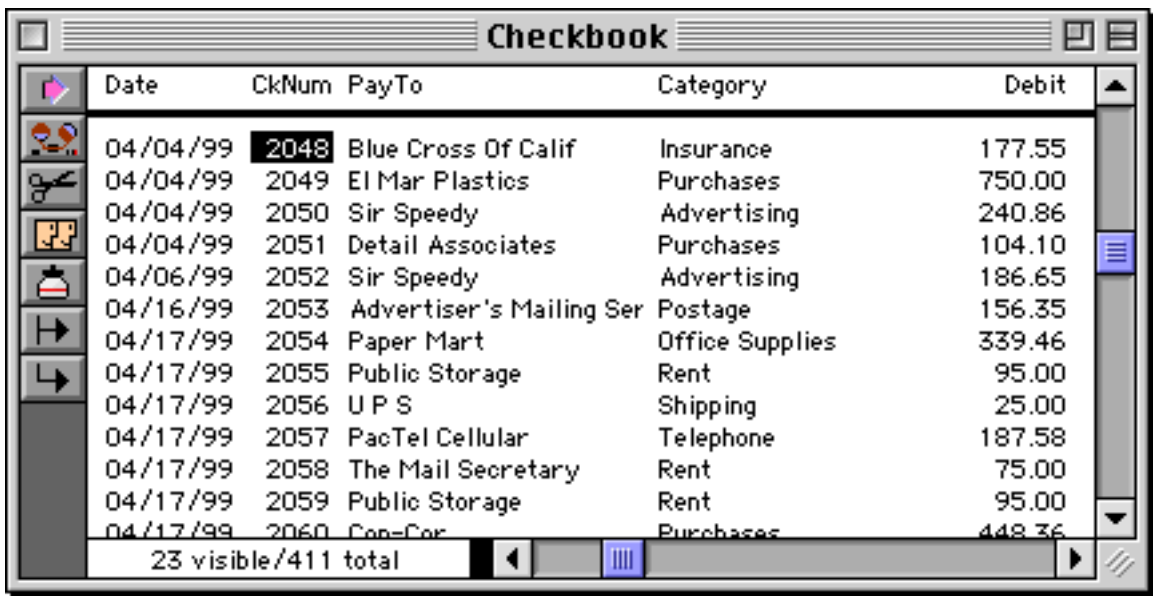
Date	CkNum	PayTo	Category	Debit
02/09/99	1952	GECC	Fixed Assets	428.39
05/02/99	2072	GECC	Fixed Assets	704.00
05/24/99	2112	GECC	Fixed Assets	74.00
06/14/99	2158	C M S	Fixed Assets	1,168.75
07/03/99	2175	GECC	Fixed Assets	250.00
07/18/99	2200	SSG LaserWorks	Fixed Assets	793.00
08/21/99	2243	GECC	Fixed Assets	725.00
09/18/99	2275	T.W. Bender Group	Fixed Assets	2,814.33
09/19/99	2280	GECC	Fixed Assets	352.00
09/26/99	2296	TesLabe	Fixed Assets	2,465.00

If you click on a cell in the bottom row of the crosstab, the **Select Original Data** tool will select all of the data for the entire column. For example, you could click on the total for April 99.



xtab	Jan 99	Feb 99	Mar 99	Apr 99	May 99	Jun 99
	0.00	0.00	0.00	0.00	0.00	0.00
	1.02	8,134.13	7,501.90	1,024.52	7,541.18	828.00
Auto		240.21	33.32		119.05	
Equipment Rental	96.05	73.14		79.69	105.44	
Fixed Assets		428.39			778.00	1,168.75
Insurance	761.63	601.48	1,539.14	522.63	1,220.45	461.99
Legal Fees	223.52	95.00			799.55	15.00
Maintenance		310.00	872.25	132.00	1,012.63	368.38
Office Supplies	426.93	915.50	1,579.74	339.46	1,265.34	281.52
Postage	150.00	315.00	110.00	156.35	115.00	
Printing					96.68	
Purchases	17,083.42	3,386.78	19,557.48	2,268.19	4,796.82	5,172.74
Rent	4,070.83	7,742.19	3,566.30	265.00	7,630.56	3,874.00
Shipping	231.72	192.00	830.69	25.00	220.00	
Taxes	549.00		2,008.98	734.33	513.51	155.76
Telephone	141.09	736.66	1,027.01	305.56	915.96	202.31
Utilities		402.78	384.49		529.76	213.58
TOTAL	**26,575.21**	**23,573.26**	**39,011.30**	**5,852.73**	**27,659.93**	**12,742.03**

When you choose the **Select Original Data** tool, the data sheet will show all of the checks written for April 99 in every category.



Date	CkNum	PayTo	Category	Debit
04/04/99	2048	Blue Cross Of Calif	Insurance	177.55
04/04/99	2049	El Mar Plastics	Purchases	750.00
04/04/99	2050	Sir Speedy	Advertising	240.86
04/04/99	2051	Detail Associates	Purchases	104.10
04/06/99	2052	Sir Speedy	Advertising	186.65
04/16/99	2053	Advertiser's Mailing Ser	Postage	156.35
04/17/99	2054	Paper Mart	Office Supplies	339.46
04/17/99	2055	Public Storage	Rent	95.00
04/17/99	2056	U P S	Shipping	25.00
04/17/99	2057	PacTel Cellular	Telephone	187.58
04/17/99	2058	The Mail Secretary	Rent	75.00
04/17/99	2059	Public Storage	Rent	95.00
04/17/99	2060	Cop-Cor	Purchases	448.36

If you click on the grand total value in the lower right hand corner of the crosstab, the **Select Original Data** tool will select the entire database.

To re-select the entire original database, activate a data sheet or form window, then choose the **Select All** command from the Search Menu. Be sure to re-select the entire database before you re-calculate the crosstab.

Warning: If the database has been edited since the crosstab was calculated, the **Select Original Data** tool may not be able to locate the original data. If the database contained invisible (unselected) data when the crosstab was calculated, the **Select Original Data** tool may select this unselected data. If this is a problem, use the **Remove Unselected** command before calculating the crosstab (see "Permanently Removing Unselected Data" on page 378). Make sure you have a backup copy of your data on disk before you use this command.

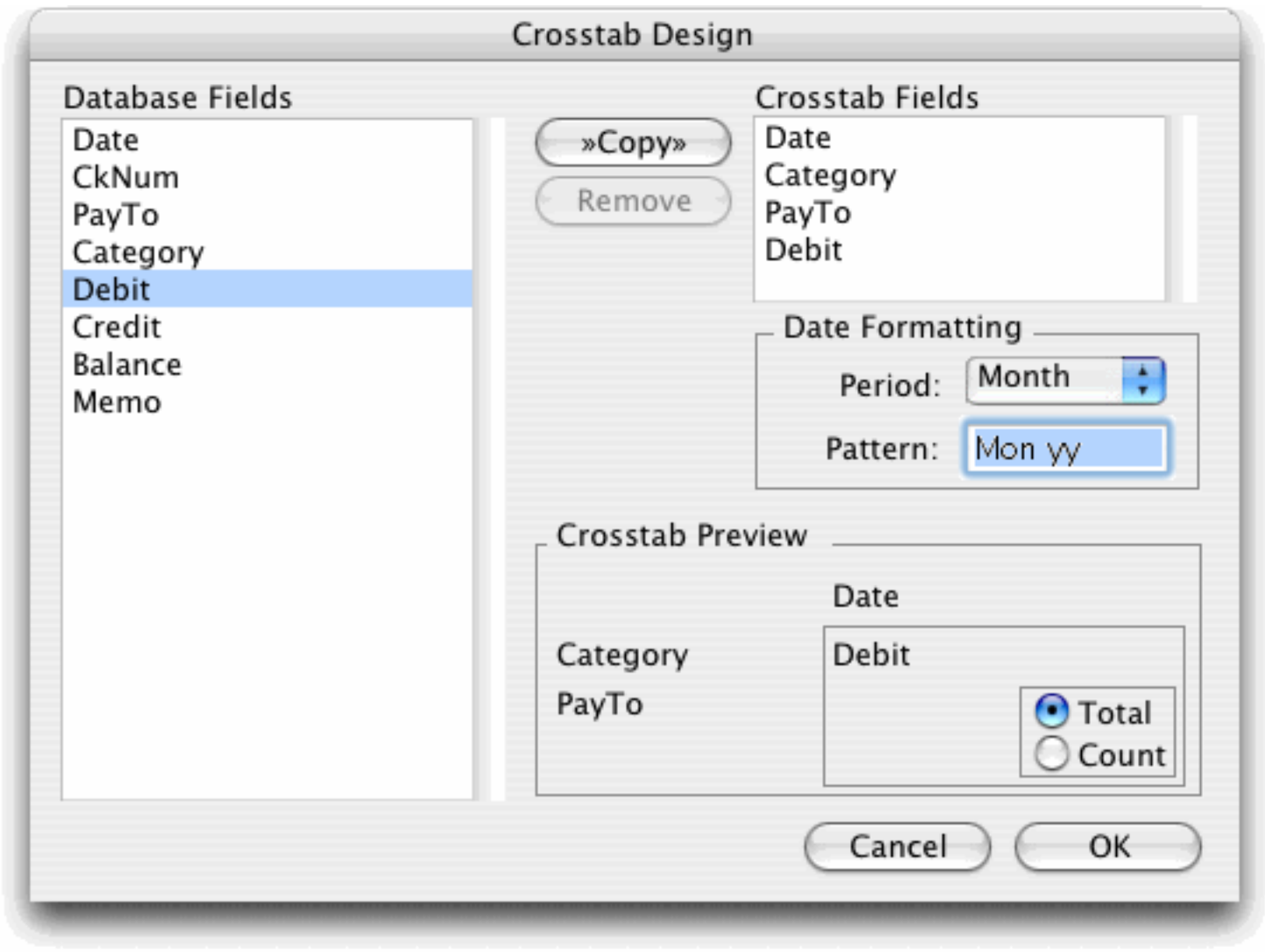
Crosstabs Based On Selected Data

To build a crosstab summary based on a subset of the database, use the **Find/Select** dialog before calculating the crosstab. For example, you can include this year's checks in a crosstab summary while excluding previous years. Be sure you select the data before you calculate the crosstab. (If you forget, just go back and select the data, then recalculate the crosstab).

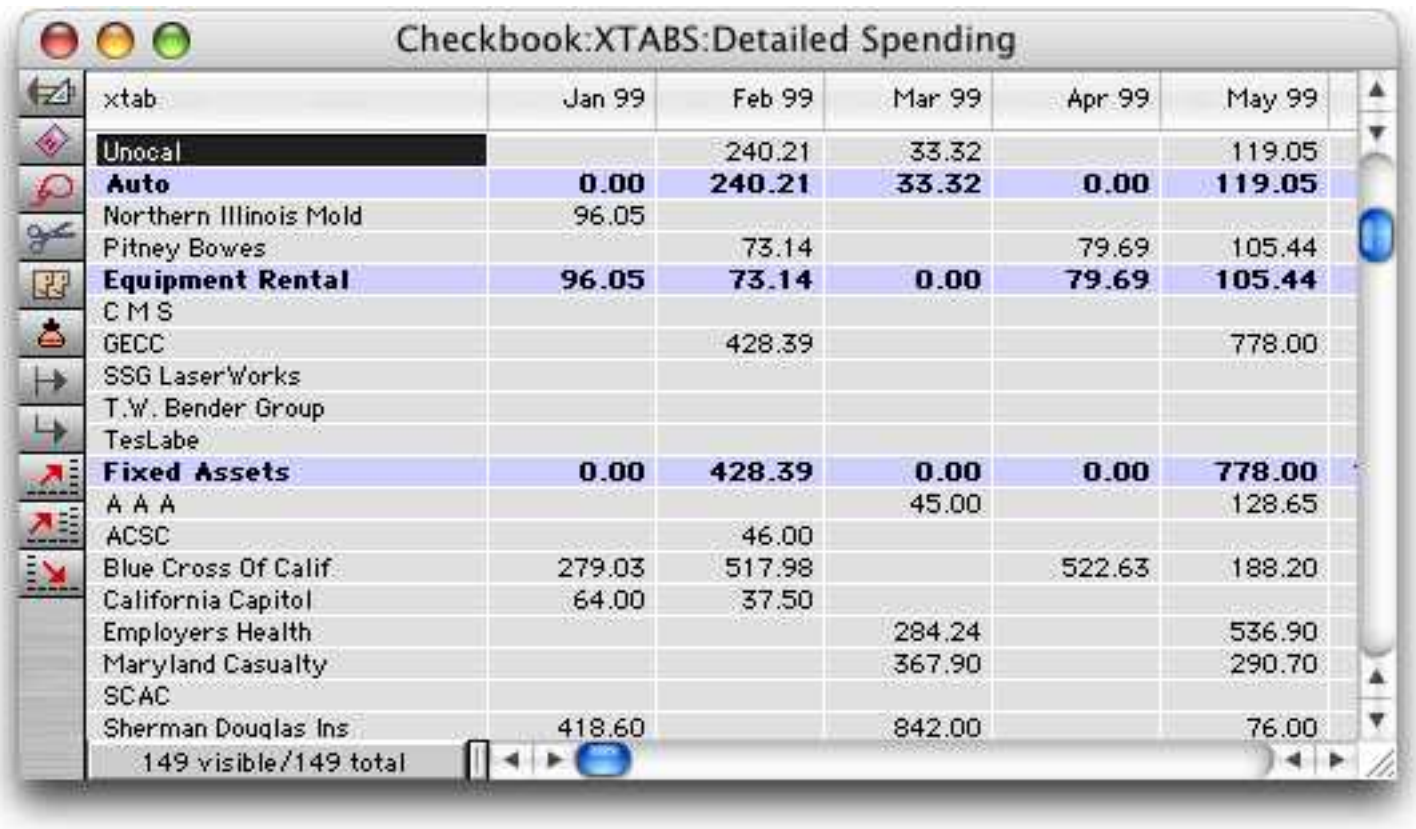
Crosstabs Containing Outlines

Most crosstabs have just two **category fields**—one across the top and one down the left side. It is possible, however, to create a crosstab with two (or more) category fields down the left side of the crosstab. In that case the category fields will be combined in an outline structure down the left side of the crosstab. You can use Panorama's outline tools to expand and collapse the crosstab to show more or less detail. See "STEP 3 - OUTLINE" on page 409 for more information on these outline tools.

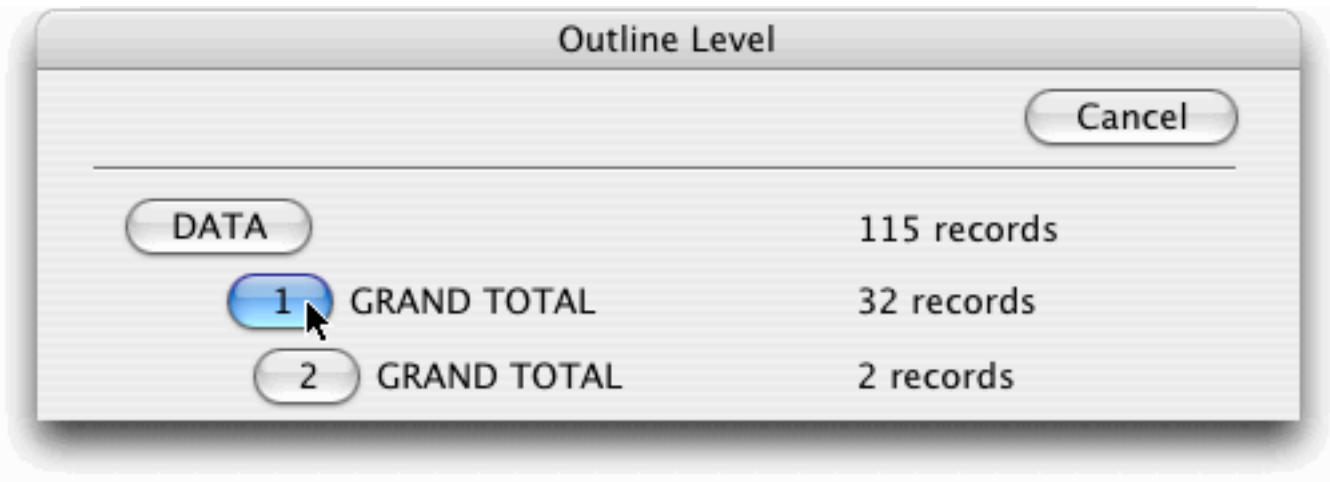
To set up a crosstab with more than one category field down the left, you must copy additional fields into the Crosstab Fields list. The first field is always the top category field, while the last field is always the tabulation field. The fields in the middle are the left category fields. Here is on example of what the dialog should look like when set up for a two level outline crosstab.



This crosstab breaks down spending per month not just for each category, but also for each vendor within each category.



Using the **Outline Level** command in the Sort Menu you can collapse the outline to show just the category summaries (see “The Outline Level” on page 409).



The collapsed outline looks kind of like our original, one-level crosstab.

Checkbook:XTABS:Detailed Spending

xtab	Jan 99	Feb 99	Mar 99	Apr 99	May 99	
Auto	0.00	240.21	33.32	0.00	119.05	
Equipment Rental	96.05	73.14	0.00	79.69	105.44	
Fixed Assets	0.00	428.39	0.00	0.00	778.00	
Insurance	761.63	601.48	1,539.14	522.63	1,220.45	
Legal Fees	223.52	95.00	0.00	0.00	799.55	
Maintenance	0.00	310.00	872.25	132.00	1,012.63	
Office Supplies	426.93	915.50	1,579.74	339.46	1,265.34	
Postage	150.00	315.00	110.00	156.35	115.00	
Printing	0.00	0.00	0.00	0.00	96.68	
Purchases	17,083.42	3,386.78	19,557.48	2,268.19	4,796.82	
Rent	4,070.83	7,742.19	3,566.30	265.00	7,630.56	
Shipping	231.72	192.00	830.69	25.00	220.00	
Taxes	549.00	0.00	2,008.98	734.33	513.51	
Telephone	141.09	736.66	1,027.01	305.56	915.96	
Utilities	0.00	402.78	384.49	0.00	529.76	
TOTAL	26,575.21	23,573.26	39,011.30	5,852.73	27,659.93	1

17 visible/149 total

There's a big difference, however. We can use the Outline tools to selectively expand and / or collapse sections of the crosstab (see "Expanding and Collapsing Specific Details" on page 413).

Checkbook:XTABS:Detailed Spending

xtab	Jan 99	Feb 99	Mar 99	Apr 99	May 99	
Office Supplies	426.93	915.50	1,579.74	339.46	1,265.34	
Postage	150.00	315.00	110.00	156.35	115.00	
Printing	0.00	0.00	0.00	0.00	96.68	
Purchases	17,083.42	3,386.78	19,557.48	2,268.19	4,796.82	
Rent	4,070.83	7,742.19	3,566.30	265.00	7,630.56	
Airborne Express		35.40				
AIRS			138.07			
American Customs House Broke		34.00				
B J D Trucking Inc.						
Burlington Air Express						
Consolidated Freight			150.20			
Federal Express	178.75		320.00		170.00	
U P S	52.97	122.60	222.42	25.00	50.00	
Shipping	231.72	192.00	830.69	25.00	220.00	
Taxes	549.00	0.00	2,008.98	734.33	513.51	
Telephone	141.09	736.66	1,027.01	305.56	915.96	
Utilities	0.00	402.78	384.49	0.00	529.76	
TOTAL	26,575.21	23,573.26	39,011.30	5,852.73	27,659.93	1

25 visible/149 total

Don't forget, you can click on any cell in any cell, including a summary cell, and use **Select Original Data** to see the raw data that went into the cell.

Sorting a Crosstab

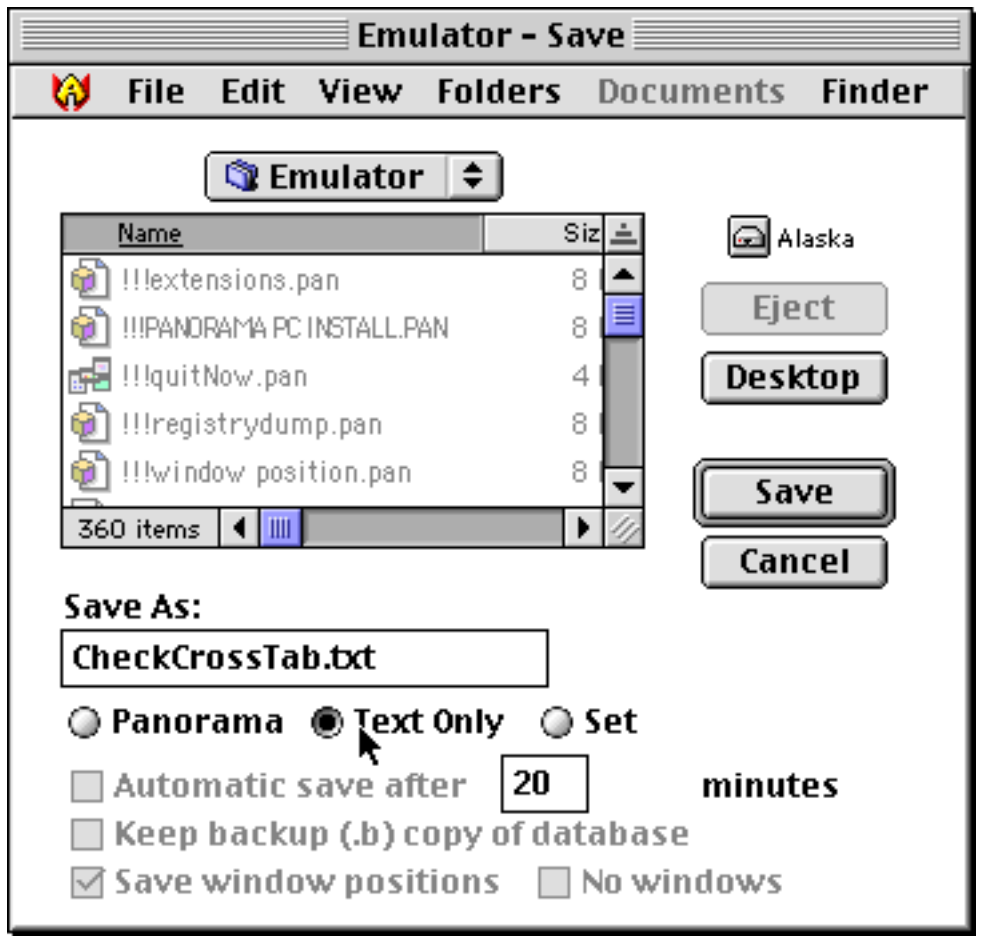
Crosstabs are automatically sorted by category. Use the **Sort Up** and **Sort Down** commands to sort the crosstab by other columns.

Removing and Renaming Crosstab Tables

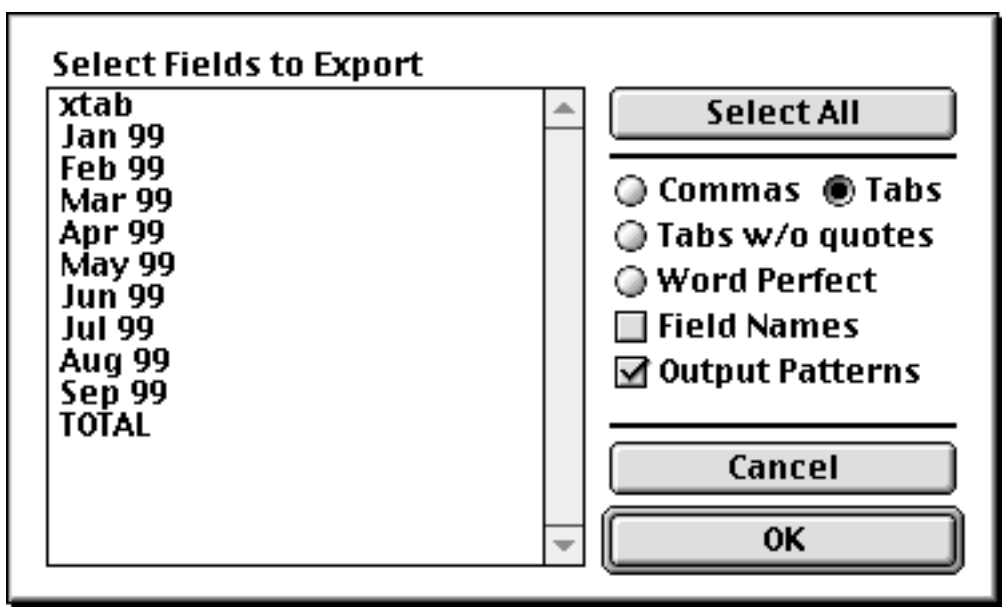
To remove a crosstab table, choose **Delete Crosstab** from the Setup Menu ("Deleting a Form, Crosstab or Procedure" on page 237). To rename a crosstab, choose **Rename Crosstab** from the Setup Menu (see "Renaming a Form, Crosstab or Procedure" on page 237). A crosstab name may be up to 25 characters long. You can also change the order of the crosstabs within the view menu using the **Re-Arrange Crosstabs** dialog (see "Changing the Order of Forms, Crosstabs or Procedures" on page 237).

Exporting a Crosstab Table

Use the **Save As** command to export the data in a crosstab table. Type in a file name and choose the Text Only option.



This brings up a second dialog allowing you to choose the columns you want to export and the export format (tab delimited, comma delimited, etc.).



Press the **Select All** button to select all the columns, then press **Ok** to export the data. See "Exporting a Text File" on page 165 for more information on the export dialog.

Once the crosstab data is exported, you can import it into another program or back into another Panorama database for further manipulation.

Chapter 12: Data Processing



This chapter describes some of the most powerful commands in Panorama. These commands allow you to automatically transform and modify large amounts of existing data. Many different kinds of transformations are possible, including mathematical calculations, re-arranging characters or words, transforming individual characters (for example converting from lower to upper case), and transformations based on patterns in the data.

The commands described in this chapter are very powerful. In a few seconds you may be able to make changes to your data that would otherwise require tedious hours of manual data entry. Like any power tool, these commands should be treated with respect. For insurance, you should **Save** your database before you begin trying to transform it (see "Saving a Database" on page 131). If you mangle your data, you can always get it back with the **Revert to Saved** command (see "Revert to Saved" on page 133).

Most of the commands in this chapter are found in the Math Menu. However, this doesn't mean that only numbers can be transformed. Unless specified otherwise, these commands can transform all kinds of data, including text, numeric, dates, and choices.

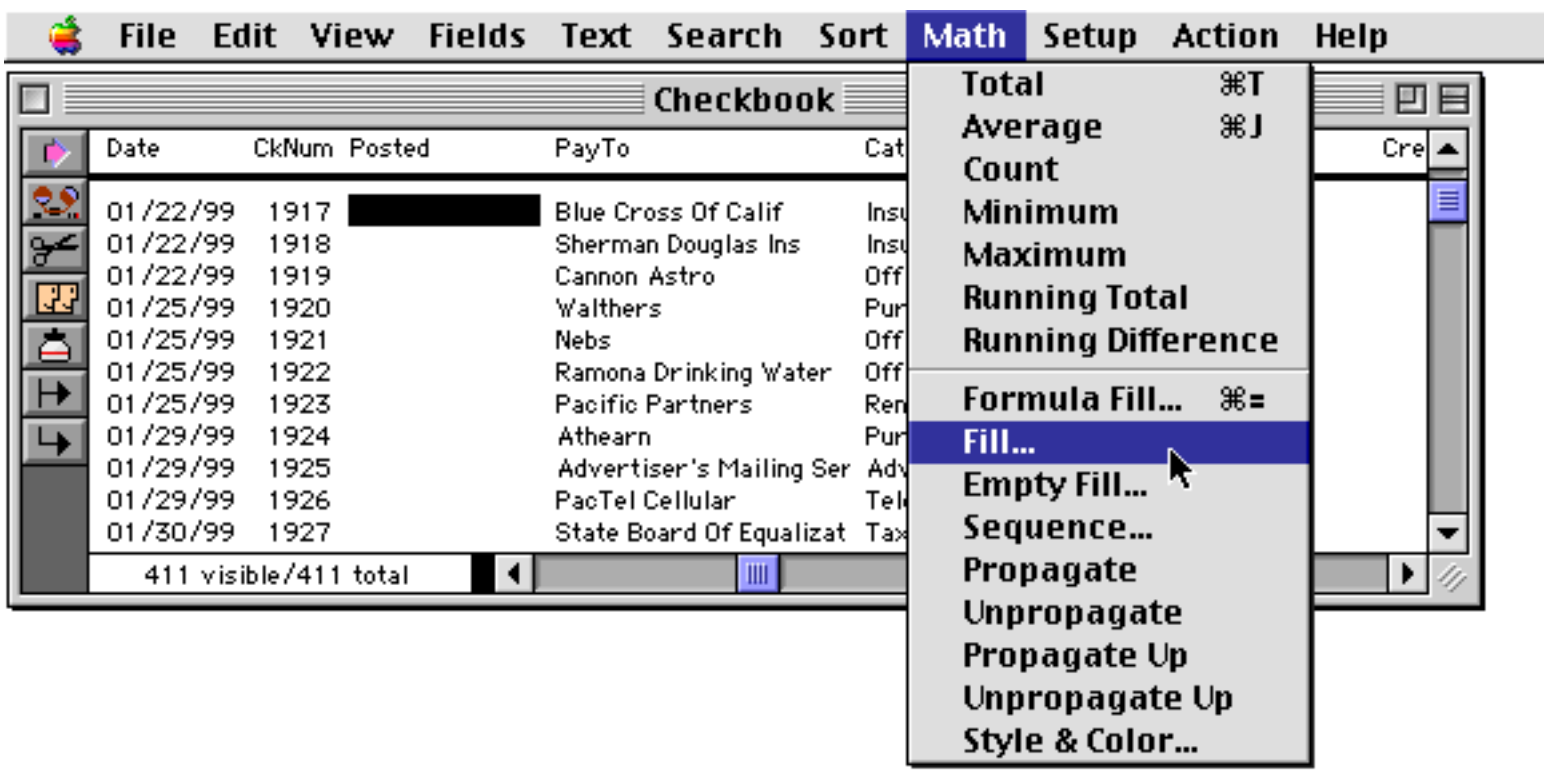
Transforming Selected Data

The transformation commands described in this chapter may be used on an entire database, or on a selected subset. The **Find/Select** (or **Formula Find/Select**) command is used to select the data you want to transform, then the commands described in this chapter are used to transform the data. Only the selected data will be transformed—the invisible data will be left untouched. See "The Find / Select Dialog" on page 369 for more information on selecting a subset of the database.

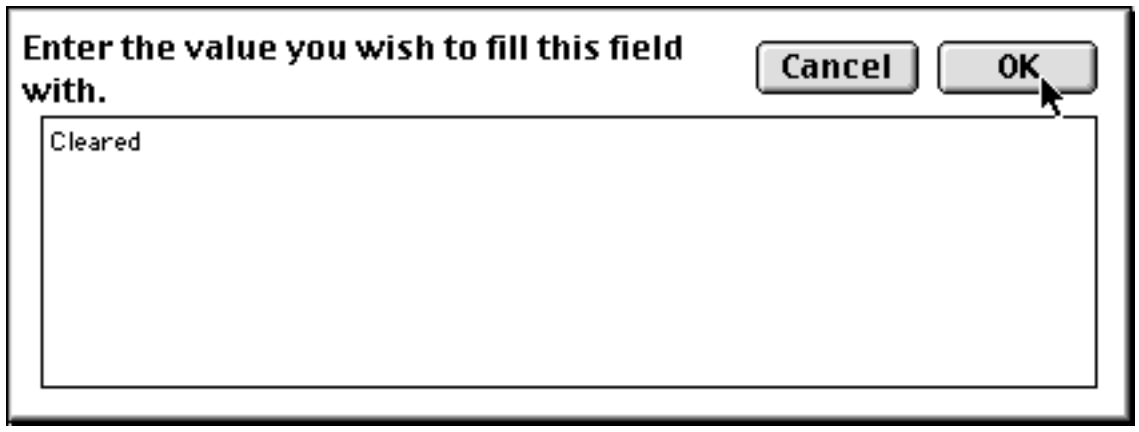
The same rules apply to data that has been collapsed with the outline tools. If data is invisible because it has been collapsed, it will not be transformed. Only data that is both selected and expanded will be transformed. See "Summaries and Outlines" on page 393 for more information on outlines.

Filling a Field with a Fixed Value

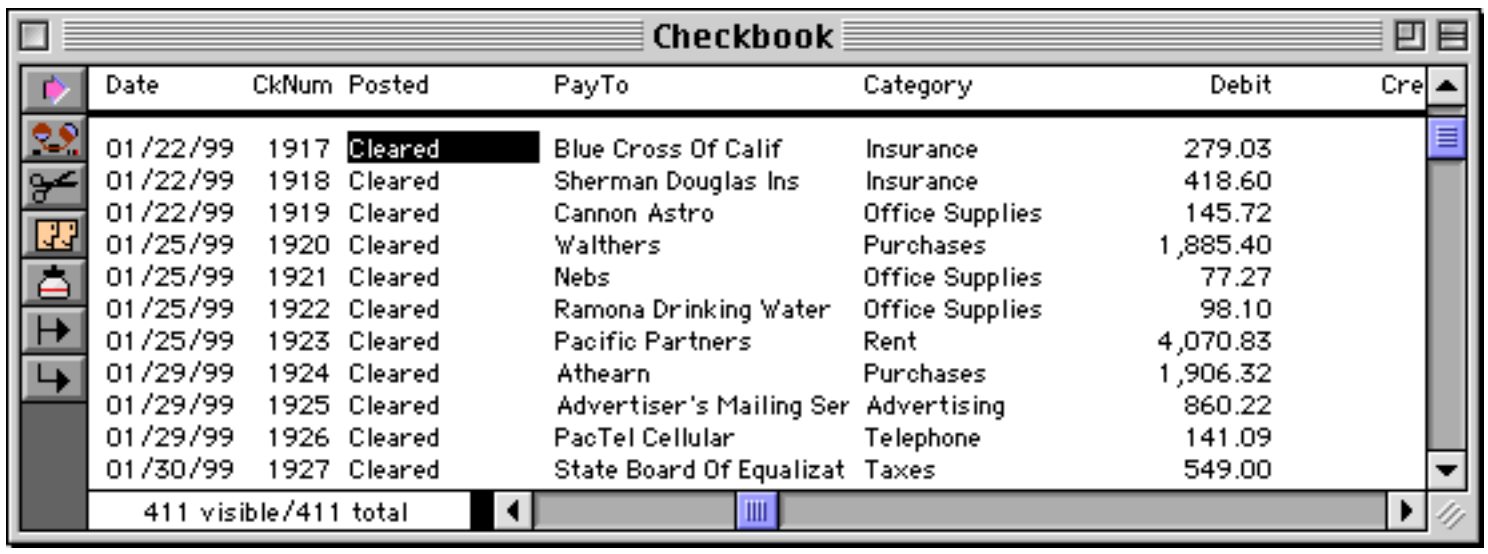
The **Fill** command fills all the selected cells in the current field with a single value. Any data already in the field is destroyed (although you can get it back with **Undo**). To use the **Fill** command, just click on the field you want to fill and choose **Fill** from the Math Menu. Type in the value you want to fill the field with and then press the **Ok** button. For example, suppose you wanted to fill every cell in the Posted column in the database below with the text Cleared. Start by clicking on the Posted column and choosing the **Fill** command.



Now type Cleared into the dialog box.



When you press OK, every selected cell in this field will be replaced with Cleared. In this case all of the cells are empty, but they will be replaced whether they are empty or not. (See “Filling Empty Cells” on page 463 if you don’t want to disturb cells that already have data in them.)



The new data must be compatible with the field that is being filled. For example, you cannot fill a numeric field with n/a because n/a is not a numeric value. Panorama will warn you if you attempt to fill a field with an incompatible value.

Filling a Field with a Formula

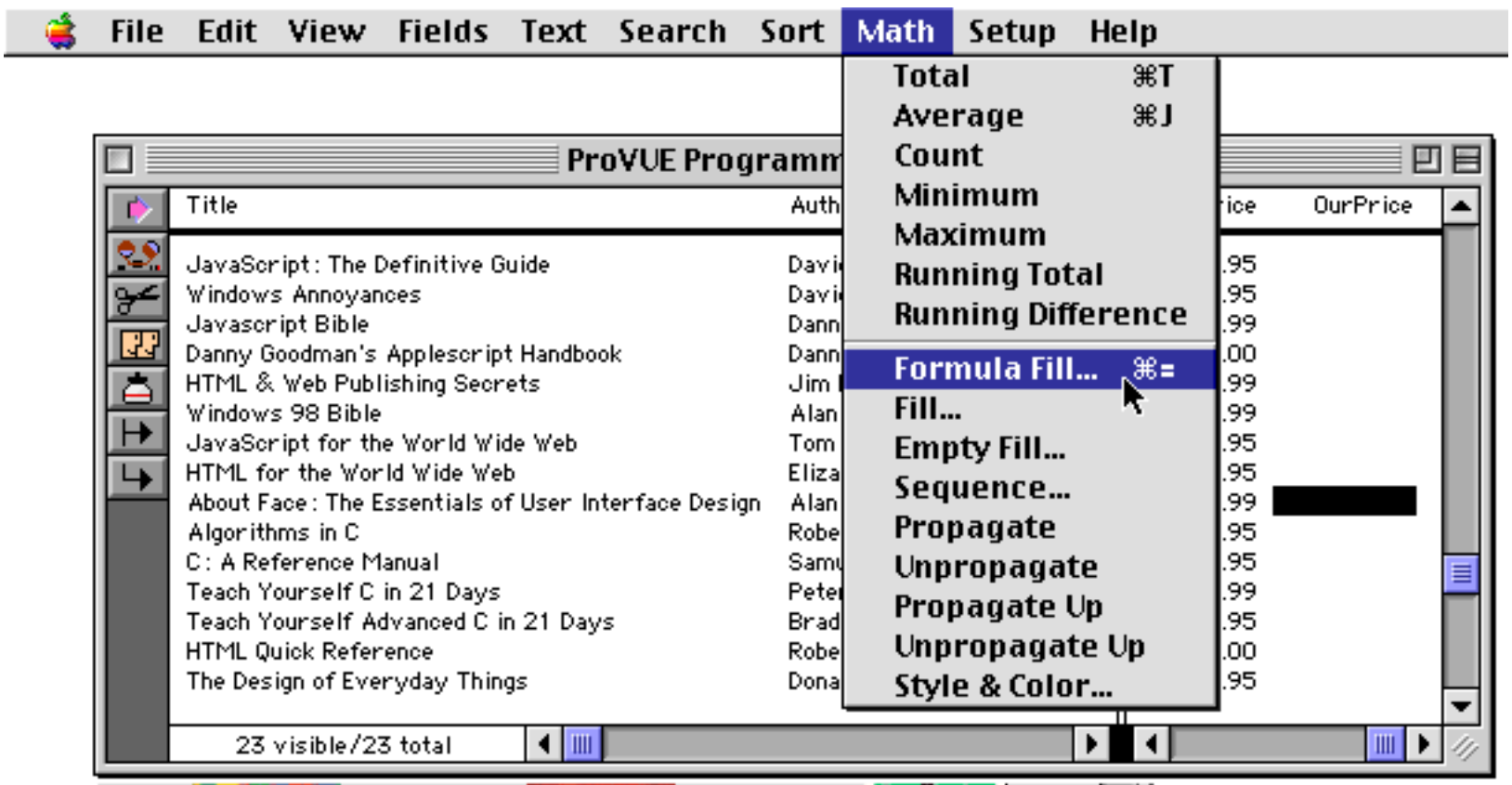
The **Formula Fill** command fills all the selected cells in the current field with the result of a formula. Any data already in the field is destroyed (although you can get it back with **Undo**). See “Formulas” on page 1165 for more information on formulas.

The formula must match the data type of the field being filled. For example, a numeric formula can only be used if the current field contains numeric data. Panorama will display an error message if you use a formula that results in an incorrect data type.

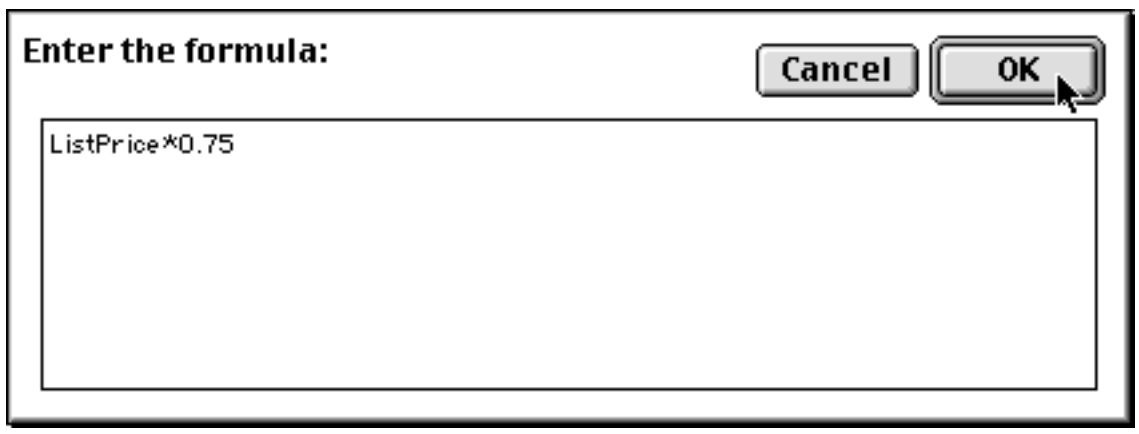
Use the **Formula Fill** command when you need to perform a calculation on every selected record in the database. See “Automatic Calculations” on page 340 if you need to calculate a value immediately when data is entered.

Numeric Calculations With Formula Fill

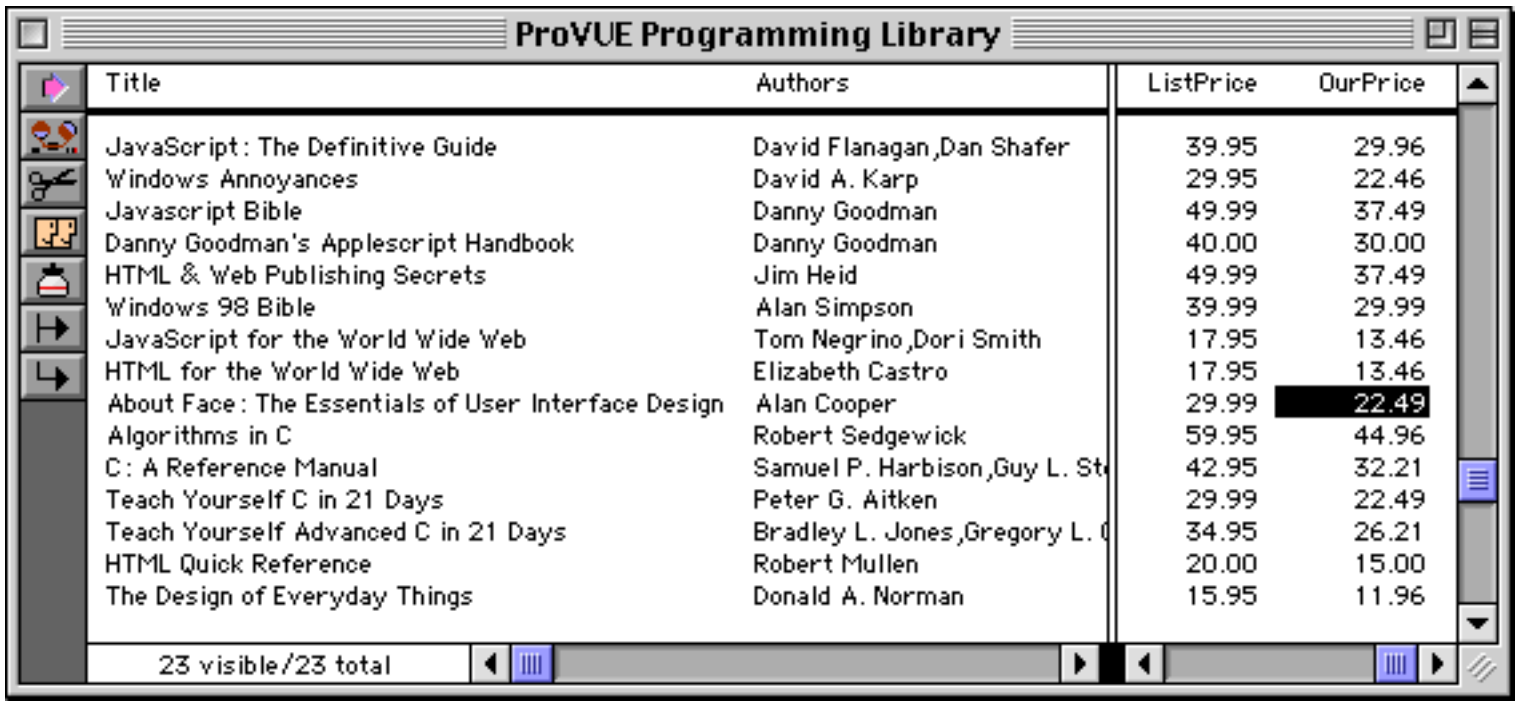
Use **Formula Fill** to perform calculations using the existing data. Use this command to calculate totals within a record, discounts, percentages, etc. For our example we'll use a database with two price fields: ListPrice and OurPrice. We'll use the **Formula Fill** command to calculate OurPrice based on a 25% discount from the list price. Start by clicking on the OurPrice field and choosing **Formula Fill** from the Math menu.



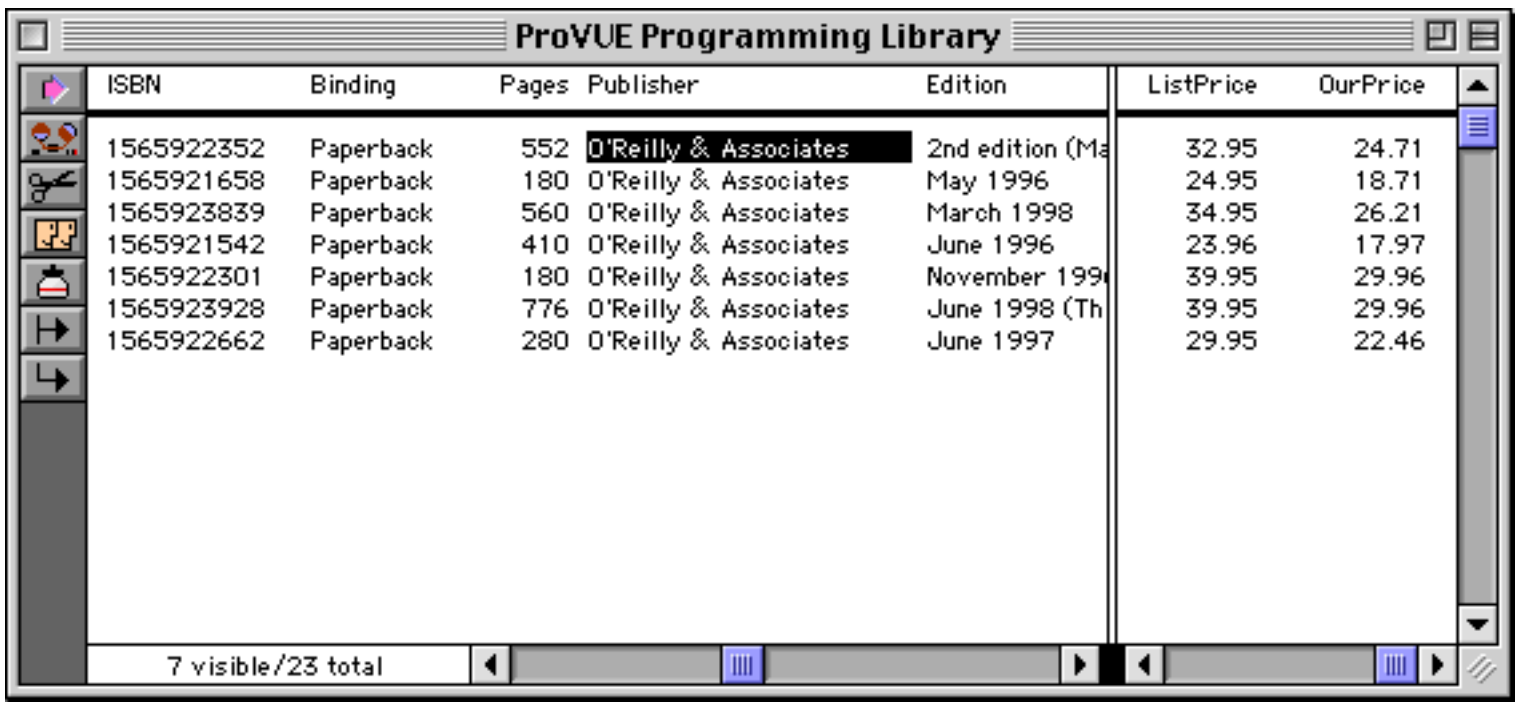
Now enter the formula to calculate the discount.



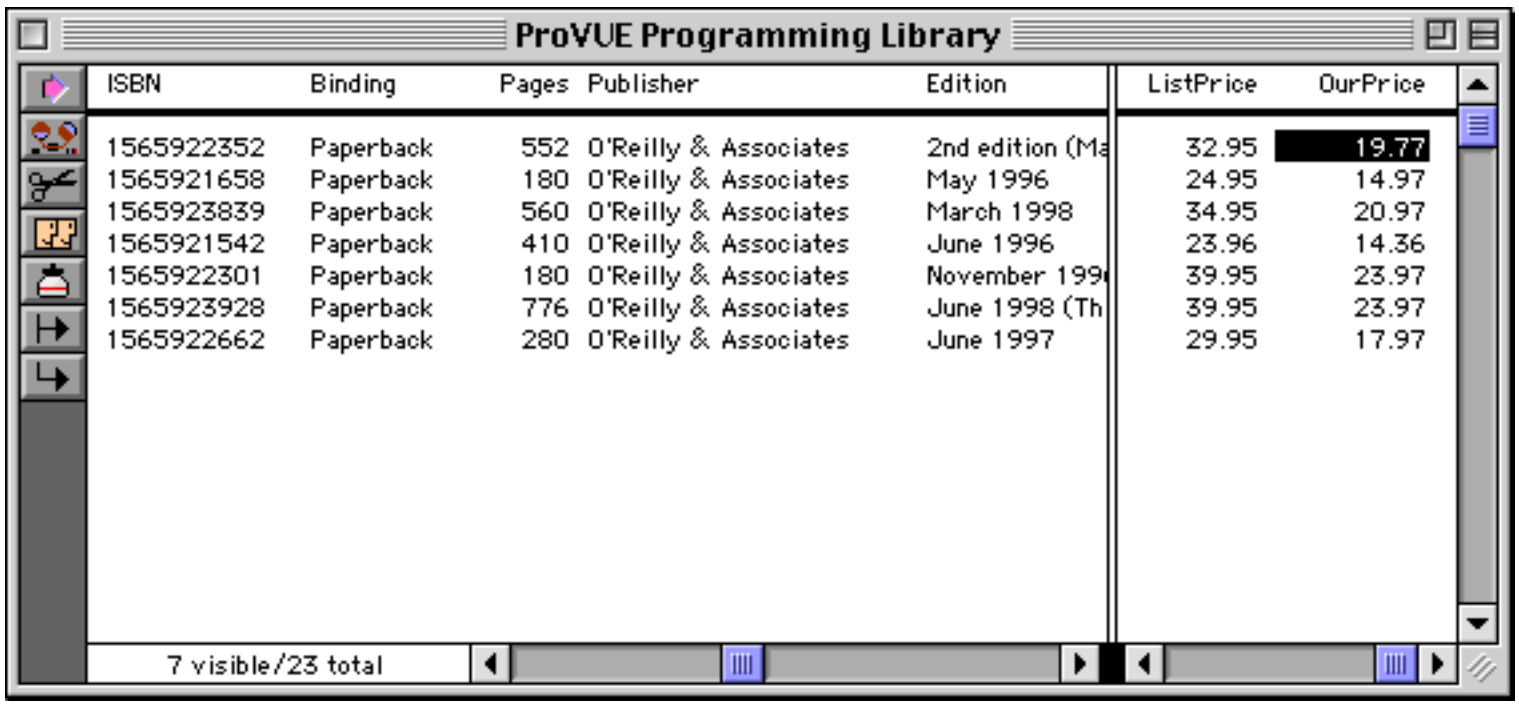
When you press **OK** Panorama will calculate the prices.



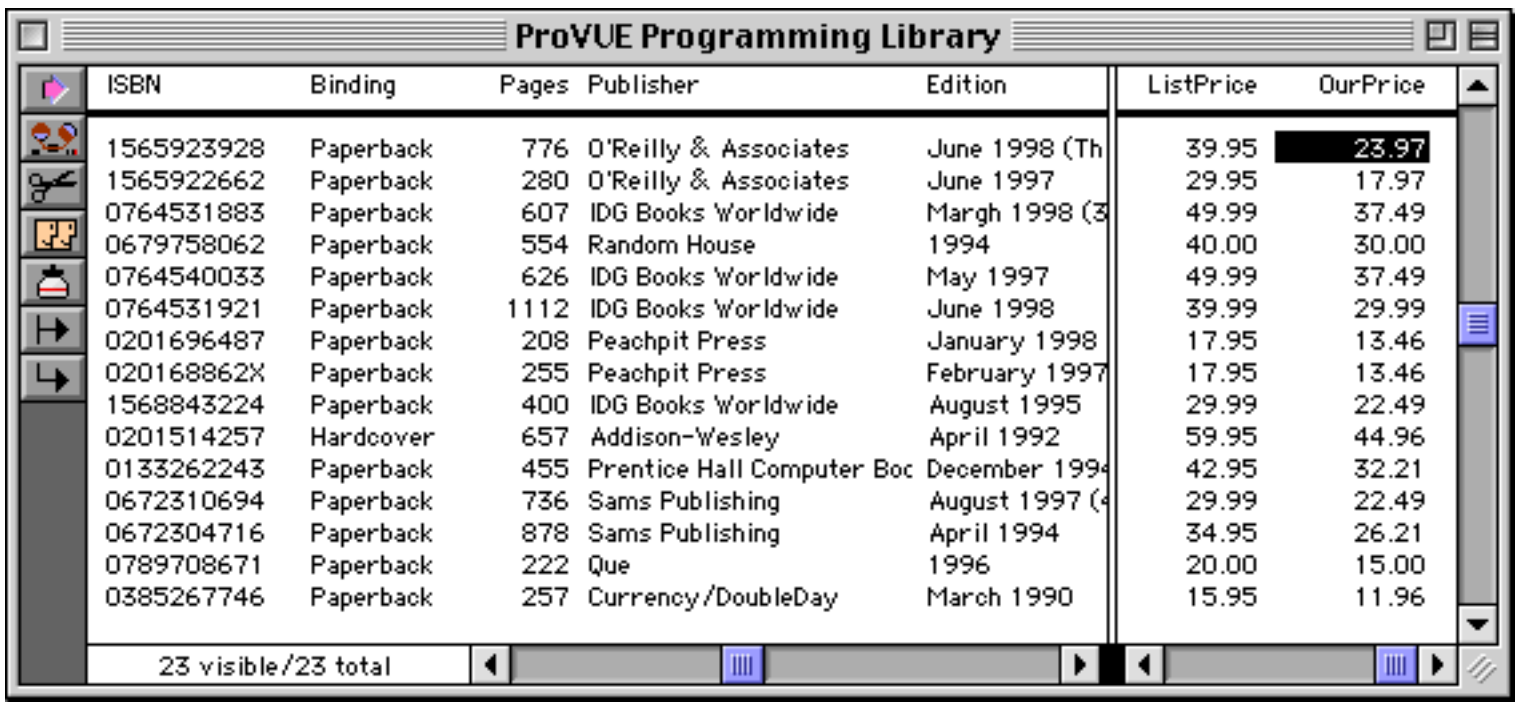
As a further example, suppose that through a special contract you are able to offer a 40% discount on books published by O'Reilly & Associates. To change the discount for these books, start by using the **Find/Select** command to select only the books for this publisher (See "The Find / Select Dialog" on page 369).



Now click on the OurPrice field, and use the **Formula Fill** command again to calculate the new discount for these books. The new formula is `ListPrice *0.6`.



When you **Select All** you'll see that all the other books have kept their 25% discount. **Formula Fill** (and in fact all of the commands described in this chapter) does not touch unselected records.



If you wanted to do this calculation often you will probably want to create a procedure to automate the process. You can record the procedure (see "Creating a Procedure with the Recorder" on page 1348) or type it in yourself (see "Writing a Procedure from Scratch" on page 1352). Here is the procedure.

```
field OurPrice
formulafill ListPrice*0.75
select Publisher="O'Reilly & Associates"
formulafill ListPrice*0.60
selectall
```

In this case it would also be possible to get the same effect in a single formula using the `?(` function (see "The ? Function" on page 1271), without using the **Find/Select** command. Here is the simplified form of the procedure.

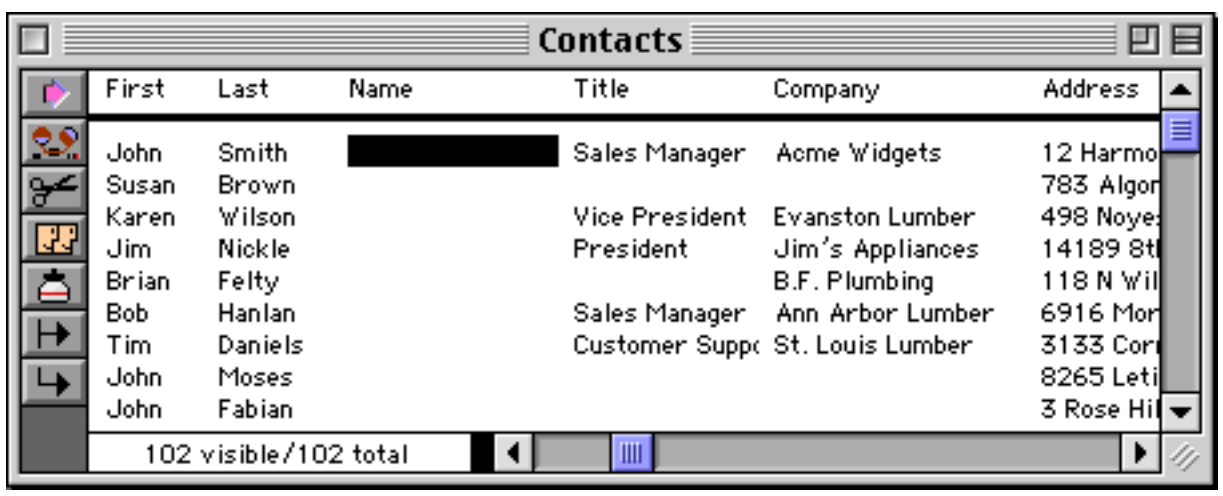
```
field OurPrice
formulafill ListPrice*?(Publisher="O'Reilly & Associates",0.60,0.75)
```

Remember that if you use **Formula Fill** to calculate numeric values, you must be filling a numeric field. (If you need to **Formula Fill** a text field with numeric values, use the `str(` or `pattern(` functions to convert the numbers to text. For example, the formula `str(Price-Cost)` can be used to fill a text field.) See "Converting Between Numbers and Strings" on page 1227 for more information.

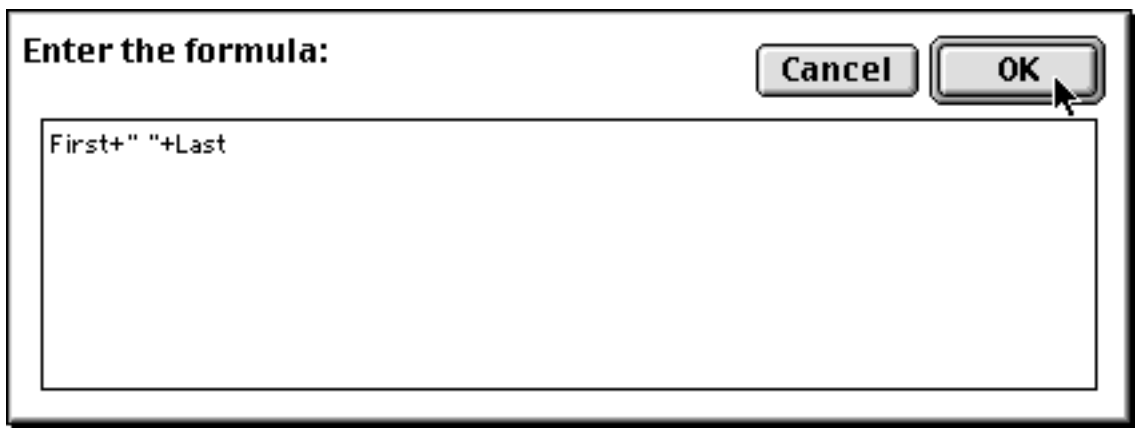
Using Formula Fill to Transform Characters

Using the **Formula Fill** command you can combine multiple fields, split a field apart, re-arrange words or phrases, and translate characters (for example, converting uppercase to lower case). Use the + operator to join (concatenate) text fields together.

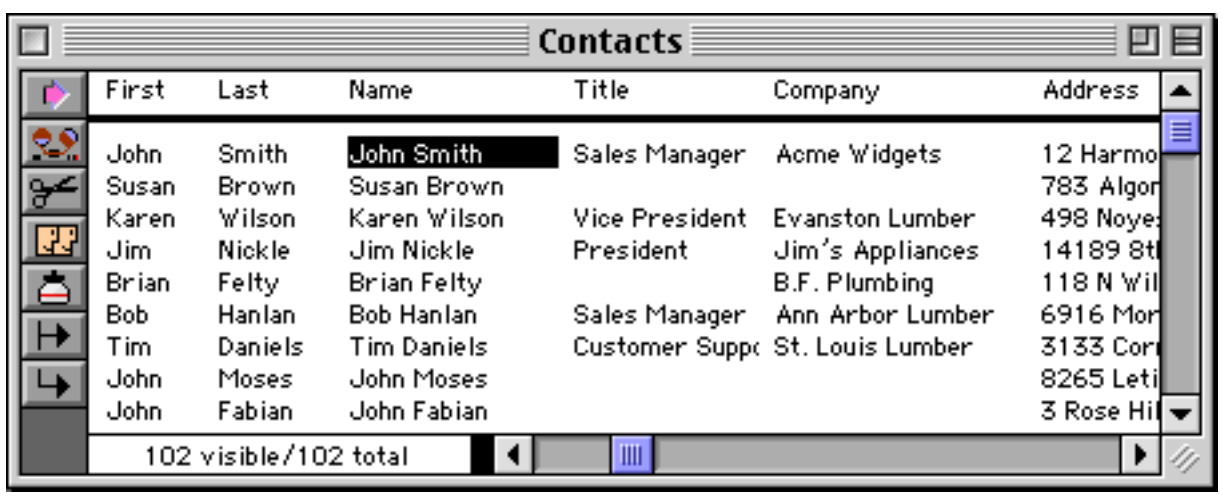
For example the formula `First+" "+Last` will combine the first and last names (with a space in between) into a single field. We'll start by inserting a blank `Name` field into our Contacts database (See "Add Field" on page 249).



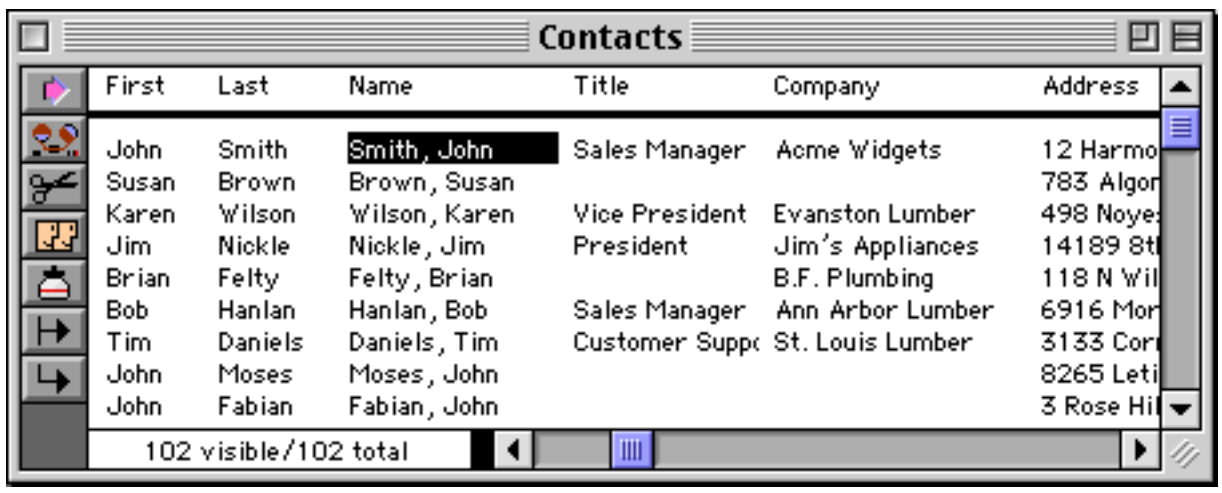
Now select the **Formula Fill** command from the Math Menu and type in the formula `First+" "+Last`.



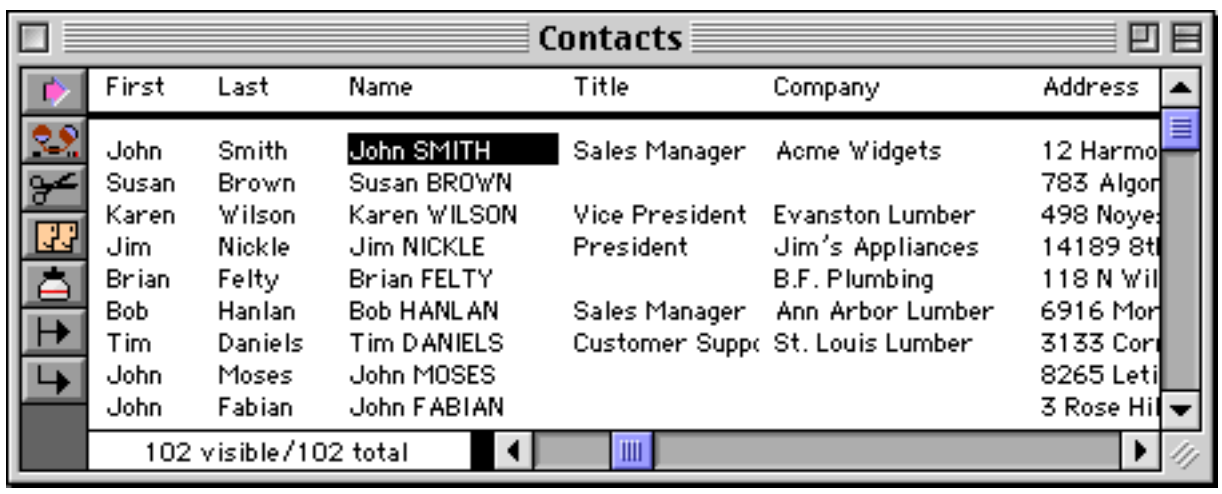
When you press **OK** the Name field will be filled in.



By using a different formula we can change the results. For example, if you wanted the last name first, you would use the formula `Last+", "+First` (see "Gluing Strings Together" on page 1212). Here's the result.



Use the `upper(`, `lower(`, and `upperword(` functions to convert text between upper and lower case (see "String Modification Functions" on page 1224). For example, if you wanted the last names in all upper case the formula `FirstName+" "+upper(LastName)` would be used. Here is the result.



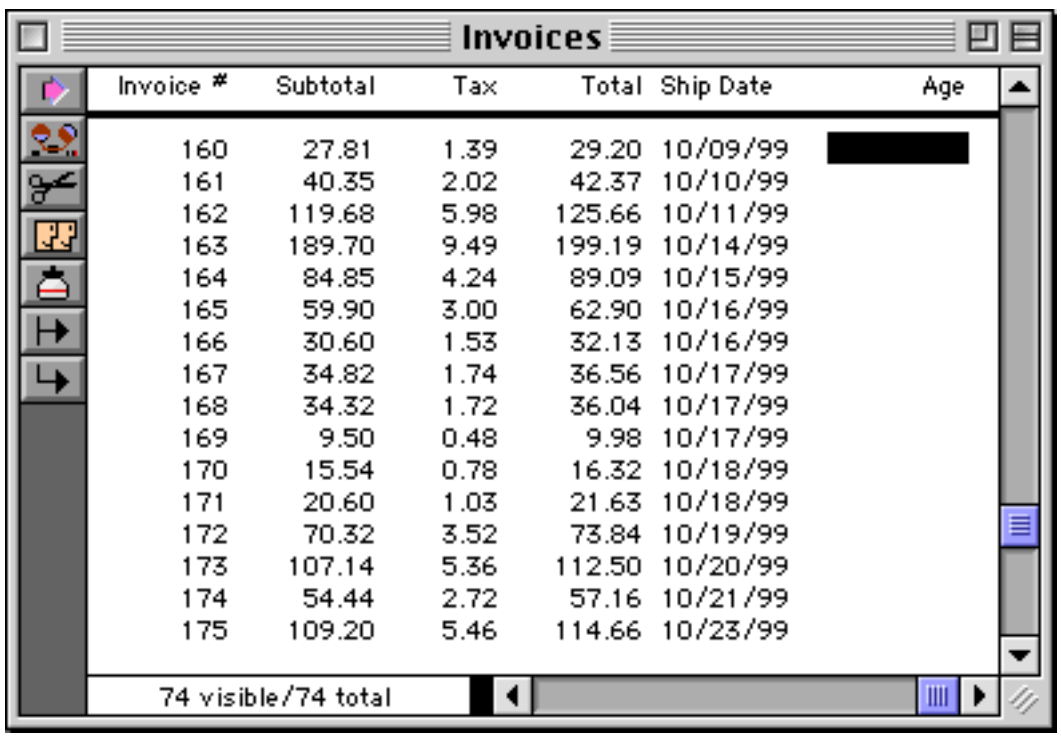
Use text funnels to split a field apart or to re-arrange words or phrases. Text funnels allow a formula to extract part of a cell based on a fixed character position within the cell, or based on patterns and context within the cell. See "Taking Strings Apart (Text Funnels)" on page 1214 for a complete explanation of text funnels. In this example we'll use the formula `FirstName[1,1]+". "+LastName` to fill the field with the first initial and the last name. Here is the result.



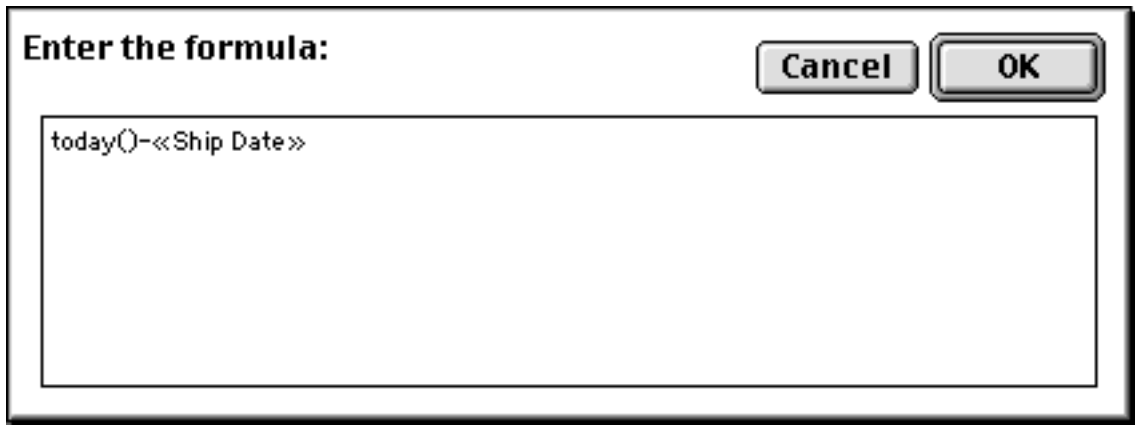
Date Calculations with Formula Fill

Use the **Formula Fill** command to calculate the difference between dates, or to adjust dates. See “HTML Generating Functions” on page 1247 for details on performing calculations with dates.

A typical use for date arithmetic is aging of an accounts receivable database.

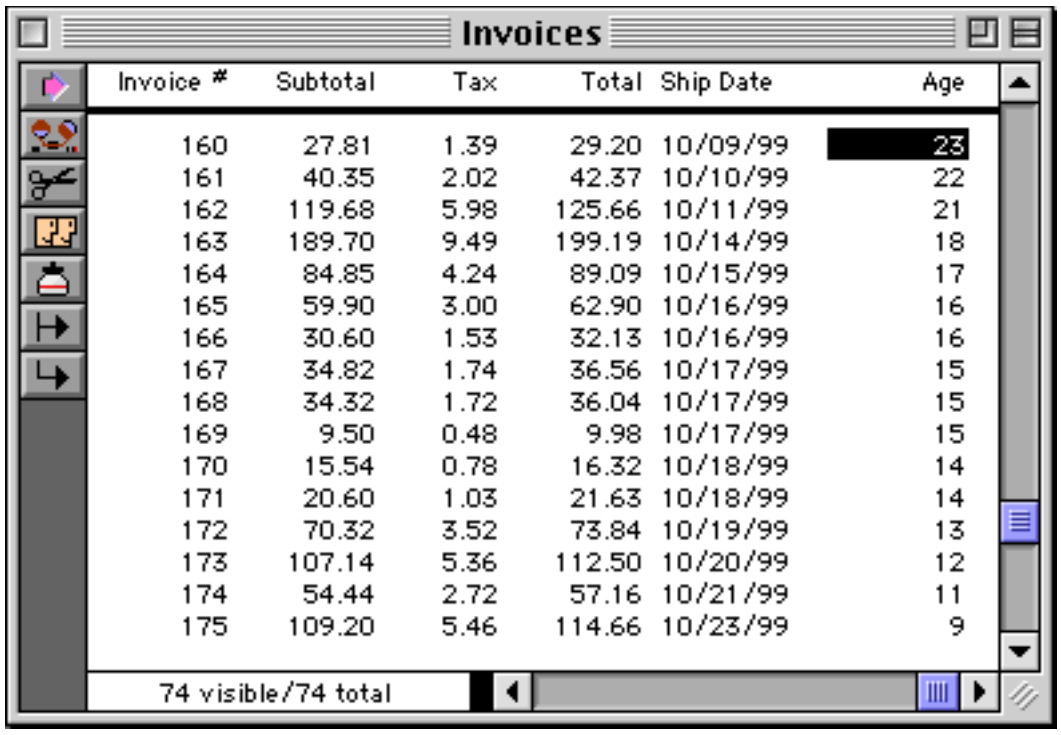


To calculate the age of an invoice based on the current date, use the **Formula Fill** command with the formula shown here:

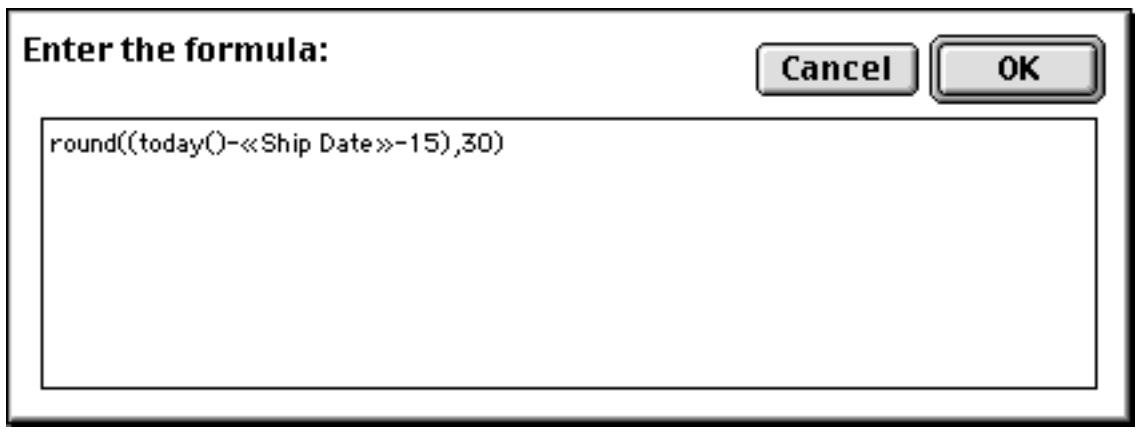


(Notice the chevrons (« and ») around the field name. These are necessary because of the space in the field name. You can simply let Panorama type the field name in for you by choosing from the **Field** menu.)

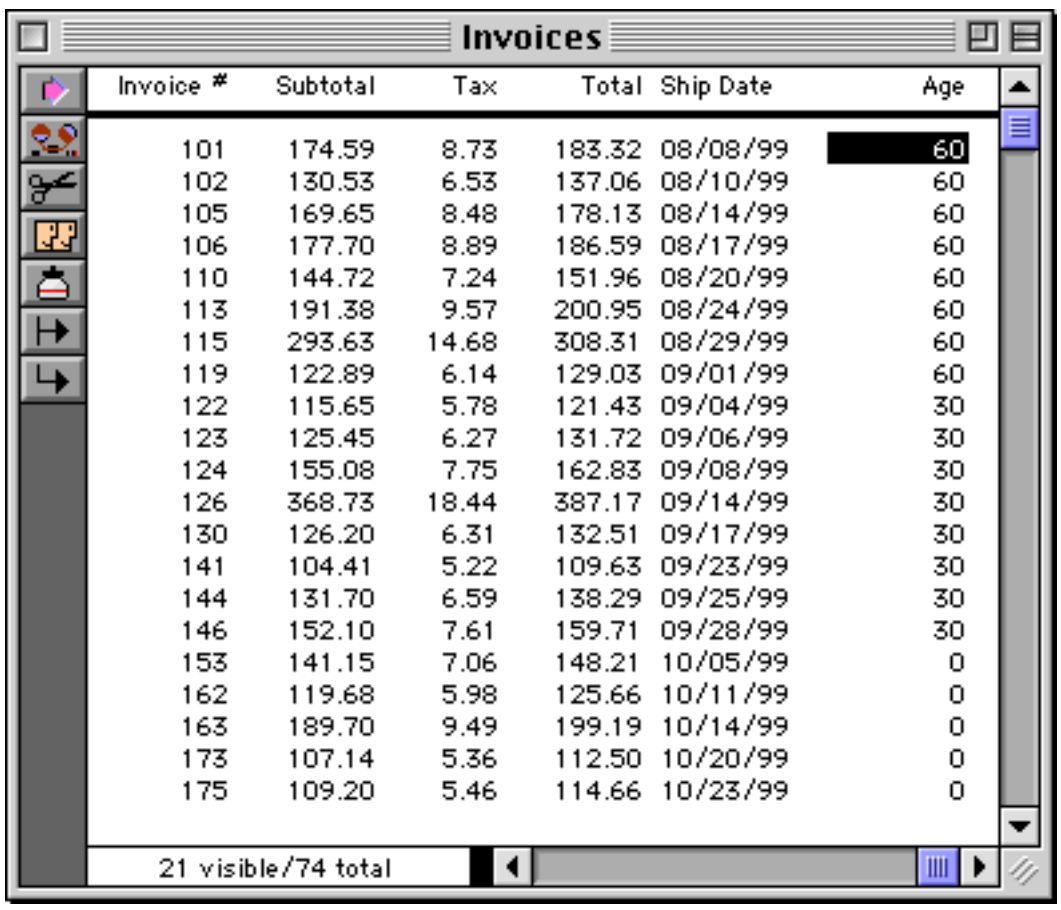
Press **OK** to calculate the age of each invoice.



If you want to calculate ages rounded to the nearest 30 day interval use the formula below instead.



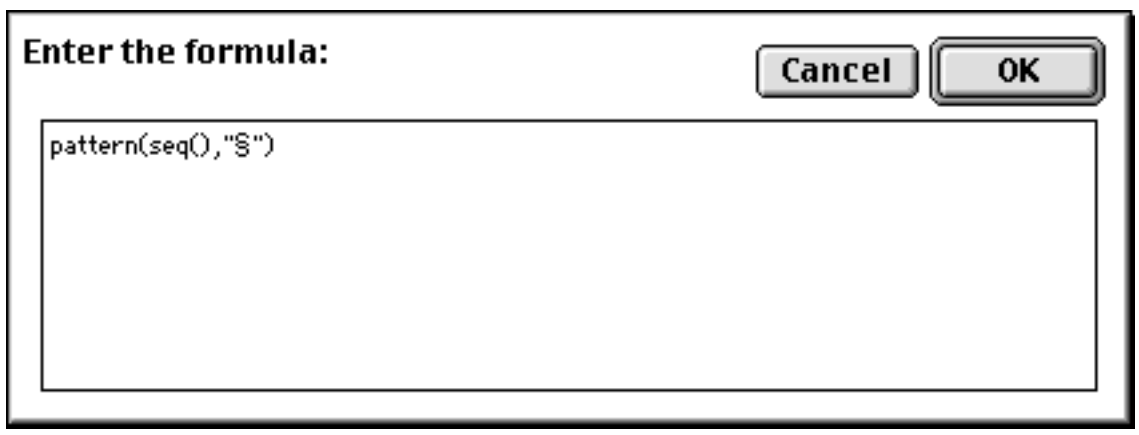
Press **OK** to calculate the age of each invoice rounded to the nearest 30 days.



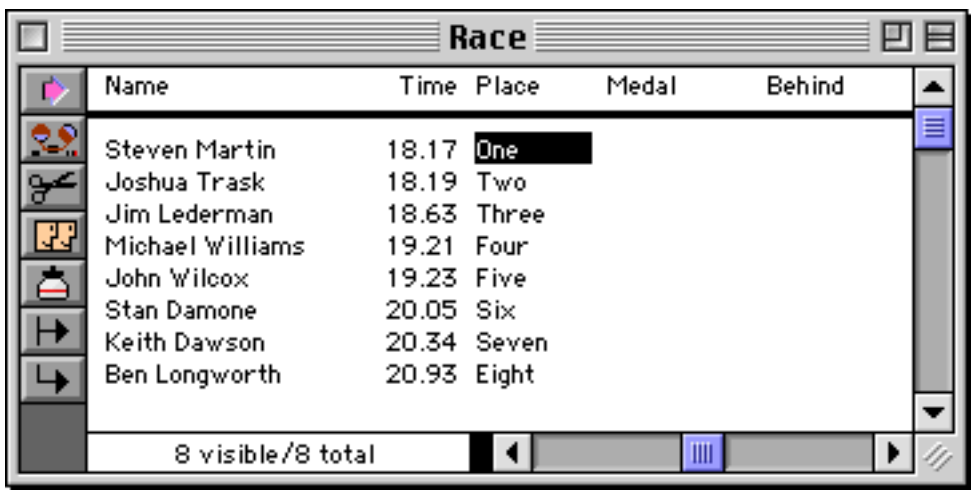
For more information on the `today(` and `round(` functions see “TODAY(” on page 5862 and “ROUND(” on page 5682.

The SEQ Function

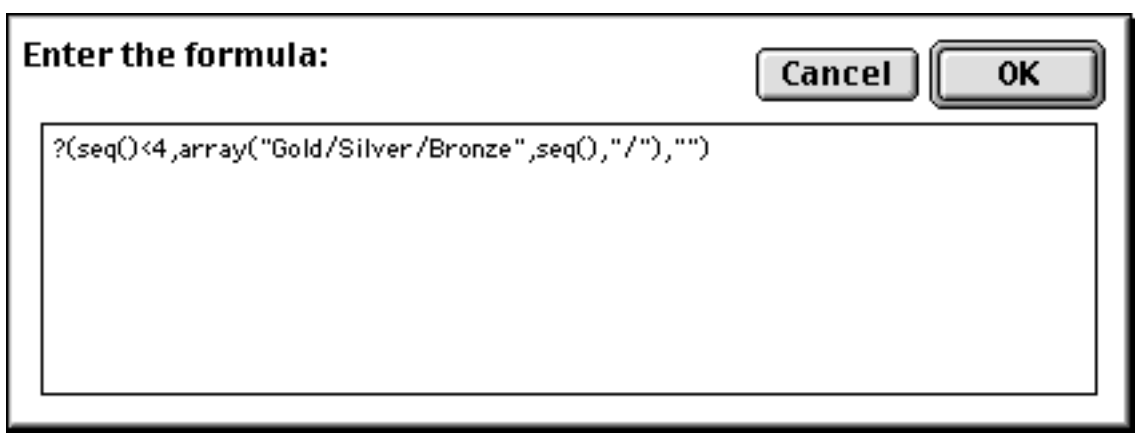
The `seq(` function is a special function for use with the **Formula Fill** command. This function returns a unique number for each selected record, starting with 1 at the top of the database. Use this function if you need a unique record number in a formula. Here is an example that fills a column with the words One, Two, Three, Four, etc.



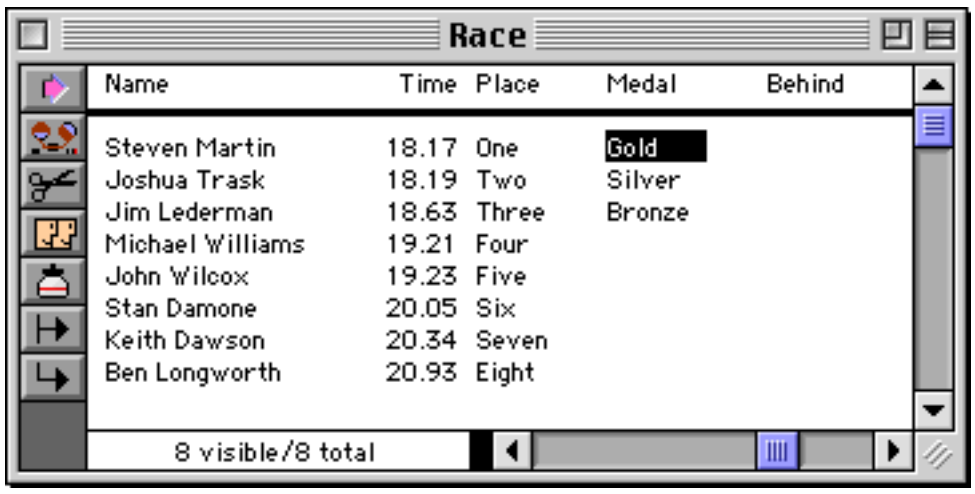
When you press **OK** the field is filled in (see "Displaying Numbers as Words" on page 294 for more information on this output pattern.)



Here is another example that uses the `seq(` function to assign medals to the first three finishers in the race.



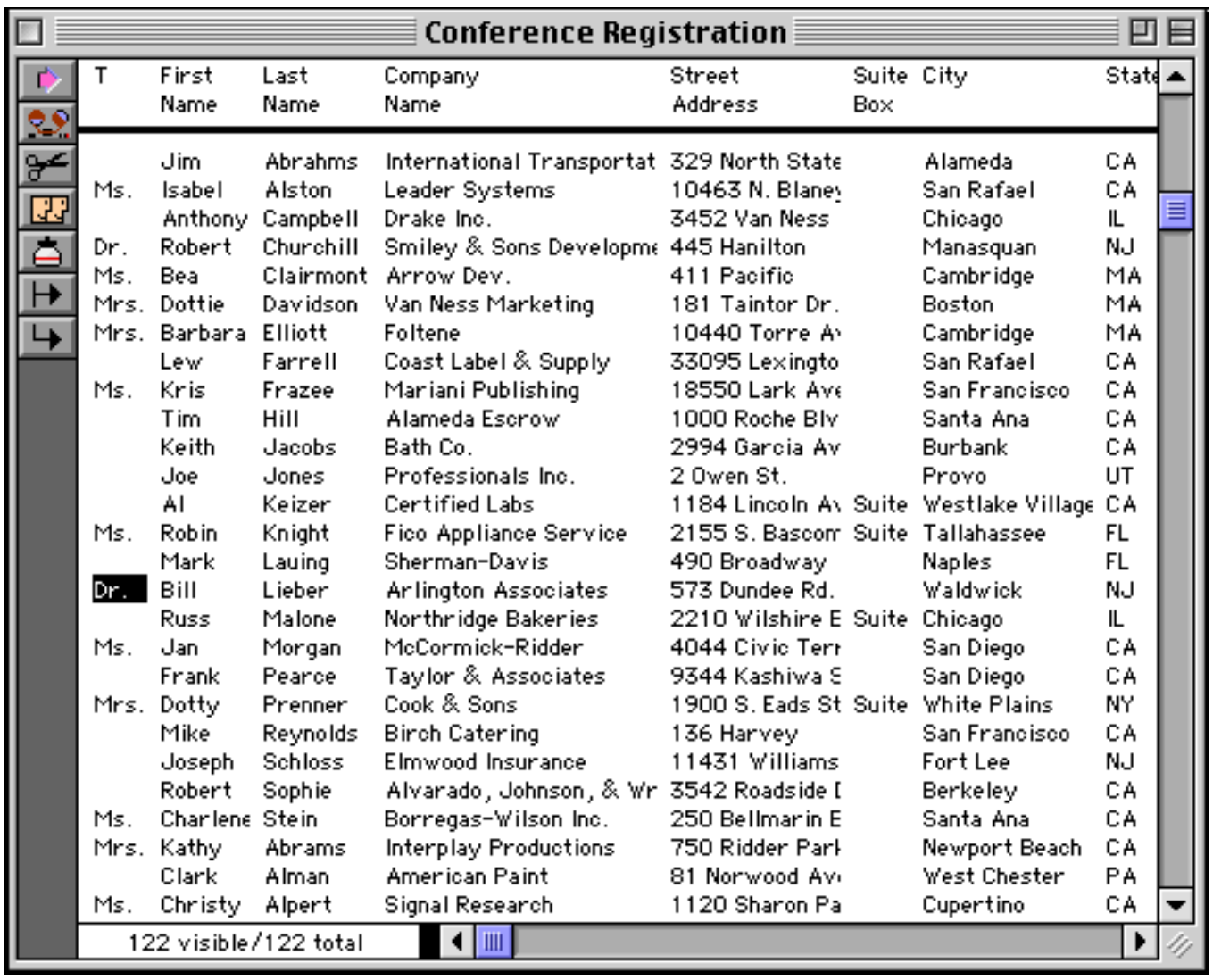
The first three finishers are assigned gold, silver, and bronze medals, with all of the other records left blank.



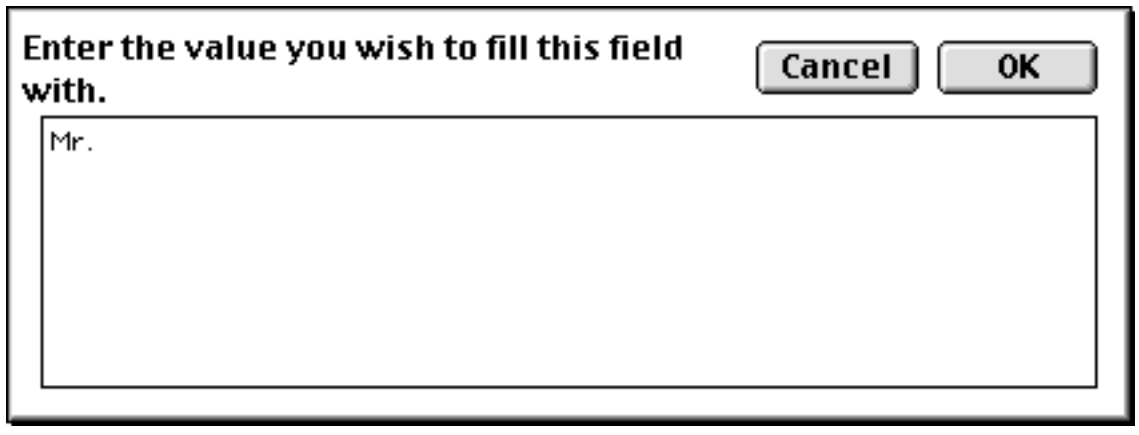
See "Text Arrays" on page 1236 for more information on the `array(` function used in this example.

Filling Empty Cells

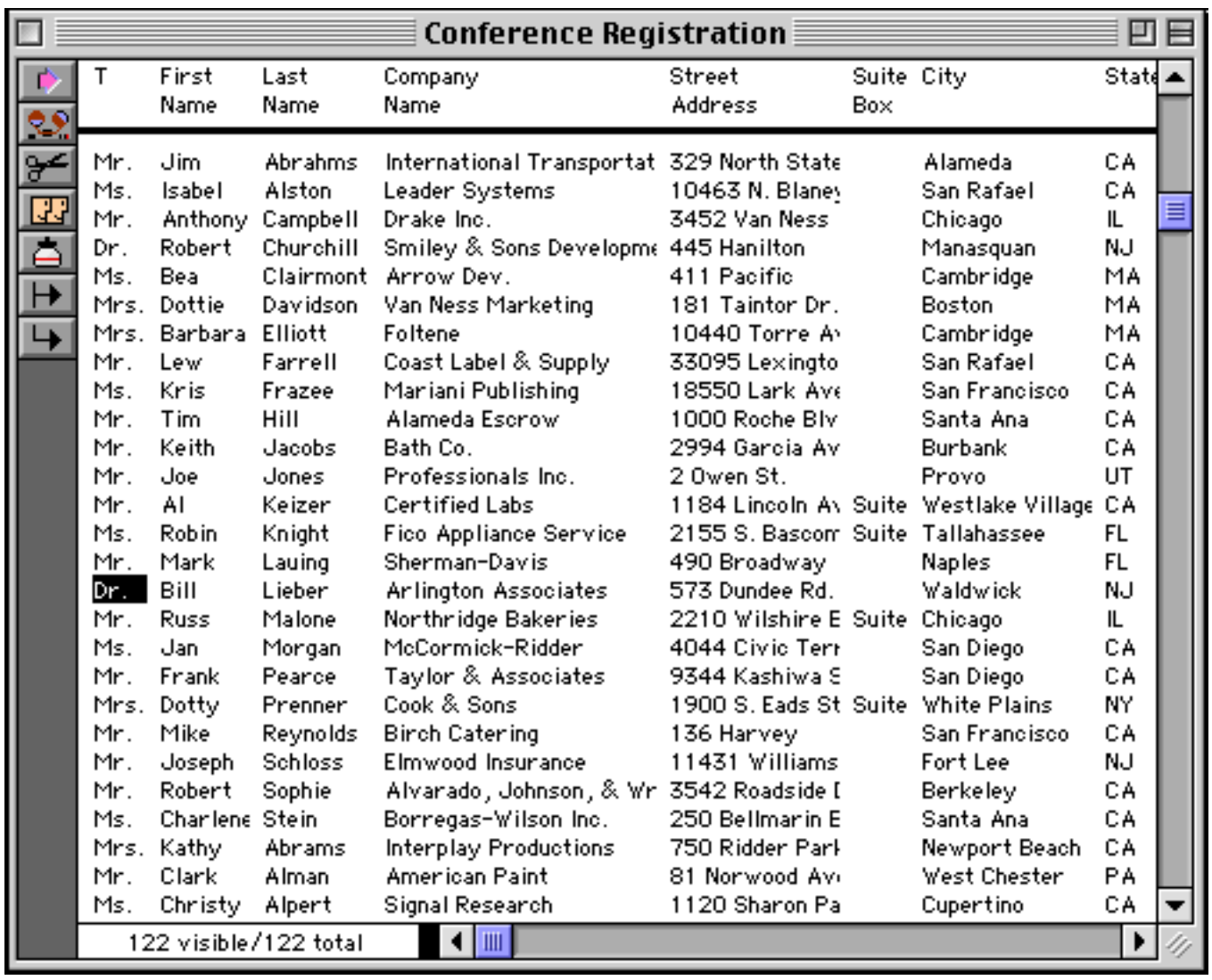
The **Empty Fill** command is very similar to the **Fill** command (see "Filling a Field with a Fixed Value" on page 452). However, the **Empty Fill** command will not destroy the data already in the field. In fact, **Empty Fill** will only fill cells that are completely empty. Here is a database where some of the name prefixes have been left blank.



Using the **Empty Fill** command these empty cells can quickly be filled with Mr.



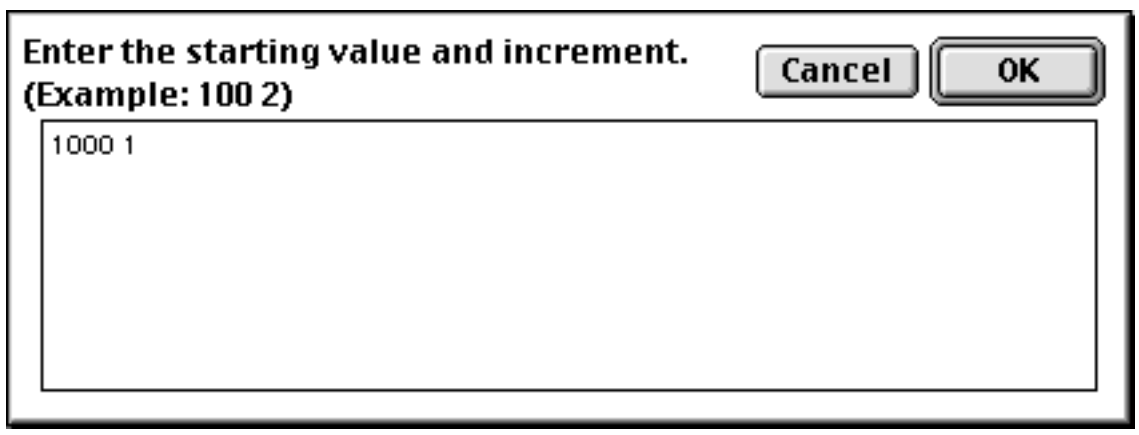
Here's the finished result.



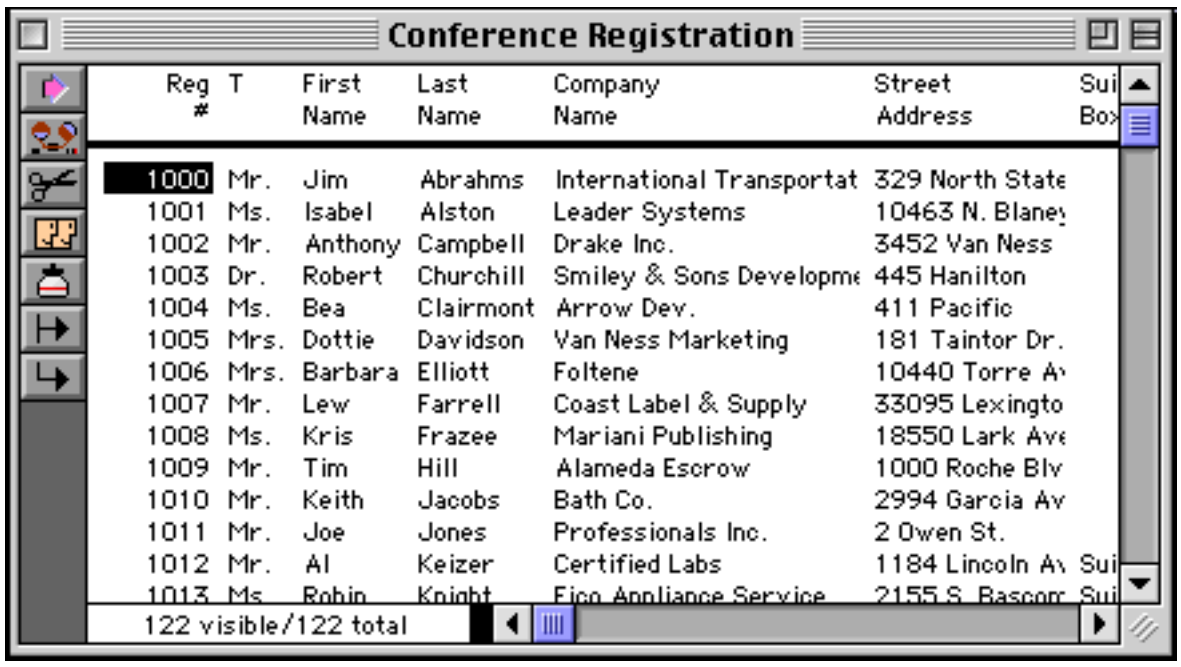
Automatic Numbering

The **Sequence** command fills the current field with a numeric sequence (for example 1, 2, 3 or 100, 110, 120). The **Sequence** command only works with numeric fields, you cannot sequence a text, date, or choice field.

To use the **Sequence** command, first click on the field you want to fill, then choose the **Sequence** command from the Math Menu. Type the first value in the sequence, then a space, then the increment value.



For example, type 1000 10 if you want to fill the field with the sequence 1000, 1010, 1020, etc. Press **OK** to actually fill the field. In this example the sequence number starts at 1000 and increments by one.



The sequence can start with any number and increase by any value, including non-integer values or negative values. The table below shows four examples of starting and increment values.

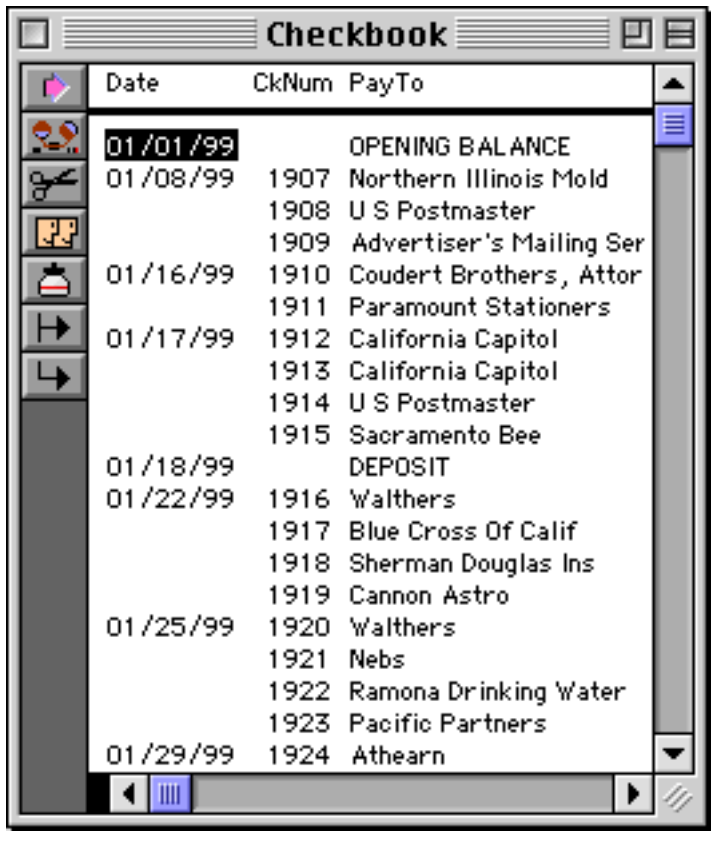
1 1	5 5	1 0.1	100 -1
1	5	1.0	100
2	10	1.1	99
3	15	1.2	98
4	20	1.3	97
5	25	1.4	96

If the database contains summary records, the sequence count will reset to one after each summary record. If you want to sequence the current field without restarting at summary records, use the **Formula Fill** command with the formula `seq()`. See “Summaries and Outlines” on page 393 for more information on summary records. See “Filling a Field with a Formula” on page 453 for more information on the **Formula Fill** command.

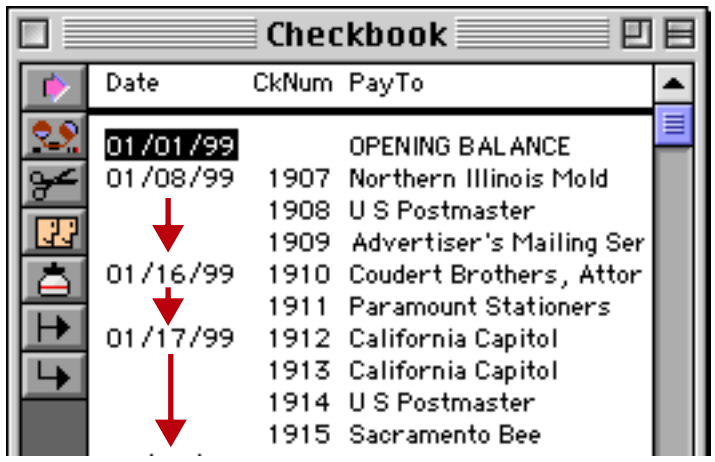
Propagate

Like **Empty Fill**, the **Propagate** command fills all the empty cells in the current field. However, instead of filling the empty cells with a fixed value, the **Propagate** command propagates filled data cells into the empty data cells (if any) below them.

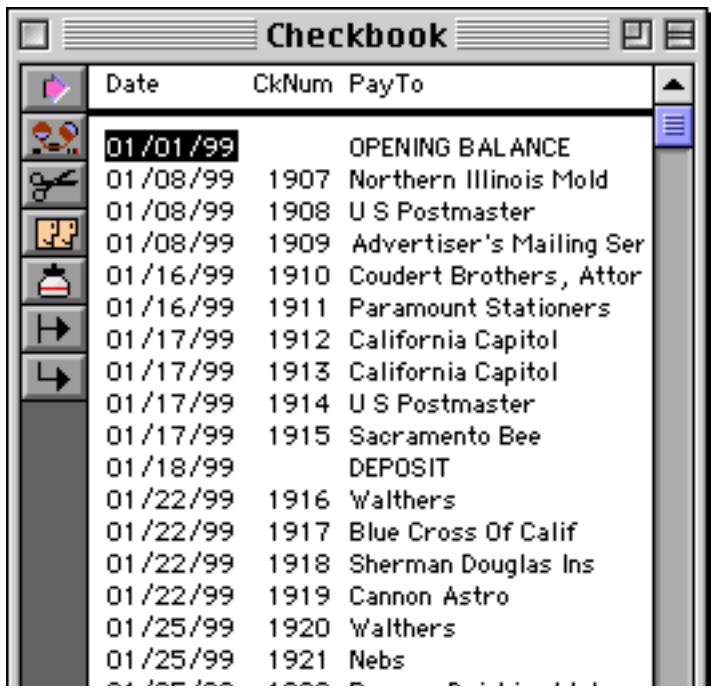
To illustrate, here is a database where the date was only entered for the first check written each day. For example, checks 1907, 1908 and 1909 were all written on January 8th, but the date has only been filled in for check 1907.



The **Propagate** command will fill in the empty cells, as shown by the arrows in this illustration.

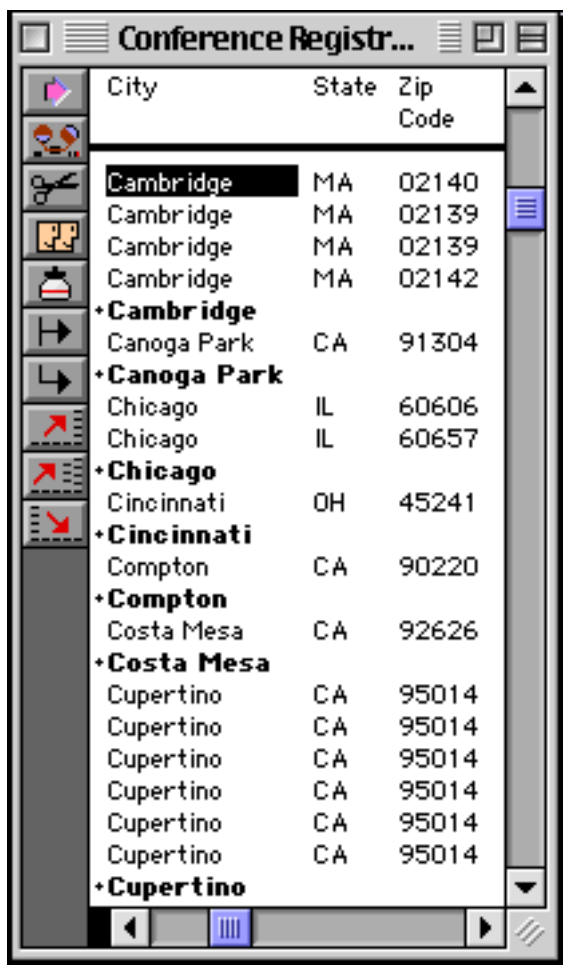


Here is the actual result after the **Propagate** has completed.

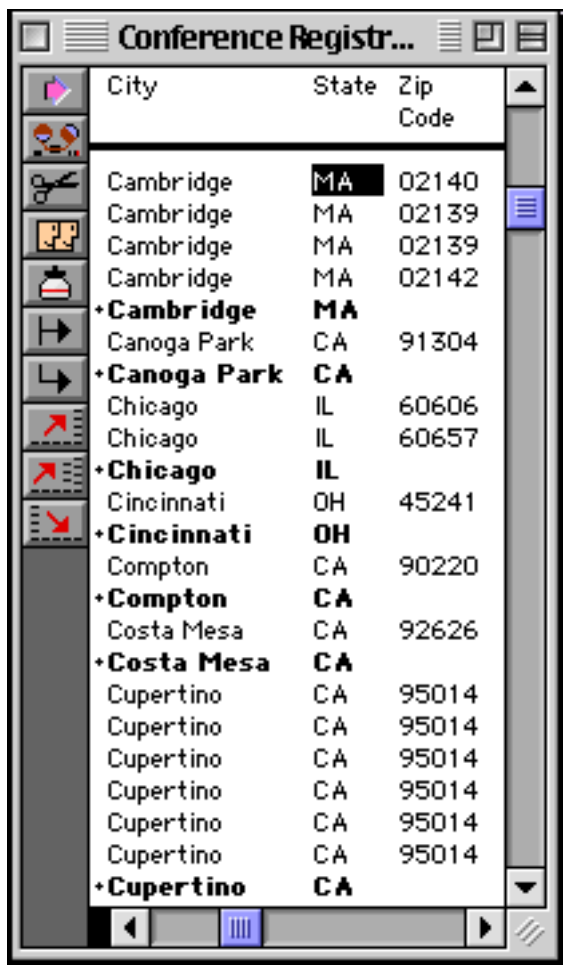


The **Propagate Up** command performs the same operation upside down, propagating filled data cells into the empty data cells above them.

The **Propagate** command can be used to copy information into newly created summary records. The **Group Up** command creates summary records but leaves all but one field blank. (see "STEP 1 - GROUP" on page 399 for more information on the **Group Up** command.) Use the **Propagate** command to copy other fields from the data records into the summary records. In this example the database has been grouped by city, but not by state. This means that the State field in each summary record is blank.



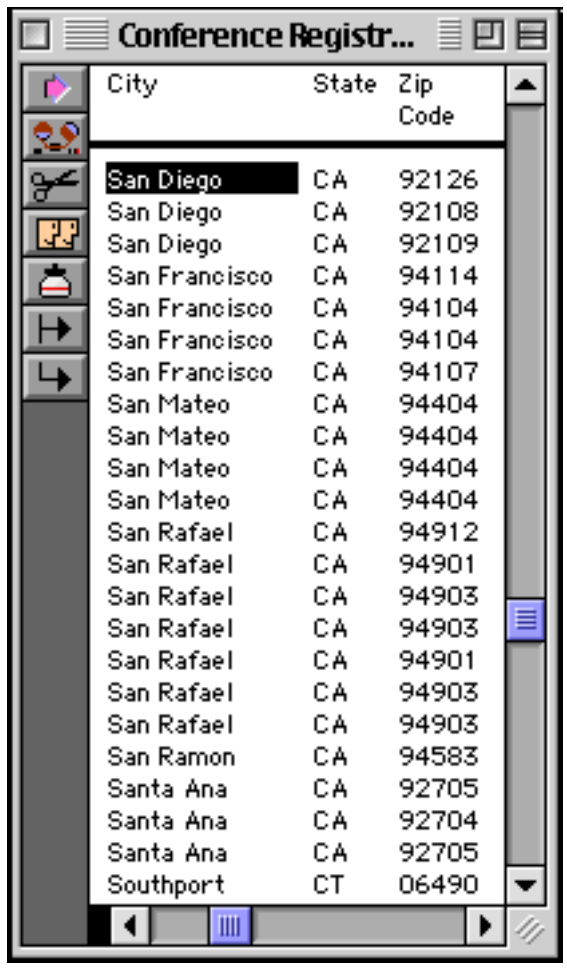
To fill in the state, click on the state field and choose the **Propagate** command.



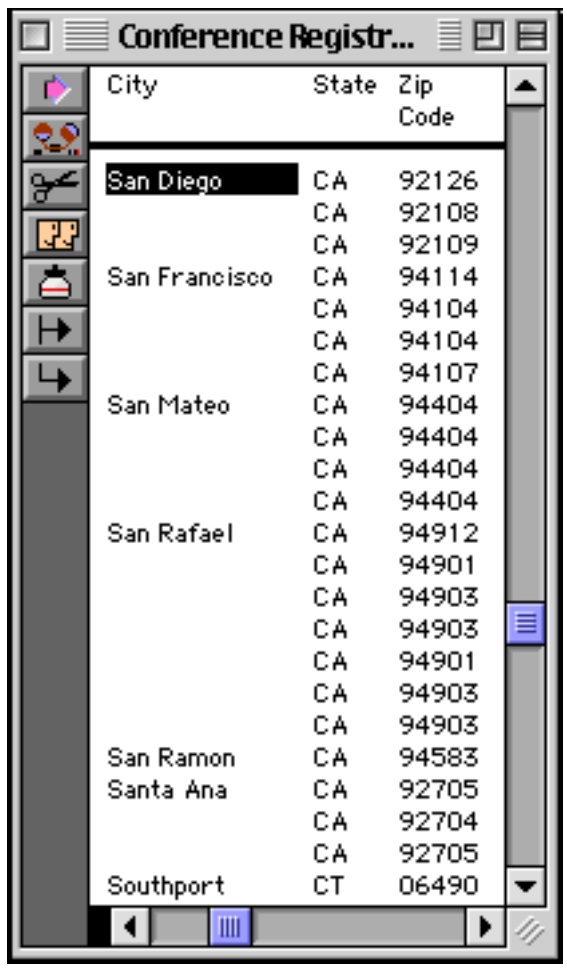
You could repeat this process to fill in the zip code. (However, this may not generate the results you want since some cities have multiple zip codes, as shown by Cambridge and Chicago in this example.)

UnPropagate

This command performs the exact inverse of the **Propagate** command. If the same value appears in two or more consecutive data cells, the **Unpropagate** command empties the second and subsequent data cells. Here is a database that has been sorted by city.



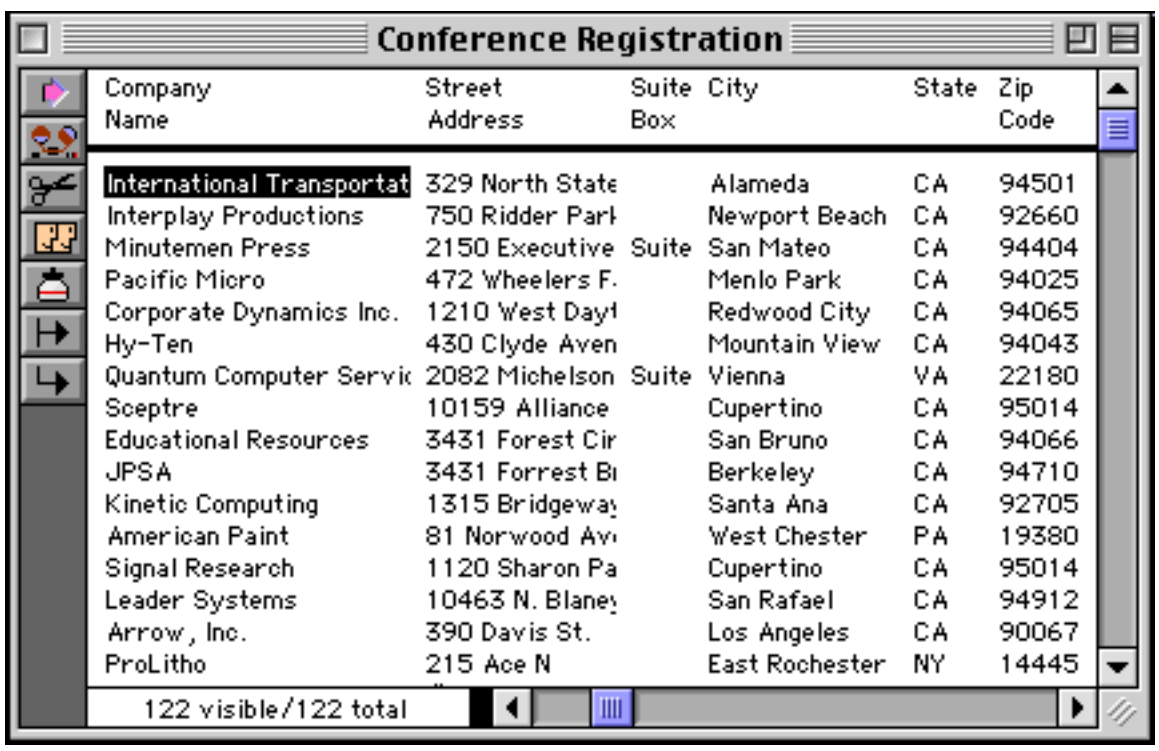
The **Unpropagate** command eliminates all but the first entry for each city.



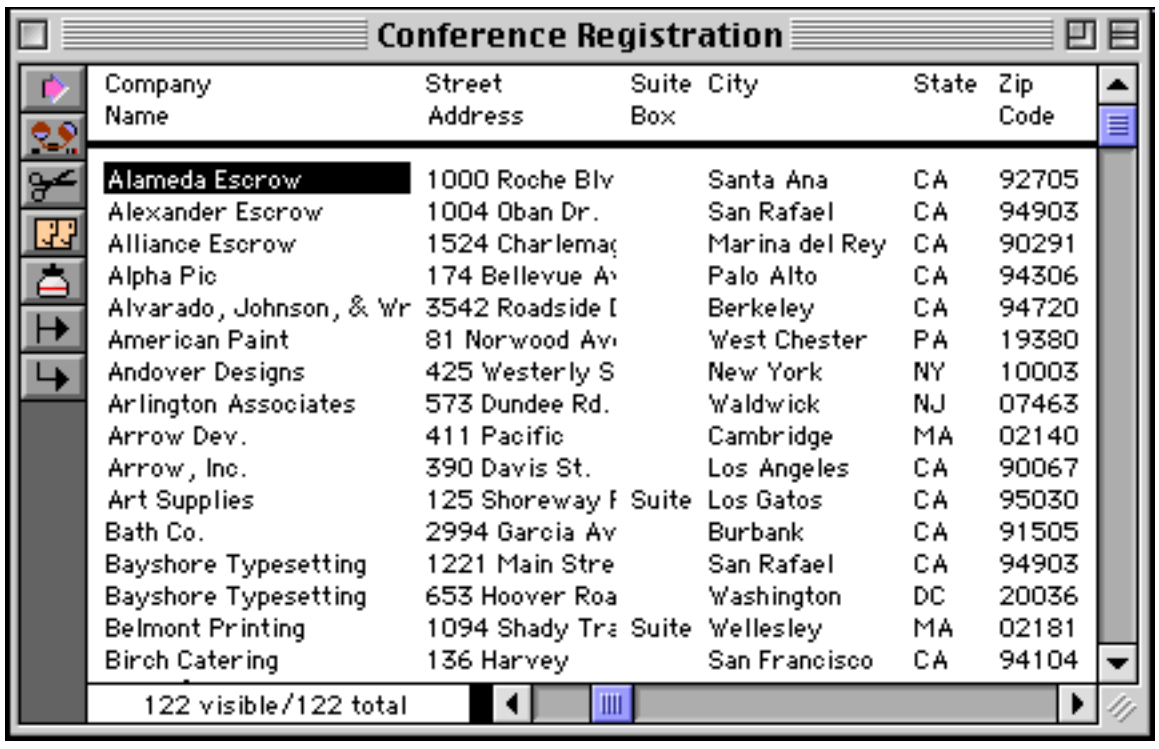
The **Unpropagate Up** command performs the same operation upside down, leaving the last of several duplicate values while clearing the others.

Using UnPropagate to Eliminate Duplicates

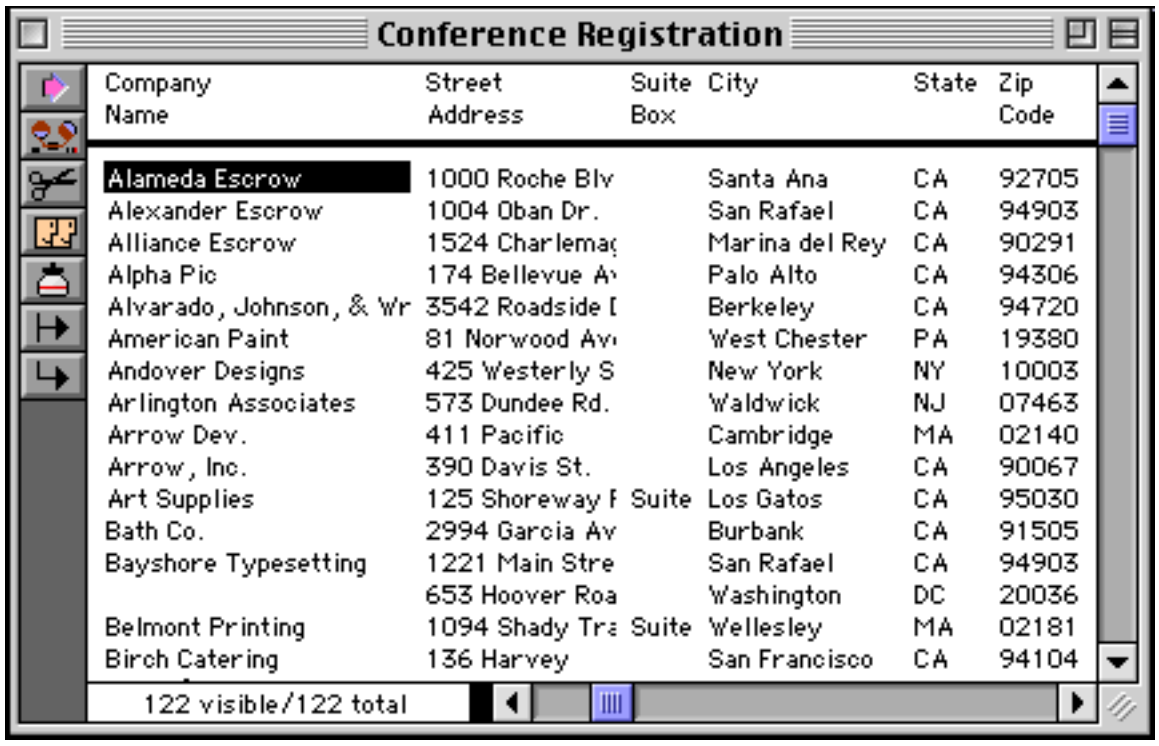
The **UnPropagate** command can be used to eliminate duplicate values in a database. The first step is to click on the field that contains the potentially duplicate values, for example Name or Company. If you want to eliminate duplicates over multiple fields (for example an entire address) you must create a new field and use the **Formula Fill** command to combine the data into a single field.



The next step is **SortUp** the database. This brings all the duplicate values together. For example, there are two Bayshore Typesetting entries in this database.



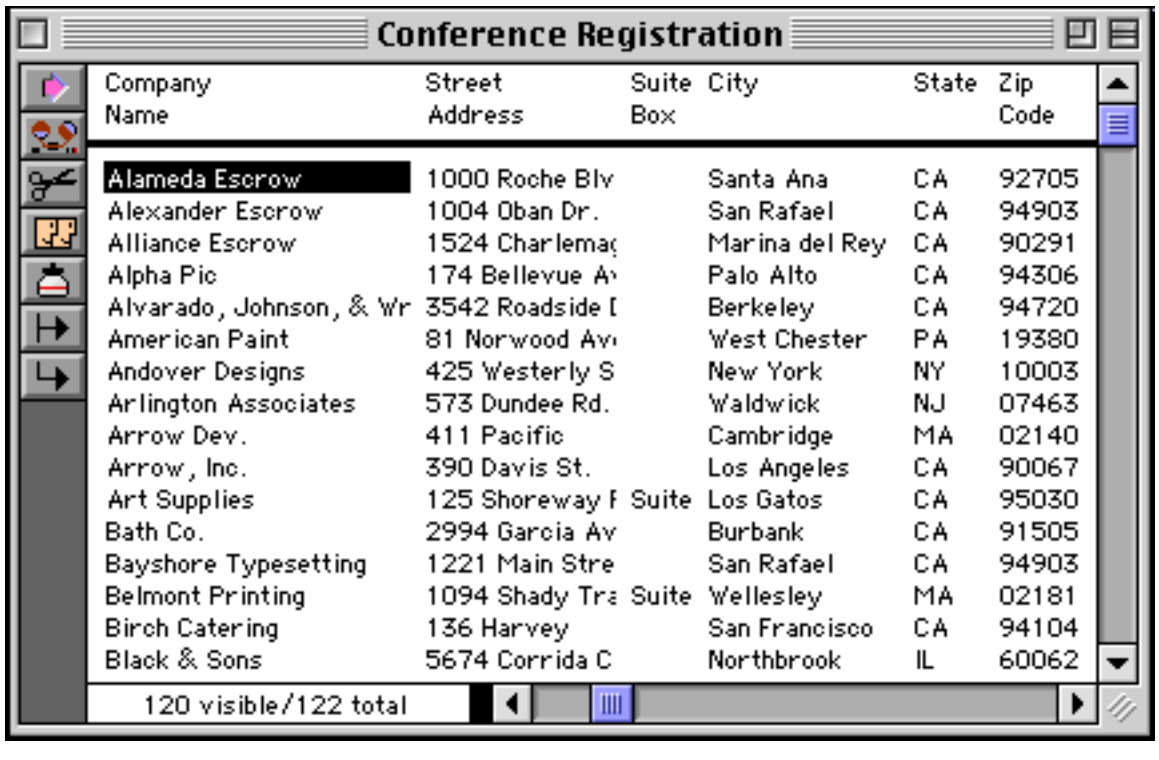
The next step is the **UnPropagate** command. Wherever a duplicate value appears in the data cell, the **UnPropagate** command clears the cell.



Now use the **Find/Select** command to select the non-empty data cells. Just pick Not Equal from the pop-up menu.



All of the duplicate records will disappear when you press the **Select** button. In this database there were two duplicate companies, so there are now 120 selected (non-duplicate) records.



The final step is to permanently remove the duplicate records with the **Remove Unselected** command.

It's possible to create a procedure that will automatically perform all of these steps for you. This procedure will remove all of the duplicate entries in the current field.

```
sortup
unpropagate
select «» <> ""
removeunselected
```

Tip: One possible problem with this technique is that all cells that start out empty will be removed. For example if you are removing duplicate company names but some records don't contain company names, the records without company names will be removed. To fix this problem, use the **Empty Fill** command to fill the empty names with a unique value like n/a before you start, then use **Find/Select** to select all values not equal (≠ or) to n/a. Then perform the rest of the steps listed above. Here is a revised version of the procedure that takes care of this problem.

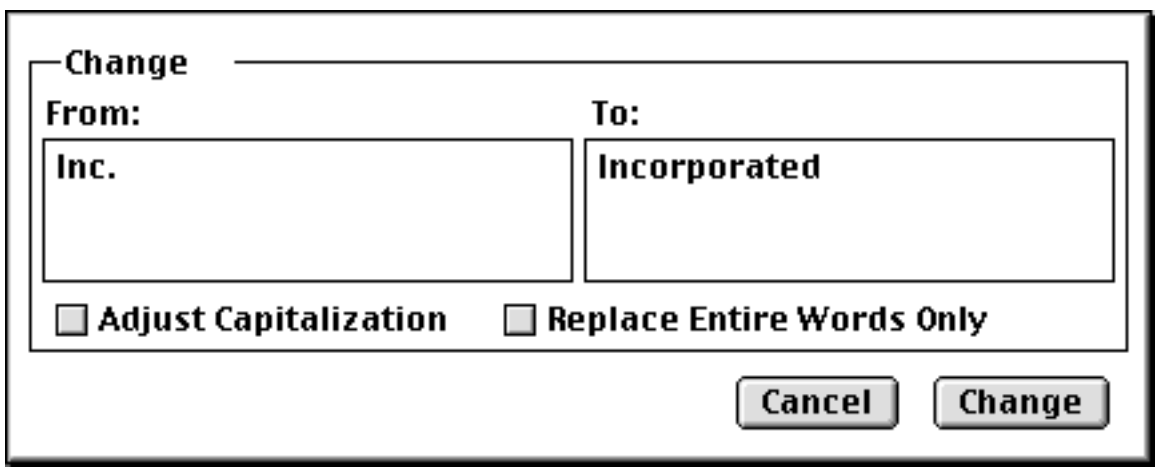
```
emptyfill "!empty!"
select «» <> "!empty!"
sortup
unpropagate
select «» <> ""
removeunselected
formulafill ?(«» = "!empty!" , "" , «»
```

Warning: Keep in mind that all of these techniques will blindly remove all but the first duplicate entry. In this example, there were two entries for Bayshore Typesetting. However, they were probably not really duplicates, since one was in Washington, DC and the other in San Rafael, CA. There is no way for an automatic technique like this to know which of these is really correct, or even if they are really duplicates at all. If you want to manually examine duplicate records instead of blindly deleting them, use the **Select Duplicates** command in the Search Menu. See "Select Duplicates" on page 388 for more information on this command.

Change (Find and Replace)

The **Change** command (in the Search menu) finds and replaces a word or phrase in the current field. For example, you can use the **Change** command to replace every occurrence of Inc. to Incorporated, or every occurrence of Purchase Order to P.O.

The **Change** dialog allows you to specify the original (From) and the new (To) word or phrase.



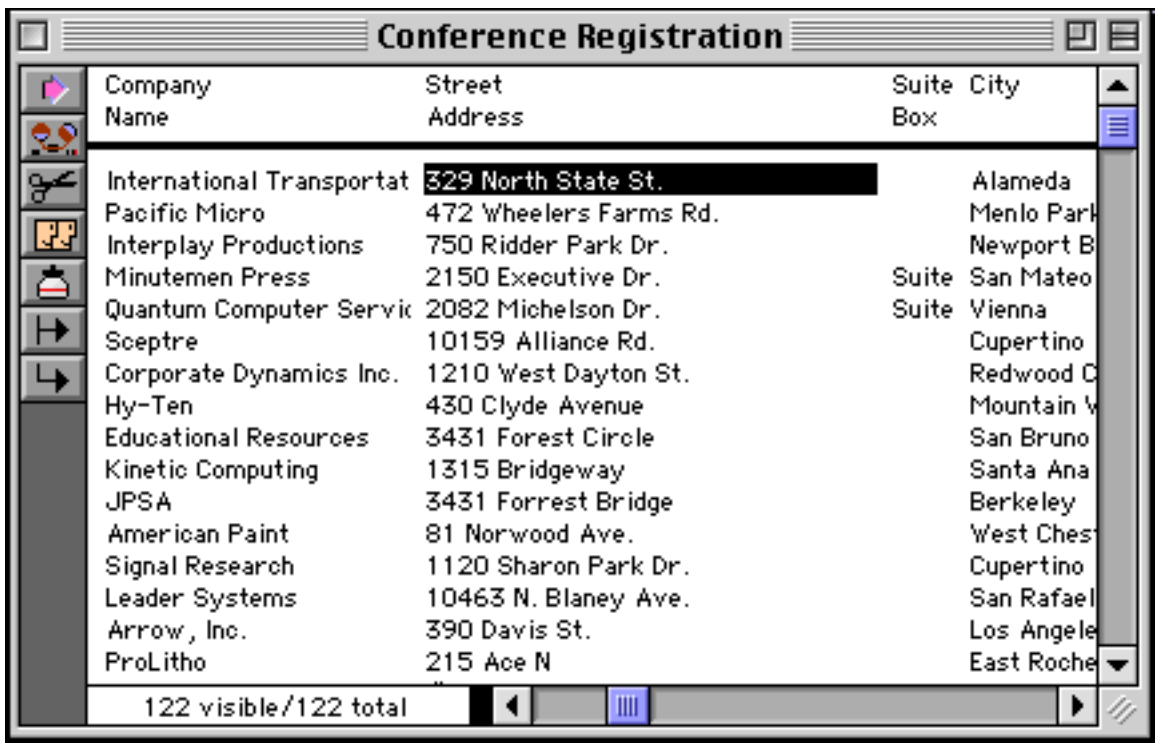
The Adjust Capitalization option allows you to specify whether you want capitalization to be adjusted as the word or phrase is replaced. If you check this option, Panorama will automatically adjust the capitalization of the new word or phrase as it is inserted into the database. If you leave this option off, capitalization is not adjusted. In fact, if the Adjust Capitalization option is off, only words or phrases that exactly match the capitalization typed into the dialog will be replaced. The table below shows the result of replacing Inc. with Incorporated with Adjust Capitalization both off and on.

Original	Adjust Capitalization OFF	Adjust Capitalization ON
Inc.	Incorporated	Incorporated
INC.	INC.	INCORPORATED
inc.	inc.	incorporated

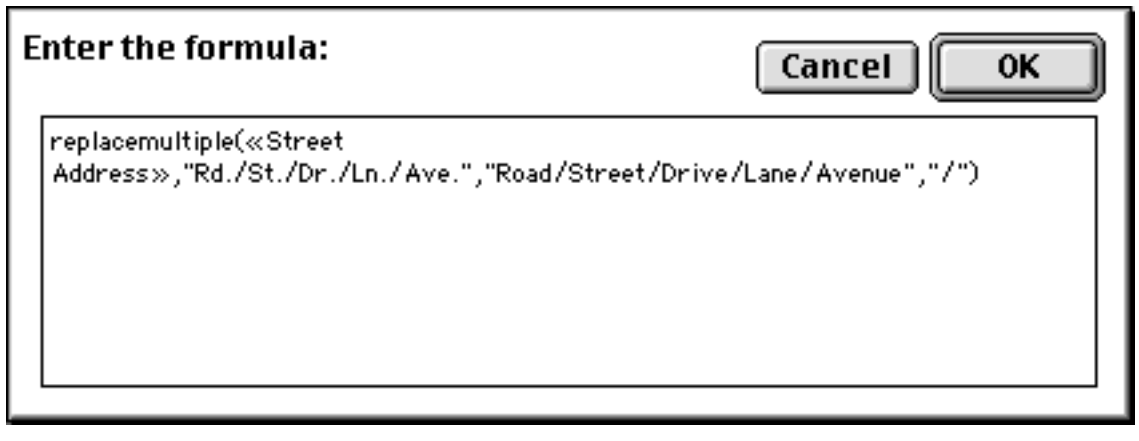
The Replace Entire Words Only option tells Panorama to replace only entire words, not sections of words. For example, if you ask Panorama to change is to was, it will also change this to thwas. This is, of course, wrong. To prevent this, just check the Replace Entire Words Only option.

Changing with the Replace(Function

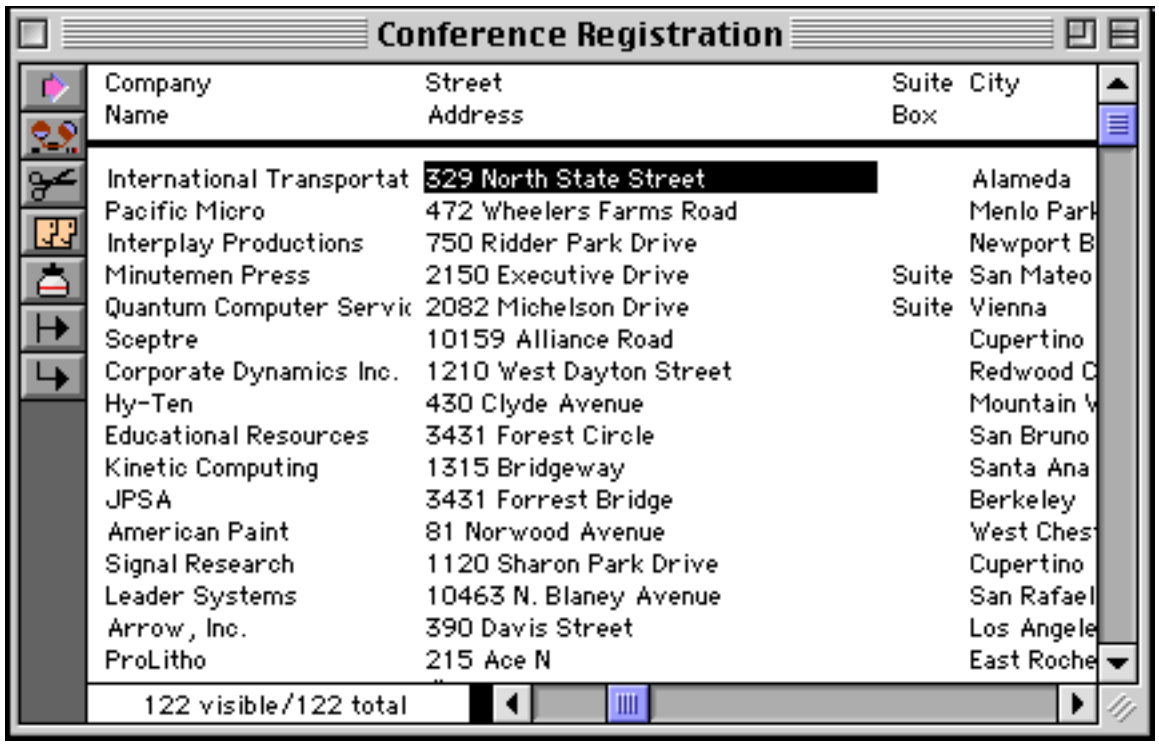
The **Change** command is not the only way to replace words or phrases. You can also use the **Formula Fill** command and the `replace(` or `replacemultiple(` functions (see "String Modification Functions" on page 1224). This technique is especially handy if you need to replace several words or phrases at once. For example, consider the addresses in the database below.



Suppose you wanted to expand the abbreviations in these addresses: St. to Street, Dr. to Drive, etc. You could do this by using the **Change** command over and over again. Or you can simply use the `replacemultiple(` function to replace all of the abbreviations in one fell swoop.



Press **OK** to replace all of the abbreviations at once:

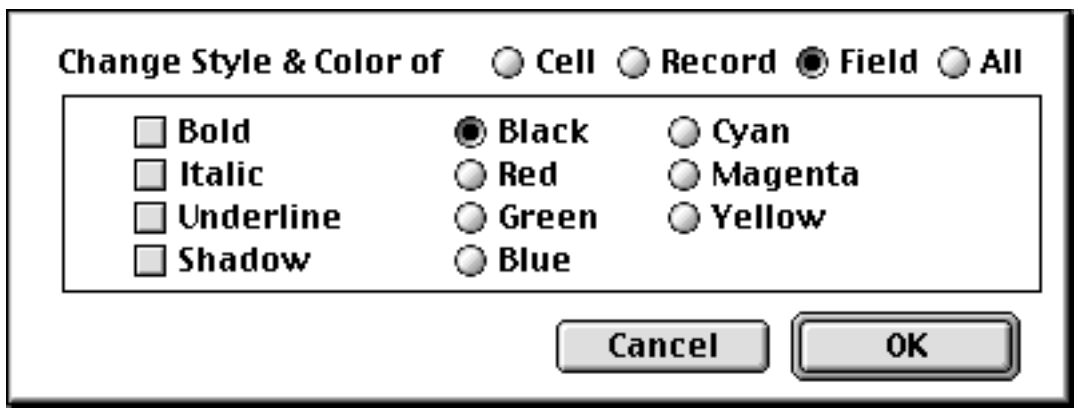


See “Filling a Field with a Formula” on page 453 for more information on Formula Fill.

Data Style and Color

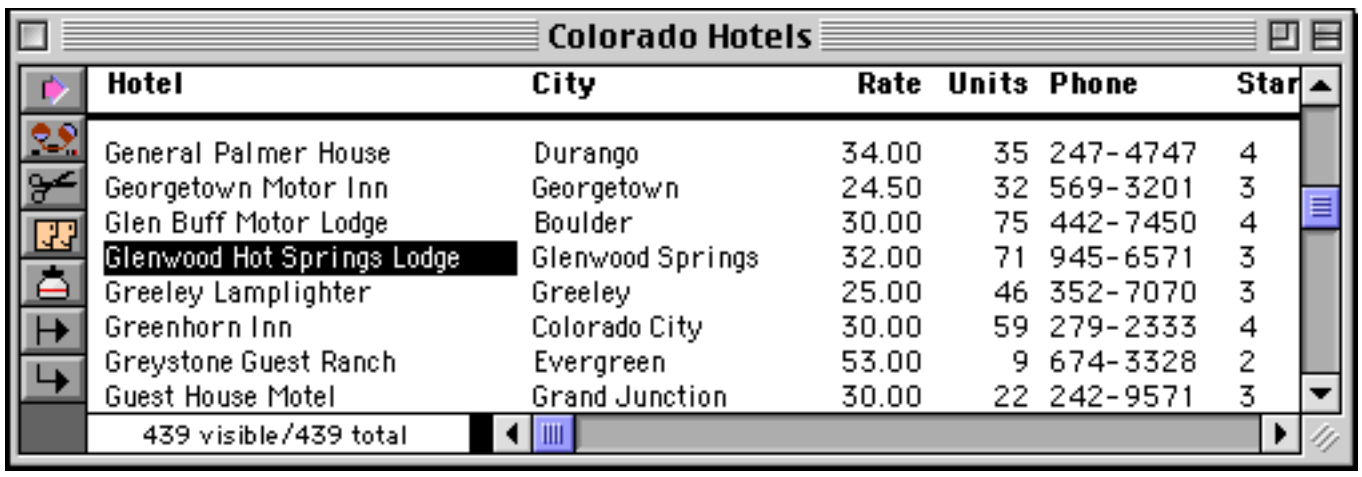
In addition to the data stored in each cell, Panorama also keeps track of the style (plain, bold, italic, etc.) and (to a limited extent) color (red, green, etc.) of each cell. Use the **Style & Color** command (Math menu) to change the style or color of one or more data cells.

The **Style & Color** dialog allows you to specify what cells to change and what style or color to use.

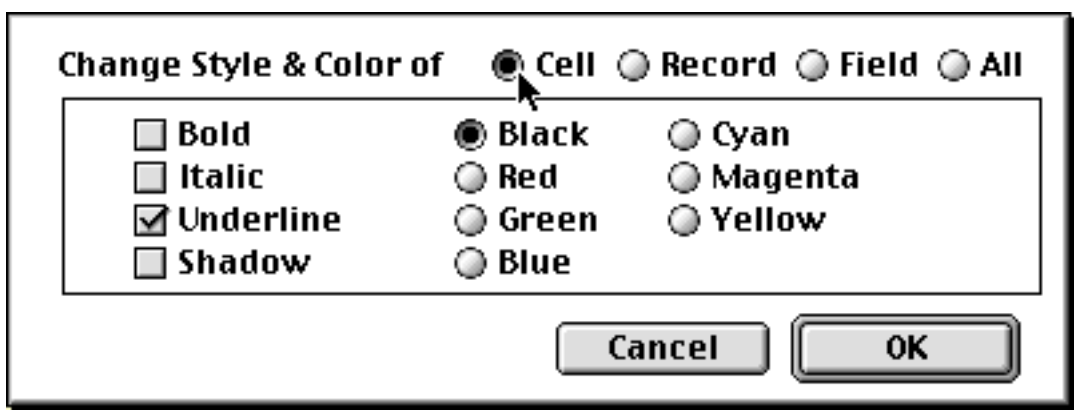


Choose Cell to change the style or color of the current cell, Record to change the style or color of all the cells in the current record, Field to change the style or color of all the selected cells in the current field, or All to change the style or color of every selected data cell in every field.

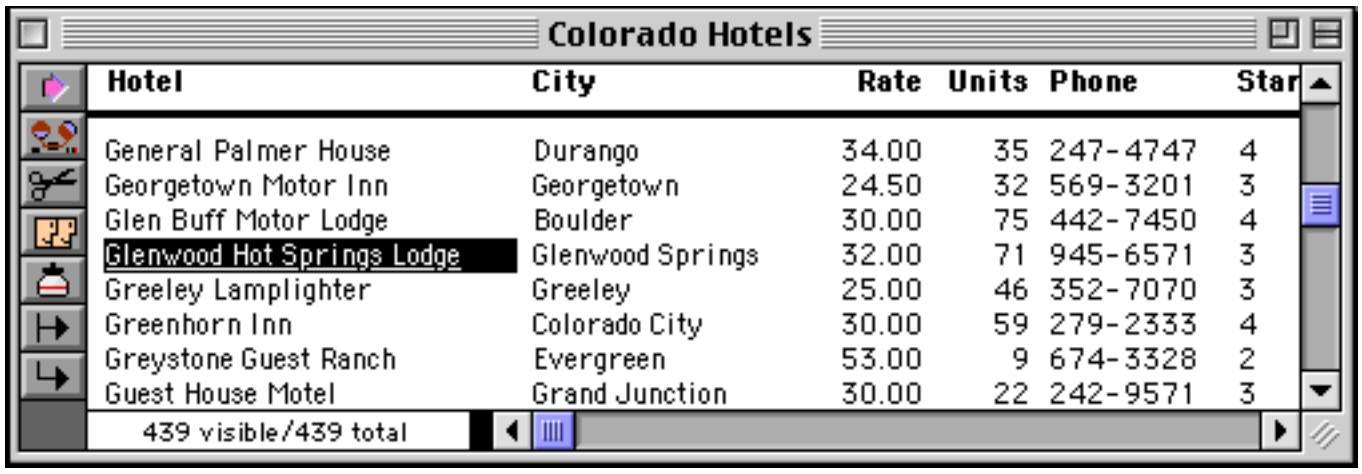
Let's see how to underline an individual cell. Start by clicking on the cell you want to underline.



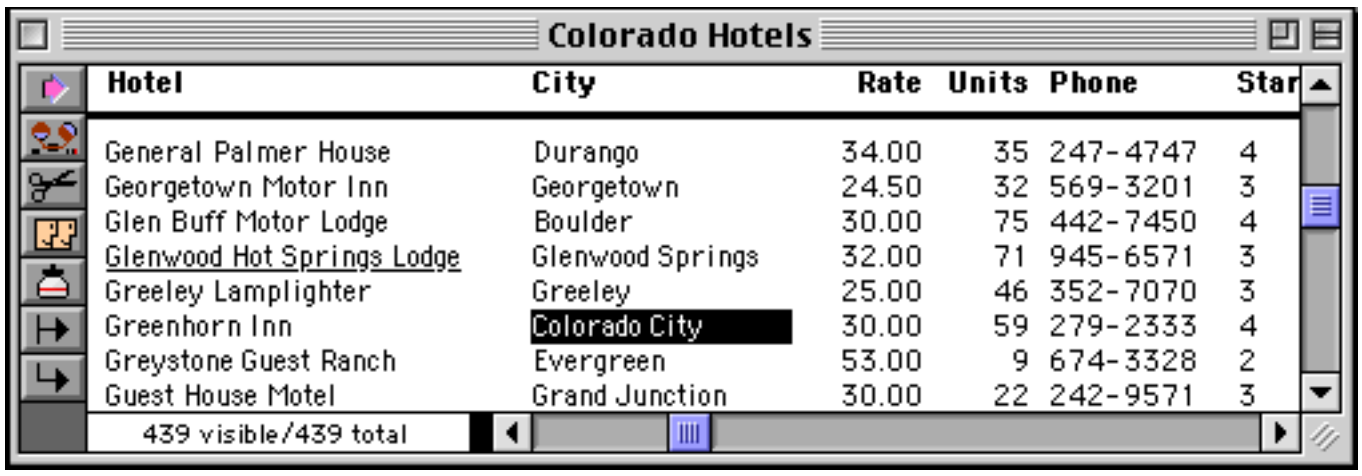
Now choose the **Style & Color** command, and click on Cell and Underline.



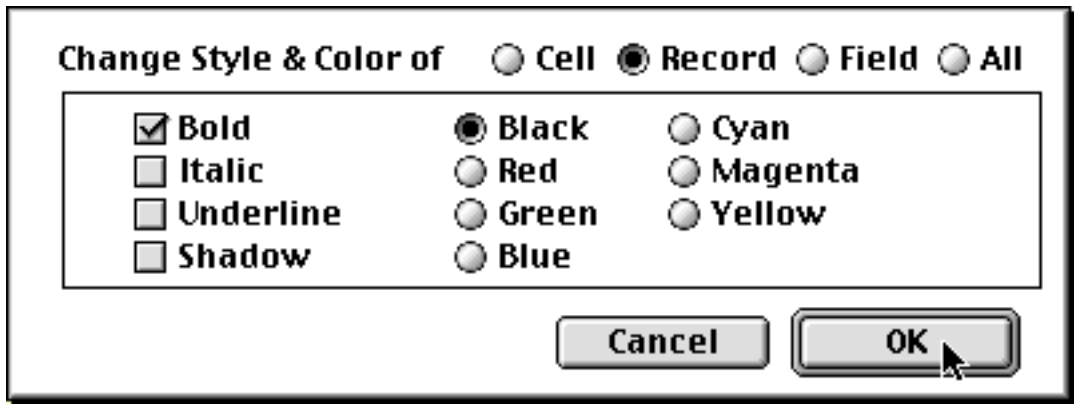
When you press **OK**, the cell will be underlined.



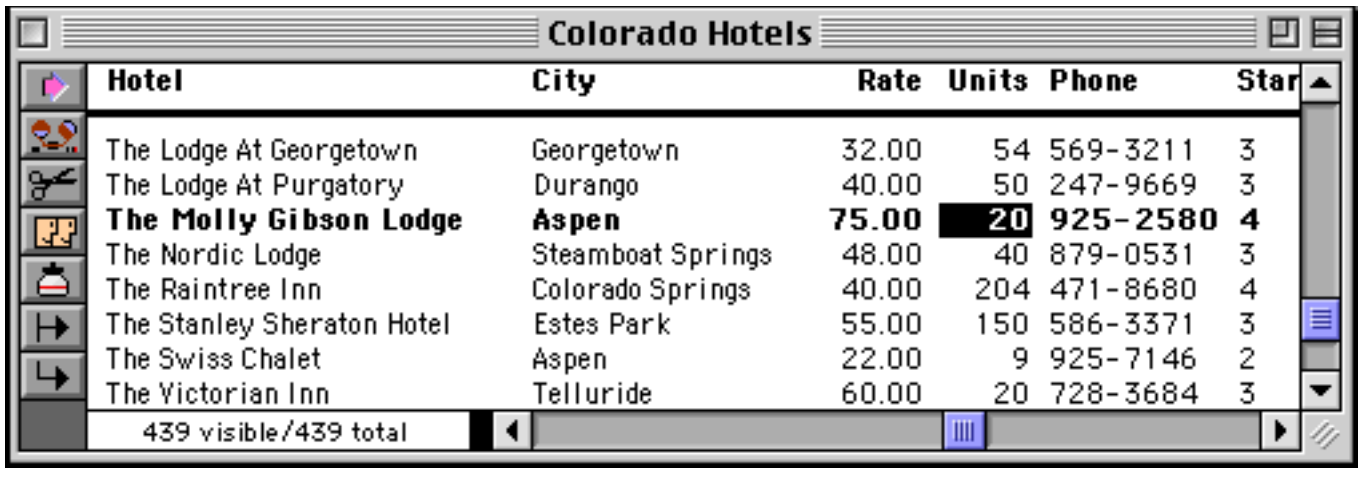
It's easier to see the underline if you click on another cell.



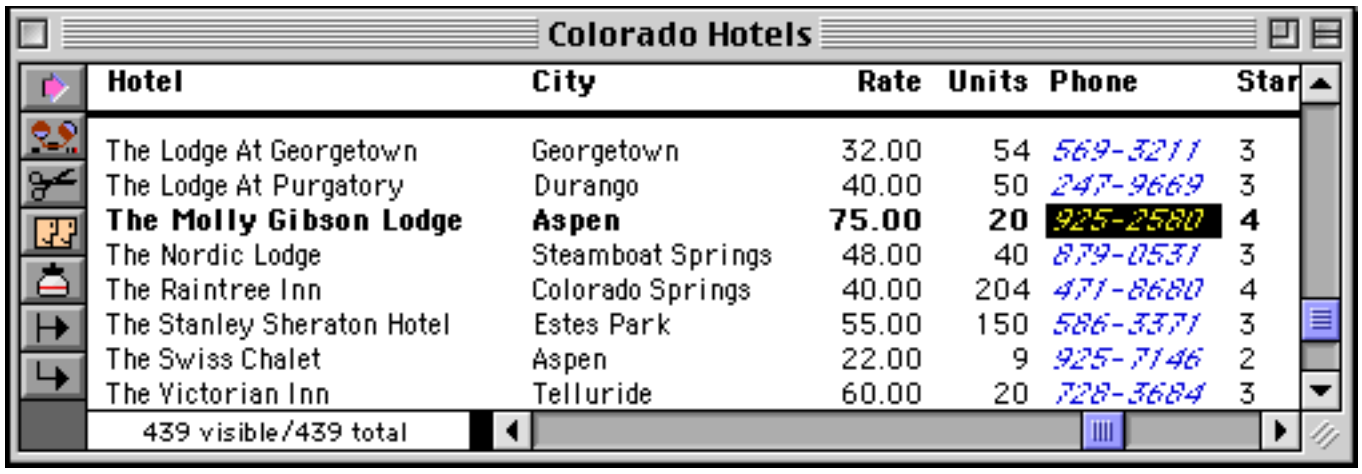
Using an almost identical process we can make an entire line bold. In the **Style & Color** dialog, choose Record and Bold.



Press **OK** to make the record bold.

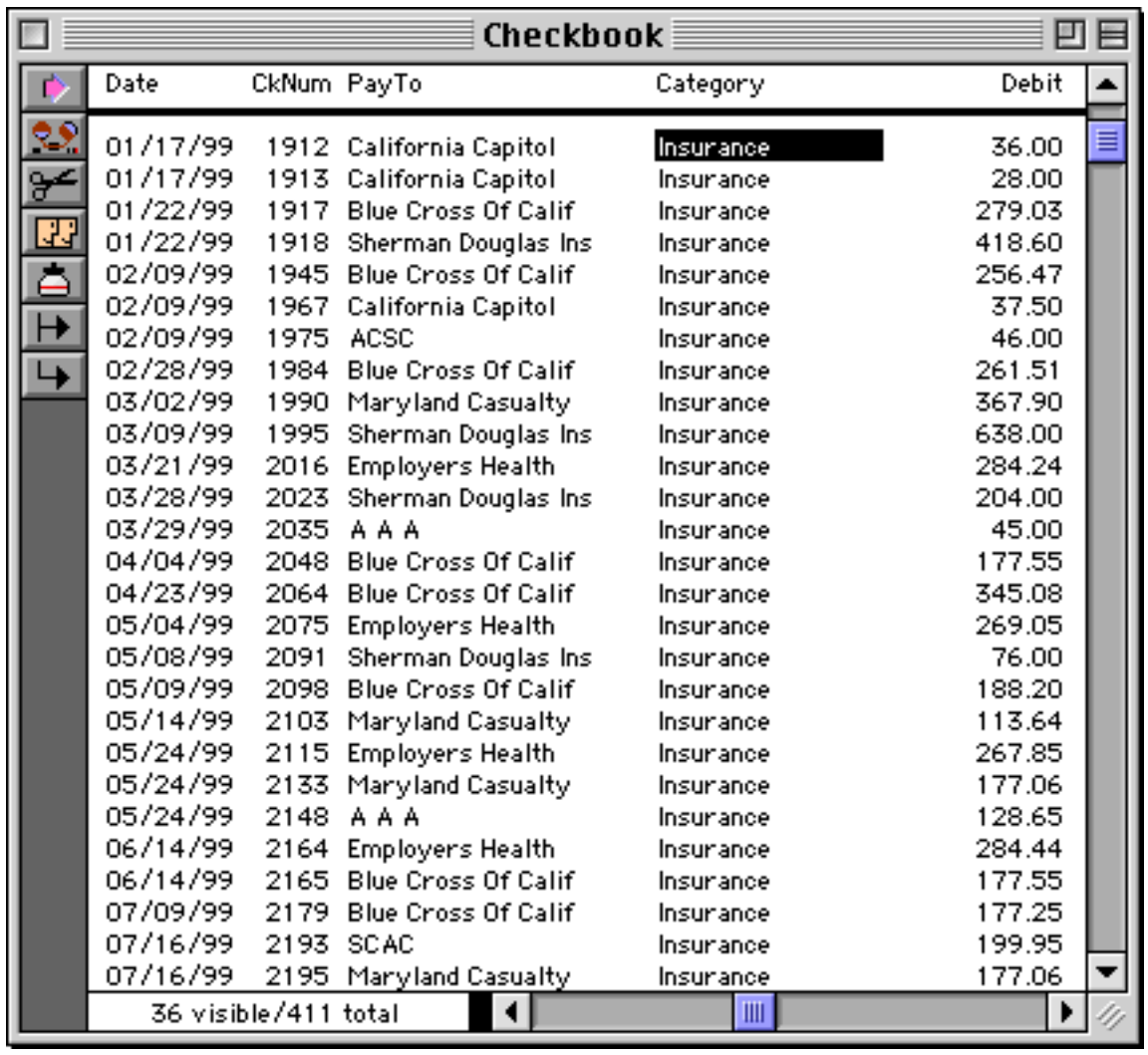


By choosing Field, Blue and Italic we can make all Phone Numbers appear in italic blue, as shown here.

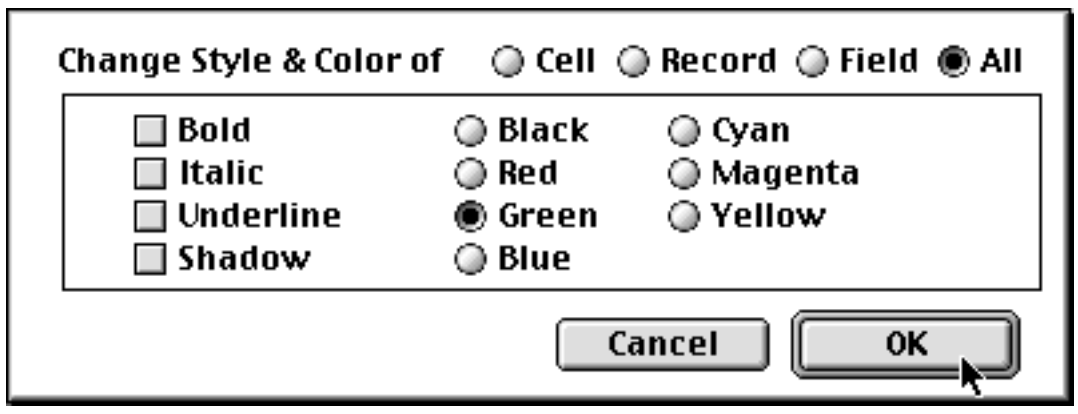


Notice that the italic blue has overridden the bold applied in the previous example.

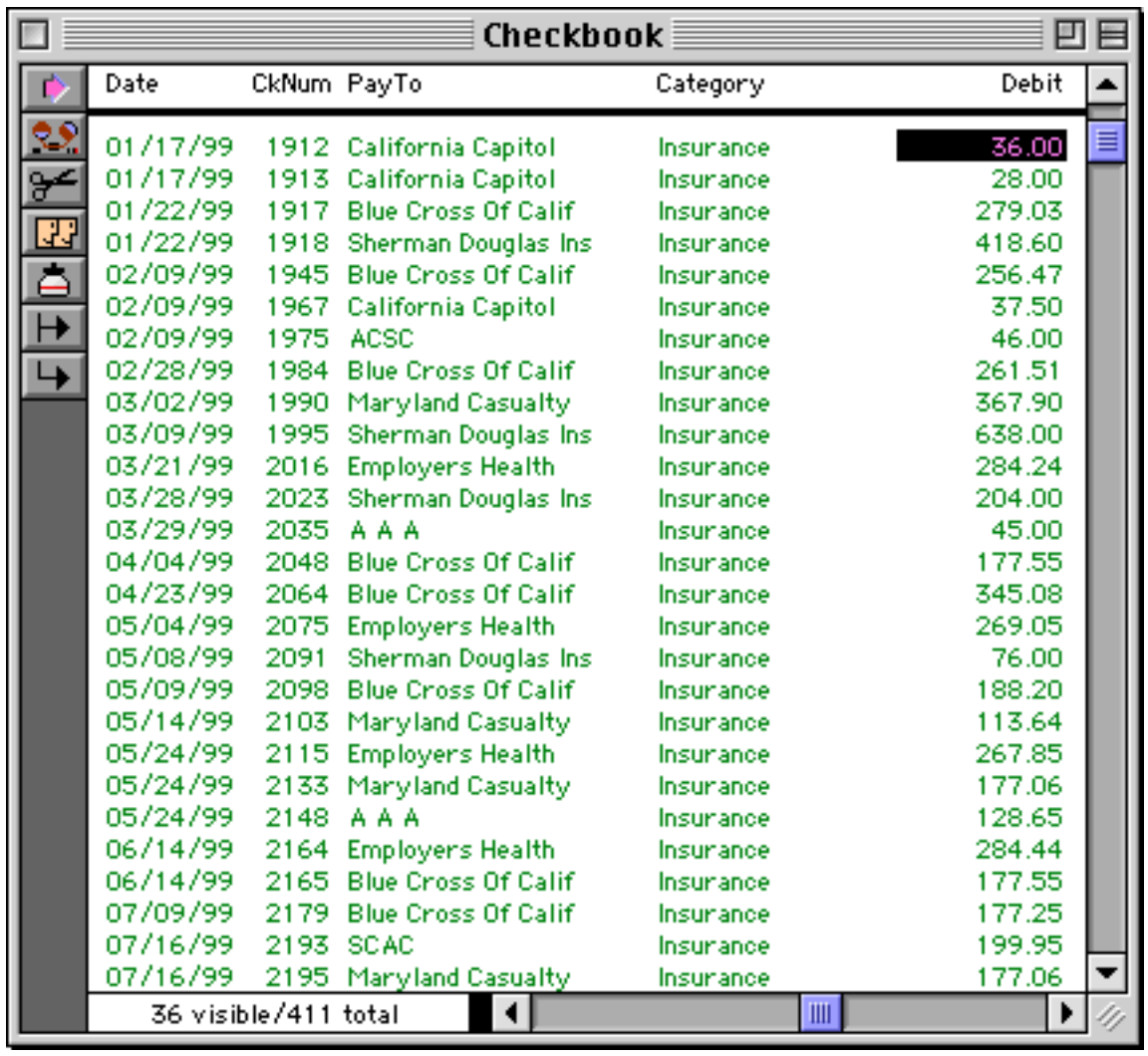
For our final example we will go to a checkbook database and mark all insurance payments in green. Start by selecting Insurance from the Category field. (See "Select" on page 375 for more information on the **Find/Select** command.)



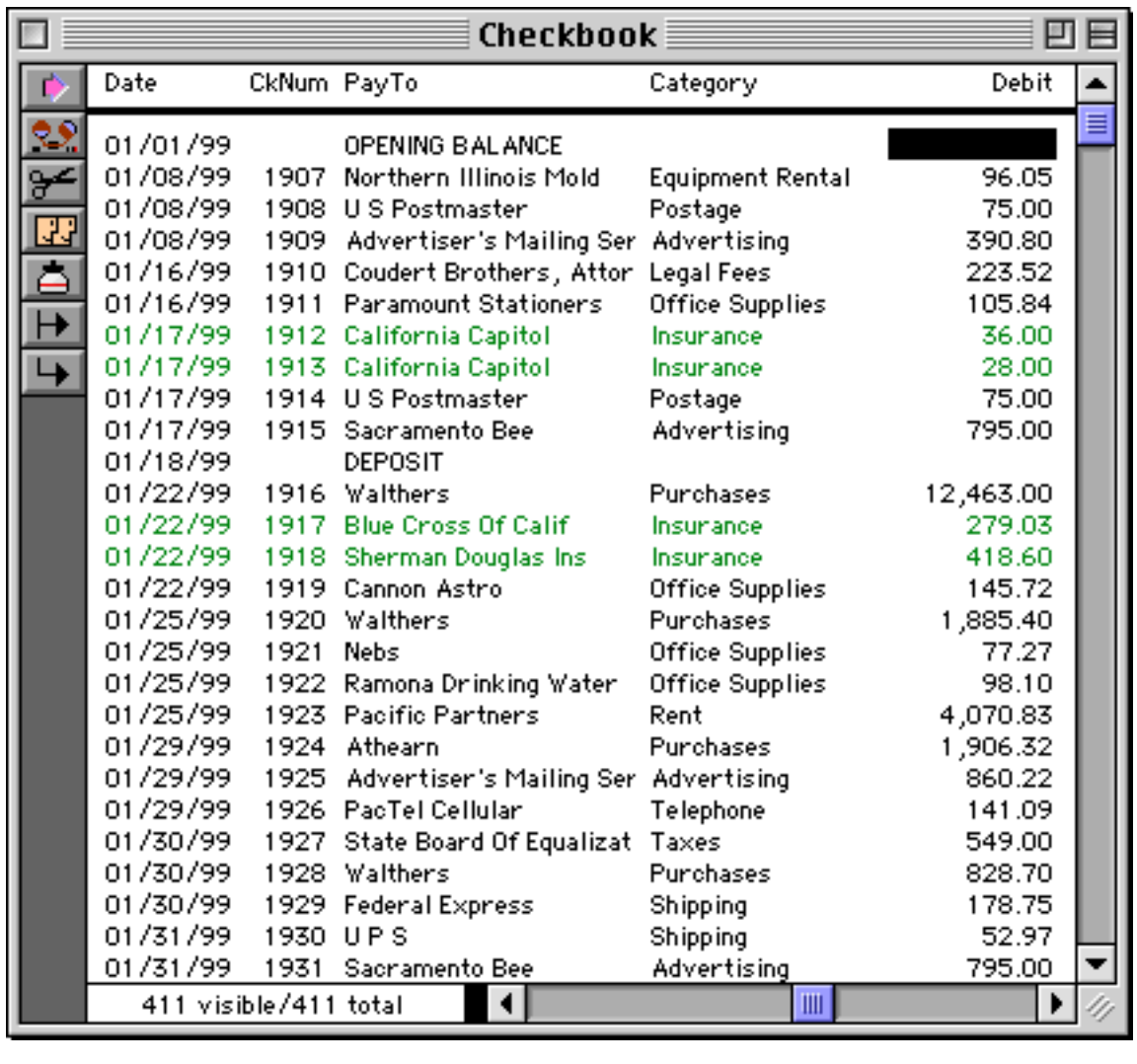
Now click on the Debit field, and choose the **Style & Color** command. Click on All and Green.



When you press **OK** everything visible will turn green.



Use the **Select All** command to see all of the records. The green insurance records are mixed in with the others.



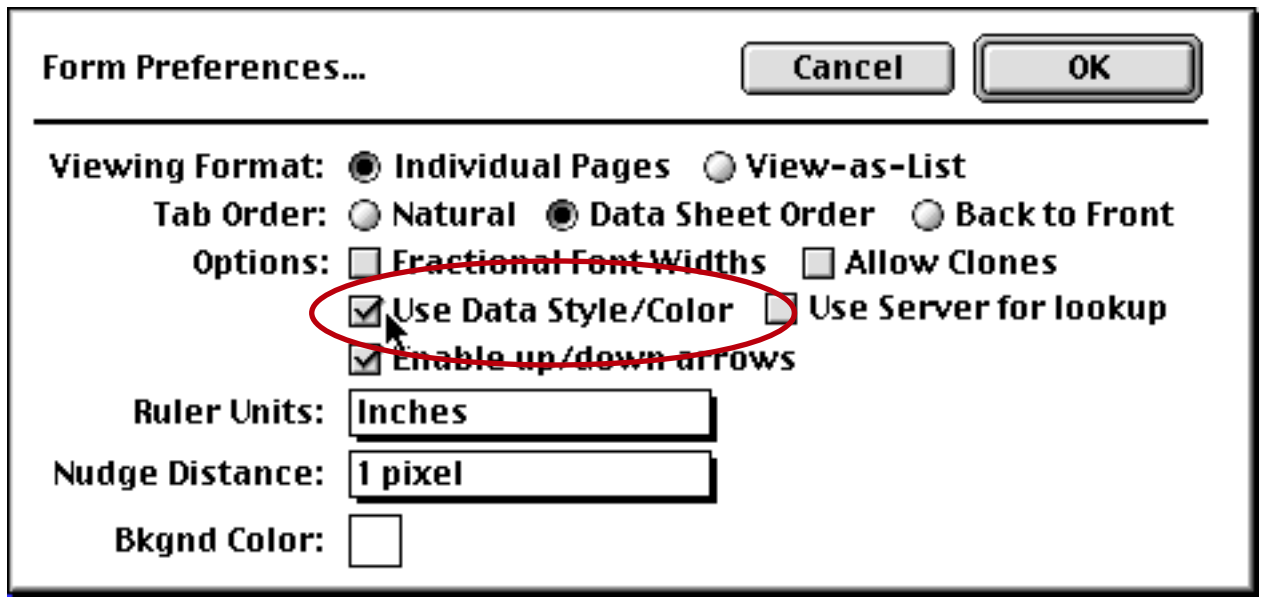
A cell retains its style and color until the data is modified. Any data modification (editing, formula fill, etc.) will cause the cell to revert to plain black.

Every data cell that is not plain black takes up an extra byte of storage. For example a database with 10 fields and 500 records will expand by 5K bytes if you change every data cell to blue or italic (or both).

Displaying Data Style and Color in Forms

Panorama always keeps track of the style and color of every data cell. However, Panorama does not always display a cell using its style and color! Why not? In a form, you can specify the permanent style and color of any object using the Text and Graphics Menus. This permanent style and color usually overrides the style and color of individual data cells.

If you want the style and color of data cells to override the permanent style and color, open the **Form Preferences** dialog (Setup Menu) and check the Use Data Style/Color. When this option is checked, the style and color of the each data cell overrides the permanent style and color specified in graphic design mode. Note: This option works only with data cells, it does not work with Text Editor SuperObjects. (See "Text Editor SuperObject" on page 639 for more information on Text Editor SuperObjects.)



When you are using the data sheet, Panorama always displays the style and color of each individual data cell.

Accessing Style and Color in a Formula

Panorama formulas can use the `fieldstyle(` function to access both the style and color of individual data cells. When combined with the **Formula Find/Select** command, these functions allows you to select data based on its style or color. (See "Formula Find / Select" on page 387 for more information on this command.)

The basic syntax for the `fieldstyle(` function is:

```
fieldstyle(fieldname)
```

This function returns the style and color of a data cell— bold, italic, etc. The fieldname parameter is a string, so it should usually be in quotes—for example `fieldstyle("Price")="bold"`. If the data cell has more than one style or color, this function will return all of them, for example red bold italic. Use the `contains` operator (see "String Testing Functions" on page 1223) to check for a specific style or color, for example

```
select fieldstyle("Name") contains "italic"
```

To check if a cell is plain, use a formula like this

```
fieldstyle("Address")=""
```

For more information on this function see "FIELDSTYLE(" on page 5221.

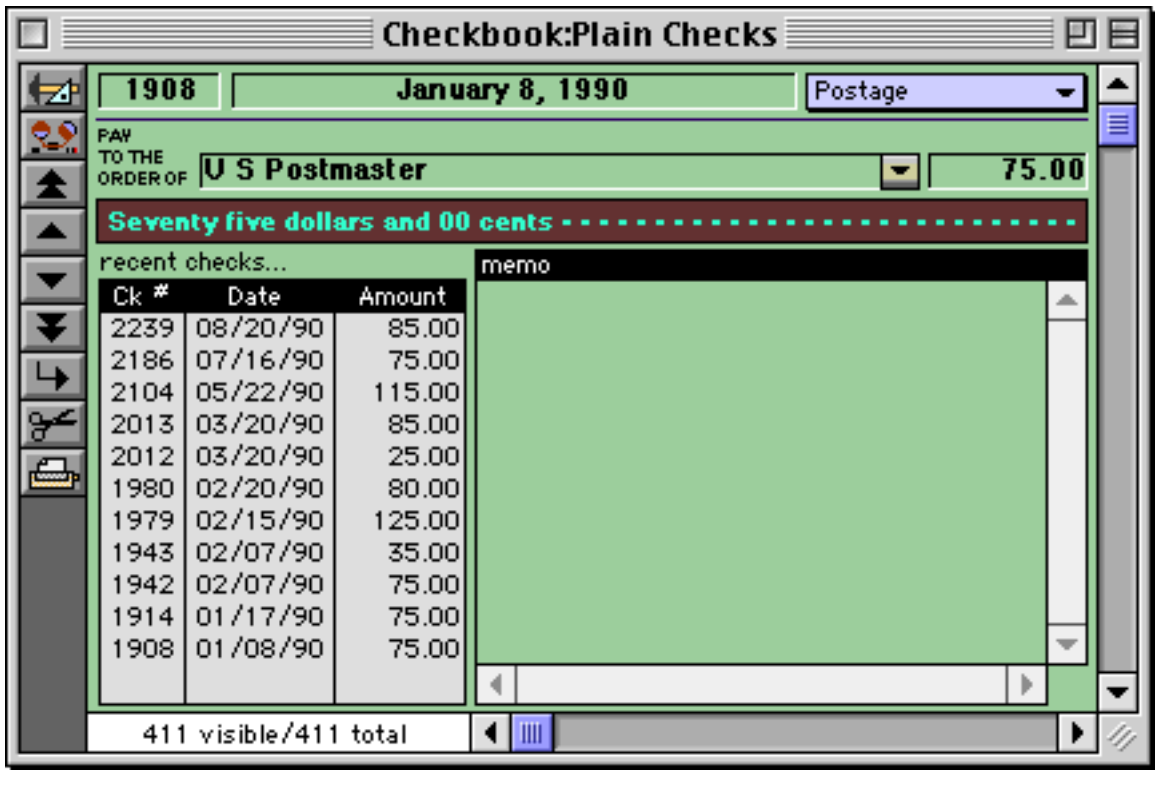
Chapter 13: Introduction to Forms



Panorama has two interfaces for displaying and editing data — the data sheet and forms. So far most of this manual has concentrated on using the data sheet. Starting with this chapter we'll introduce a much more flexible way to display and edit data: the form.

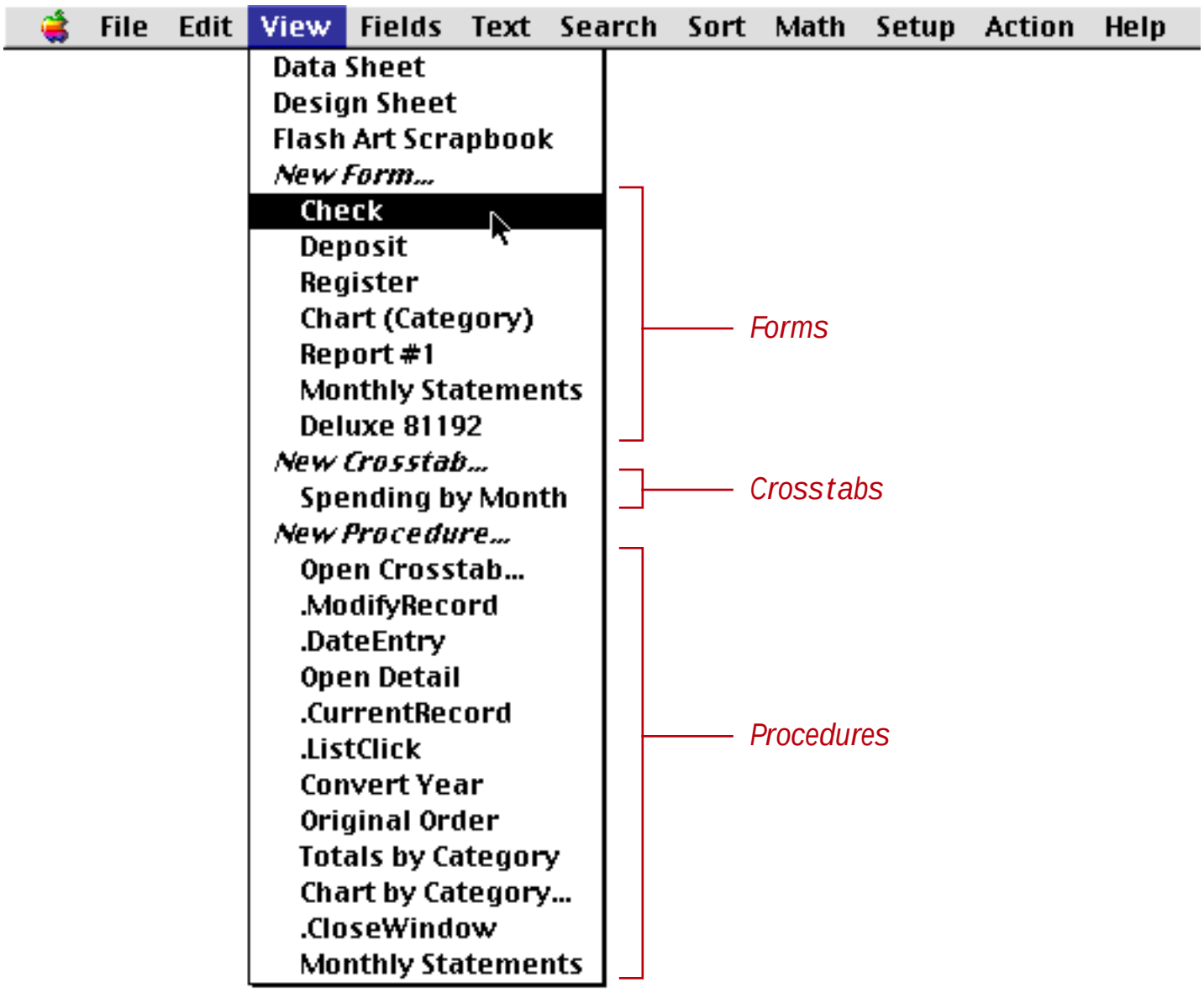
A single database only has one data sheet but it may contain many forms. You can design each form for a specific purpose, for example entering data, printing a mailing label, or printing a report.

The data sheet displays a fixed format of rows and columns. You can change the text font and the width of the columns, but beyond that you don't have any control over the data sheet's appearance. Each form, on the other hand, is completely customizable. You can (and in fact must) set up the placement of each item on the form, including data, text and artwork. The form view is much more flexible than the data sheet view, but it is also more work to set up. Here is a typical example of a form. Notice that the window name shows the database name, Checkbook, followed by the form name, Plain Checks.



Opening a Form

The **View** Menu lists all the views in a database, including forms. The pre-defined views appear at the top—data sheet, design sheet, and flash art scrapbook. Next come the views you've created—forms, crosstabs, and procedures. The View Menu also contains commands for creating your own new views—new form, new crosstab, and new procedures.

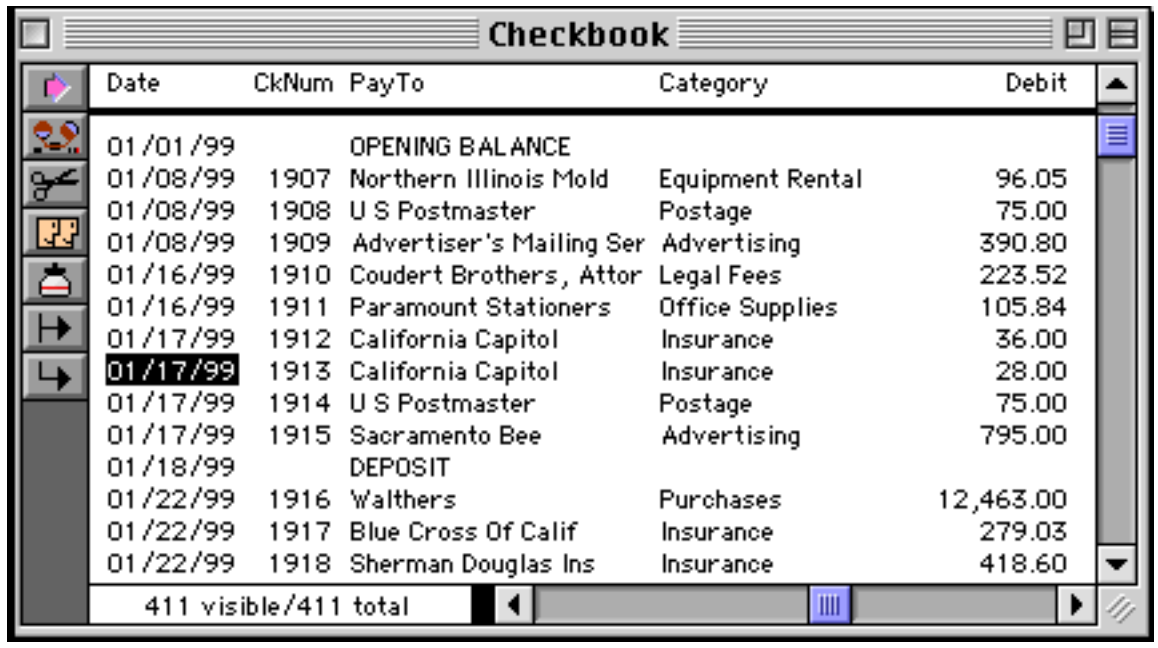


To open a form within the current window, simply choose the form from the menu. You can flip back and forth between different forms (or other views, like the data sheet) at any time.

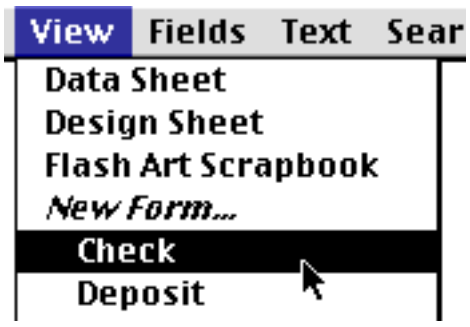
Opening A Form in a New Window

If you wish, you can open a form in a new window, allowing you to see two different views of the database at once. To open a form in a second window the same size as the current window, hold down the Alt key while you select from the View Menu. (If you are using a Macintosh, hold down the Control key.) The new window will appear slightly below and to the right of the original window.

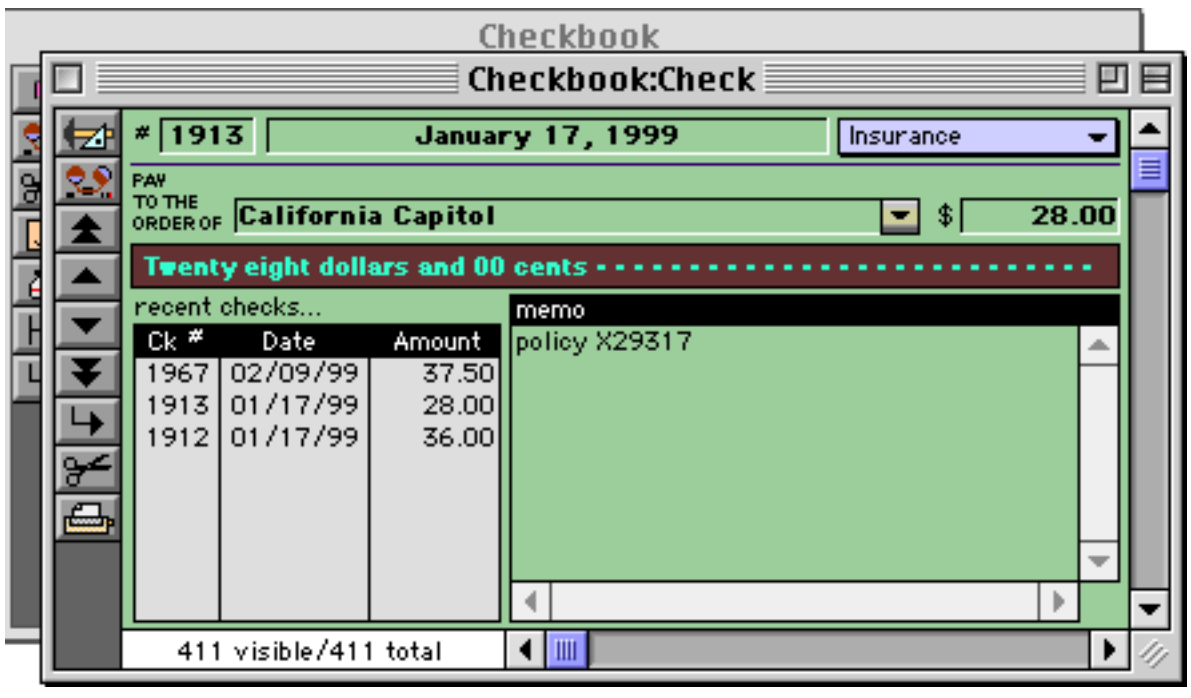
1) Start with one window



2) While holding down the Alt key (PC) or the Control key (Mac), make a selection from the View menu.

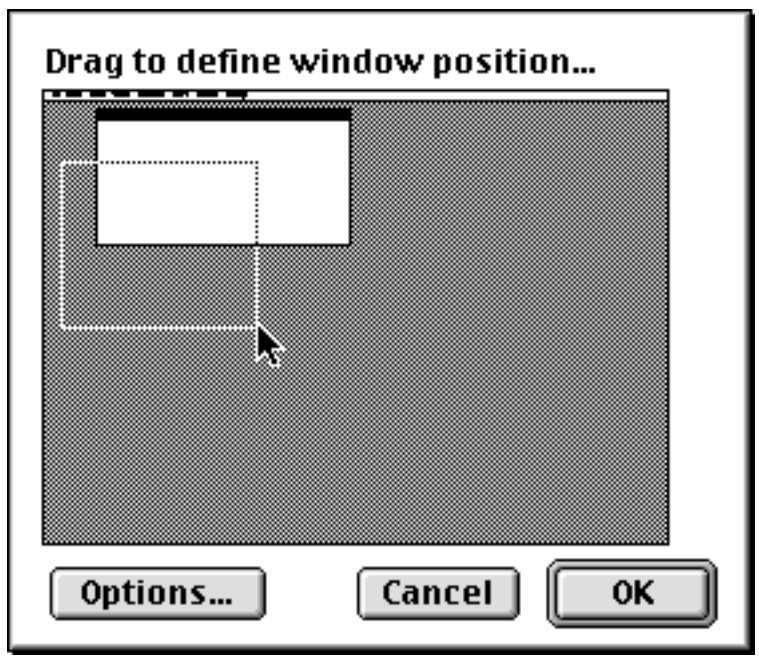


3) The new window appears sightly below and to the right...



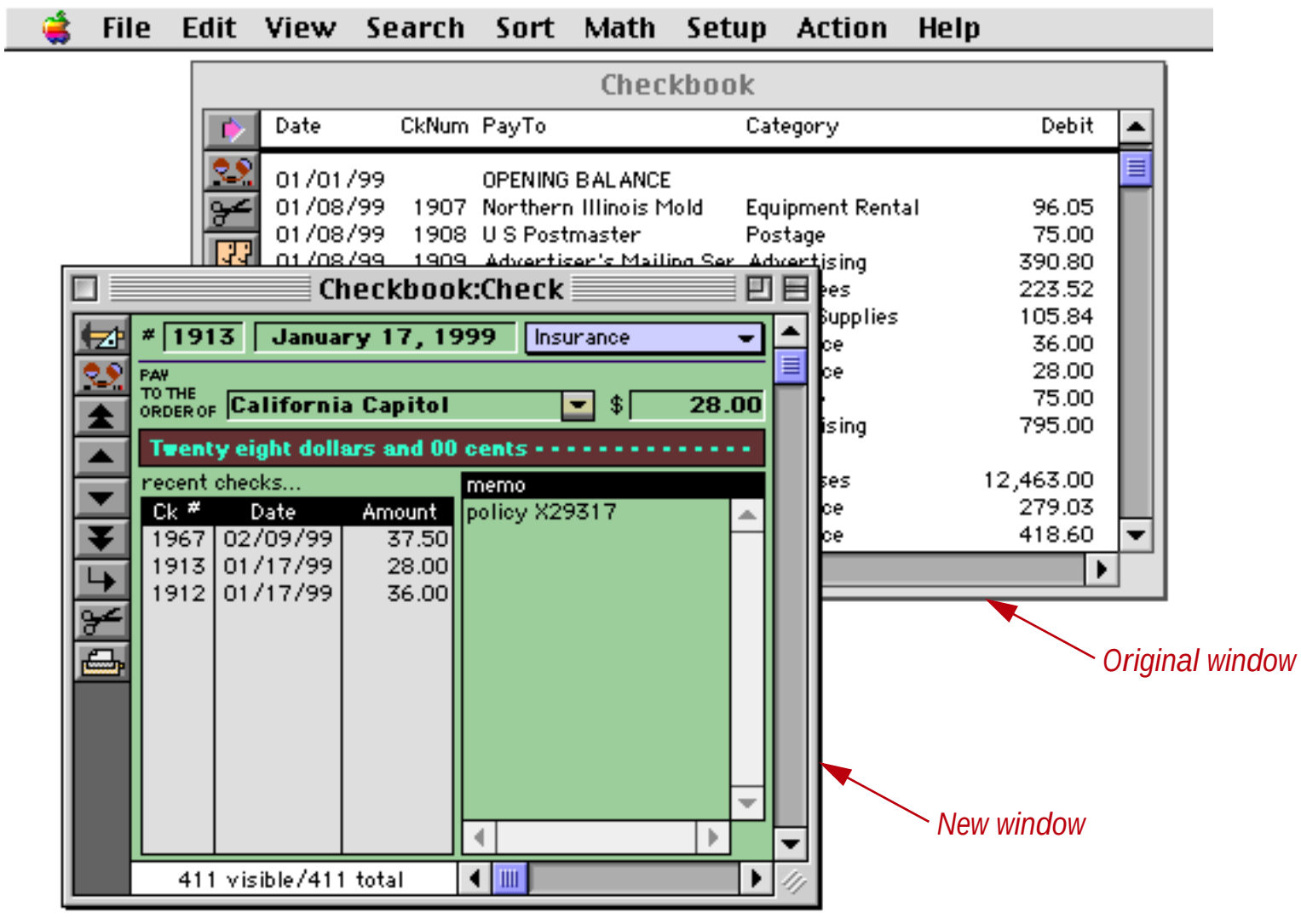
The new window will track the original window. Any changes made in this new window automatically appear in all other windows, and when any navigation is done in one window (moving up or down within the database) all of the other windows will follow along.

Another technique allows you to control the exact size and position of the new window in advance. To use this technique, hold down the Control key while you select from the View Menu. (If you are using a Macintosh, hold down the Command key.) After you choose the view you want to open, the Window Options dialog will appear shown below. This dialog shows a miniature view of the entire computer screen, along with the positions of every window.



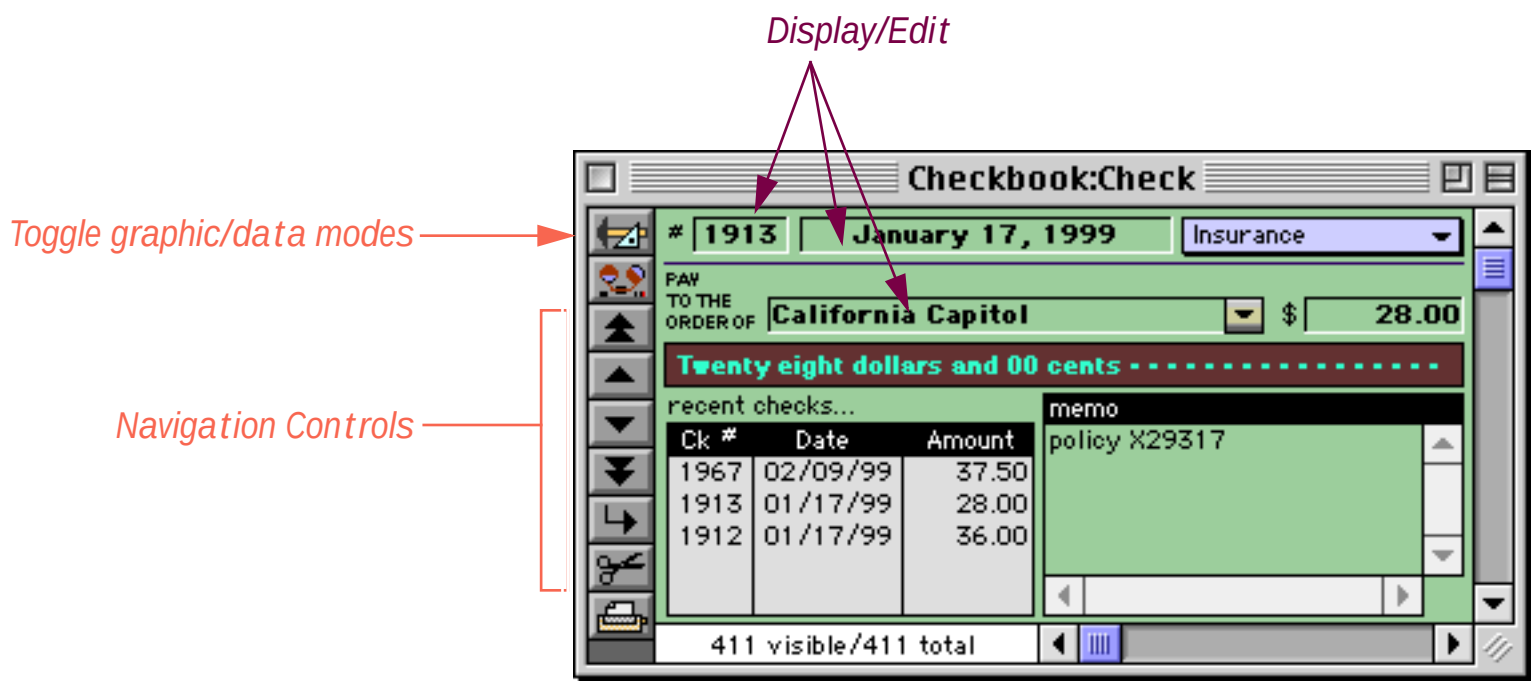
The Options button in the Window Options dialog allows you to selectively eliminate up to four components from a new window—the tool palette, scroll bars, and drag bar. See “Window Options” on page 226 for more information about eliminating these components.

To define the position and size of the new window, simply drag a rectangle across the miniature screen, as shown above. If you don’t get the position quite right, simply drag again. When you press the Ok button the new window will open in the location you have specified.

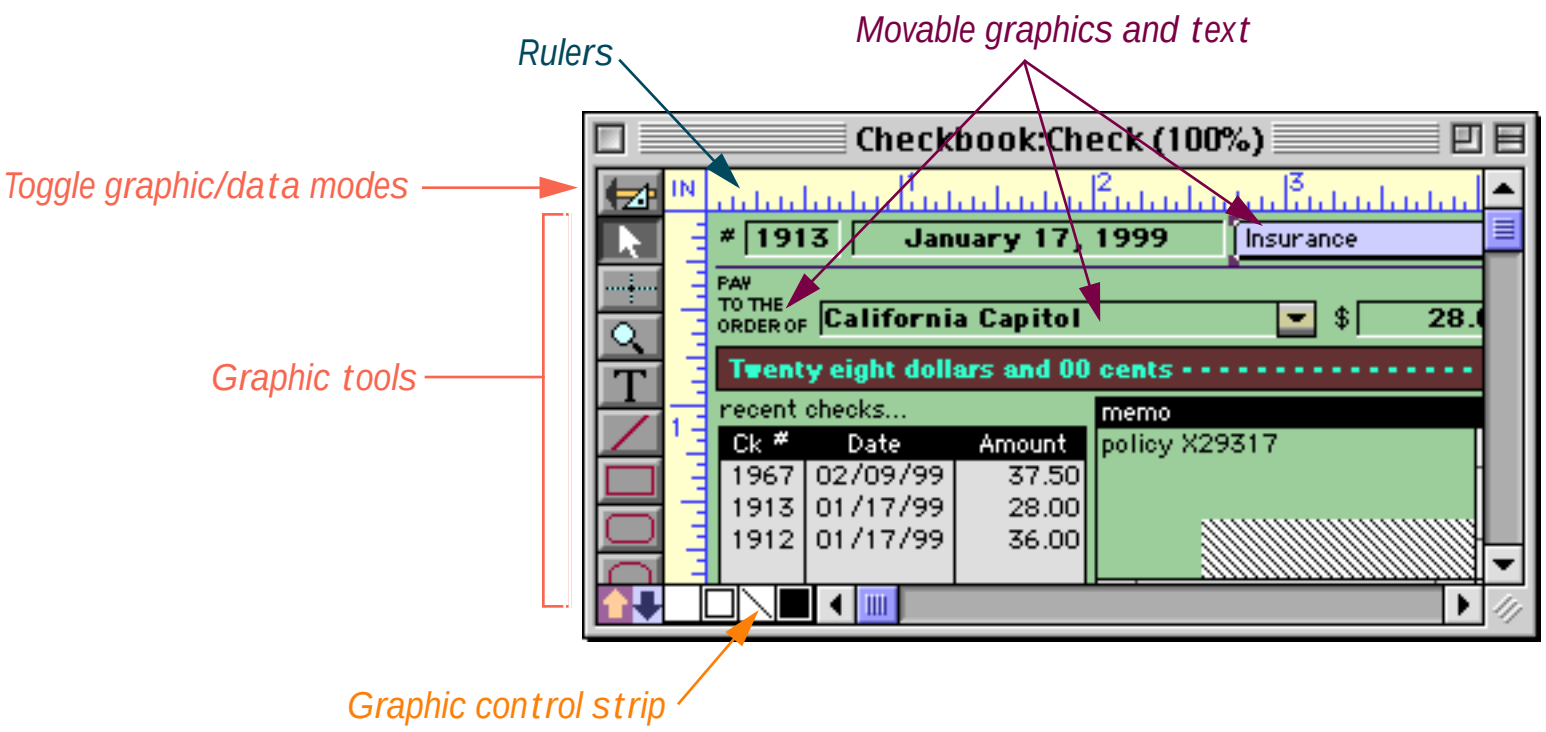


Form Modes: Data Access vs. Graphic Design

Unlike other views, the Form View operates in two distinct modes—data access and graphic design. Data access mode (also called "data mode") is the default mode. In this mode you can view and display data, and navigate through the database.



Graphic design mode (also called "graphics mode") functions like an electronic drafting table. In this mode you design the form by drawing lines, boxes, and other graphic elements. This mode is very similar to many drawing and page layout programs. Graphic design mode is easily recognized by the rulers that appear at the top and left edges of the windows.

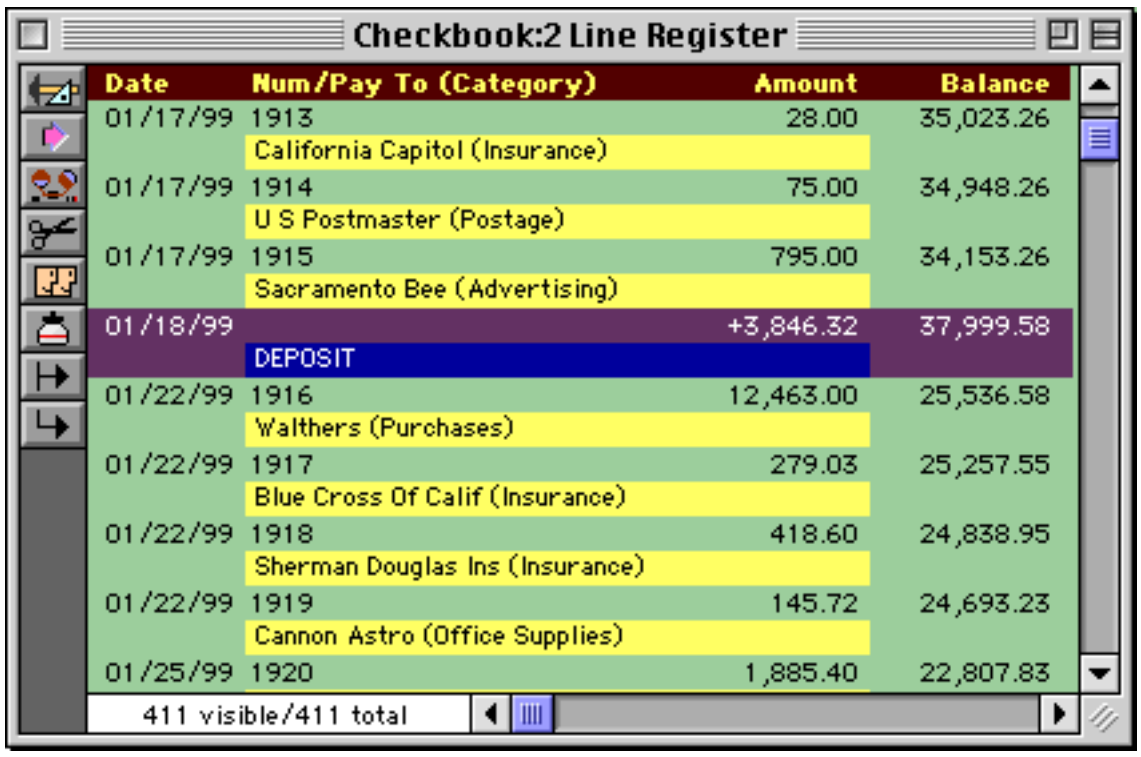


To switch between data access and graphic design modes, click on the  tool. Each click on this tool toggles the window between the two modes.

Form Operation: Individual Pages vs.View-As-List

Panorama allows you to set up blank forms as individual pages or as a continuous sheet (view-as-list). When forms are set up as individual pages you see one record at a time. You can flip through the records just as you would shuffle through a stack of paper forms. All of the examples of forms you've seen so far are individual page forms.

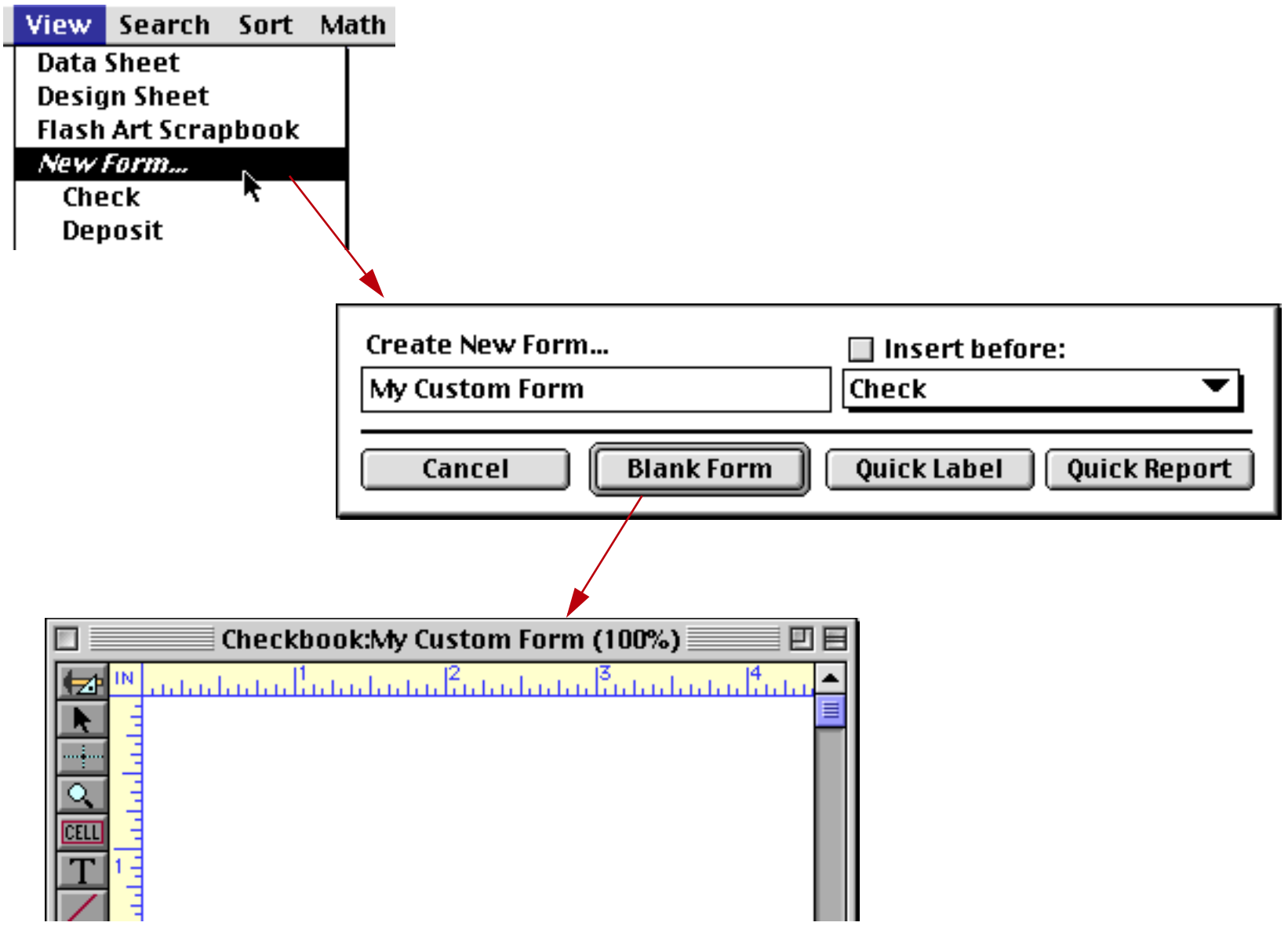
A view-as-list form displays data as a continuous sheet, as shown below. Instead of flipping from record to record, you scroll up and down through the data in a manner similar to the data sheet. However, unlike the data sheet, a view-as-list form allows you to arrange the data any way you like, and even include graphics in the display. On the other hand, view-as-list forms are slower than the data sheet (because of the overhead in displaying the graphics) and they are much more work to set up.



Unless you tell it otherwise, Panorama sets up a new form as individual pages. To convert the form to a continuous sheet you must use the **Form Preferences** command (Setup menu) to set the View-as-List option. You will also have to define the boundaries of the form by setting up a data tile (and optional header tile, see "Adding a View-As-List Header" on page 899). For more information about setting up view-as-list forms see "Creating a View-As-List Form" on page 892.

Creating a New Form

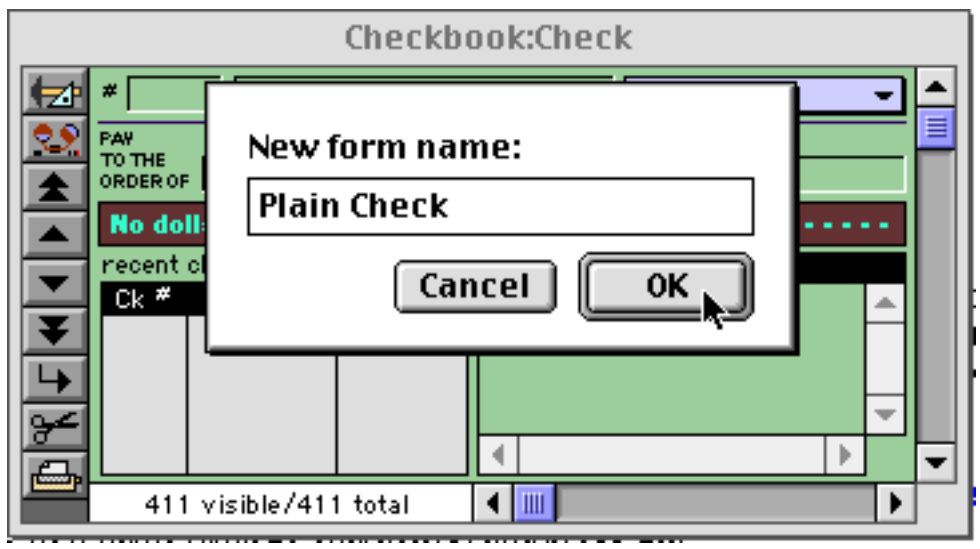
To create a new view, choose **New Form** from the View Menu. A dialog box will appear asking you to name the new form. A form name may be up to 25 characters long and can contain any letter, number or punctuation.



When you create a new form, it usually becomes the last form in the **View** Menu. If you wish, you can insert the new view into the middle of the **View** Menu. To do this, check the Insert before button and use the pop-up menu directly below the Insert before button to specify the position of the new view. You can also re-arrange the order of the forms using the **Re-Arrange Forms** command in the Setup menu. See “Changing the Order of Forms, Crosstabs or Procedures” on page 237 for more information on this process.

Renaming a Form

To rename the currently visible form choose **Rename Form** from the Setup Menu. Type in the new name (limit 25 characters) and press Ok.



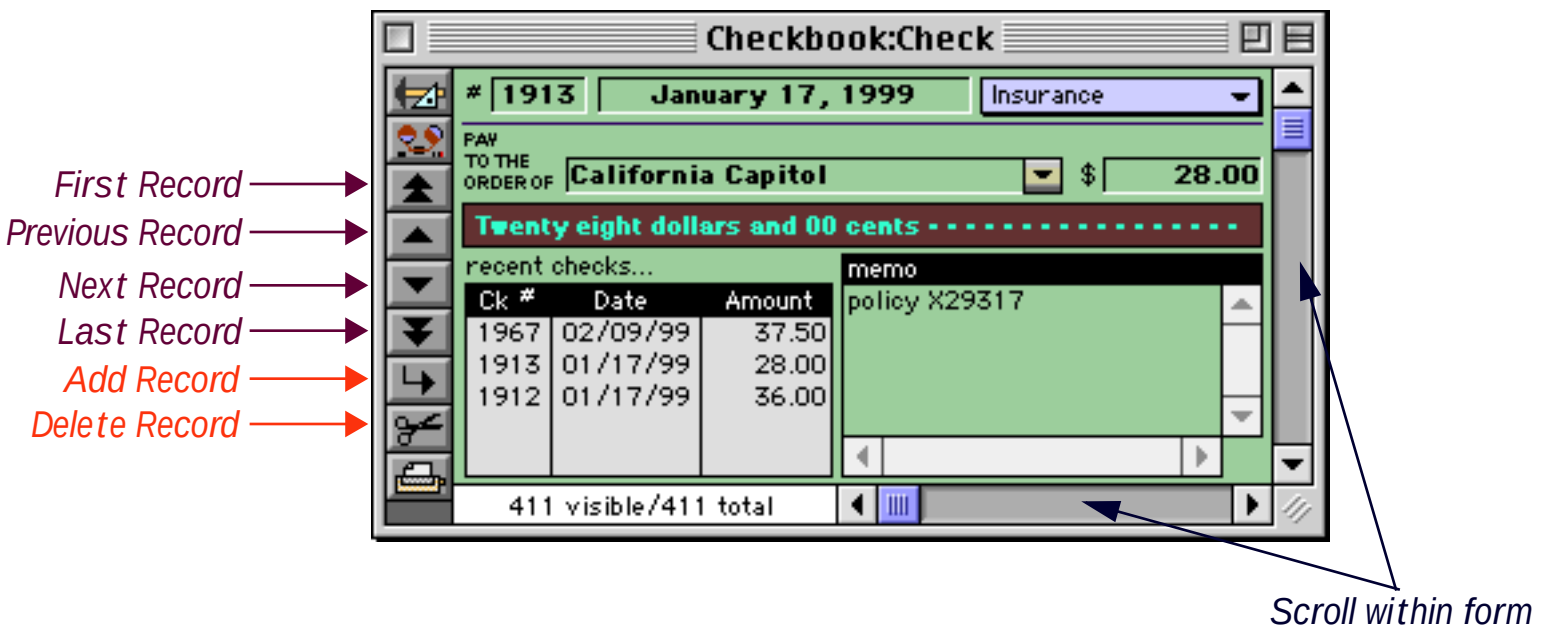
Deleting a Form

To delete a form choose **Delete Form** from the Setup Menu. Since you cannot undo after you delete a view, Panorama will ask you if you are sure before it actually deletes the form. Note: If the form is the only window open for this database, Panorama will close the entire file when it deletes the form. To avoid this, open an additional window (the data sheet or another form) before you delete the form.

You can also delete forms with the **Re-Arrange Forms** command in the Setup menu. This is the fastest way to remove several forms at once. See "Changing the Order of Forms, Crosstabs or Procedures" on page 237 for more information on this process.

Browsing the Database With a Form

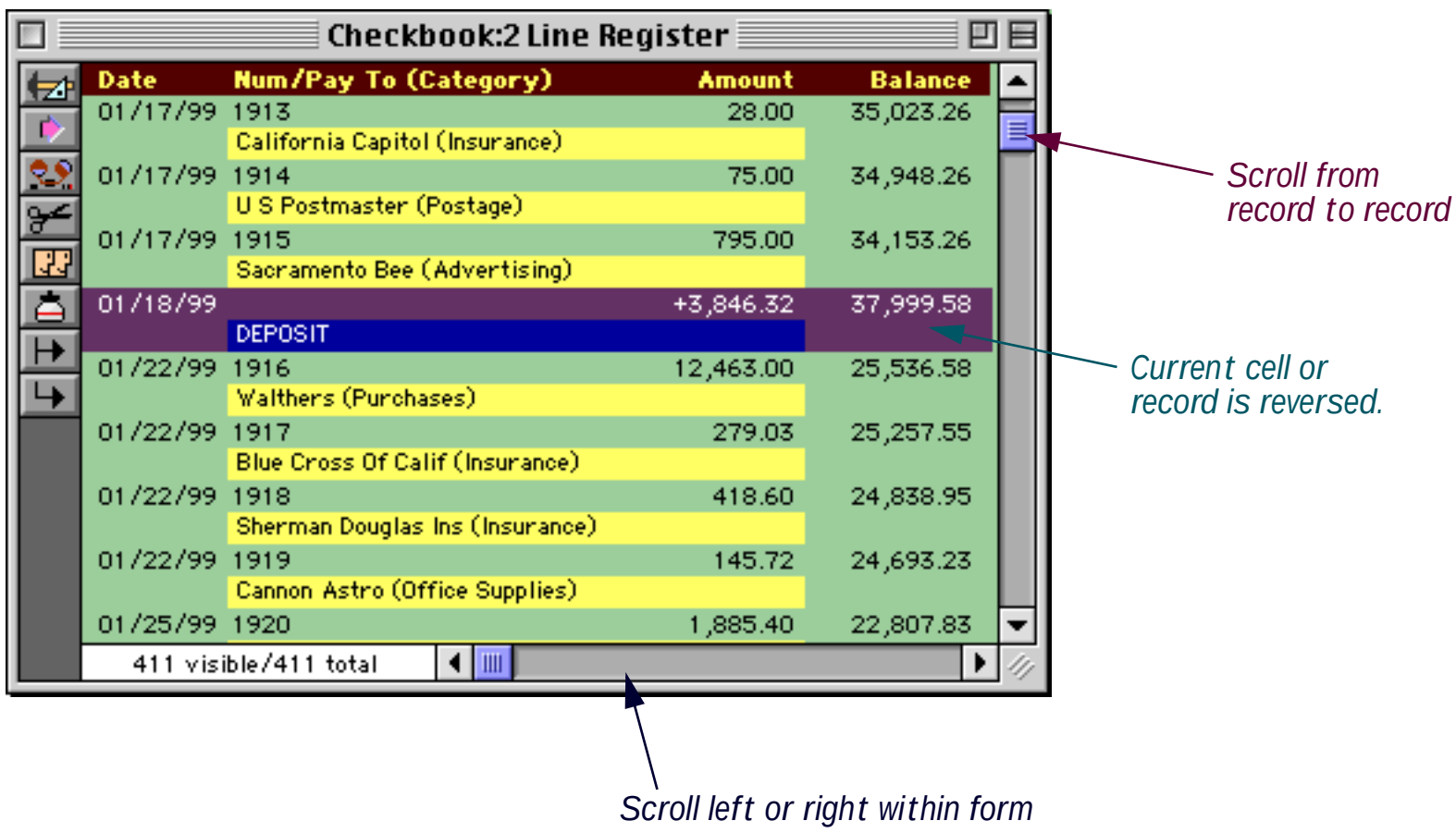
When working with a normal form (as opposed to a view-as-list form) Panorama displays one record at a time. You can navigate from record to record using the VCR style buttons in the tool palette.



If the form is taller and/or wider than the current window, you can use the scroll bars on the bottom and right to slide the form around within the window. The form can also slide automatically as you tab from cell to cell within the form.

Browsing the Database With a View-As-List Form

When working with a view-as-list form, Panorama display several records at a time — one row per record. You can click on any visible record to make it active, or use the vertical scroll bar to move to any record within the database (just like the vertical scroll bar in the design sheet). You can also use the up and down arrow keys to move up or down one record at a time.



If the form is wider than the window, you can use the horizontal scroll bar at the bottom of the window to slide the form left or right within the window. See “View-As-List Forms” on page 889 to learn how to create view-as-list forms.

Chapter 14: Graphic Design



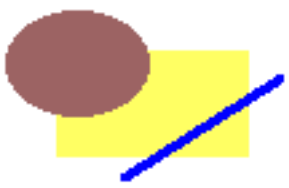
Panorama has a built in graphic editor for creating and modifying the layout of forms and reports. If you've used an object oriented graphic editor before you will find many familiar features.

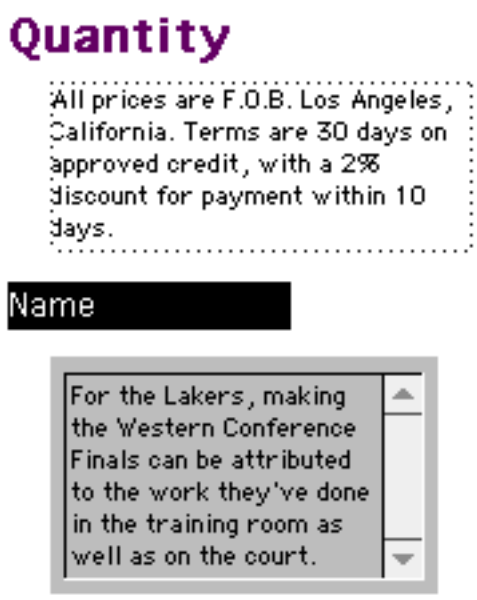
Graphic Objects

Panorama forms are built with **graphic objects** (also called simply **objects**). Each object is treated as a unit (rather than as a collection of dots), and each object has a specific shape, position, size, and color. You can easily modify an object without disturbing the other objects—for instance sliding a rectangle to a new position or changing the diameter of a circle. Most of the next few chapters deals with techniques and shortcuts for arranging graphic objects on the surface of the form.

Types of Graphic Objects

Panorama has over two dozen different types of graphic objects. Objects fall into five classes: Shapes, Text, Multi-Media, Buttons, and Layout. Each type of object has its own characteristics and appearance, as shown in the following table.

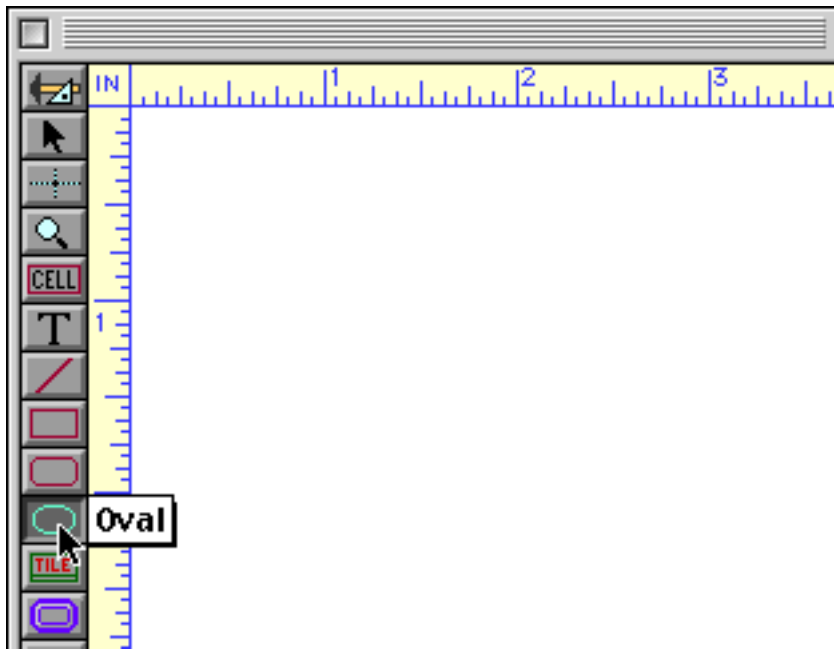
Class	Samples	Object	Tool	Description
Shapes		Line		Simple line at any angle.
		Rectangle		Simple rectangle, may be filled or transparent.
		Round Rectangle		Rectangle with curved corners.
		Oval		Oval (or circle), may be filled or transparent.

Class	Samples	Object	Tool	Description
Text		Click Text		Displays simple text captions.
		Auto Wrap Text		Displays one or more paragraphs of text. May contain fields or variables merged within the text.
		Text Display		Displays text based on a formula. The text can scroll within the object, may be aligned in 9 different positions within the object, and can scale based on the size of the form.
		Data Cell		Used for editing fields. When double clicked, an expandable pop-up editing box appears (similar to editing in the data sheet).
		Text Editor		Used for editing fields or variables. Unlike the Data Cell, there is no pop-up editing box (more like other software applications).
		Word Processor		Used for editing a field or variable containing stylized text. The text may contain different fonts, sizes, styles, margins and tab stops.
Multi-Media		Picture	none	Displays a fixed image. May be used for backgrounds, logos, etc.
		Flash Art		Displays changing images (PICT format) based on a formula. Images may be part of the database or stored as separate files on the disk.
		Super Flash Art		Displays images or QuickTime movies with advanced features like scroll bars, advanced alignment and scaling, and hypertext.
		Chart		Displays column, bar, line, area, scatter and pie charts.
		Flash Sound		Automatically plays sound (Macintosh only).

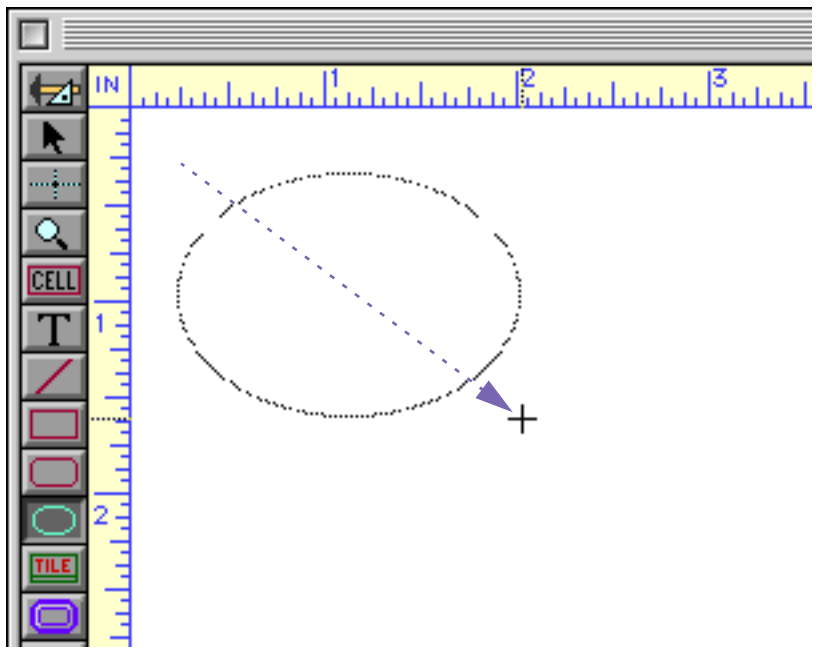
Class	Samples	Object	Tool	Description
Buttons		Button		Generic button tool for creating push buttons, radio buttons and checkboxes (fields only). (For new applications we recommend the new button tools listed below.)
		Push Button		2 and 3 dimensional push buttons in a variety of styles.
		Data Button		2 and 3 dimensional checkboxes and radio buttons in a variety of styles. These buttons may be tied to a field or a variable.
		Sticky Push Button		2 and 3 dimensional buttons that look like push buttons but act like checkboxes or radio buttons.
		Pop-Up Menu		Pop-up menu tied to a field or variable. May use any font, text size, color, or number of columns.
		List		Scrollable list.
		Flash Art Push Button		Push button with custom artwork.
		Flash Art Data Button		Checkbox or radio button with custom artwork.
		Scroll Bar		Standalone scroll bar.
Layout		Tile		Used for report and view-as-list layout.
		Super Matrix		Display a repeating matrix that may contain graphics and data. Options include grid lines and the number of rows and columns.
		Auto Grow		Adjusts other objects as window size changes, making the form “elastic.”

Creating a Graphic Object

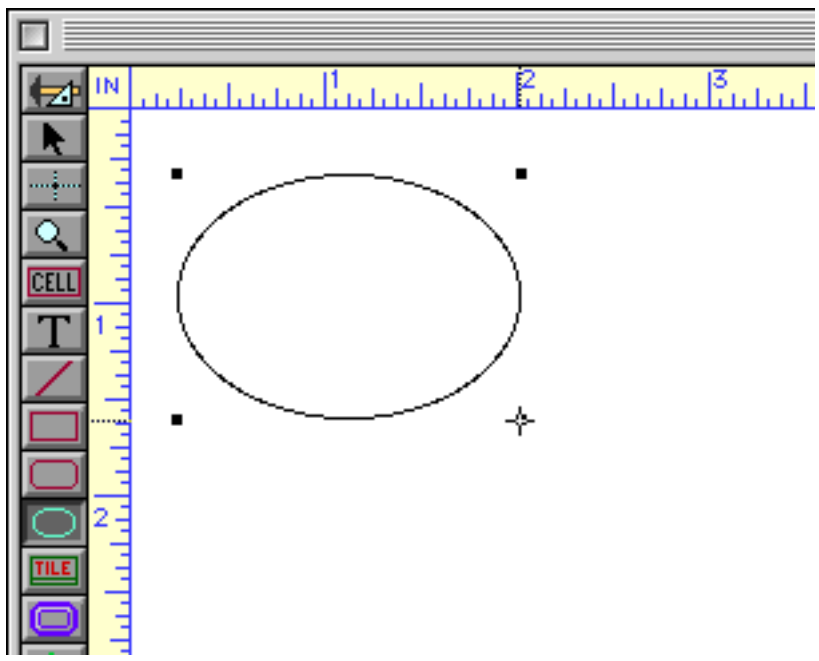
To create a new graphic object, first click on the appropriate tool in the tool palette. For example, to draw an oval you would click on the **Oval** tool.



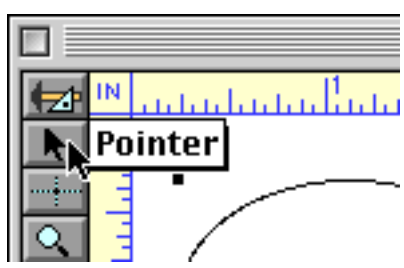
Then move the cursor onto the form and drag the mouse across the surface of the form to define the location and size of the new object (the dragging motion is shown by the dashed arrow in the illustration below). A gray outline of the new object will follow the mouse.



When you release the mouse, the new object will appear.

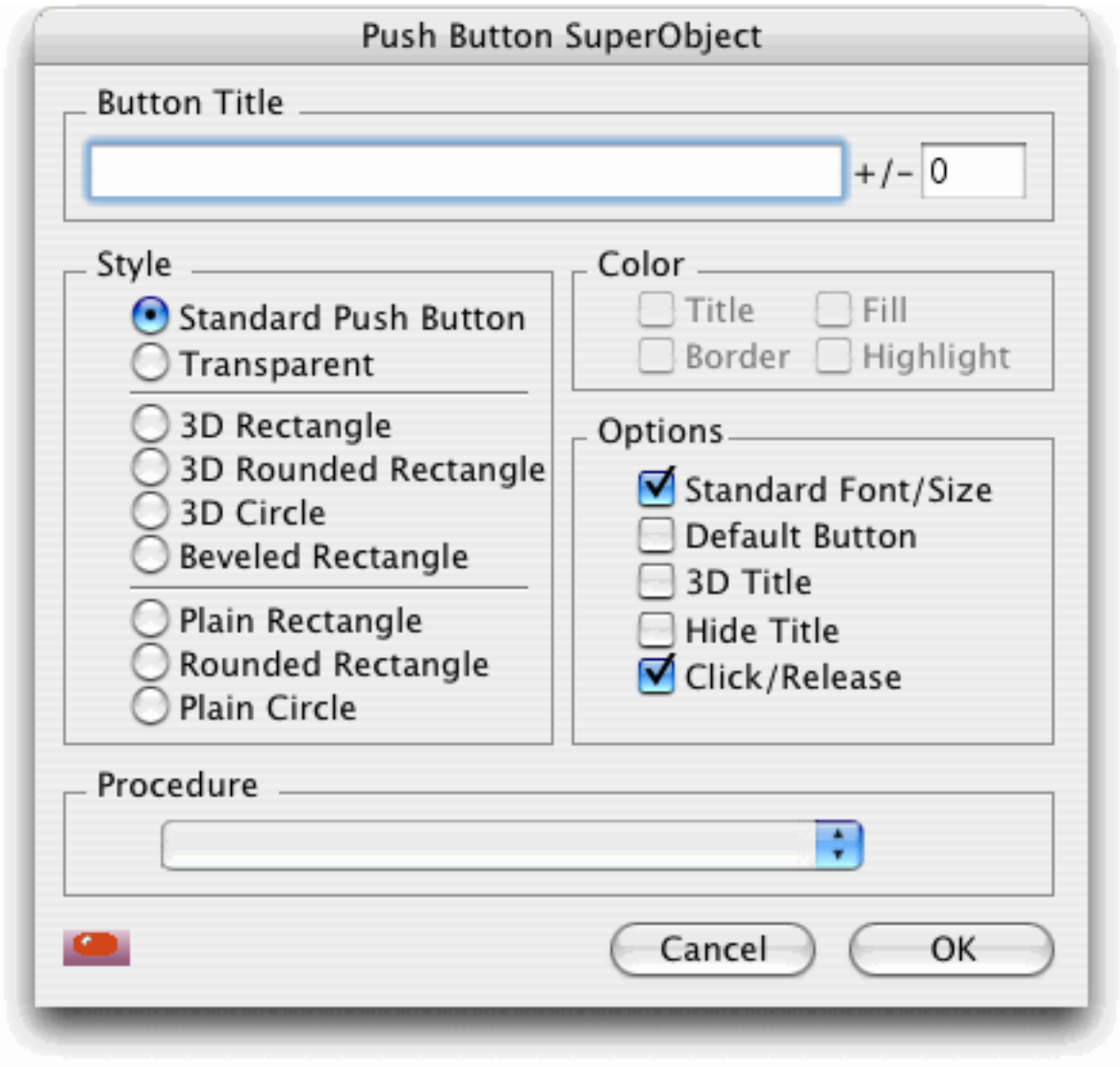


Each time you drag across the form you will create a new shape. When you are finished creating shapes, click on the **Pointer** tool.



Don’t forget to click on the Pointer tool when you are done! If you don’t, the next time you click you will create another graphic object.

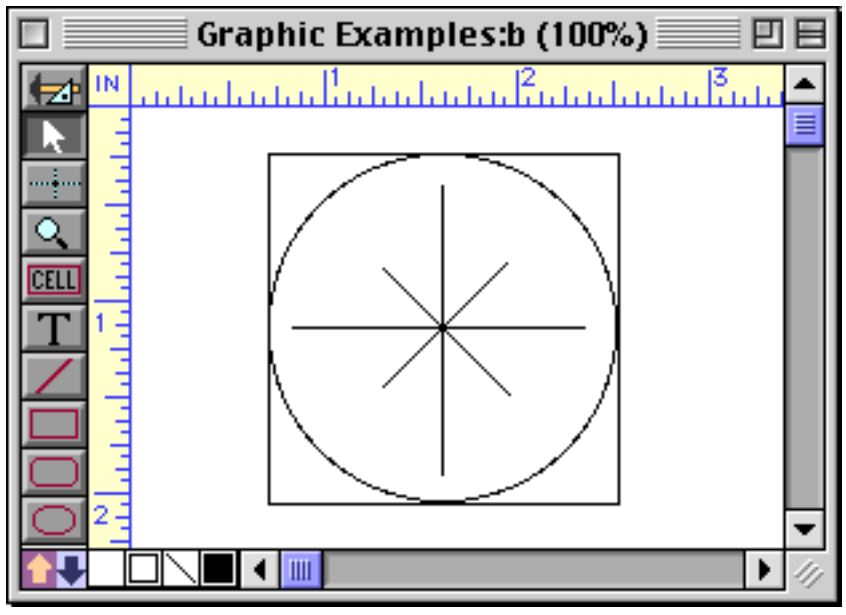
The procedure for creating more complex objects is the same as for simple objects: 1) select the tool, 2) drag the mouse across the form. However, when you release the mouse after creating a more complex object a dialog will appear allowing you to configure the new object. Each type of complex object has its own dialog. For example, here is the dialog for creating a push button.



Rather than describing the dialog for each type of item here, each will be described in detail later along with the corresponding objects.

Creating Perfect Squares, Circles and Lines

If you press the **Shift** key while you create a rectangle or oval, Panorama will automatically force the new shape to be a perfect square or circle. The **Shift** key was used to create the illustration below.

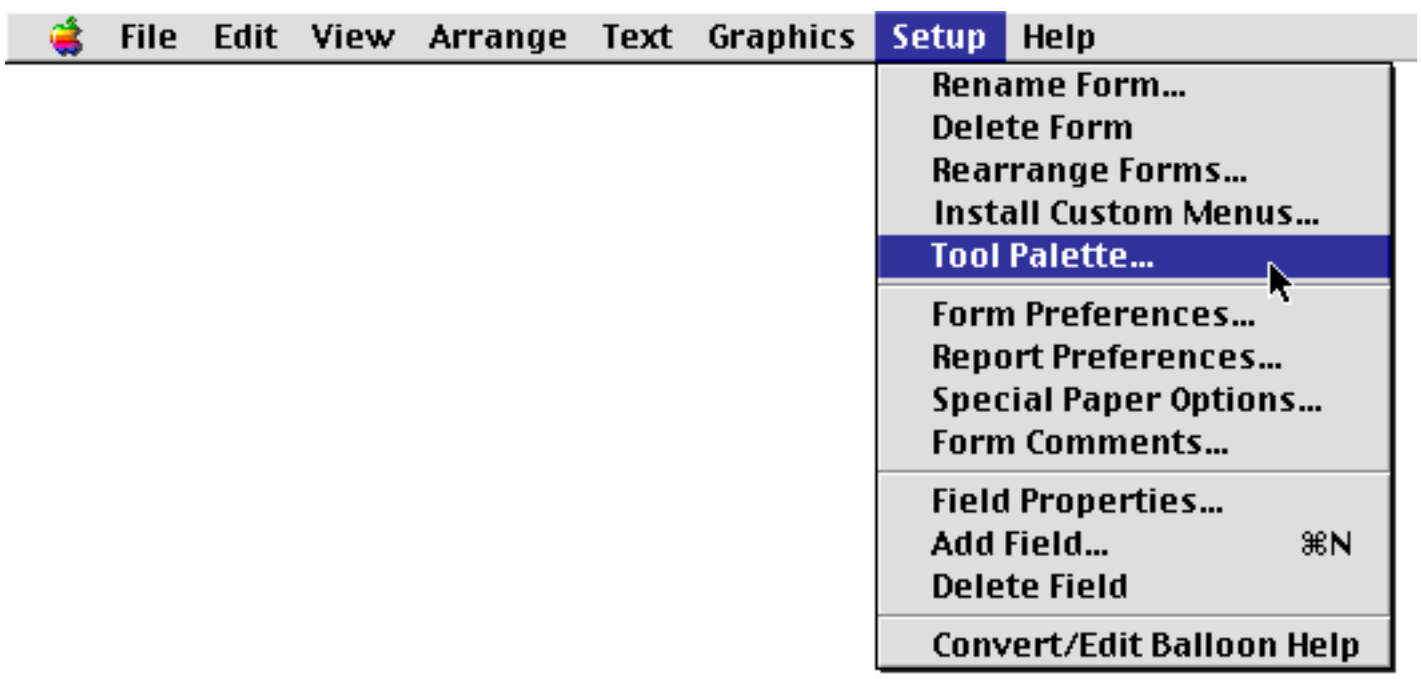


If you press the **Shift** key while creating a line Panorama will force the alignment of the new line to a multiple of 45 degrees (0°, 45°, 90°, 135°, etc.).

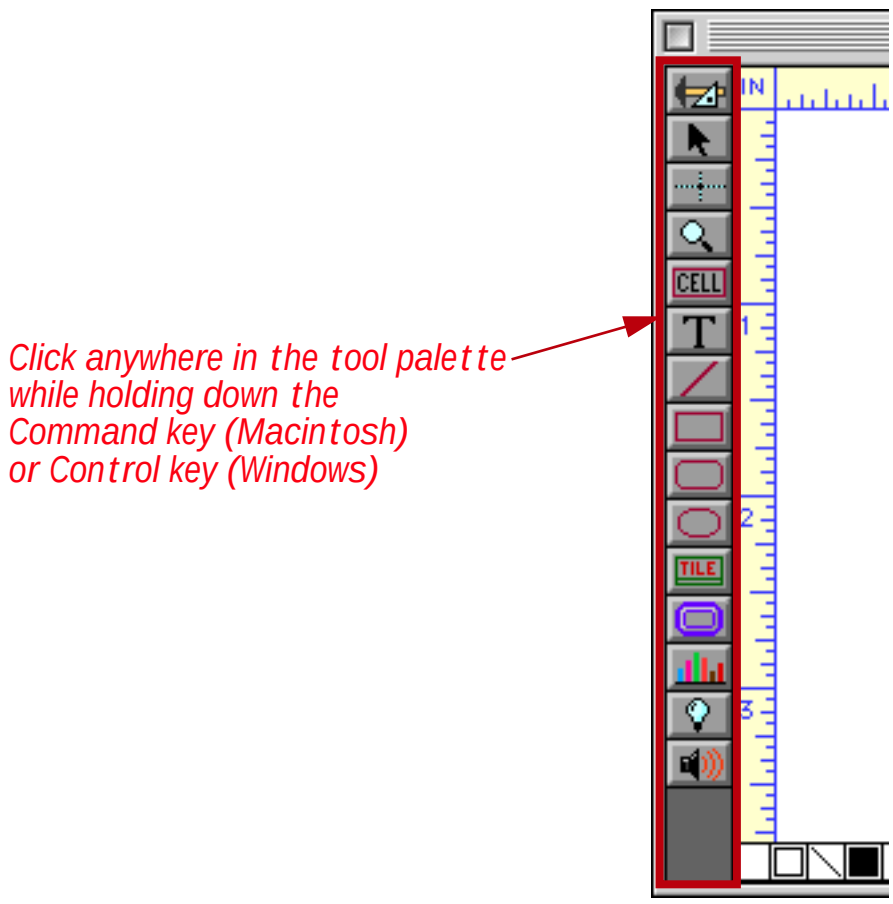
Customizing the Tool Palette

There are a total of 29 graphic tools available for use in Panorama. Many computer screens are not large enough to handle this complete palette of tools (and we expect the number of tools to increase in future versions). To get around this problem, Panorama allows you to customize the graphic tool palette on the fly. You can configure the palette to contain only the tools that you need right now in any order you want. If your needs change later, you can simply reconfigure the palette at any time.

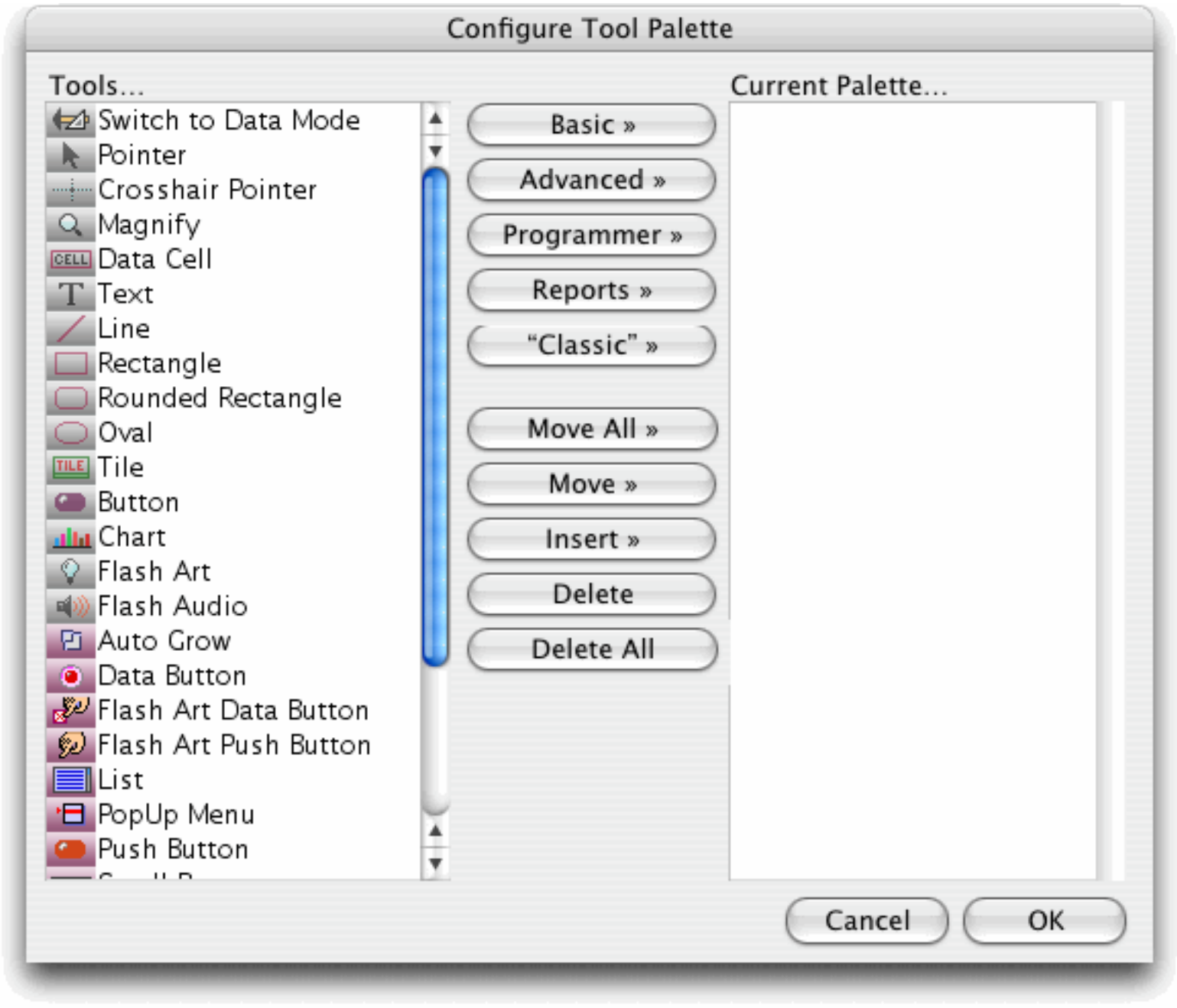
To customize the graphics tool palette, choose **Tool Palette** from the Setup menu.



Tip: You can also open his dialog by holding down the **Command** key (Macintosh) or **Control** key (Windows) and clicking anywhere in the graphic tool palette.



The **Configure Tool Palette** dialog contains two lists of tools. On the left is a list of all tools available.



On the right is a list of the tools you currently have installed. If this list is empty, Panorama will use the default tool palette (which is the original tool palette from Panorama II). Tools can be moved into the palette on the right either individually (one at a time) or in groups. To move an individual tool from the left to the right, double click on the tool in the left. Or you can select the tool (or tools) and press the **Move** or **Insert** button.

To move an entire group of tools at once press the **Basic**, **Advanced**, **Programmer**, **Reports** or **"Classic"** buttons. The table below shows the tools included by each of these options.

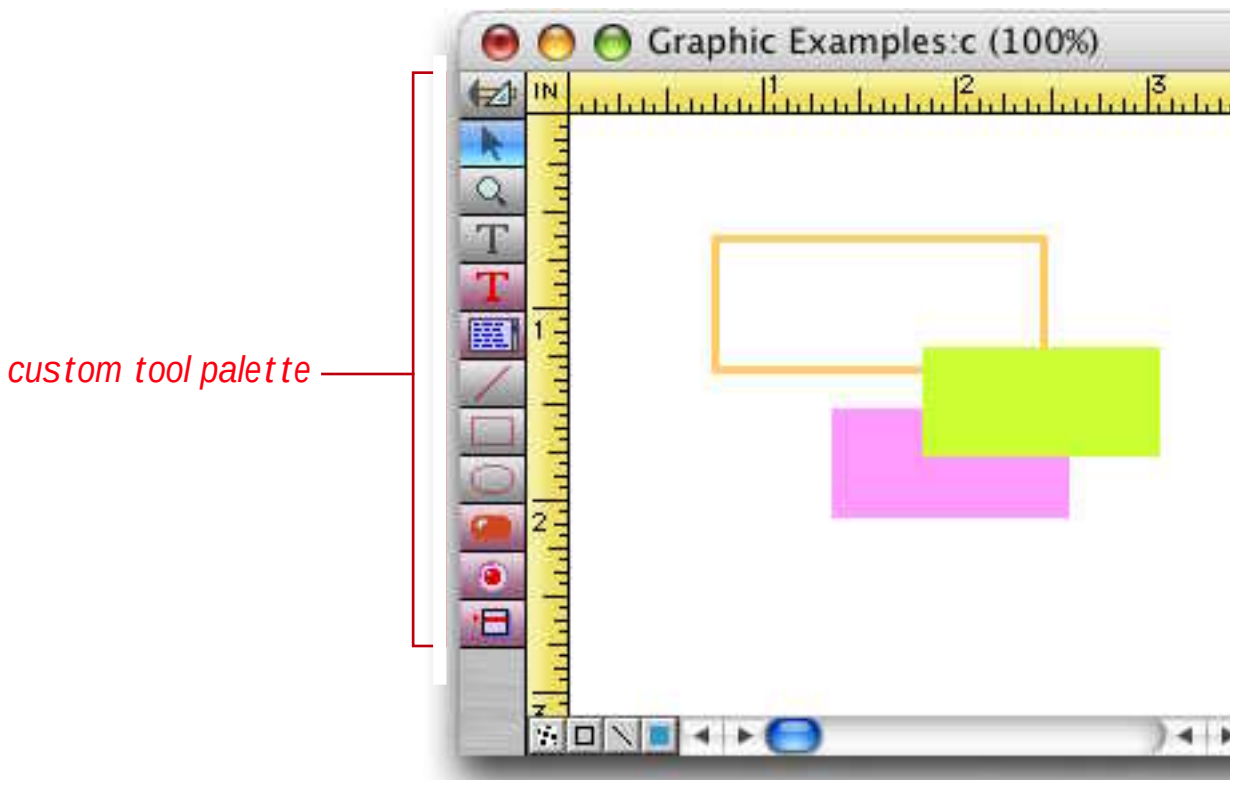
Basic	Advanced	Programmer	Reports	"Classic"
Switch to Data Mode Pointer Crosshair Pointer Magnify Data Cell Text Editor Text Text Display Line Rectangle Rounded Rectangle Oval Push Button Data Button	Switch to Data Mode Pointer Crosshair Pointer Magnify Data Cell Text Editor Text Text Display Line Rectangle Rounded Rectangle Oval Tile Push Button Data Button Sticky Push Button PopUp Menu Super Flash Art Word Processor (Cell)	Pointer Crosshair Pointer Magnify Text Editor Text Text Display Line Rectangle Oval Super Flash Art Word Processor (Cell) Button Push Button Data Button Sticky Push Button PopUp Menu List Flash Art Push Button Flash Art Data Button Scroll Bar Super Matrix Auto Grow	Switch to Data Mode Pointer Crosshair Pointer Magnify Text Text Display Line Rectangle Rounded Rectangle Oval Tile Super Flash Art	Switch to Data Mode Pointer Crosshair Pointer Magnify Data Cell Text Line Rectangle Rounded Rectangle Oval Tile Button Chart Flash Art Flash Audio

To delete one or more tools from the right hand list, select the tool(s) and press the **Delete** button. You can also delete a tool from the right hand list by double clicking on it. The **Delete All** button clears the list on the right so you can start over or go back to using the default tool palette.

Here is an example of a custom palette configuration:



When you press the **OK** button the new palette configuration will become active, like this.



If you need to change the tool palette again later, just open the dialog again and make the necessary adjustments.

Using the Keyboard to Select Common Tools

Usually you will use the mouse to select the tool you want to use. The most common tools, however, can also be activated with the keyboard. This saves you a trip to the tool palette each time you want to select one of these tools. The table below lists the tools that can be selected with the keyboard. Note that these keys are pressed by themselves — not in combination with any other key.

Tool	Key	Notes
		Press the **Escape** key to toggle between Graphics Mode and Data Access Mode (disabled if tool palette has been disabled)
		Press the **P** key to select the Pointer tool (except when typing or editing text with the T tool).
		Press the **Enter** key to select the Pointer tool (at any time, even when typing or editing text with the T tool).
		Press the= key to toggle the crosshair cursor on / off (See “Nudging to the Crosshair Cursor” on page 515)
		Press the **T** key to select the Text tool
		Press the **L** key to select the Line tool
		Press the **R** key to select the Rectangle tool
		Press the **O** key to select the Oval tool

SuperObjects

The astute observer will notice that the list of tools in the configuration dialog is divided into two groups - gray tools and purple tools. The purple tools are actually not built in to Panorama, but are special plug-ins called **SuperObjects**. Because they are written as plug-in tools, ProVUE can develop new SuperObjects faster and with more capabilities than for standard objects. You can expect to see many more SuperObjects added to Panorama in the future. You may also notice that some SuperObjects perform functions similar to regular objects, but with more features. However, as far as you, the user, are concerned, you don't really have to worry about whether an object is a SuperObject or not. The techniques for creating and modifying SuperObjects and regular objects are the same.

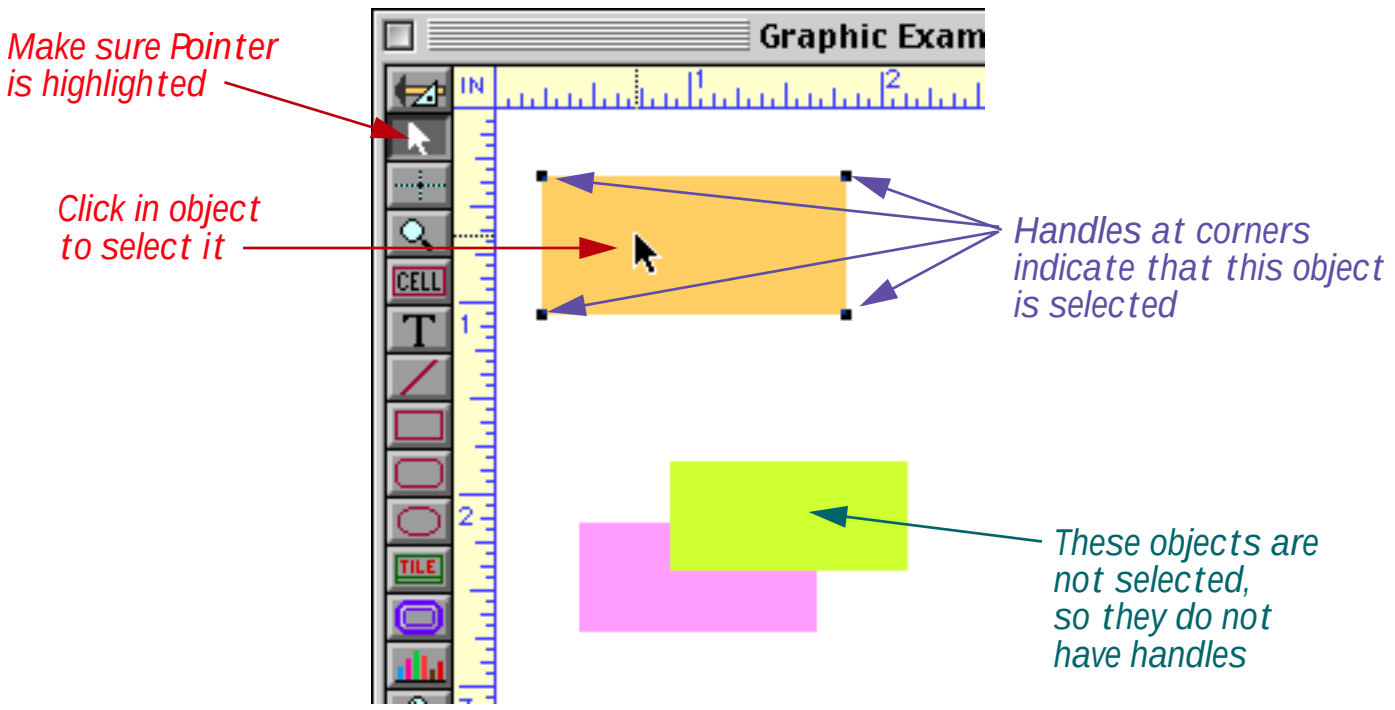
Modifying Objects

Once an object is created it is far from being cast in stone. At any time you can go back and move, resize, change the color, change the alignment, or make virtually any other change. You can change objects one at a time or in groups. About the only change you cannot make is changing an object into another type of object (for example, you cannot change a square into a circle or change a rectangle into a pop-up menu).

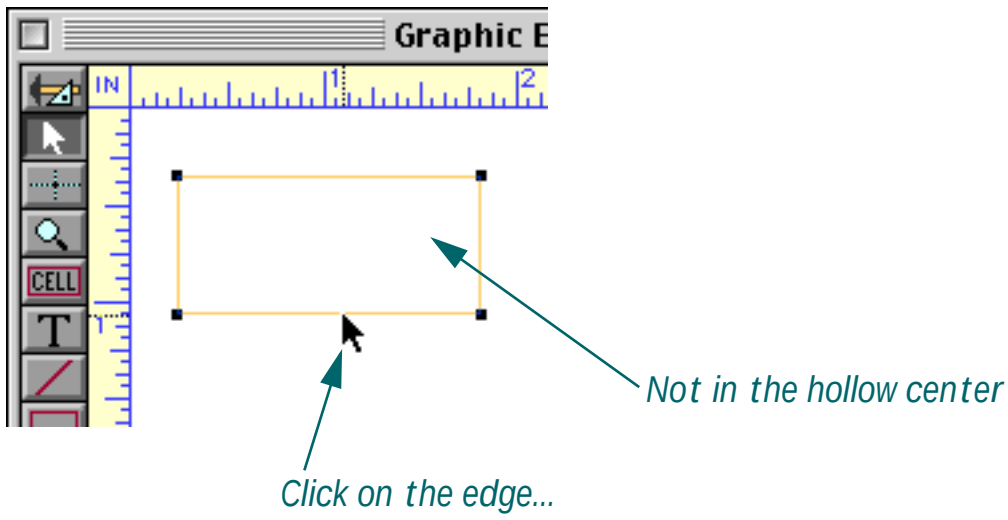
Selecting a Single Object

Before you can modify an object (change its size, color, pattern, etc.) you must select the object. Selecting an object (or objects) tells Panorama that you want to work with that object. The **Pointer** tool is used for selecting objects. If the **Pointer** tool is not highlighted, click on it before you try to select an object. One of the most common mistakes made by new users is to try to select an object when a tool other than the **Pointer** tool is highlighted.

There are two ways to select an object. The simplest is to click on the object. When an object is selected, **handles** appear at the corners of the object. The **handles** let you know the object is selected and waiting for you to do something with it.



If an object is hollow (transparent, or filled with NONE) you must click on the border of the object to select it. Objects with thin borders may be difficult to click on. If you find it too difficult, remember that you can also select an object by dragging a selection marquee around it (see the next section).

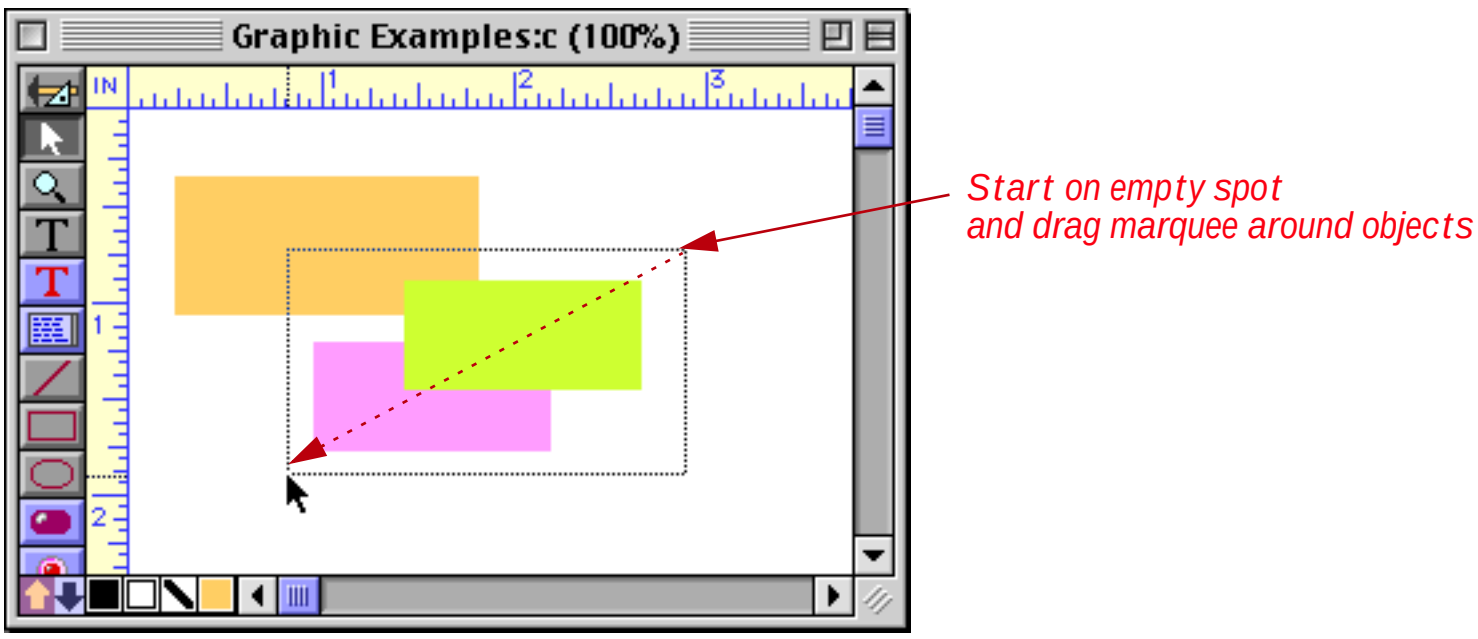


It's possible for one object to be hidden behind another object, making it impossible to click on. See "Selecting a Completely Hidden Object" on page 570 to learn how to select hidden objects.

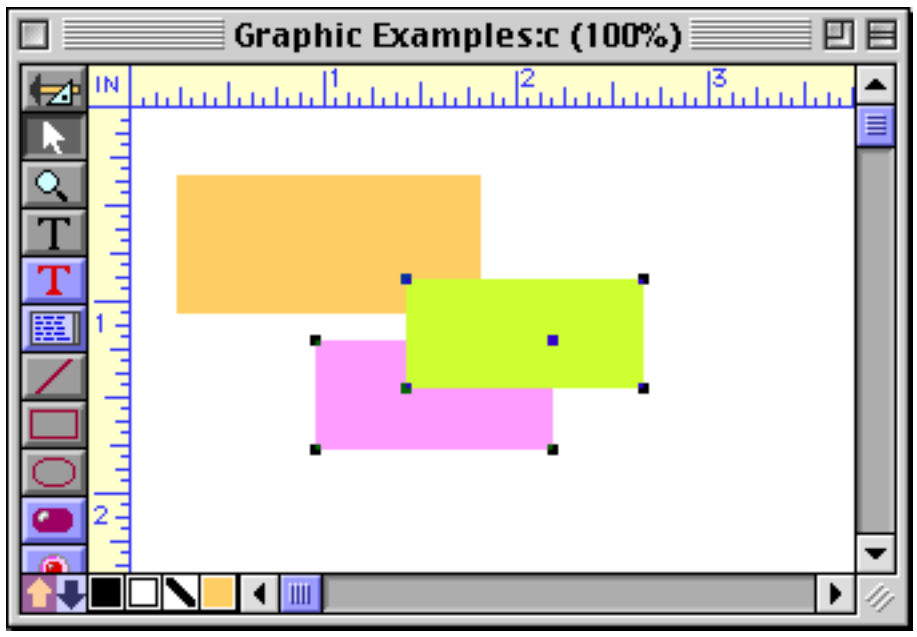
Selecting Multiple Objects at Once

Sometimes you may want to modify several objects at a time. You can select multiple objects by clicking on each object while holding down the **Shift** key, or by dragging a **selection marquee** around the objects. (You can also unselect an object that is already selected by holding down the **Shift** key and clicking on it.)

The selection marquee is simply a dotted line that appears when you drag the pointer across the surface of a form. The marquee is like a lasso—it selects any object that is completely enclosed within it. In this example the marquee is used to select two objects while leaving a third unselected.

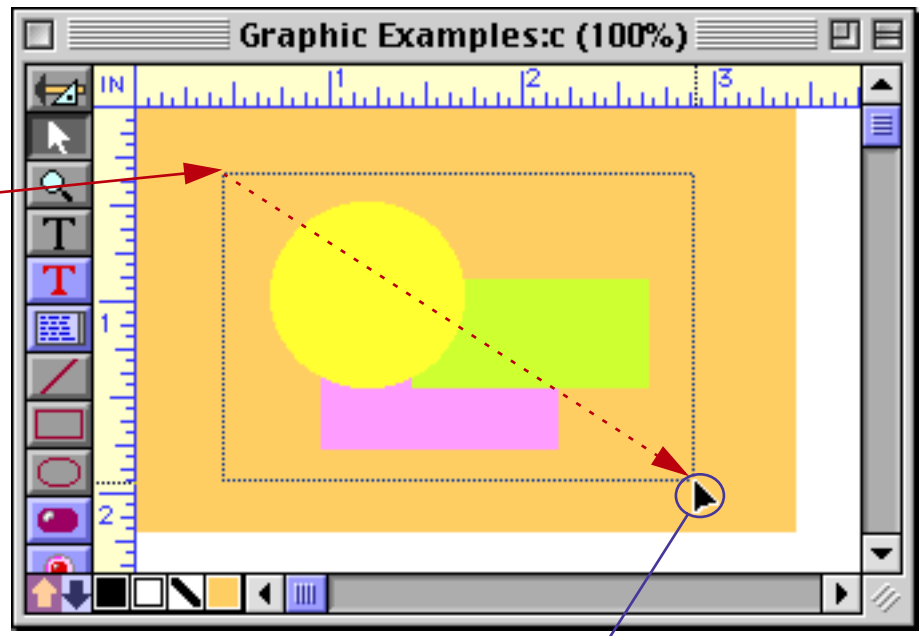


When you release the mouse the two rectangles will be selected.



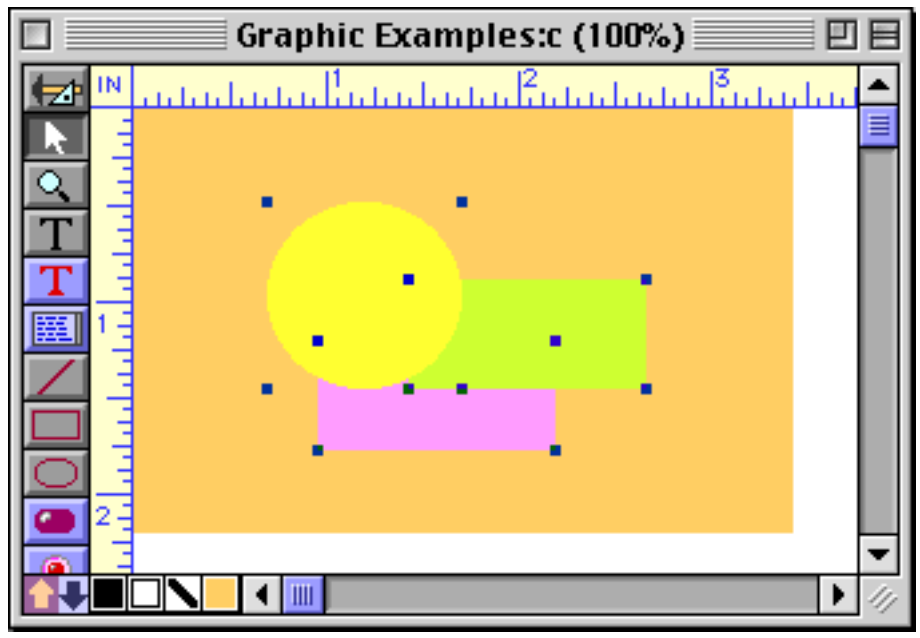
To drag a marquee you normally need to start on an empty spot on the form. If an empty spot isn't convenient, just hold down the **Space Bar** and drag the marquee. Holding down the **Space Bar** removes the stem from the cursor arrow and disables clicking on objects, allowing you to drag a marquee anywhere.Either way, drag the marquee all the way around the objects you want to select. Only objects that are completely inside the marquee will be selected. In this example we'll select the three smaller objects but not the large orange rectangle.

Hold down Space Bar and click anywhere to start dragging, even on top of an object

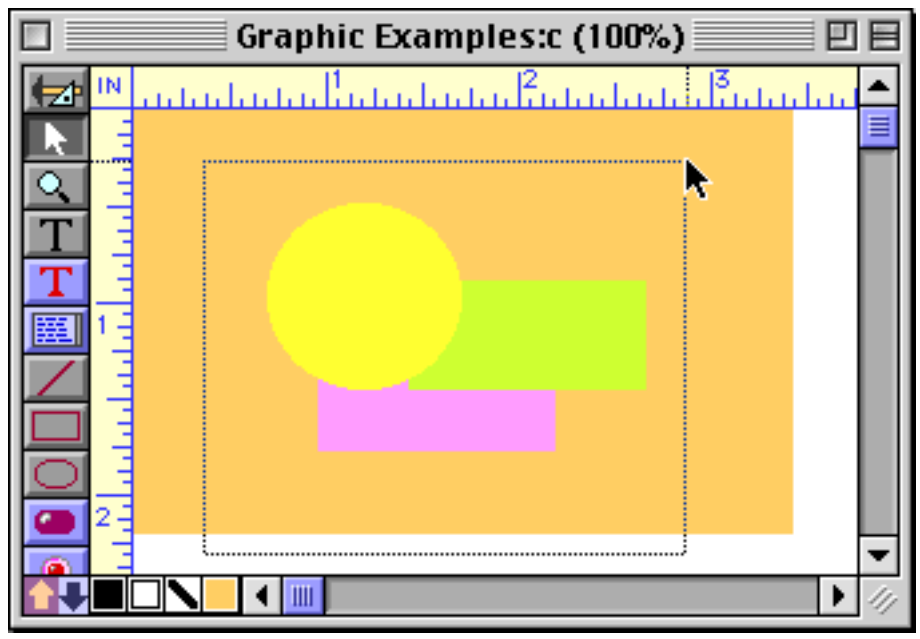


When the Space Bar is pressed, the cursor arrow loses it's tail

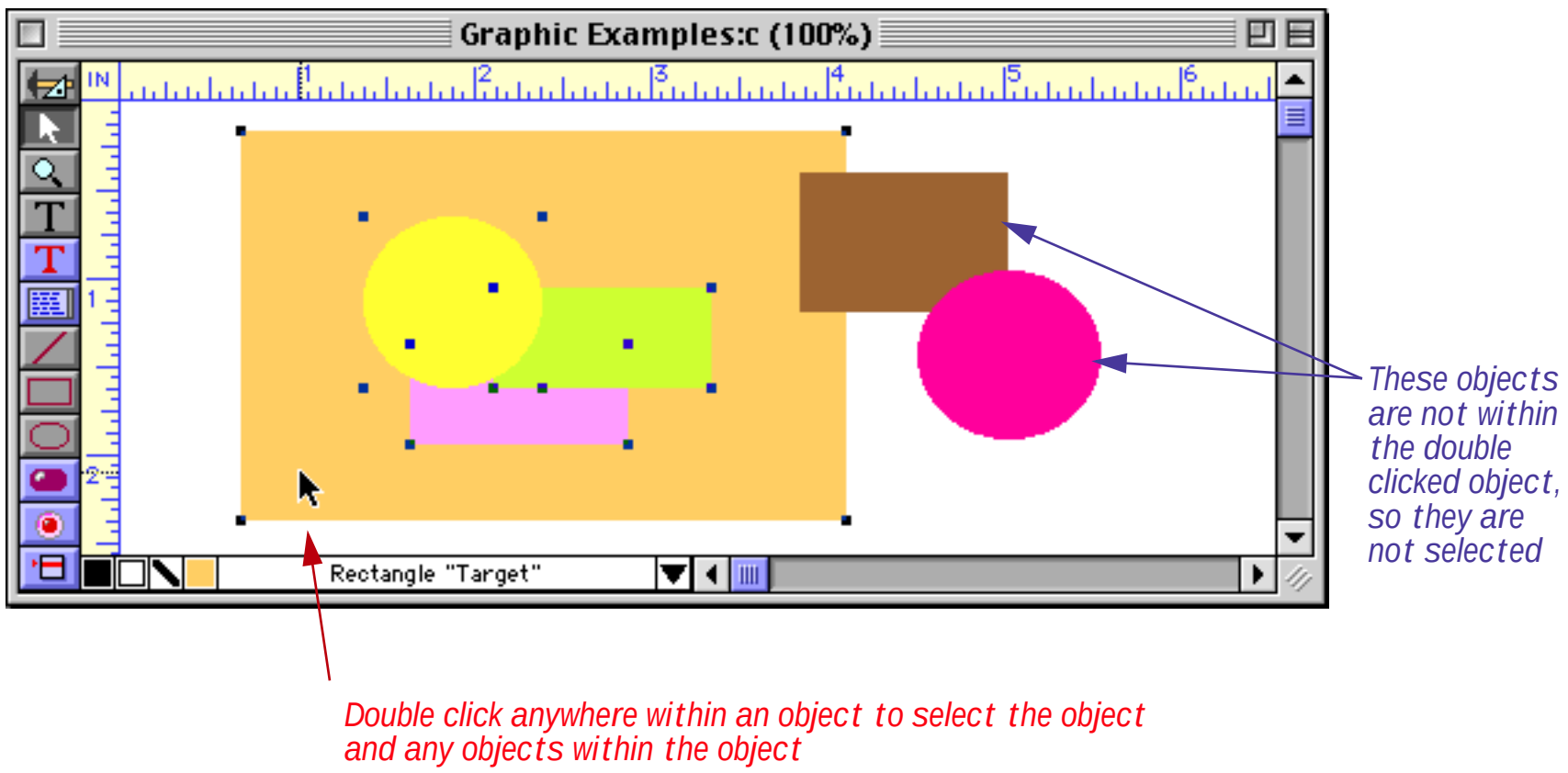
When you release the mouse, the three objects inside are selected. The large orange rectangle is not selected and has not moved.



Of course this is not the only way to select these three objects. You could hold down the **Shift** key while you clicked on each object, or in this case you could start dragging from the empty area at the bottom or right, like this.



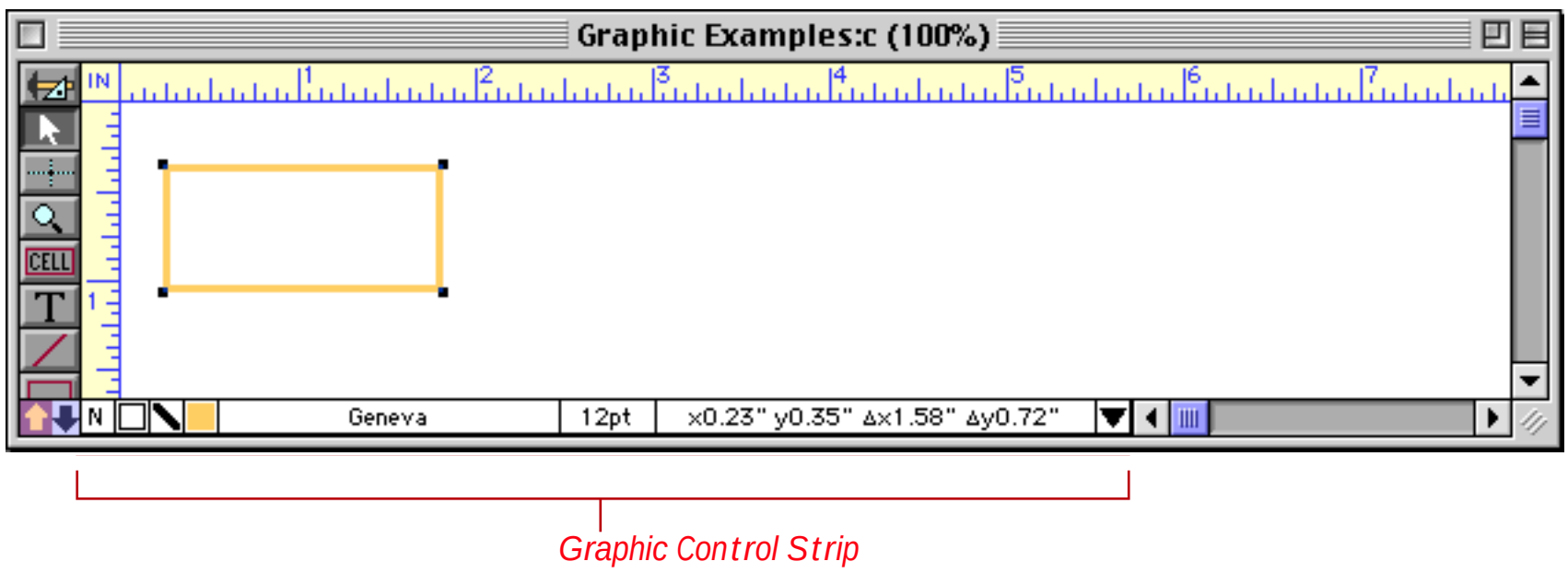
Double-clicking is another shortcut for selecting multiple objects. Double-clicking on an object selects all the objects inside the object as well as the object itself.



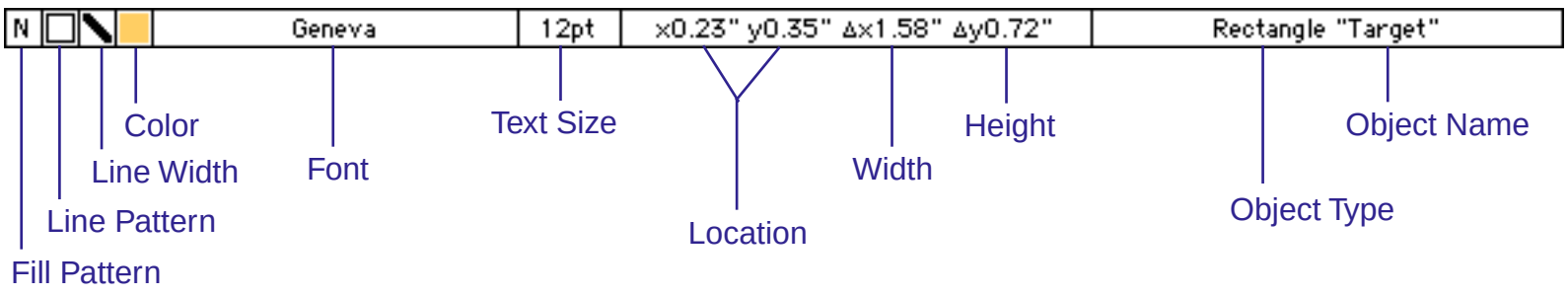
There's one more way to select multiple objects. The **Select All Objects** command in the Edit menu will select every object in the form. (To unselect all objects, click on an empty spot within the form.)

The Graphic Control Strip

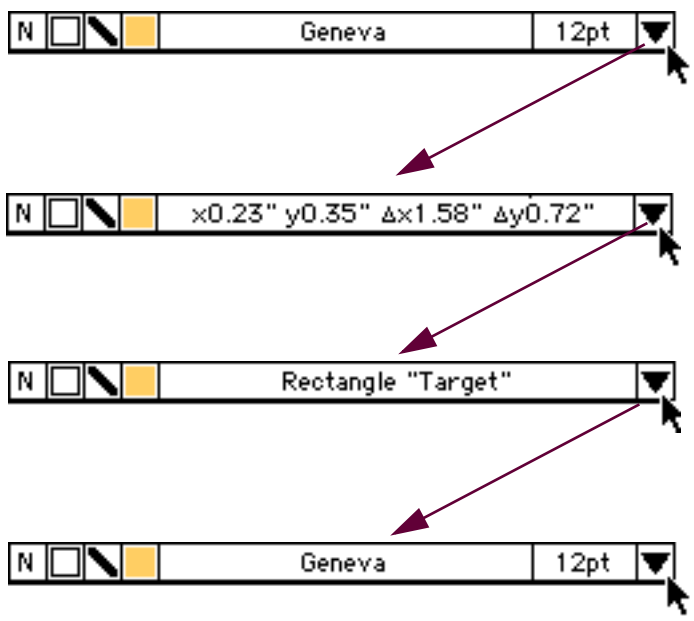
When a Panorama form window is in graphic design mode, a **graphic control strip** usually appears along the bottom of the window. The graphic control strip occupies some of the space that is normally used by the horizontal scroll bar. (If the window is too narrow for both the horizontal scroll bar and the graphic control strip, the control strip will disappear.)



The graphic control strip displays information about the currently selected object or objects (if any), and also allows you to easily change some of the properties of the currently selected objects with pop-up menus and dialogs. The complete graphic control strip has eleven elements.



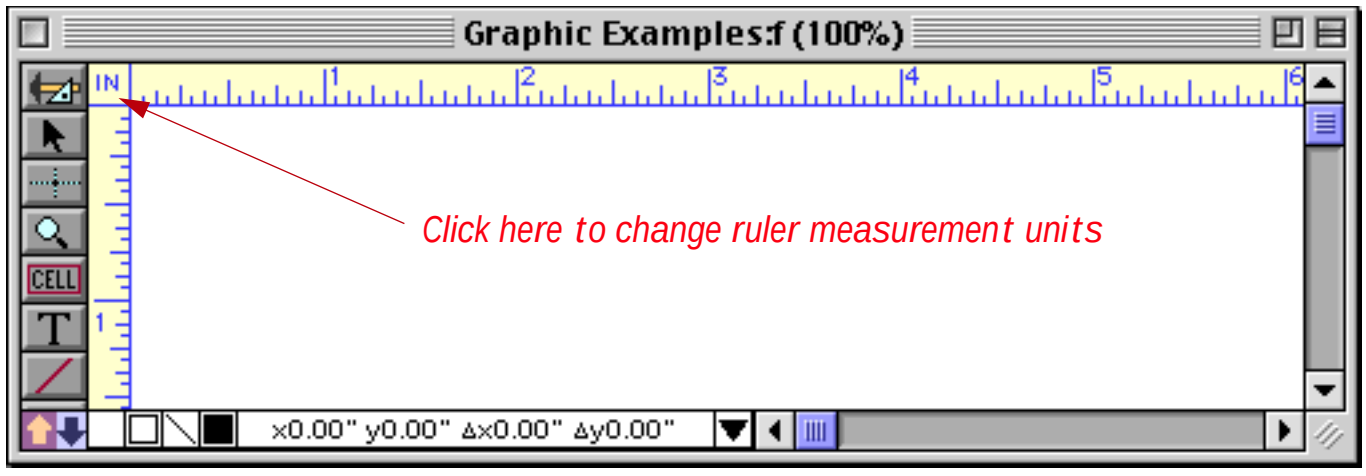
If the window is not wide enough for all seven of these elements, the control strip will automatically adjust to show fewer elements. When this happens, an extra triangle icon appears at the end of the control strip. Clicking on this icon cycles through the **Font/Text Size**, **Dimension**, and **Object Type/Object Name** control strip elements (the first four elements are always visible unless the window is extremely narrow).



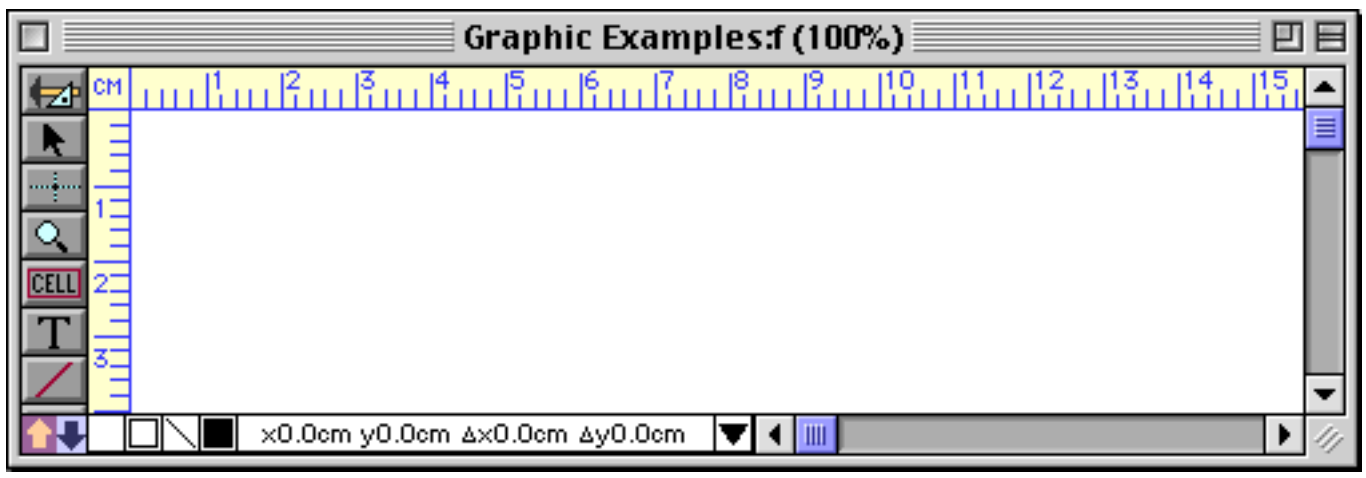
Rulers

The graphic editor always displays rulers along the top and left sides of the graphic window. The rulers always start from zero in the upper left hand corner of the entire form (not the window). An indicator in each ruler follows the mouse as you move it across the form. These indicators help you measure objects. Note: The rulers do not represent where an object will be printed on a piece of paper. They are only a convenience for sizing and positioning objects on the screen.

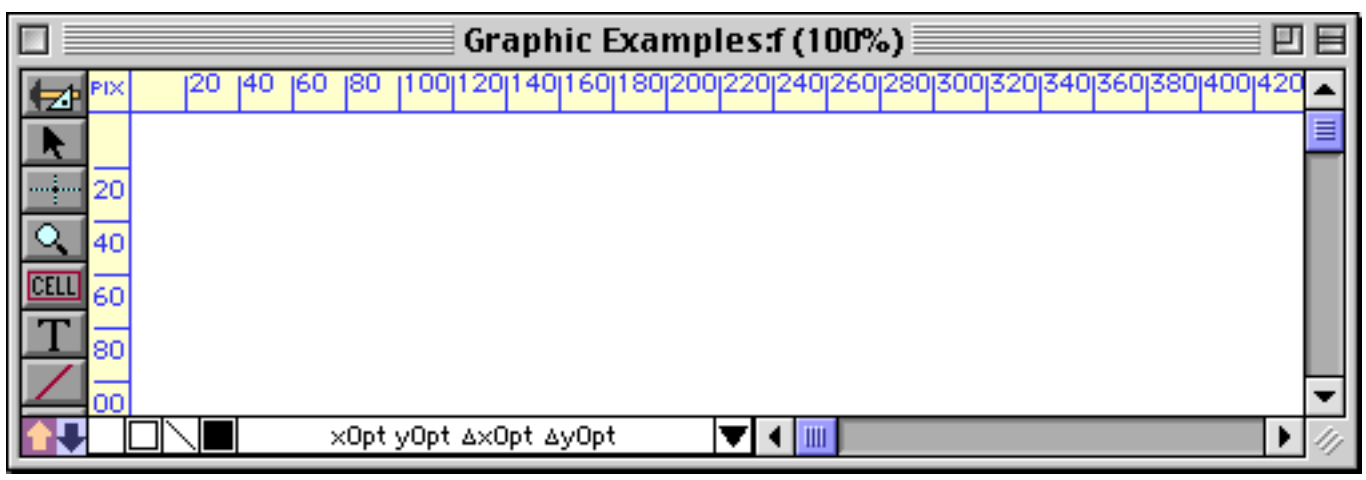
The rulers usually show measurements in inches. Click on the box in the upper left corner to toggle between different measurement units—inches, centimeters, pixels, pica and elite.



Clicking once changes the ruler to centimeters.



A second click changes the ruler to pixels. A pixel is one screen dot, or 1⁄72 inch.



Pica (1⁄6 inch) and elite (1⁄12 inch) rulers can be handy for designing graphics that need to be overlaid on pre-printed forms.

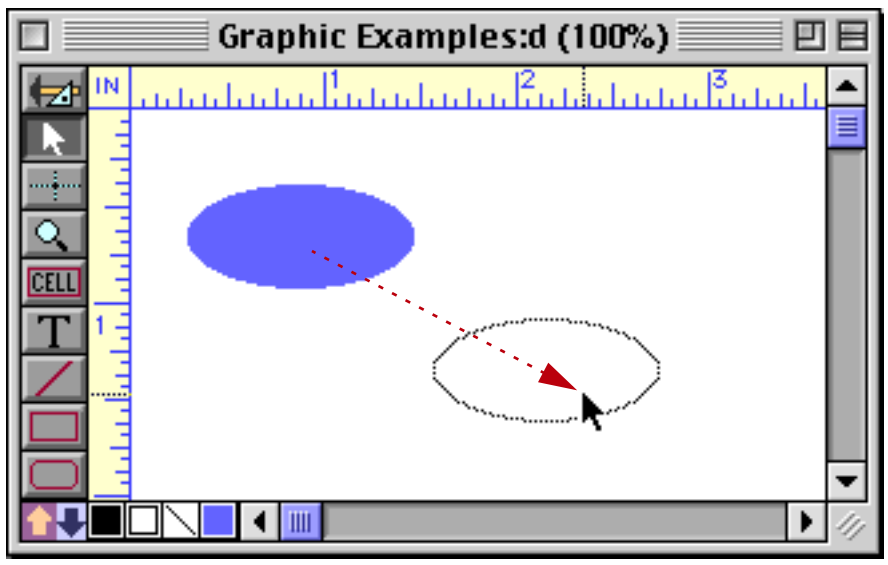
You can also change the measurement units with the **Form Preferences** dialog in the Setup menu.



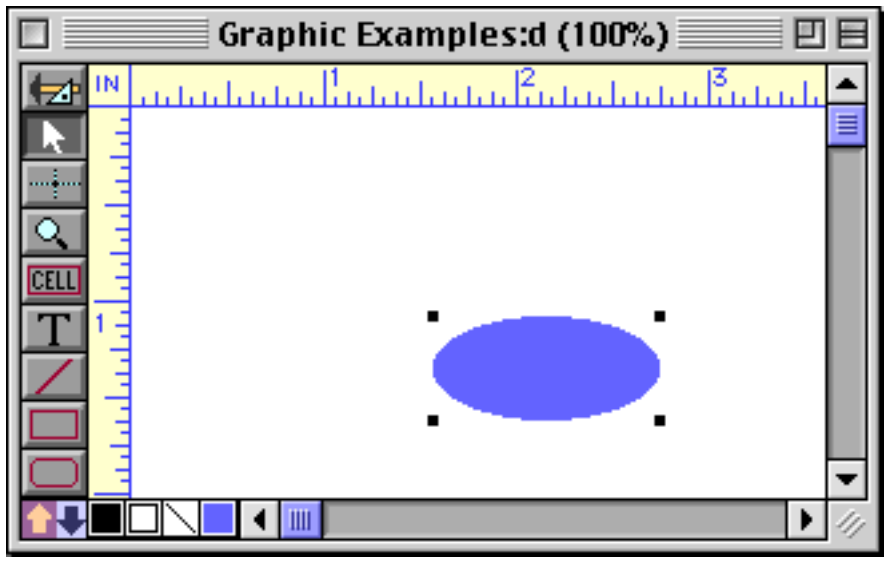
Use the **Form Preferences** dialog if you want to permanently change the default measurement units for this form.

Moving a Single Object

There are several ways to move a single object, including dragging, nudging, and using the dimensions dialog. To drag an object, the **Pointer** tool must be highlighted. Press on the object, then drag the object to its new position. If you drag the object near the edge of the window, the form will automatically scroll.

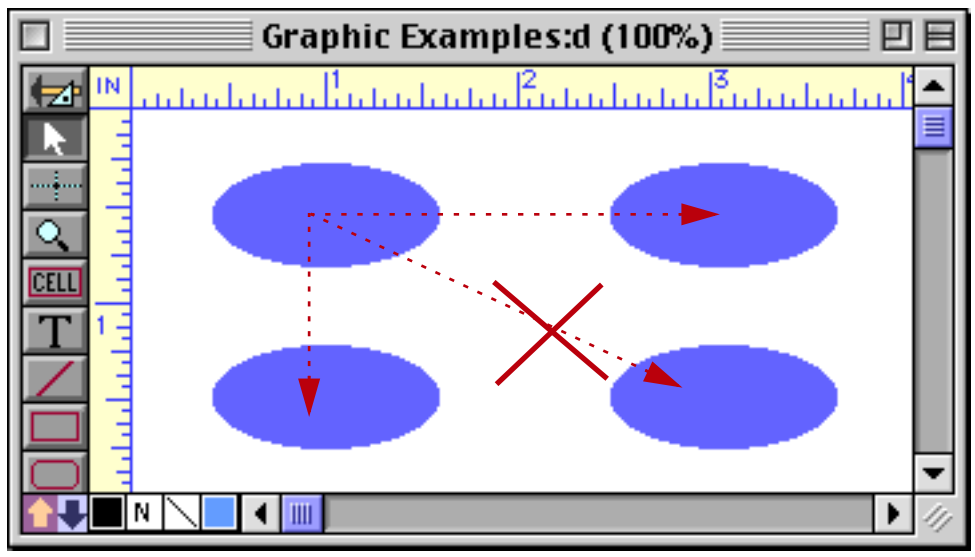


When you release the mouse, the object will move to the new position.



If an object is hollow (transparent, or filled with NONE), you must click on the border of the object to drag it.

If you want to move the object horizontally or vertically (but not diagonally), hold down the **Shift** key as you drag the object. When the **Shift** key is held down you won't be able to drag an object diagonally, as shown below.

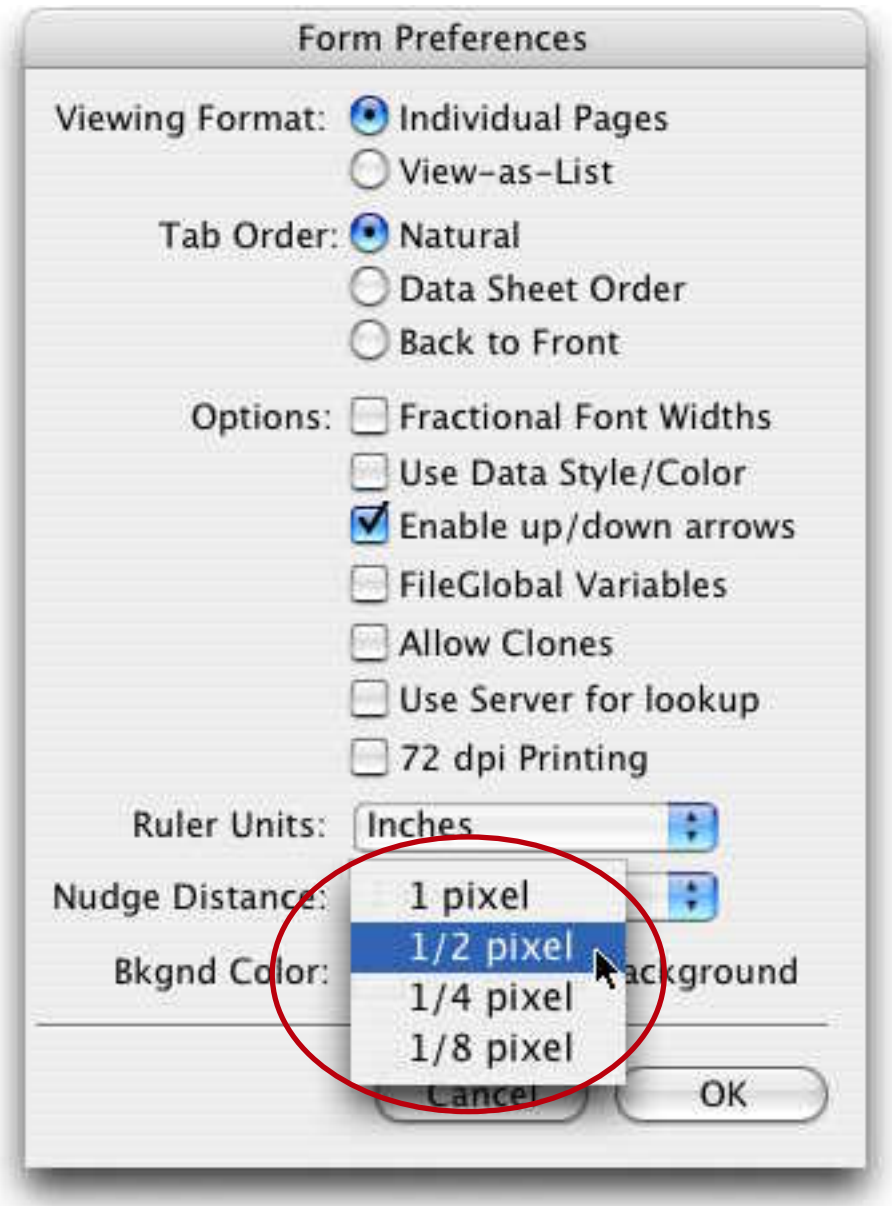


If you don't have far to go you might consider nudging the object instead of using the mouse (see below). This allows you to exactly position the object or handle in one pixel (or less) increments.

Nudging an Object (or Objects)

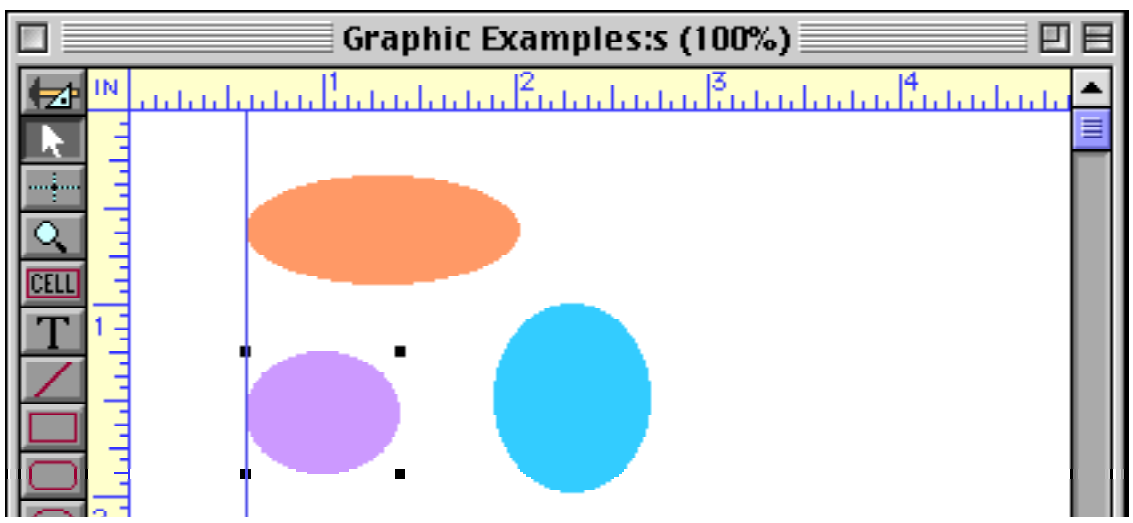
You can use the arrow keys (←, →, ↓, ↑) to nudge selected objects into position. Each time you press an arrow key, the object (or objects) moves one pixel in the direction of the arrow.

For even finer adjustments you can reduce the nudge distance using the **Forms Preferences** dialog. (Setup menu). Using this dialog you can set the nudge distance to 1⁄8, 1⁄4, 1⁄2, or 1 pixel.

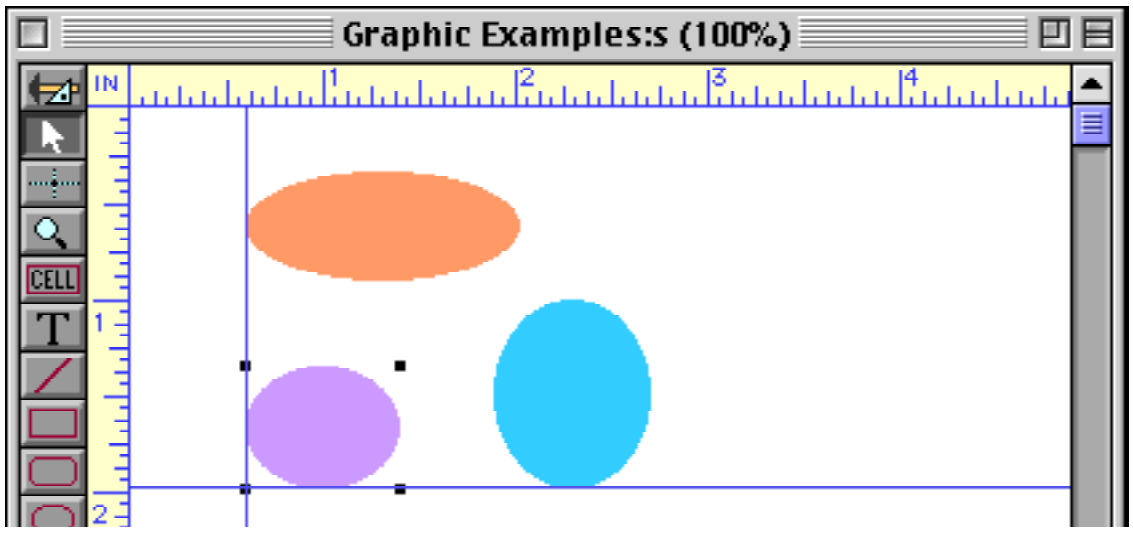


Nudge “Auto Guides”

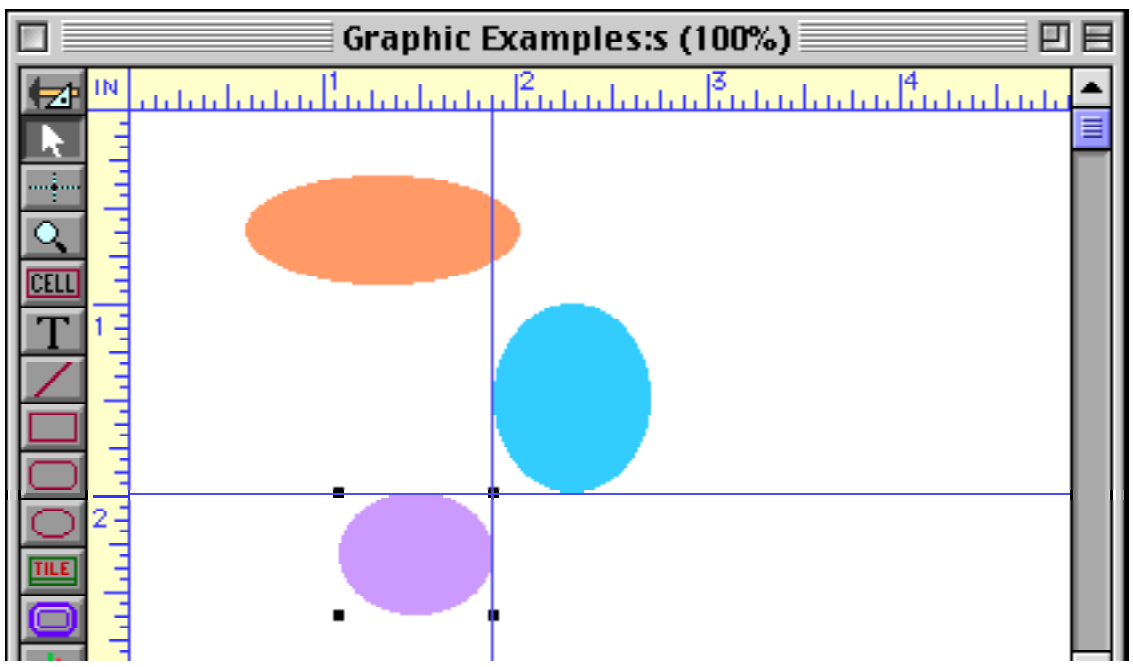
As you nudge an object (or objects) Panorama checks to see if the object is aligned with any other objects on the form. When an alignment occurs a blue guide line briefly appears. In this illustration the guide line appeared as the purple oval was nudged to the left, the guide automatically appeared when the left edge of the purple oval was aligned with the left edge of the pink oval.



The automatic guide will disappear when you click the mouse or press any key, or it will simply disappear by itself after a few seconds. If more than one edge is aligned the multiple guide lines will appear, like this.



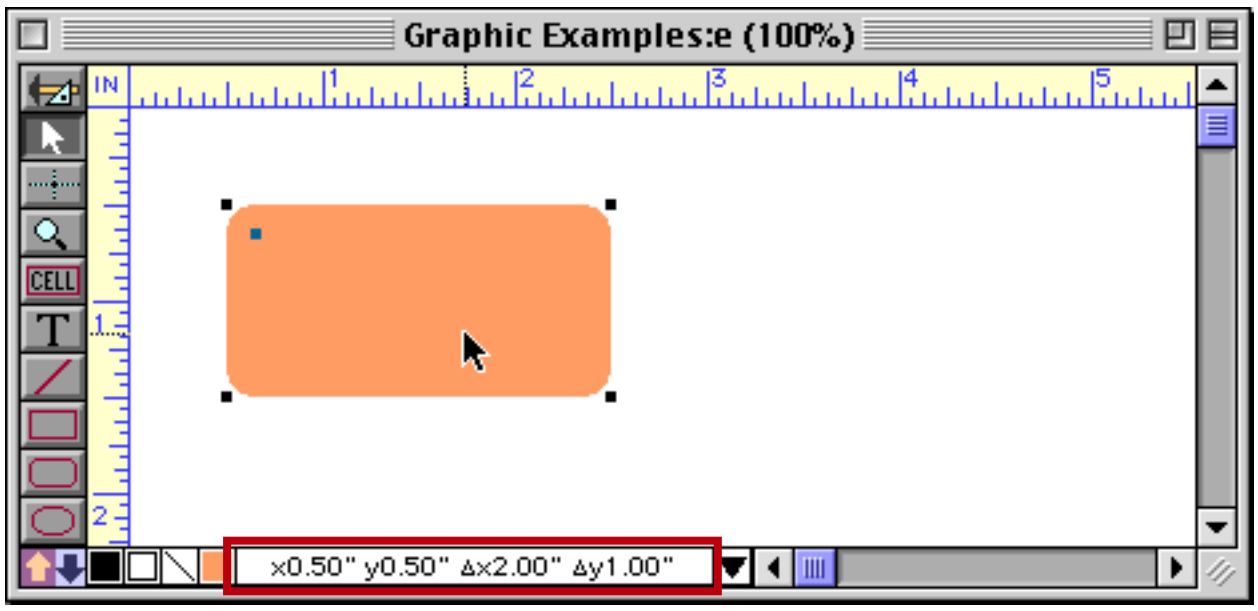
The alignment doesn't have to be top to top or left-to-left, if any edge of the nudged object(s) aligns with any edge of any other object the guides will appear.



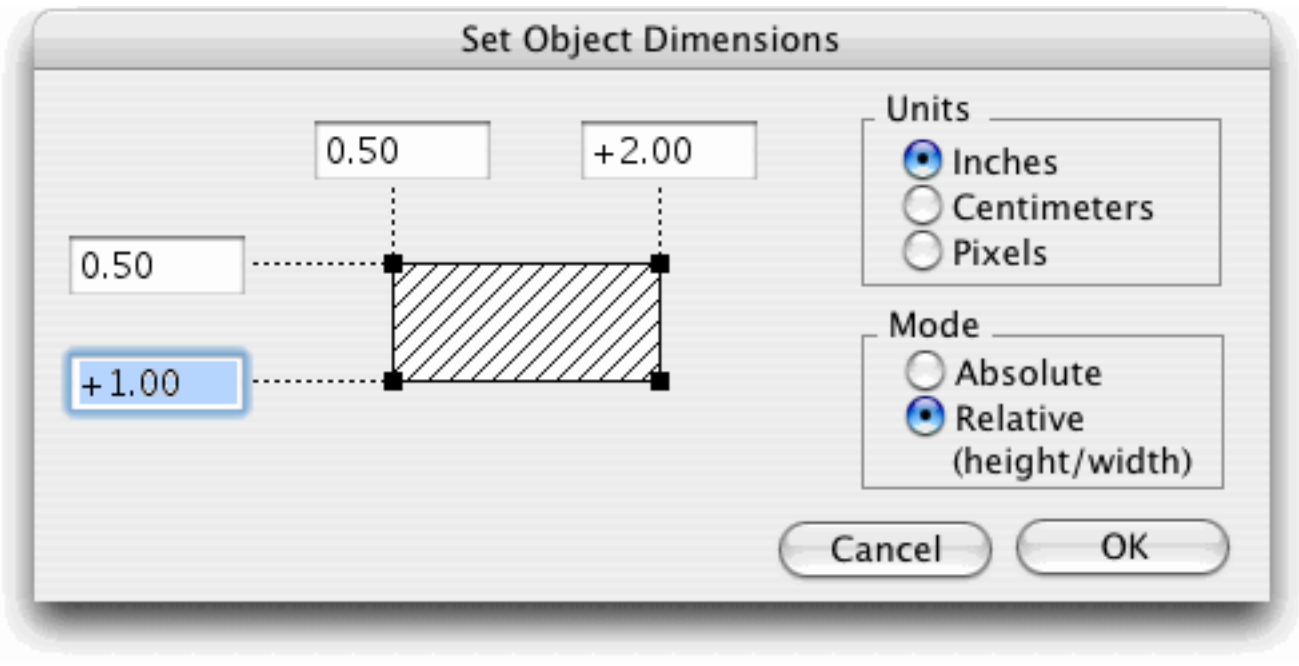
The guides can also appear when nudging an object's size, see "Nudge Size "Auto Guides"" on page 514.

Viewing and Setting Exact Object Dimensions

If the window is wide enough, the Graphic Control Strip will show the exact location and size of the currently selected object. (This is only valid if a single object is selected — it does not reliably display the location or size of multiple objects.) To see the exact location and size of any object, simply click on the object.



The Dimensions dialog allows you to display and change the exact dimensions of any object. To use this dialog, simply select an object and click on the dimensions in the Graphic Control Strip (you can also choose **Dimensions** from the Edit menu).

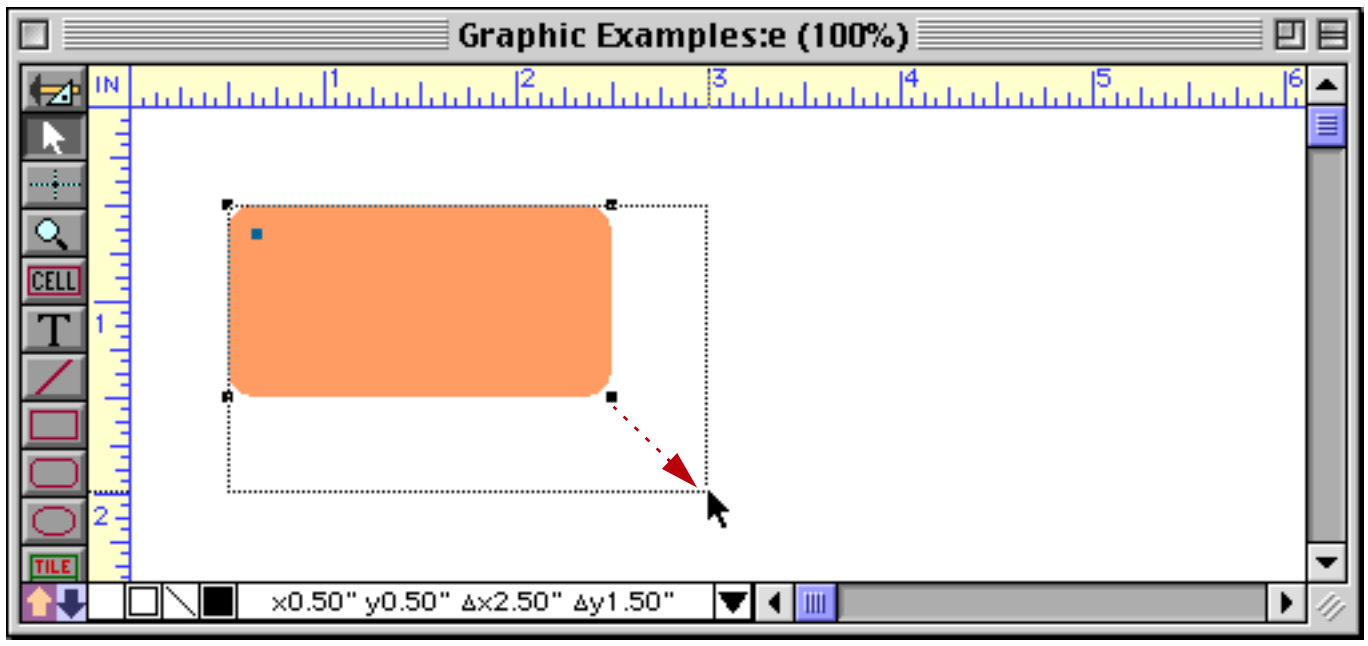


The **Dimensions** dialog gives you the choice of absolute or relative dimensions. **Absolute dimensions** display the position of all four corners of the object, that is each corner's position from the top left corner of the form. **Relative dimensions** display position of the top left corner of the object along with the size of the object; that is, the relative distance from the top left corner to the bottom right corner. Use relative dimensions when you want to move an object without changing its size, or change the size of an object without moving it. (Note: When using Relative dimensions, the object's height and width must have a + symbol in front of the number, as shown above.)

The **Dimensions** dialog can work with dimensions in inches, centimeters, or pixels. The dialog will default to the current ruler measurement units. (See "Rulers" on page 506 to learn how to set the ruler units.) Dimensions in inches or centimeters will be rounded to the nearest $^{1}/_{576}$ inch (0.017 inch).

Changing the Size of a Single Object

To change the size of an object, first select the object with the **Pointer** tool. Then use the mouse to drag one of the corner handles. As you drag the handle, an outline of the object will follow the mouse. Release the mouse when the corner is in the correct spot.

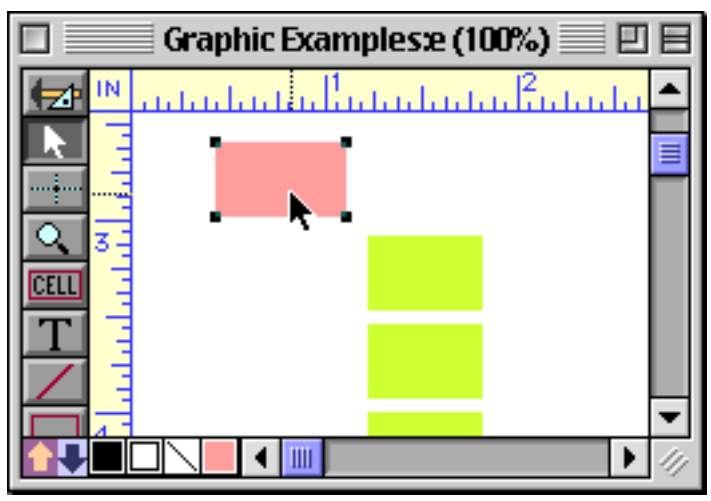


If you want to change the width or height of an object (but not both at once), hold down the **Shift** key while you change the size. Holding down the **Shift** key prevents the corner from moving diagonally.

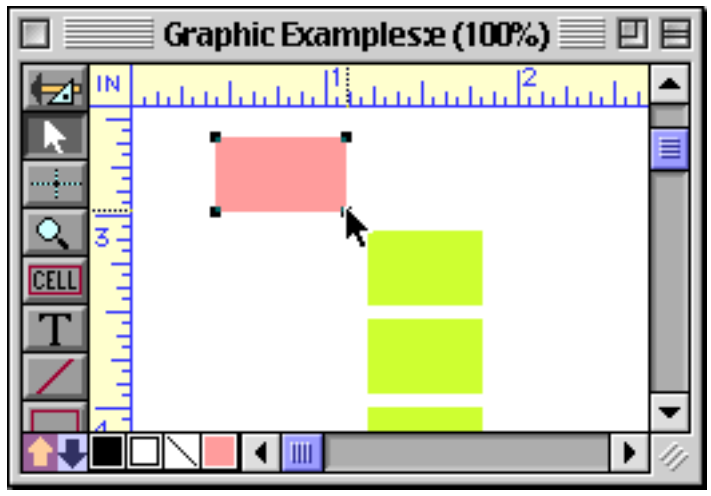
Nudging the Size of an Object

The arrow keys (←, →, ↓, ↑) usually nudge the entire object. However, after you click or drag a handle, the arrow keys will nudge just that handle. Each time you press an arrow key the handle will move one pixel in the direction of the arrow. In other words, each time you press an arrow key the object will grow (or shrink) one pixel in the direction of the arrow (or less than a pixel if you have changed the nudge distance using the **Form Preferences** dialog).

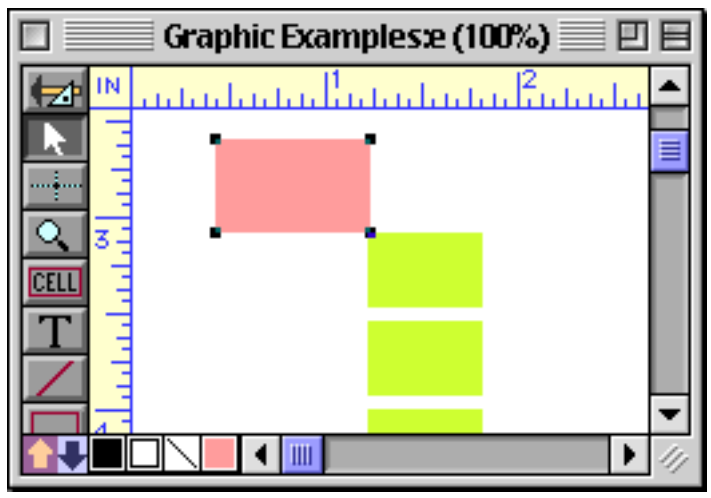
Let's look at the procedure step by step. Start by clicking on the object whose size you want to adjust.



Now click on the corner you want to adjust.



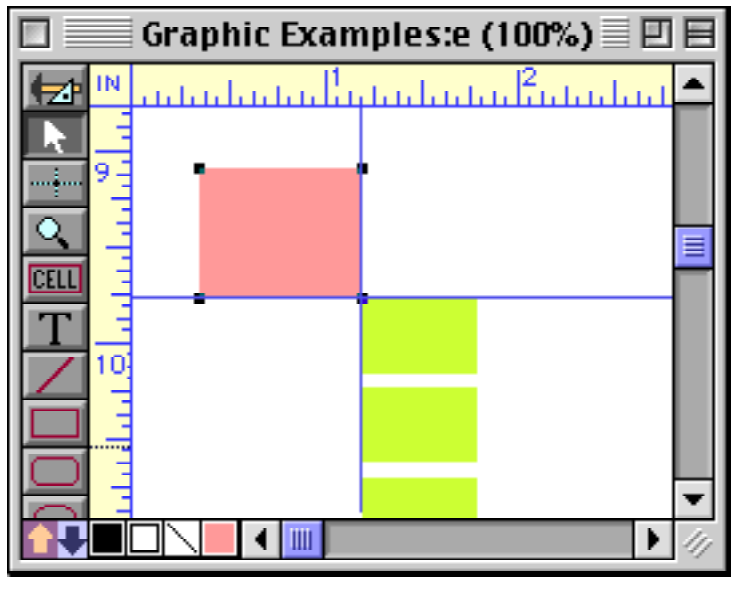
Use the arrow keys to adjust the size of the object in small increments. In this case we pressed the ↓ and → keys about half a dozen times each.



As soon as you click on another object, the arrow keys go back to nudging the entire object instead of just the corner.

Nudge Size “Auto Guides”

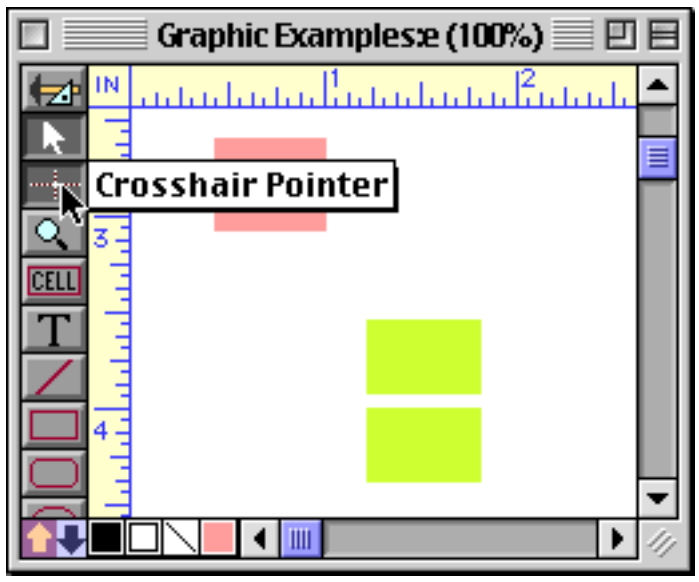
As you nudge the size of an object, Panorama checks to see if any of the edges of the resized objects are aligned with the edges of any other objects. If any edge is aligned a temporary blue guide appears. In this illustration the lower right hand corner of the pink square has been nudged, and is now aligned in two directions with the yellow rectangle.



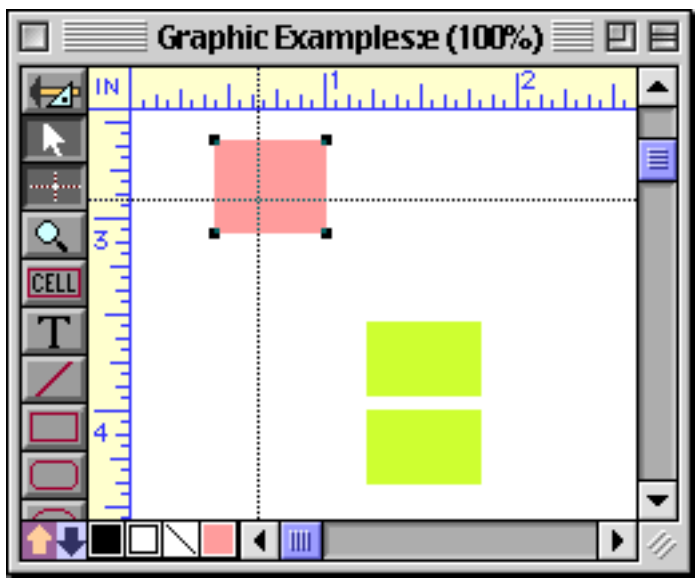
For more information on auto-guides, see “Nudge “Auto Guides”” on page 510.

Nudging to the Crosshair Cursor

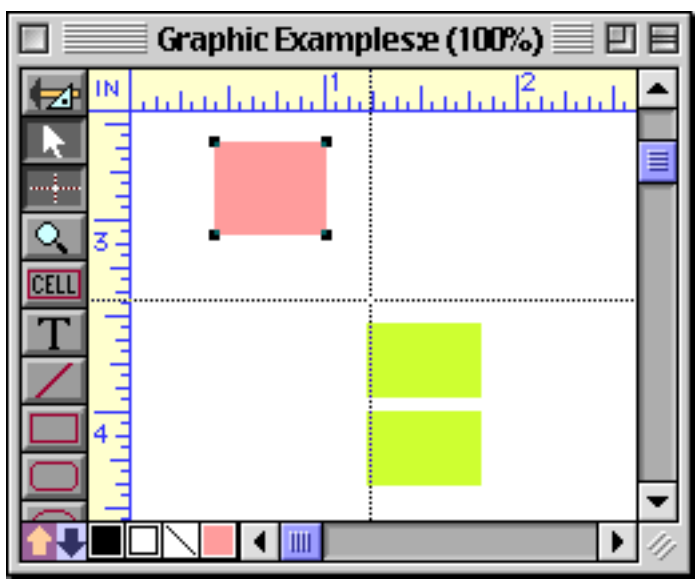
You can use the **Crosshair** cursor to help you nudge objects into alignment. (However, this technique is usually not necessary now that Panorama has automatic guides that appear when nudging (see "Nudge "Auto Guides"" on page 510) so you may want to simply skip this section.) Start by clicking on the **Crosshair** tool to turn on the crosshairs.



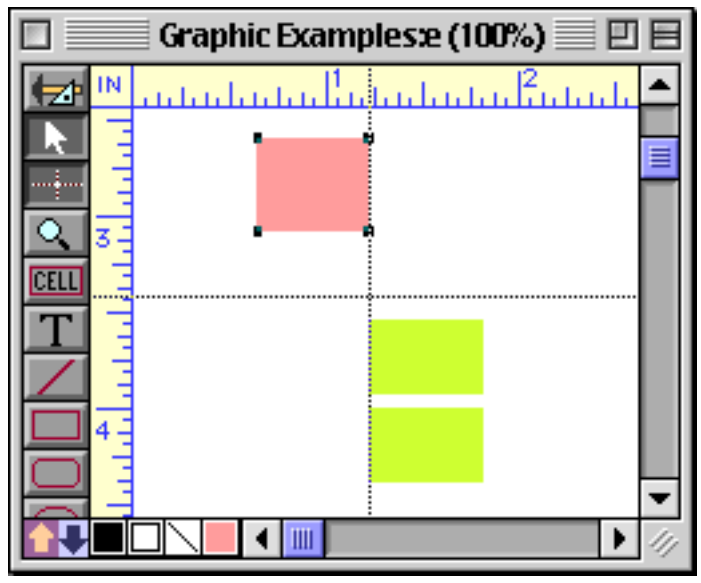
Select the object you want to nudge,



then move the crosshair to the desired spot.



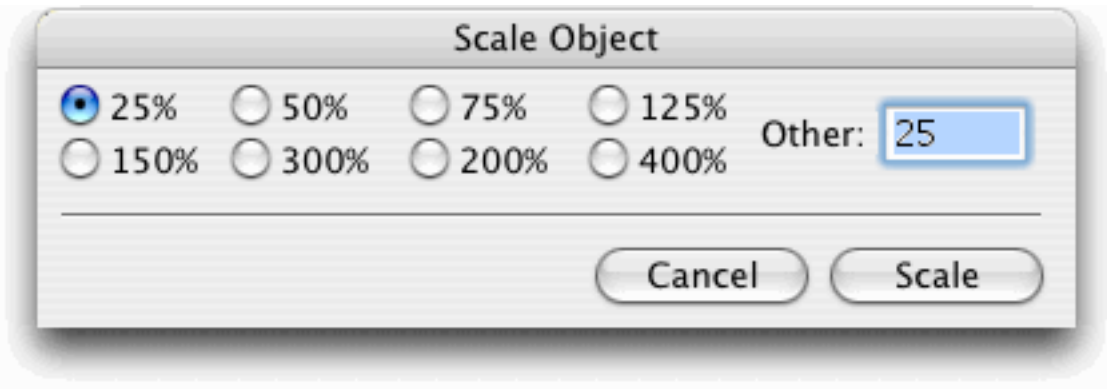
Use the arrow keys to nudge the object until it is aligned with the crosshair. In this case we nudged the entire object, but we could also adjust the size by clicking on the corner as described in the previous section.



Finally, turn off the crosshair by clicking on the **Crosshair** tool again.

Percentage Scaling

Use the **Scale** command (Arrange menu) to expand or shrink an object by an exact percentage.

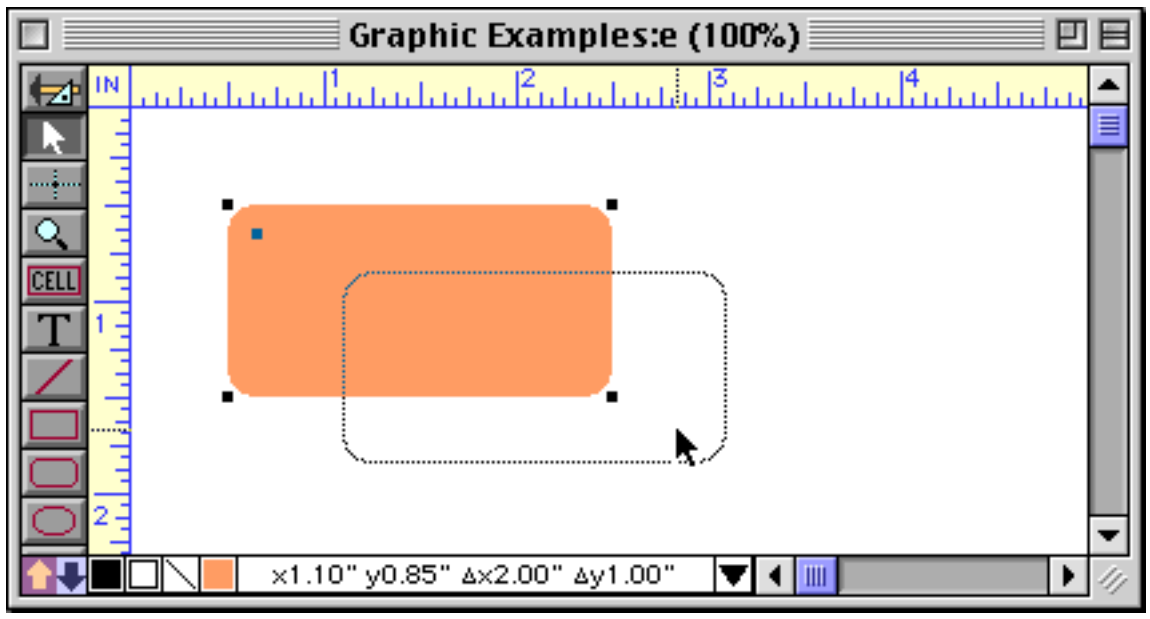


You can choose one of the pre-defined scales or type in any percentage between 1% and 999%.

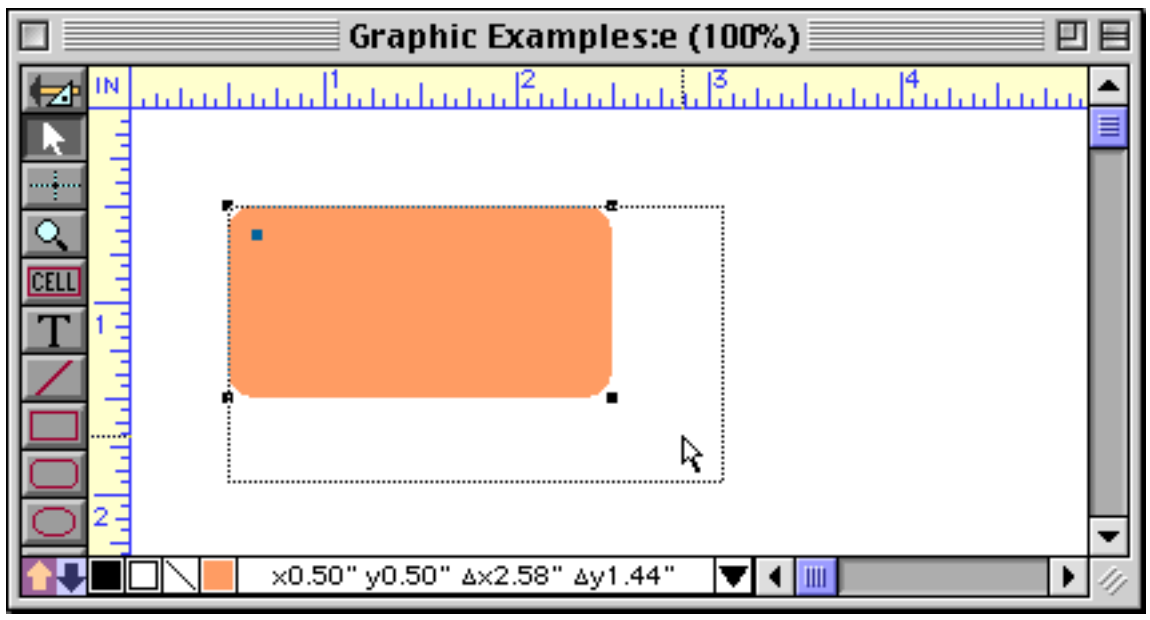
Resizing Without Handles

Dragging on the inside of an object normally moves the object. But if you hold down the S key (the letter S) while you drag, dragging on the inside of an object resizes the object—just like dragging on a handle. This feature can be very handy when you are working on a cluttered form—the handle you want may be hard to find. To remind you that the S key is pressed, the cursor turns into a hollow arrow. When you release the S key Panorama will go back to normal operation.

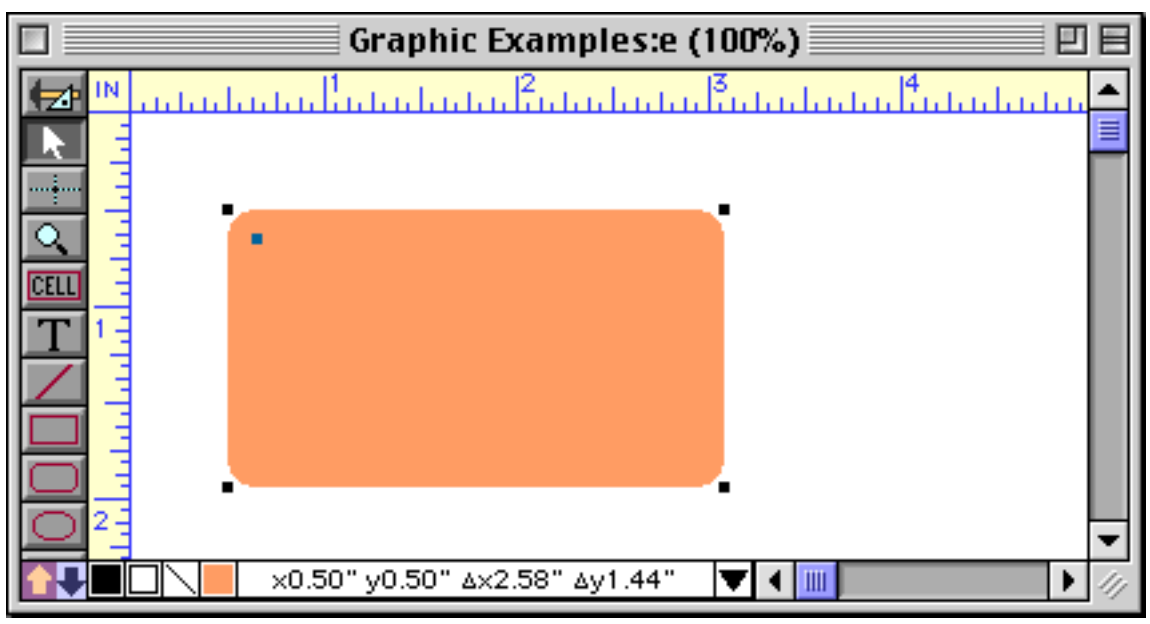
This operation is easier to see than to explain with words. Here's what happens when you drag on the object normally — it simply moves:



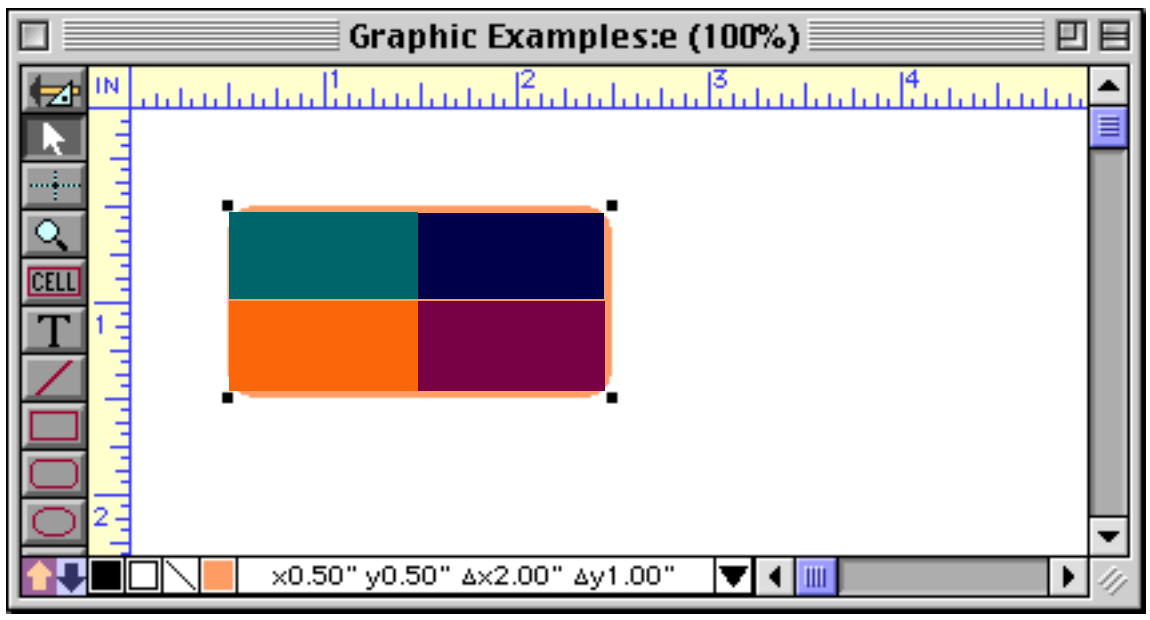
But if you hold down the S key (the letter S) while you drag, the object will change size instead of moving. (Notice the hollow mouse arrow.)



When you release the mouse the object will expand or shrink.



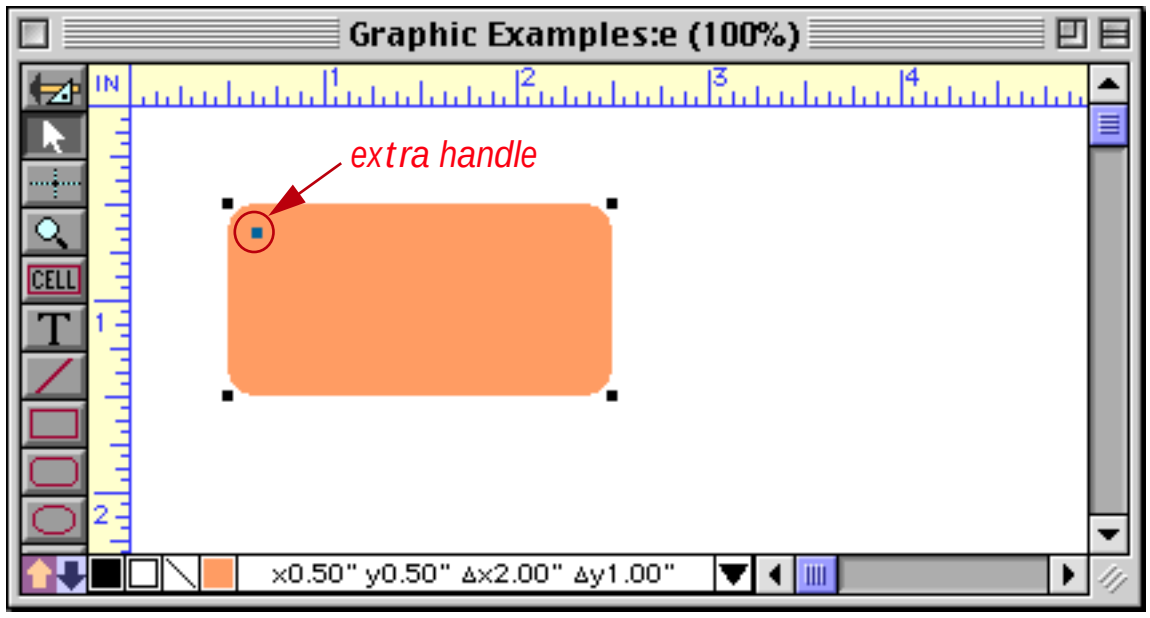
Here's another way to view the operation of the **S** key. When you press this key, Panorama behaves as if the handles at the corners of the object had expanded to fill the entire object. No matter where you click, you are clicking on a handle. Therefore, you cannot drag the object, but only resize it. The four colored rectangles in the illustration below symbolize the expanded handles (actually, they would be slightly bigger than this, so that they would cover the entire object).



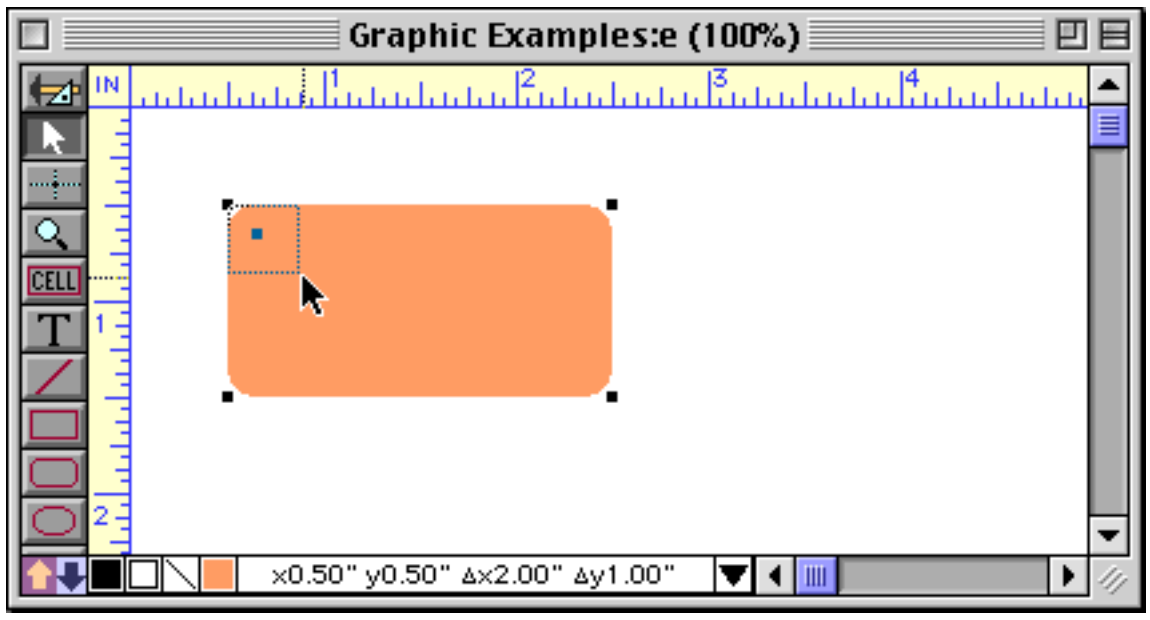
If you want to change only the width or height of the object, but not both, hold down the **Shift** key at the same time as you hold down the **S** key. This will prevent the object from changing size diagonally.

Changing the Radius of Round Corners

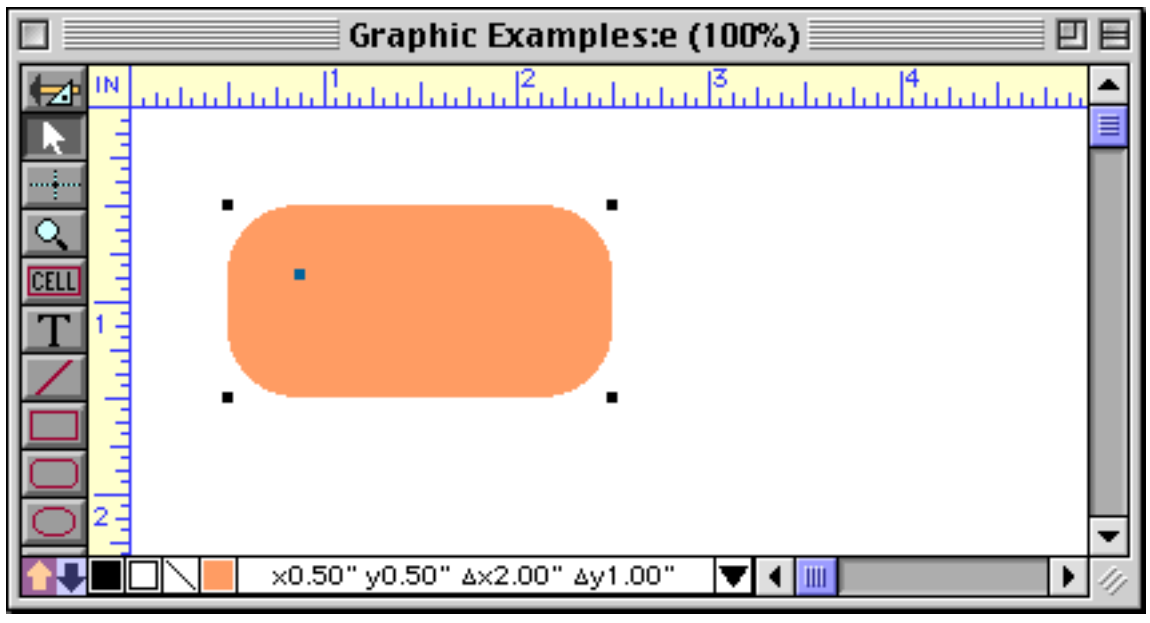
Rounded rectangles have an extra handle in the upper left hand corner.



This handle is used to adjust the radius of the corner. Drag the handle towards the center of the rectangle to increase the radius, drag it towards the corner to reduce the radius.



The new radius will appear when you release the mouse.



You can drag the corner diagonally if you want an elliptical corner. Hold down the **Shift** key if you want a circular corner.

Removing Objects

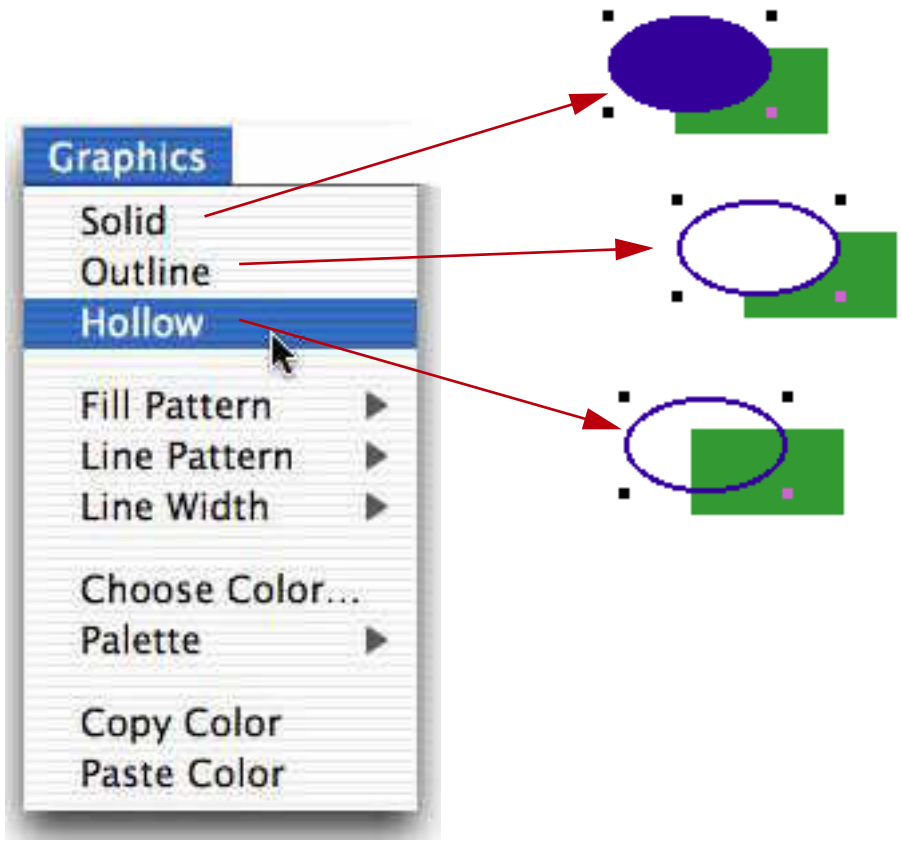
To completely remove one or more objects from the surface of the form first select the objects (See “Selecting a Single Object” on page 501 and “Selecting Multiple Objects at Once” on page 502). Then choose **Cut** or **Clear** from the Edit menu. **Cut** places the object on the clipboard so that it can be pasted back into the form in a new location. You can also remove the object by pressing the **Delete** or **Backspace** key (this is the same as choosing **Clear**).

Modifying Object Attributes

New objects are usually white with a thin black border. To customize an object you can change the fill pattern, border (pen) pattern, line thickness, and color. You can customize these with the Graphic Control Strip or the **Graphics** menu.

Solid, Outline and Hollow Objects

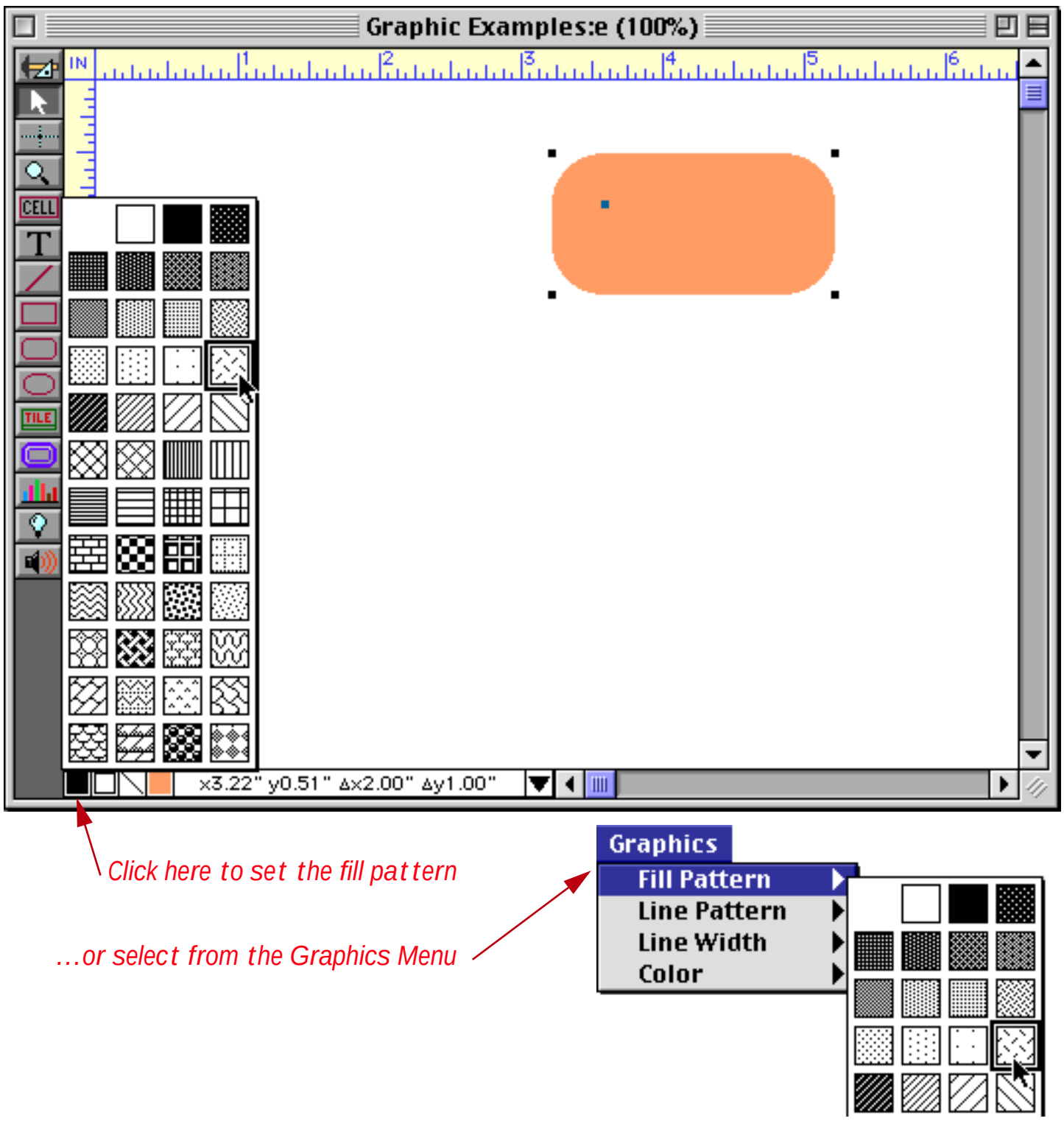
The first three choices in the Graphics menu, **Solid**, **Outline** and **Hollow**, set the fill and line patterns of the selected objects to their most common choices, as shown in the diagram below.



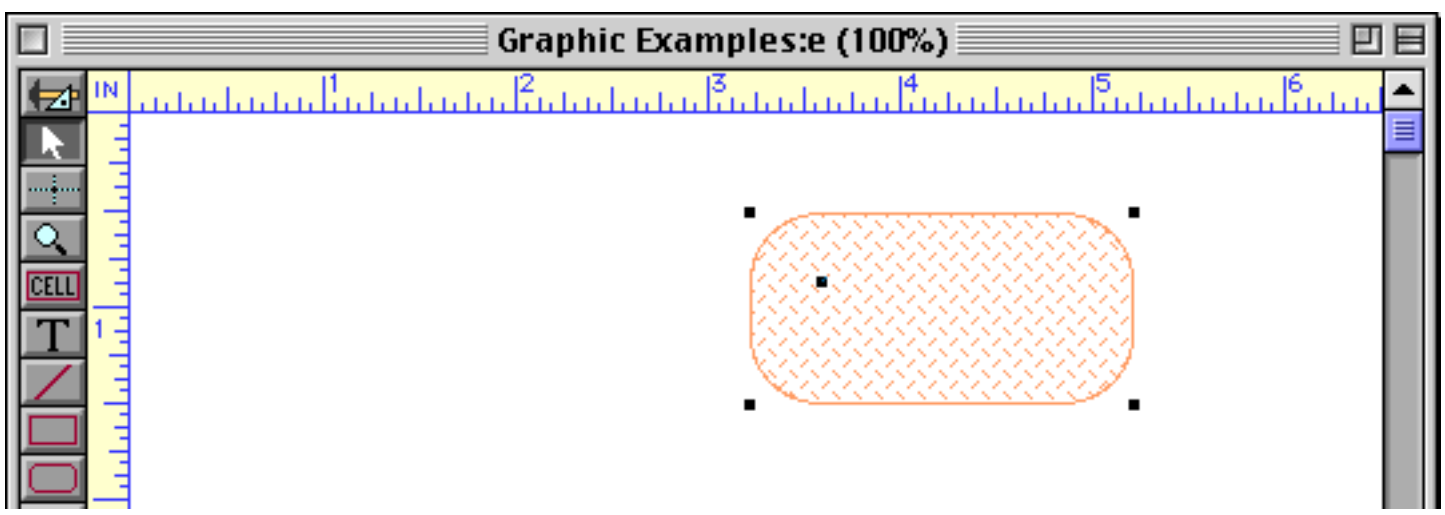
Solid objects are colored all the way through. Outline objects are filled with white with a colored border. Hollow objects have a colored border but a transparent center.

Fill Pattern

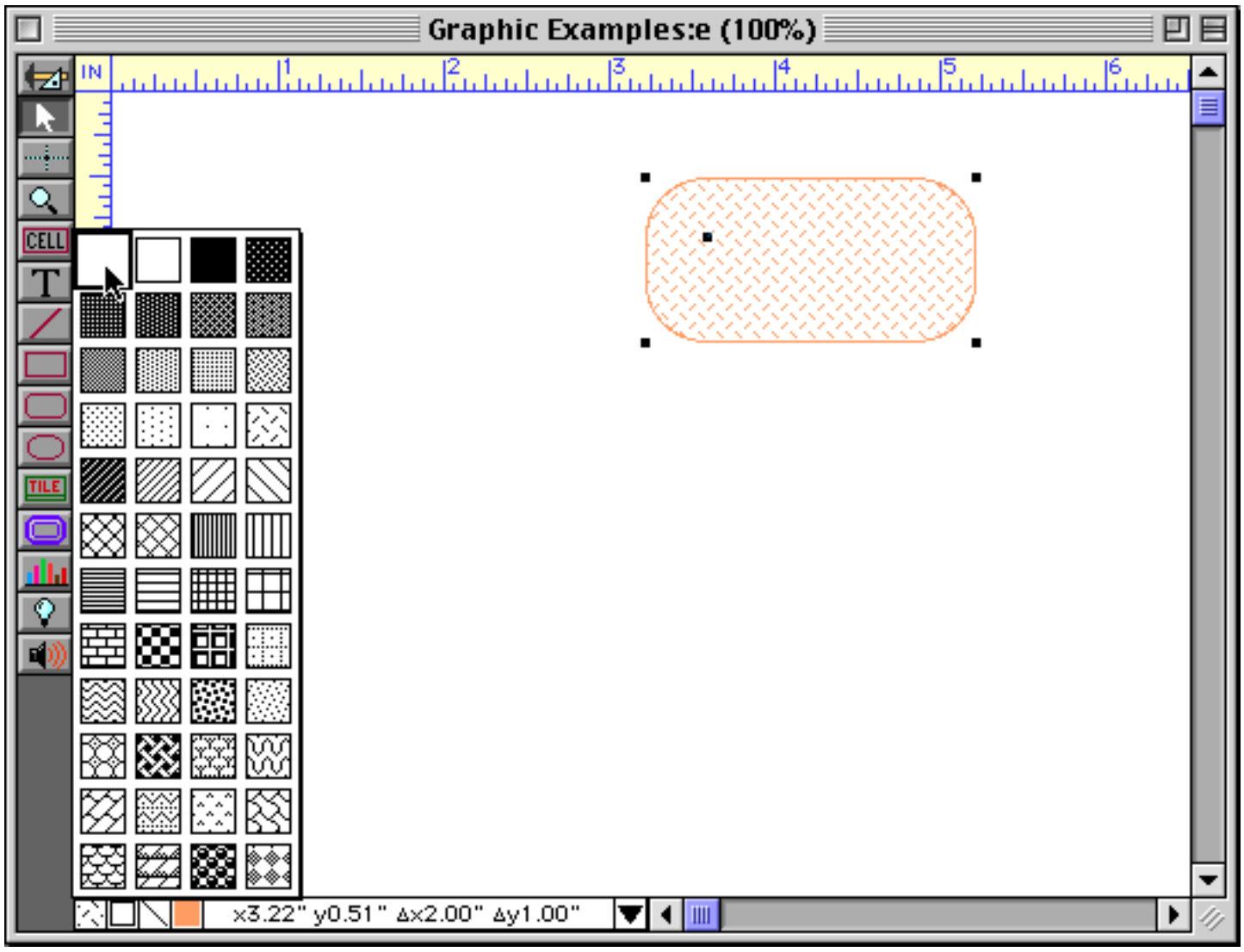
The **Fill Pattern** menu in contains 40 different fill patterns.



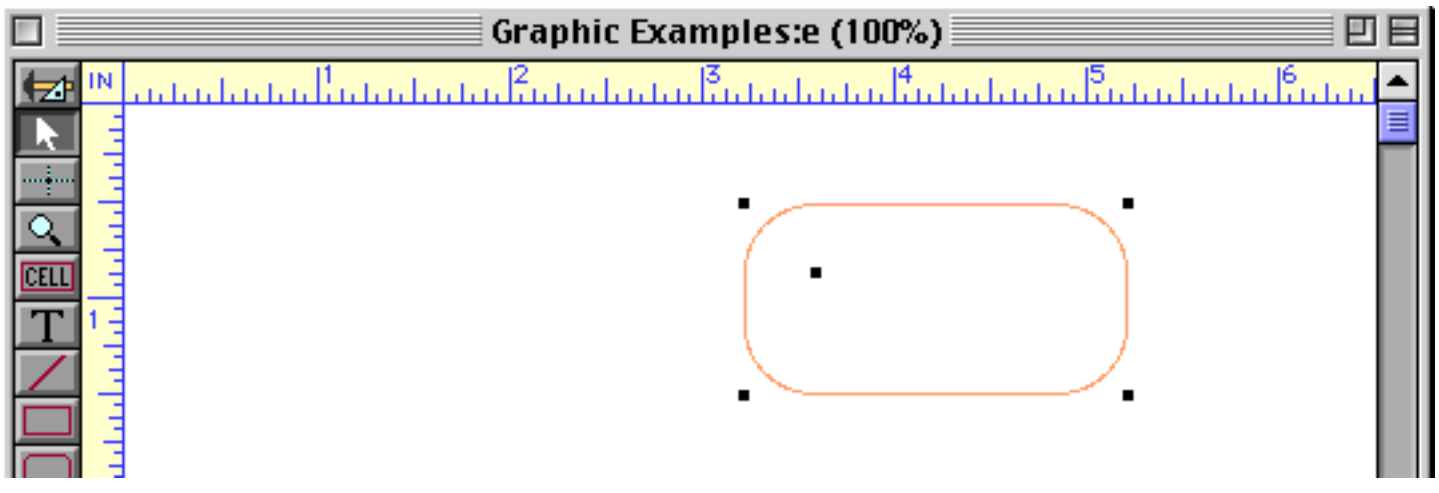
To fill the inside of an object with one of these patterns, first select the object (or objects), then choose the pattern from the **Fill Pattern** menu. When you release the mouse the object(s) pattern will change.



To make a hollow (transparent) object, choose the empty pattern in the top left corner of the Fill Pattern sub-menu.

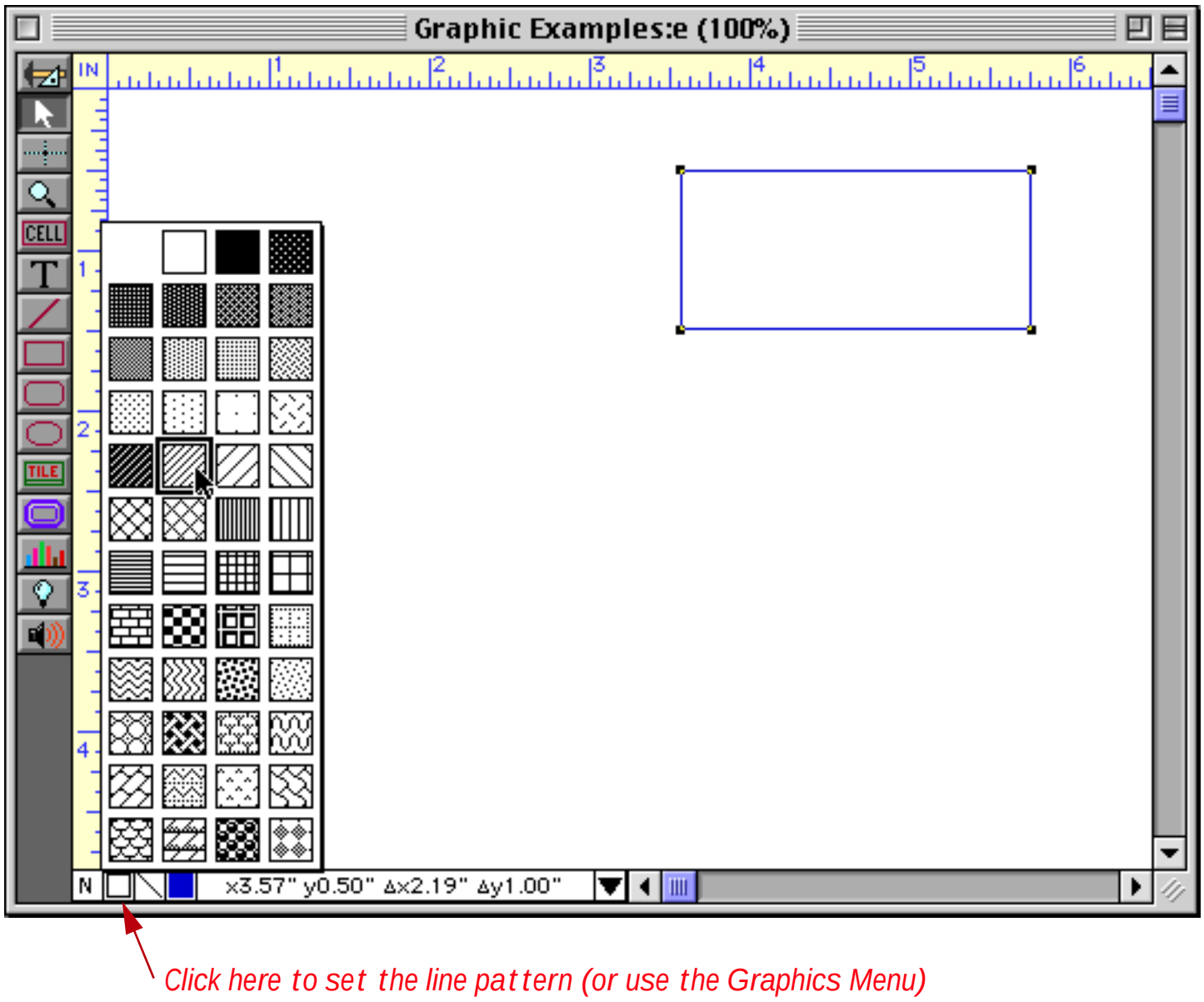


Here’s the transparent object.

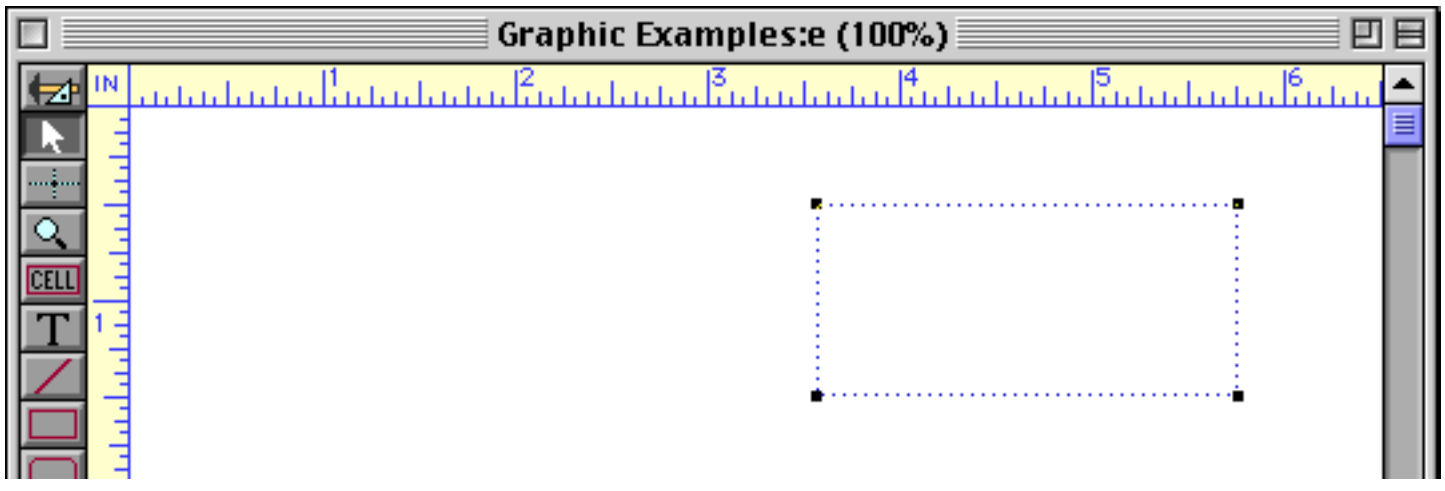


Line Pattern

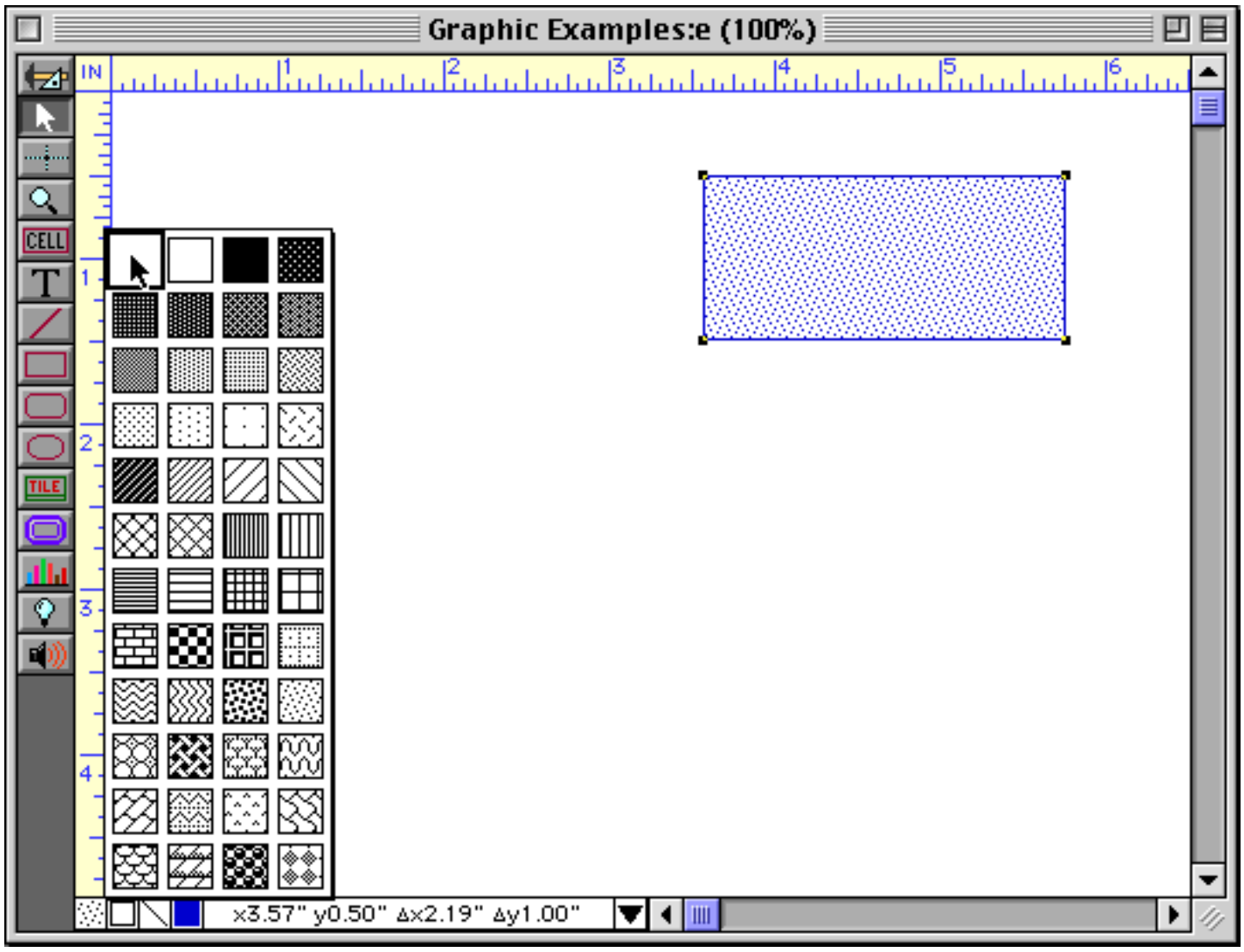
The **Line Pattern** menu contains 40 different patterns that can be used to draw lines and borders. Only a few of these patterns are really useful for drawing lines.



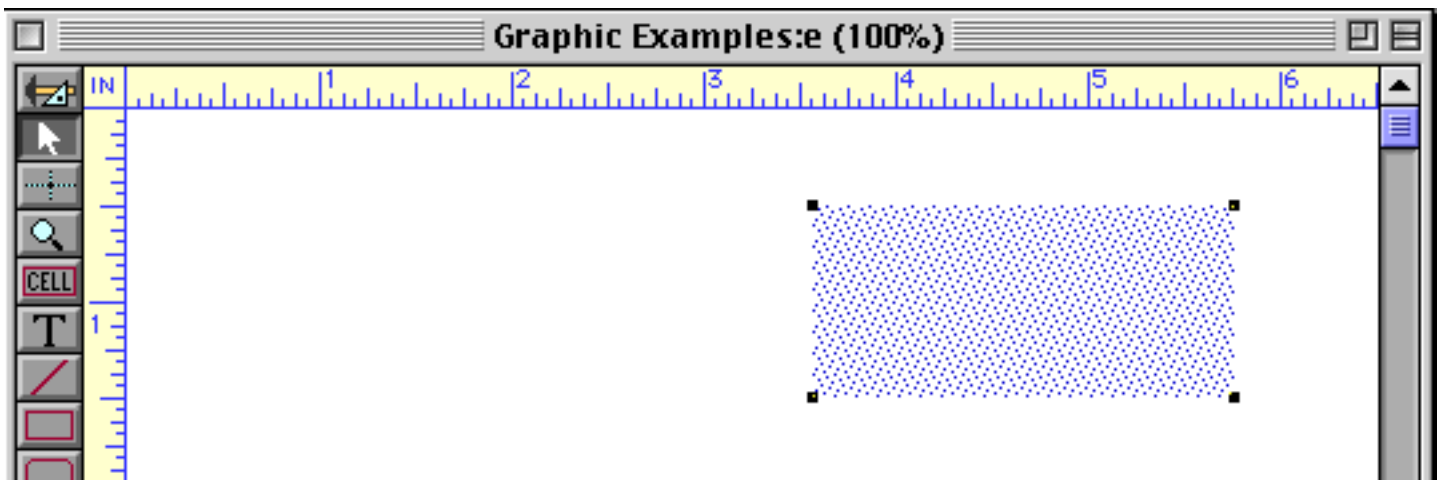
To create a dotted horizontal or vertical line choose one of the diagonal patterns.



The most common line pattern is basic black. If you want an object with no border at all, choose the empty pattern in the top left corner.

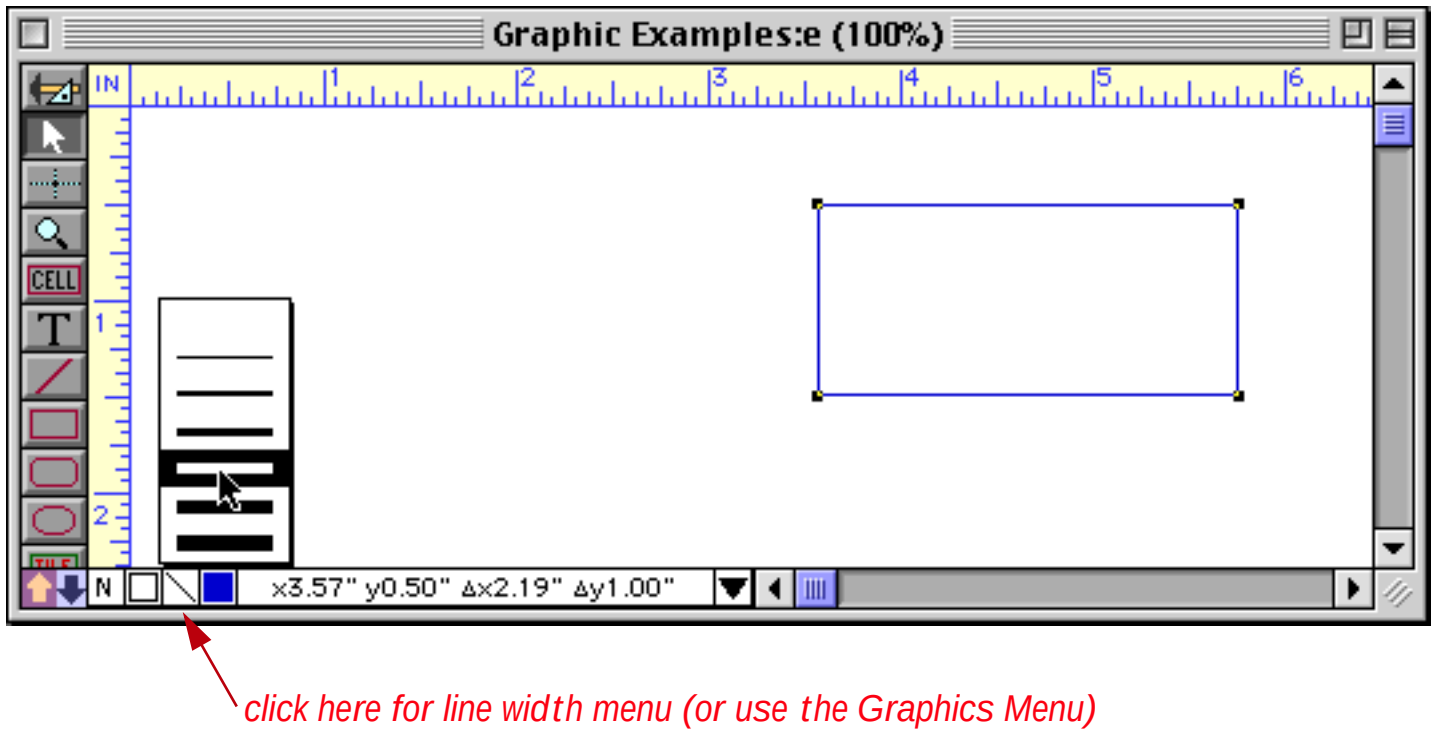


The result is an object without a border.

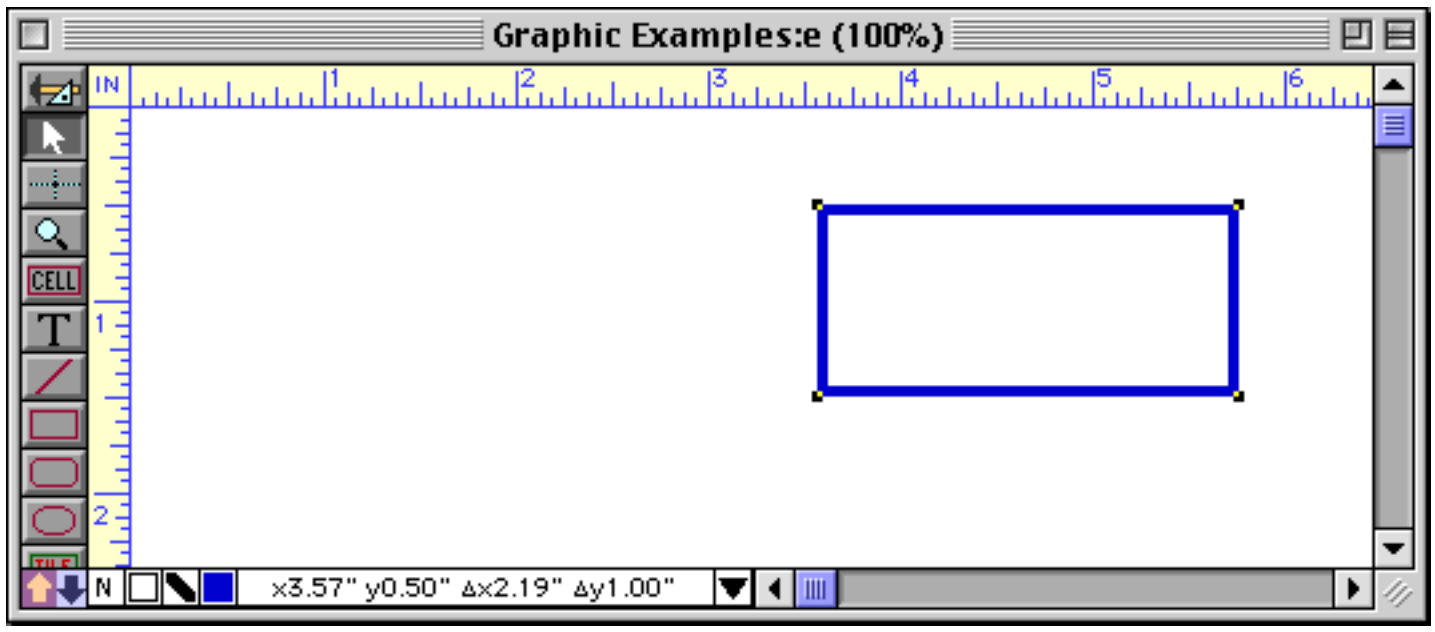


Line Width

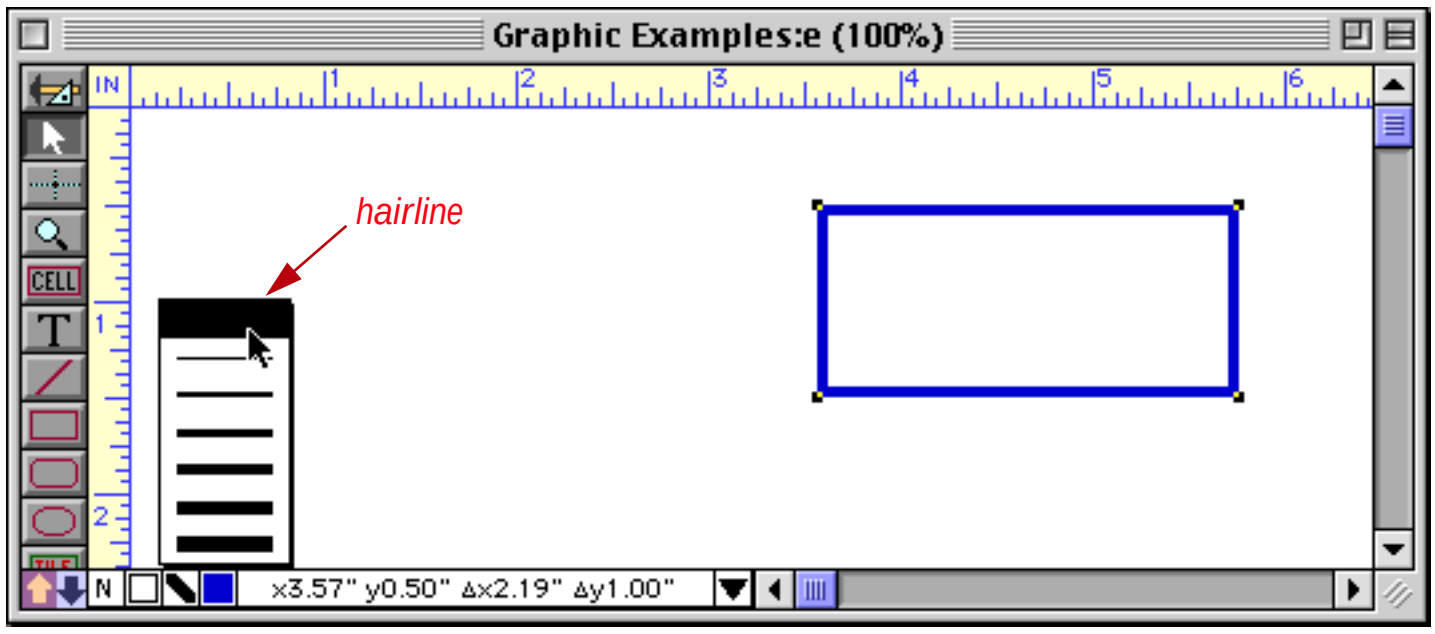
The **Line Width** menu contains seven different line / border thicknesses from 1⁄4 to 6 points.



When you release the mouse the line width of the object will be updated. Notice that the object's line width also appears in the Graphic Control Strip.



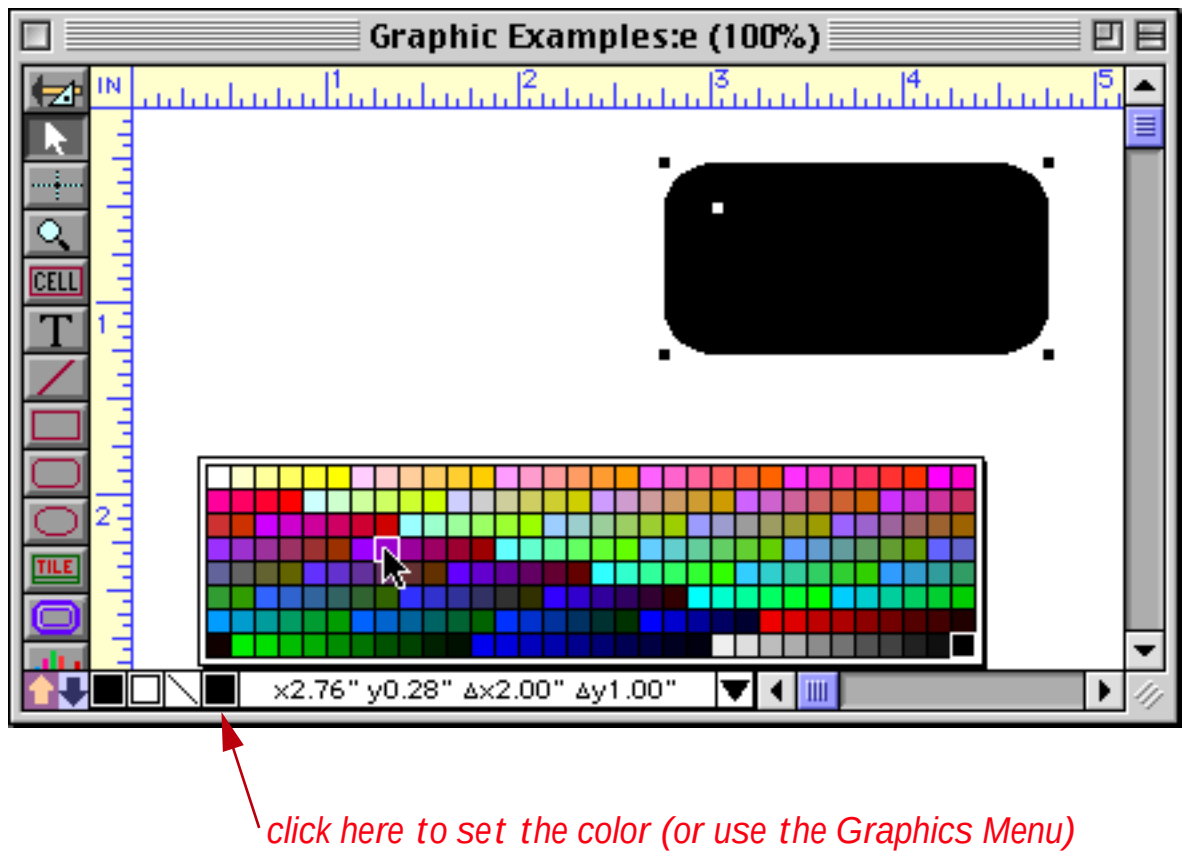
The top spot in the menu, which appears to be empty, actually represents a 1⁄4 point line (1⁄300 inch) hairline.



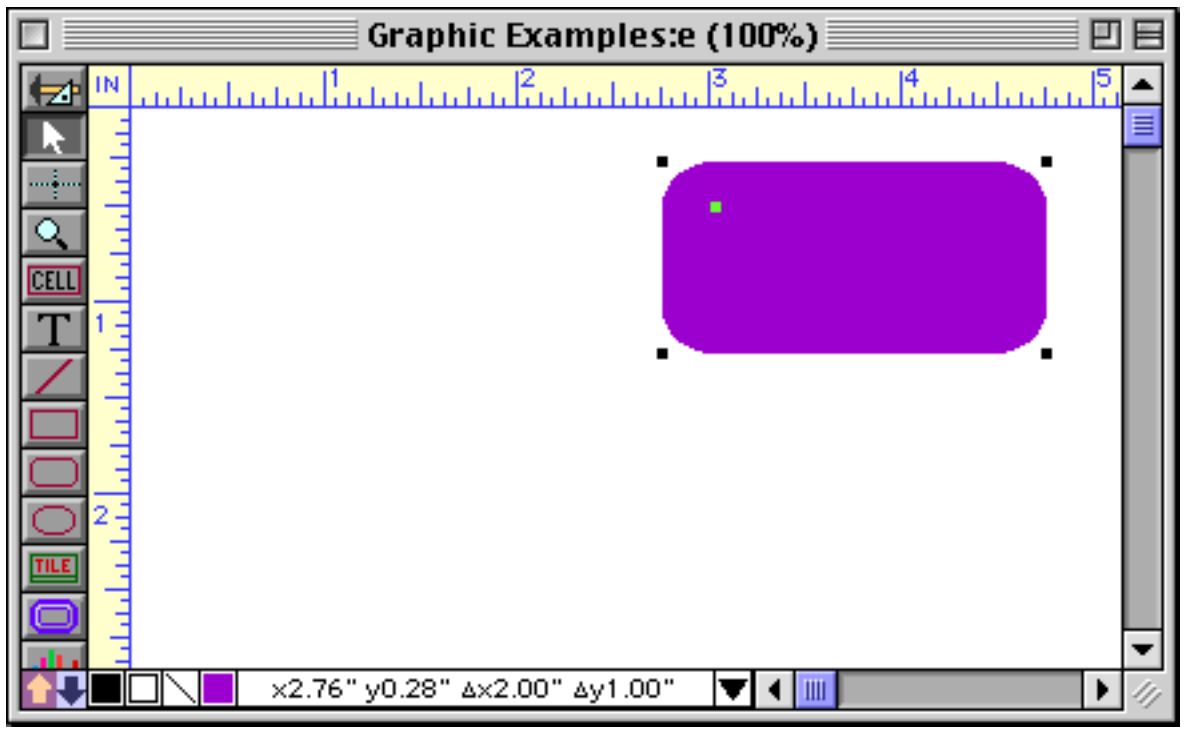
Hairlines can be printed on Postscript laser printers and imagesetters— on the screen and on other types of printers hairlines will appear as ordinary 1 point lines.

Color

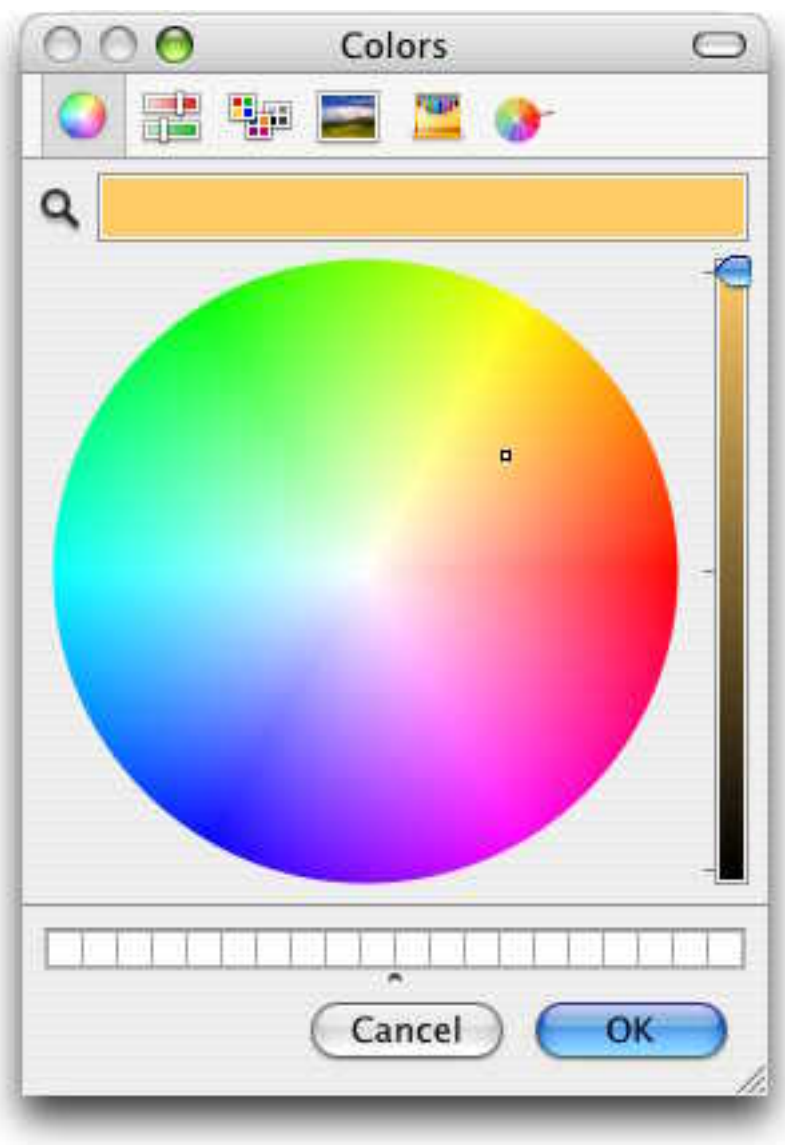
The **Color** menu contains 256 colors. To color an object, first select the object with the **Pointer** tool, then pick the color from the menu.



When you release the menu the object(s) will change color.



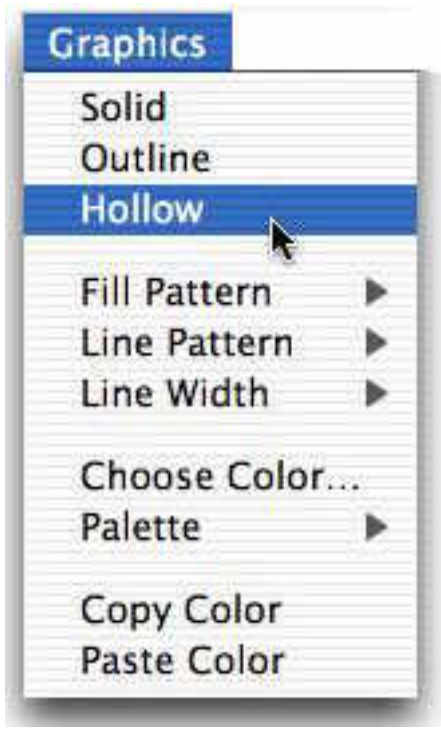
To choose a color that is not one of the 256 colors in the palette, use the **Choose Colors** command in the Graphics menu. The system's standard Color selection dialog will appear, allowing you to choose any of millions of colors.



You can also open this dialog by holding down the Control key (Mac) while clicking the color swatch in the Control strip. On PC systems you can right click the swatch.

Copying and Pasting Colors

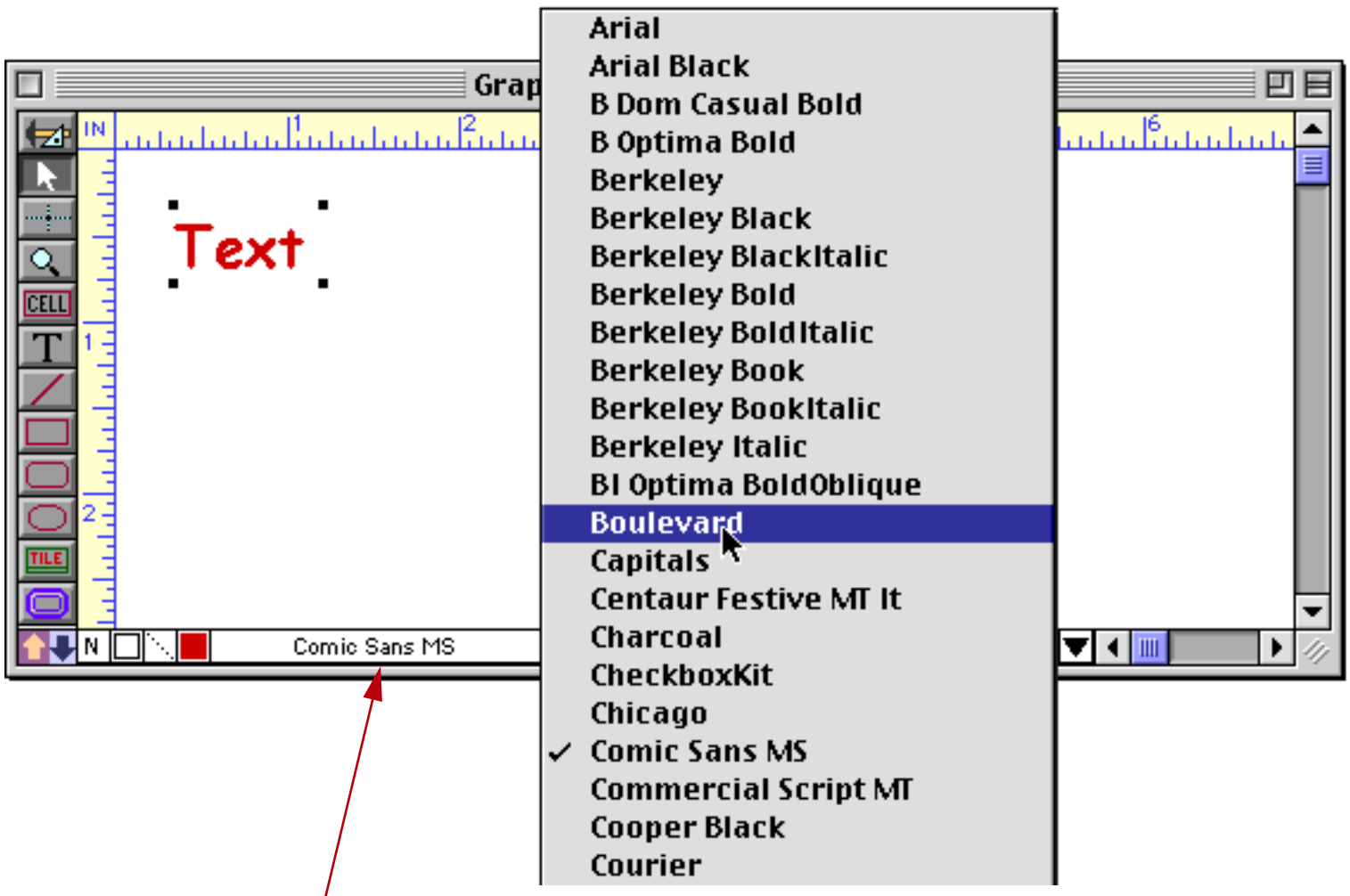
Sometimes you'll want one object to exactly match the color of another object. You can use the **Copy Color** and **Paste Color** commands to transfer a color from one object to one or more other objects.



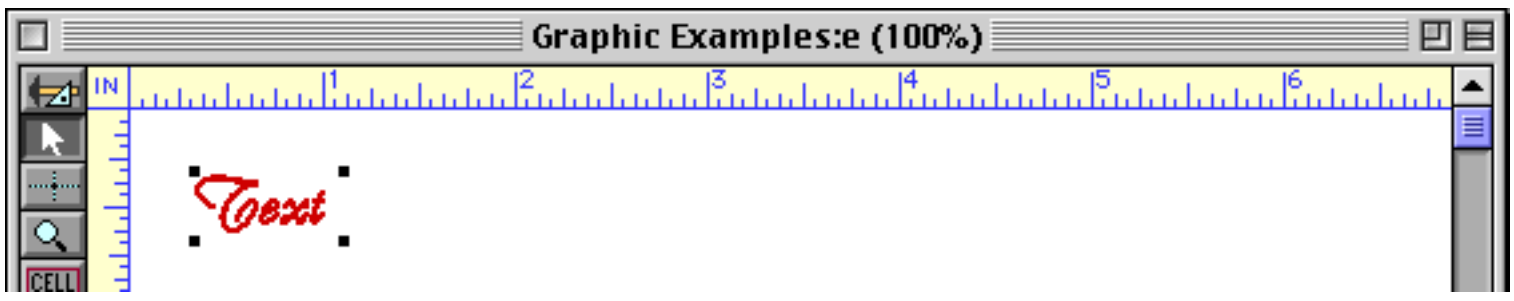
Start by selecting the object that has the color you want, then choose the **Copy Color** command. This copies the color into a special clipboard. Now select the other objects that you want to set to this color and choose the **Paste Color** command. The selected objects will change to the same color as the original object. (Note: The color clipboard is completely separate from the normal clipboard that is used for other copy and paste operations.)

Font

The **Font** menu lists all the fonts installed on your system. To change the font of an object that contains text, first select the object with the **Pointer** tool, then pick the font from the menu.



When you release the mouse the object will change to the new font.



You can use the **Font Usage Wizard** to display a list of all the fonts used in every form in your database. See “Font Usage” on page 25.

Maintaining Fonts across Multiple Computers and Platforms

If a font has the same name on two different computers then it can be used in a database on either computer — even if they are different types of computers (Macintosh vs. Window PC). For example, suppose your database is created on a Macintosh and uses the Adobe fonts Palatino and Optima. If the database is transferred to a Windows PC system these fonts will continue to display properly if the Palatino and Optima are installed on the Windows PC.

No matter what type of computer you are using Panorama will check for the necessary fonts each time it opens a database. If a database uses one or more fonts that are not installed on this system an alert will appear listing the fonts that are not available on your system.



To allow you to continue to use the database the objects that use these fonts are switched either to Geneva (Macintosh) or Alpine (Windows). If you want to keep the original fonts you must close the database without saving it, install the necessary fonts, then re-open the database. Once the database has been saved you cannot go back to the original fonts except by manually setting up the fonts again, object by object.

Universal Fonts

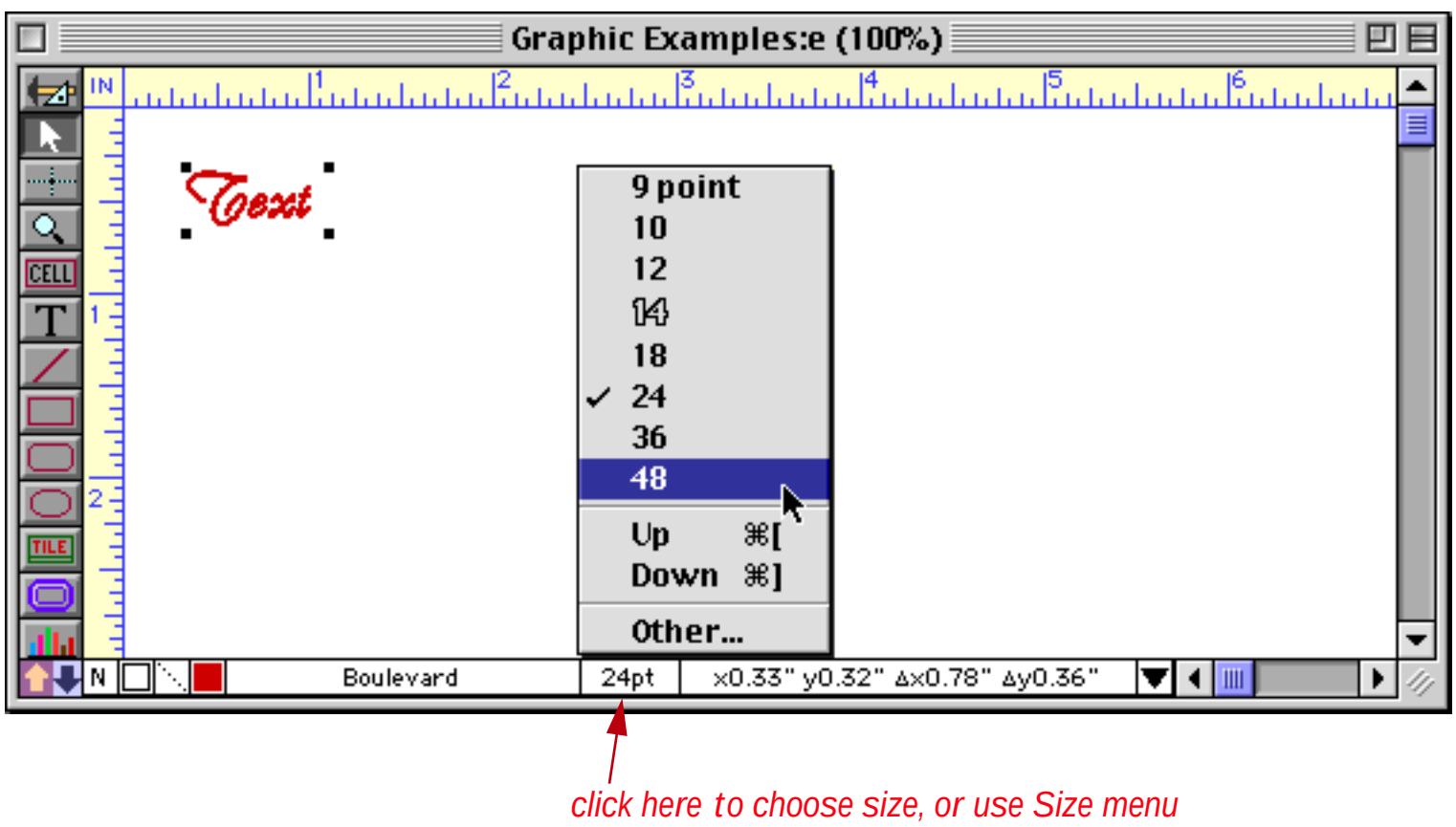
Panorama has special handling for four special fonts.

Macintosh	Windows
Geneva	Alpine
New York	Yankee
Chicago	City
Monaco	Block

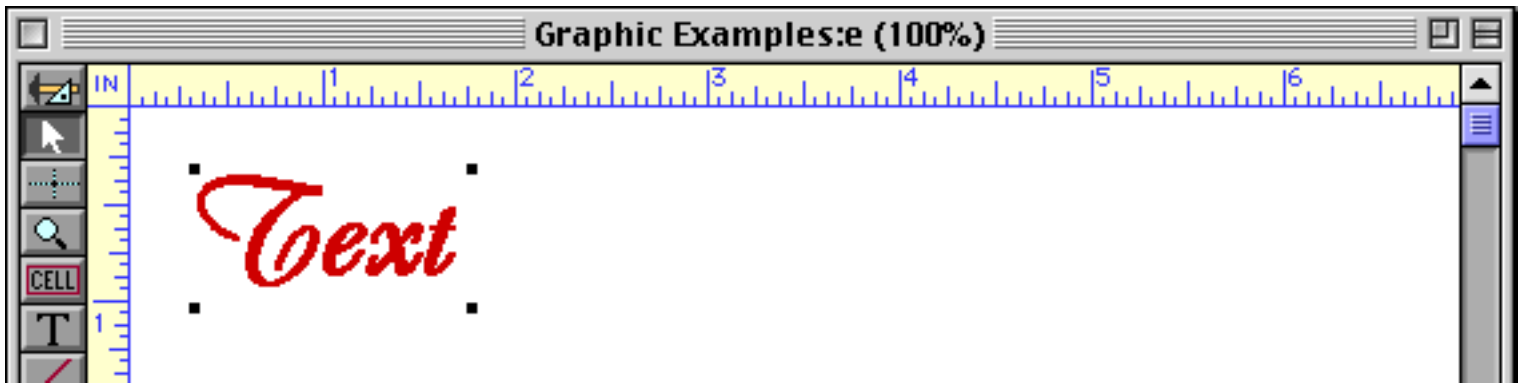
On the Macintosh these four fonts are always present as universal fonts, so you can rely on them always being available. We have created the four equivalent fonts for Windows computers to guarantee that these fonts are always available on any computer. For example, if you create an object using the Geneva font on a Macintosh computer it will automatically be translated to the Alpine font when displayed on a Windows PC computer. If you want to make sure that your database will display properly on any computer you should restrict yourself to using only these four fonts.

Text Size

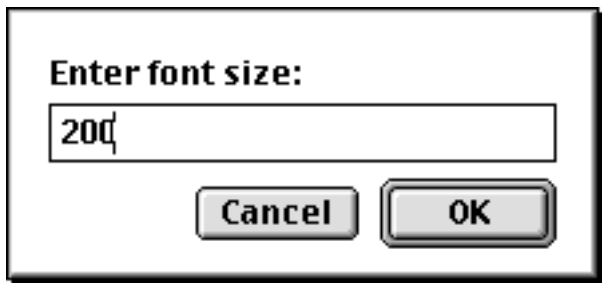
The **Size** menu allows you to adjust the text size of an object. To change the text size, first select the object with the **Pointer** tool, then pick the size from the menu.



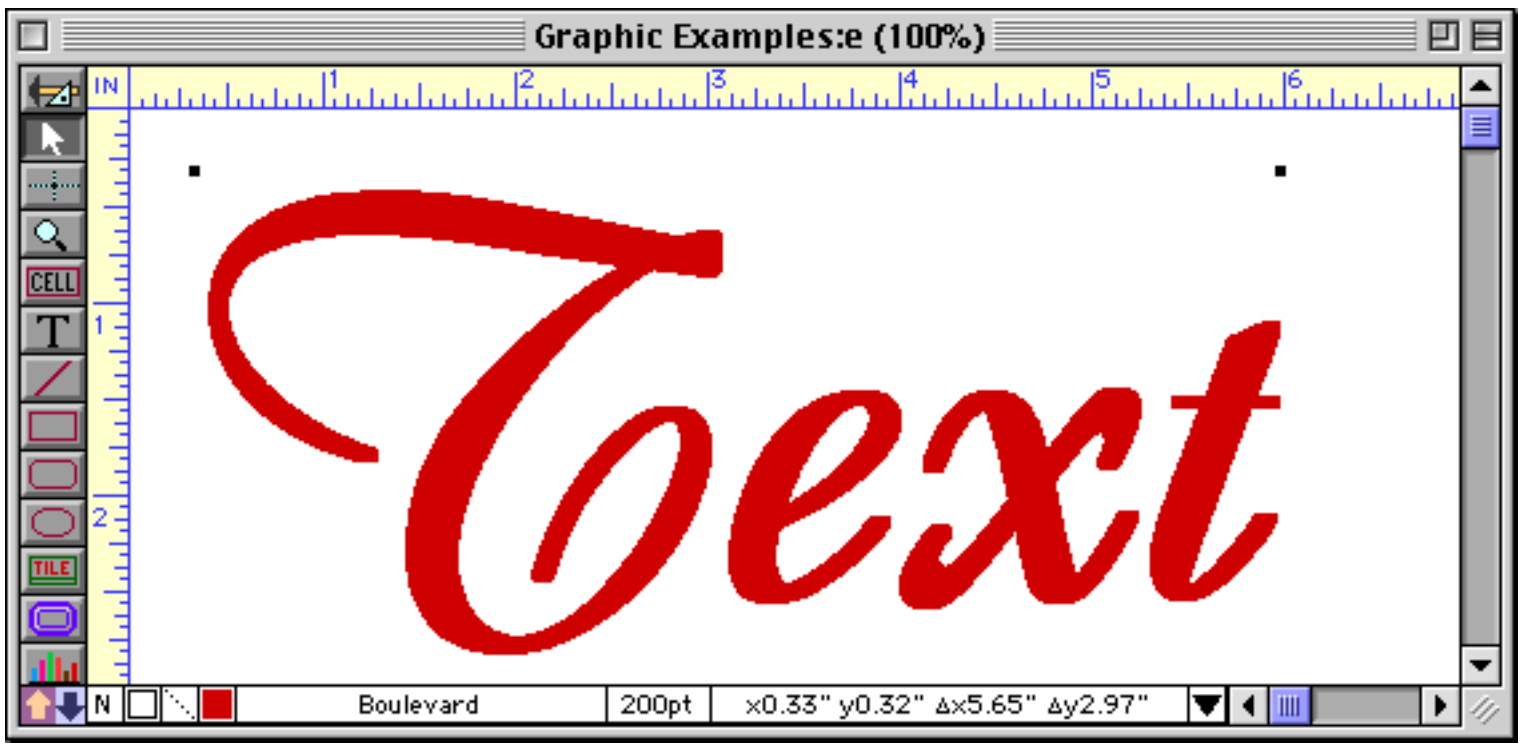
When you release the mouse, the size of the text will change.



If the size you want is not listed in the menu, choose **Other**. A dialog will appear asking for the new font size.



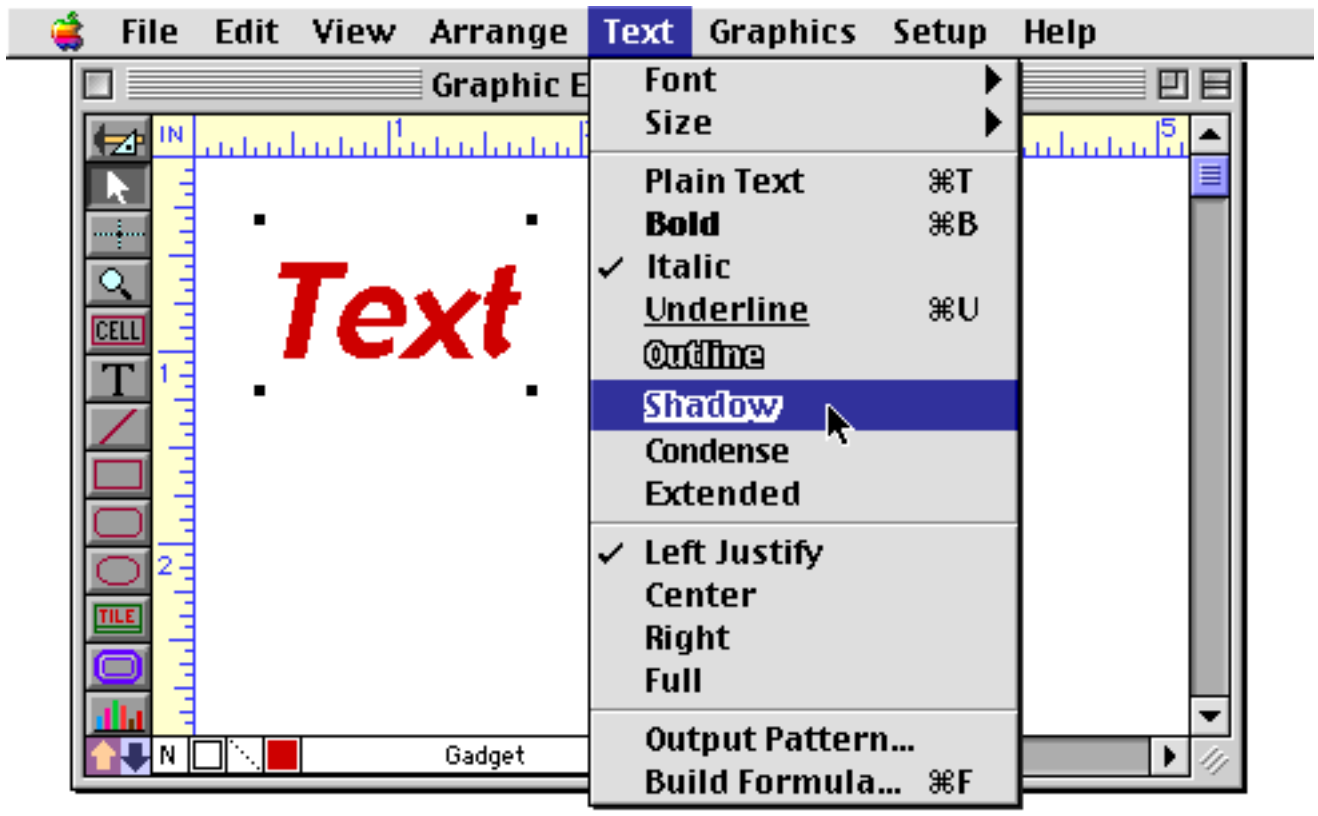
You can type in any integer font size you like — big or small.



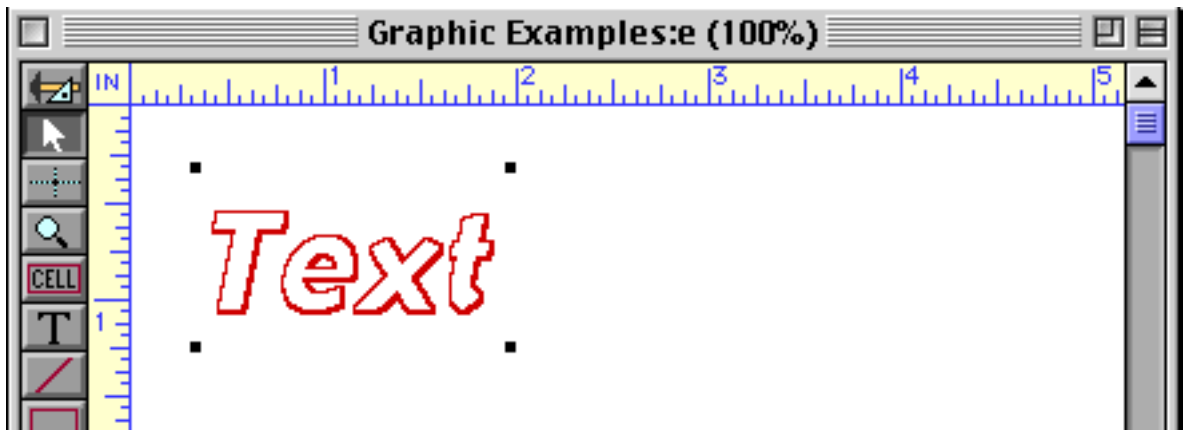
You can also make small adjustments to the text size by selecting **Up** or **Down**. These commands increase or decrease the text size by one point.

Text Style

The **Text** menu allows you to change the style of an object containing text.



You must change the style of the entire object — all the characters in the object must have the same style.

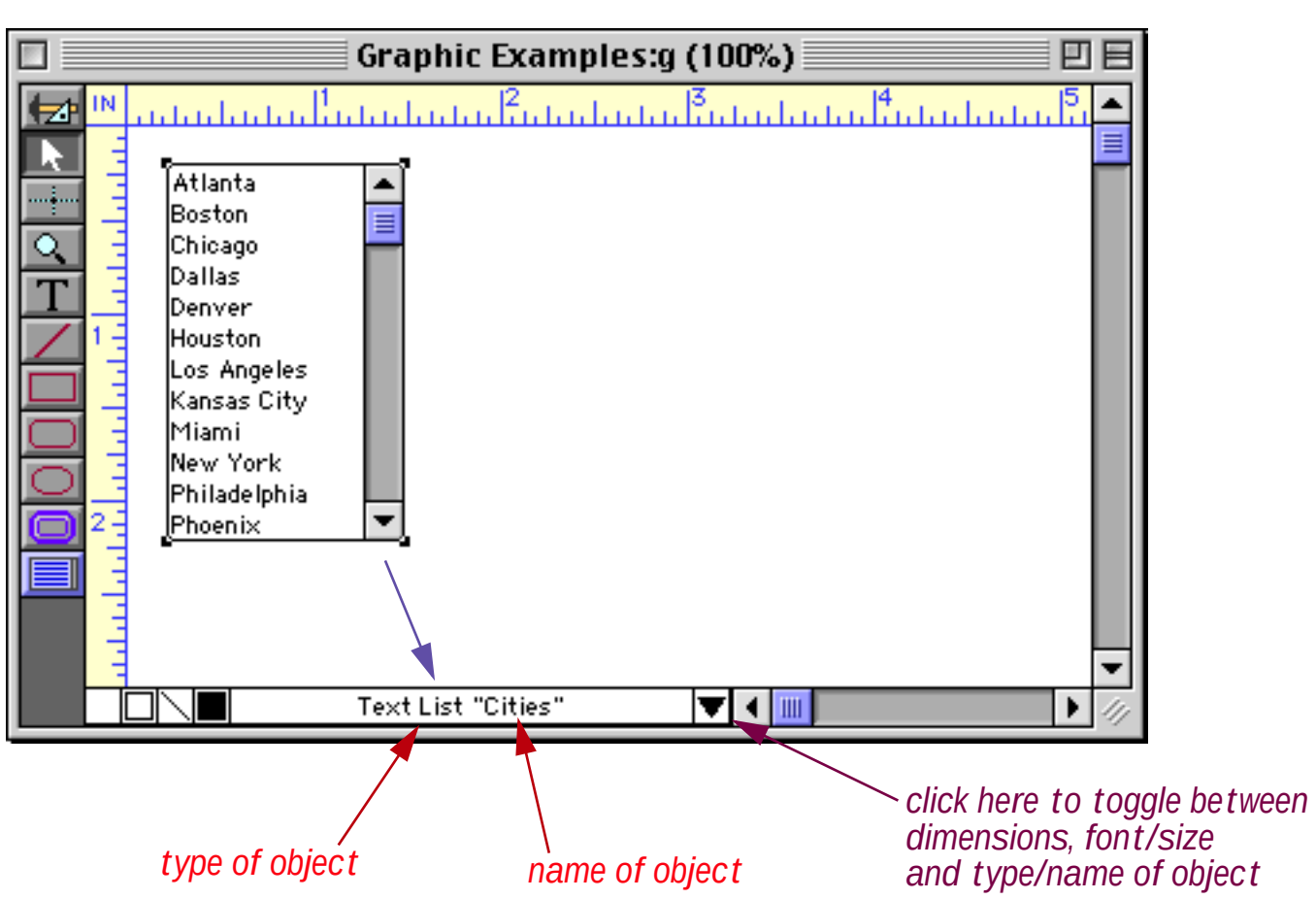


The only exception to this rule is the Word Processor SuperObject, which allows different styles, fonts, sizes and colors to be mixed. See “Word Processor SuperObject” on page 673 for more information.

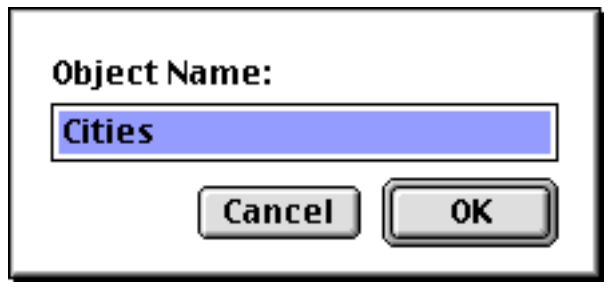
Object Type/Object Name

Panorama allows a name to be assigned to any graphic object. The name can be used to identify the object when programming. If an object has a name, a program can get information about the object, or even modify the object on the fly. If you are not planning to do any programming with an object, there is no need to assign a name to it.

The **Graphic Control Strip** can display the type of the currently selected object, along with its name (if it has a name). Object types describe what the object is: Rectangle, Oval, Line, Button, Data Cell, etc.



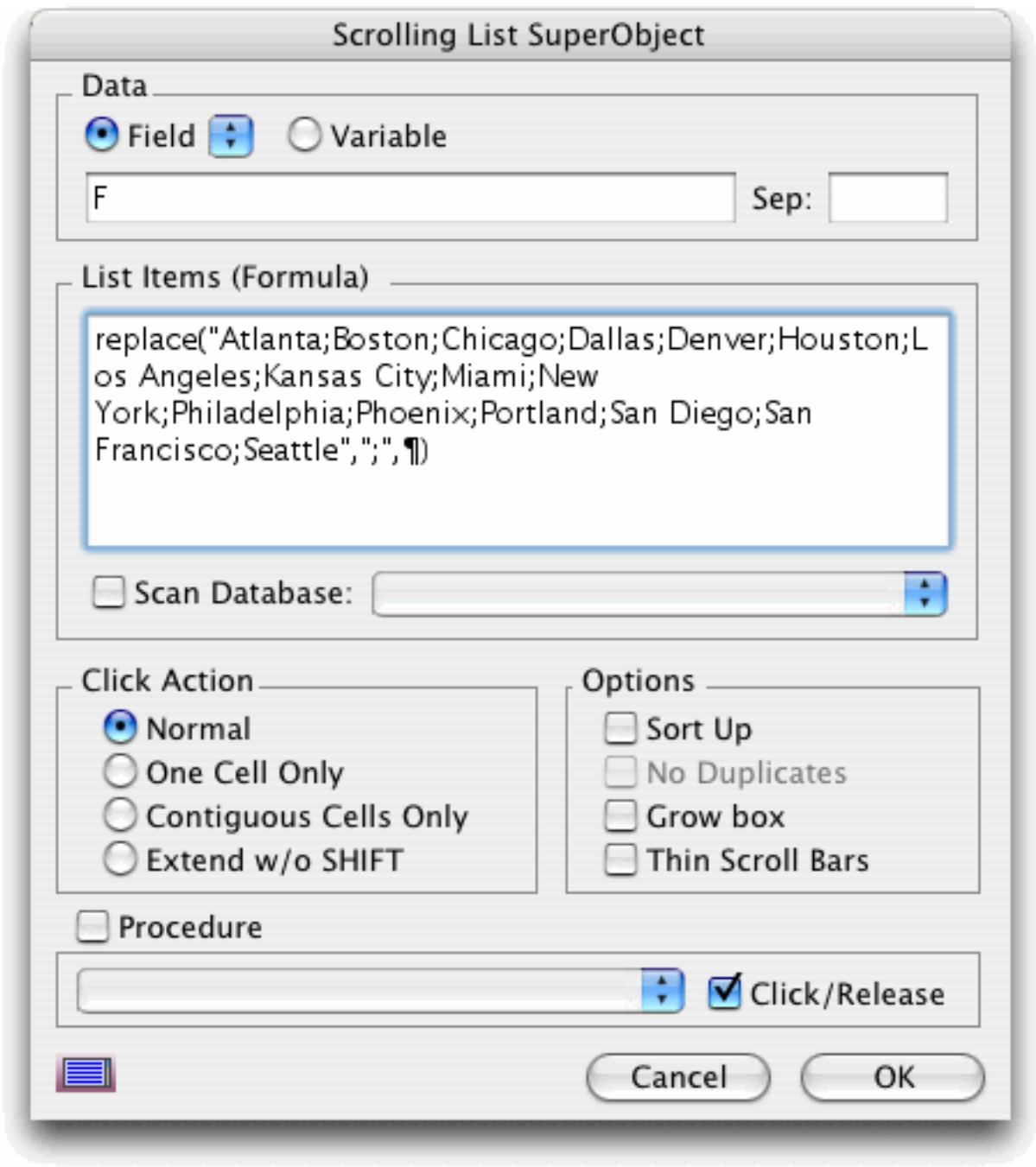
You cannot change the type of an object, but you can change its name. To do so, either click on the object type or name in the Graphic Control Strip, or choose **Object Name** from the Edit menu.



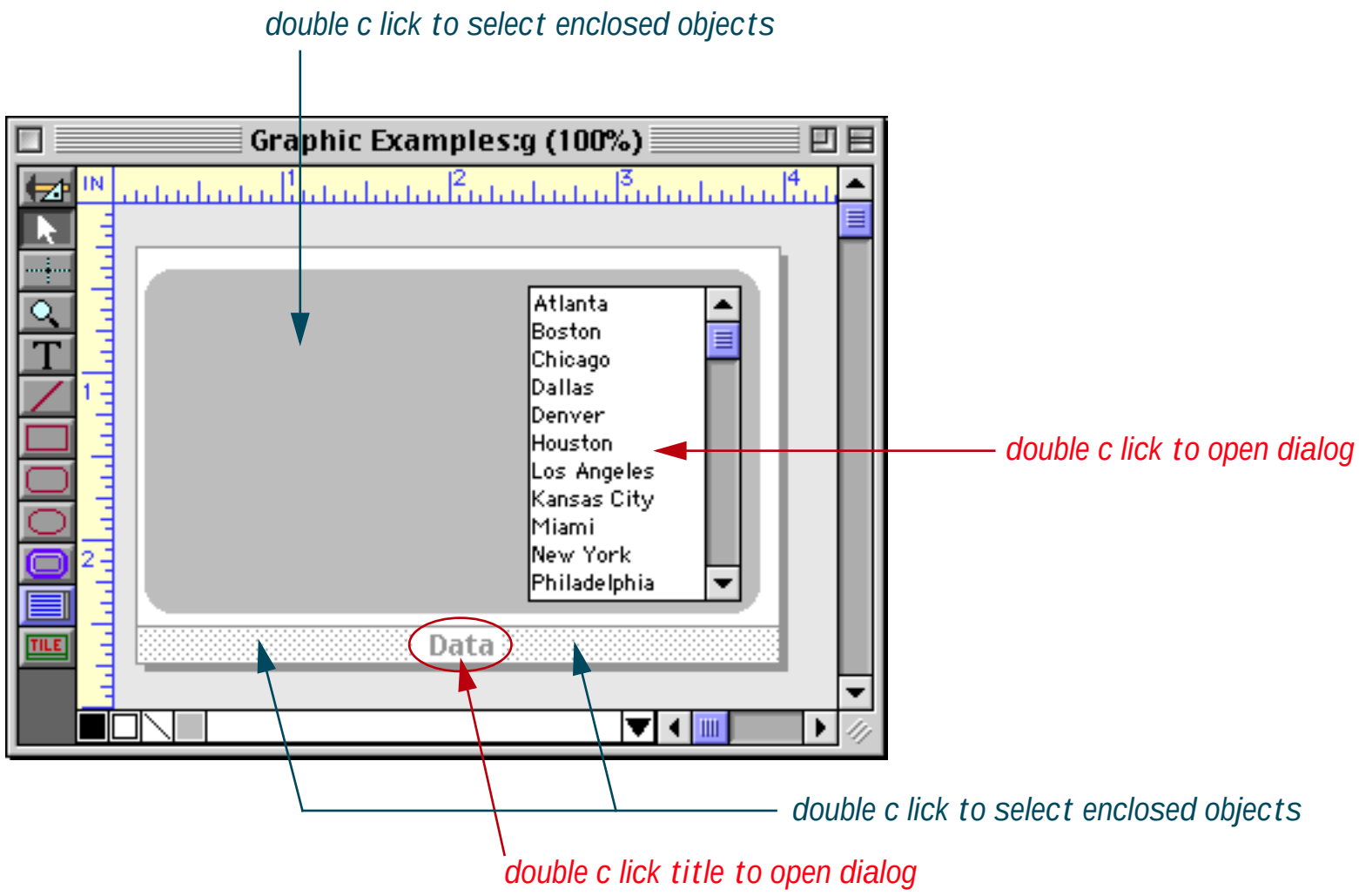
You can use any name you want for the object. You can even have two objects with the same name, which can be handy if you always want to program them the same way. For more information on programming objects via their name see "Programming Graphic Objects on the Fly" on page 1753 and "Program Control of SuperObjects™" on page 1786.

The Object Properties Dialog

Many graphic objects have dialogs that control various options for the object. To open the dialog for a particular object, first select the object and then choose the **Object Properties** command in the Edit menu. This command will work for data cells, tiles, buttons, flash art, flash sound and all SuperObjects. There is no dialog for basic shapes like rectangles, ovals, etc. The Object Properties dialog is the same dialog that appears when you create the object. Here is a typical Object Properties dialog.



Another way to open the **Object Properties** dialog is to simply double click on the object in question. In fact, this is the most common way to open this dialog. With most types of objects you can double click anywhere in the object. For a report tile, however, you must double click on the name of the tile. If you double click on the drag bar of the tile (outside the name), Panorama will not open the Object Properties dialog, but instead will select all the objects inside the tile. (See "Working with Tiles" on page 1050 to learn more about report tiles.) The illustration below shows a report tile with two other objects, a list and a rounded rectangle.



If an object doesn't have an **Object Properties** dialog (for example basic shapes like rectangles and ovals) then double clicking on the object will select all of the object inside the object. "Selecting Multiple Objects at Once" on page 502 for more information about this technique.

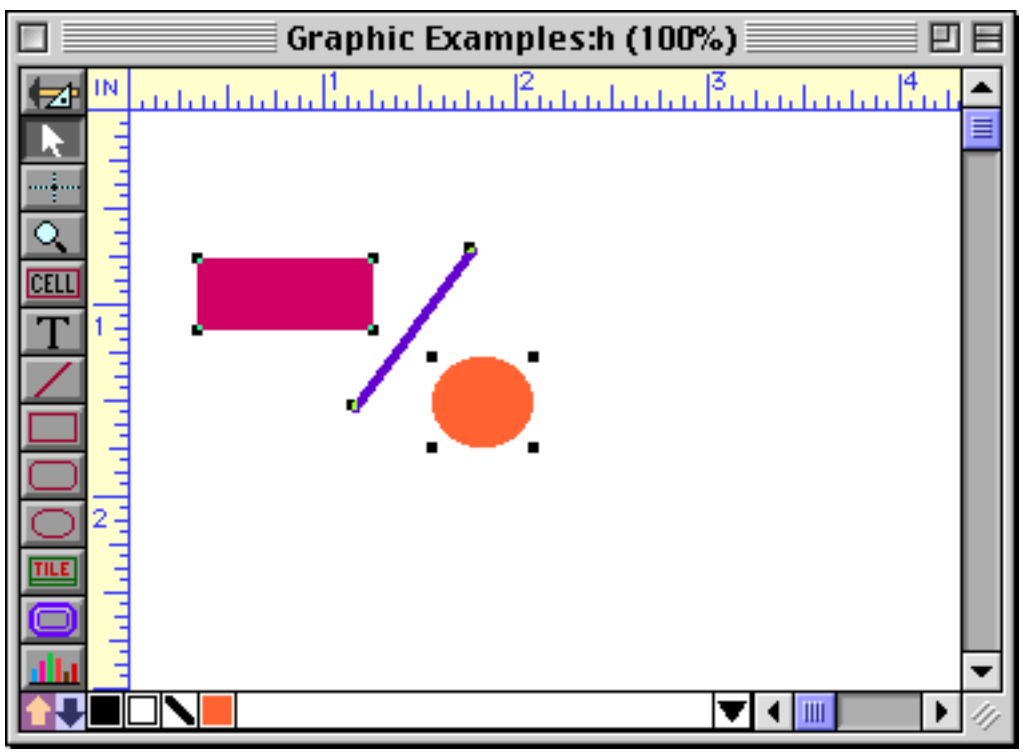
Note: In older versions of Panorama (Panorama 2.x) you needed to click on the object with the corresponding tool to open the Object Properties dialog. For example to open the Object Properties dialog for a button you needed to select the **Button** tool, then click on the button. This will still work, but is no longer necessary with the **Object Properties** command in the Edit menu and the double clicking (with the pointer) shortcut.

Working With Multiple Objects

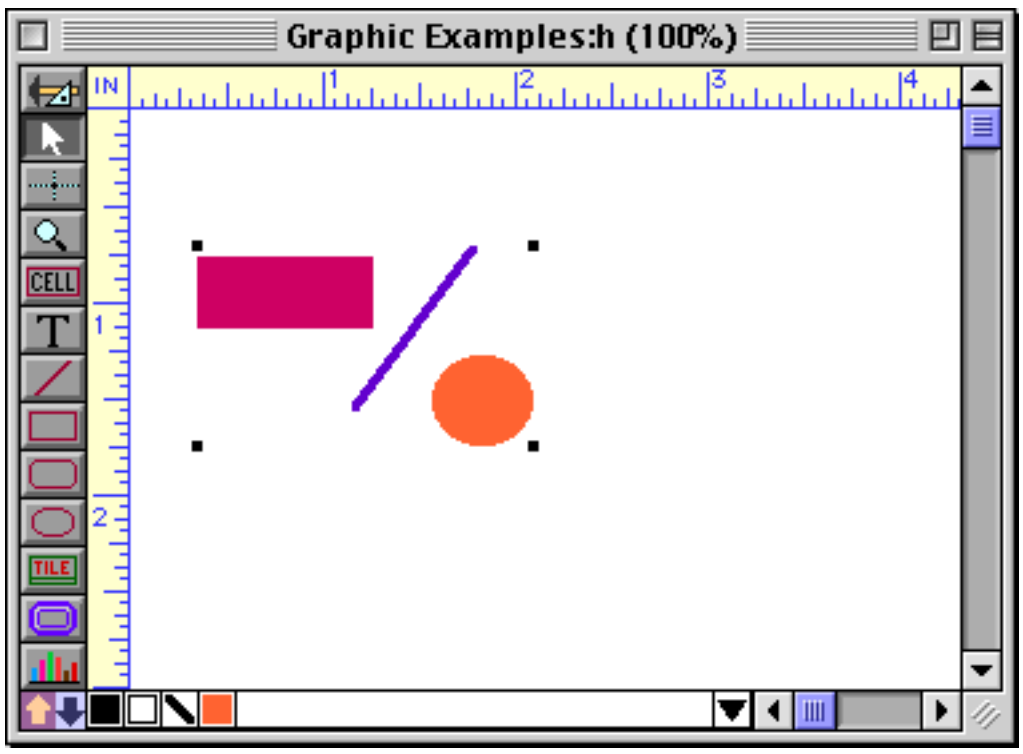
Most forms contain dozens or even hundreds of objects. Often you'll need to move, resize, or align several objects at once. Panorama has a number of tools to make these kinds of tasks easier.

Grouping Objects Together

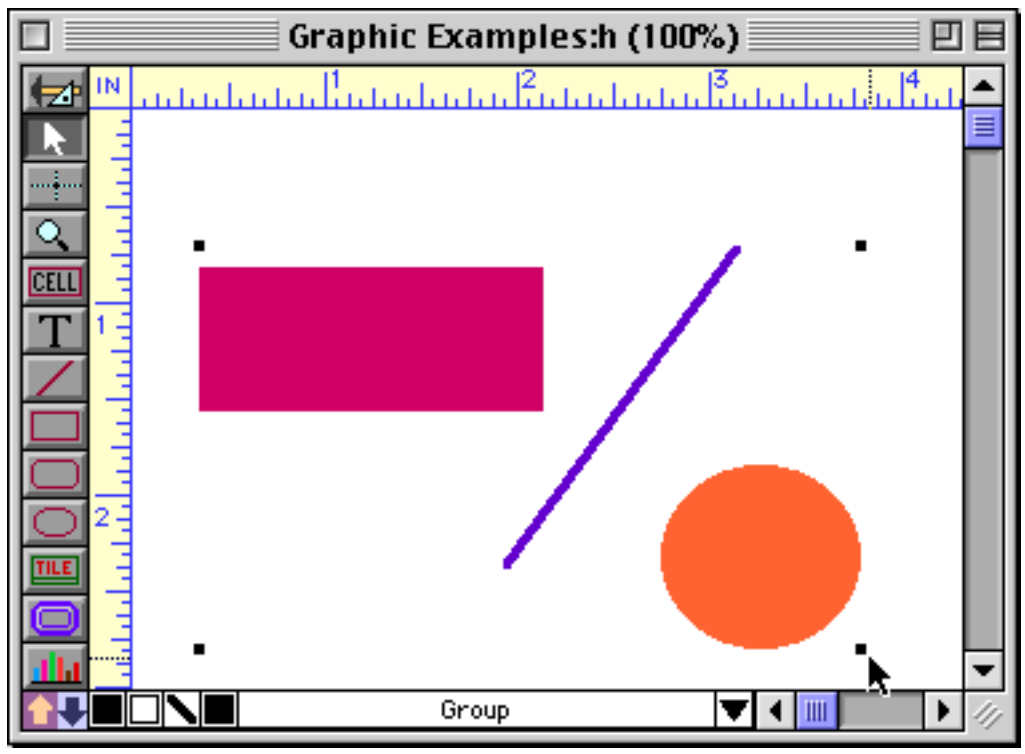
The **Group** command (in the Arrange menu) gathers several objects together and combines them into a single object. Once the objects are grouped together, they act as a single unit. The group can be selected, moved, copied, or resized just as if it was a single object. To group several objects together, first select the objects.



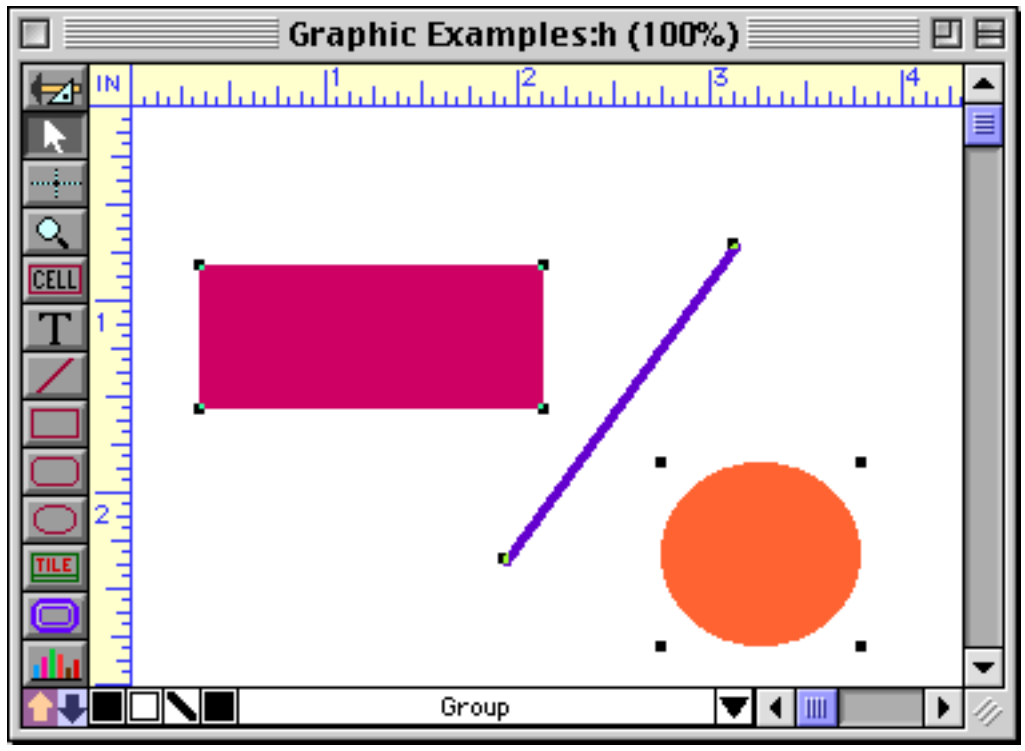
Once the objects are selected, choose the **Group** command in the Arrange menu. The handles at the corners of each individual object will disappear, while new handles appear at the corners of the new composite object.



The group now acts as if it was a single object. You can move it, resize it, copy it or delete it, all as a single entity. For example if you expand the group object, all of the component objects within the group will expand proportionally.



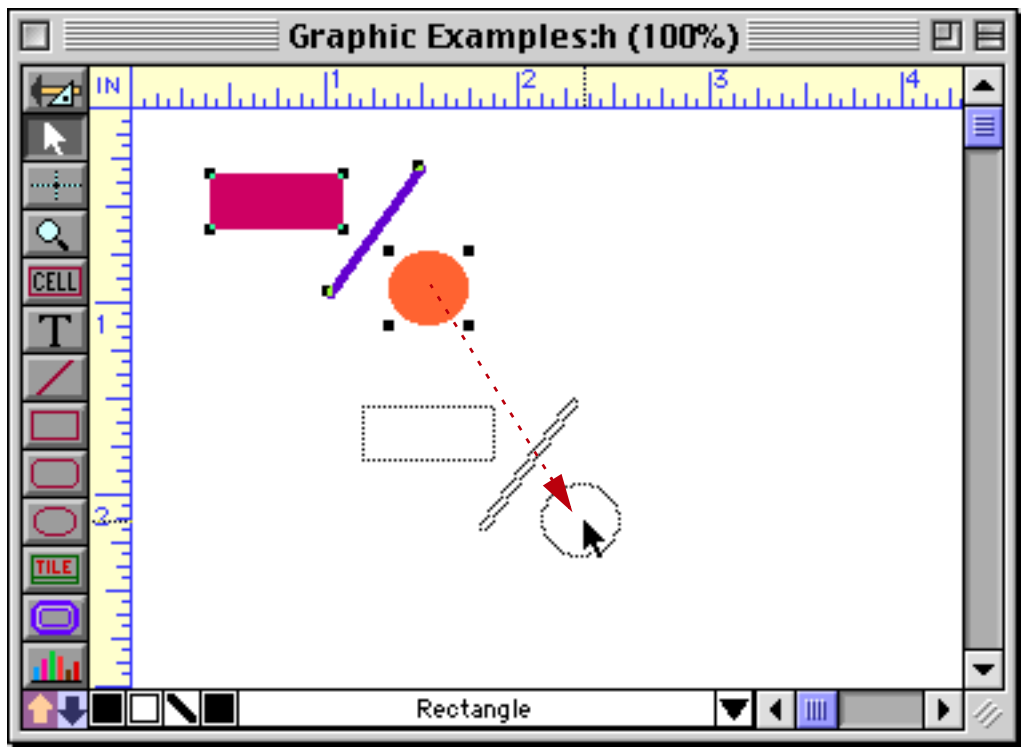
If you want to manipulate one of the individual objects within a group you must reverse the process with the **Ungroup** command. This command splits a group back into the original separate objects.



Tip: Don’t confuse the **Group** command in the Arrange menu with the **Group** command in the Sort menu. The **Group** command in the Sort menu is used to collect data into groups, while the **Group** command in the Arrange menu is used to collect graphic objects into groups. See “STEP 1 - GROUP” on page 399 for more information about the other **Group** command.

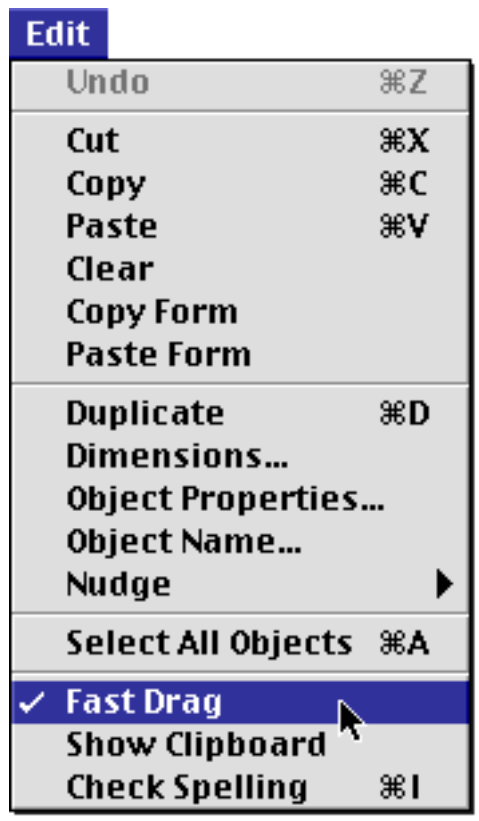
Moving Multiple Objects

You can move several objects at once by selecting the objects and then dragging one of them. The other selected objects will follow as you drag. You can also nudge the selected objects with the arrow keys (see “Nudging an Object (or Objects)” on page 509).

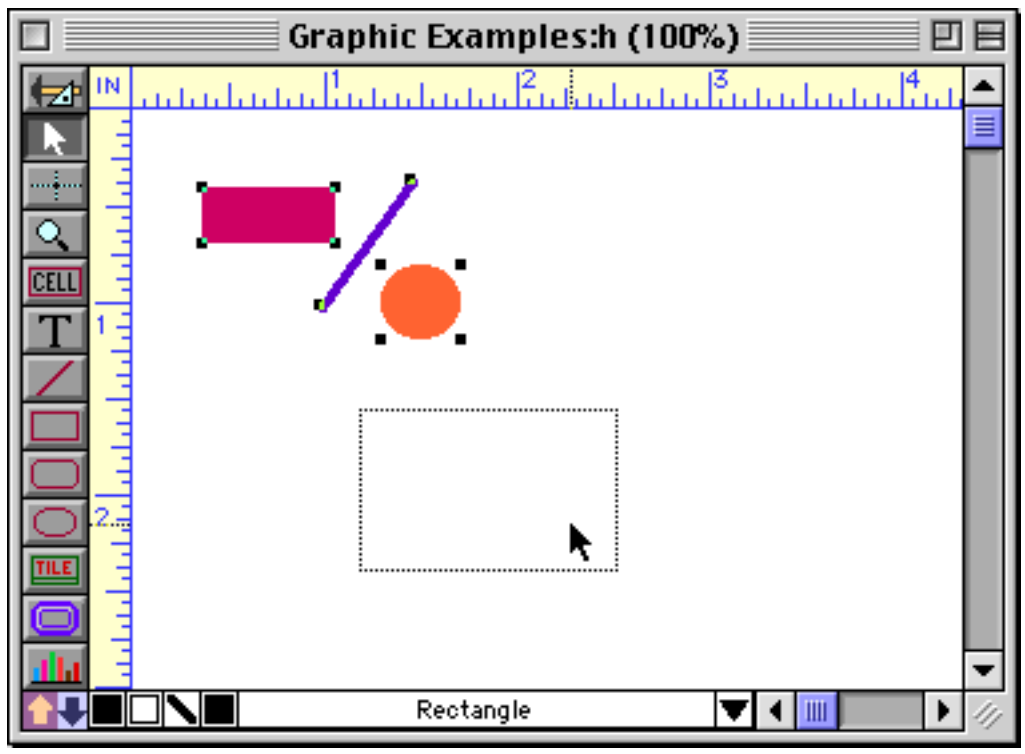


Fast Drag

As you drag multiple objects, a gray outline of the objects follows the movements of the mouse (as shown in the previous illustration). If you have a very slow computer and are dragging a large number of objects, there may be a delay while Panorama creates this gray outline. You can eliminate this delay by checking the **Fast Drag** option in the Edit menu.



When this option is checked the delay is reduced, but the gray outline may be less detailed.

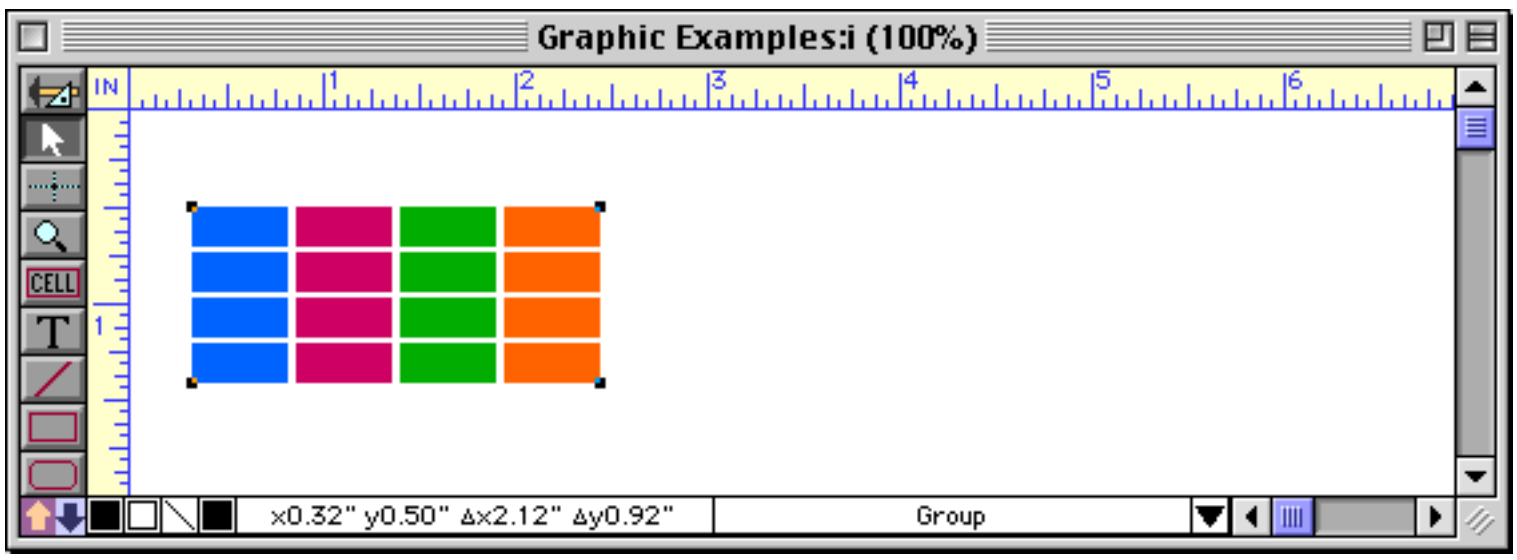


The **Fast Drag** option remains on until you turn it off or quit Panorama.

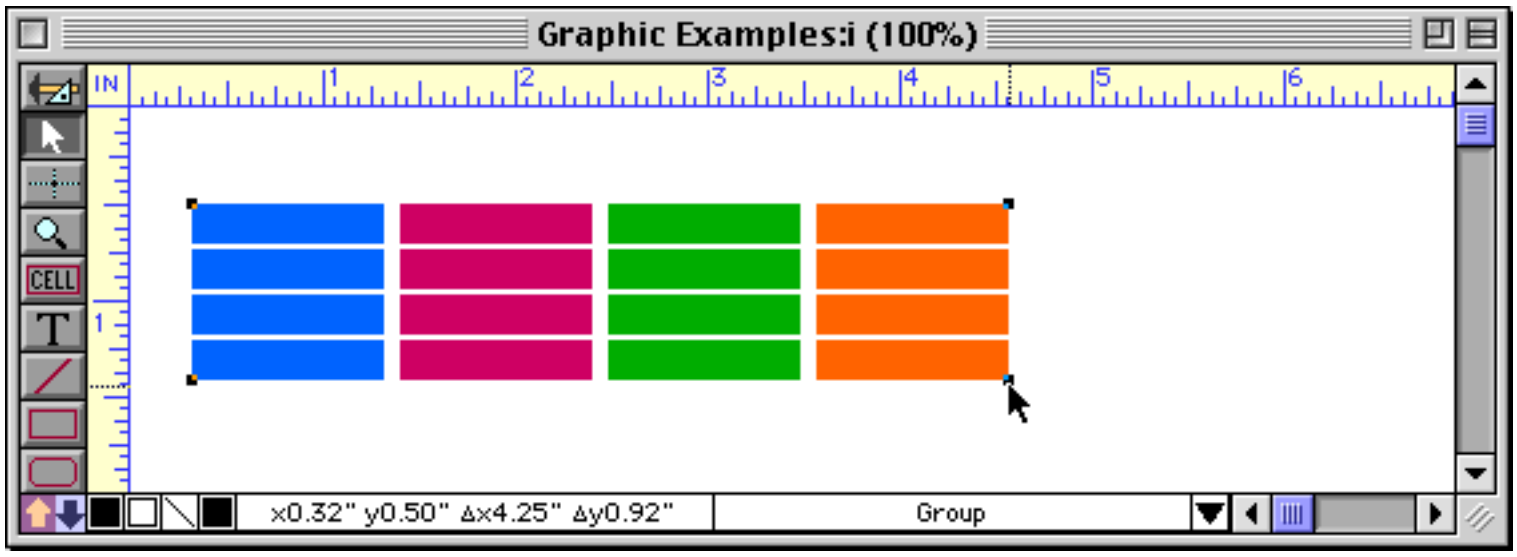
Resizing Multiple Objects

You have several choices available for resizing multiple objects. You can combine objects into a group and then change the size of the group. You can use the **Dimensions** dialog to make each selected object a fixed height or width, or to increase or decrease the size of each selected object. Finally, you can use **cluster resize** to change the width of a column within a table, or the height of a row.

When you change the size of a group object, all of the dimensions inside the group change as well. For instance, if you double the width of a group, all of the individual objects within the group will also double in width. The horizontal spacing between the groups will also double. Here is a group of objects before the width is adjusted…

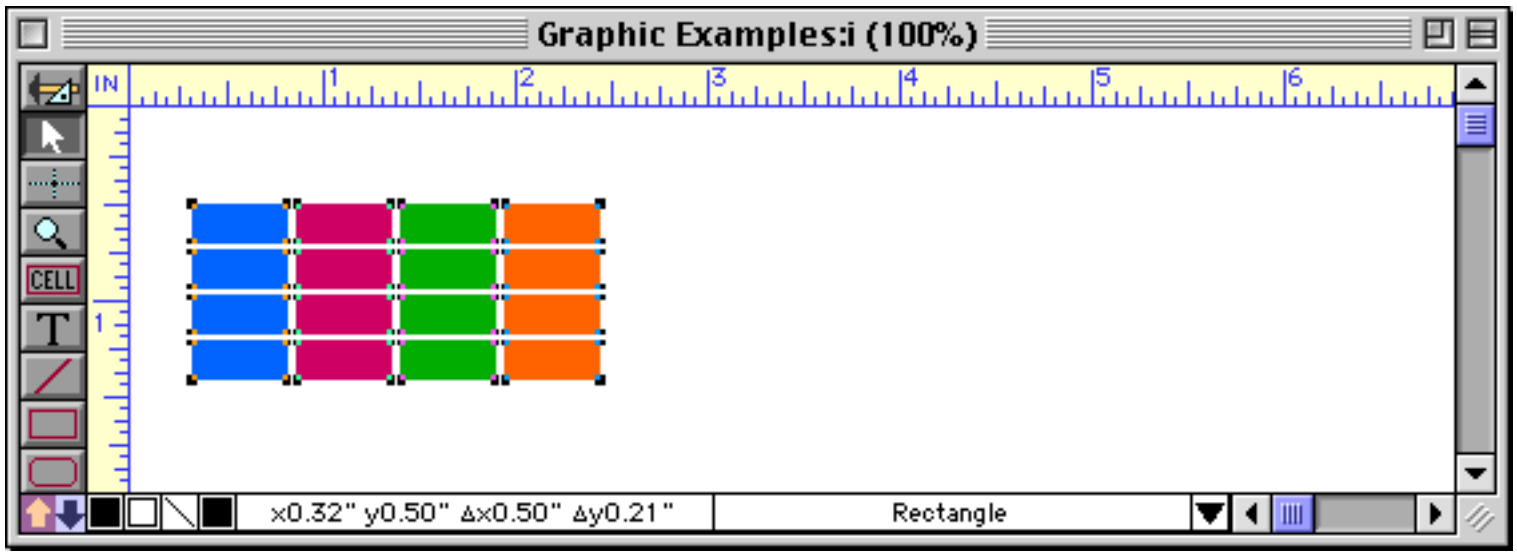


and after.

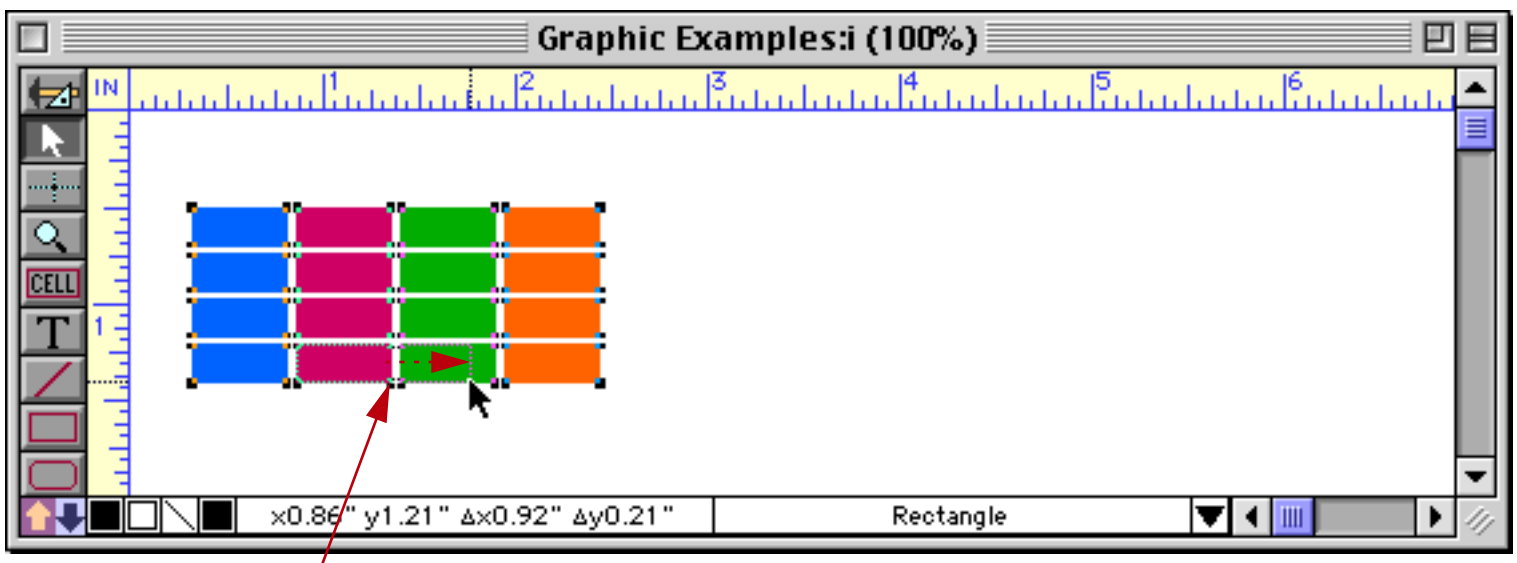


Cluster Resize

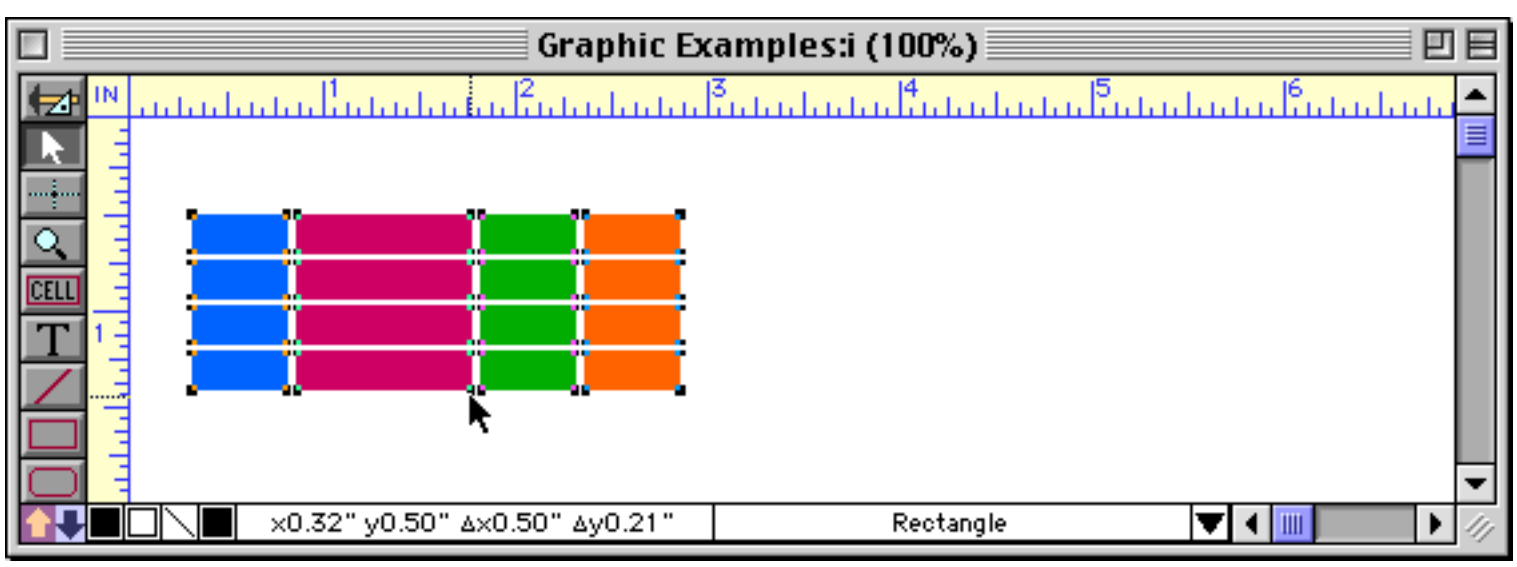
Panorama's **Cluster Resize** is automatically active whenever you select several objects and then change the size of one of them. Cluster resize adjusts the sizes and locations of the other selected objects to match the change. For instance, suppose you have a matrix of boxes, and you want to increase the width of one column.



Start by selecting all of the objects (see "Selecting Multiple Objects at Once" on page 502). Then grab one of the handles in the column you want to make wider and drag it to expand the box.

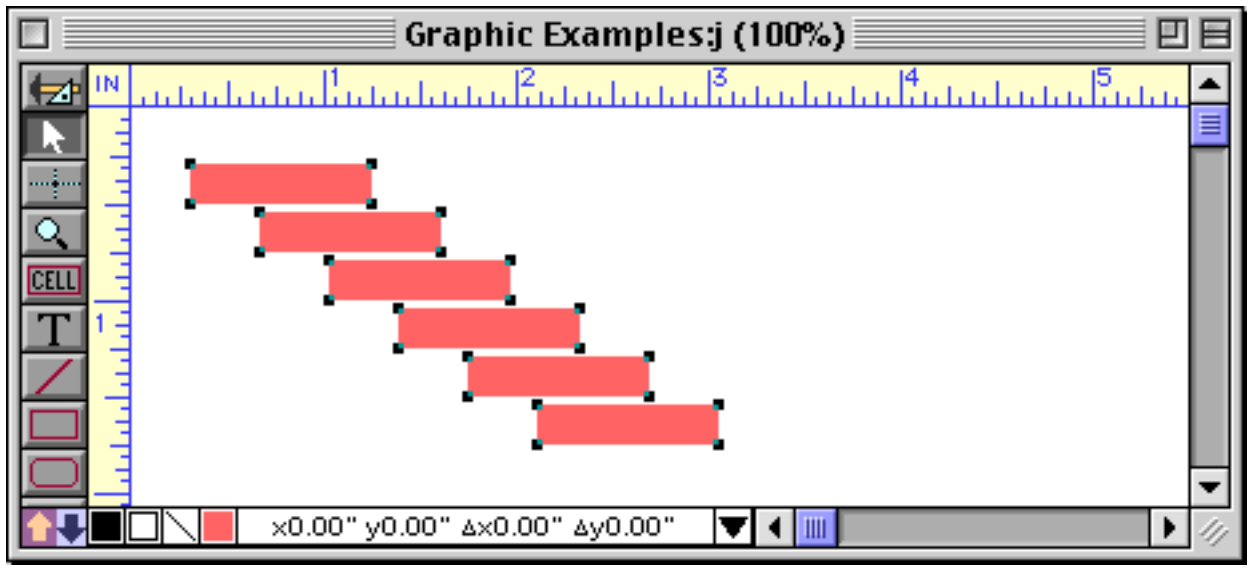


When you release the mouse, cluster resize will kick in. In this case, it will automatically increase the width of all the other boxes in the same column. It will also shift the boxes on the right to make room for the expansion.

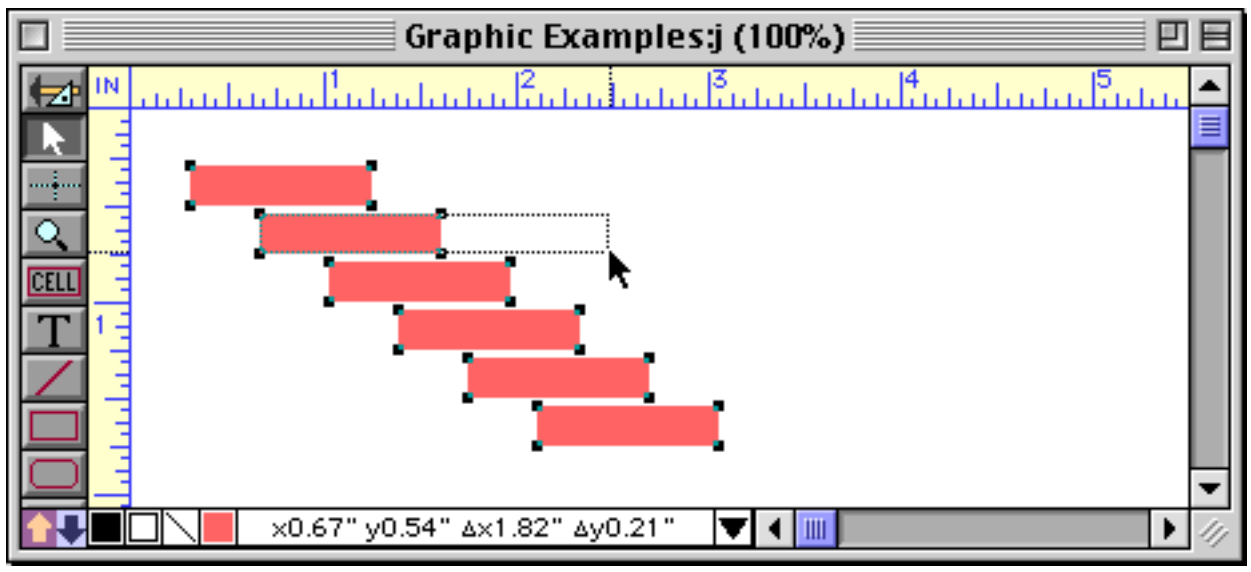


At first glance cluster resize may look a little bit like magic, but actually it is quite simple. After you change the size of any object, Panorama checks to see which edge (or edges, for diagonal moves) of the object you moved—top, bottom, left, or right. It then adjusts the corners of any other selected objects that are in that direction. For example, if you move the right edge of an object, any objects that are even with or to the right of that edge will be adjusted.

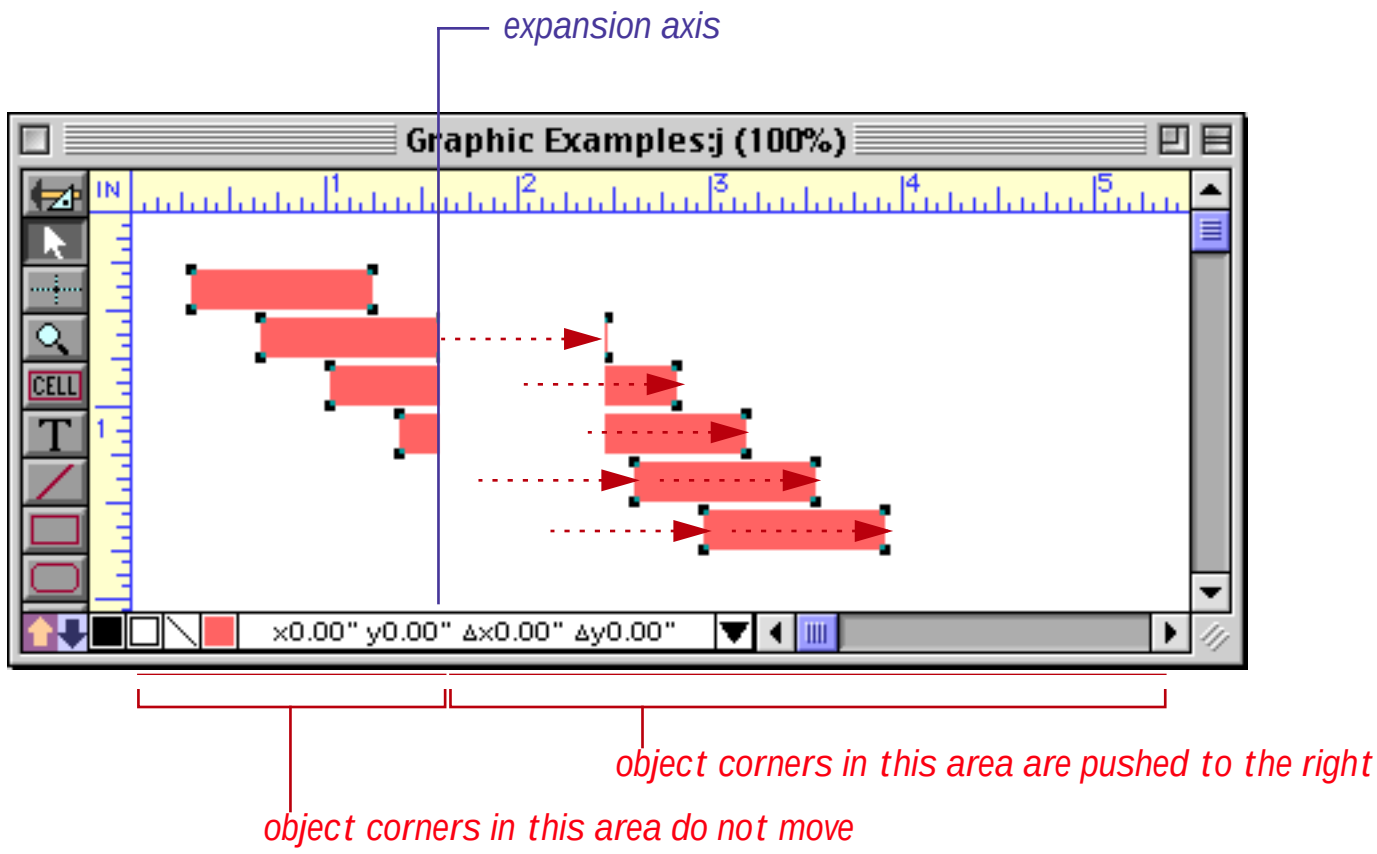
Here's another example. We'll start with six objects.



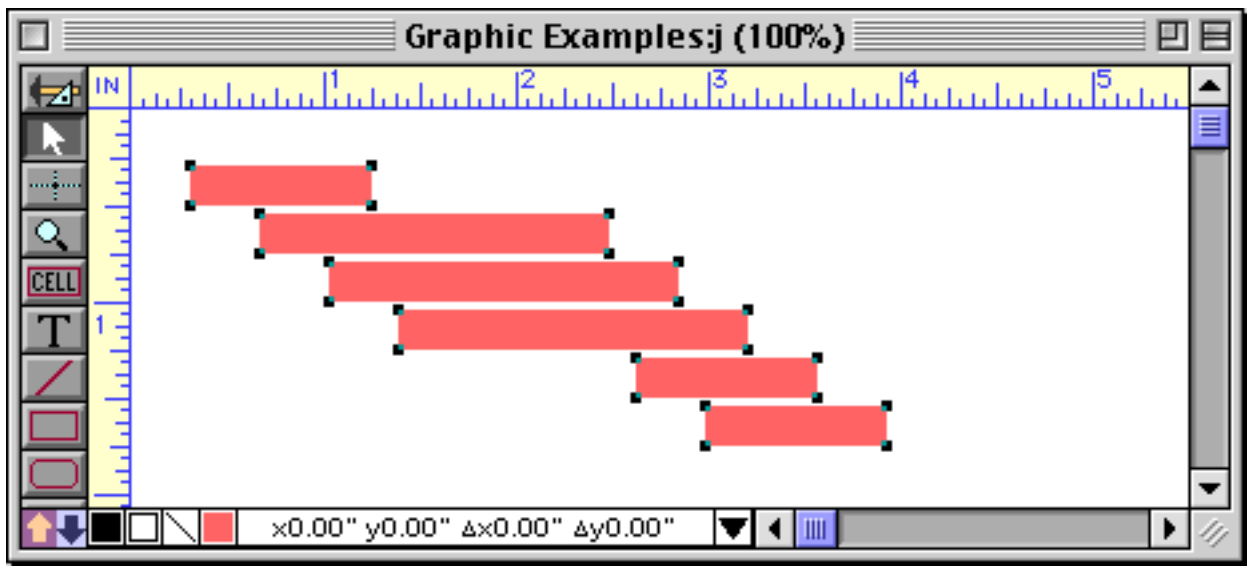
We'll slide the right edge of the second object to expand the object about an inch.



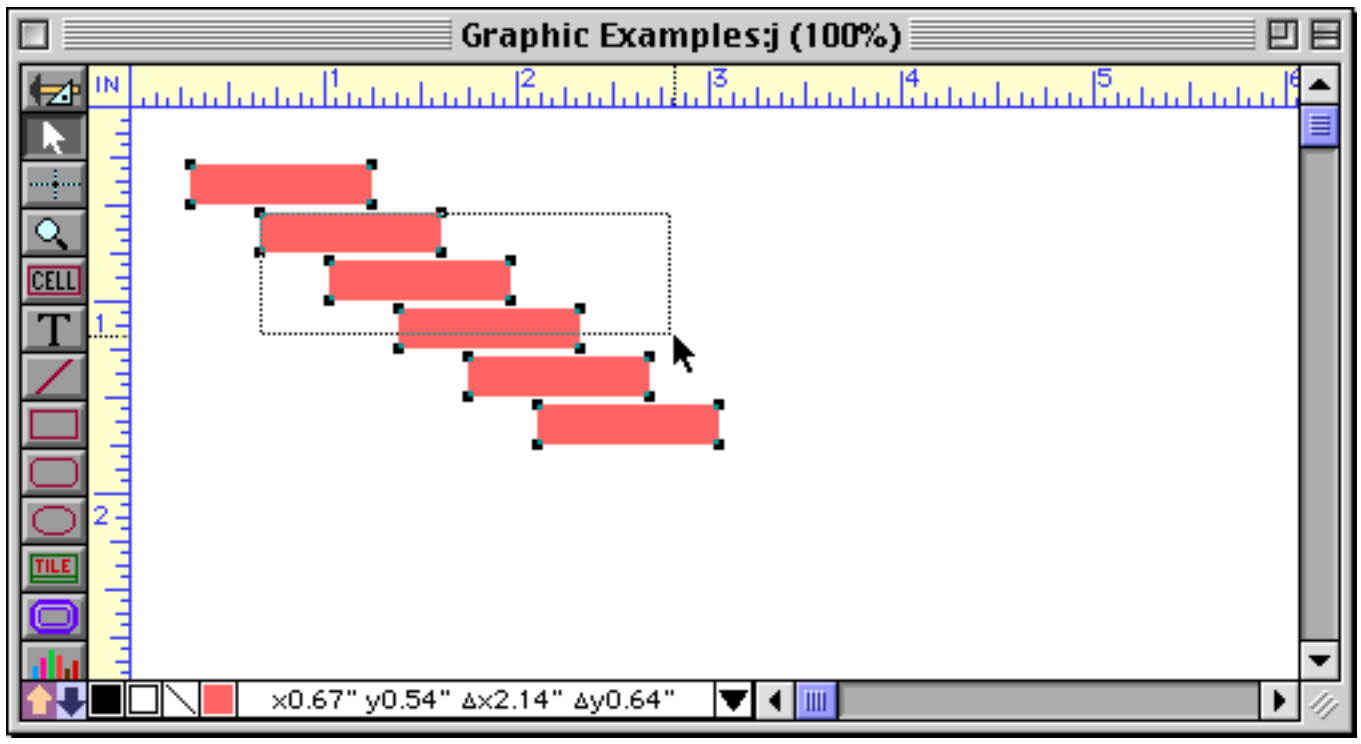
Here's how Panorama will adjust the other five objects.



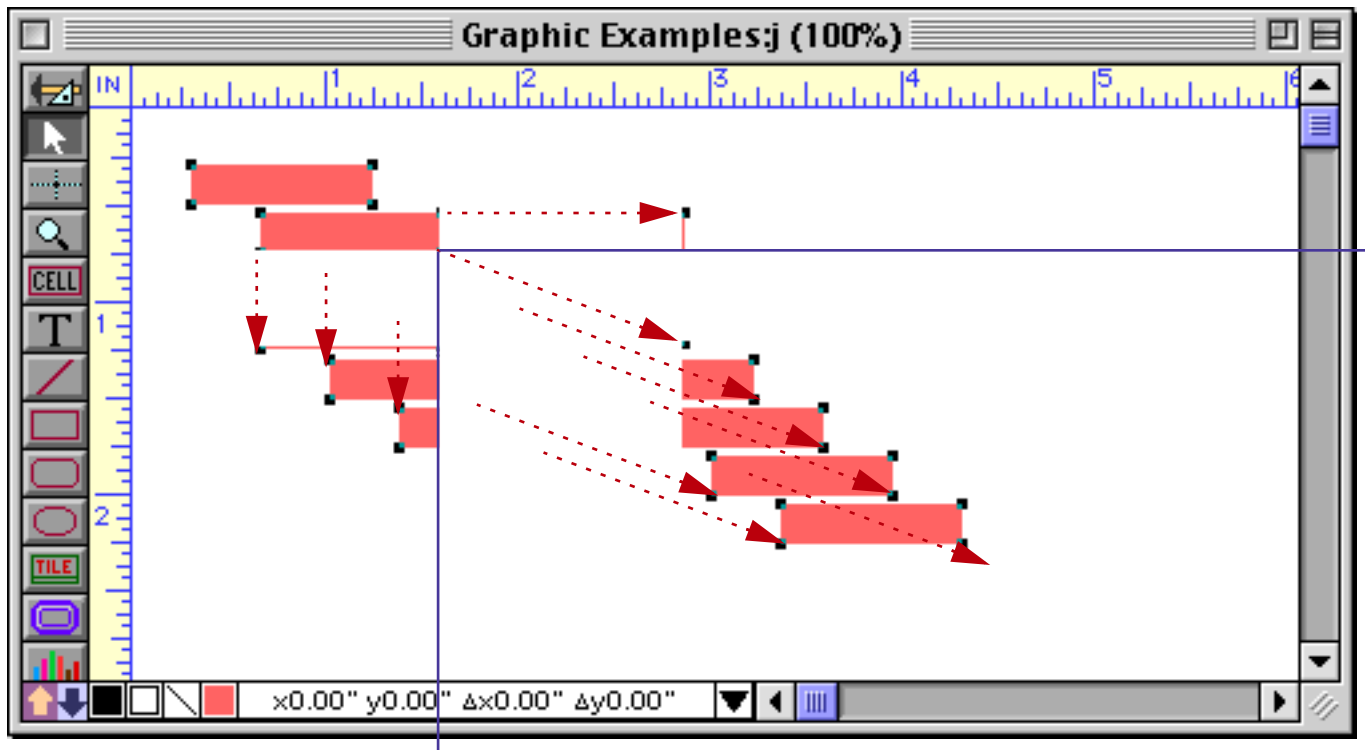
The first object is completely unaffected. This is because we expanded to the right and this object is completely to the left of the expansion axis. The third and fourth objects straddle the expansion axis. Therefore, they grow to match the growth of object number 2. The final two objects are completely to the right of the expansion axis. Therefore, they stay the same size and slide to the right. Here's the final result when you release the mouse.



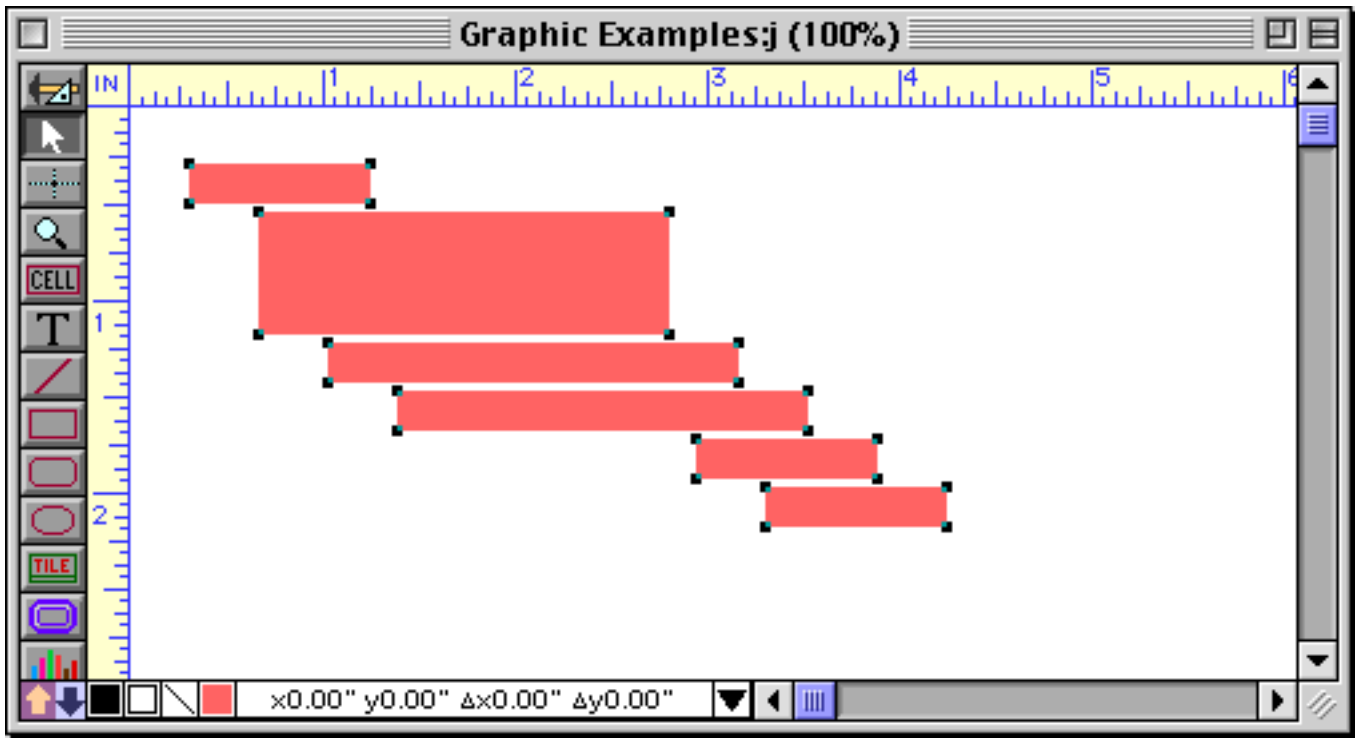
Cluster resize isn't limited to horizontal expansion. In this example we'll stretch the second object both horizontally and vertically.



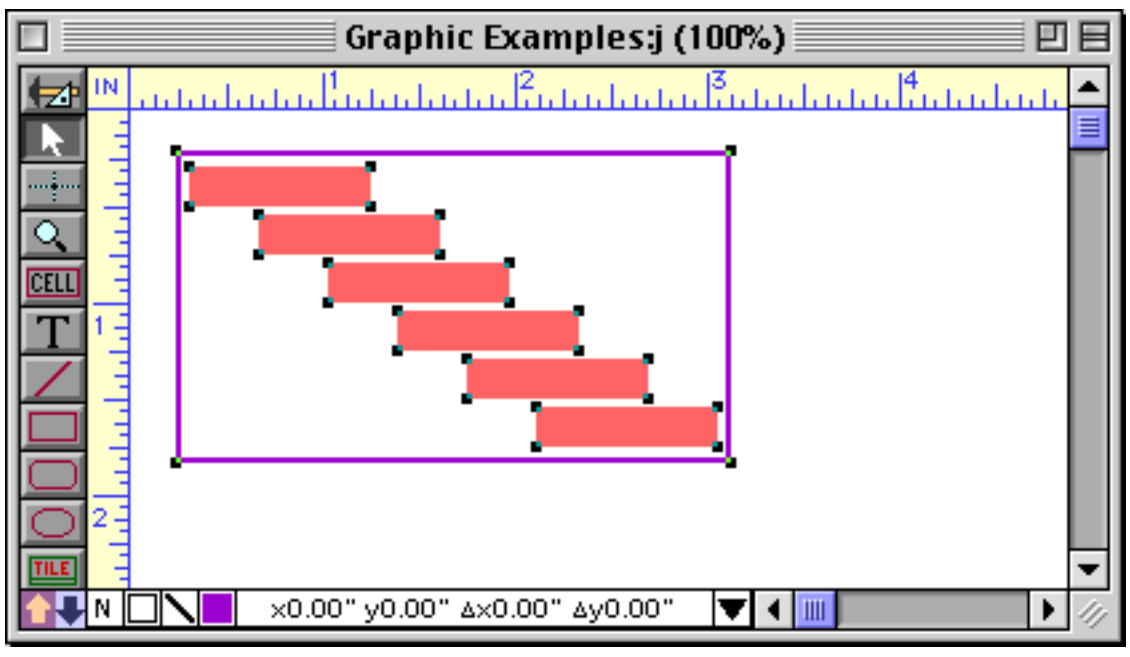
Now there are two expansion axis, one horizontal and one vertical. Depending on what quadrant they are in, a point will stay in place, move horizontally, move vertically, or move diagonally.



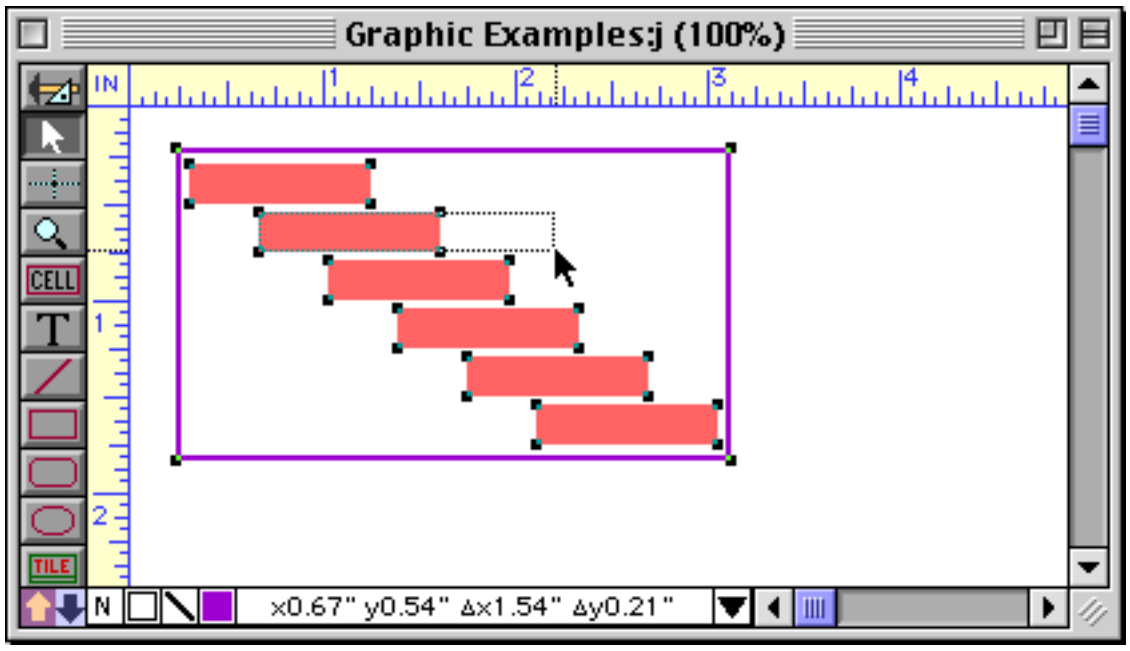
Here's the final result.



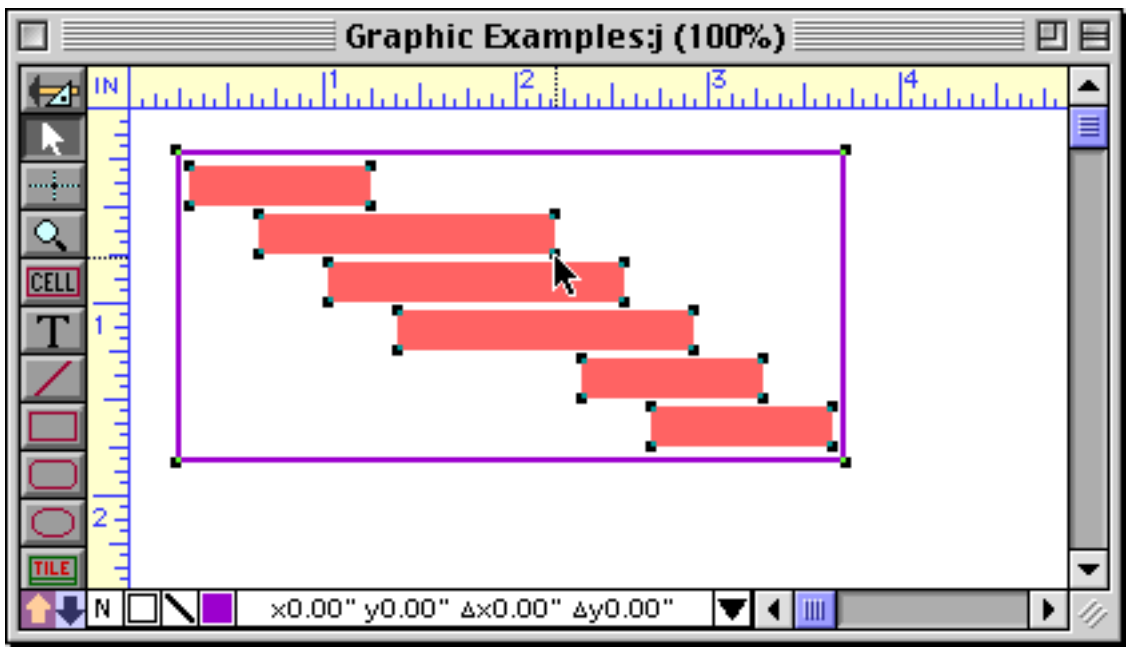
Cluster resize affects all selected objects, even when objects are nested inside each other. If you select nested objects and then change the size of the innermost object, cluster resize will adjust the size of the larger object. This makes sense because the right edge of the outer object is to the right of the right edge of the inner object (say that three times fast!). A real world example of nested objects is a border around a table (or borders around individual columns in a table). Cluster resize makes sure that if you resize one of the columns in the table the borders will adjust as well. Just make sure that you always change the innermost object, and cluster resize will take care of the rest.To illustrate cluster resize with nested objects we've added a border around our six objects.



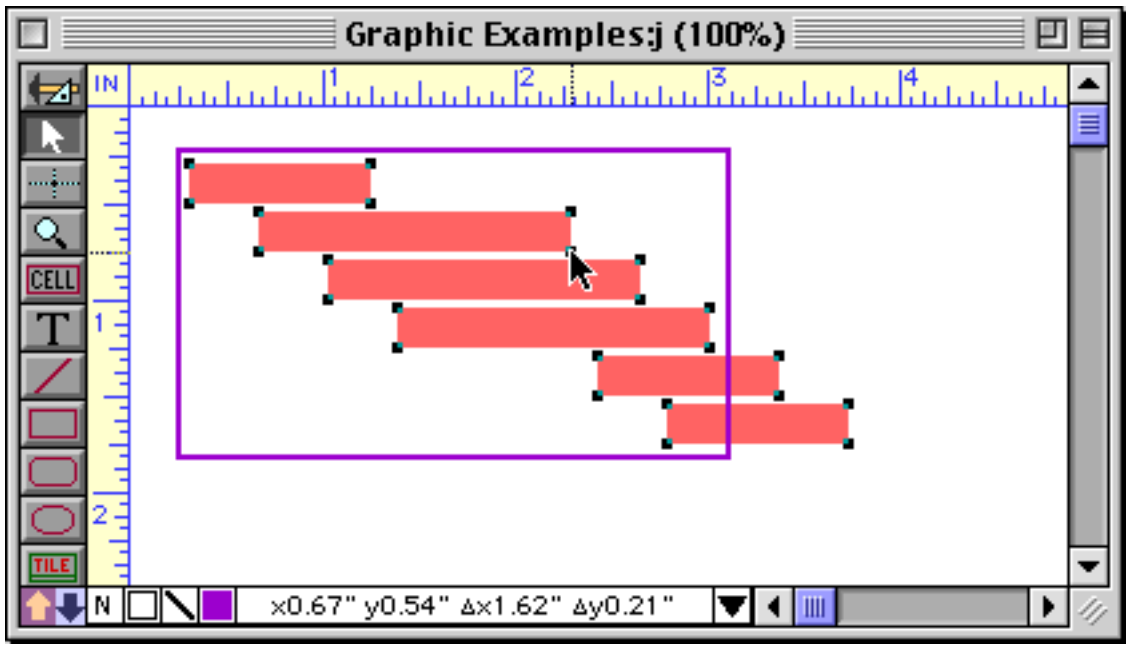
Once again we expand the second object.



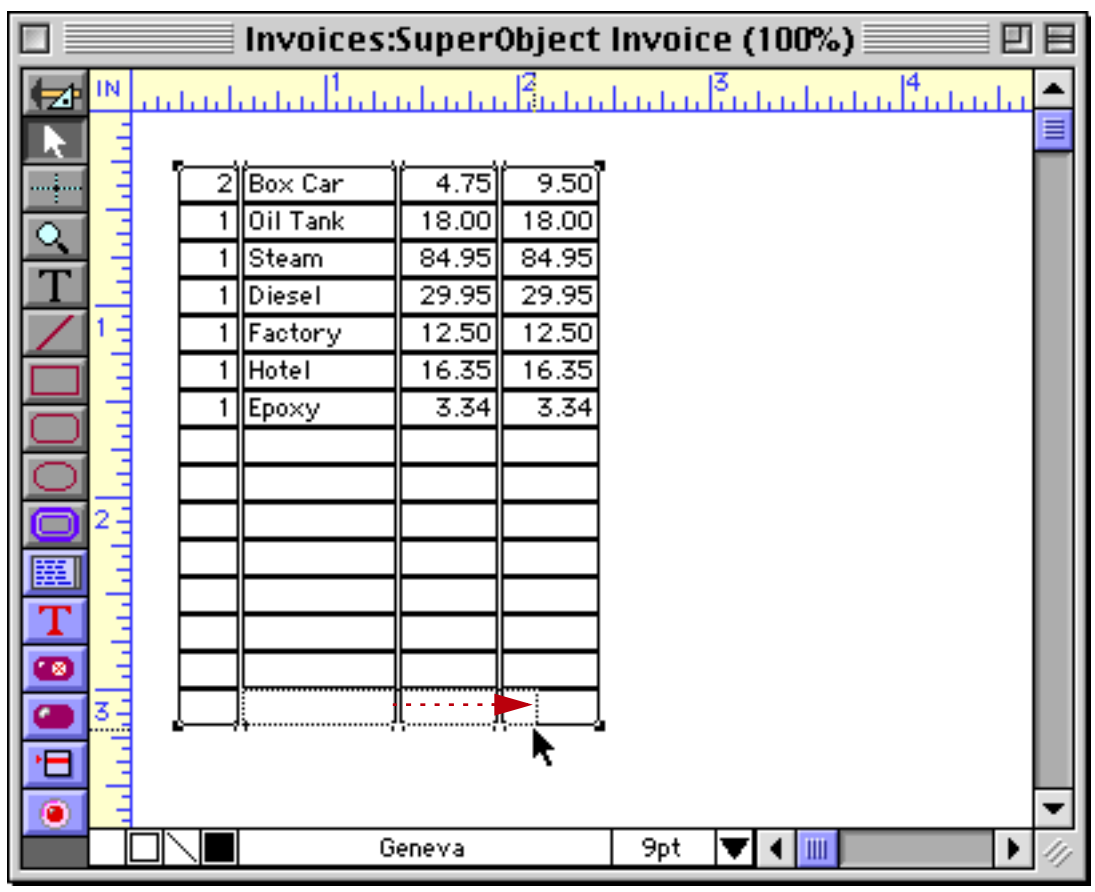
The border also expands. Since it straddles the expansion axis, the left side of the border stays put while the right side expands to fit the expanded objects inside it.



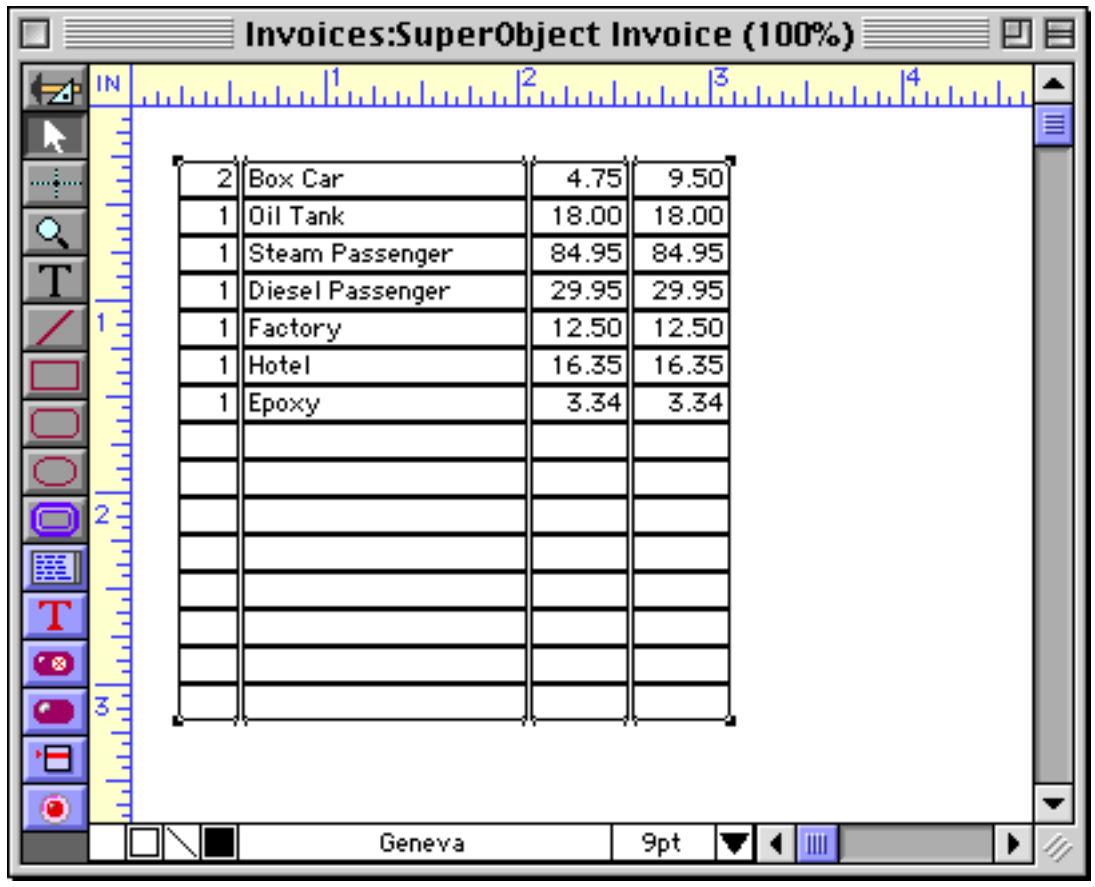
Only selected objects are affected. If the border is not selected, it won't expand.



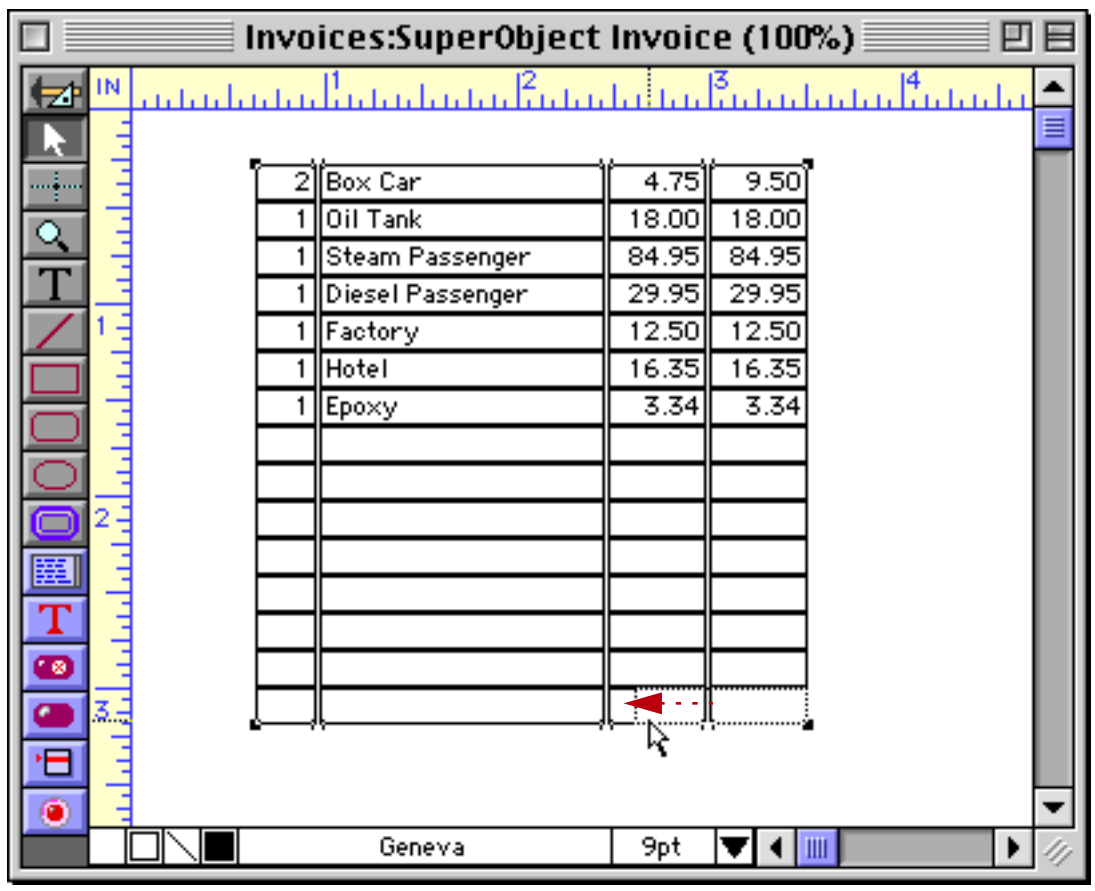
Cluster resize makes it easy to adjust the size of rows or columns in a table. Just select all of the objects in the table, then adjust the size of any object within the table.



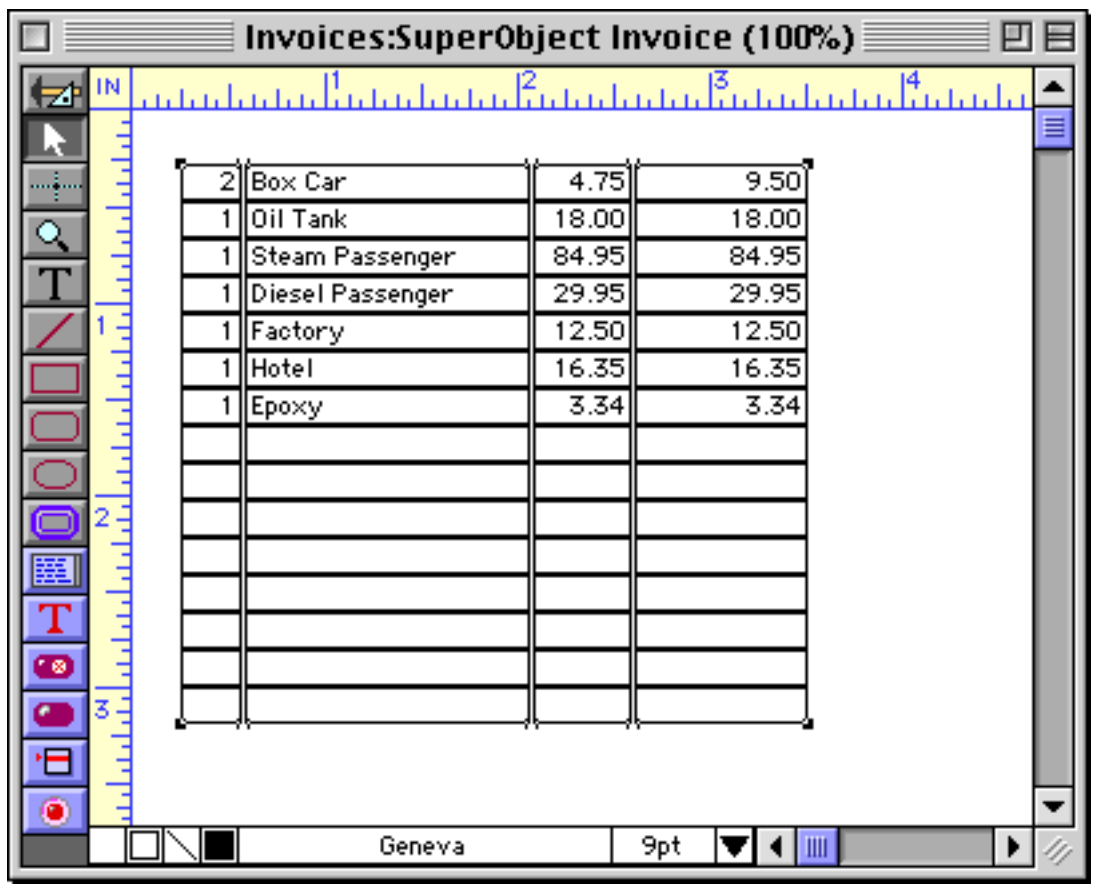
Cluster resize adjusts the other objects to maintain the table structure.



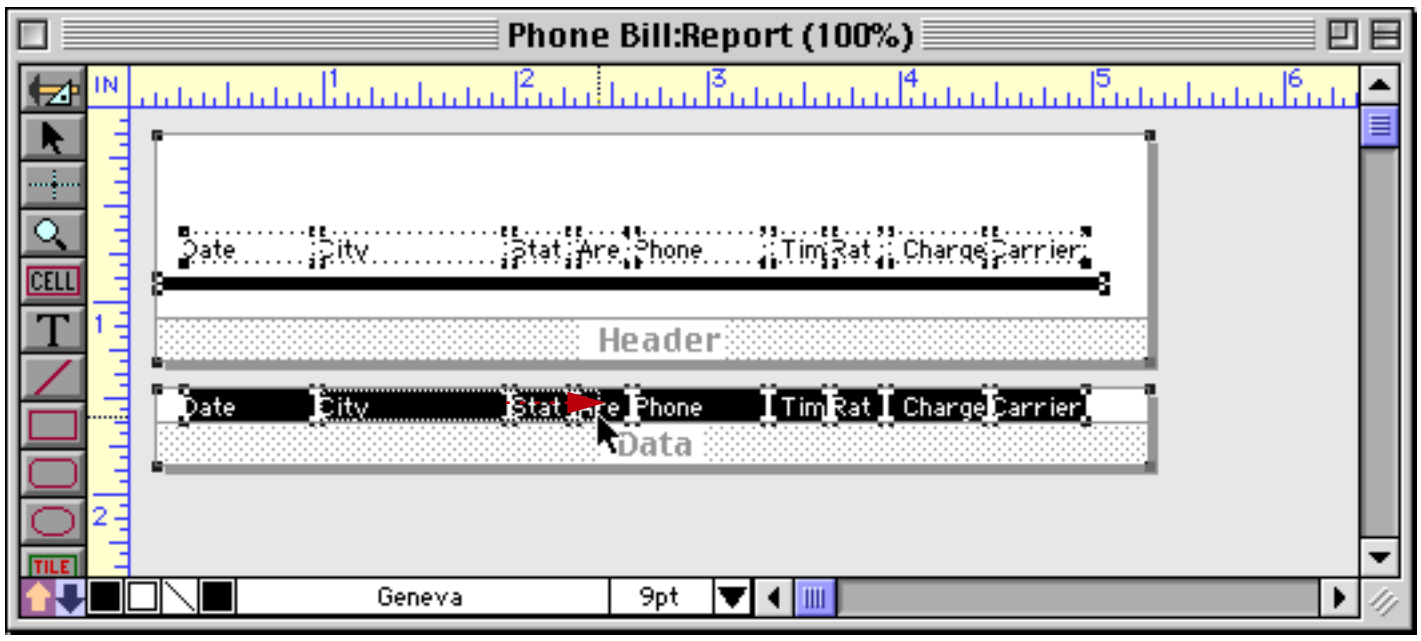
So far all of our examples have shown expanding to the right and / or down. However, you can also expand to the left, or shrink. In this case we'll expand the last column to the left. (When you have this many handles packed into a small area, it can be difficult to hit the one you want. To make it easier we held down the S key, which expands the handles. See "Resizing Without Handles" on page 516 for more information on this technique. We also held down the **Shift** key to make sure that the expansion was entirely horizontal and not diagonal.)



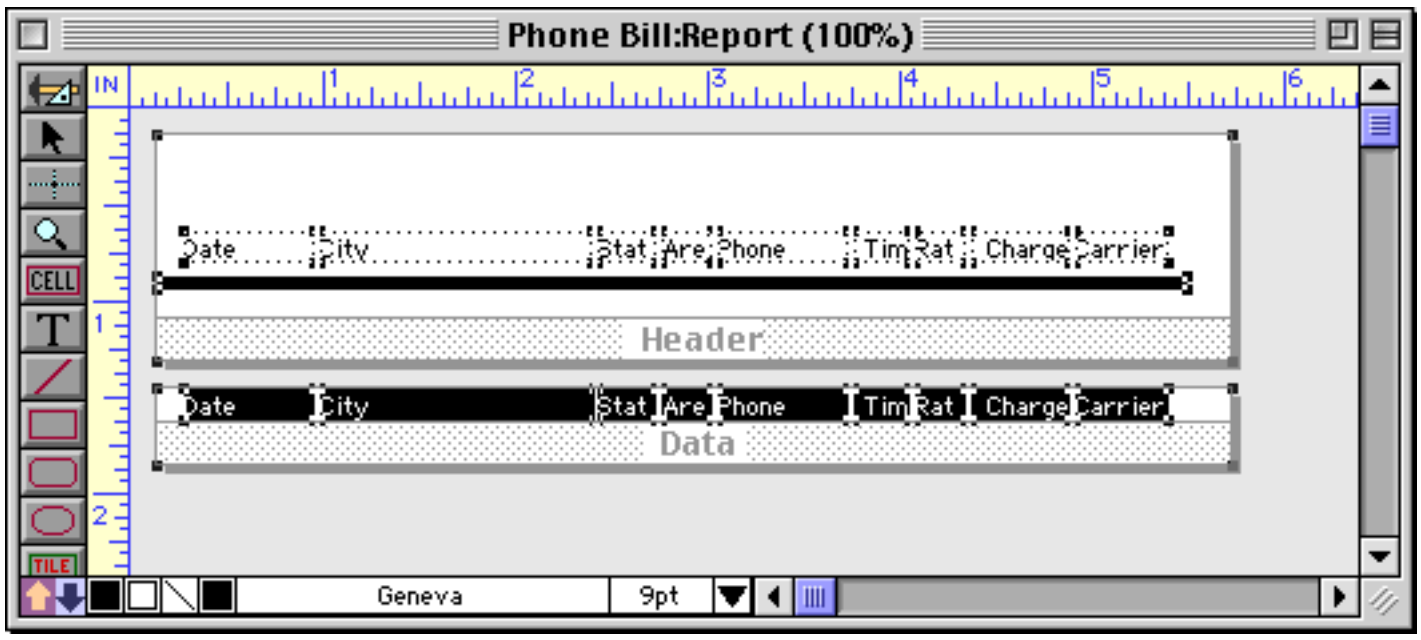
When the mouse is released the entire table is adjusted.



Cluster resize is also great for working with reports. If you keep your report tiles lined up, then changing the width of an item in the body of the report automatically adjusts the width of the same item in the report header. (Remember that all the objects must be selected. Only selected objects will be adjusted.) In this example all the objects are selected and we expand the width of the City field.



Cluster resize automatically adjusts all of the other objects, including the report tiles.



By the way, cluster resize also works when you nudge the size of an object using the arrow keys (see "Nudging the Size of an Object" on page 513). Select the objects, then click on the handle you want to nudge. As you use the arrow keys to resize the object, cluster resize will automatically adjust the other objects. You can combine this with the crosshair cursor (see "Nudging to the Crosshair Cursor" on page 515) to quickly and accurately nudge a collection of objects exactly into place.

Don't forget that when you are resizing an object you can use the **Shift** key to make sure that only the height or width changes, but not both. For example, if you are adjusting the width of a column using cluster resize, hold down the **Shift** key so that the height of the row doesn't change also. You can also hold down the S key to make sure that you are resizing, and not dragging. See "Resizing Without Handles" on page 516 for more details on this technique.

Cluster Resize Troubleshooting

Sometimes you may run across a situation where cluster resize doesn't seem to work. You change the width of a column, but some of the objects don't get adjusted. What's the problem?

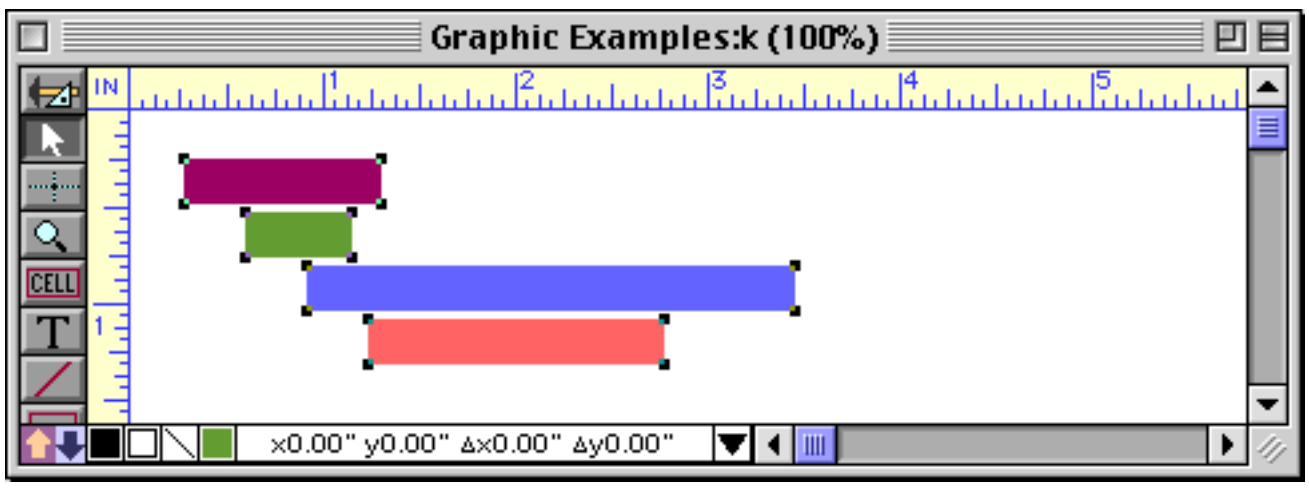
This problem occurs when not all the objects in the column are lined up on the right. Usually this is because the objects are not quite the same width. If one of the objects in a column is 1 or 2 pixels shorter than the other objects the short object won't get adjusted.

The solution is to make sure your objects are all lined up before you start. If necessary, you can use the **Align** command to force all the objects into alignment See "Aligning Objects" on page 553 to learn about this command.

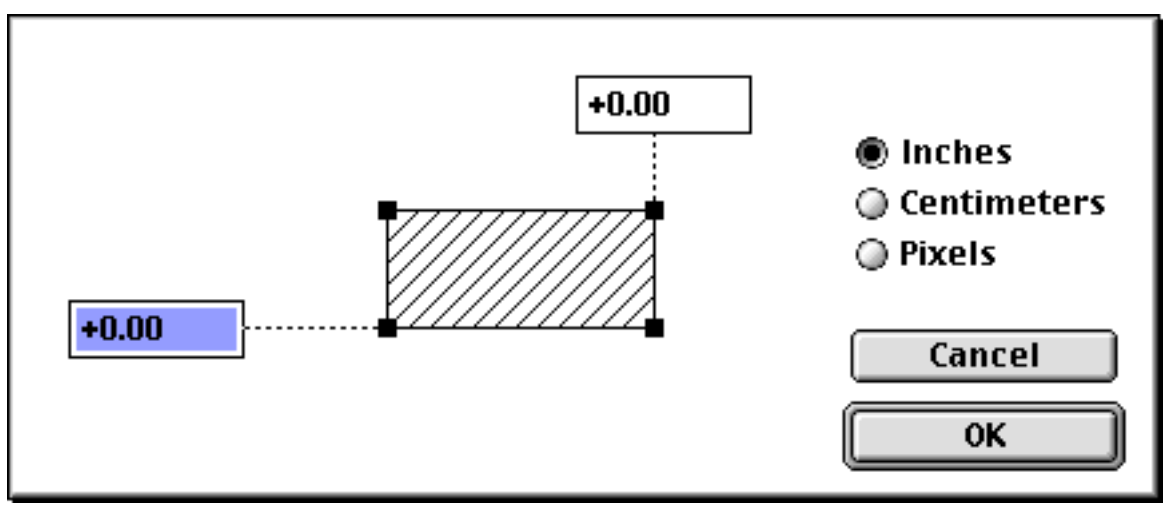
Setting Exact Dimensions of Multiple Objects

When the **Dimensions** dialog (in the Edit menu and the Graphic Control Strip) is used with a single object, you can specify both the position and size of the object. But if several objects are selected, the **Dimensions** dialog cannot specify the position—only the size of the object. You can use this dialog to set the width and / or height of all the selected objects—for example, you can make all the selected objects 2 inches wide. Or you can use the dialog to increase or decrease the size of each selected object—for example, you can make every selected object 1⁄2 inch wider.

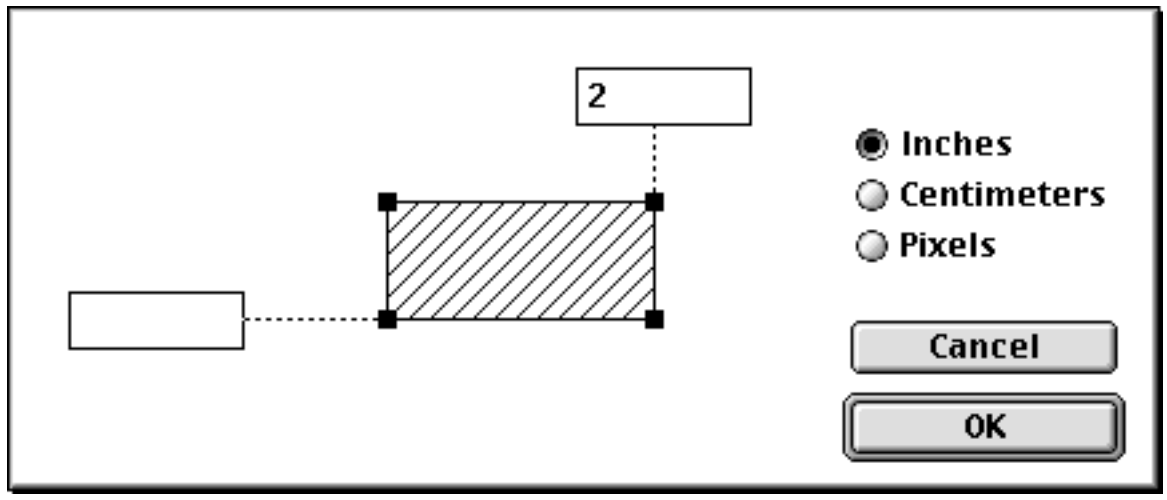
To set the height or width of several objects at once, first select the objects. Then open the **Dimensions** dialog. Type in the new height and / or width, then press **Ok**. All of the selected objects will be set to the specified height and / or width. For example, suppose you have a collection of miscellaneous objects like this:



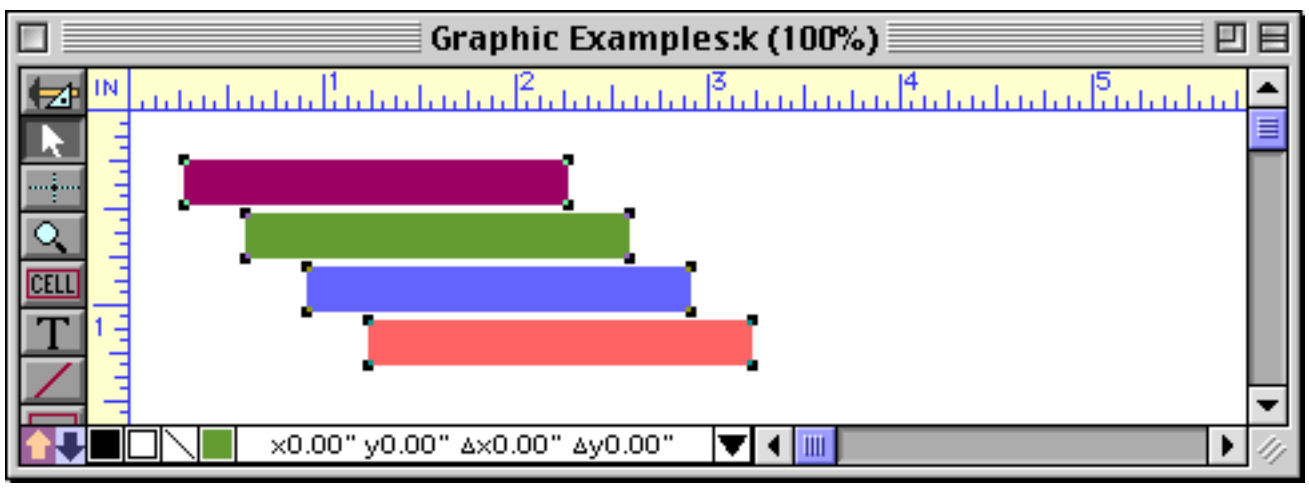
To make all these objects 2 inches wide, open the Dimensions dialog (either by clicking in the dimensions area of the Graphic Control Strip or by choosing **Dimensions** from the Edit menu).



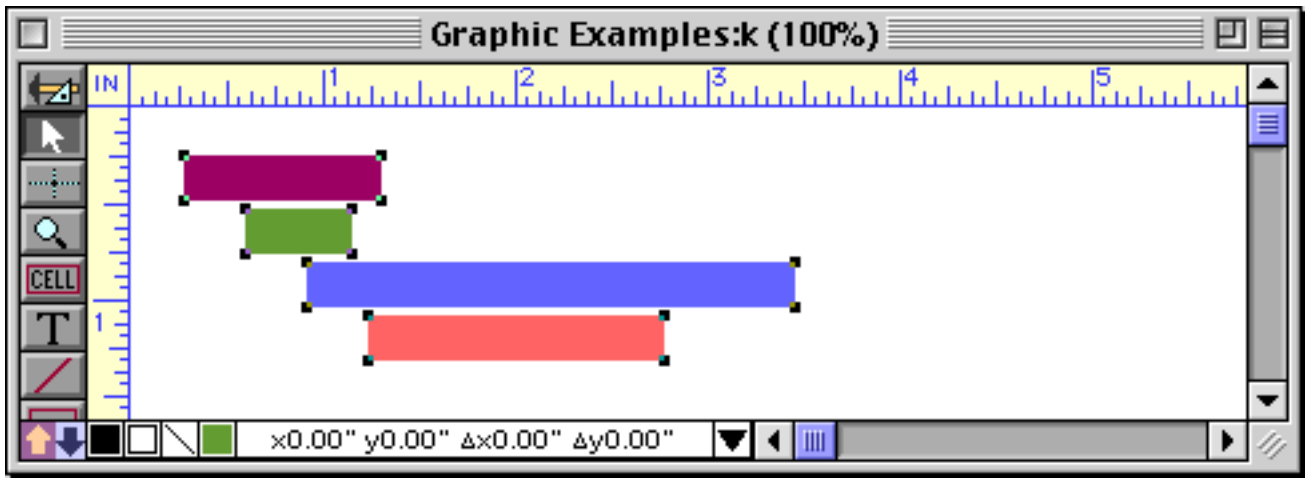
Set the width to 2 and erase the height.



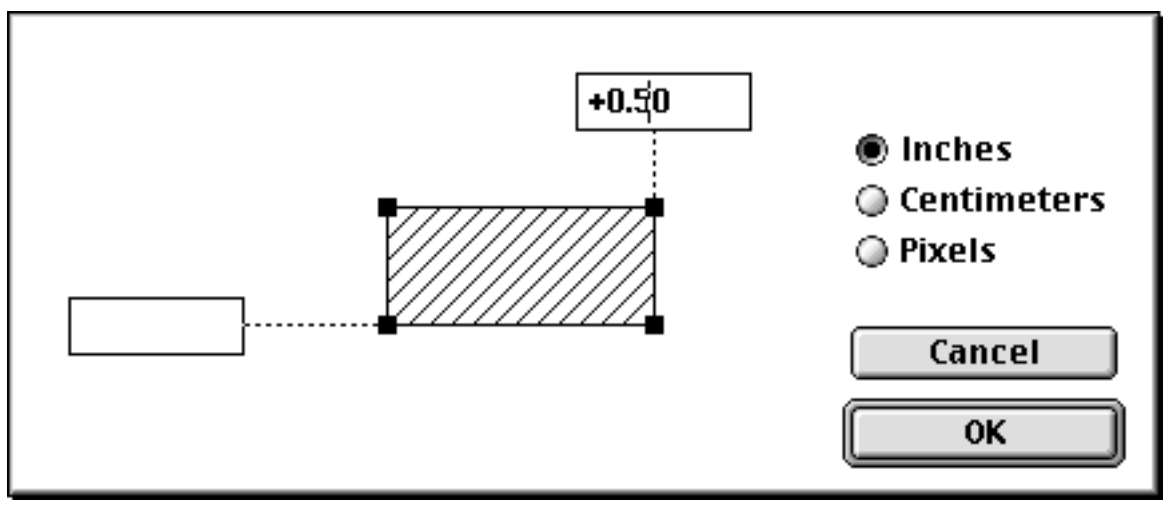
When the **Ok** button is pressed all of the selected objects will be changed to 2 inches wide while retaining their original locations and heights.



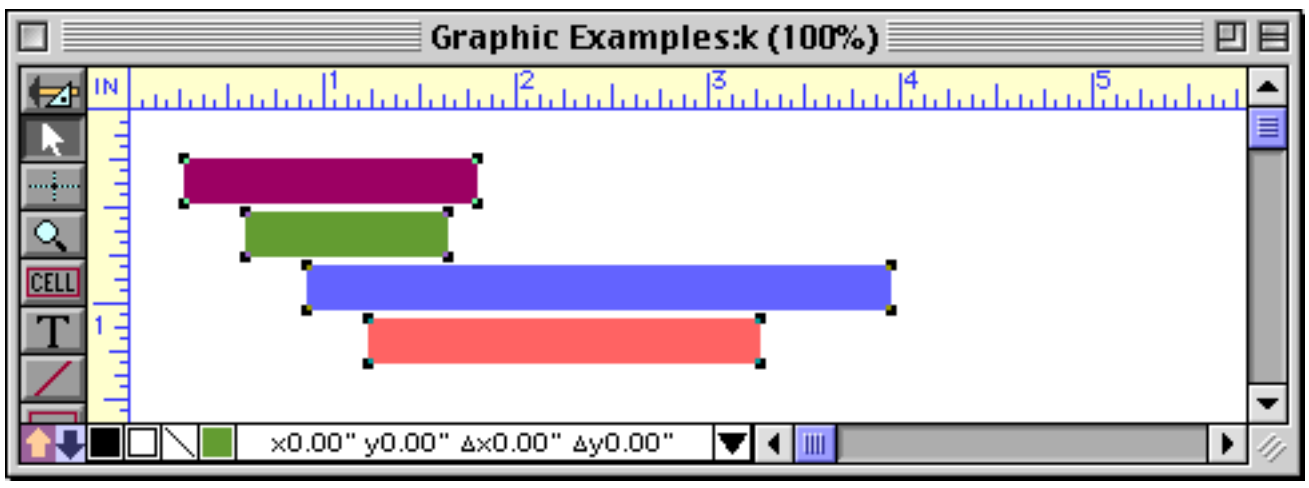
To increase or decrease the size of several objects at once, first select the objects.



Then open the **Dimensions** dialog. Type in the increase or decrease amount, with a + (increase) or - (decrease) sign in front of the number. For example, to make all selected objects 1⁄2 inch wider enter +0.5.



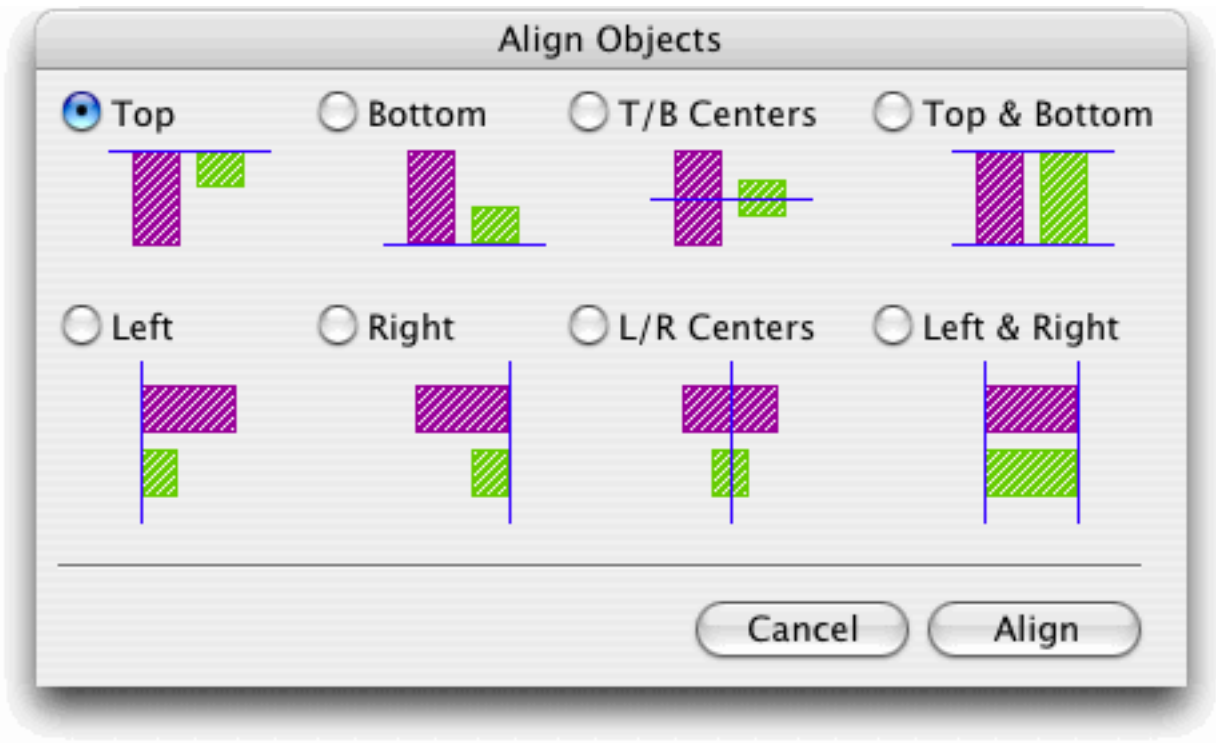
When the **Ok** button is pressed all of the objects will get 1/2 inch wider.



If you are not careful, the **Dimension** dialog can turn a form into a big mess in a hurry. If this happens, don't panic—just **Undo**!

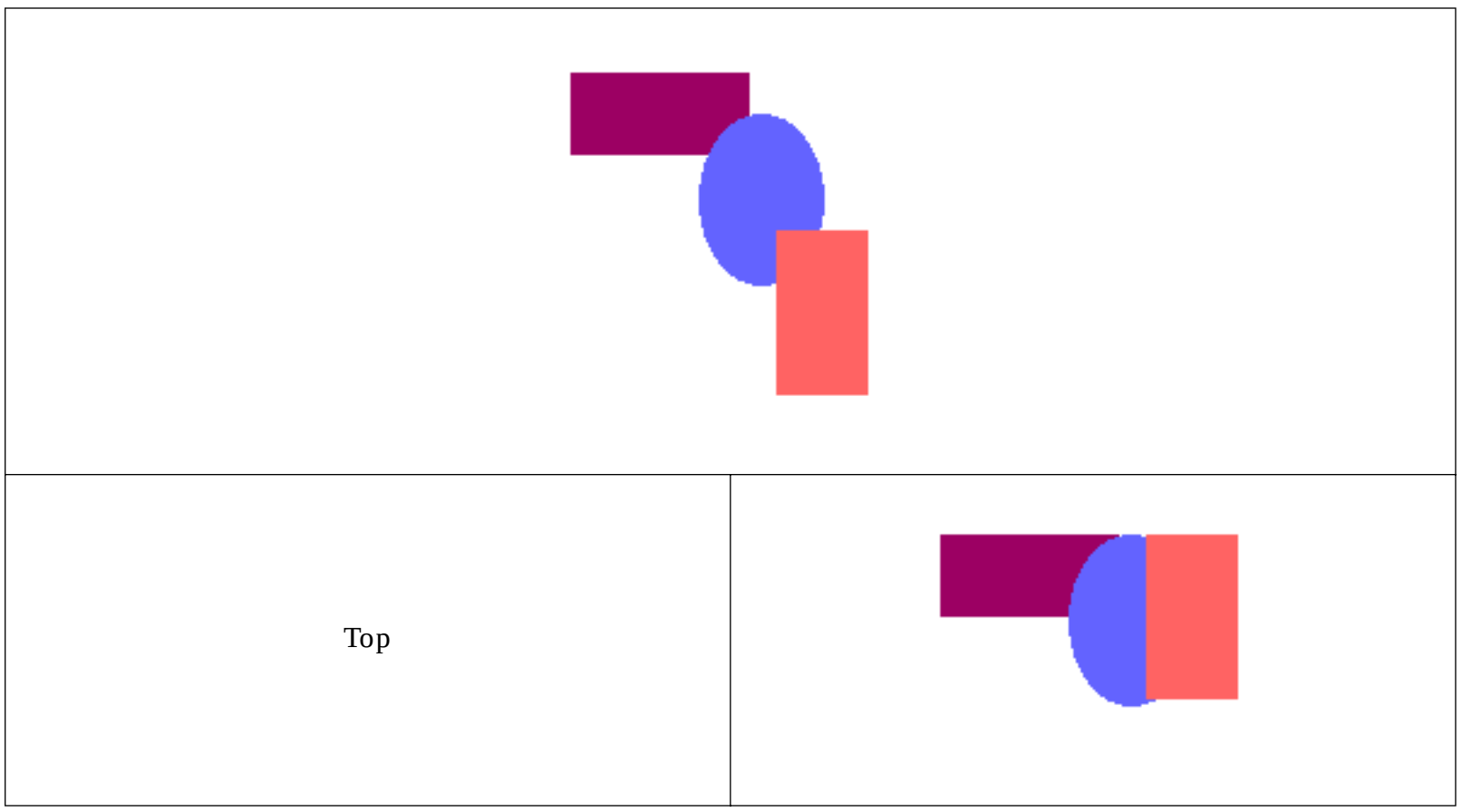
Aligning Objects

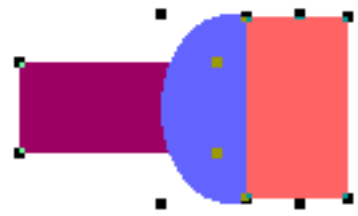
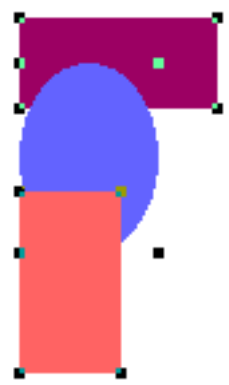
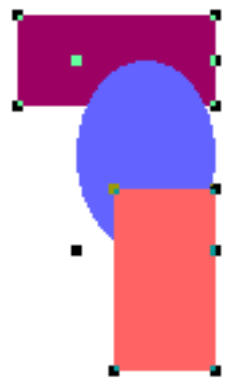
If you need to line up several objects, you can either do it by eye or you can let the **Align** command (in the Arrange menu) do it for you. The Align dialog gives you eight options for aligning objects in different directions.

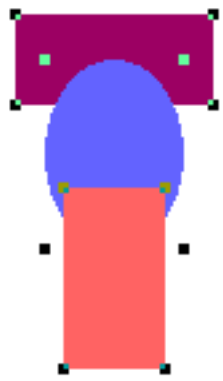
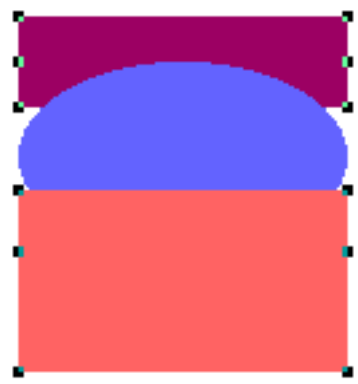


Most of the alignment options simply shift the objects to align them, but the **Left & Right** and **Top & Bottom** options actually change the sizes of the selected items. Instead of shifting the objects, these options actually expand the selected objects to make them all the same width or all the same height. If you ask for **Left & Right**, all the selected objects will be expanded to the width of the widest object. If you ask for **Top & Bottom**, all the selected objects will be expanded to the height of the tallest object.

This table shows how a collection of three objects is affected by the different alignment options.

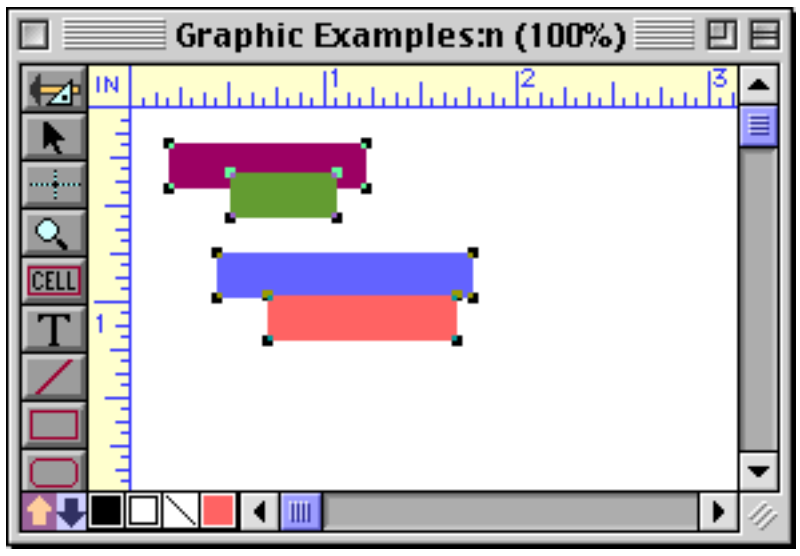


Bottom	
T/B Centers	
Top & Bottom	
Left	
Right	

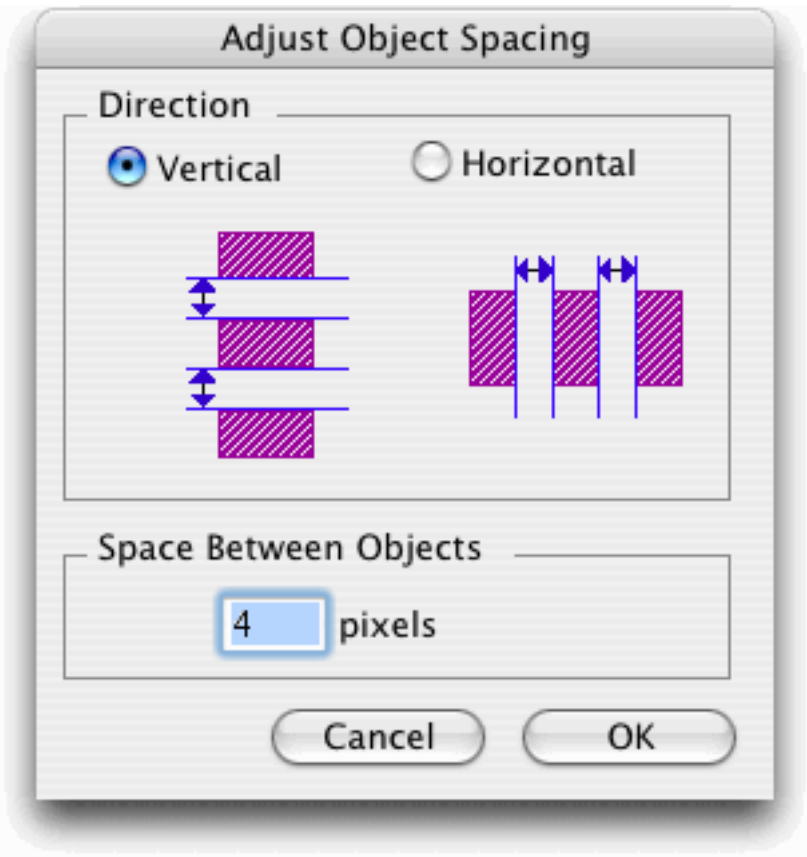
L / R Centers	
Left & Right	

Adjusting Spacing Between Multiple Objects

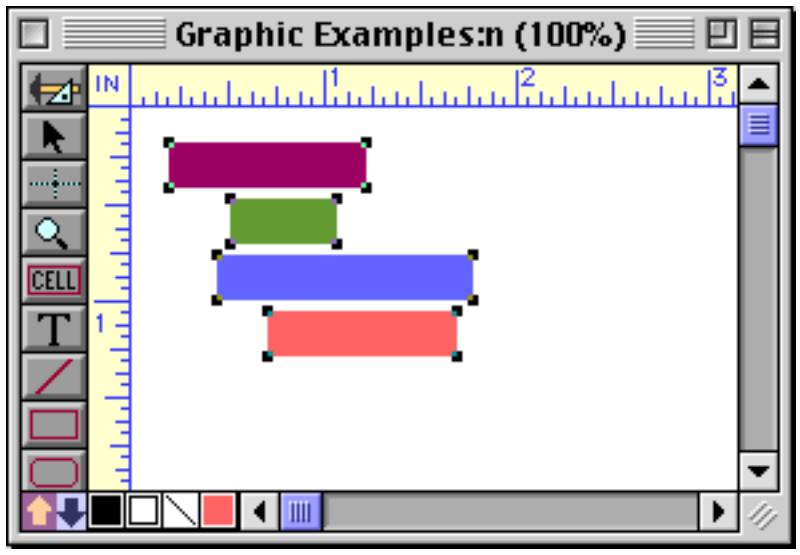
The **Spacing** command (in the Arrange menu) allows you to adjust the vertical or horizontal spacing between multiple objects. The dialog will shift the selected objects so that they are evenly spaced. For example, you could start with a somewhat random collection of objects like this.



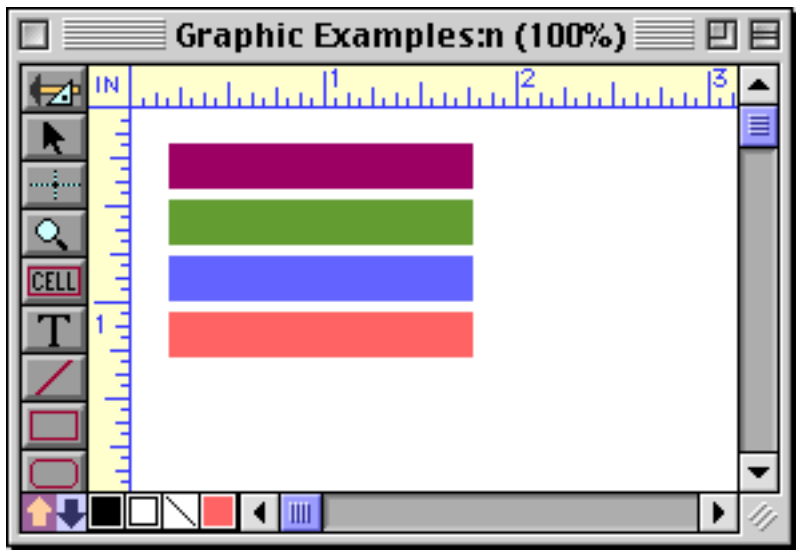
Open the **Spacing** dialog and set the spacing to 4 pixels between the objects.



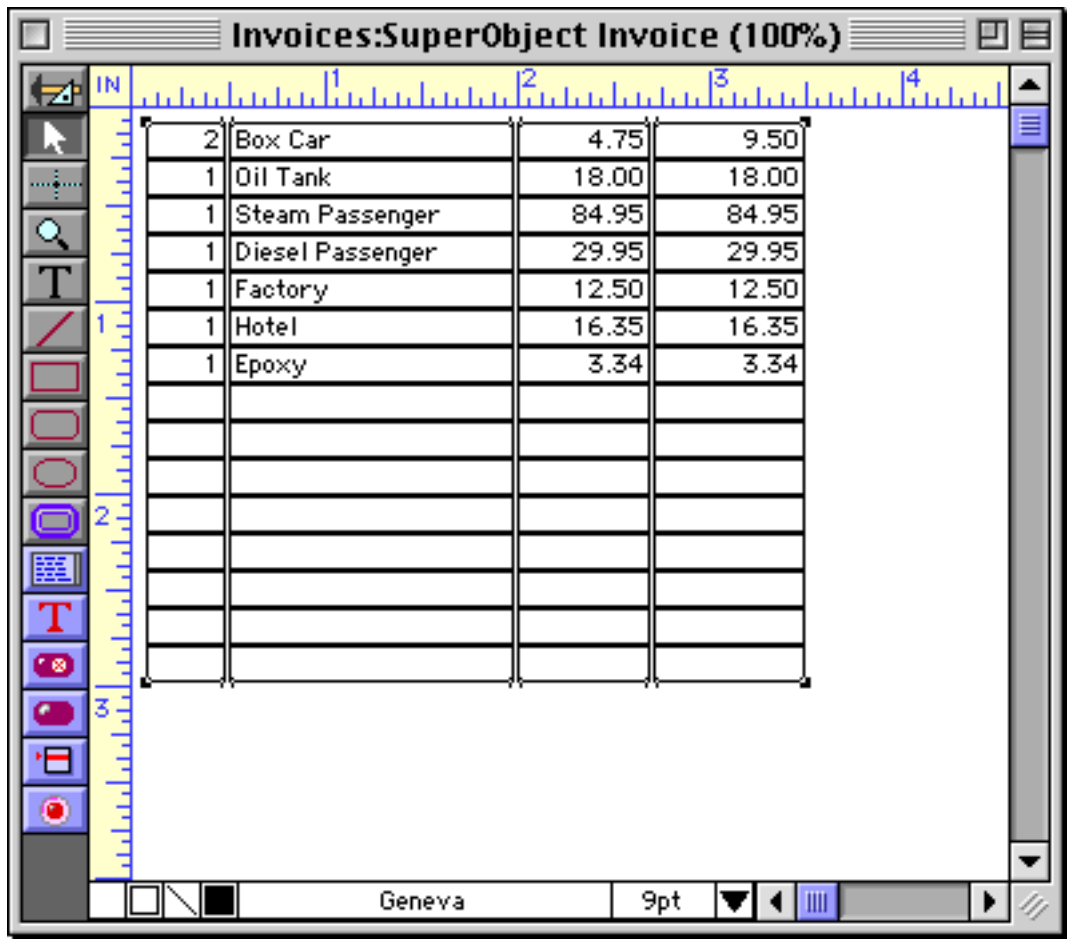
When the **Ok** button is pressed the objects will slide vertically into place.



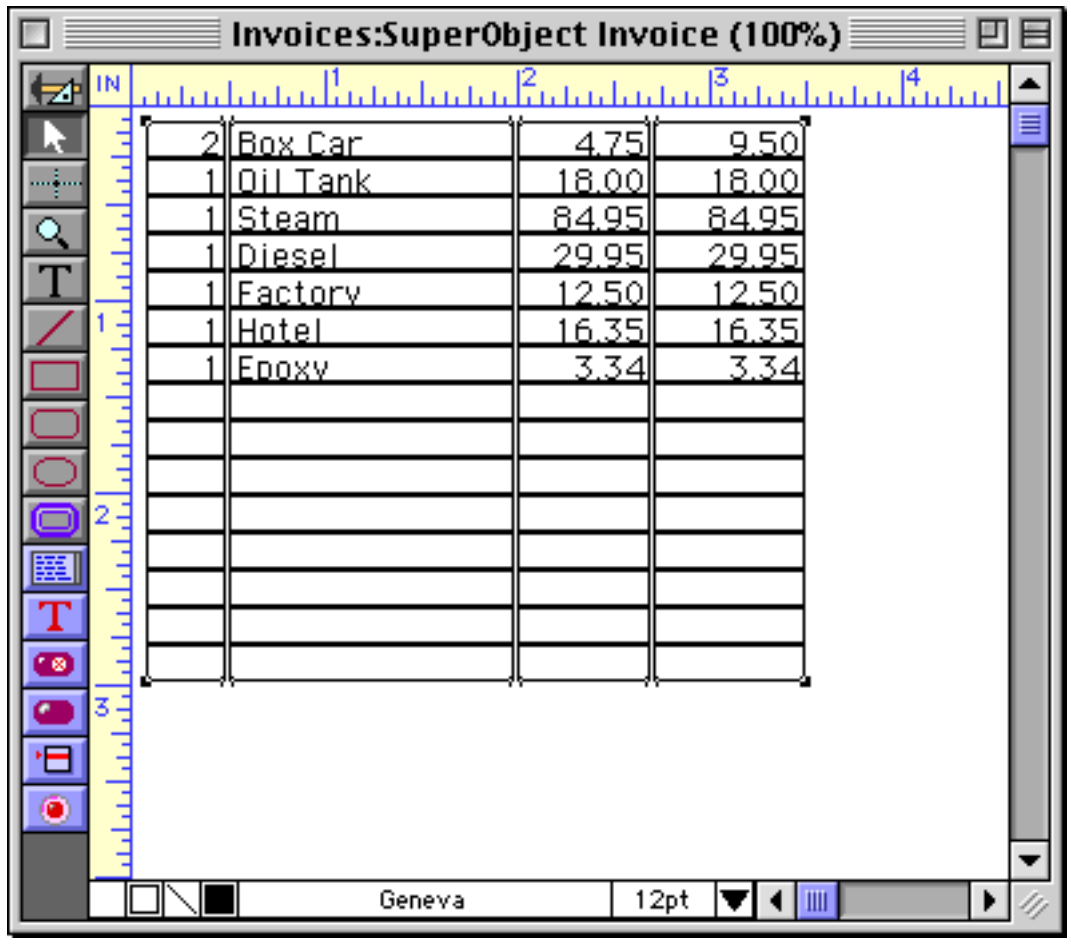
To finish our cleanup we'll use the **Align** command to make all of the objects the same width (by selecting Align **Left & Right**).



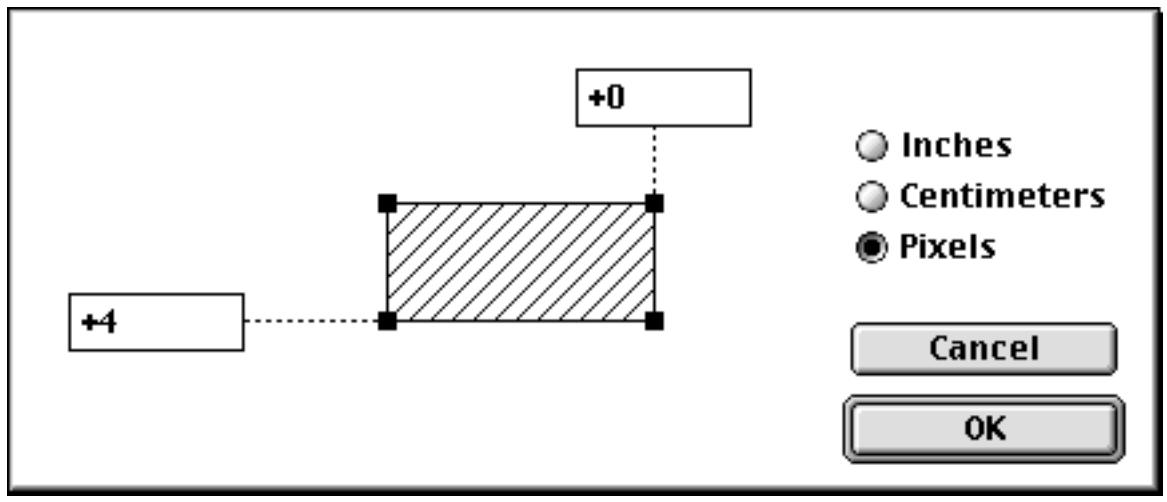
If the selected objects are arranged in a table, the **Spacing** dialog will preserve the table structure. For example, if the font size of a table is increased the individual rows in the table will overlap. This can be fixed with the Spacing dialog. To illustrate this, we'll start with a table that contains 9 point text.



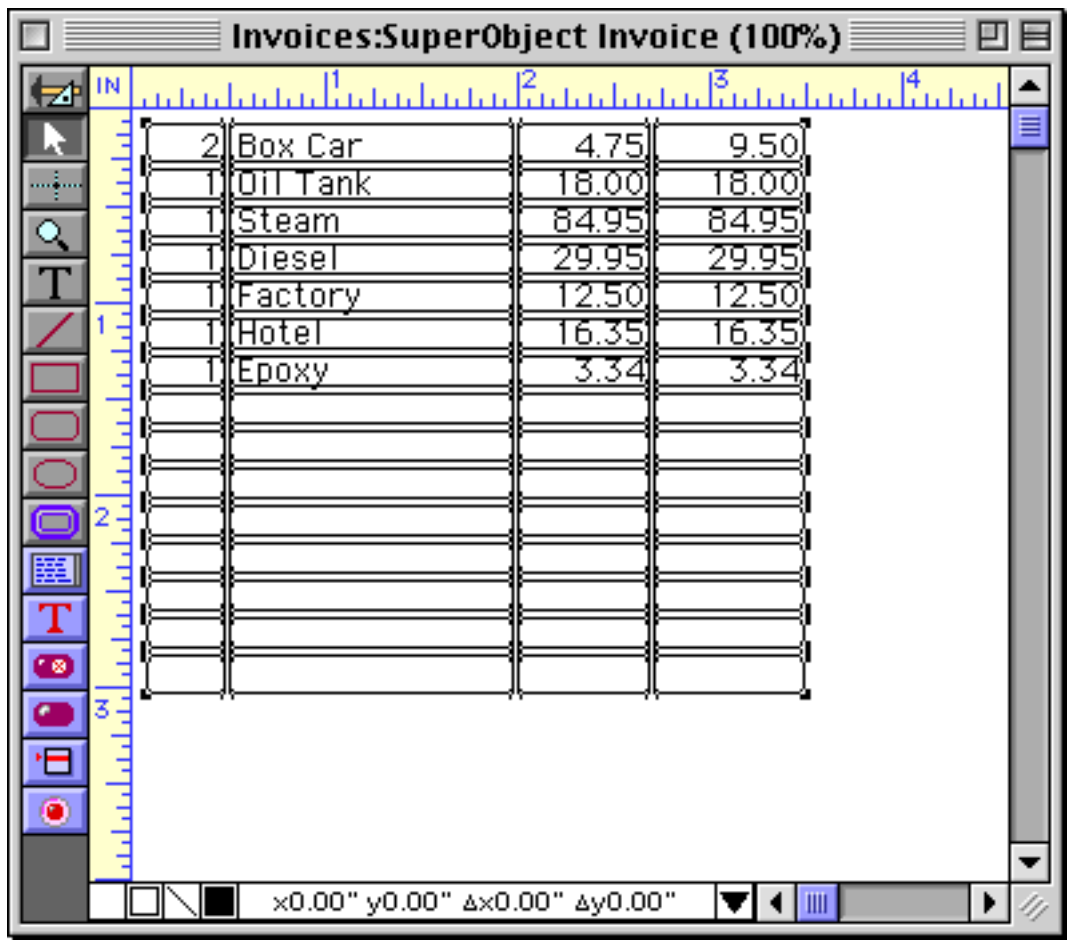
We'll start by converting the text from 9 points to 12 points. Our table doesn't look so neat anymore!



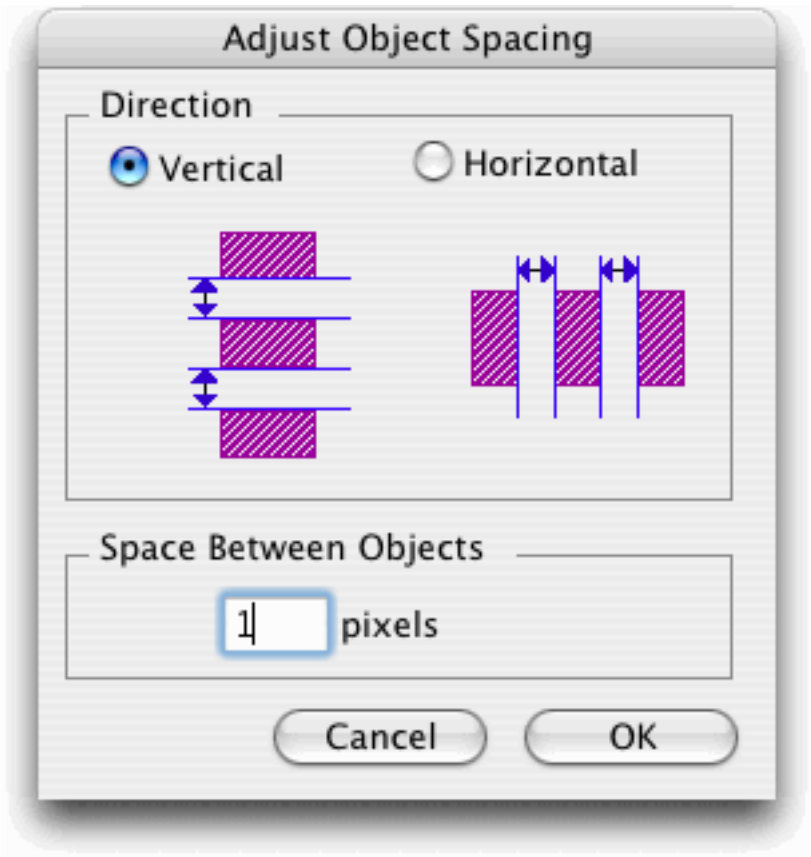
The next step is to use the **Dimensions** dialog to increase the height of each object by four pixels (see "Setting Exact Dimensions of Multiple Objects" on page 550).



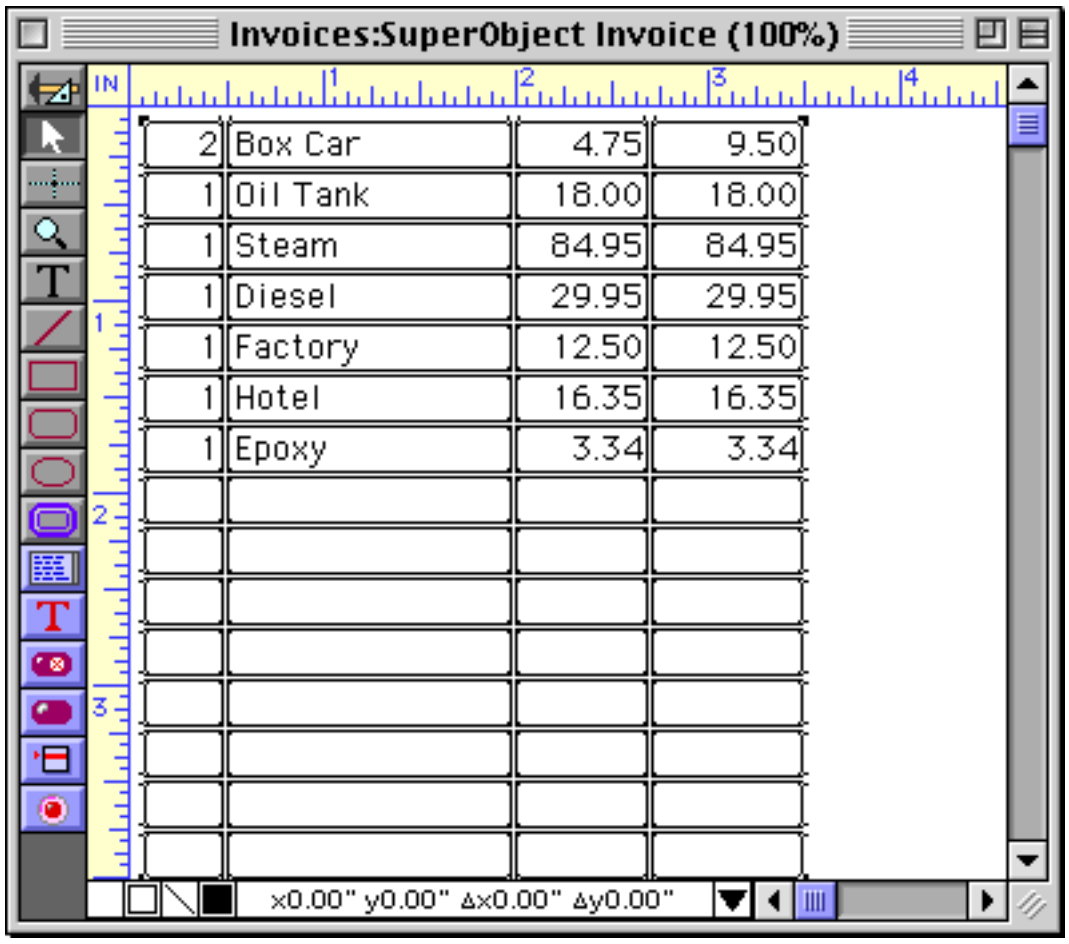
The table still doesn't look so hot.



The final step is to use the **Spacing** dialog to restore the vertical spacing between the rows.



When the **Ok** button is pressed, the table spread out and now looks great. This may have seemed like a fairly complex process, but it is much easier than manipulating all 60 of these objects individually!



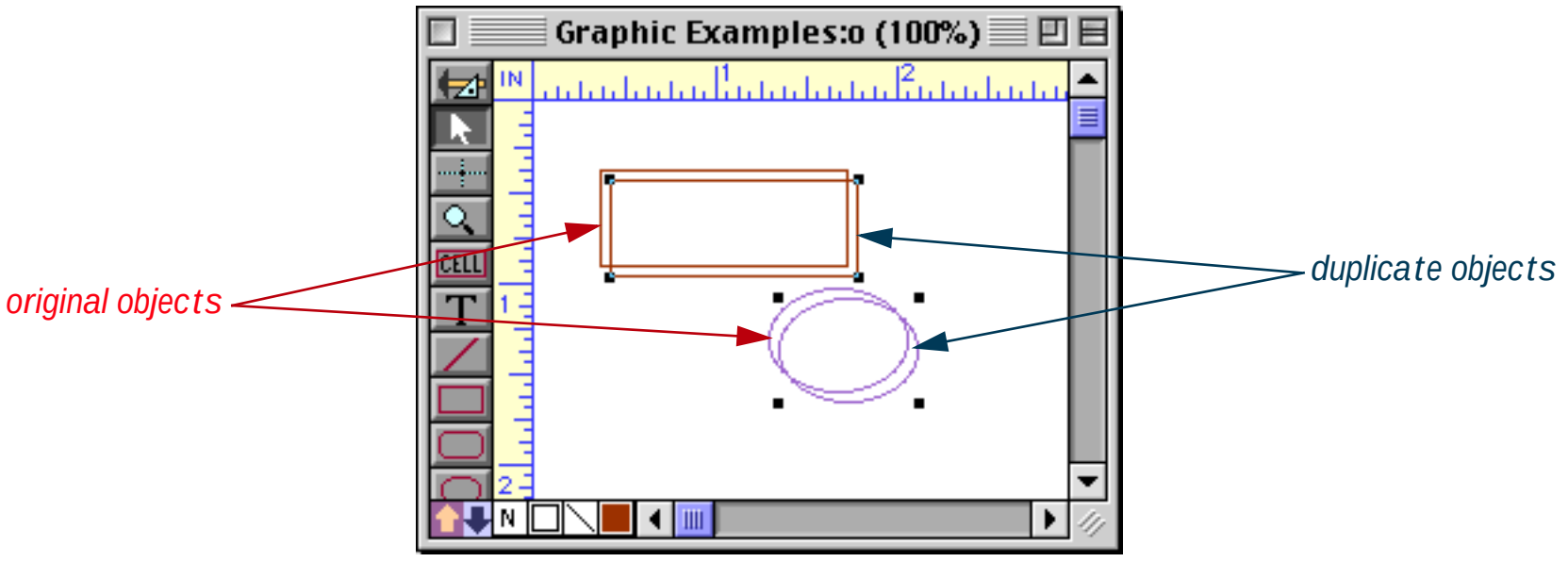
The **Spacing** dialog does not align the objects into neat rows or columns. If the objects need to be aligned use the **Align** dialog (see “Aligning Objects” on page 553).

Duplicating Objects

Why do the same work twice? Panorama has several methods for creating a duplicate of one or more objects.

Duplicate

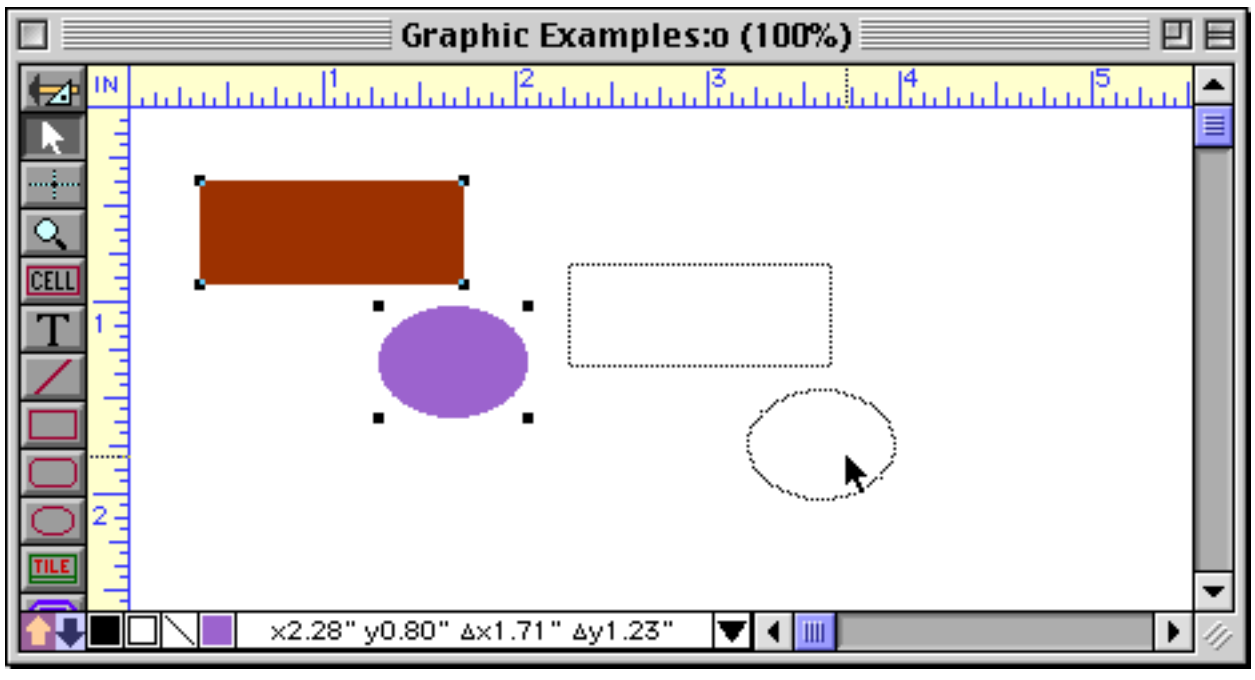
The easiest way to copy objects is with the **Duplicate** command. Just select the objects and choose **Duplicate** from the Edit menu. The new object(s) are placed just below and to the right of the originals.



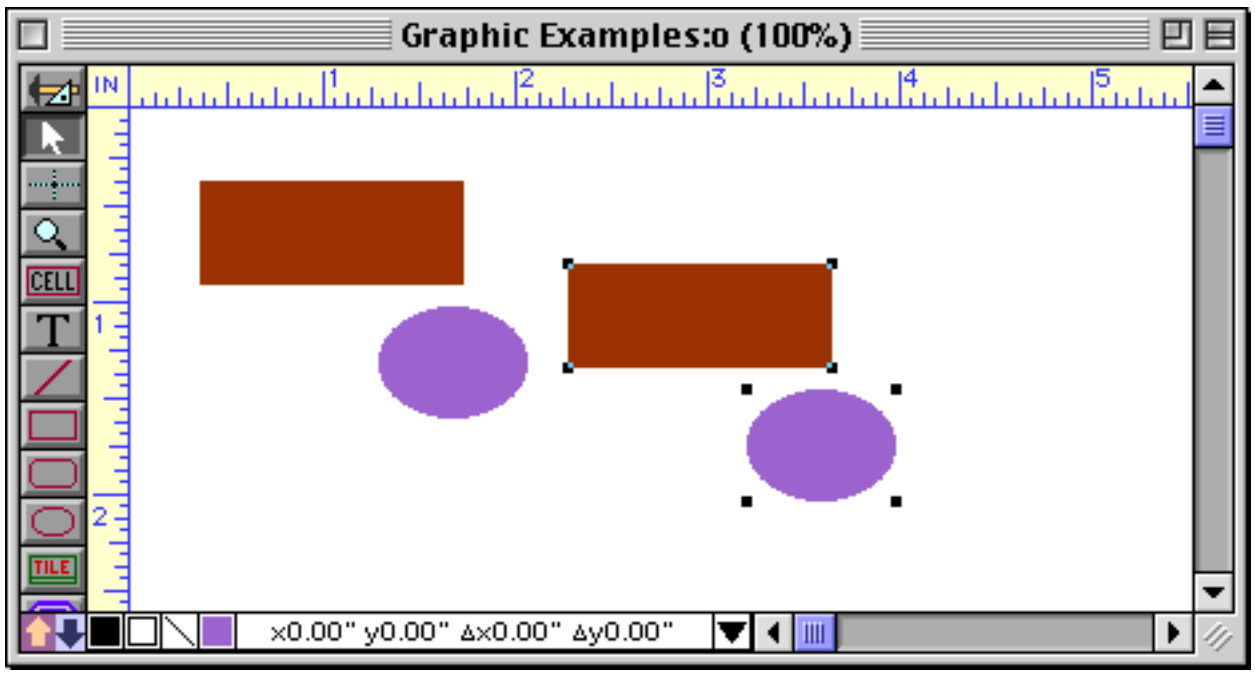
If you duplicate an object (or objects) and then immediately drag (and/or nudge) the copy to a new position, Panorama will memorize this position relative to the original object. Now if you duplicate the copy, Panorama will automatically place the copy of the copy in the same relative position.

Drag Duplicating

You can also duplicate an object by dragging the object with the a special key held down. On the Macintosh the special key is the **Option** key. On PC systems the special key is the **Alt** key. When the special key is held down you drag a copy of the object(s), instead of the original. Just hold down the key and drag the same way you would to move the object(s).



When the mouse is released a second copy of the object(s) appears at the new location.

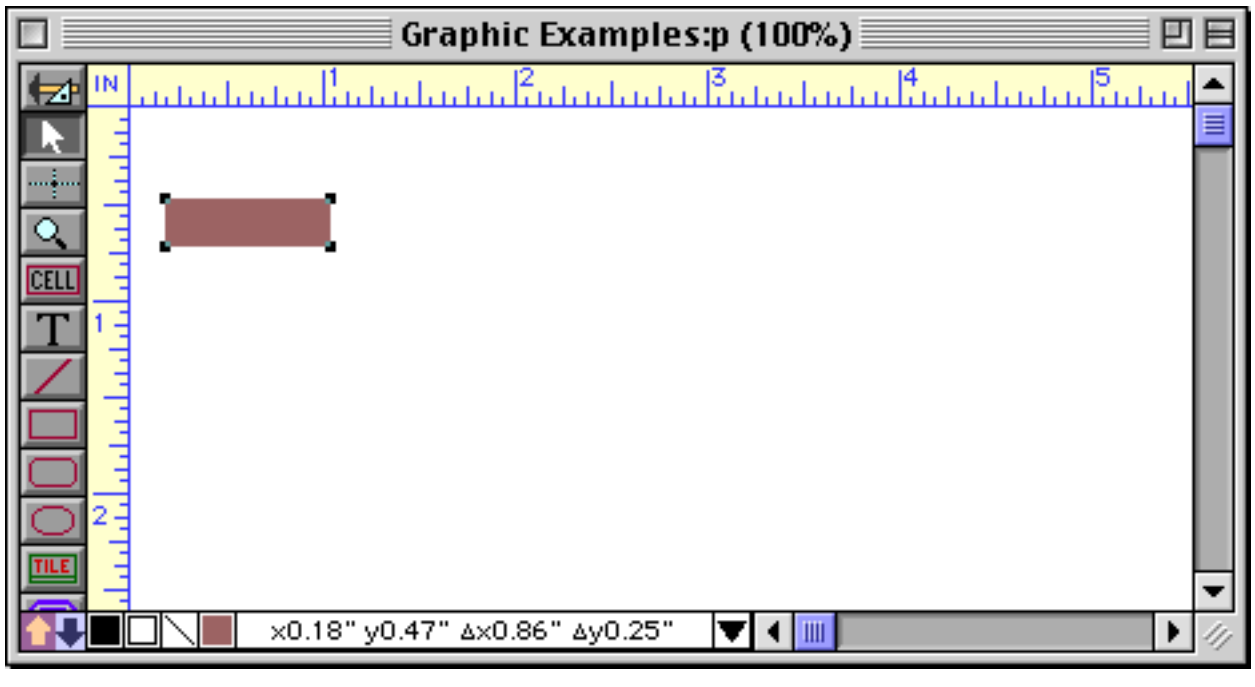


If you want the copy to line up with the original hold down **Shift** key and the **Option**/**Alt** key as you drag the object. The **Option**/**Alt** key tells Panorama to makes copies, while the **Shift** key prevents you from dragging the copy diagonally.

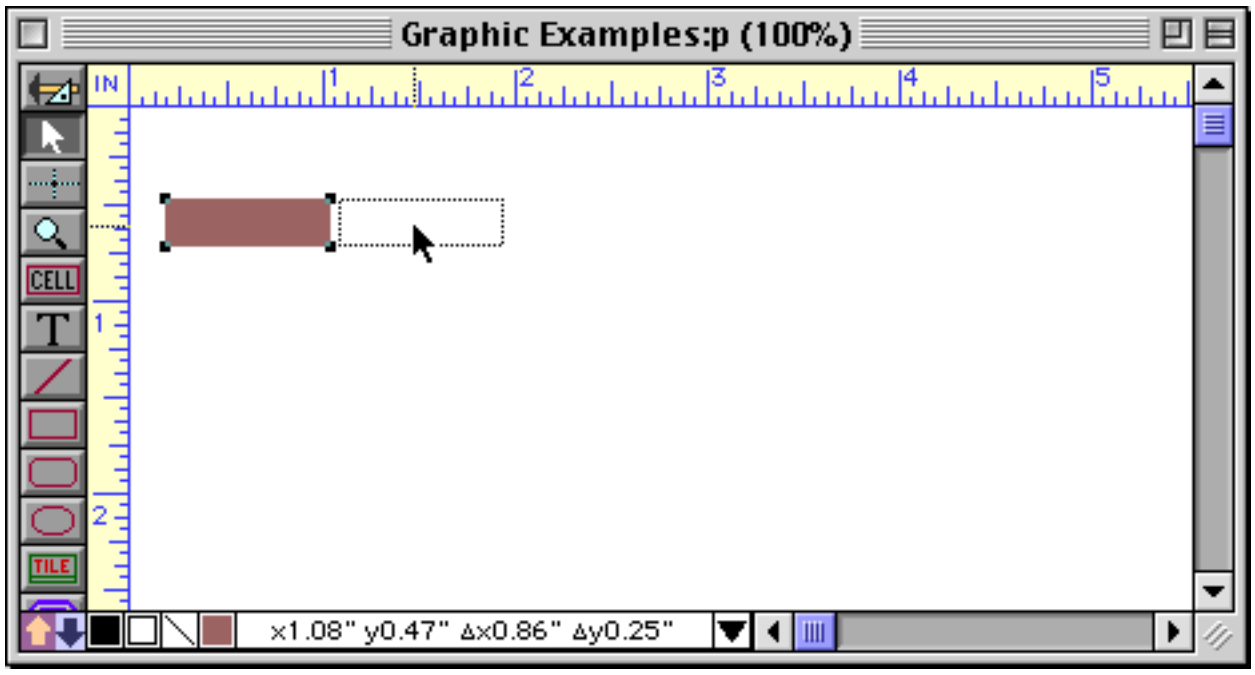
Once you have created a copy by **Option**/**Alt** dragging, you can make another copy with the **Duplicate** command. The **Duplicate** command will exactly mimic your **Option**/**Alt** drag, allowing you to quickly create an accurately spaced row or column of objects (see the next section for an example). **Warning:** If you want the **Duplicate** command to mimic your **Option**/**Alt** drag, you must not do anything in between dragging and pulling down the menu. If you click anywhere else in the form, the **Duplicate** command will not mimic the **Option**/**Alt** drag.

Step and Repeat

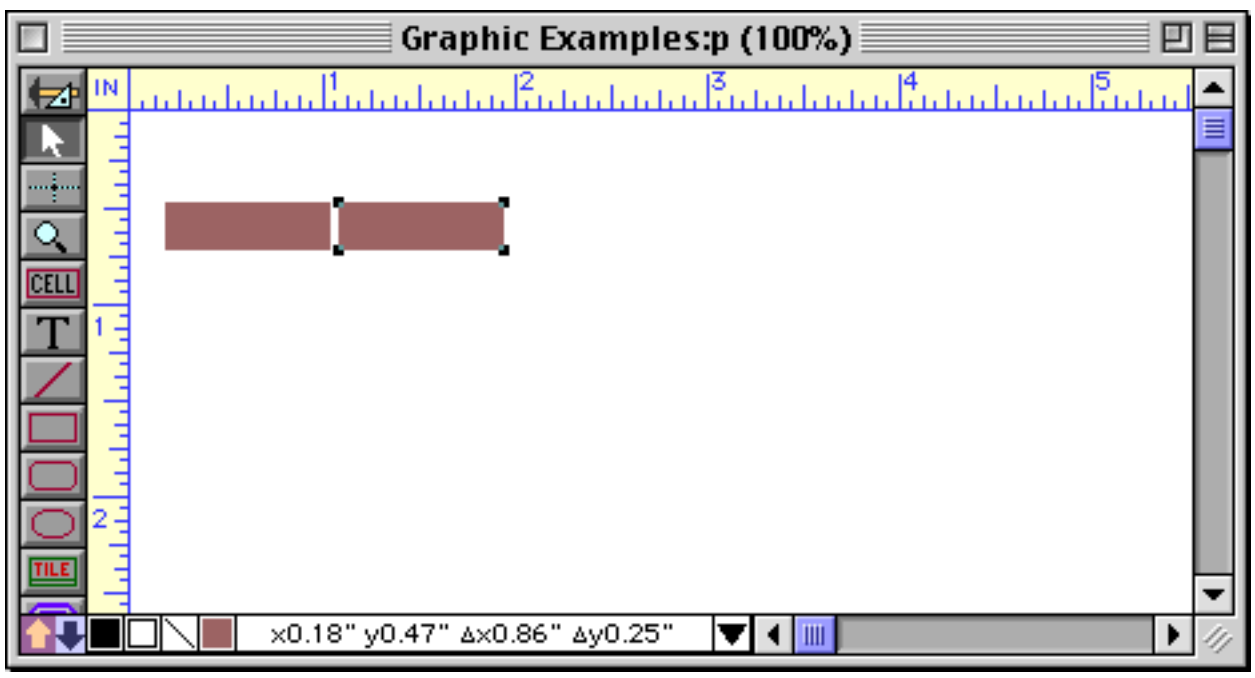
You can combine **Option**/**Alt** dragging (see previous section) with the **Duplicate** command to quickly step-and-repeat evenly spaced rows, columns, and complete tables of objects. To create a row of objects, start with just one object.



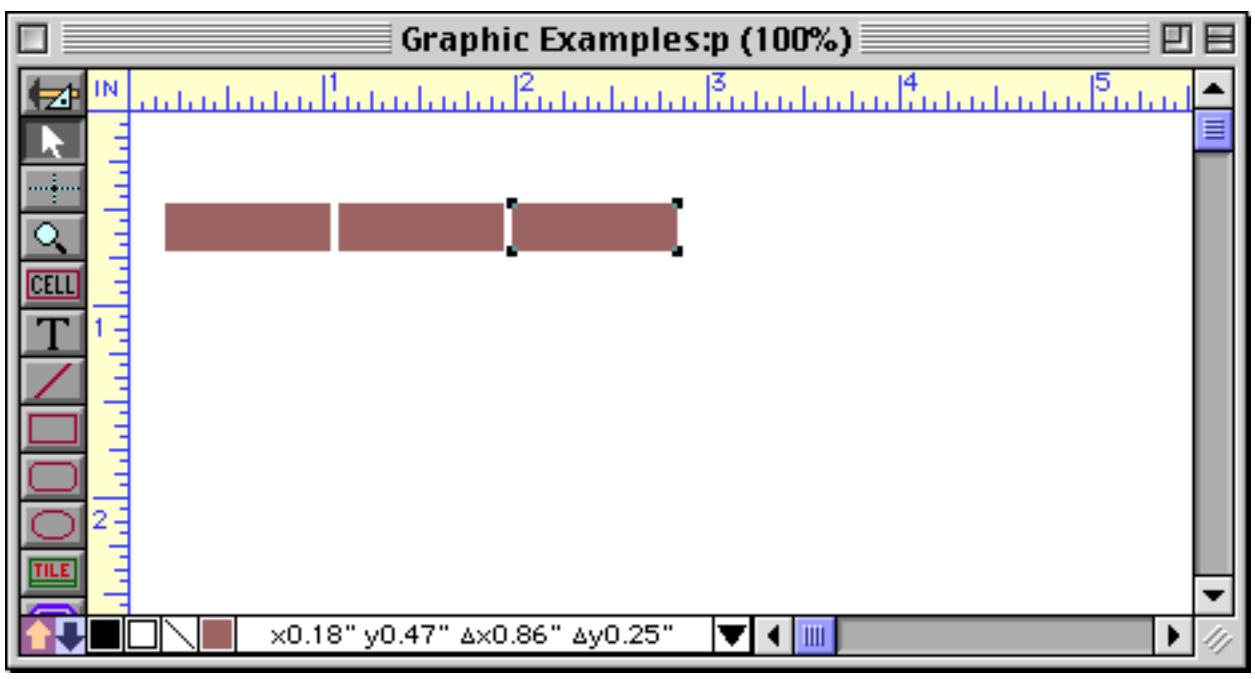
Hold down the **Shift** and **Option**/**Alt** keys and drag the object to the right.



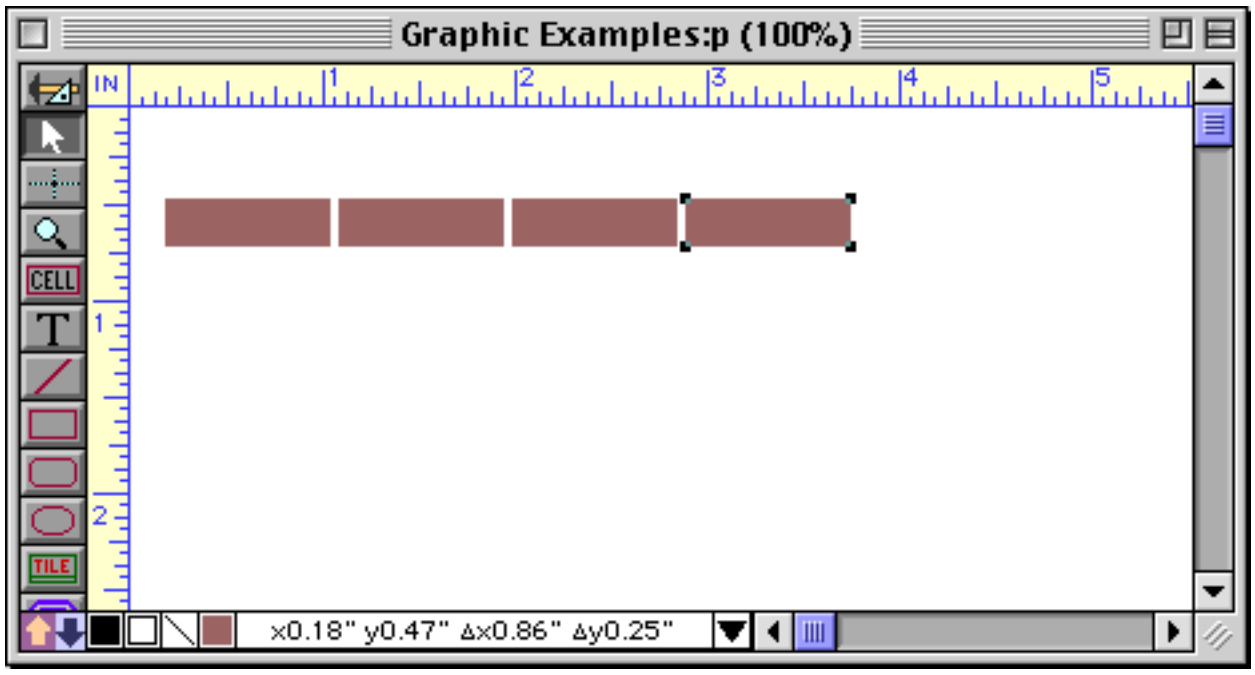
Release the mouse to create a copy.



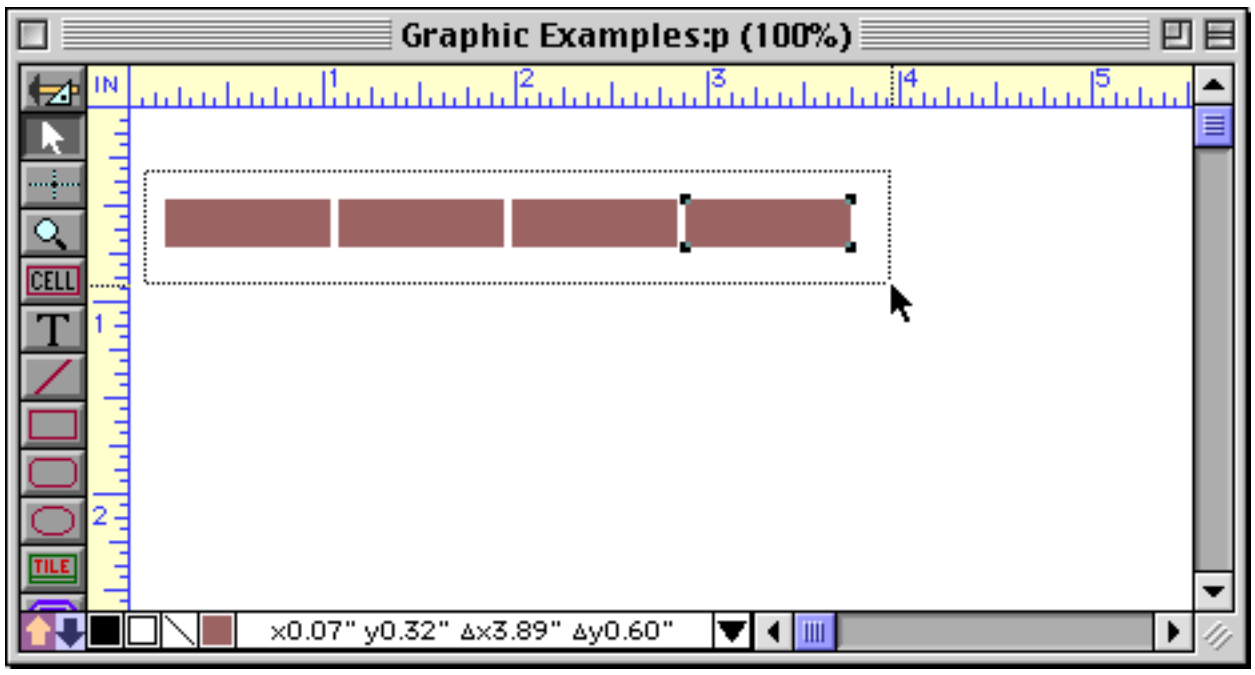
Then use the **Duplicate** command to create another copy of the object. The copy will automatically be placed at the same spacing as the first duplicate.



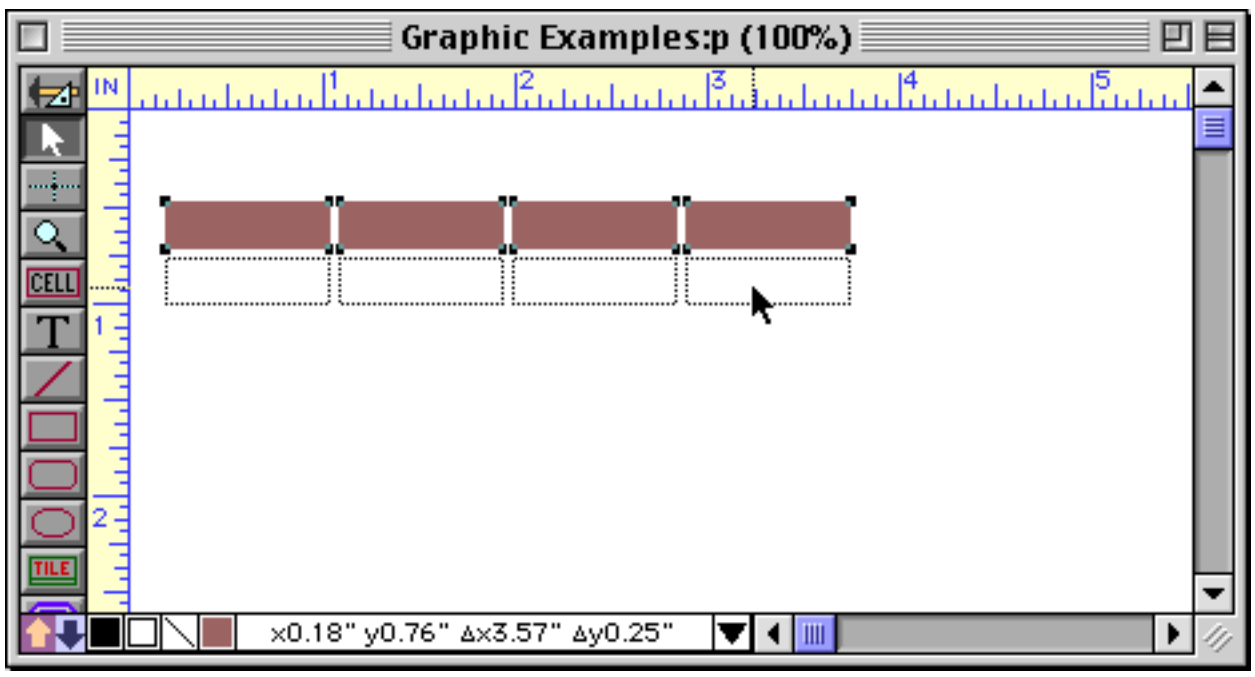
Repeat the **Duplicate** command until the row is complete.



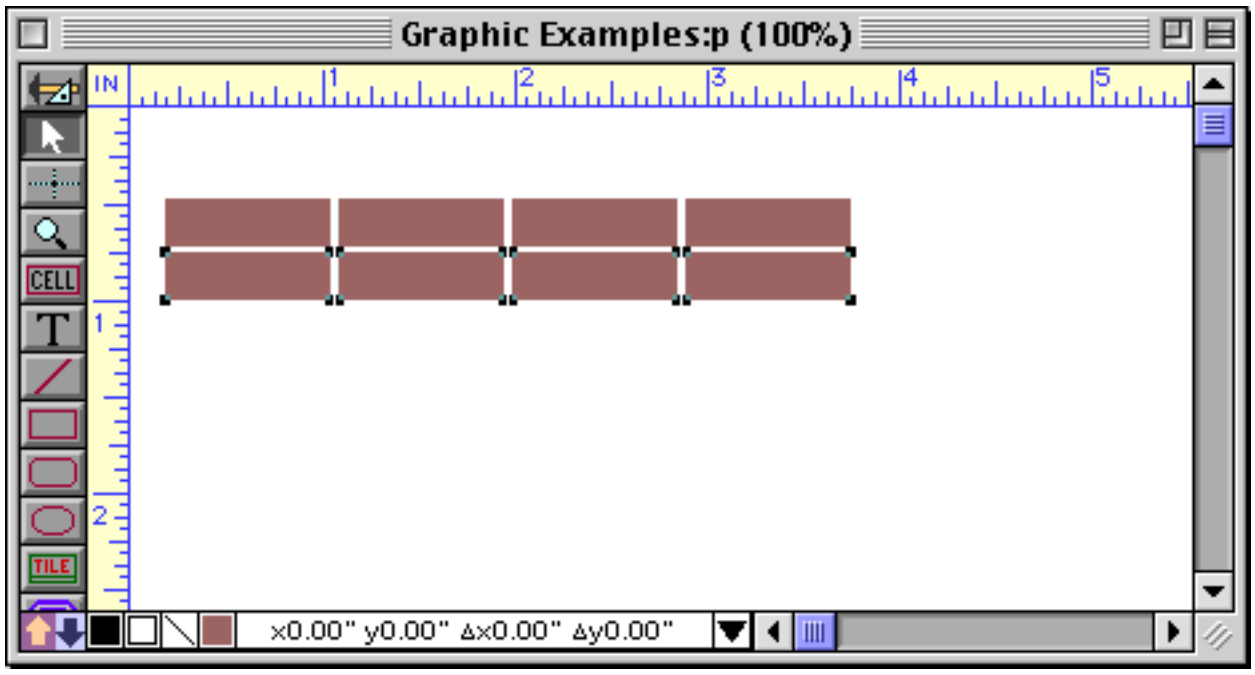
To create a complete table, select all of the objects in the row by dragging a marquee around them.



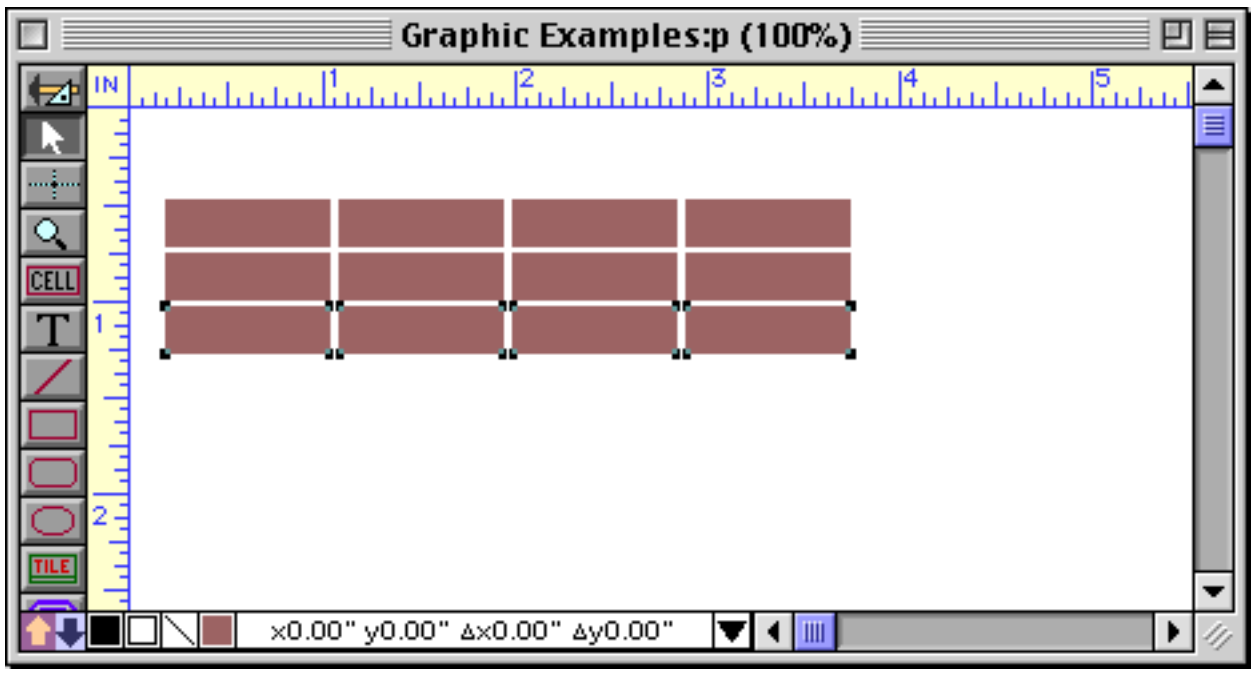
Hold down the **Shift** and **Option** / **Alt** keys and drag the row down, creating a copy of the row.



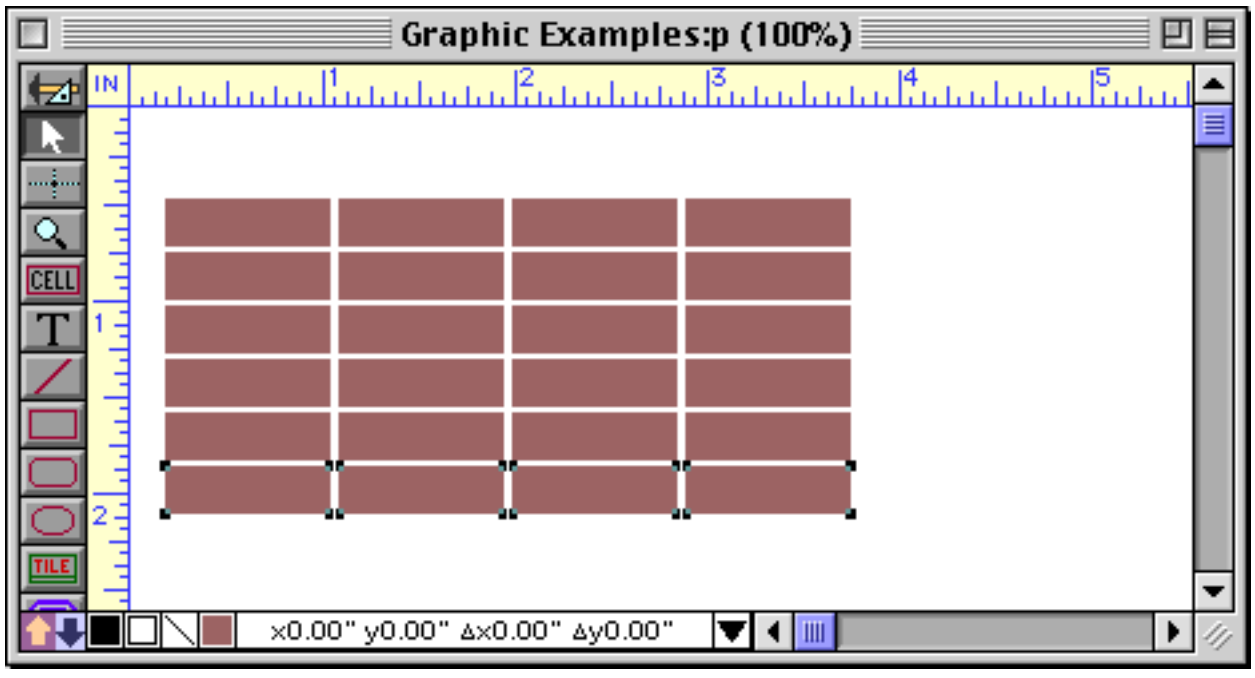
Release the mouse to create the new row.



If you don't get the spacing just right, you can nudge the row with the arrow keys (see "Nudging an Object (or Objects)" on page 509). Once the second row is in position, use the **Duplicate** command to step and repeat an additional copy of the row



Repeat the **Duplicate** command until the table is complete.



Once the table is complete, you can modify the width of columns within the table using cluster resize (see "Cluster Resize" on page 541). Or you can change the spacing between the rows and columns using the **Spacing** command (see "Adjusting Spacing Between Multiple Objects" on page 556).

Cut, Copy, and Paste

You can use the clipboard to copy objects and to transfer objects from one form to another form. The **Cut** and **Copy** commands copy the selected objects into the clipboard. **Cut** also removes the objects from the form.

The **Paste** command places a copy of the objects in the clipboard on to the form. The object (or objects) is placed into the middle of the current window.

You can also use the **Paste** command to paste in graphics created in another program. For more information see "Fixed Images" on page 739.

Copying Objects Between Forms

The **Copy** and **Paste** commands can be used to copy objects from one form to another. Just copy the objects from the original form, then switch to the second form and paste. **Tip:** Both forms must be in graphics mode to copy graphics between the forms.

Copying Objects Between Files

Using the **Copy** and **Paste** commands, you can easily copy graphic objects from one file to another. If you copy a data cell object to another file, the data cell field name may not match up with any of the fields in the second database. If this happens Panorama will automatically substitute the first field name in the second database. You can use the **Data Cell** tool to assign a different field to the cell.

Copying an Entire Form

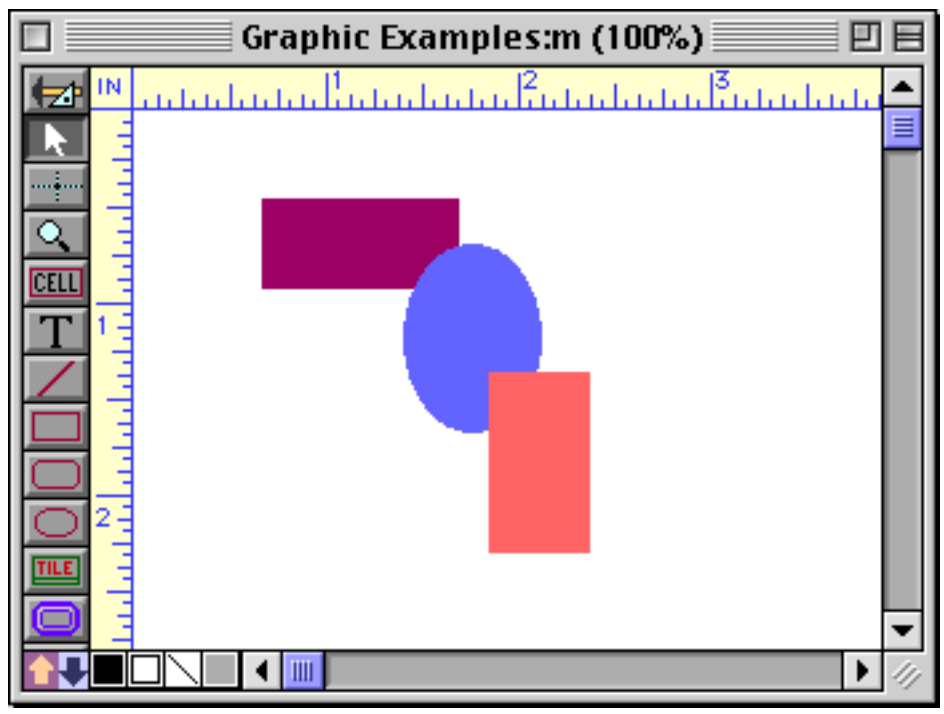
The **Copy Form** and **Paste Form** commands (in the Graphics Mode's Edit menu) allow an entire form to be copied at once -- including the form's Page Setup, custom menu setup, form comments -- everything!

To duplicate a form, start with the original form in Graphics Mode. Then choose the **Copy Form** command from the Edit menu. Then choose the **Paste Form** command. (If you want to duplicate the form in another database you must go that database, open a form and switch that form to Graphics Mode before using the **Paste Form** command.) The **Paste Form** command will ask you for the name of the new form. When you press the **Paste Form** button, Panorama will create the new form as an exact duplicate of the old form. (Notice that you do not create the new form in advance — the **Paste Form** command creates the new form for you.)

You can also use the **Copy Form** command with the regular **Paste** command to copy all the objects in a form into another form. When used this way, the **Copy Form** command is the same as choosing **Select All Objects** followed by **Copy** command. When used this way, only the objects are copied and not the Page Setup, custom menus, etc.

Overlapping Objects

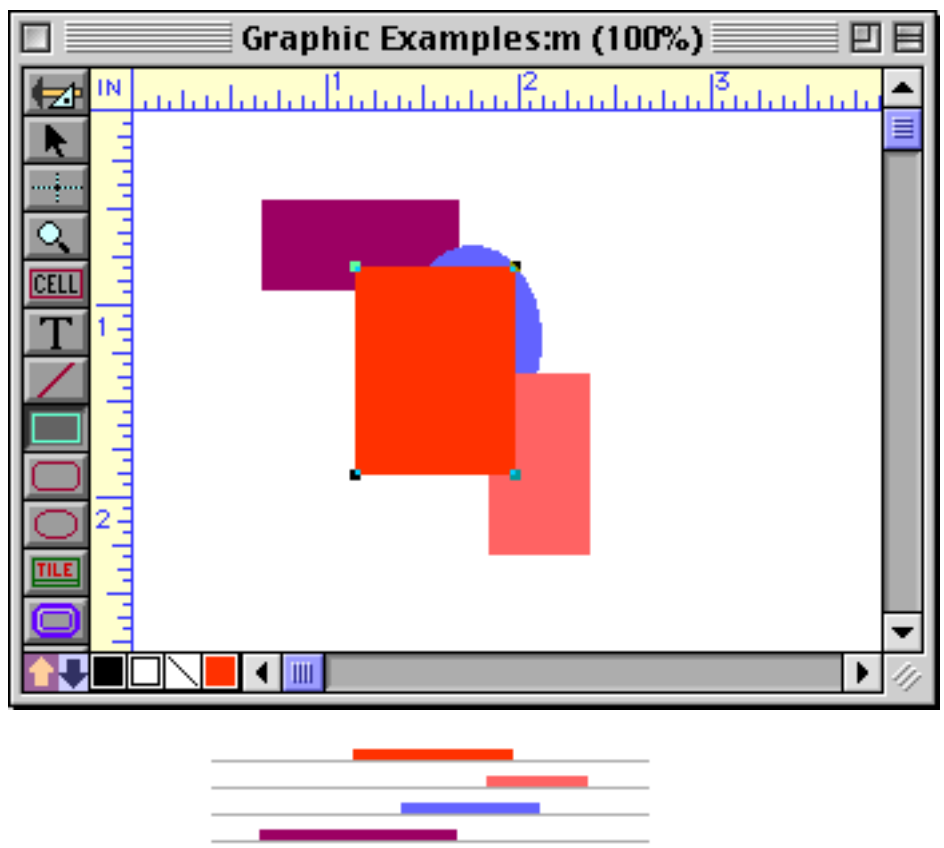
The computer screen is two dimensional, in other words, flat. So what happens if two objects overlap each other? To resolve this question, Panorama treats the objects as if they were placed on a stack of clear sheets. For example, consider the three overlapping objects in this form —



If you could look at this form on edge, it would look like this —



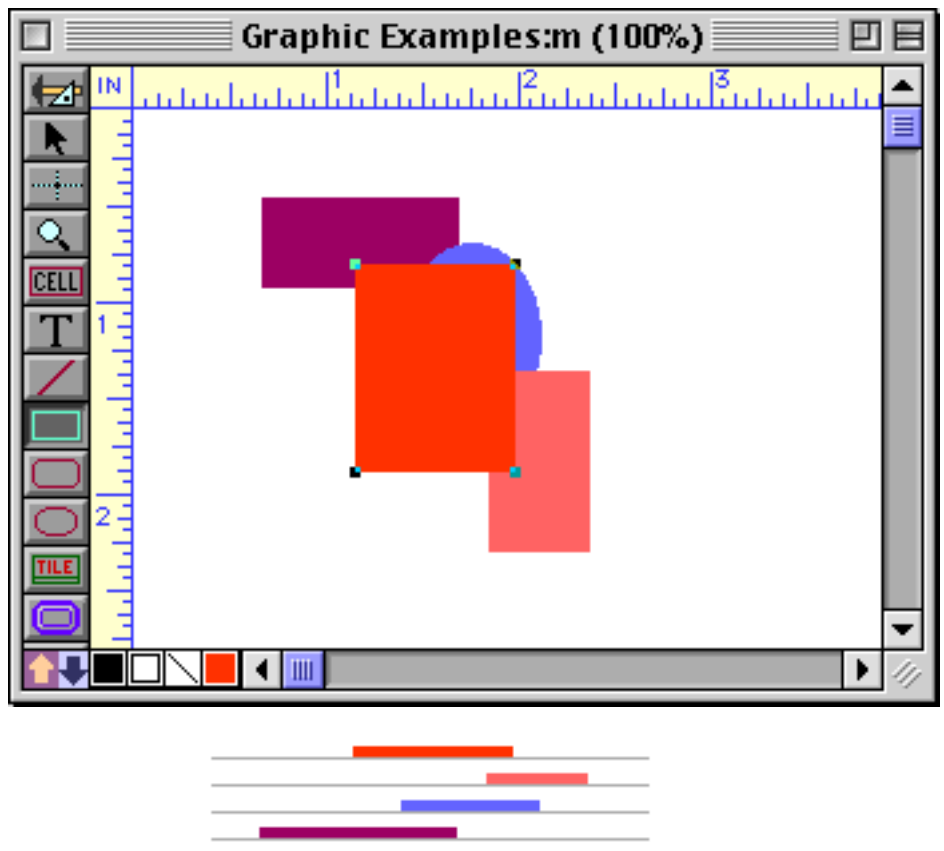
An object that is on top of another object is said to be **in front**. An object that is below another object is said to be **in back**. When you create a new object, the new object is placed in front of all other objects.



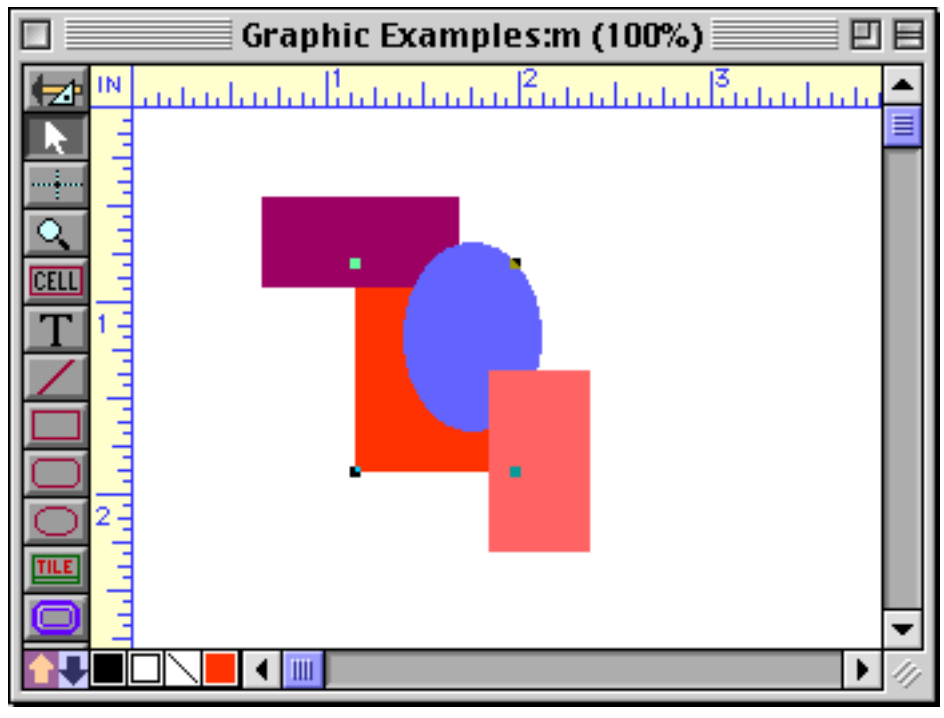
Objects that were created earlier will be partially or completely hidden if the new object overlaps them.

Changing the Stacking Order

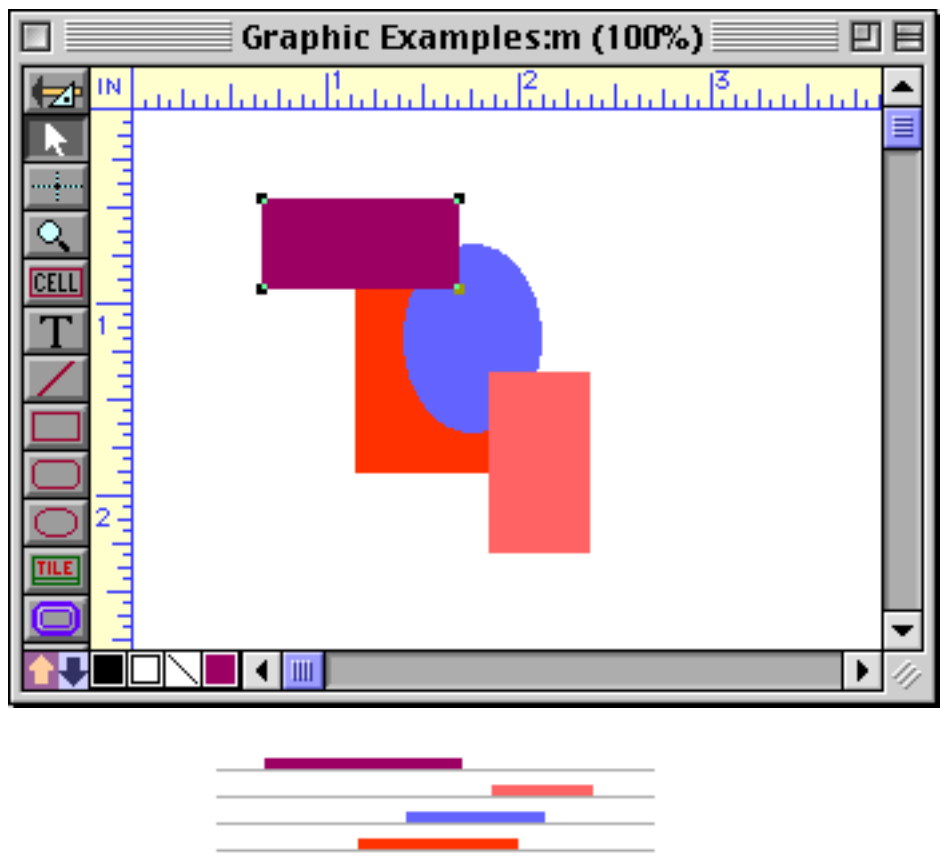
The **Bring to Front** and **Send to Back** commands (in the Arrange menu) change the stacking order of overlapping objects. To put an object behind everything else, select the object and then choose **Send to Back**. For example, suppose you wanted to move the red box behind the other three objects on this form. Here's the initial form, including the edge view.



The **Send to Back** command will move the red box behind the other objects.



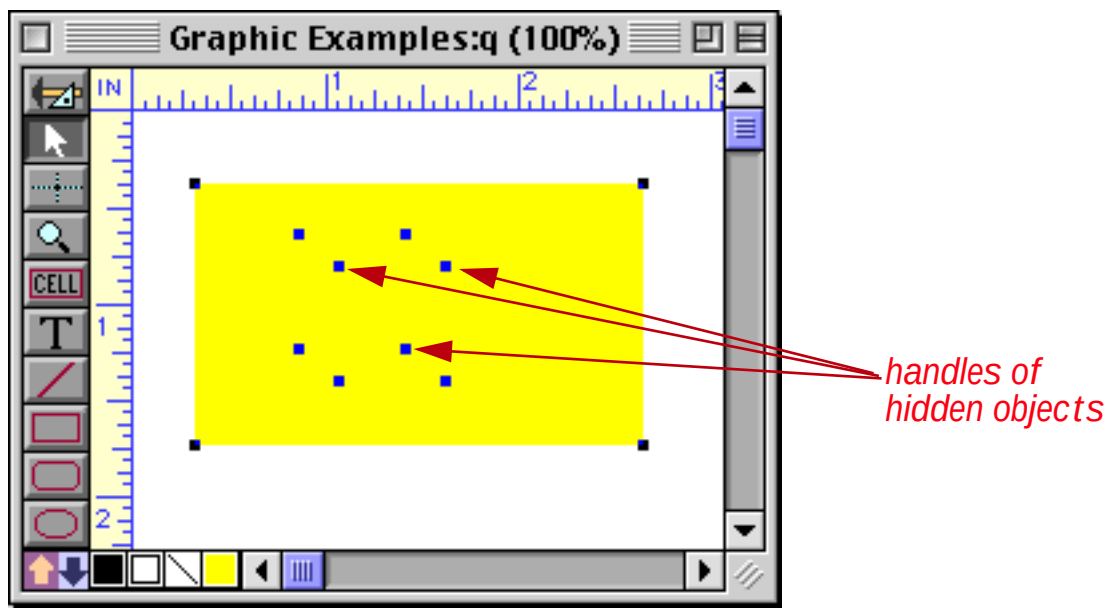
To bring an object to the front, select the object and then choose **Bring to Front**. For example, you could bring the purple box to the front (again showing the edge view).



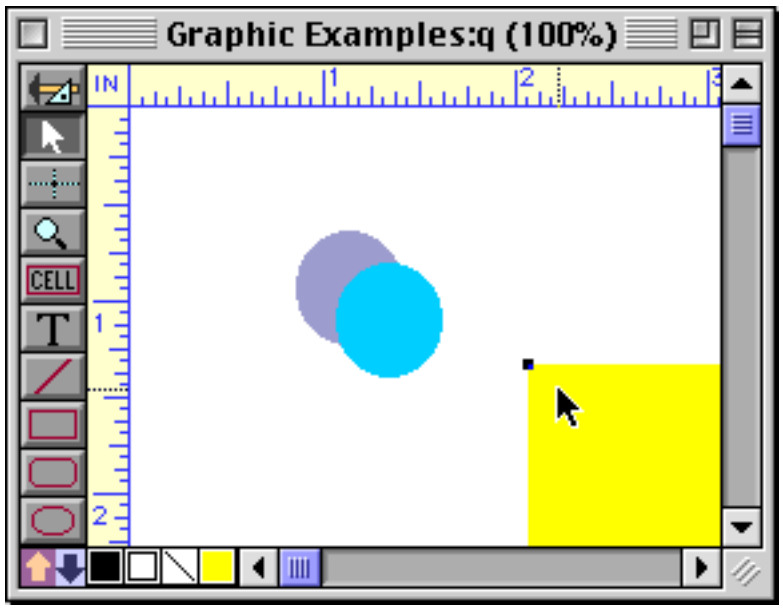
Selecting a Completely Hidden Object

If an object is completely hidden you can't click on it. However, there are several ways to select hidden objects.

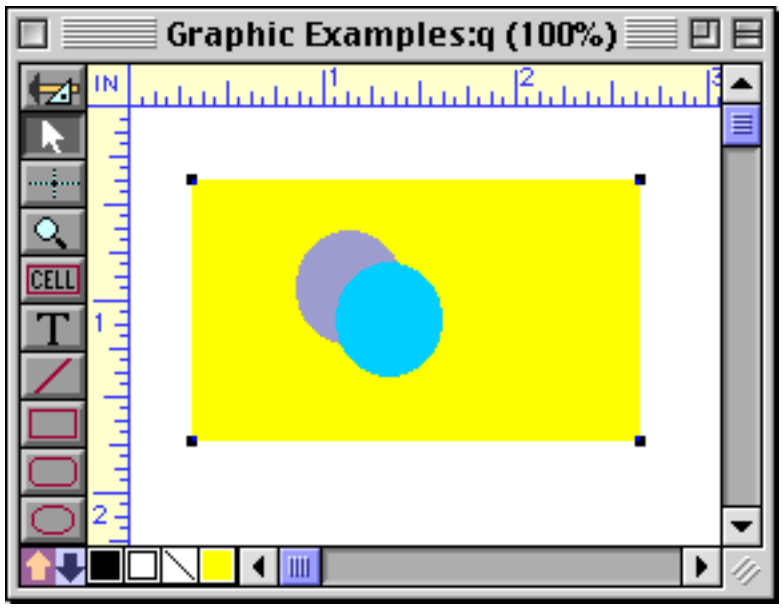
To find a hidden object, use the **Select All Objects** command (in the Edit menu). This command makes handles appear for every object, including hidden objects. For example, at first glance the form shown below would appear to have only one object — a yellow box. The **Select All Objects** command reveals that there are two hidden objects behind the yellow box.



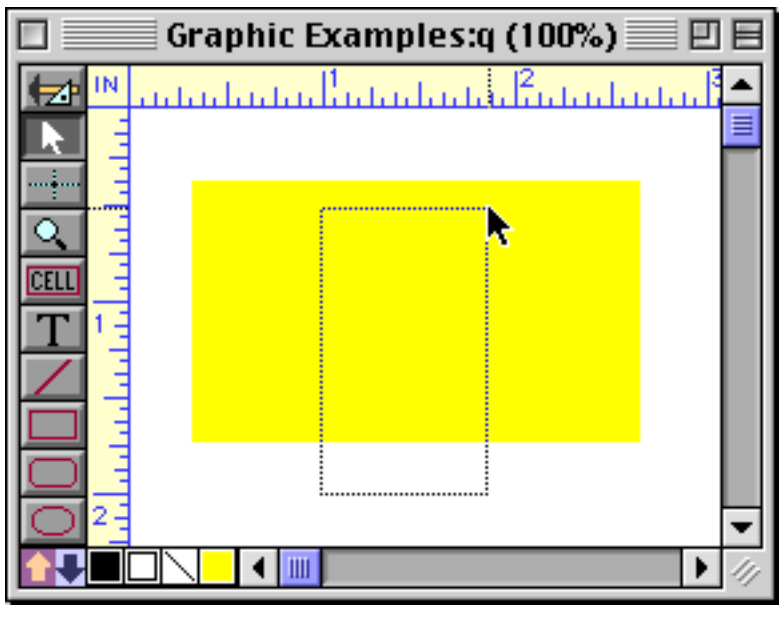
Of course, you can also find a hidden object by moving the objects in front of it out of the way —



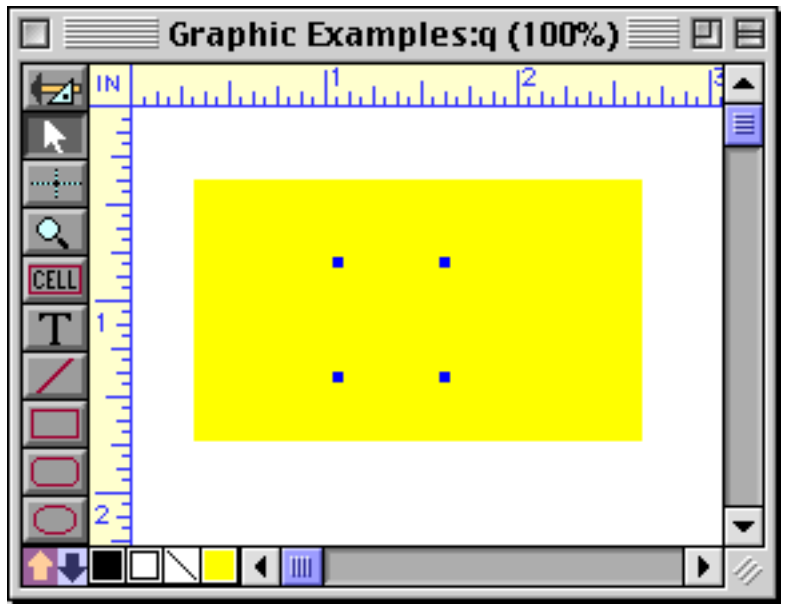
or by sending the object in front to the back.



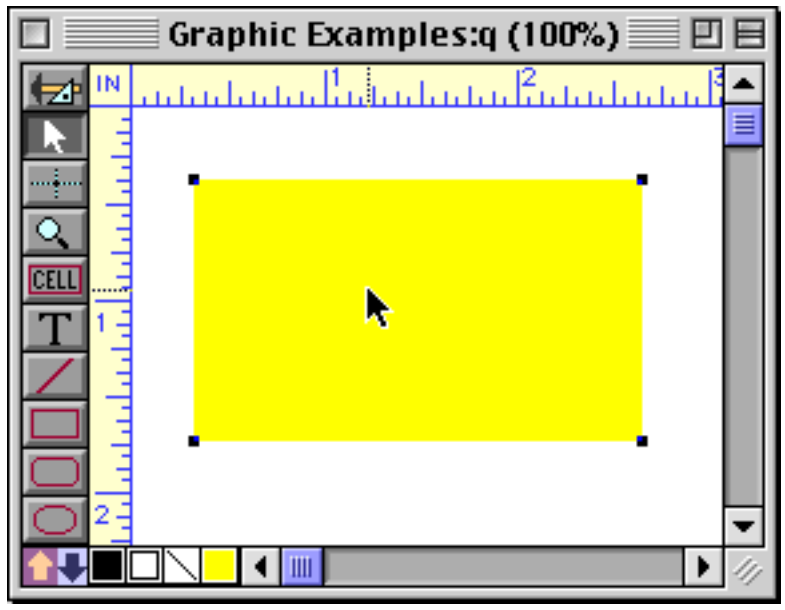
Sometimes you may be able to select a hidden object by dragging a marquee around the object.



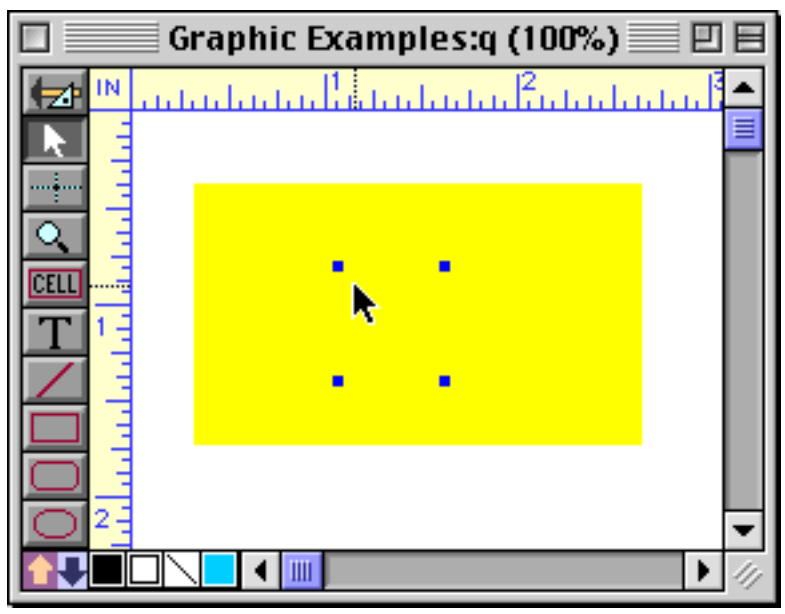
Once the object is selected you can bring it to the front, change the object properties, or nudge the object with the arrow keys.



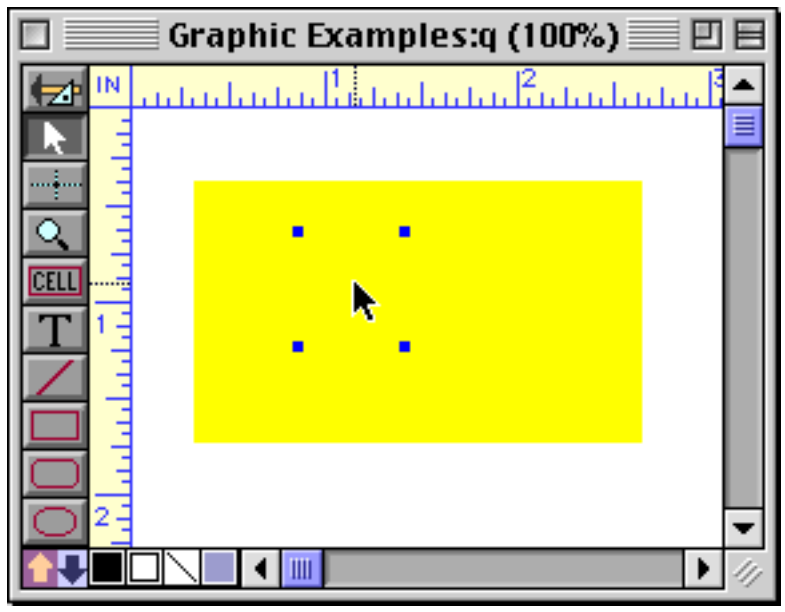
Another technique for selecting a hidden object is to hold down a special key while you click. On the Macintosh this special key is the **Command** key, on PC systems it is the **Control** key. The first time you click, the topmost object will be selected.



The next click will select the next object behind the top object (remember, you must hold down the **Command** / **Control** key).



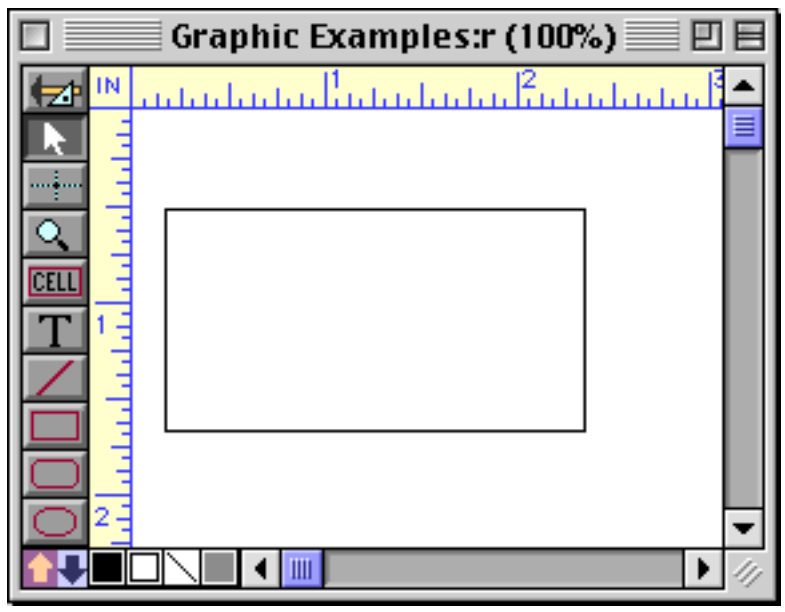
Each time you **Command**/**Control** click again the next object behind the current object will be selected



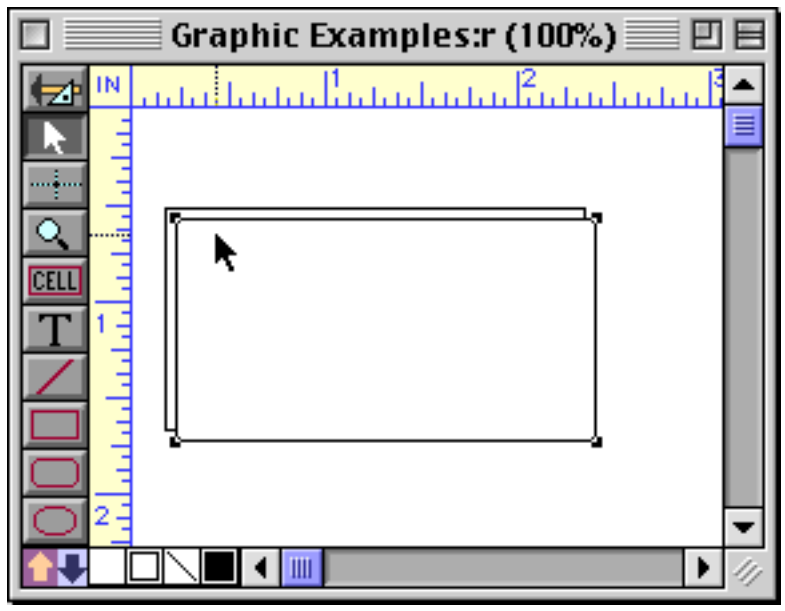
When you reach the bottom of the file Panorama will cycle back to the top and select the topmost object again. You can keep clicking around and around forever.

Making a Drop Shadow

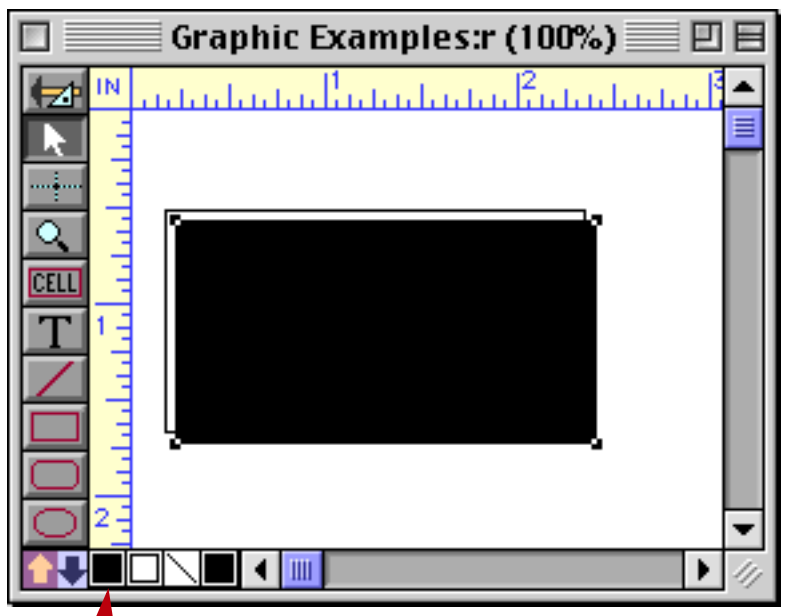
Use **Duplicate** and **Send to Back** to create a drop shadow for a box. Start with a basic box.



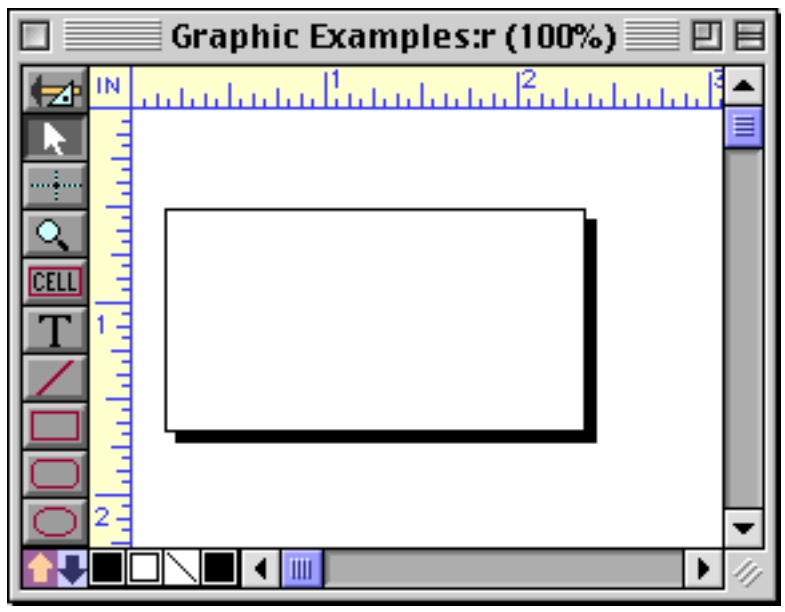
Use the **Duplicate** command to make a copy of the box (see "Duplicate" on page 561). (An alternate technique is to hold down the **Option**/**Alt** key and drag to create a copy of the box, see "Drag Duplicating" on page 561.)



Next, use the Fill menu to make the new box black (see "Fill Pattern" on page 521).



Finally, use the **Send to Back** command to move the shadow behind the original object (see “Changing the Stacking Order” on page 569).

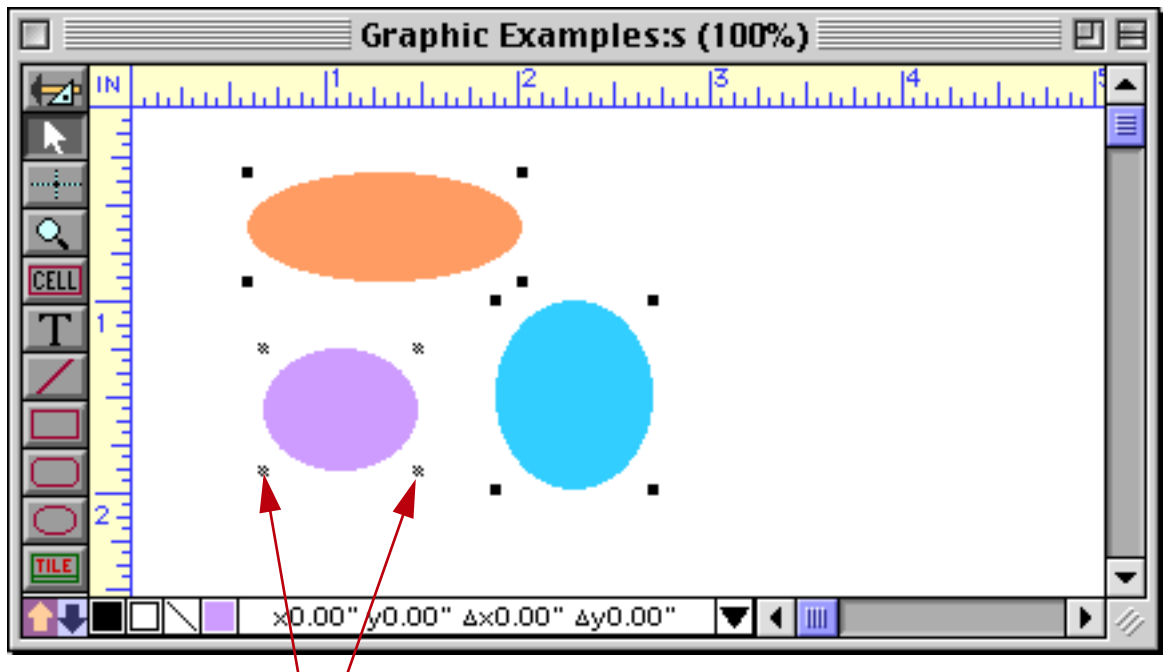


If the shadow is too large or too small, use the arrow keys to nudge it until it looks right (see “Nudging an Object (or Objects)” on page 509).

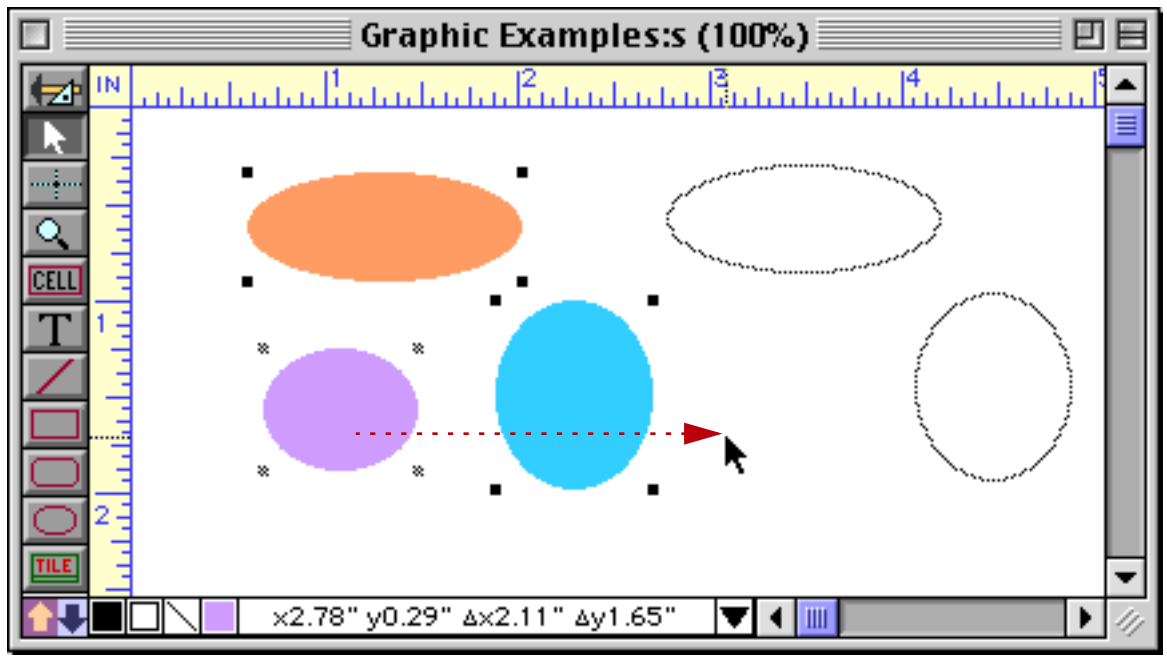
Locked Objects

Use the **Lock** command to lock objects. Locked objects can’t be moved or resized. Locking is convenient when you want to make sure the work you have already finished isn’t disturbed as you continue working.

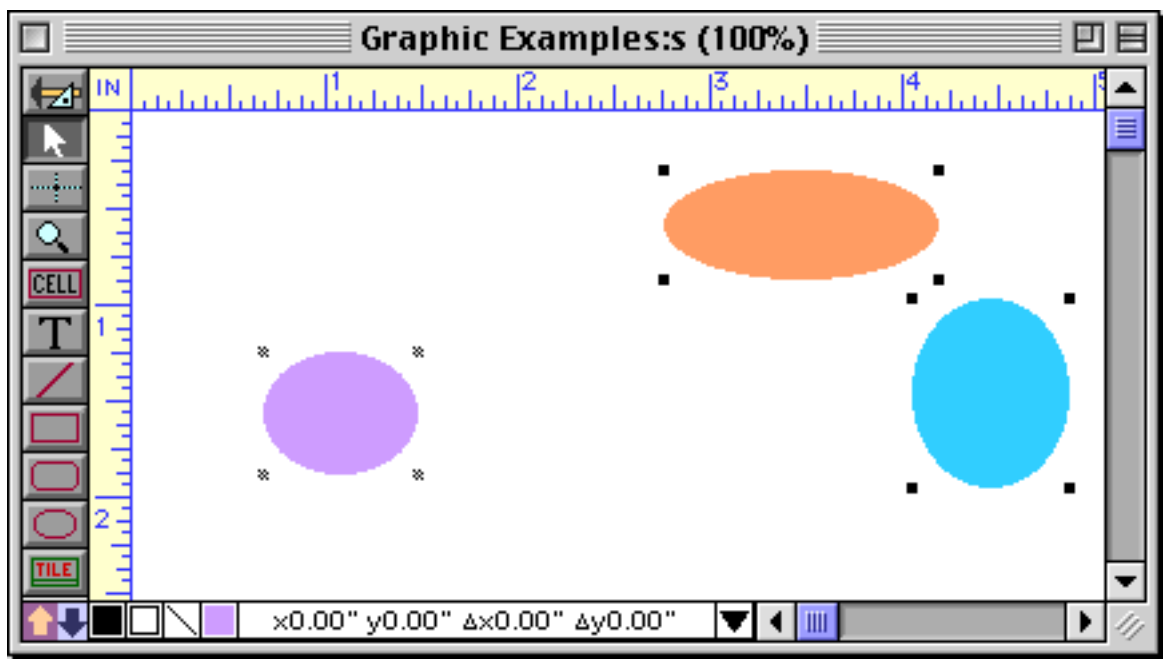
To lock one or more selected objects, choose **Lock** from the Arrange menu. The object handles will turn gray. This shows that the objects are locked. The illustration below shows three selected objects, one of which is locked.



If you attempt to drag these three objects, only the unlocked objects will actually move.



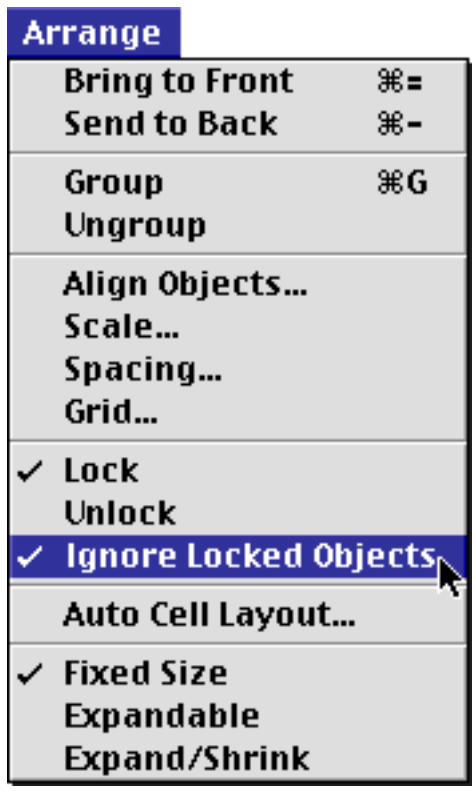
A locked object cannot be moved, resized, or have any of its attributes (color, fill pattern, etc.) changed. However, you can duplicate the object (the copy is not locked).



Use the **Unlock** command to release a locked object(s). Once they are unlocked the objects can be moved and resized normally.

Ignoring Locked Objects

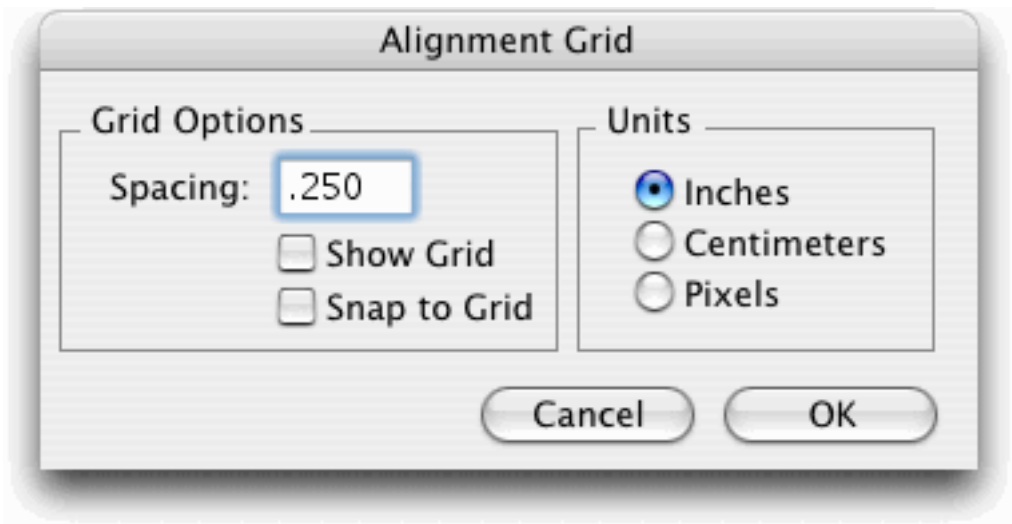
Although locked objects cannot be moved, they can still be selected by clicking on them. (Otherwise they could never be unlocked again!) However, if you check the **Ignore Locked Objects** option in the Arrange menu, you will not be able to select locked objects.



Tip: The **Ignore Locked Objects** option is useful for working on top of a large background covering the entire form. For example, you might paste a scanned image into the form, and then create data cells and other objects on top of the scanned background. By locking the scanned image and turning on the **Ignore Locked Objects** option, you won't have to worry about accidentally selecting and possibly moving the scanned background image.

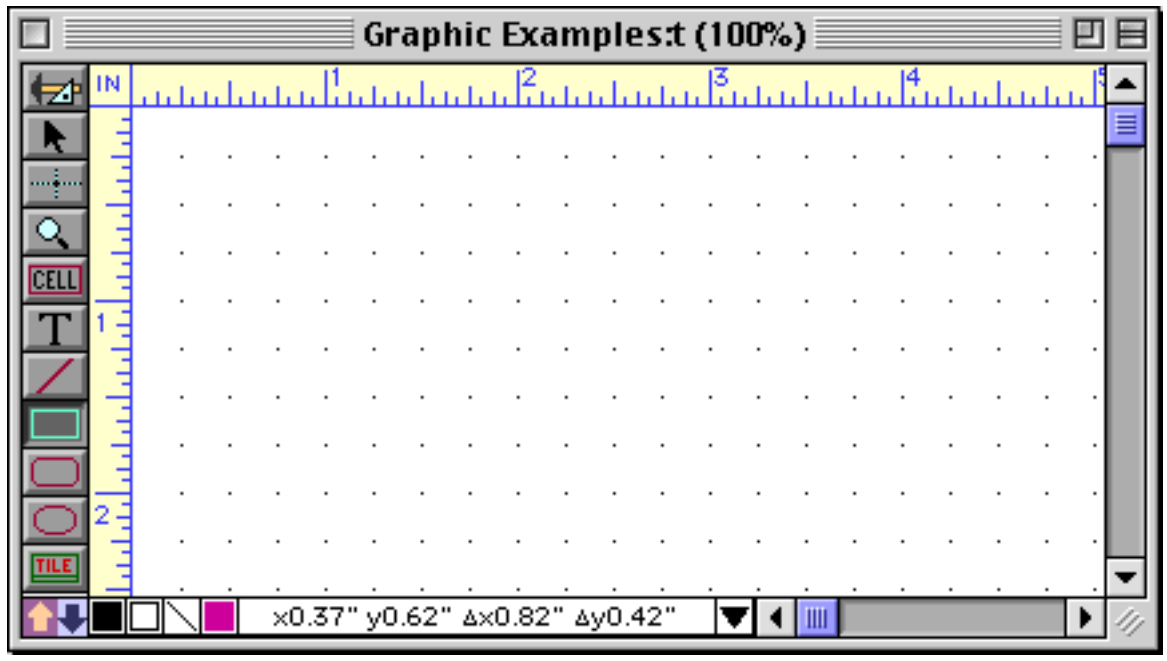
Alignment Grid

Panorama normally allows you to position objects freely anywhere on a form. The **Grid** dialog (in the Arrange menu) allows you to set up a grid that can help you arrange objects in neat rows and columns.

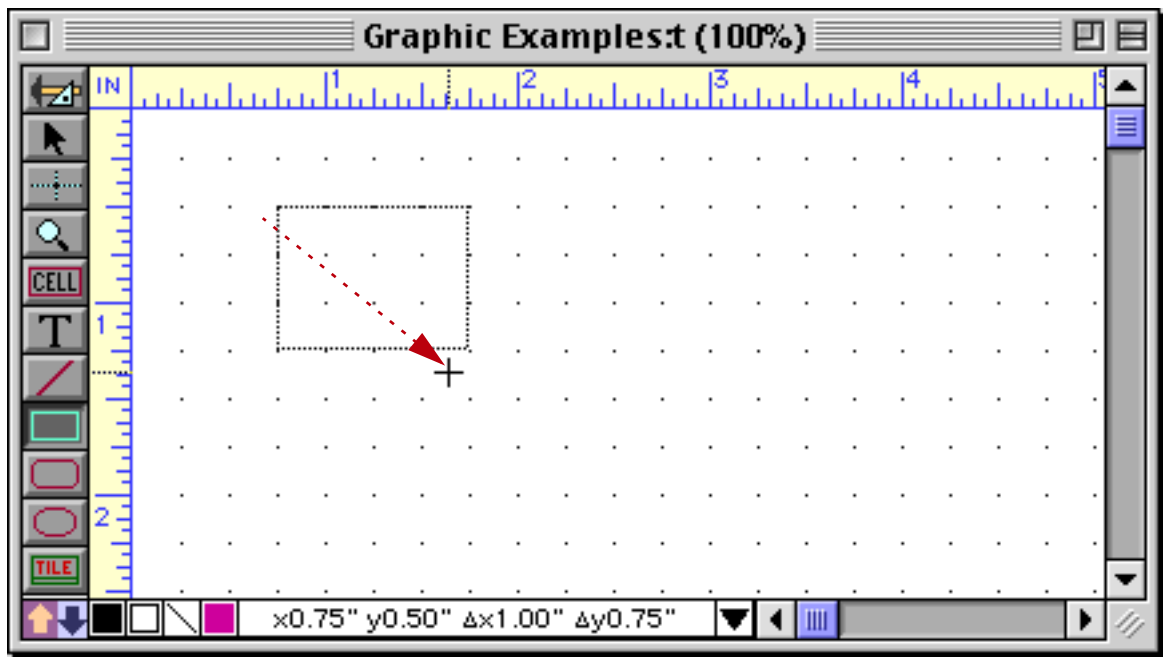


The grid spacing can be specified in inches, centimeters, or pixels. If you use inches or centimeters, the spacing will be rounded to the nearest 1⁄576 inch (For example a grid spacing of 0.1 inch will be rounded to 0.098 inches, or 7.25 pixels.)

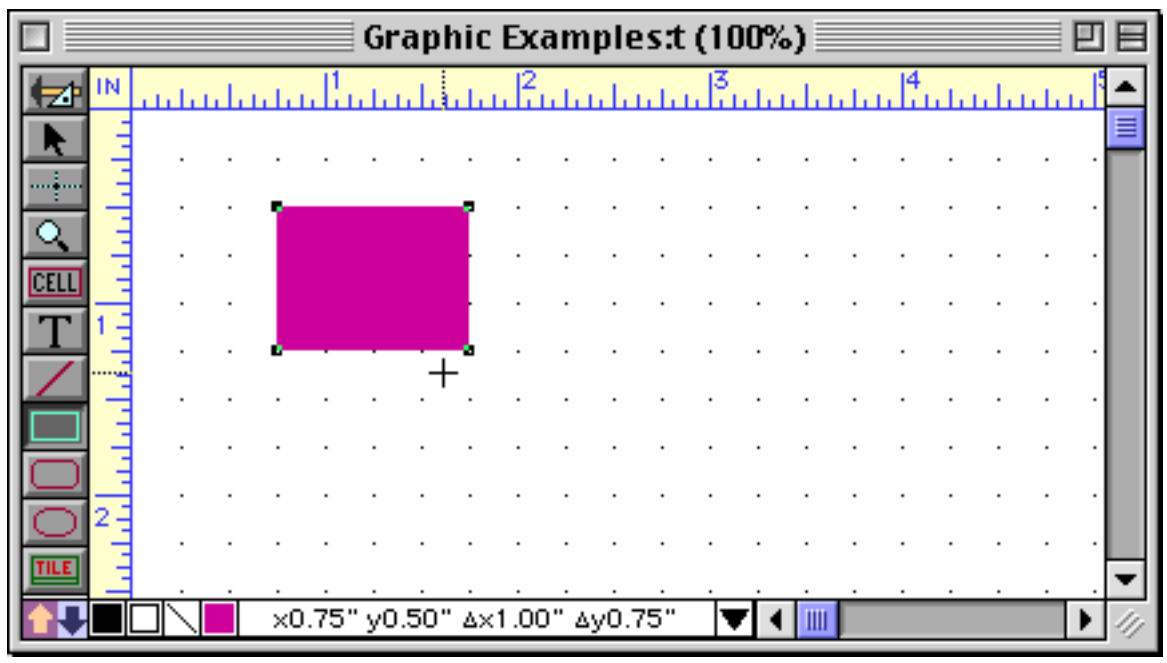
If you want to make the grid visible, check the Show Grid option. When this option is checked a dot appears at each grid point.



When the Snap to Grid option is checked, Panorama will automatically align objects to the grid. Objects are aligned to the grid whenever they are created, moved, or resized. This option does not affect objects already in the form (unless you modify them).



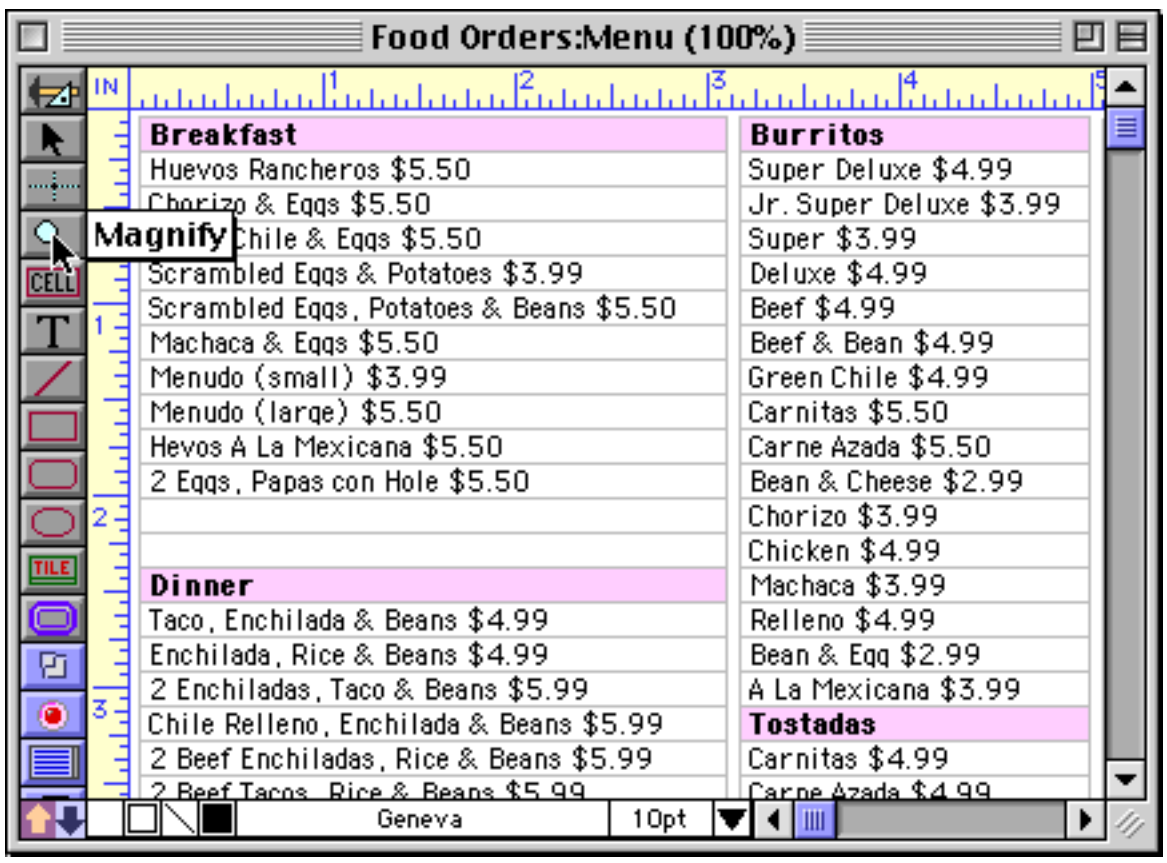
Panorama snaps the object into place as it is created.



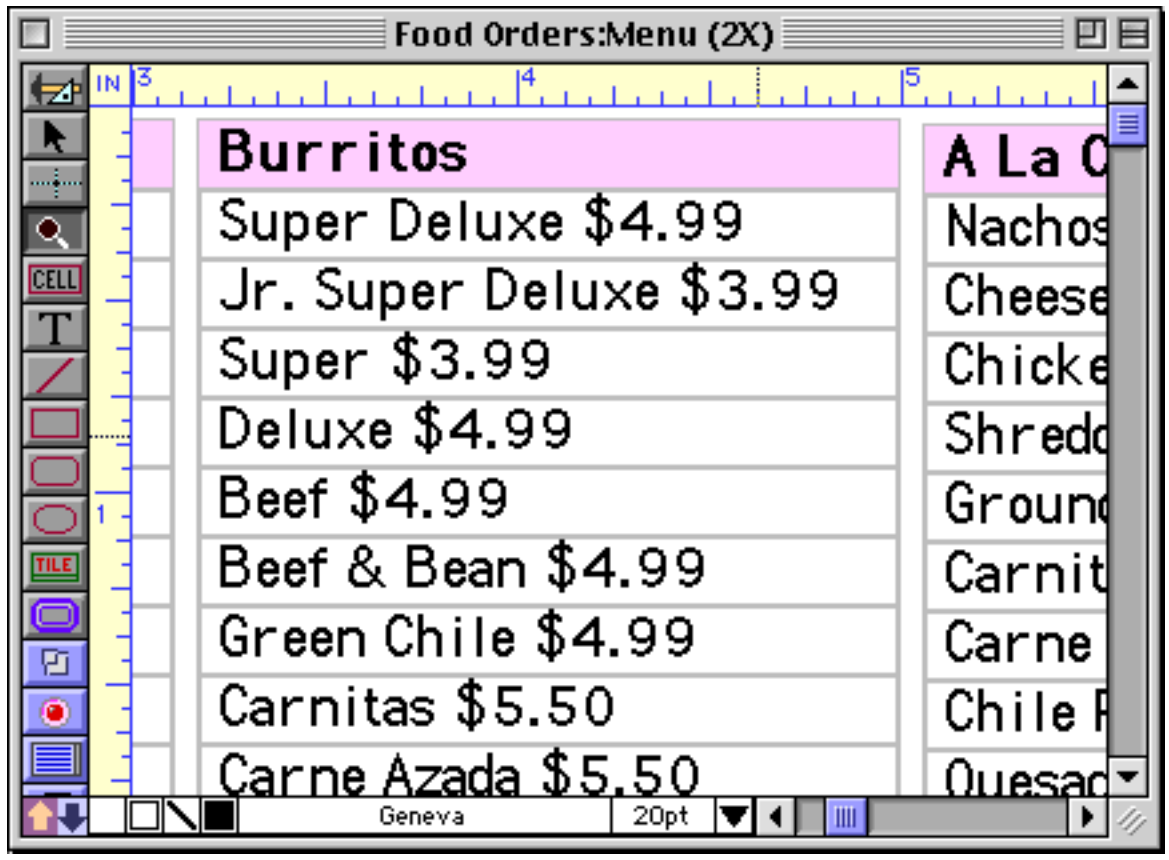
Tip: When **Snap to Grid** is on, every object you create is automatically aligned to the grid. But what if you need one or two objects that are not aligned to the grid? Instead of turning **Snap to Grid** off, you can nudge the object into place using the arrow keys. Both the position and size of the object can be adjusted with the arrow keys. See “Nudging an Object (or Objects)” on page 509.

Magnification and Reduction

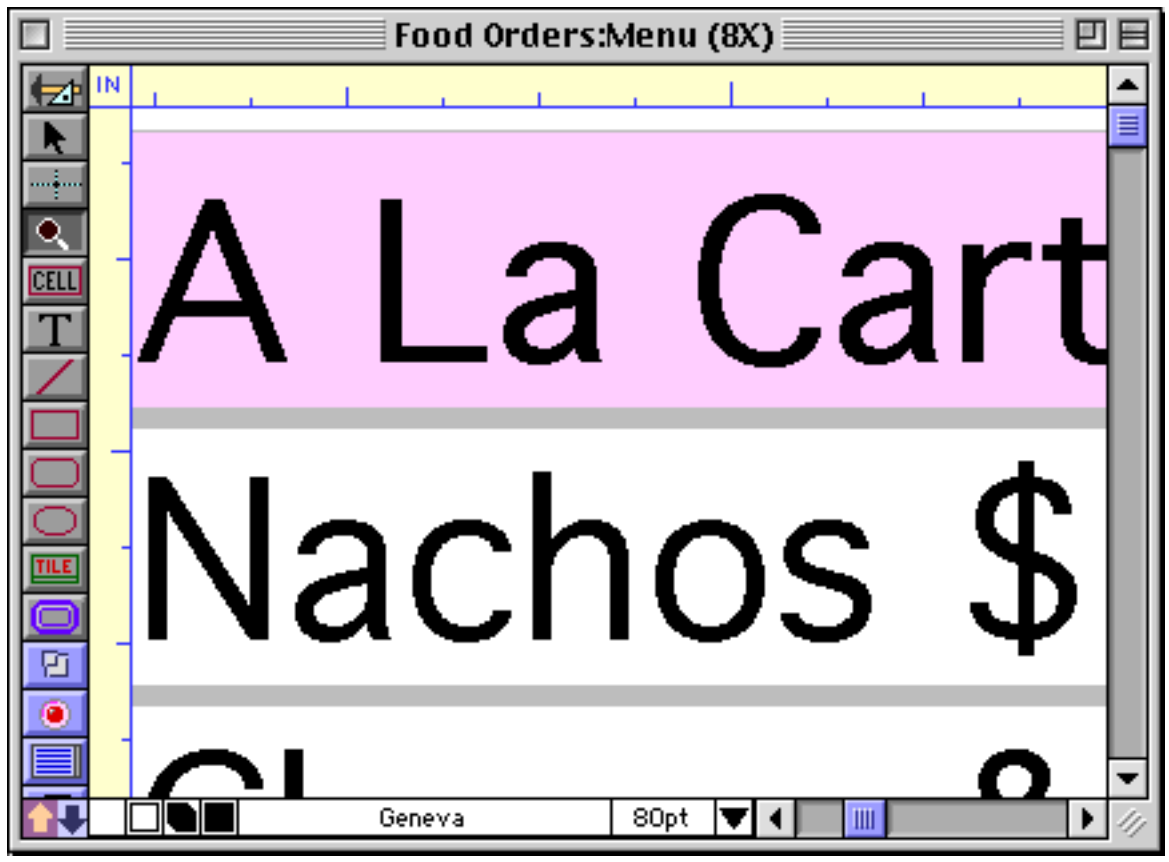
Panorama allows you to magnify the form for closer inspection, or for more accurate alignment of graphic objects. Use the **Magnify** tool to zoom in to 2x, 4x, or 8x magnification. When you select the **Magnify** tool, the cursor turns from an arrow into a magnifying glass.



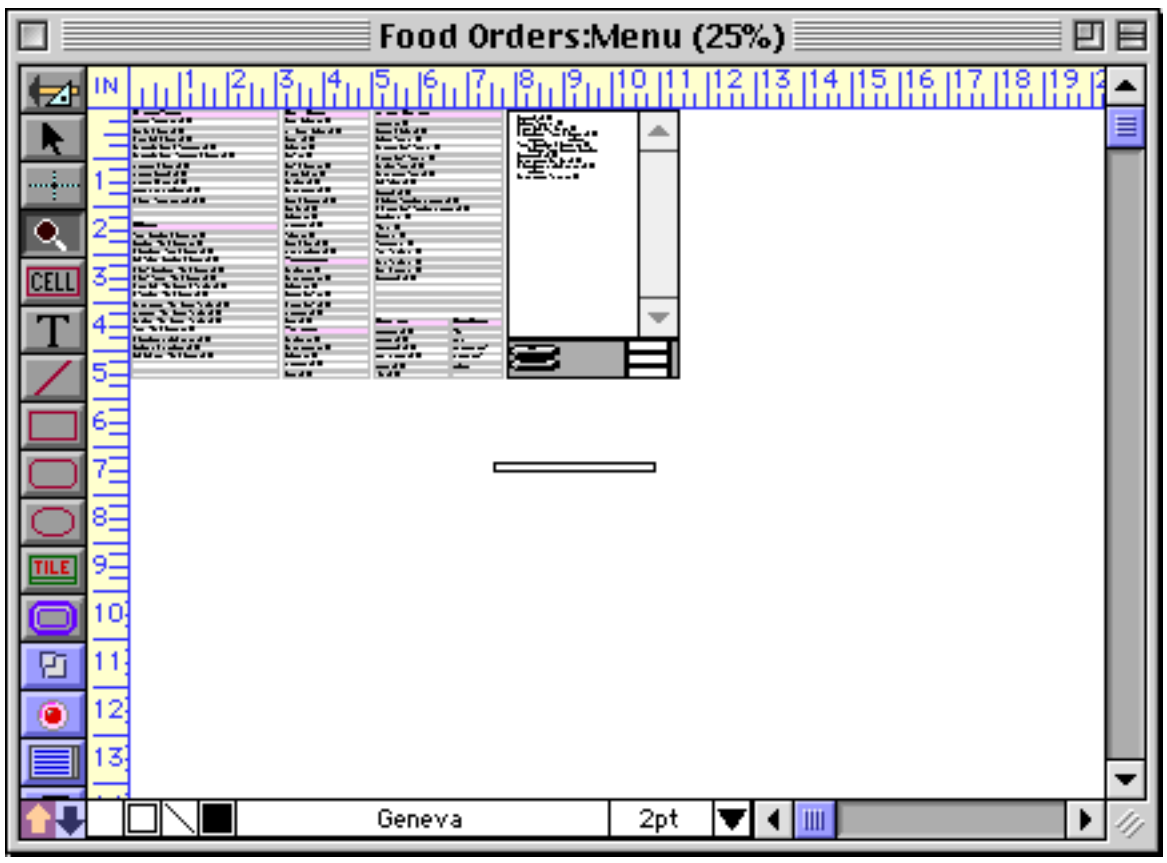
Move this magnifying glass to the spot you want to work on, then click the mouse to zoom in.



You can continue to zoom in up to 8X magnification.



Press the **Shift** key to turn the magnifying glass into a microscope. (If you are using a Macintosh pressing the **Option** key or **Space Bar** also converts the magnifying glass into a microscope. If you are using a Windows PC you can also press the **Alt** key or **Space Bar**.) When the **Shift** key is held down, each click reduces the magnification (zoom out). You can continue clicking until you get back to 1x magnification, or you can zoom out even farther and reduce to 50% or 25% magnification.



All of Panorama's graphic editing tools and menu commands work at any magnification level—all the way from 25% to 8x.

By holding down the **Command** key (Macintosh) or **Control** key (Windows) and clicking you can force Panorama to immediately return to the normal 100% view.

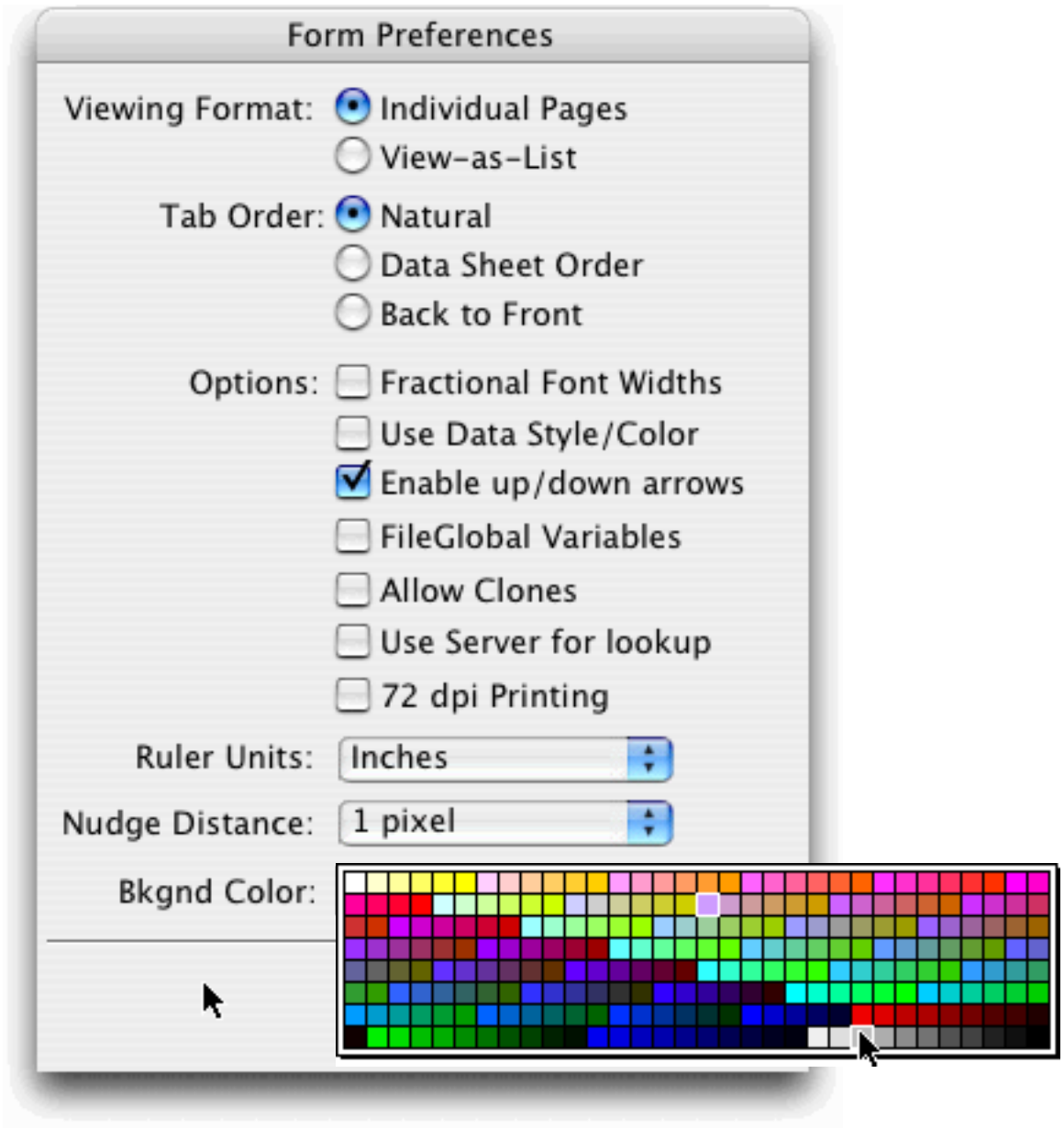
A Note About Measurement Accuracy

Panorama keeps track of the position and size of each object to an accuracy of 1⁄576 inch (1⁄8 pixel). However, this accuracy is only visible when you zoom in to 8x magnification (see previous section).

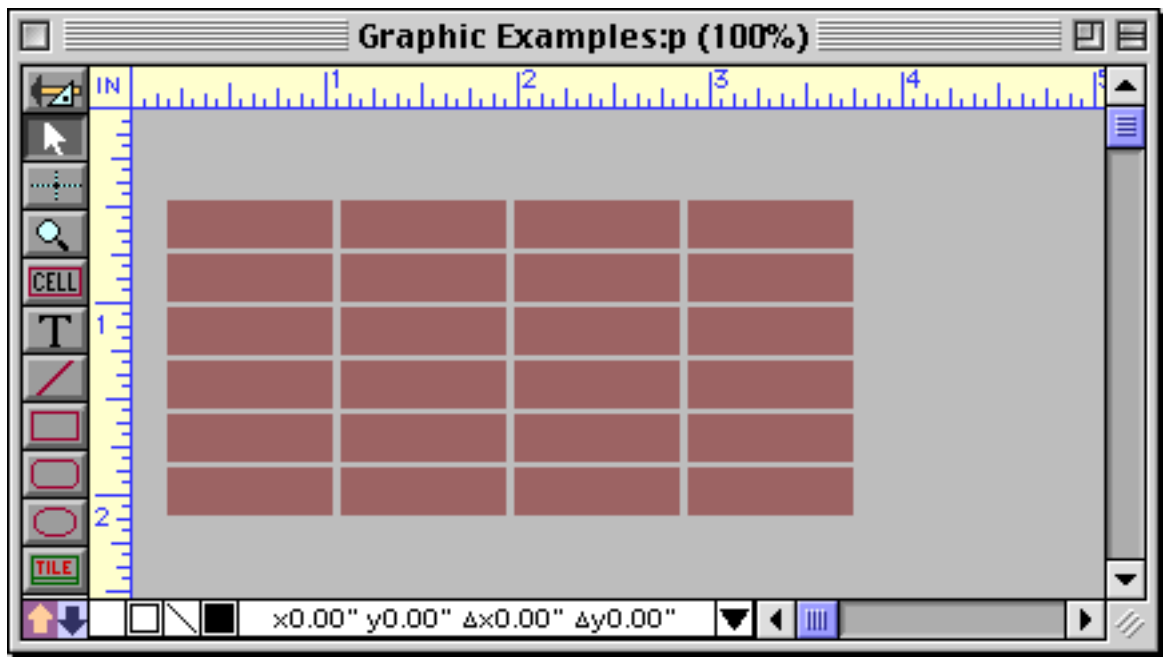
If you are going to print your form, keep in mind that printers are not 100% accurate. We have found that many printers can be up to 1⁄8 inch off over the length of a page. (For example, a 10 inch high line may actually print anywhere from about 9.9 to 10.1 inches.) This variation can cause problems if you are attempting to print unusually small labels or if you want to exactly match a pre-printed form. Most printers are more accurate horizontally rather than vertically—this is probably due to slight variations in drum speed as the paper feeds through the printer.

Form Background Colors

The default background for a form is white, but you may choose from 256 background colors for any form. To change the background color, open the **Form Preferences** dialog (in the Setup menu) and choose the background color from the pop-up menu.



Here is a typical form with a gray background color. (**Note:** If the form is displayed on a black and white monitor, Panorama will automatically use a white background even if you have specified another color.)



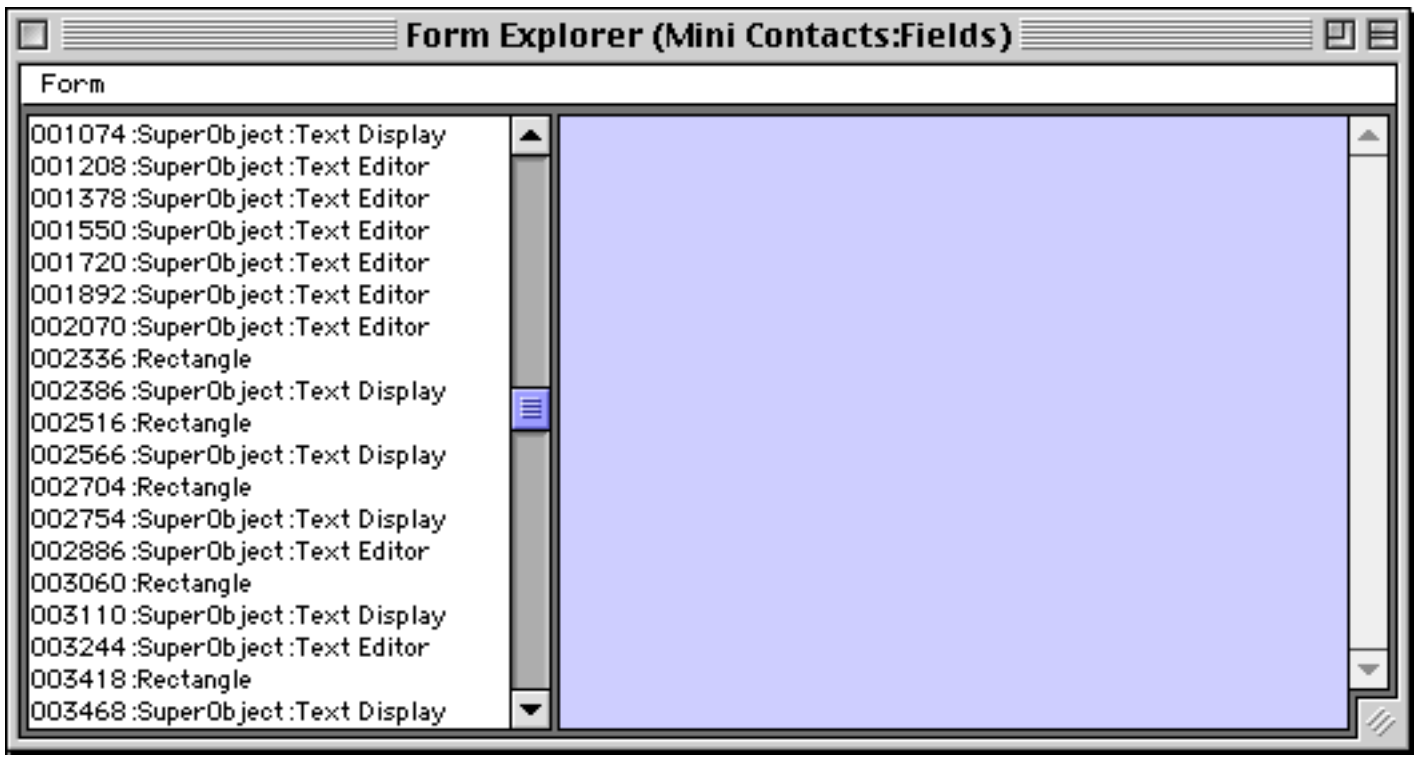
To set background colors for a view-as-list form, apply colors to the data tile and header tile. If these two tiles have different colors, the header and data sections of the form will also have different background colors. See "View-As-List Background Colors" on page 909 for more information on background colors in a view-as-list form.

It is possible to check and change the form background color from inside a procedure. See "FORMCOLOR" on page 5262.

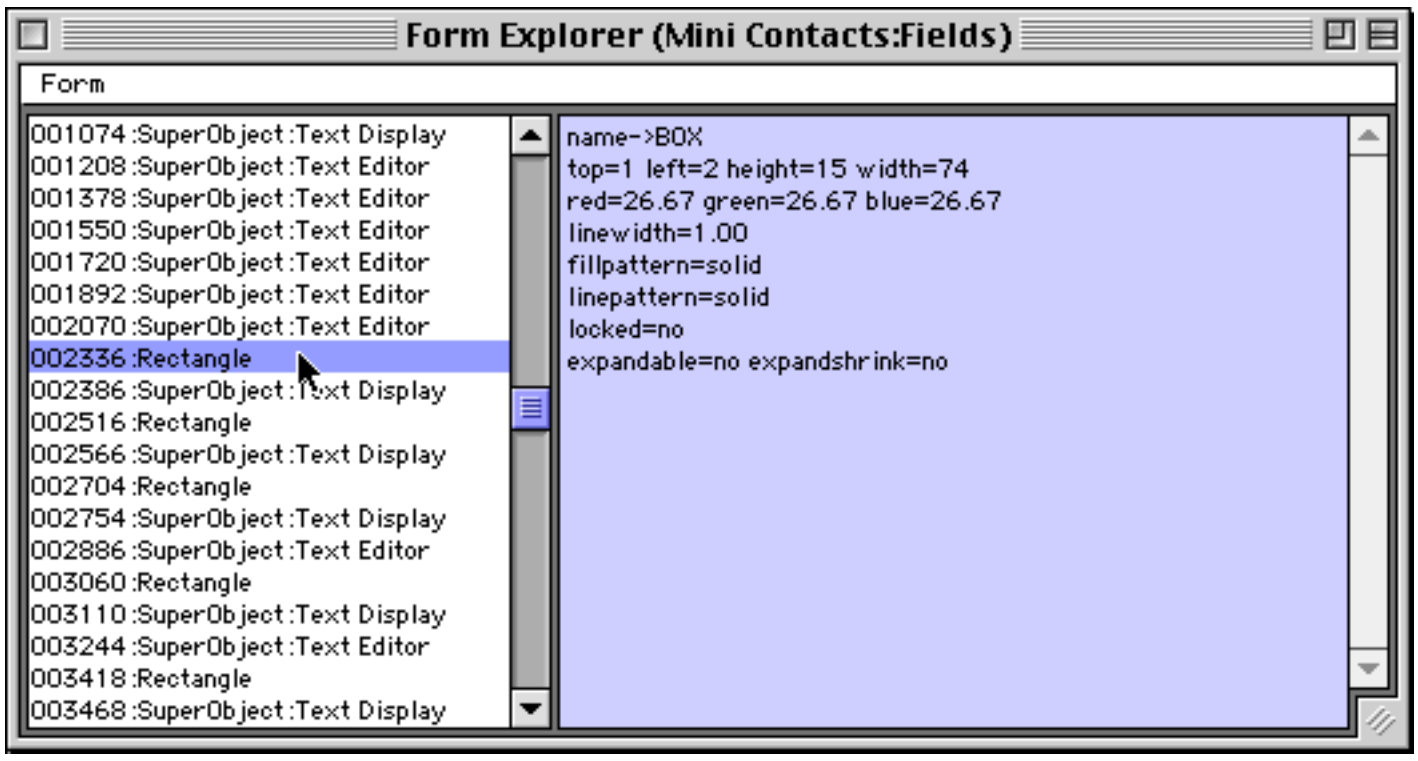
Using the Form Explorer Wizard

Panorama comes with a **Form Explorer** wizard that you can use as an alternative tool for examining and modifying (to some extent) forms. To use this wizard you must start with a form in Data Access Mode, not Graphics Mode. Choose **Form Explorer** from the Form Tools submenu of the Wizard menu.

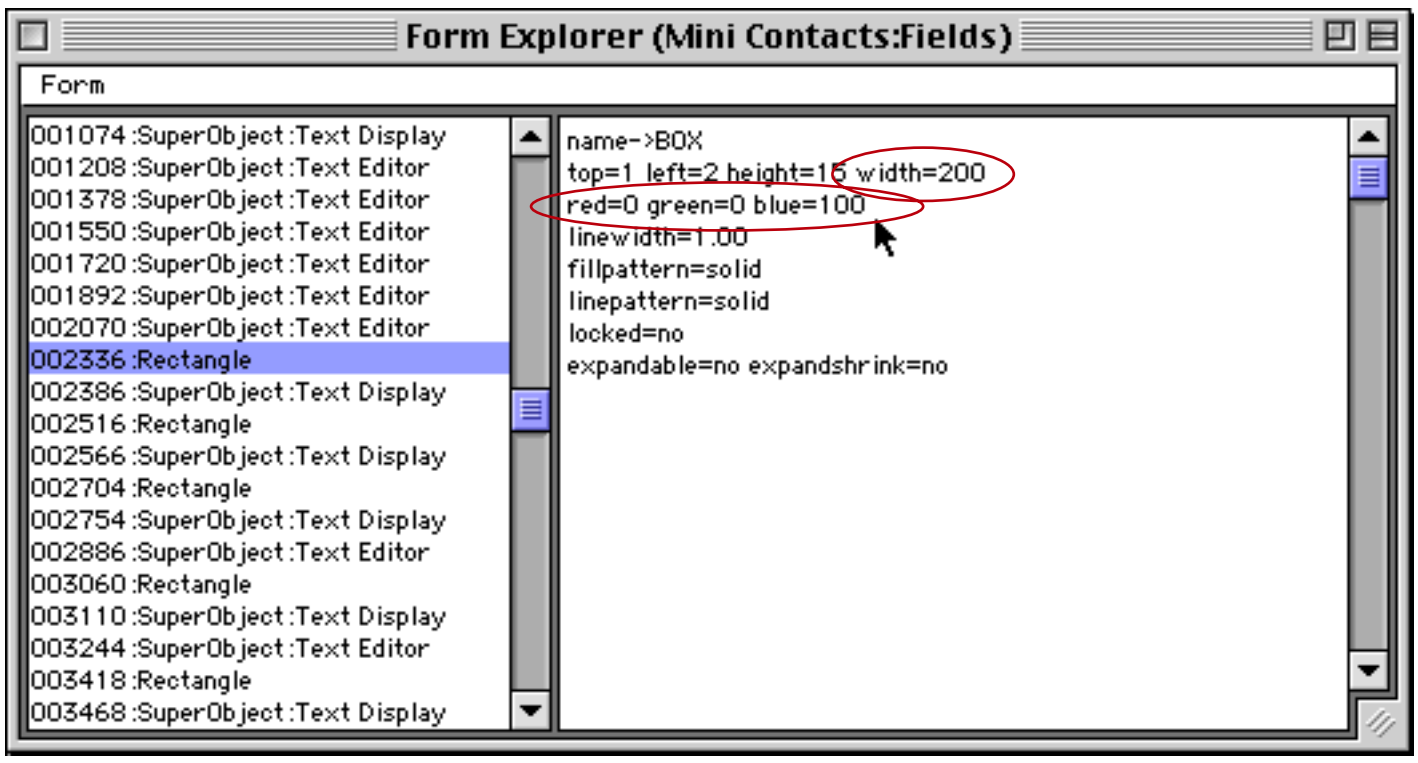
The **Form Explorer** window opens. The left hand side of this window contains a list of all of the objects in this form. The objects are listed from back (top of the list) to front (bottom of the list). The number for each object is relative to the front to back stacking order of the object - lower numbers are further towards the back.



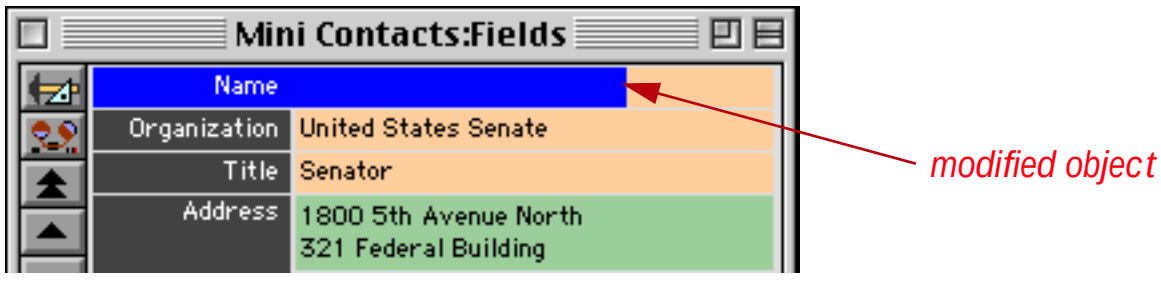
When you click on an object the object will flash on the original form (unless it is invisible). This will also cause a listing of all the object attributes to appear.



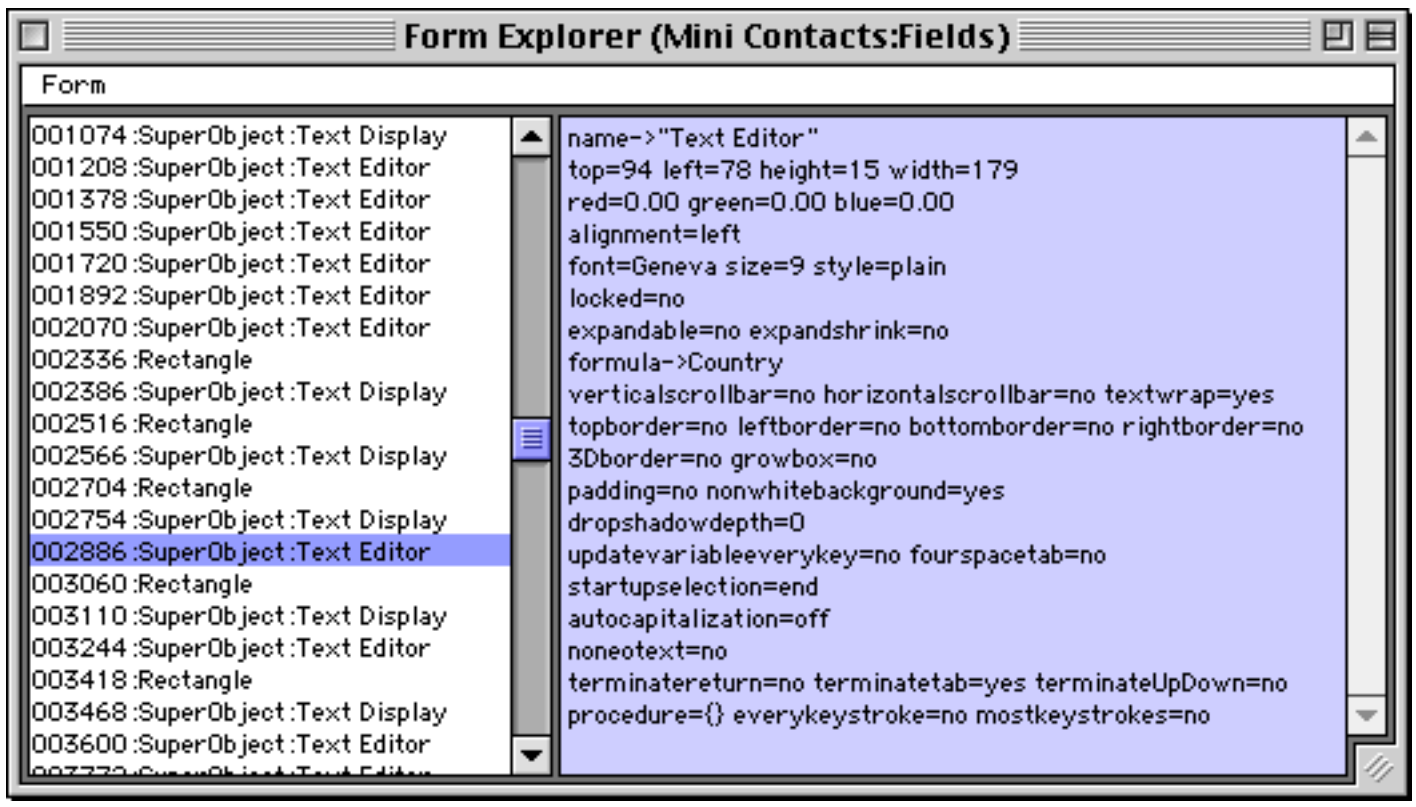
Each attribute is displayed as a name value pair, for example name->BOX or width=74. If a name / value pair is connected with an = sign then you can click on the text and modify the value. For example you can change the object width to 200 pixels and the color to solid blue.



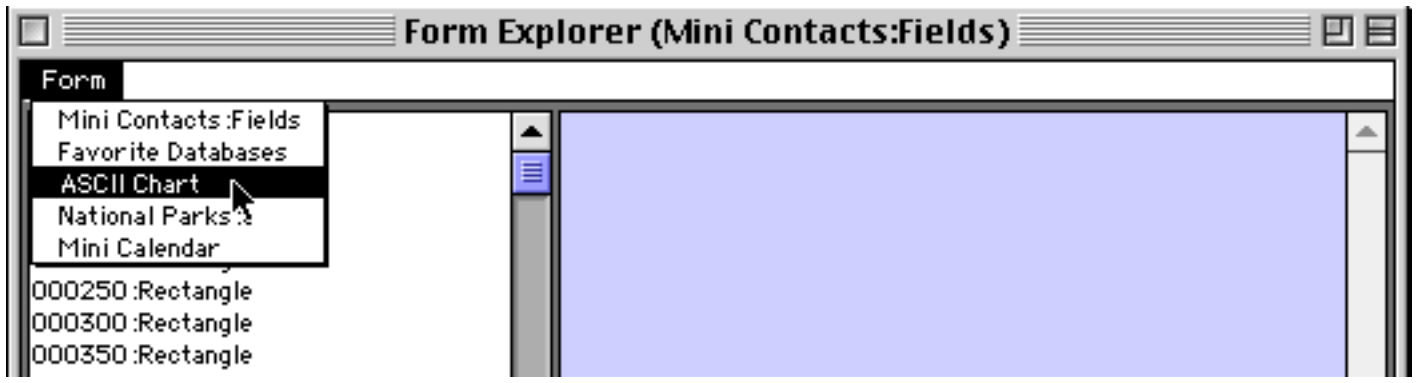
When you press the **Enter** key the object will be modified with the new attributes.



Some objects have more attributes than others. This Text Editor SuperObject has over three dozen attributes. In general the attributes correspond to the options in the object's configuration dialog. You can edit most of these attributes with the **Form Explorer** if you wish.



You can use the **Form** menu to explore any open form. You should also use the menu to update the object list after you edit the form in Graphics Mode (or you can simply close and re-open the Form Explorer).



The **Form Explorer** wizard can be a great tool for deciphering a form created by another person. You can quickly zero in on which fields and procedures are used where.

Chapter 15: Displaying and Editing Text



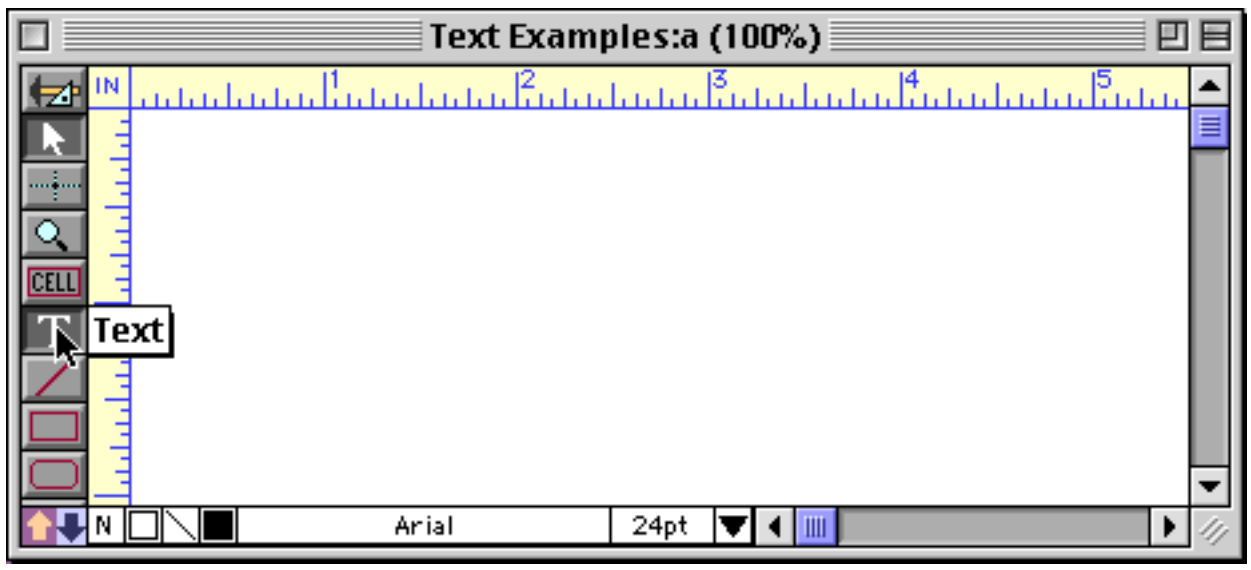
Graphics and icons are powerful tools, but written text is still the primary medium for communication and record keeping. Panorama forms can display both permanent text (for captions, titles, instructions, etc.) and textual information from the database.

Displaying Text

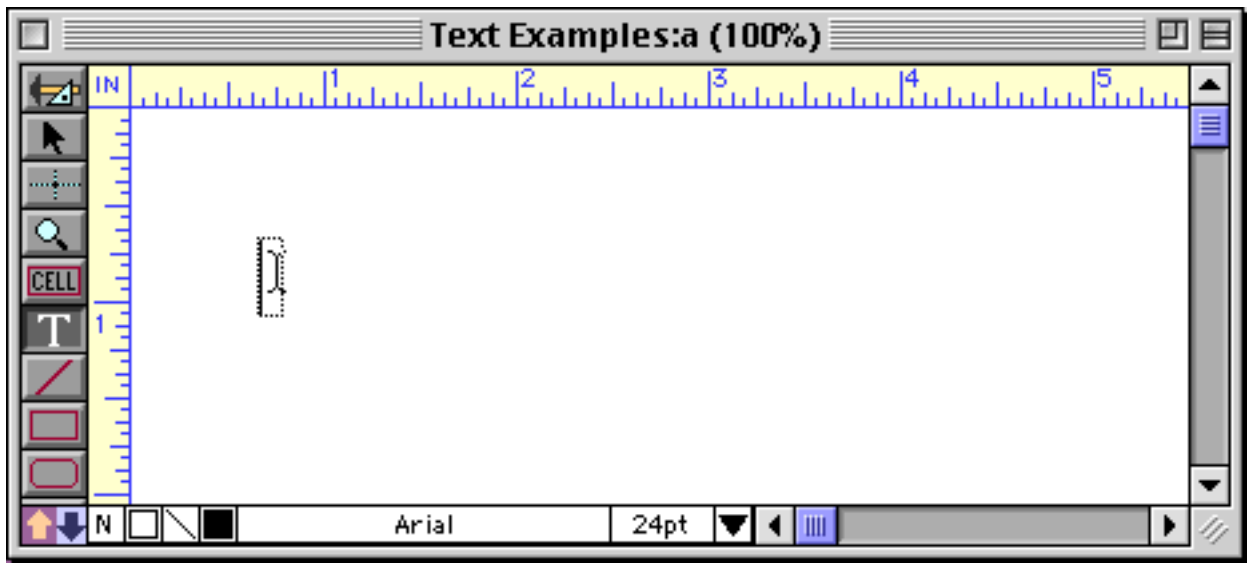
This chapter is divided into two sections, displaying text and editing text. In this first section we'll consider displaying text, both fixed text and text that changes depending on the information in the database or in a variable.

Fixed Text Objects

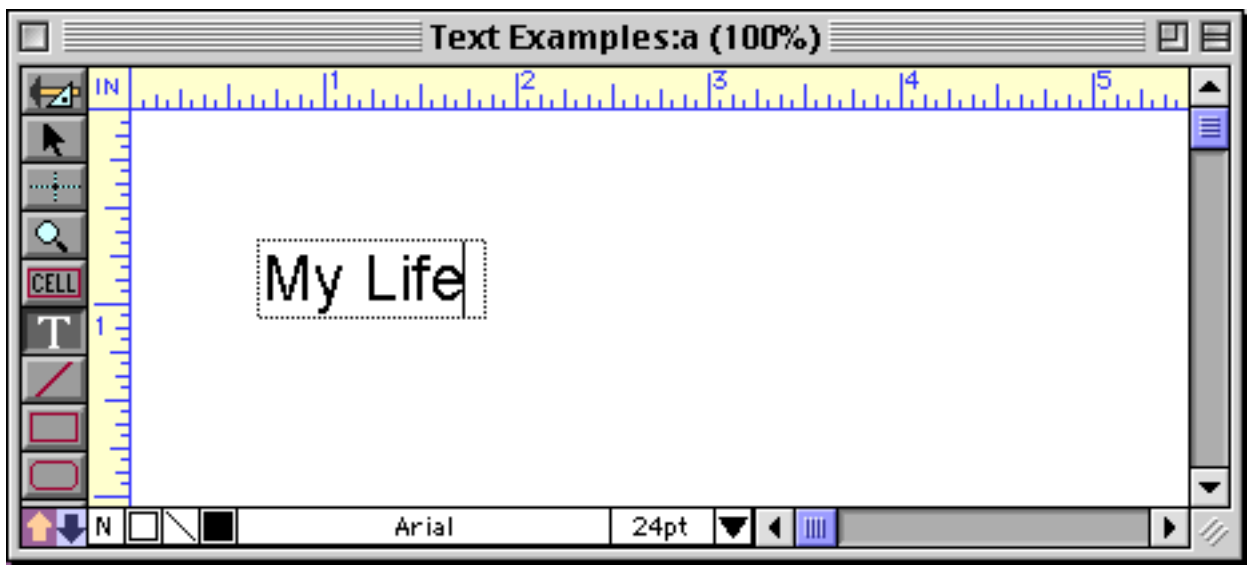
Panorama has two different kinds of fixed text, **click text** and **auto-wrap text**. Both types of fixed text are created with the **Text** tool.



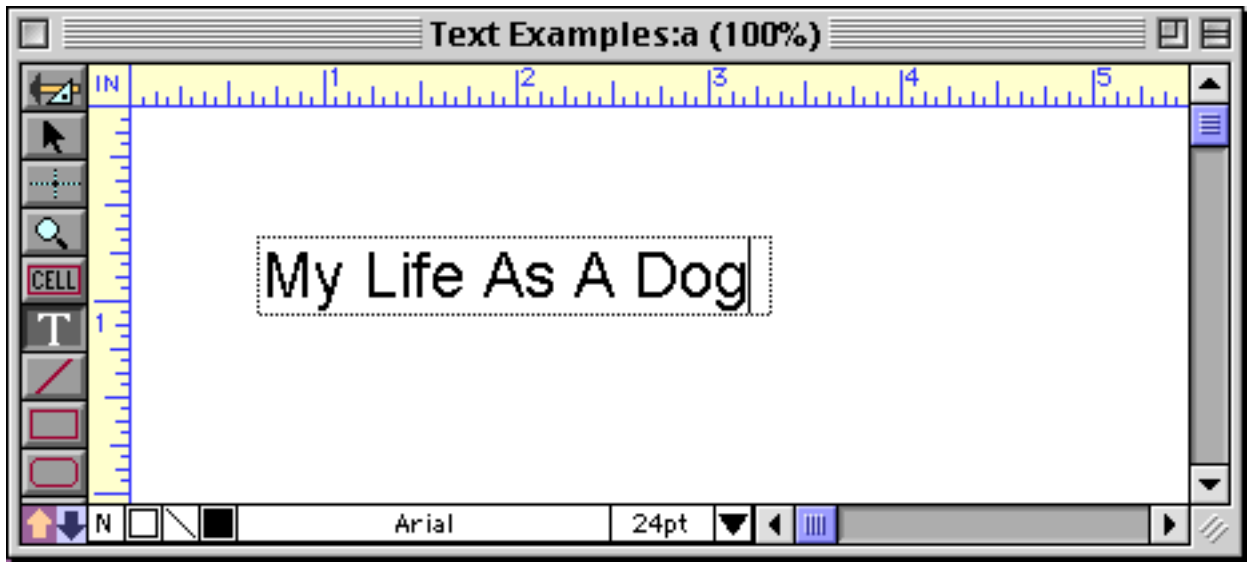
Once the **Text** tool is selected you can create click text simply by clicking on an empty spot on the form.



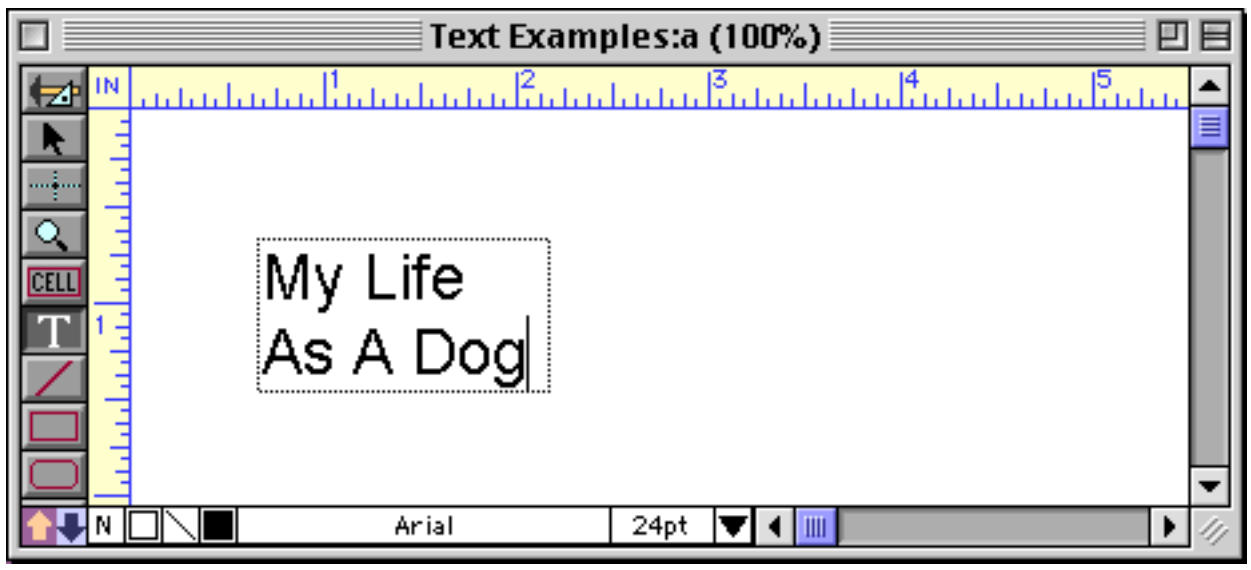
Now you can simply start typing.



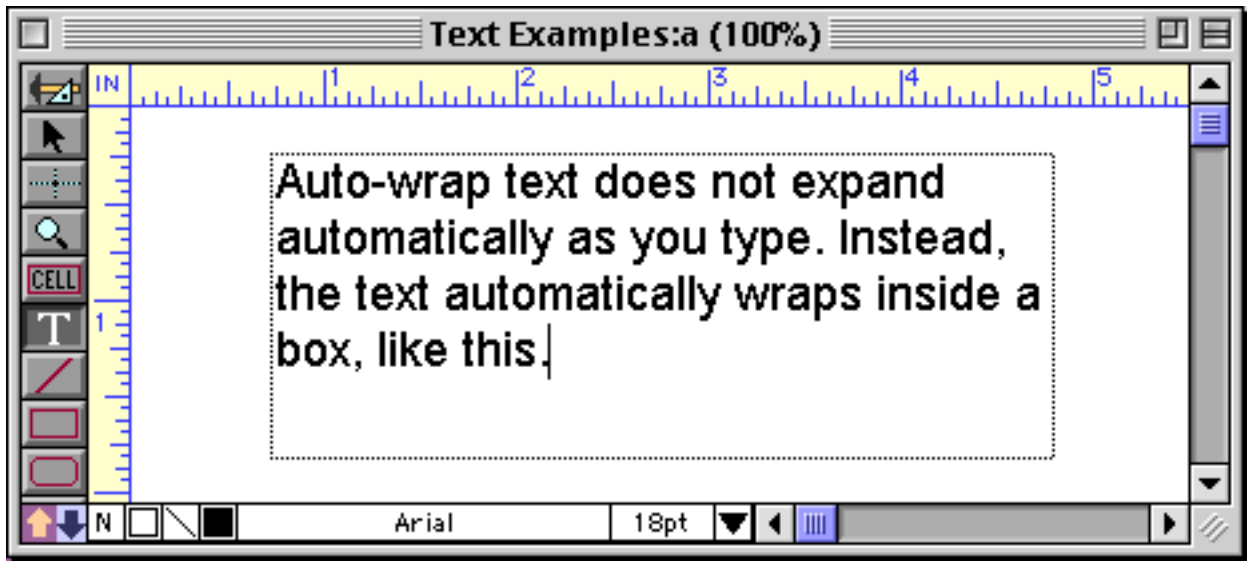
As you type each character, the click text object automatically expands.



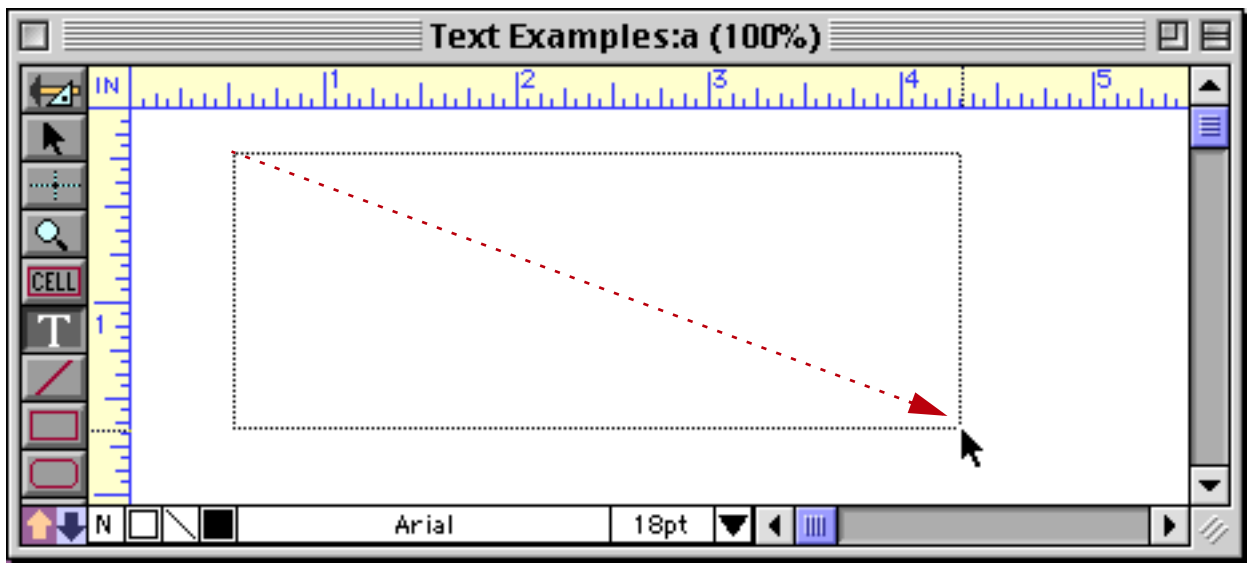
You can add additional lines to the click text by pressing the **Return** key.



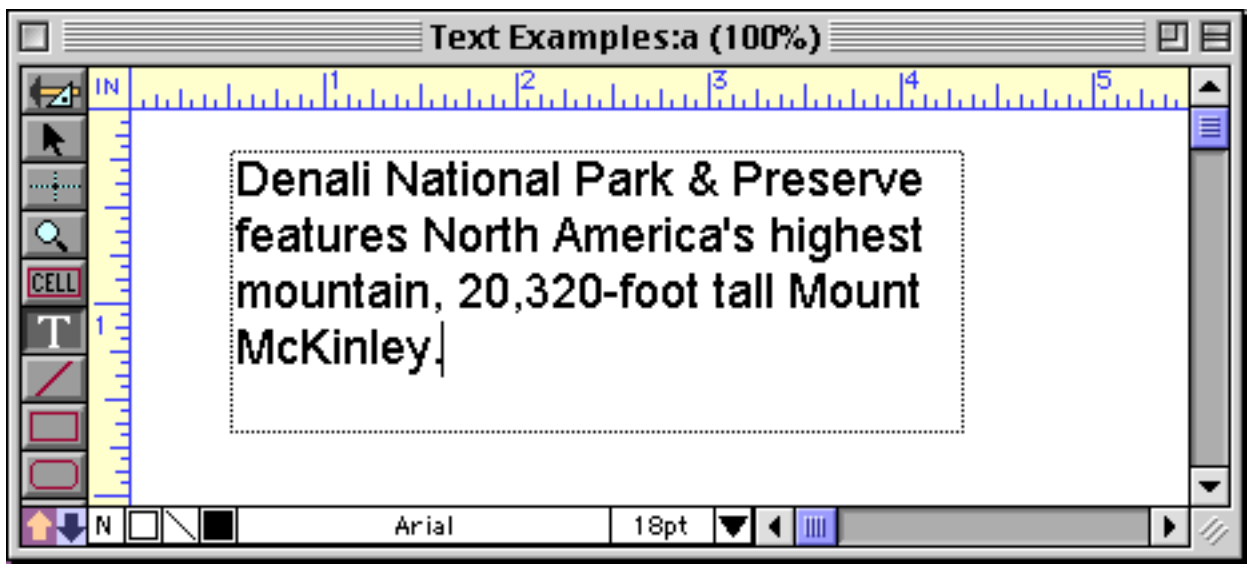
Auto-wrap text does not expand automatically as you type. Instead, the text automatically wraps inside a box.



To create **auto-wrap text**, start by selecting the **Text** tool, just like for **click text**. But instead of clicking on the form, drag the mouse to define the size and location of the box (don't worry about exact positioning, you can adjust it later).



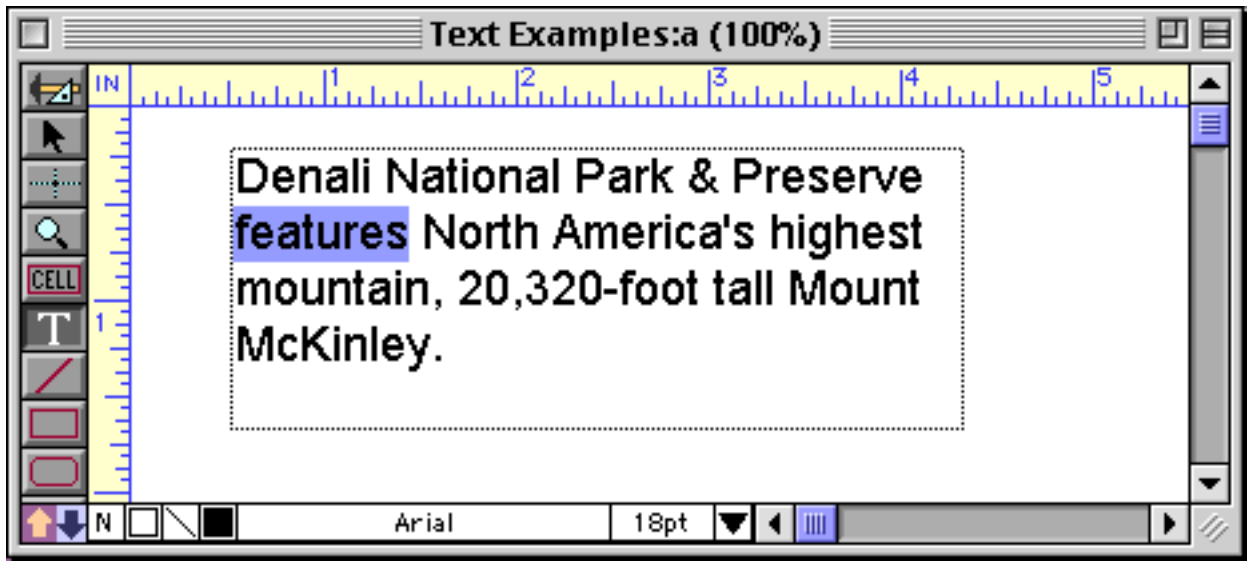
Once the box is defined you can start typing in text. The text will automatically wrap to the next line when it reaches the right edge of the box.



To help you keep track of the type of text you have created, the graphic editor displays a dotted border around all auto-wrap text objects. (This border disappears when the form is switched to data access mode.)

Editing Fixed Text

To edit text within an object, you must use the **Text** tool. Once this tool is selected, move the mouse over the text you want to edit. When the mouse moves over the text it changes from an arrow to an I-beam. Use the I-beam to edit the text—click to select an insertion point, drag to select a range of characters, or double-click to select a word.



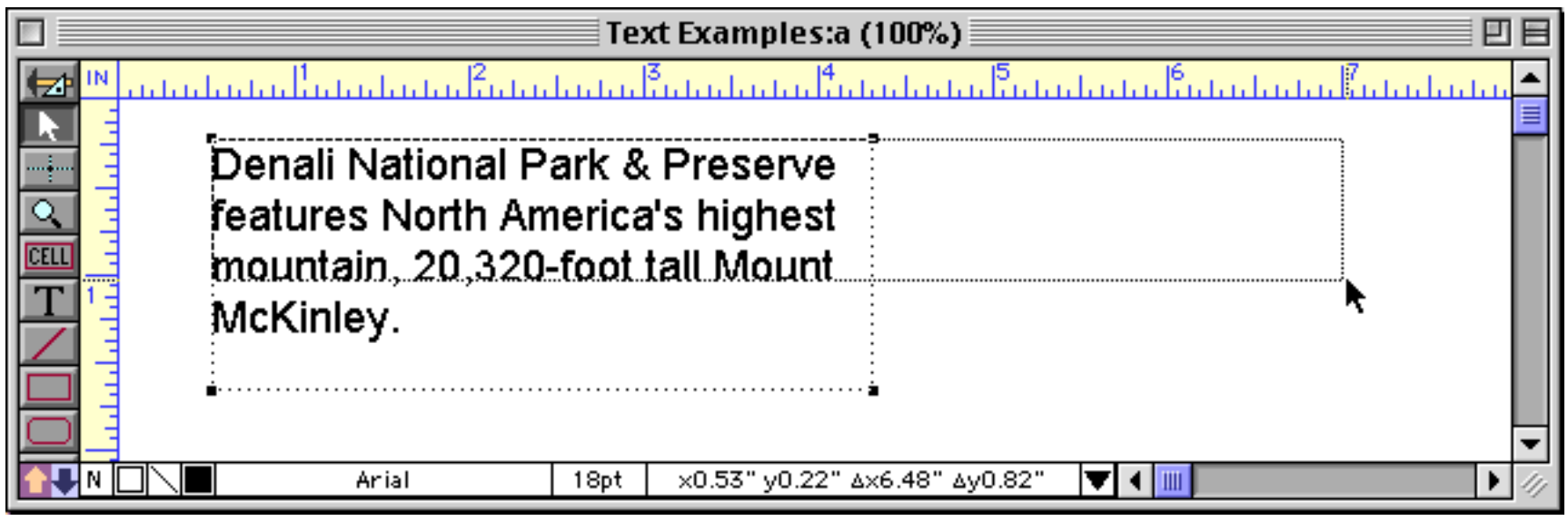
When you have finished editing the text, either click on the next text object you want to edit or click on the **Pointer** tool.

Note: If a text object doesn't contain any text at all, the entire object will be deleted from the form. The first thing you should do after creating a text object is to type at least one character into it.

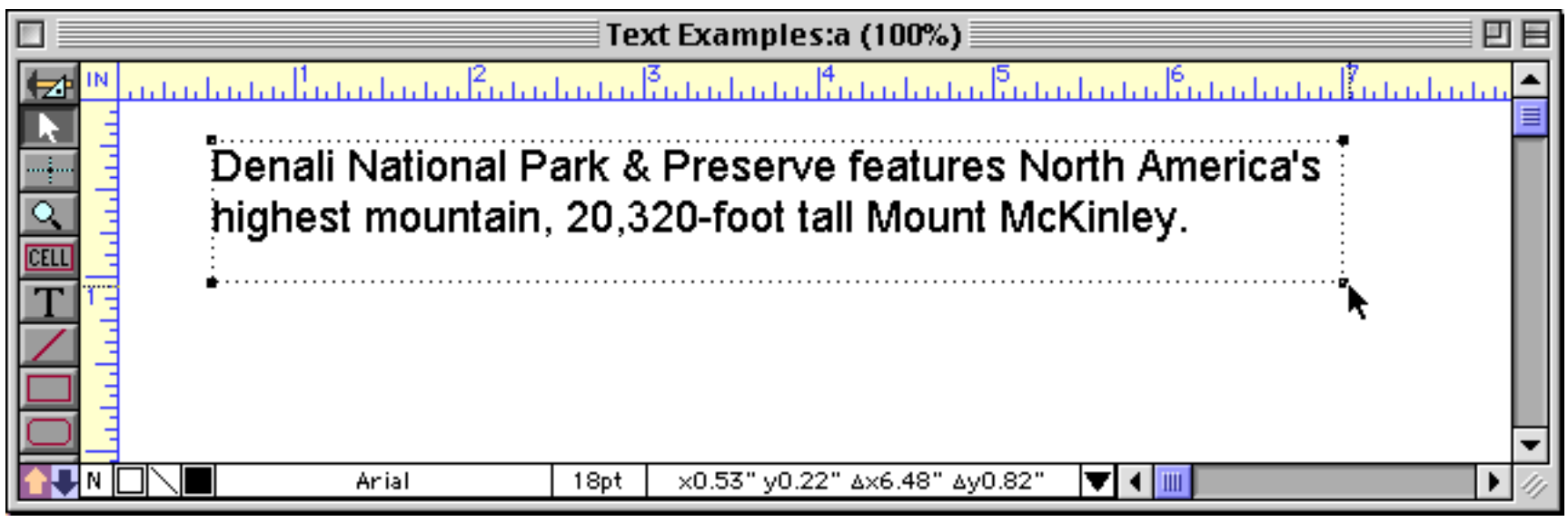
Moving and Resizing Fixed Text Objects

Text objects can be moved just like any other shape. Use the **Pointer** tool to drag the text to the new location.

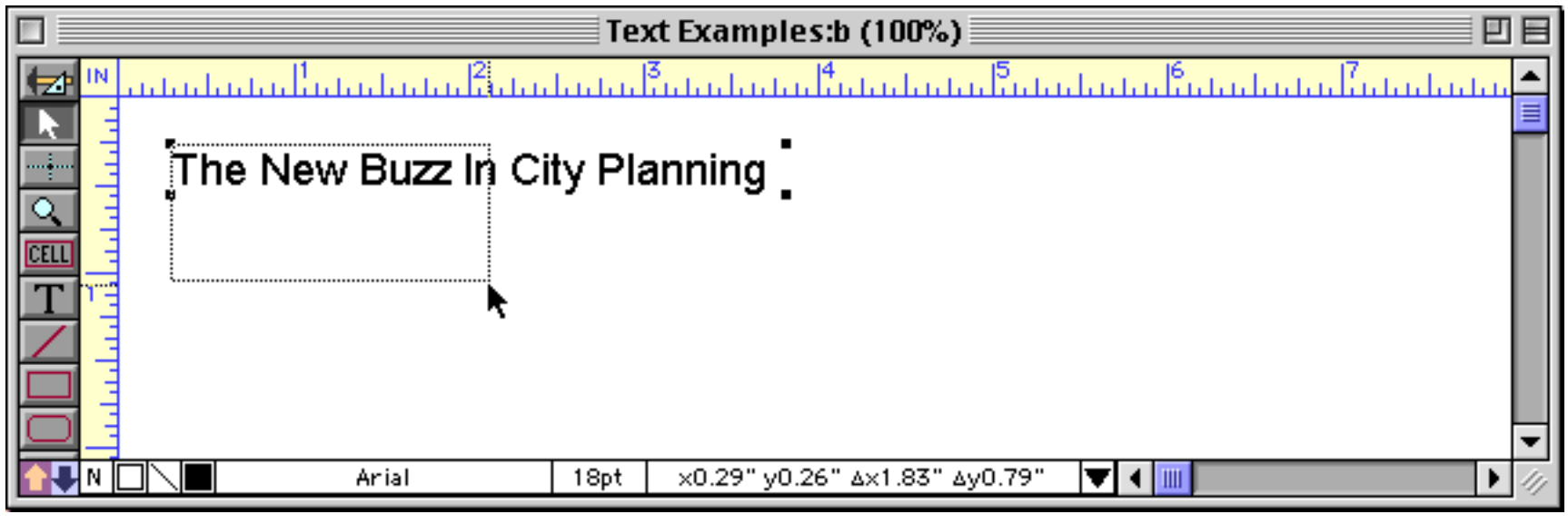
The size and shape of a text object can be changed by dragging one of the handles to a new position.



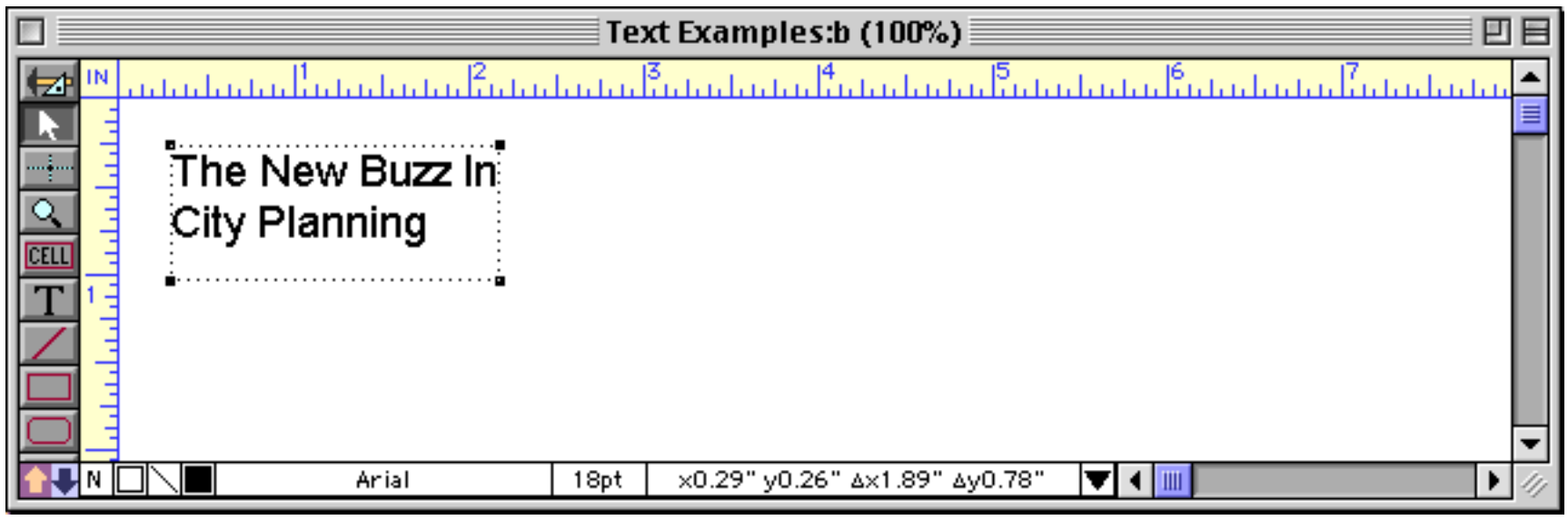
If this is done to auto-wrap text, the text will re-flow into the new size.



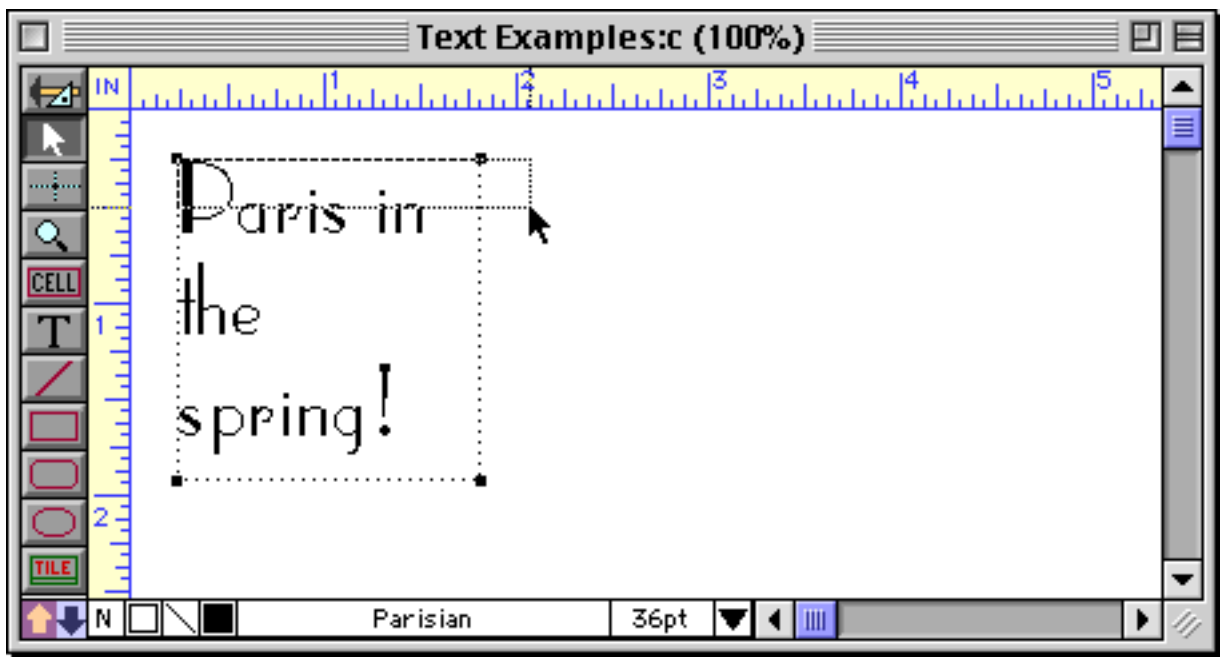
If you change the size of click-text, it will be converted into auto-wrap text.



Once the click-text is converted to auto-wrap text it will re-flow into the new size.



To convert auto-wrap text into click text, resize the object so that it is less than one line high or less than one character wide.



Panorama converts the auto-wrap text into click text.



Later the text can be converted back into auto-wrap text simply by expanding the height again.

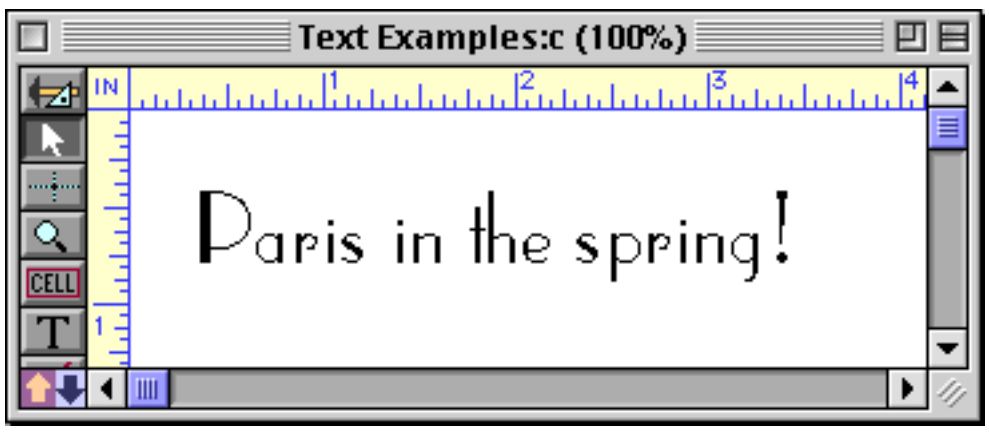
Text Font, Size and Style

Text in a form may be displayed using any font installed in your system. However, you cannot mix different text styles, sizes, or fonts within a piece of text. To change the font, size, or style of an entire text object, use either the **Pointer** or the **Text** tools to select the object (or objects), then change the text appearance by choosing from the **Font**, **Size**, and **Style** Menus. You can also change the color of the entire text object with the Color menu (in the Graphics menu or the Graphic Control Strip).

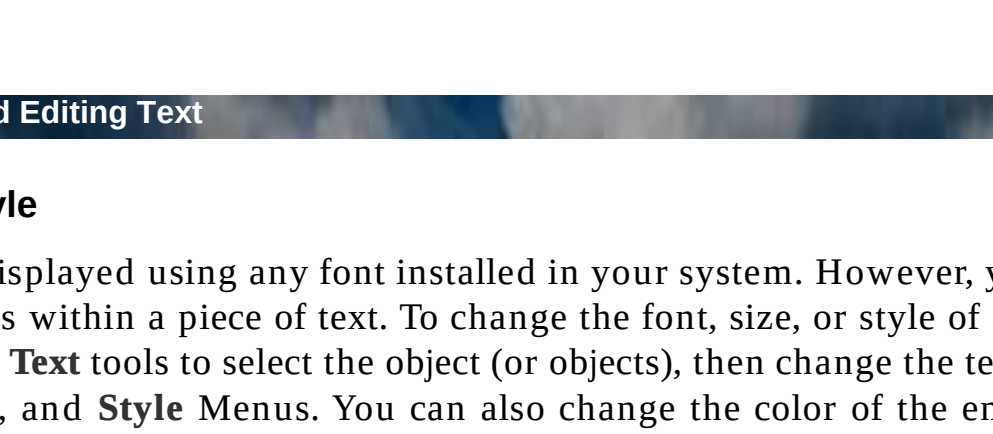
If you need a size that is not listed in the **Size** menu pick Other. You can also change the text size in 1 point increments by choosing **Up** or **Down** from the Size menu.

Creating Reverse Type (White on Black)

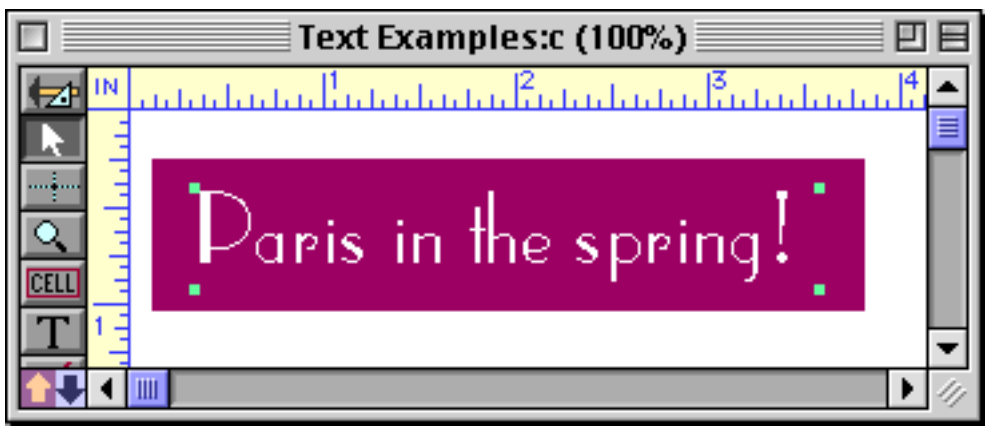
To print reverse type, start with regular text.



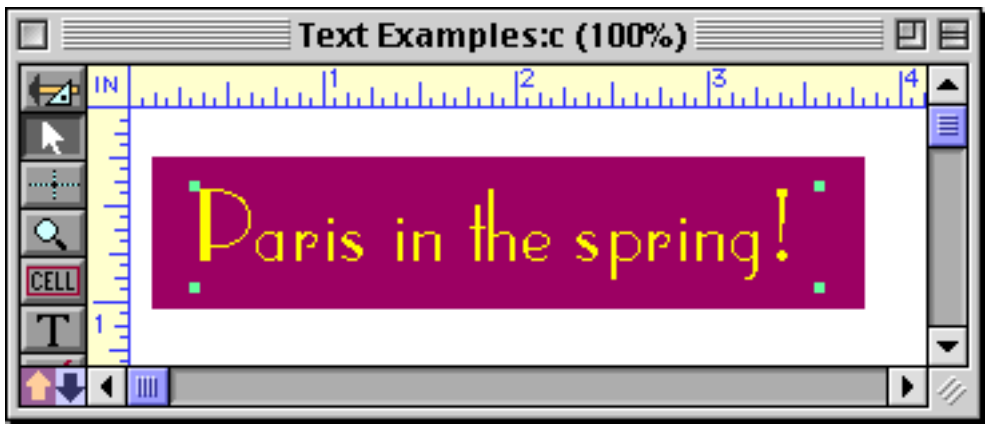
Next, put a black object (or any dark color) behind the type. Use **Send To Back** to send the dark object to the back (see “Changing the Stacking Order” on page 569).



Select the text object, then use the Line Pattern menu (see “Line Pattern” on page 523) to set the text to white.

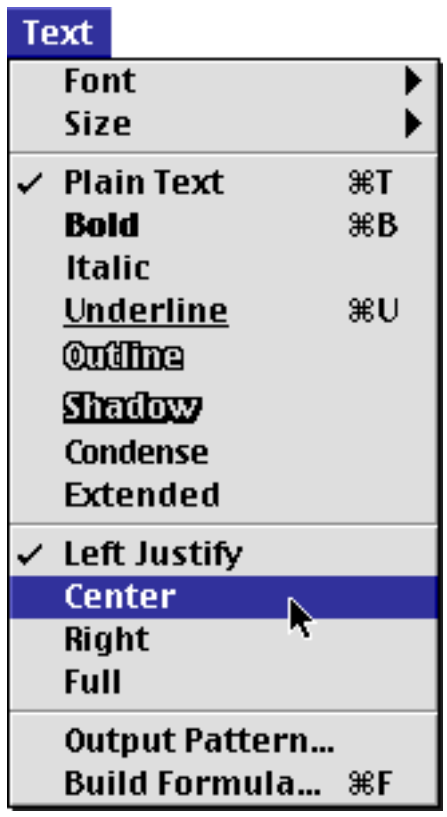


An alternative technique is to leave the Line Pattern alone and use the Color menu (see “Color” on page 526) to set the text to white or some other light color.



Text Alignment

Text is usually aligned flush left within the text object. Use the Left Justify, Center, and Right commands (Text menu) to change the alignment of the text.



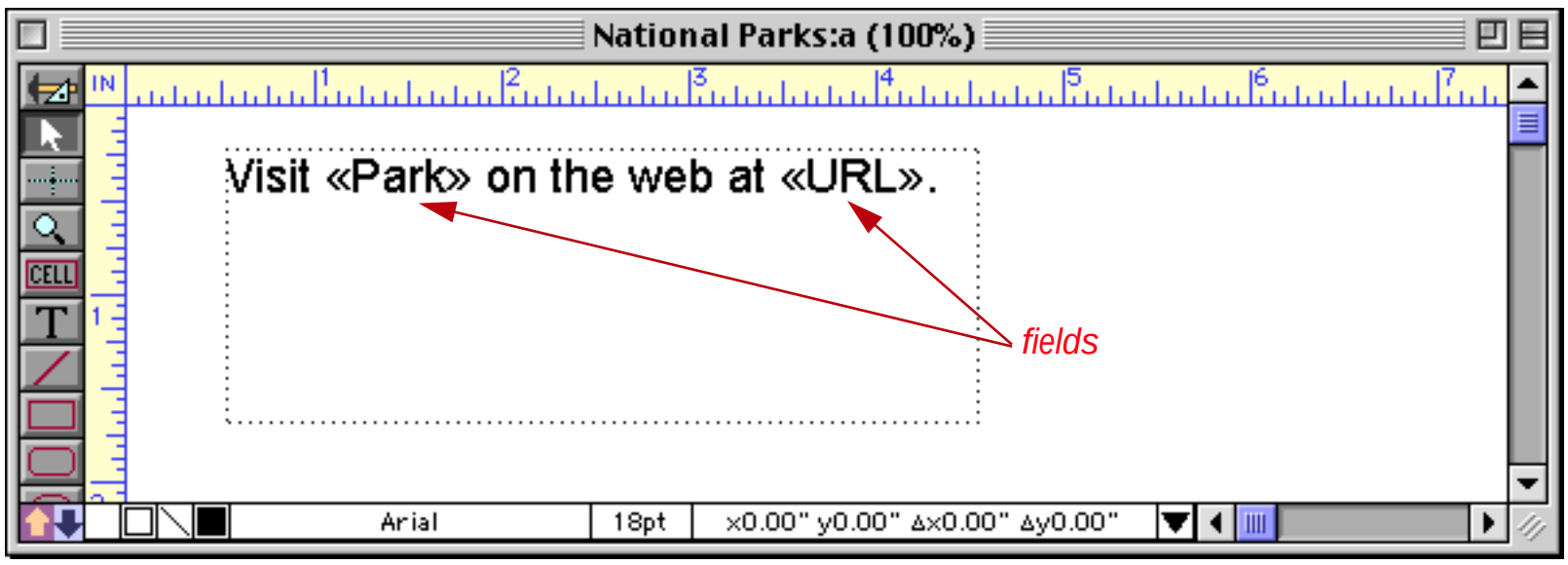
Displaying Data in Auto-Wrap Text

Auto-wrap text objects are not limited to fixed text. They can also be used to display data, either alone or in combination with fixed text.

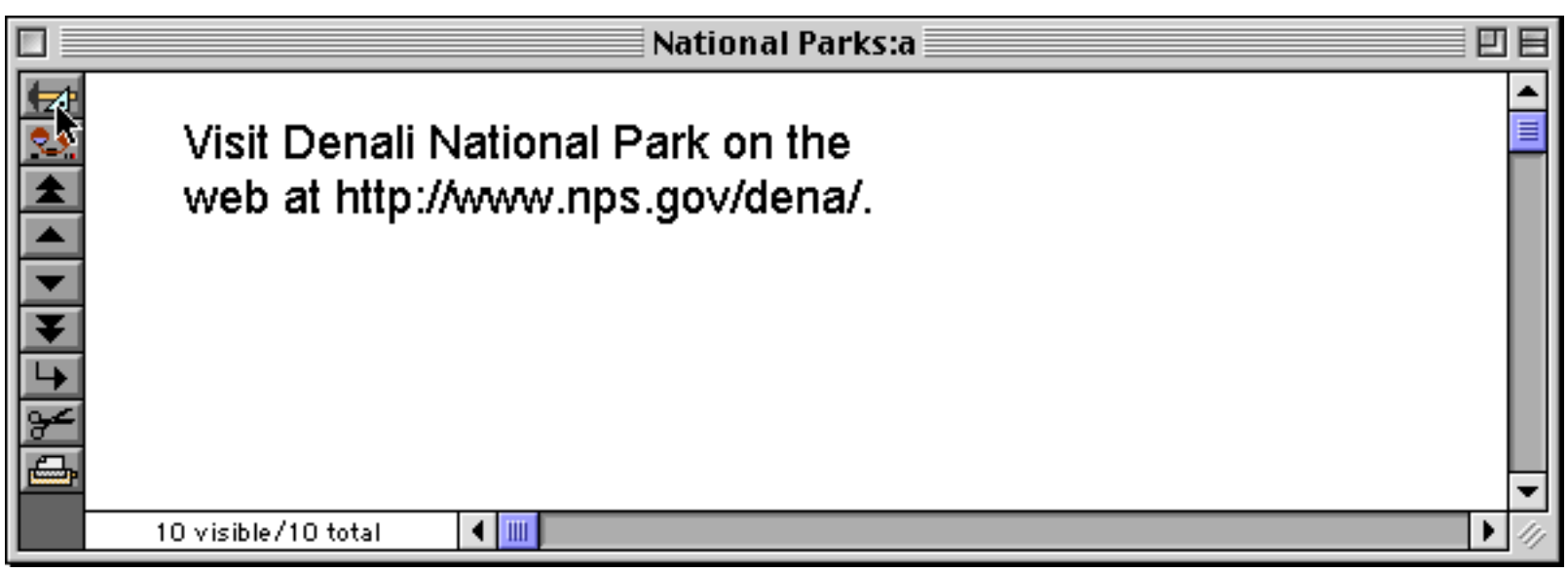
To display a field into the middle of an auto-wrap text object, type the name of the field into the text. Surround the name with the « and » chevron characters (for example «First Name» or «Zip Code»). On a Macintosh the « chevron is **Option-** and the » chevron is **Shift-Option-**. On Windows systems the « chevron is **Alt-0171** and the » chevron is **Alt-0187**. To illustrate this technique, suppose you had a database of national parks like this.



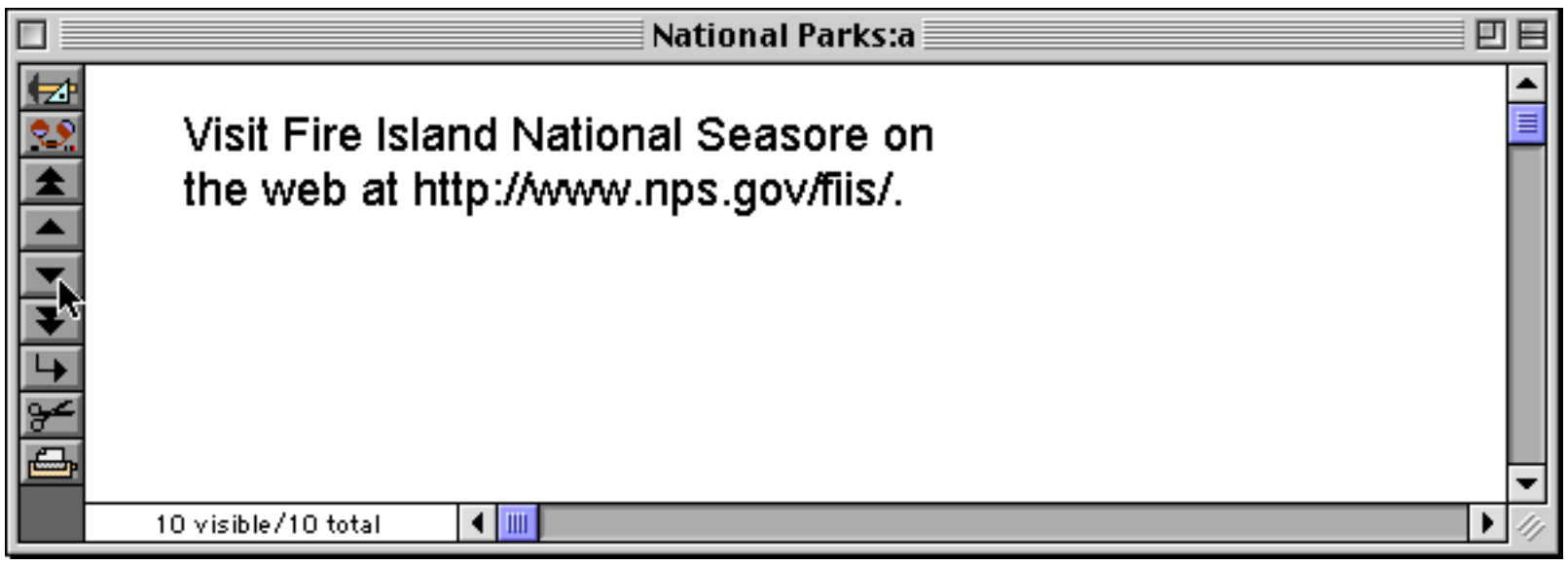
Now you can create an auto-wrap text object that contains fields, like this.



When the form is switched to Data Mode (see "Form Modes: Data Access vs. Graphic Design" on page 485) Panorama will substitute the actual data in this fields, like this.



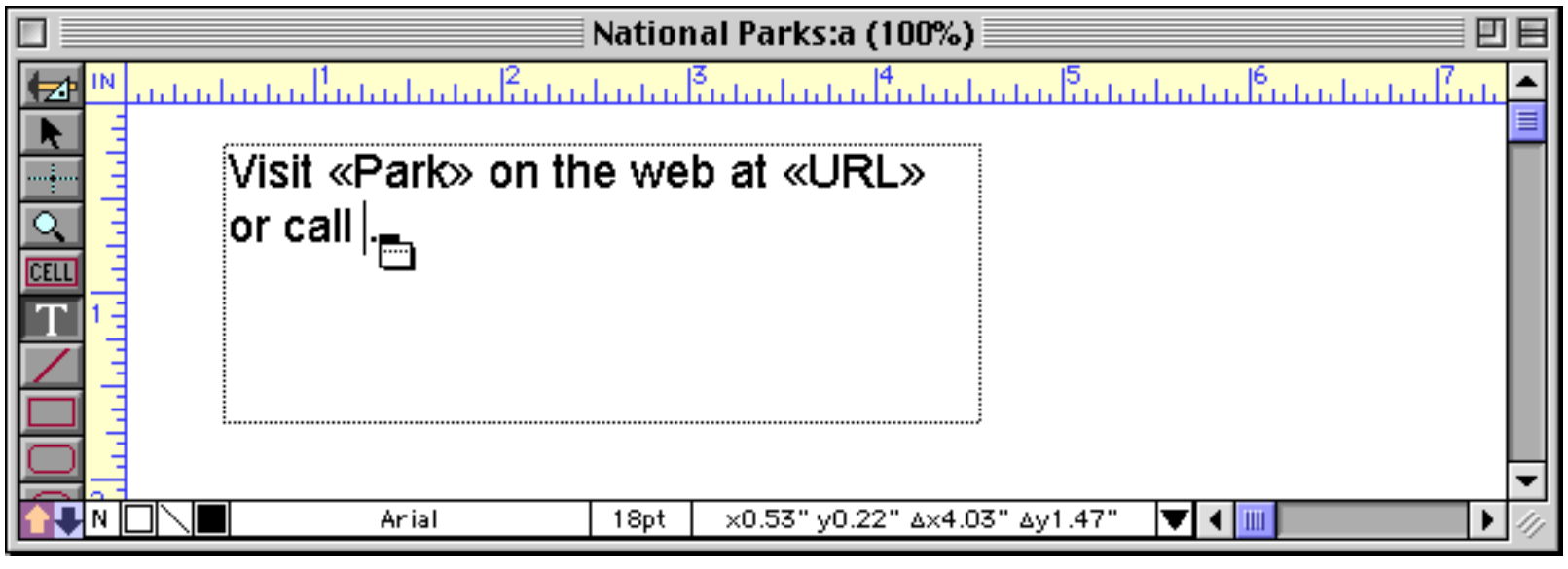
As you move from record to record, the substituted text will change appropriately.



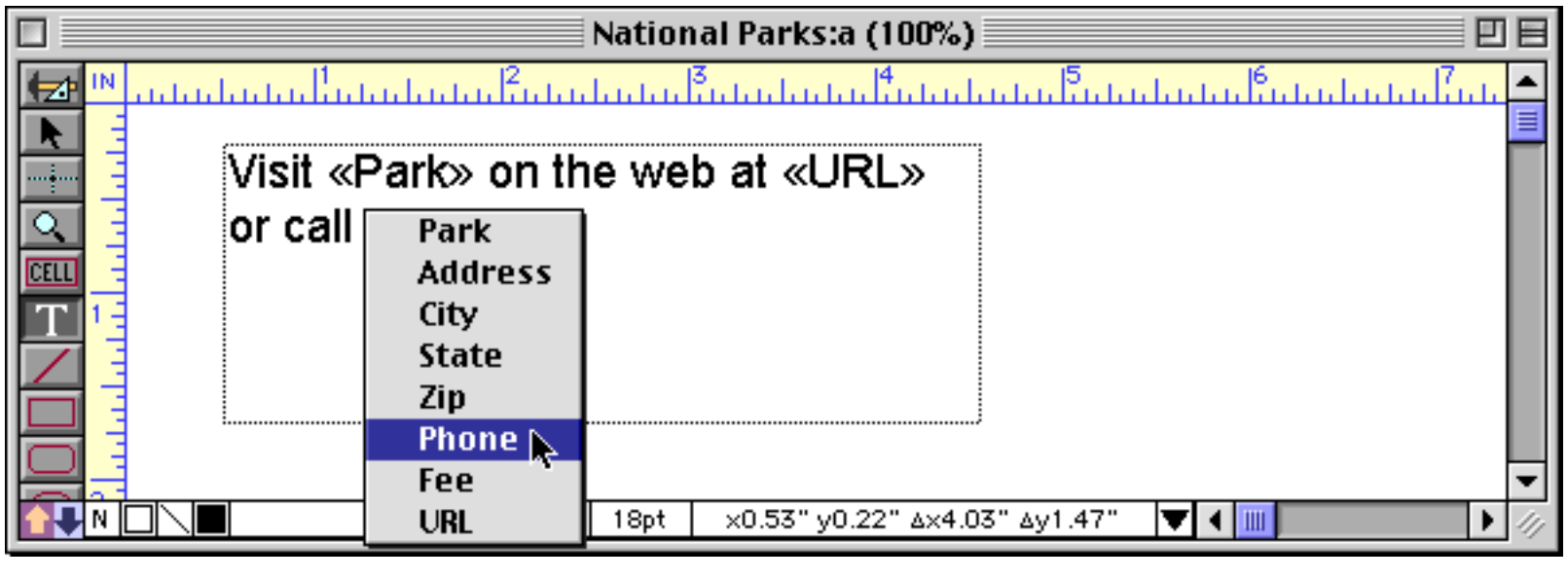
Since this technique “merges” the database information with the fixed text, it is sometimes called **data merging**.

Data Merge Pop-Up Menu

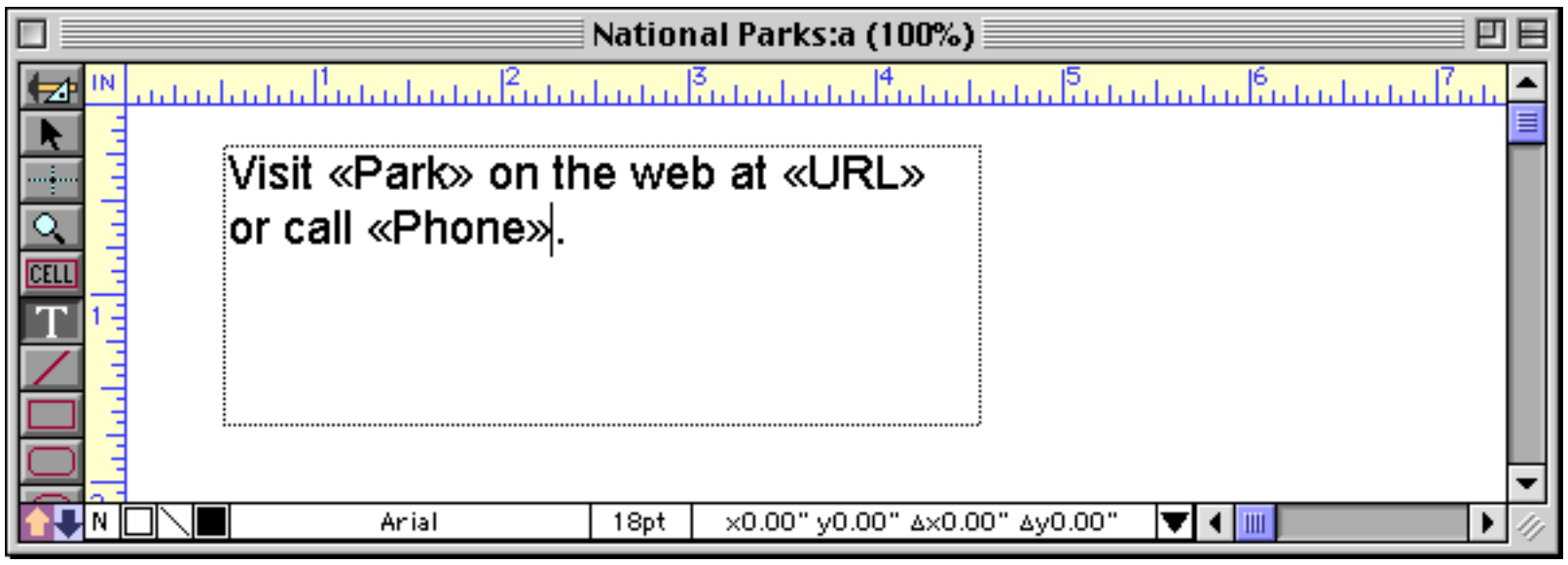
Typing in exact field names with chevrons can be a pain, so Panorama has a pop-up menu that can type in the field names for you, including the « and » chevrons. To use this menu, first select the **Text** tool. Then click on the text to create an insertion point. Once the insertion point is set, press either the **Command** key (Macintosh) or **Control** key (Windows) to change the cursor from an I-beam to a tiny menu icon.



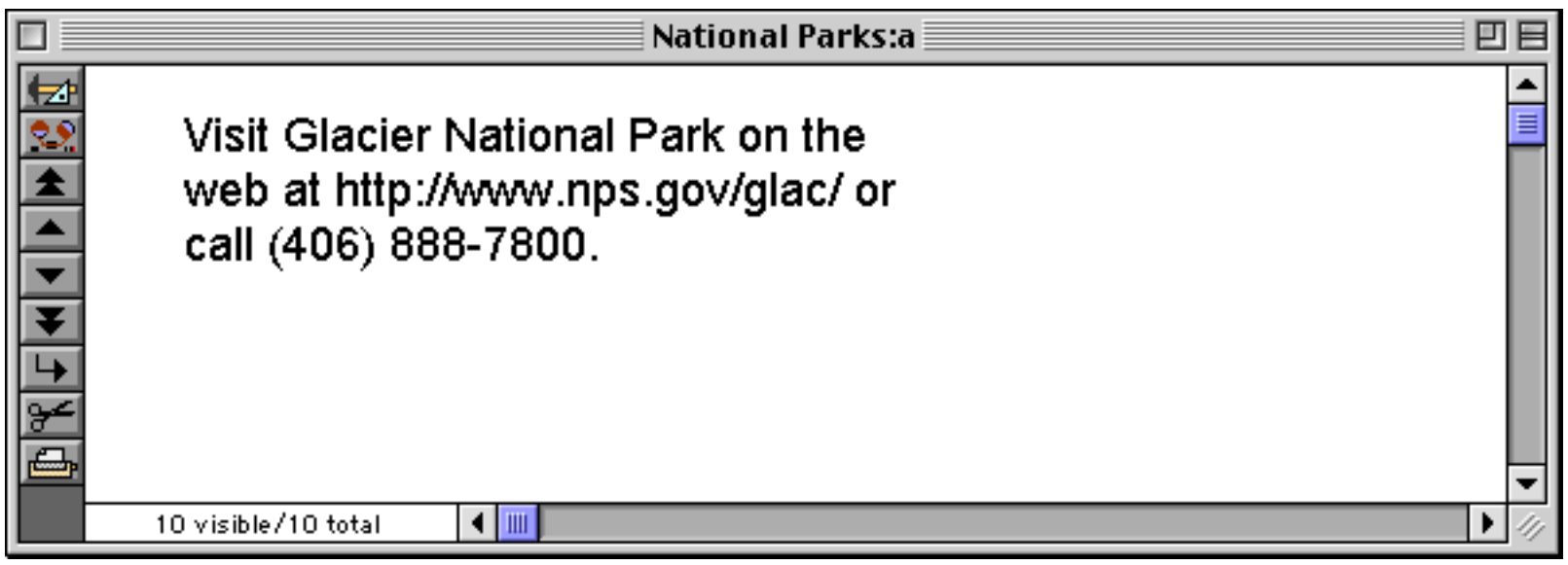
With the **Command**/**Control** key still held down, press the mouse to activate the pop-up menu



then pick the field name you want to insert.

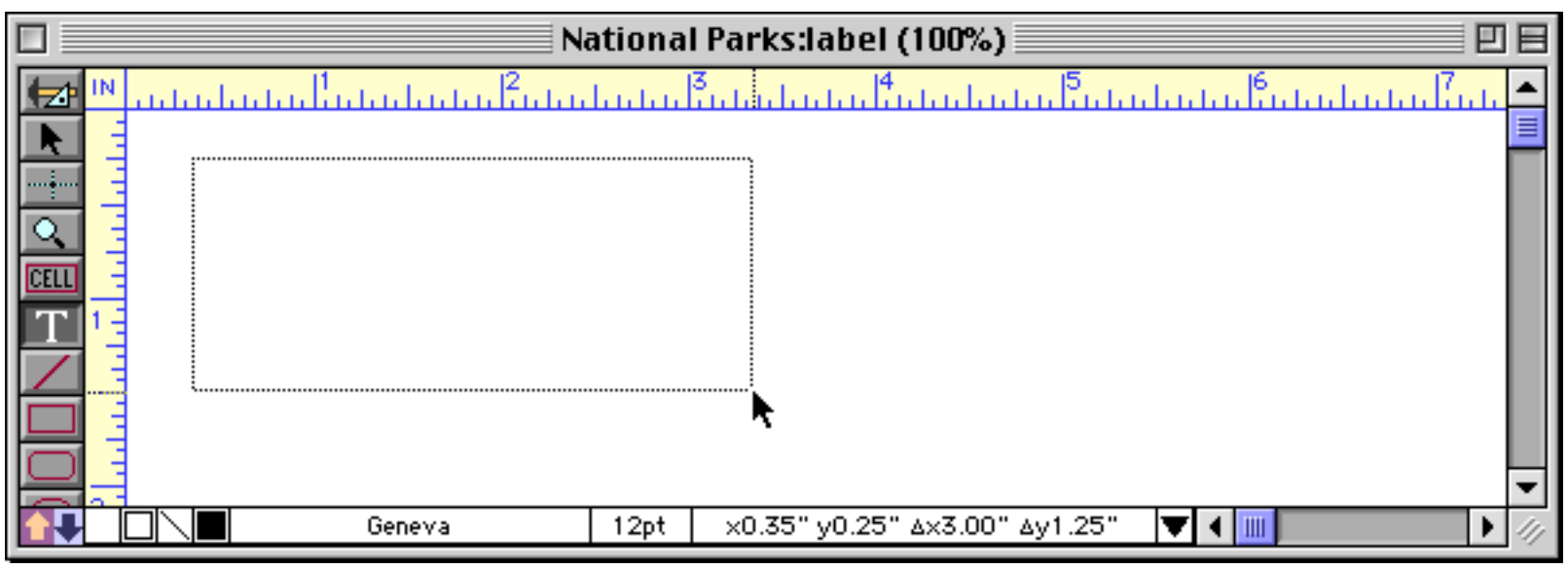


Switch back to Data Access Mode to see the final result.

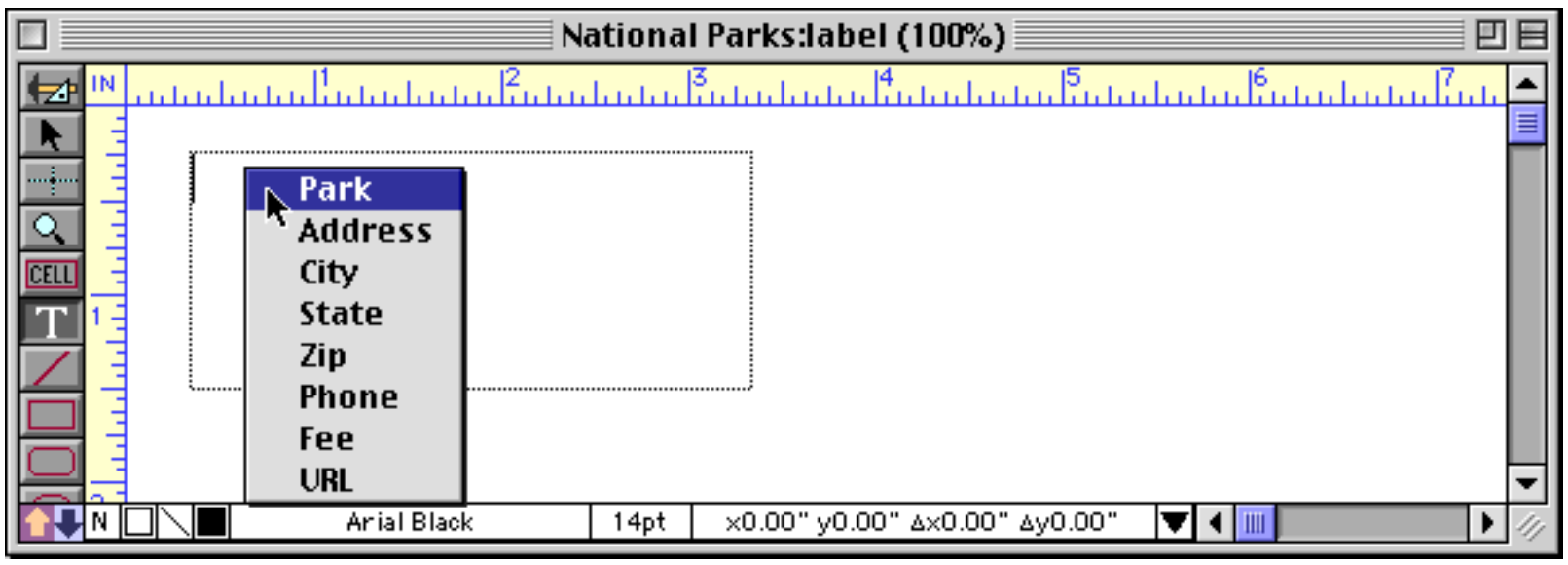


Using Data Merge to Create Address Labels

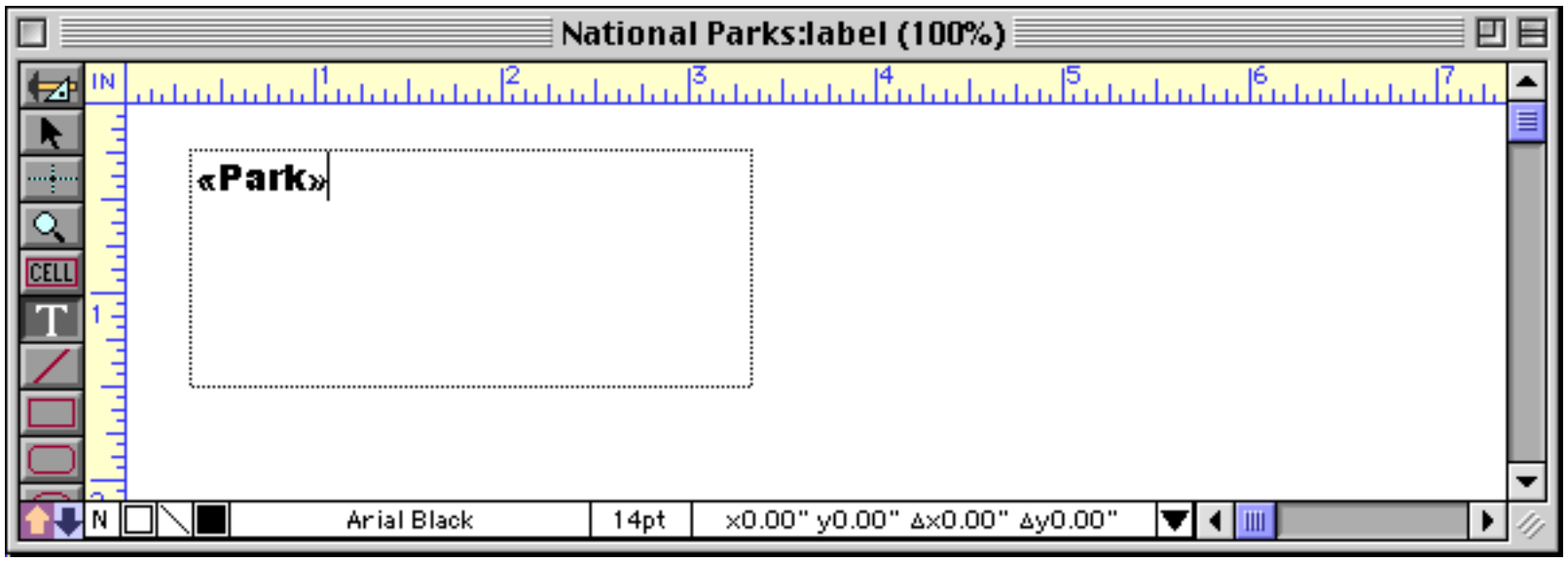
Data merge is an excellent way to create address labels. To create an address label using data merge, start by creating an auto-wrap text object the size of the label.



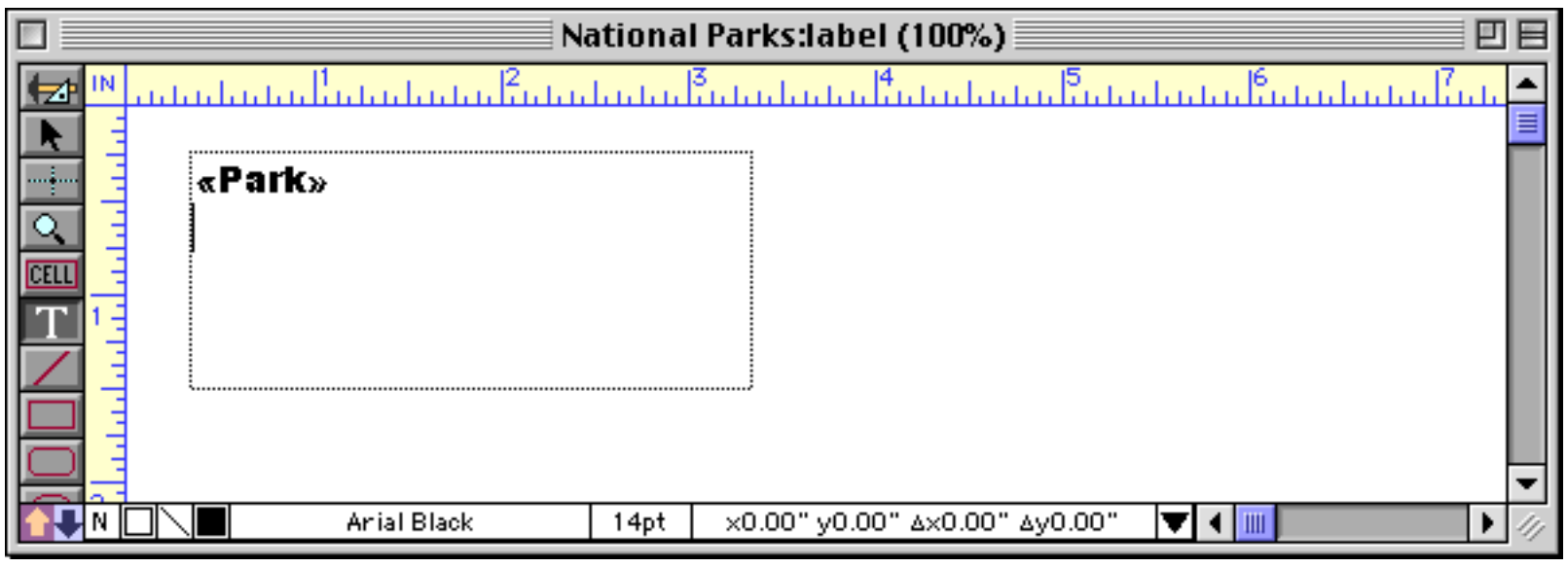
Hold down the **Command** key (Mac) or **Control** Key (Windows) and select the first field name from the pop-up menu.



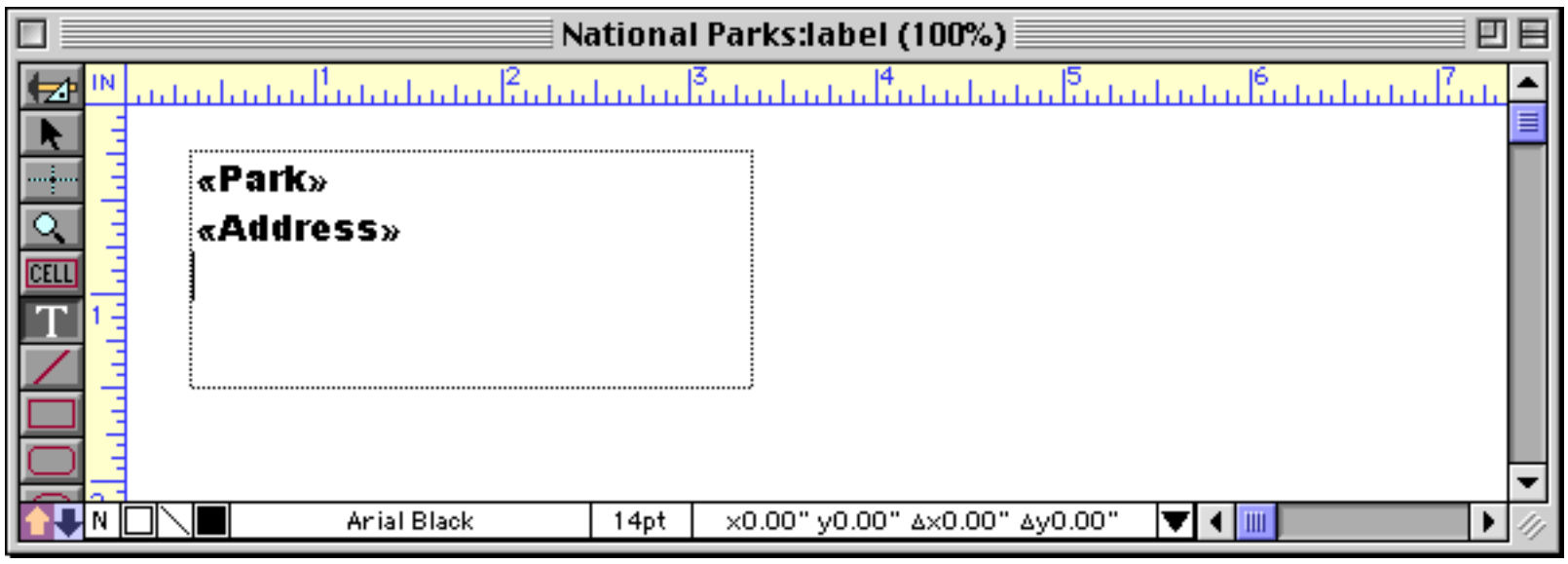
When you release the mouse button Panorama will insert the field name. The insertion point is at the end of the line.



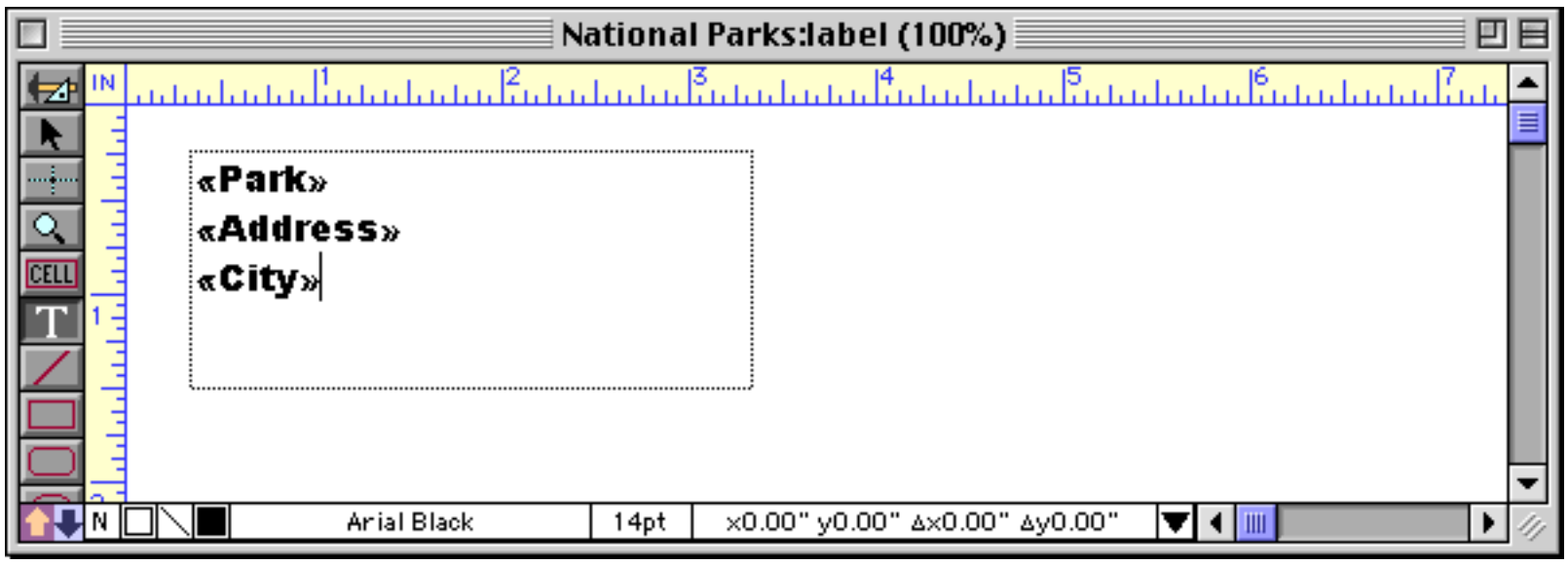
The first line is complete, so press **Return** to advance to the second line.



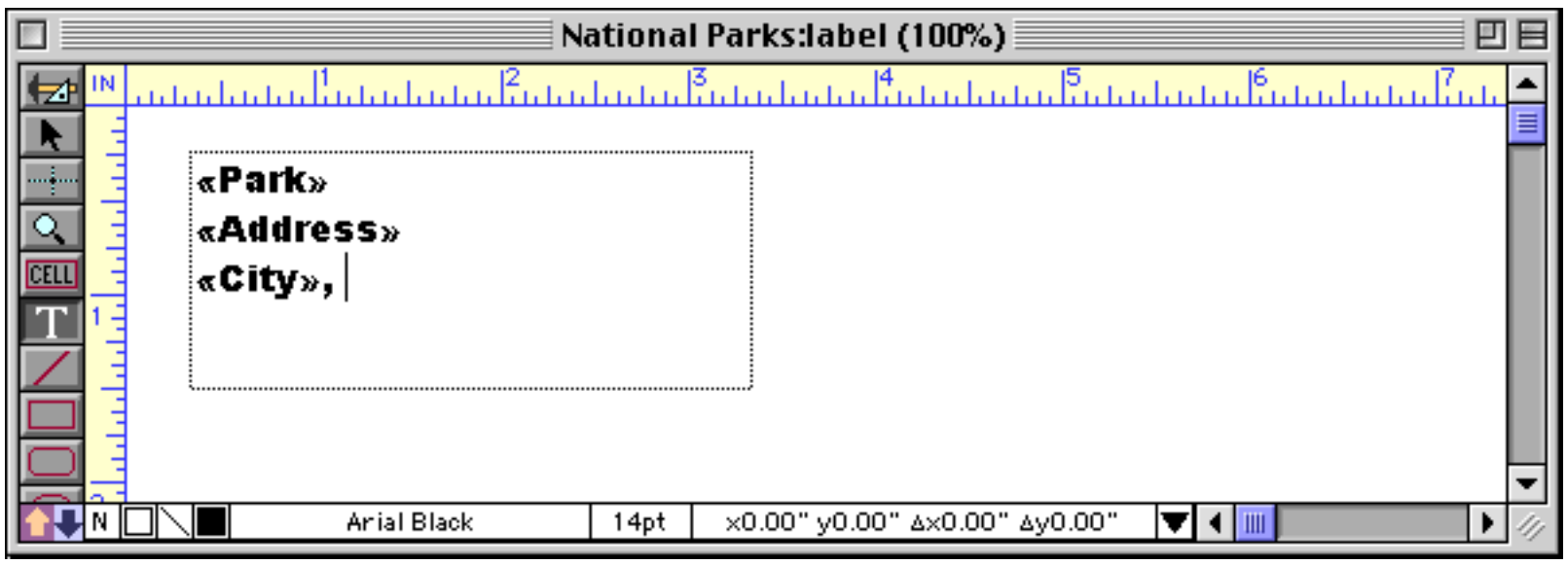
Repeat the same steps for the second line: hold down the **Command**/**Control** key, select **Address** from the pop-up menu, and press **Return**.



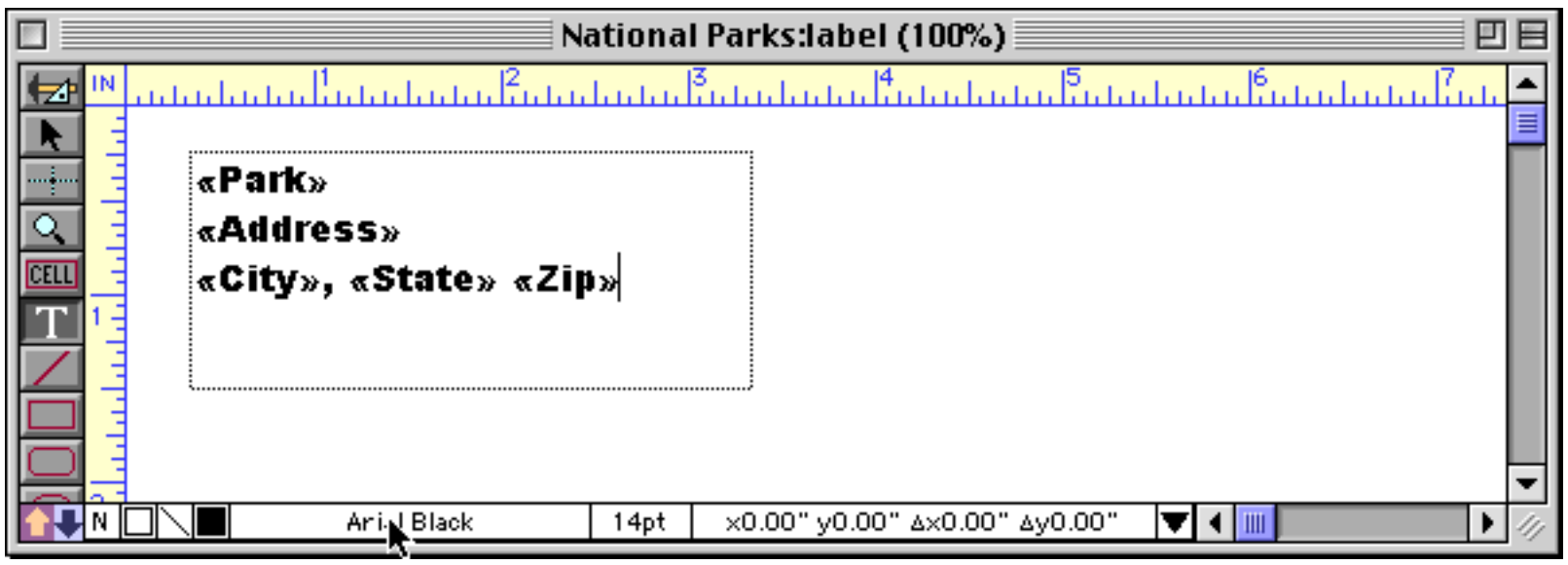
The third line contains three fields: City, State and Zip. Start by using the pop-up menu to enter the City field.



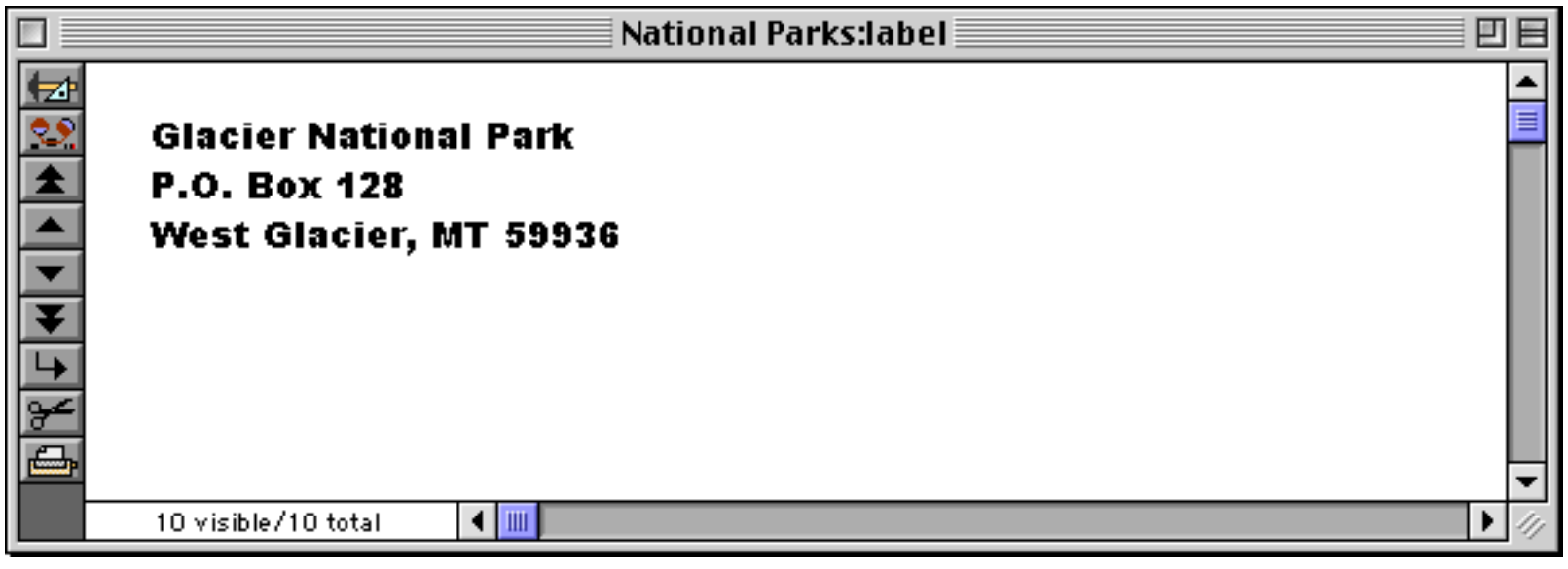
Now press the **Comma** and **Space Bar** keys.



Finish the label by inserting the State field using a pop-up menu, typing a **Space** and then inserting the Zip field.

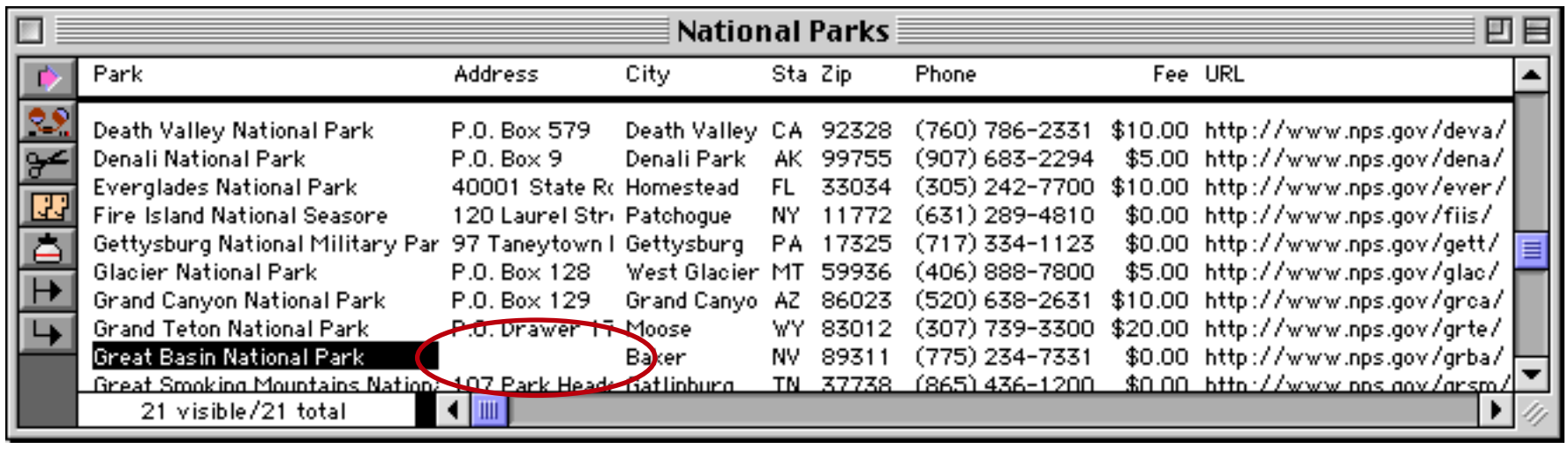


When you switch back to Data Access Mode Panorama will substitute the actual data. Voila! A label!

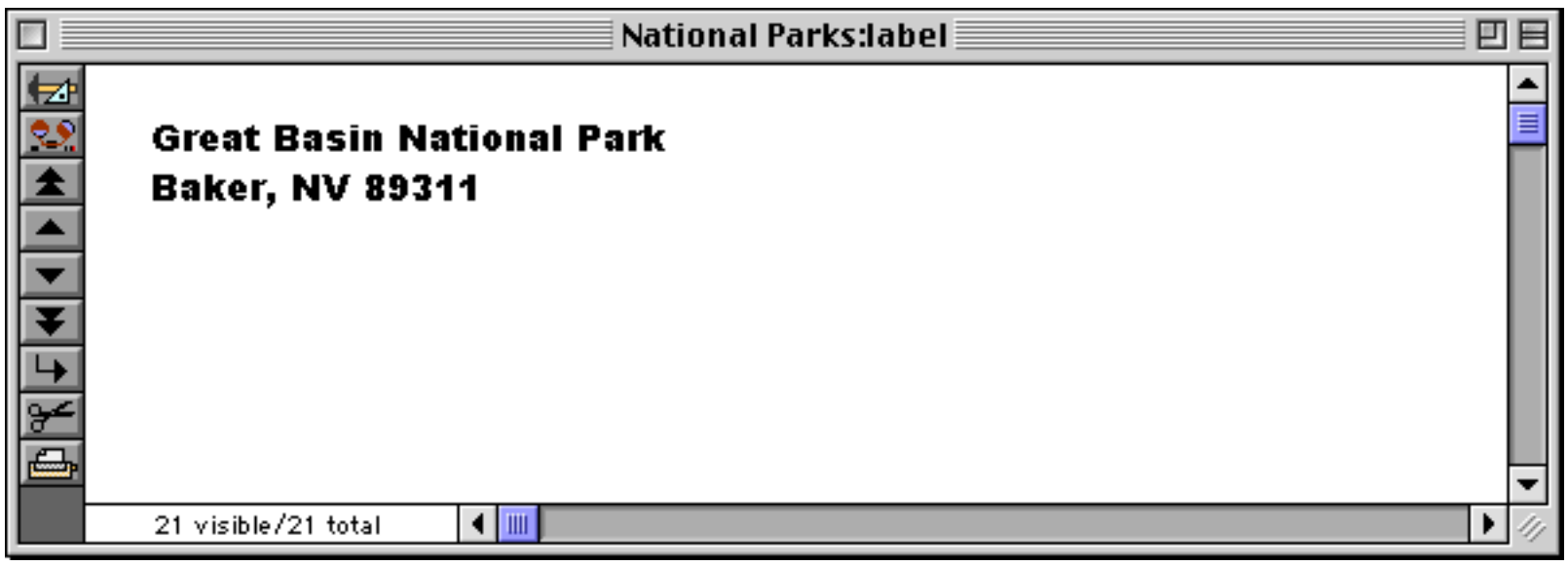


Panorama will automatically wrap the text within the rectangle you provided. It will start a new line wherever you have typed a **Return** (or when a line becomes too long to fit).

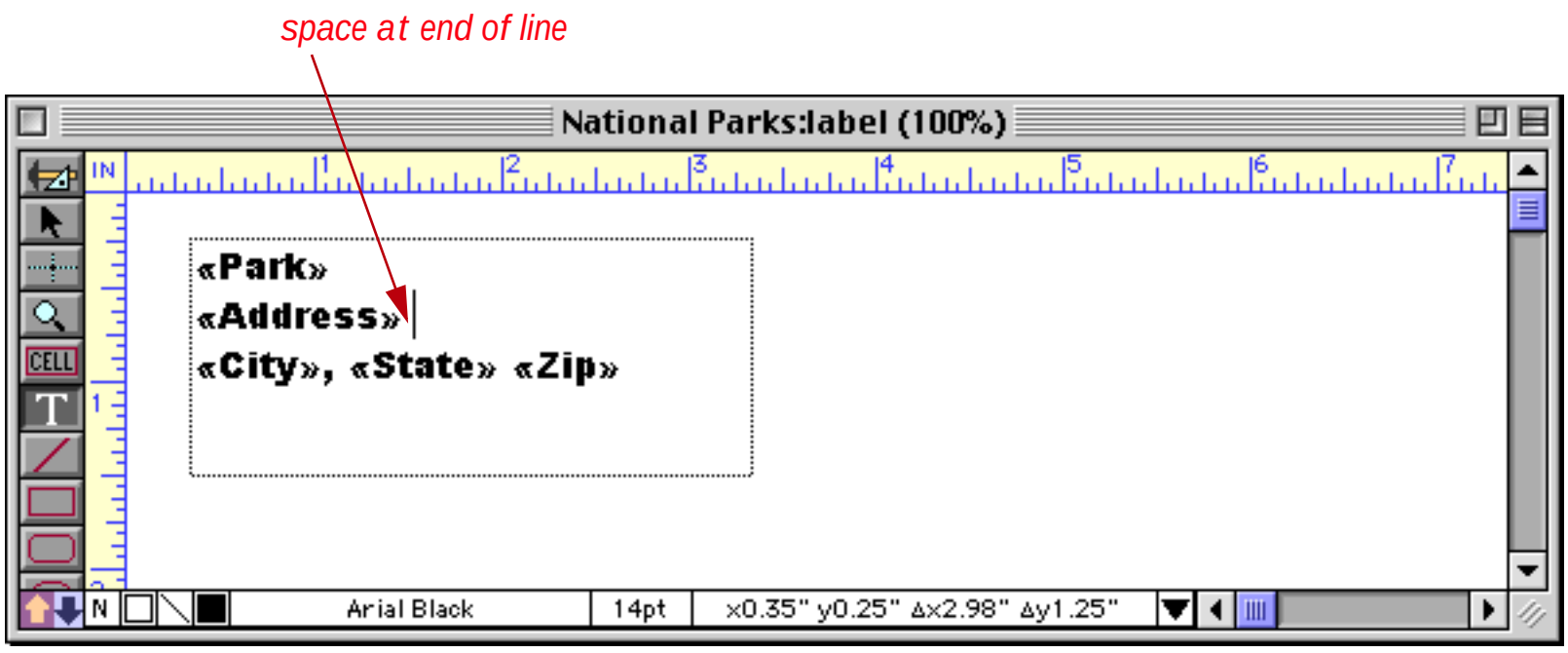
If a field is empty and that causes the entire line to be empty, Panorama will completely remove the line. For example, Great Basin National Park doesn't have a street or P.O. Box, as you can see in the data sheet.



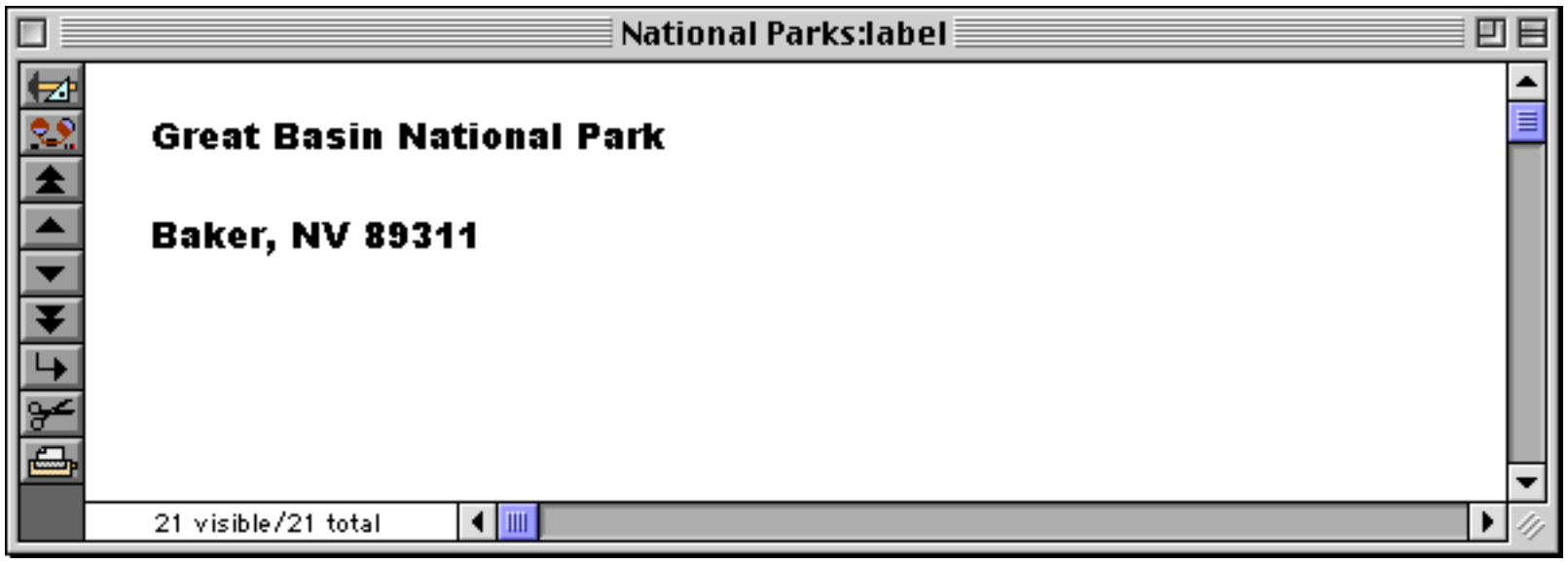
In the label, Panorama will remove the completely blank line.



If you don't want the blank line removed, put a space at the end of the line.



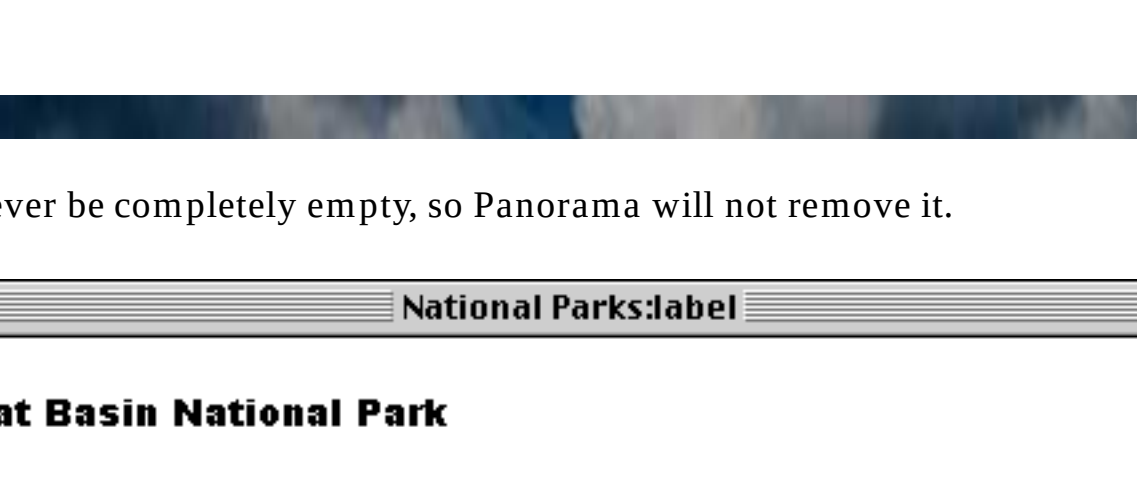
Now the line can never be completely empty, so Panorama will not remove it.



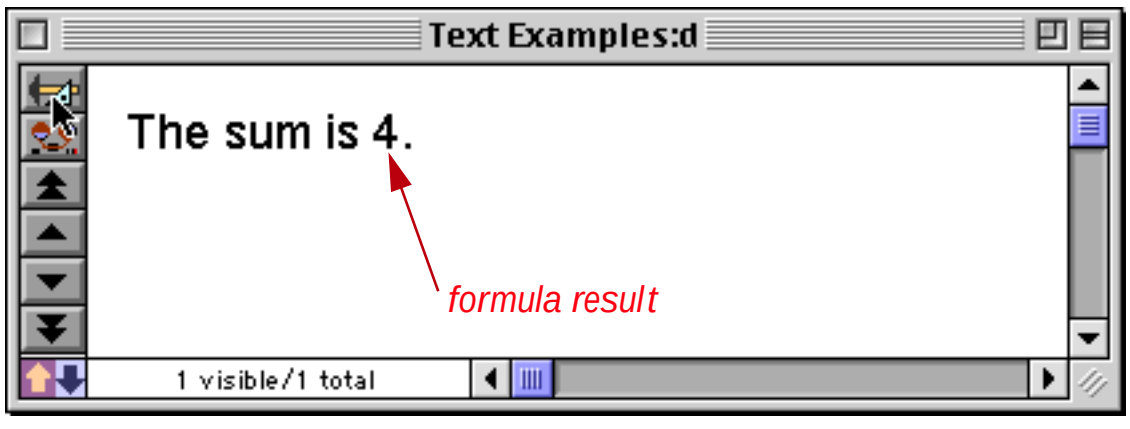
An address label can be used as part of a larger form (like an invoice), or it can be used by itself as a mailing label. If you wish to print a mailing label you must define the overall size of the label by creating one or more report tiles. Report tiles tell Panorama how to print a form. For more information on creating and printing mailing labels see “Label Fundamentals” on page 1157.

Displaying Formulas in Auto-Wrap Text

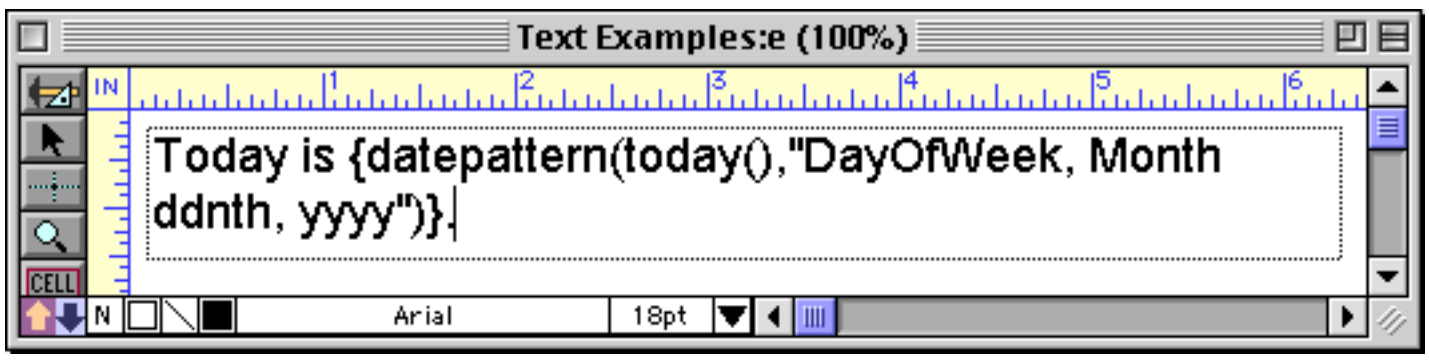
In addition to fixed text and fields, auto-wrap text can also contain complete formulas with text and numeric calculations. Simply type the formula into the text, surrounded by { and } curly brace characters. Here’s what a formula looks like in Graphics Mode.



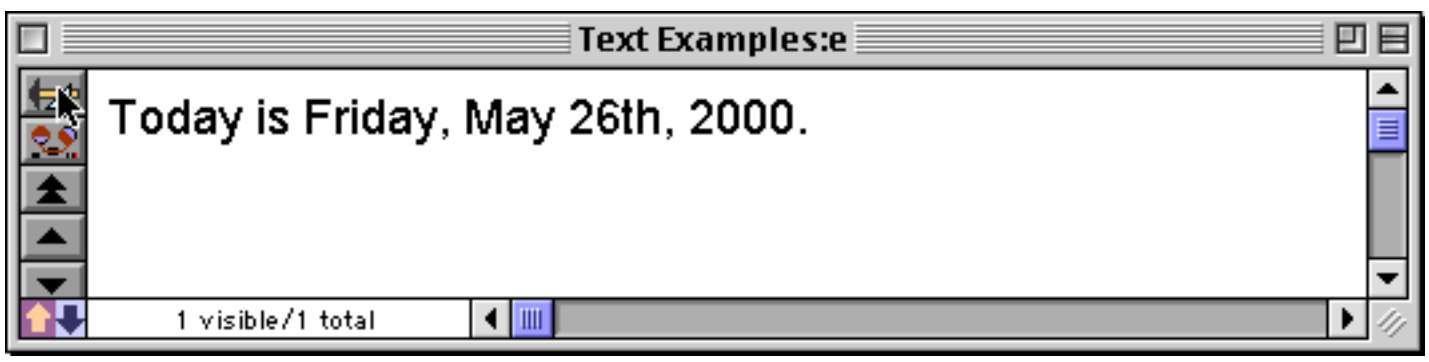
When the form is displayed in Data Mode Panorama substitutes the result of the formula instead of the formula itself.



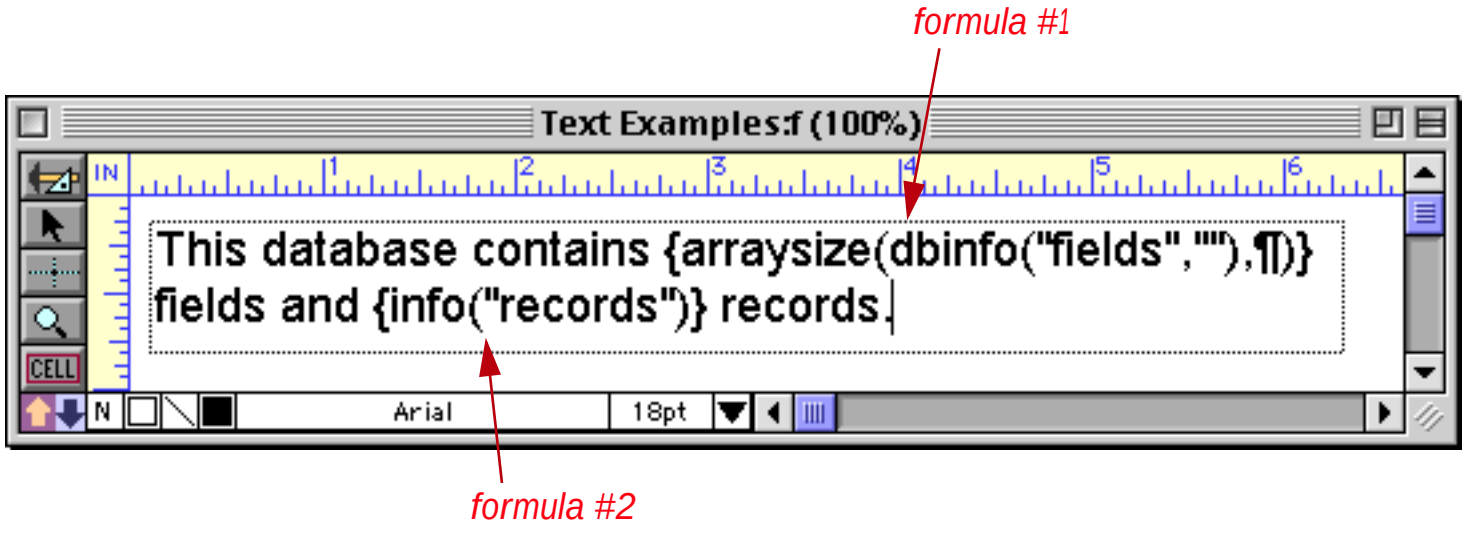
Of course 2+2 is a pretty silly formula. More useful formulas can be formed by combining fields, variables, and functions, like this.



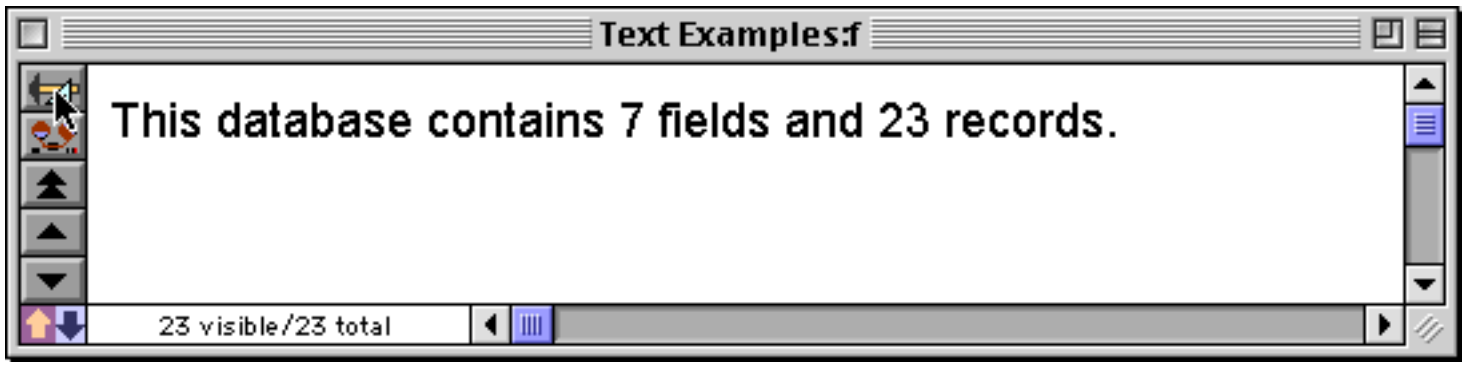
In Data Mode this auto-wrap text object looks like this:



An auto-wrap text object is not limited to a single formula. You can include as many formulas as you need.



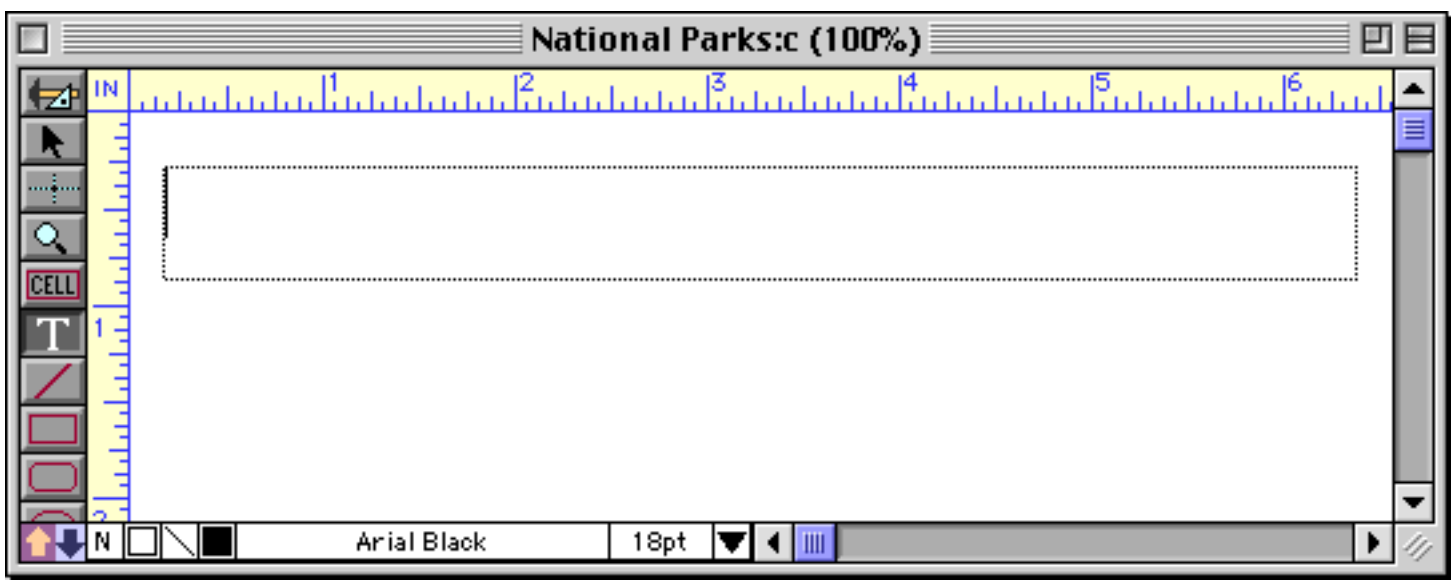
Here is the same auto-wrap text object in Data Mode. The formulas have been replaced with their results.



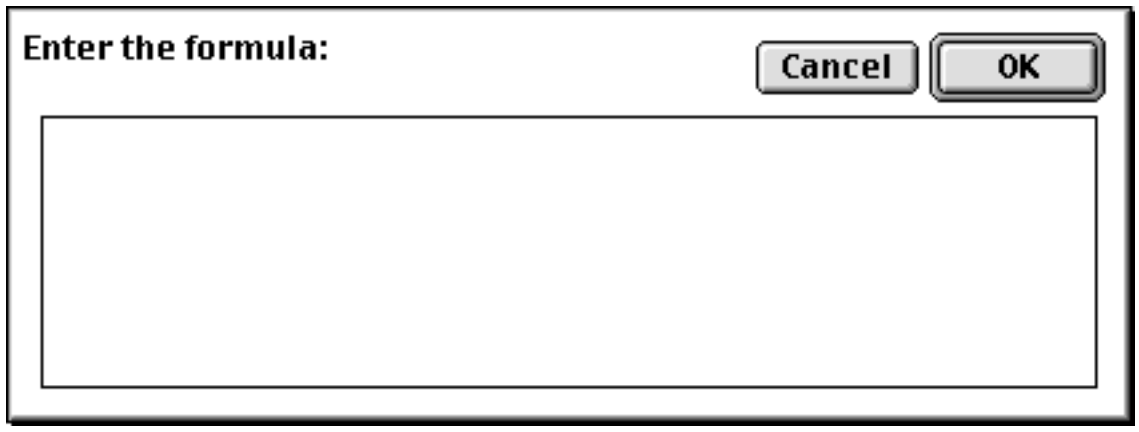
Using a formula gives you almost unlimited possibilities for combining and manipulating data on the fly as it is displayed or printed. By using formulas containing the lookup function you can display or print data from more than one database at once. You can use a formula to display or print computed information that is not stored in the database. You can use true-false formulas to display or print data only if a certain condition is met. The possibilities are almost endless. See “Formulas” on page 1165.

The Build Formula Dialog

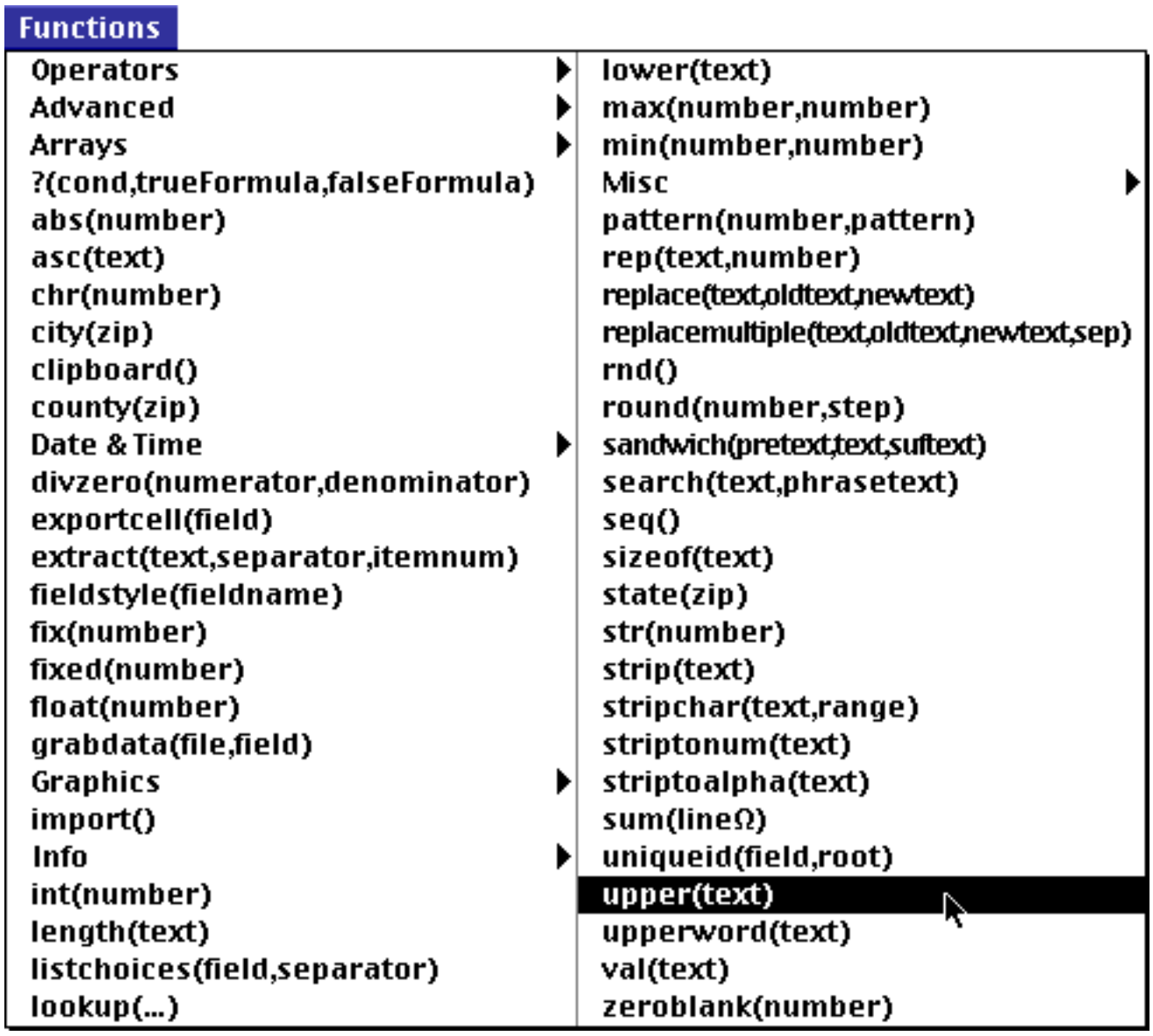
You can use the **Build Formula** dialog box to help you create formulas to merge into auto-wrap text. To create a new formula with this dialog, click on the **Text** tool, then click in the text to create an insertion point.



Now choose **Build Formula** from the Text menu. This causes a dialog to appear.



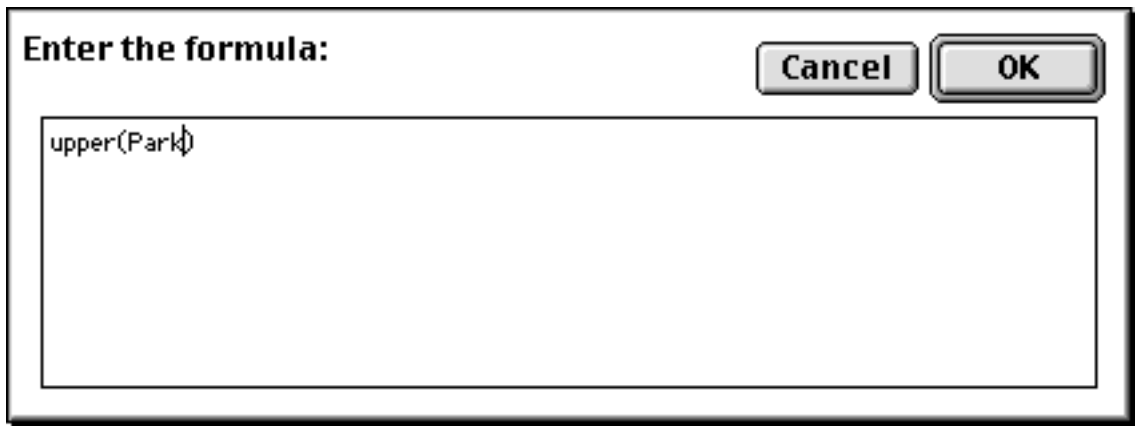
Once the **Build Formula** dialog is open, you can type in a formula using the keyboard. Or you can use the Field and Function menus to help enter the formula for you. The **Field** menu contains a list of all fields in the database; pick from this menu to type a field name into the formula. The **Function** menu contains a list of all the functions and operators available; pick from this menu to type a function or operator into the formula. For example, if you wanted to convert text to upper case, you would select the `upper(` function from the **Function** menu.



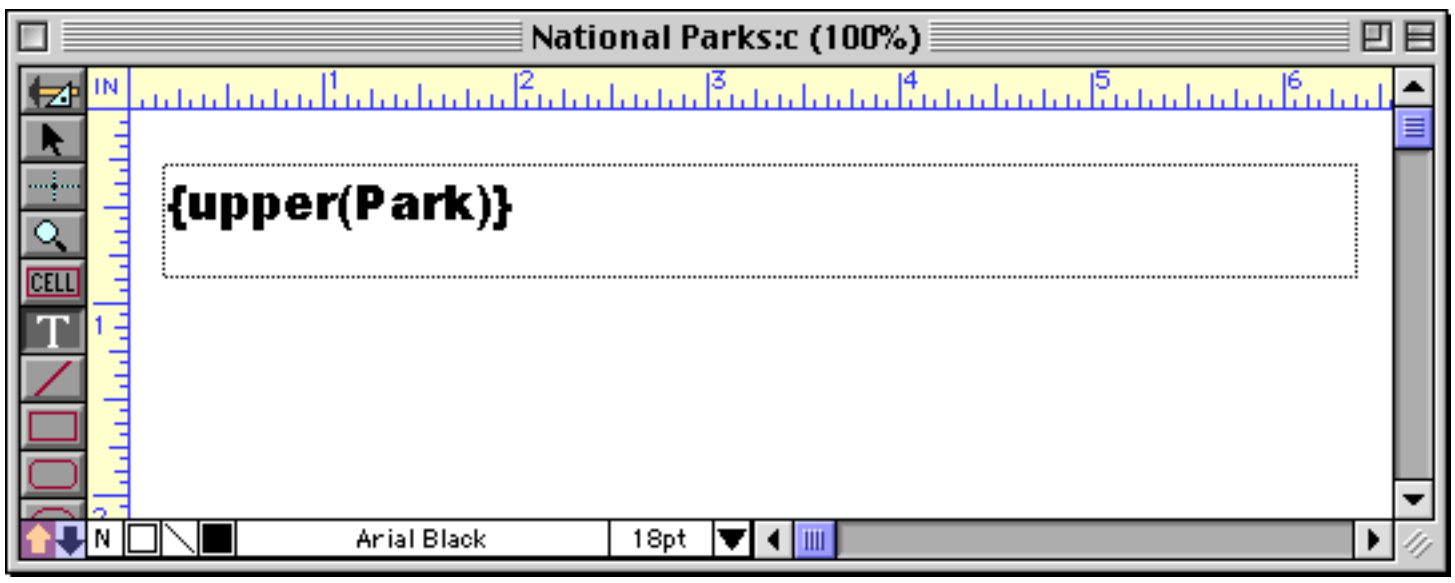
If you need to include a field name in your formula, select it from the **Field** menu.



As you select items from these menus, Panorama will insert them into the formula. (Of course you can also type text into the formula.)

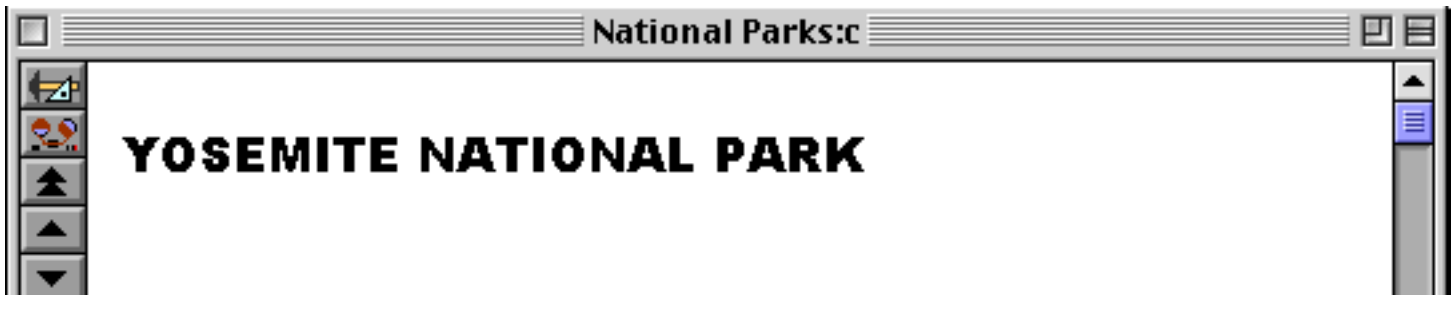


When the formula is finished, press the **OK** button. Panorama will check the formula for errors. If the formula is correct, it will be inserted into the auto-wrap text (with the required { and } curly braces automatically added).

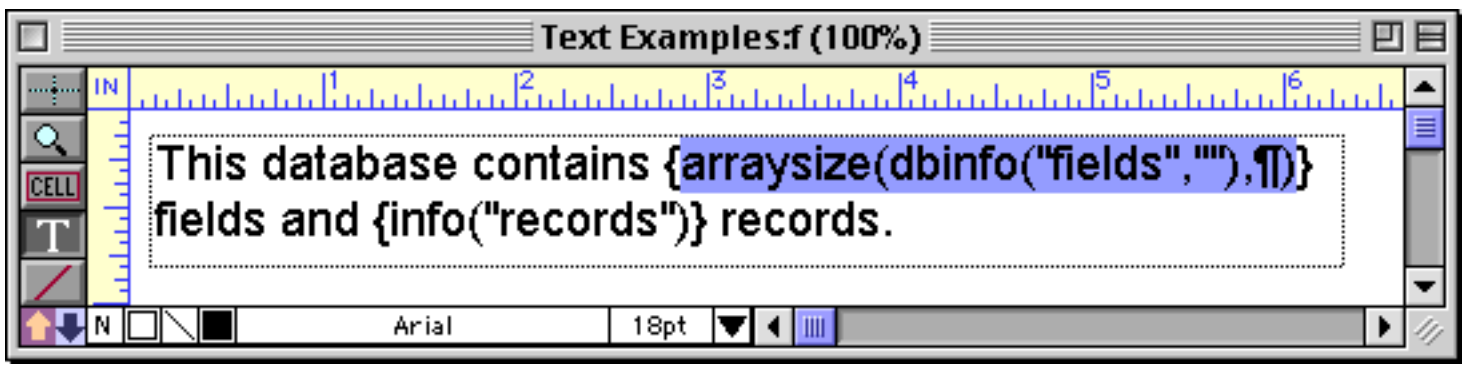


If the formula contains an error, Panorama will display an alert and highlight the location of the error. You won't be able to close the dialog until you correct the error (unless you press **Cancel**).

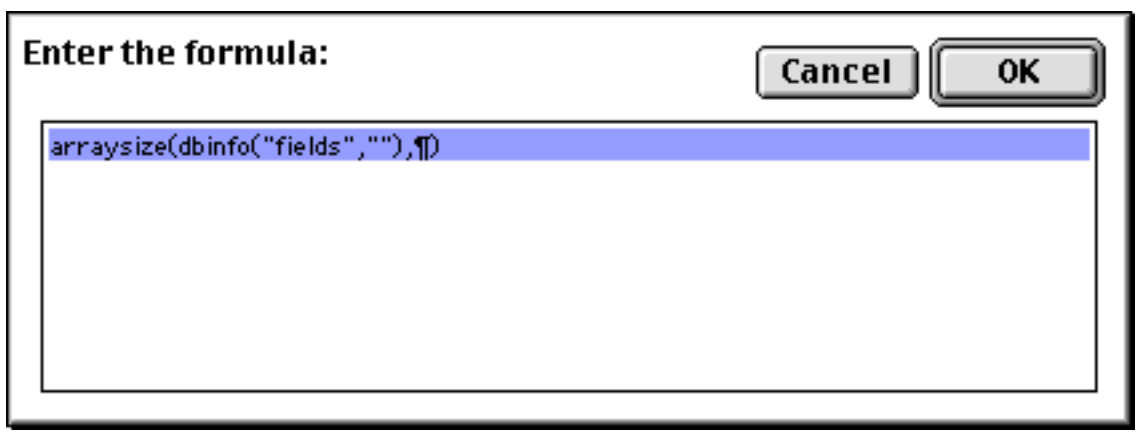
By the way, here's what our finished formula looks like in Data Mode. Notice that the Park name has been converted to all upper case (YOSEMITE NATIONAL PARK instead of Yosemite National Park).



You can also use the **Build Formula** dialog to edit a formula you have created earlier. To do this, start by dragging over the text of the formula to select it.



Once the text is selected, choose **Build Formula**.



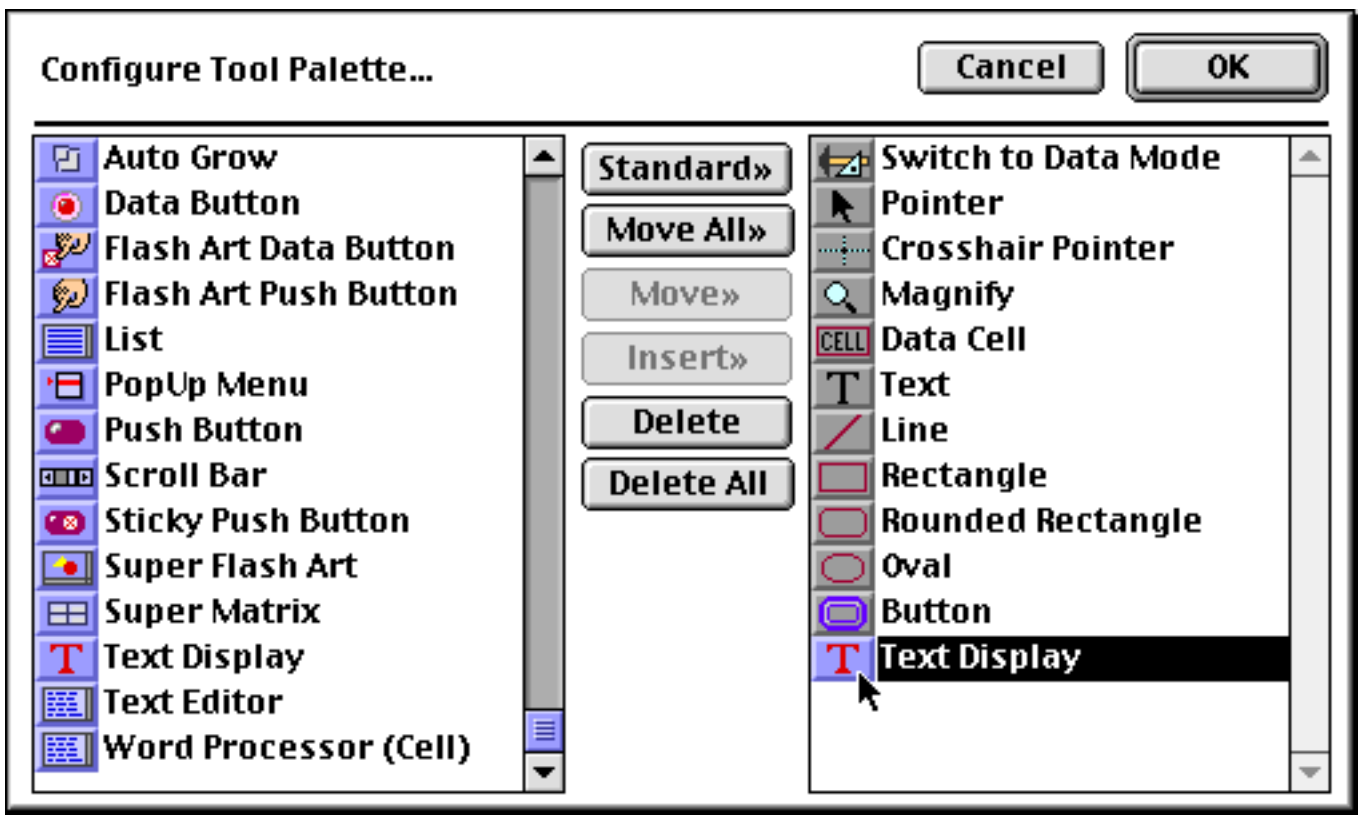
Now you can edit the formula by typing or choosing from the **Field** and **Function** menus, just like when you created a new formula.

Text Display SuperObjects™

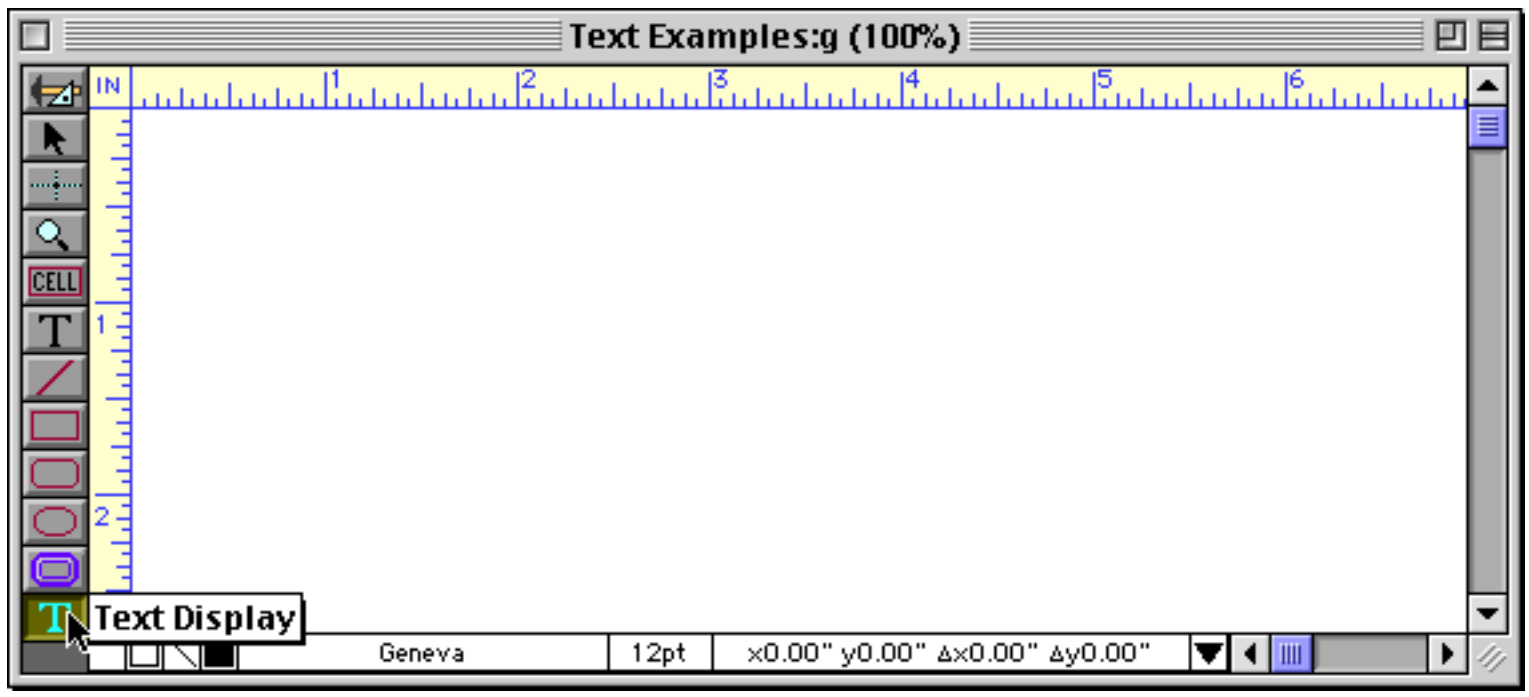
The **Text Display SuperObject** displays text based on a formula. In some ways, this is similar to an auto-wrap text, but there are many more options for calculating the formula and formatting the displayed text. You can store the formula itself in a variable (so it can be changed on the fly), align the text in any corner of the object, automatically scale the text for different size windows, and even change the color of the text on the fly.

Creating and Modifying Text Display SuperObjects

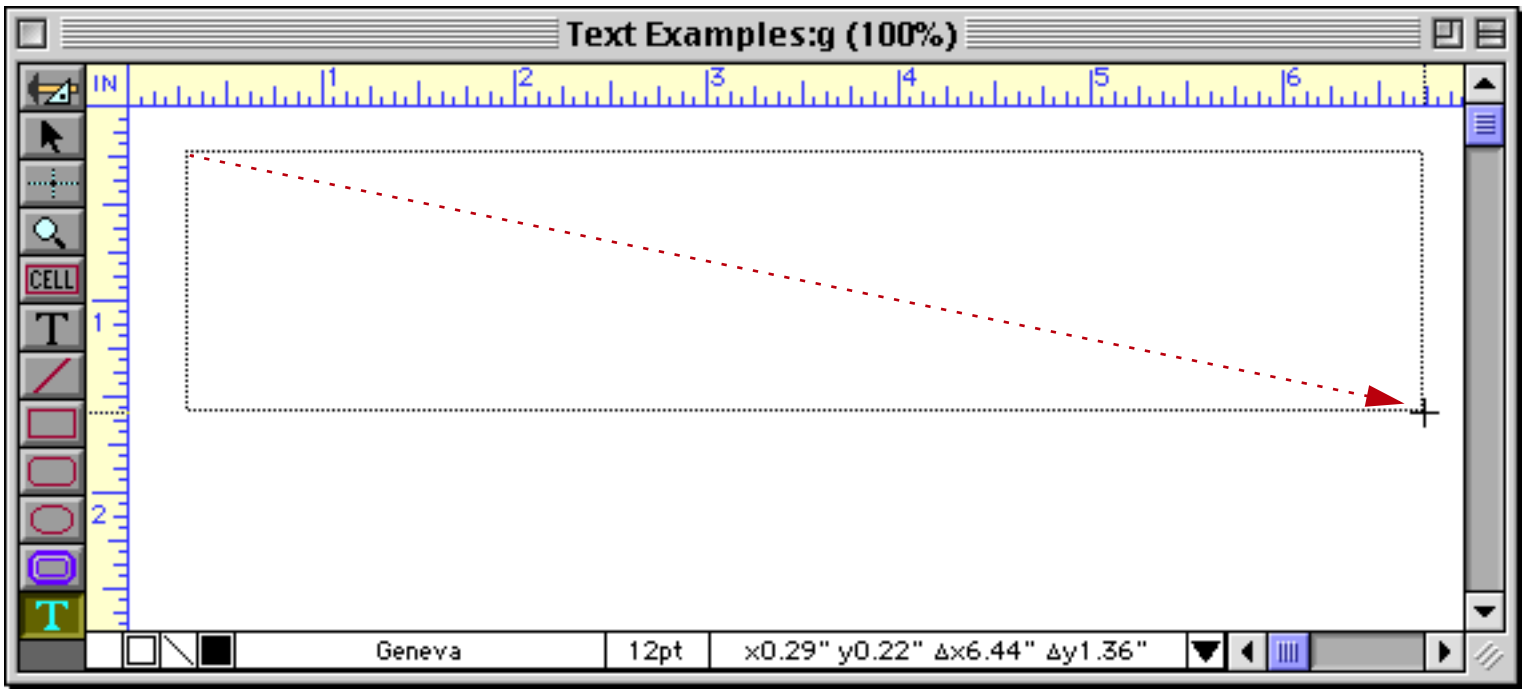
The Text Display SuperObject tool is not in the default tool palette, so you'll need to use the **Tool Palette** dialog to add this tool to the palette if it is not already there (see "Customizing the Tool Palette" on page 497).



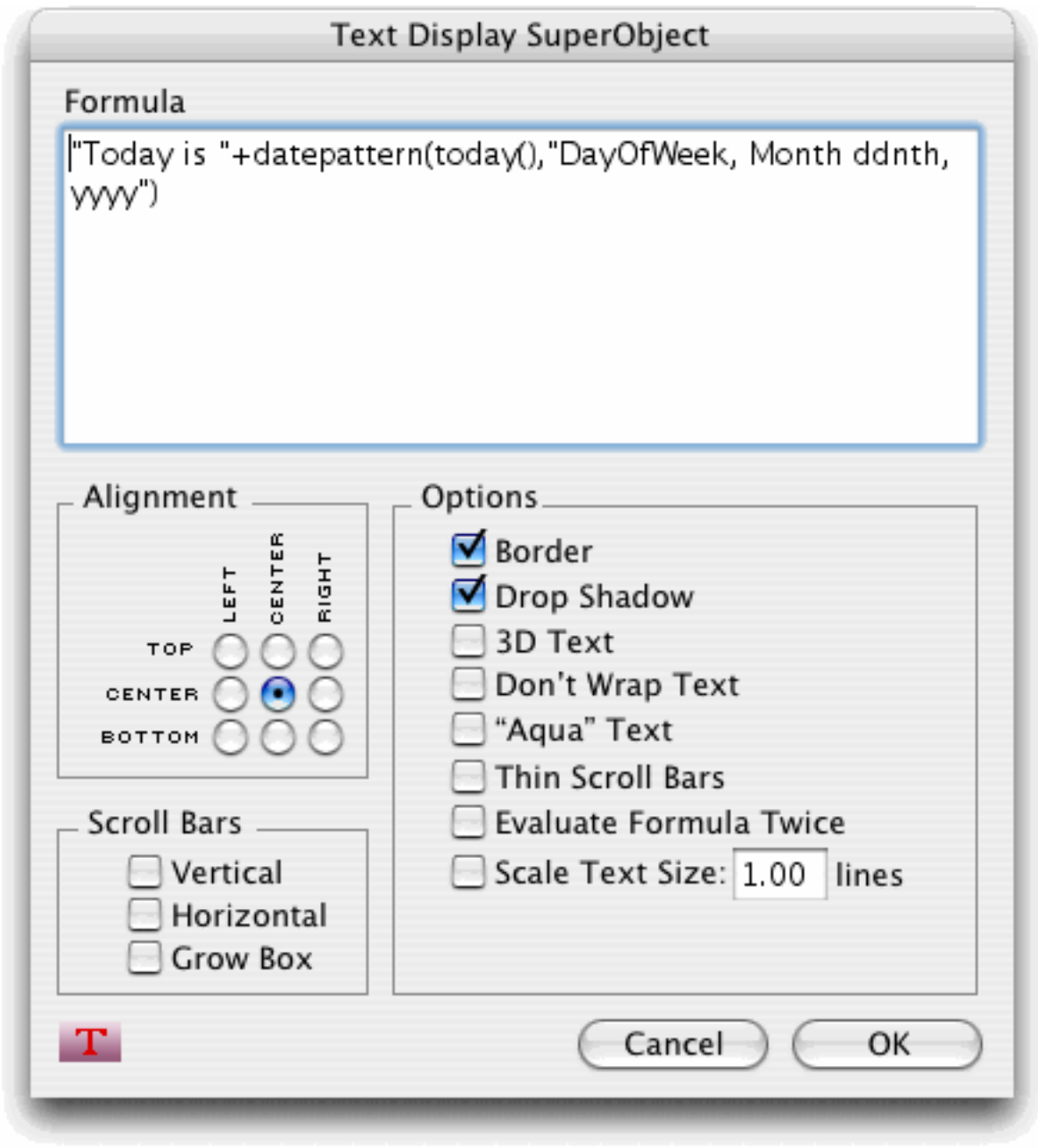
Now that the tool is added to the palette you can select it.



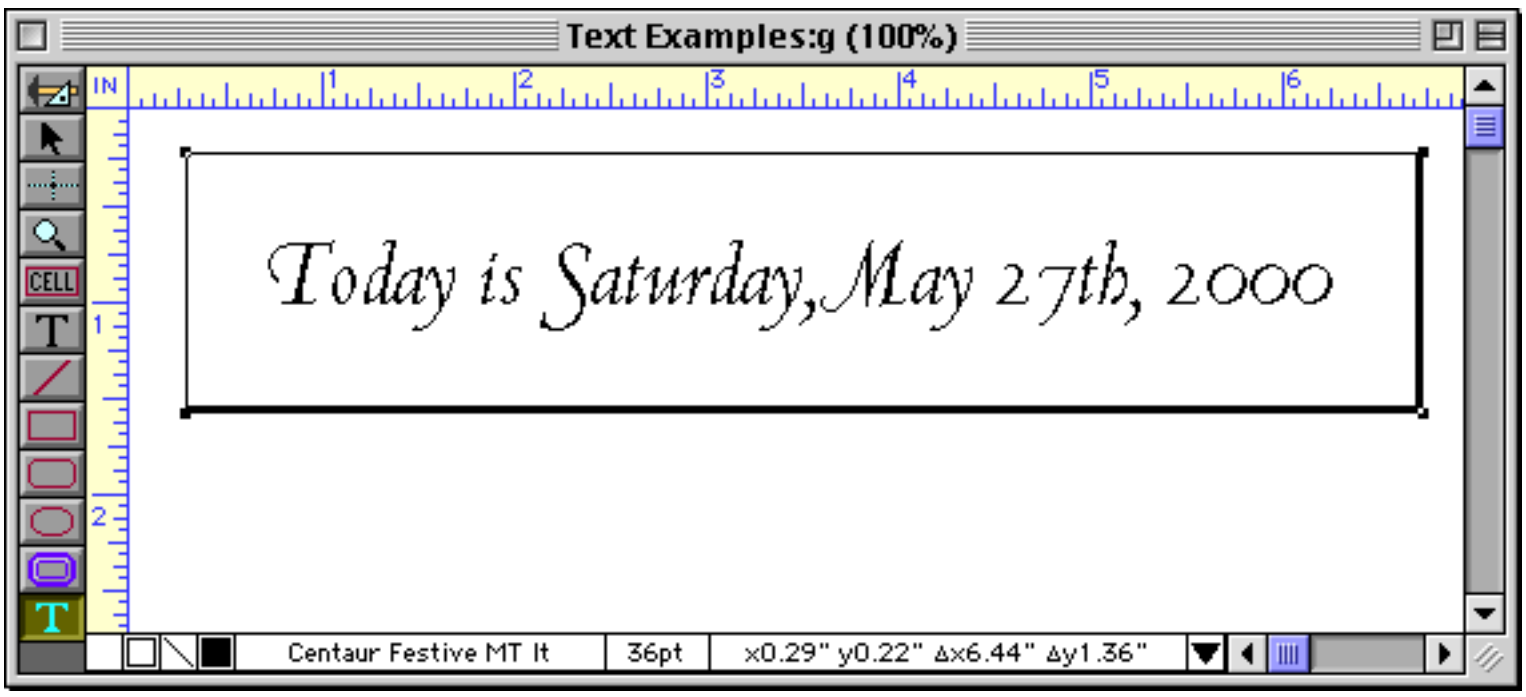
Once the tool is selected, drag the mouse across the form in the location where you want to create the Text Display SuperObject.



When you release the mouse, the Text Display SuperObject configuration dialog will appear.



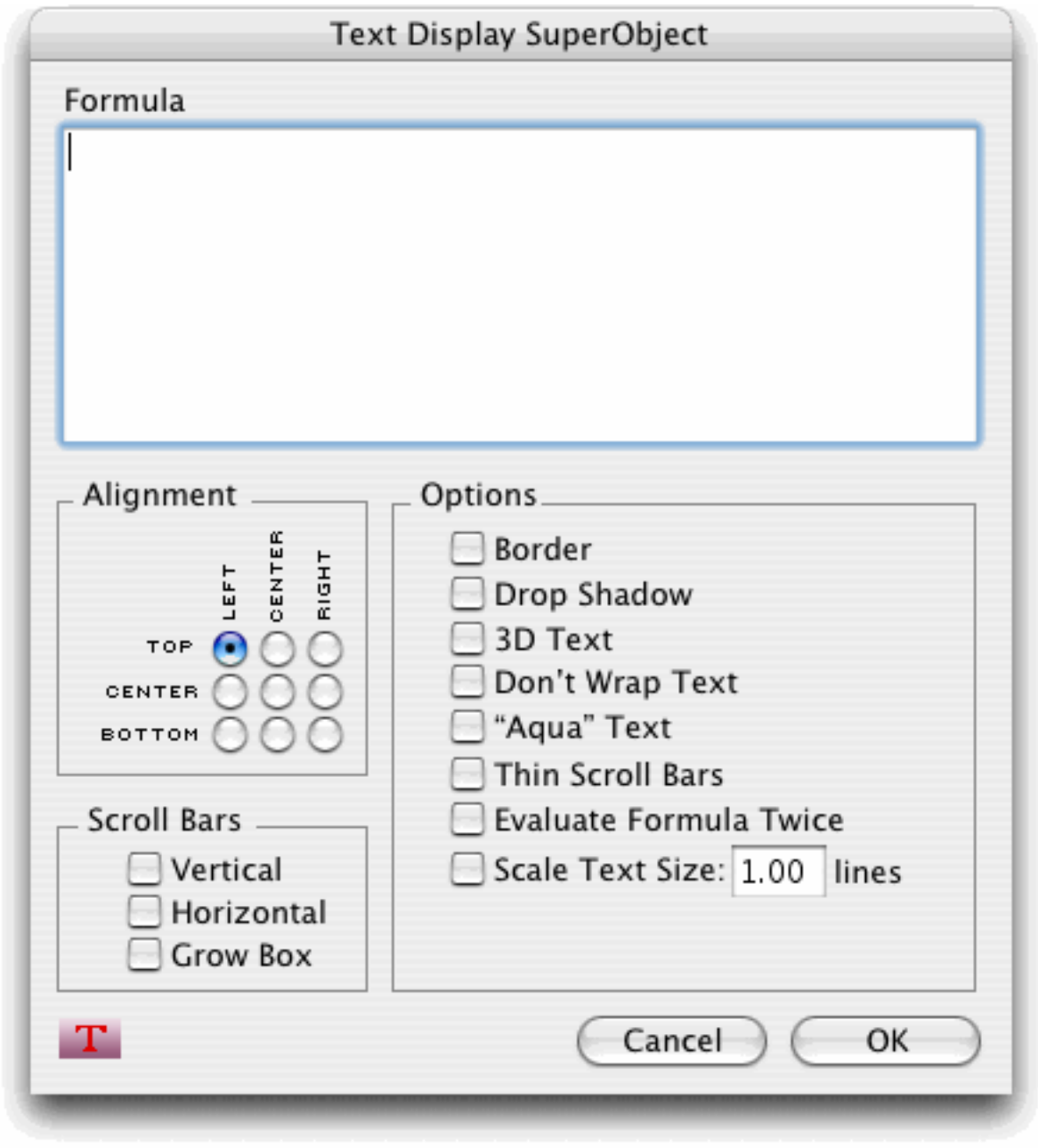
At a minimum you must enter a valid formula into the dialog. For this example we've also turned on the Border and Drop Shadow options and set the alignment to centered (these options are discussed in detail below). When the OK button is pressed the new object appears. (Notice that unlike the auto-wrap text object, the Text Display SuperObject shows the result of the formula in both Graphics Mode and Data Mode, not just Data Mode.)



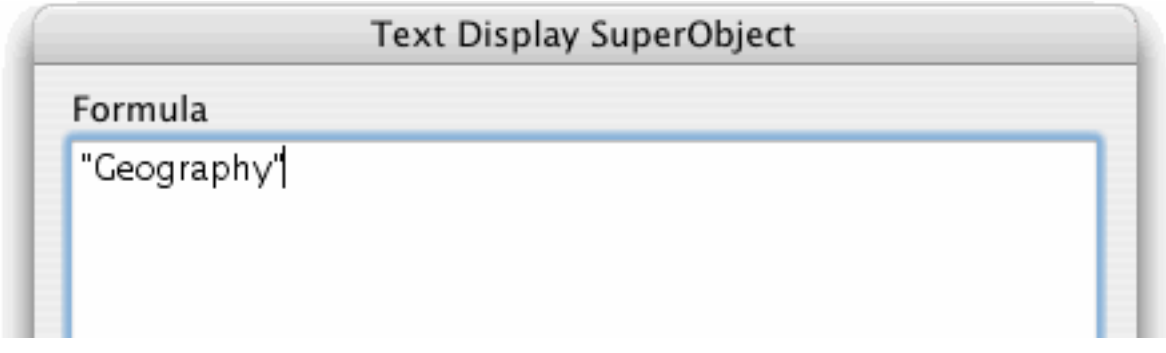
After it has been created you can modify the location, size, font, style and color of a Text Display SuperObject just like any other object. To change any of the object attributes (formula, border, alignment etc.) select the **Pointer** tool and double click on the object. The configuration dialog will appear again. Make your changes and press the **OK** button.

Text Display Options

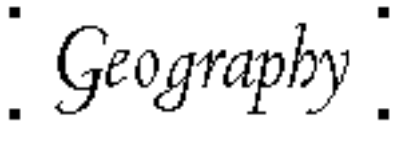
The **Text Display SuperObject** configuration dialog is divided into several sections.



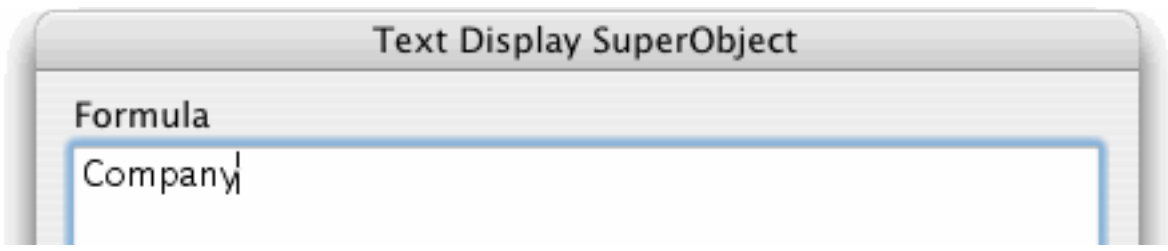
Formula: This section of the dialog specifies the formula for displaying text in this object. If you want to display fixed text, remember that you need to surround the text with quotes.



This object will always display the word Geography.



To display a field or variable, type in the name of the field or variable. You can use the **Field** menu to type in the name of a field for you (that way you don't have to worry about misspellings. Here's the formula to display the Company field.



And here is the finished object.

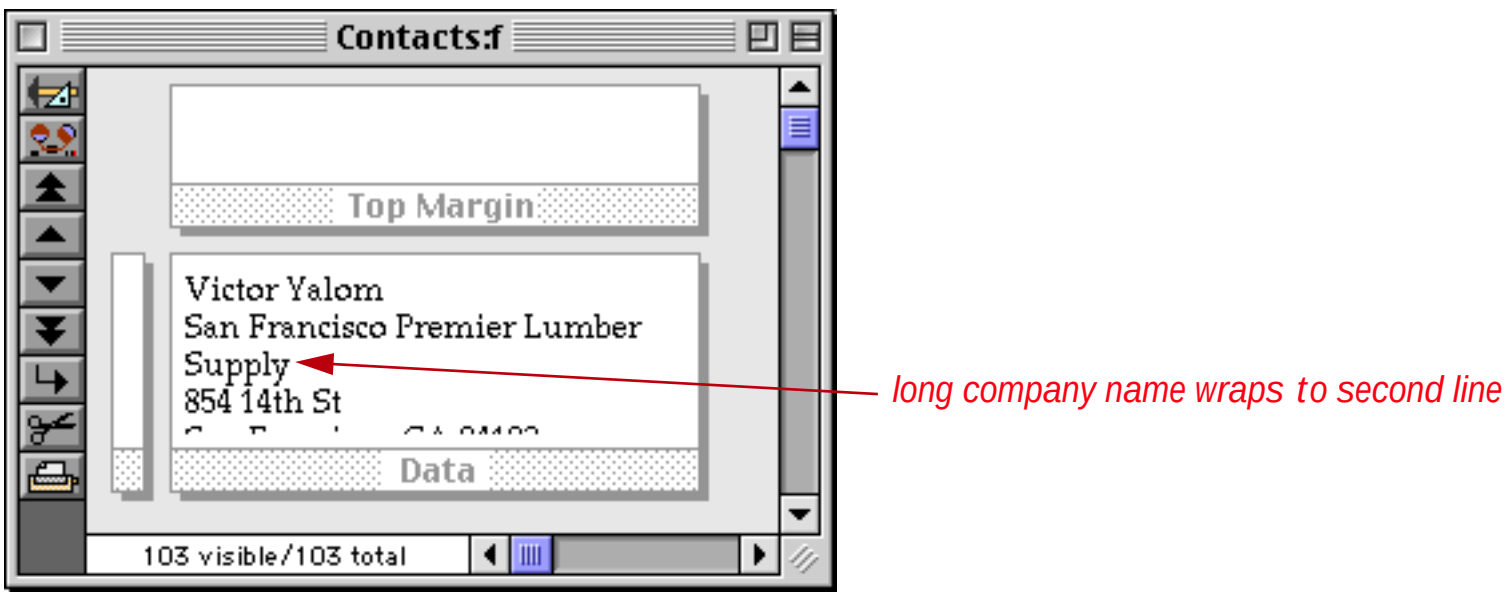
Austin Lumber

Panorama has hundreds of different functions that you can use to assemble your formula. See "Using Formulas to Display Text" on page 621 for some useful tips on building formulas for displaying text on a form. You'll find a complete description of formulas in "Formulas" on page 1165.

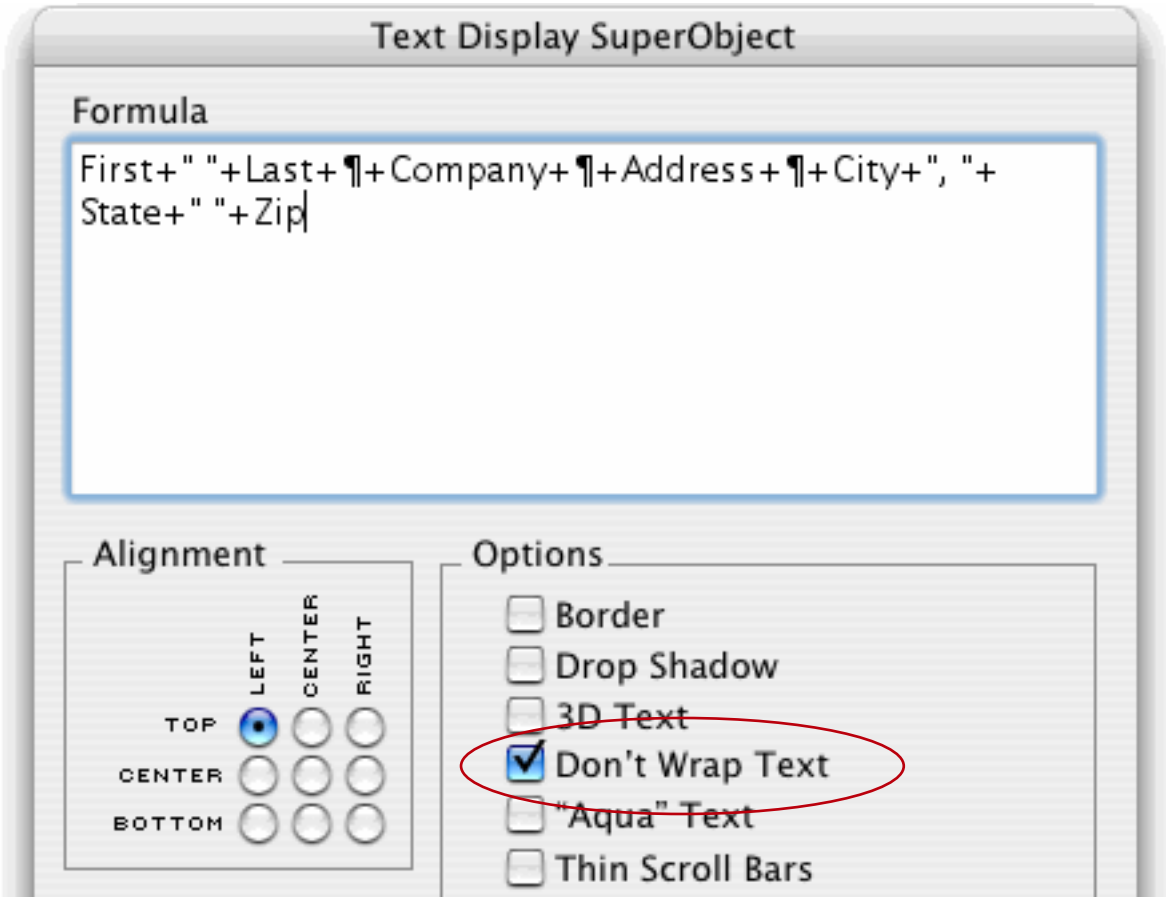
Evaluate Formula Twice: If this advanced option is enabled, Panorama will calculate the formula, then treat that result as a new formula and calculate it again to get the final result. The purpose of this feature is to allow you to store the "real" formula separately in a global or permanent variable (see "Variables" on page 1199 and "Variables" on page 1370 for more information about setting up and using variables).

Don't Wrap Text: If text is too long to fit on a single line, it will usually "wrap" around to the next line. However, if the Don't Wrap Text checkbox is turned on, the text will not wrap. Instead, the text will be cut off.

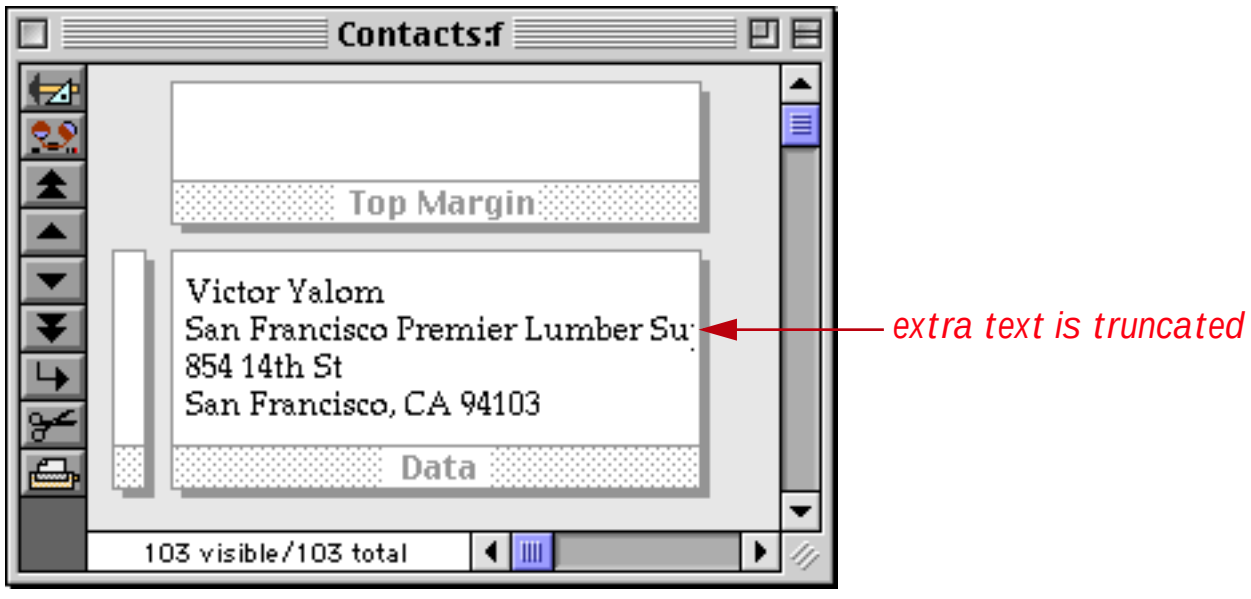
For example, suppose you are constructing a mailing label. With the Don't Wrap Text option turned off, a long company name will wrap to a second line, messing up the label.



To fix this problem, go into Graphics Mode (see "Form Modes: Data Access vs. Graphic Design" on page 485), double click on the Text Display object, and turn on the Don't Wrap Text option.



Now the long line will be cut off, and the remaining lines will stay in the correct position.



3D Text: If this option is turned on, a white shadow appears behind the text. This gives the text a "3D" effect if it appears over a colored background, like this.



Border: If this option is turned on, a one pixel border appears around the object, like this.



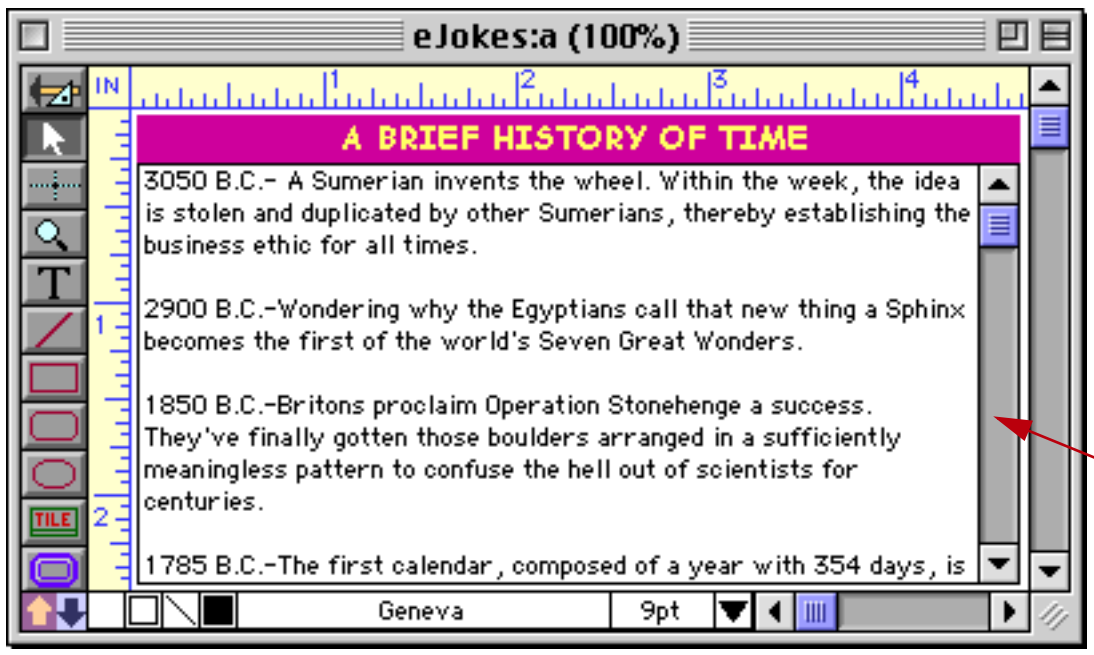
"Aqua" Text: If this option is turned on the text will be smoothed (anti-aliased) if the operating system supports that feature.



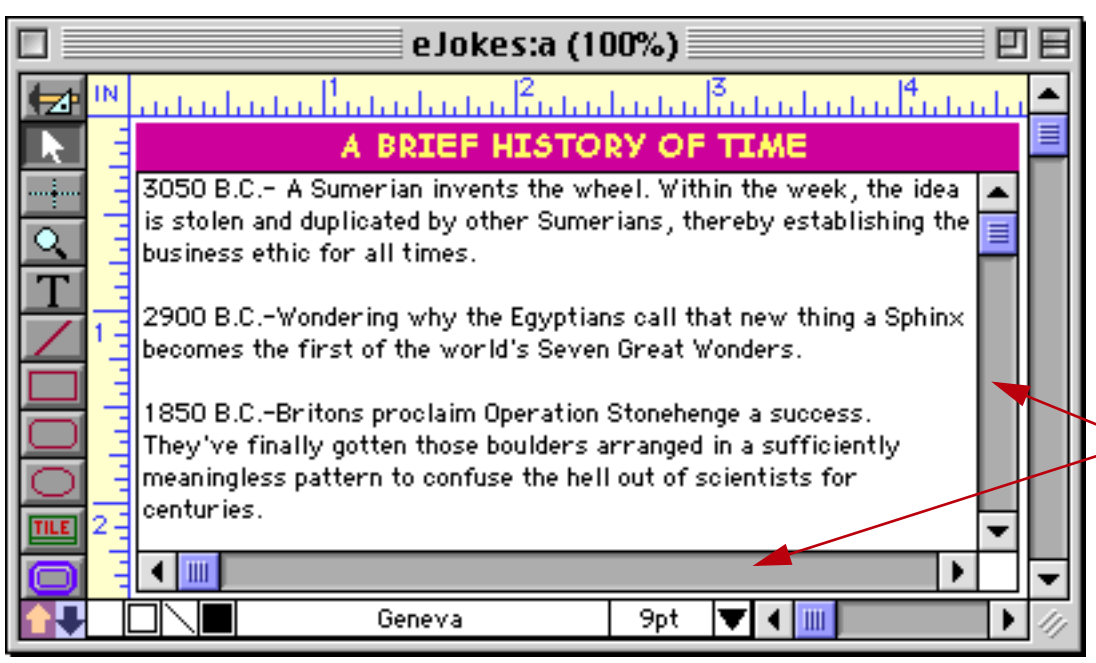
Drop Shadow: If this option is turned on, a black drop shadow appears "below" the object. This option is usually used in combination with the Border option, like this.



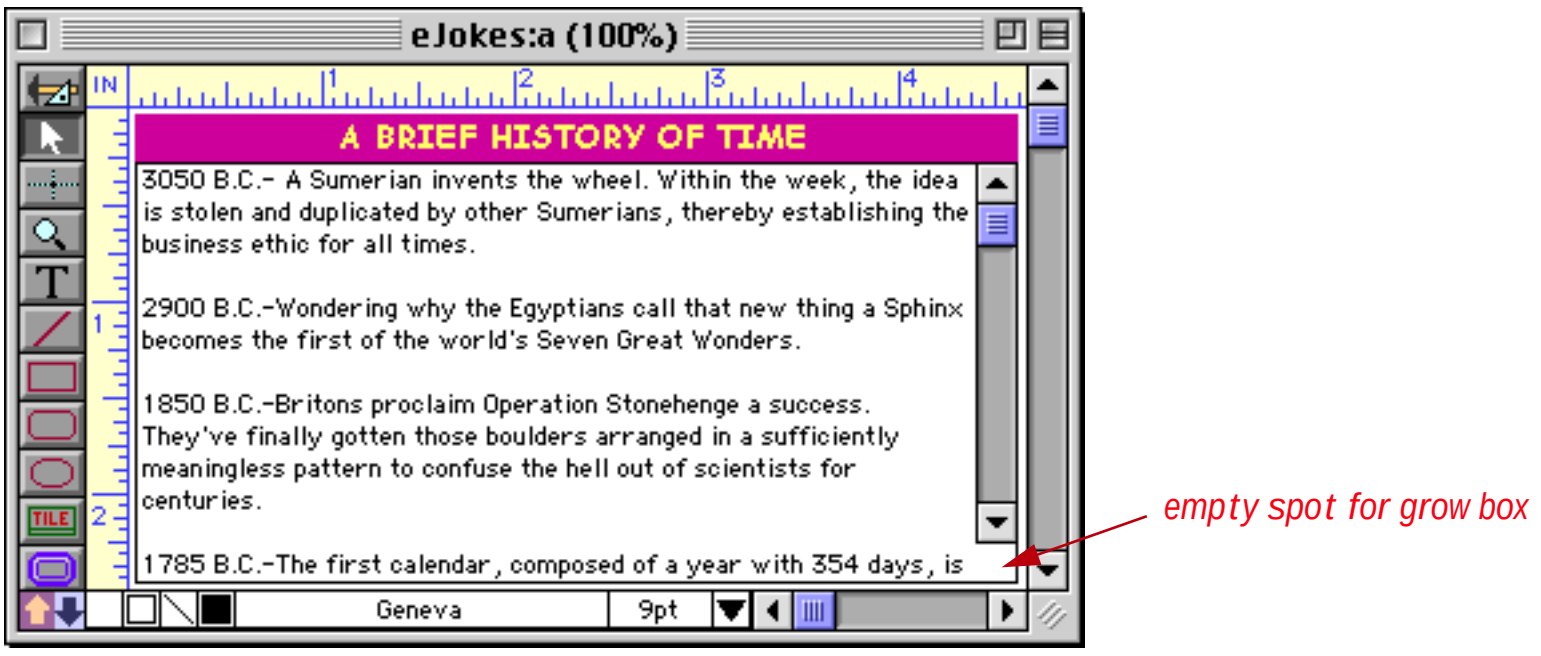
Scroll Bars: This section controls what scroll bars (if any) will be available when displaying this text, and whether space will be left for a grow box if only one scroll bar is used. Here is a Text Display SuperObject with the Vertical Scroll Bar enabled.



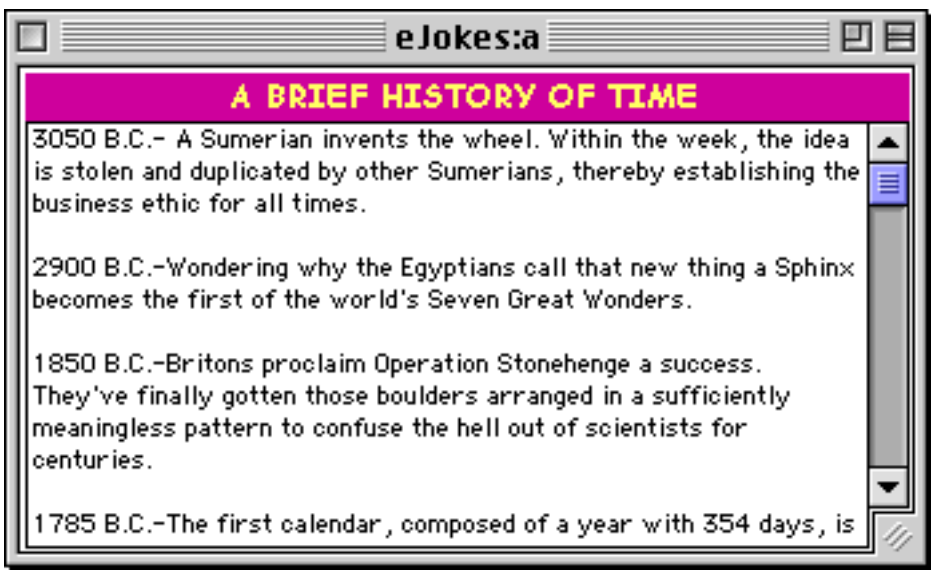
Here is the same object with both Vertical and Horizontal scroll bars.



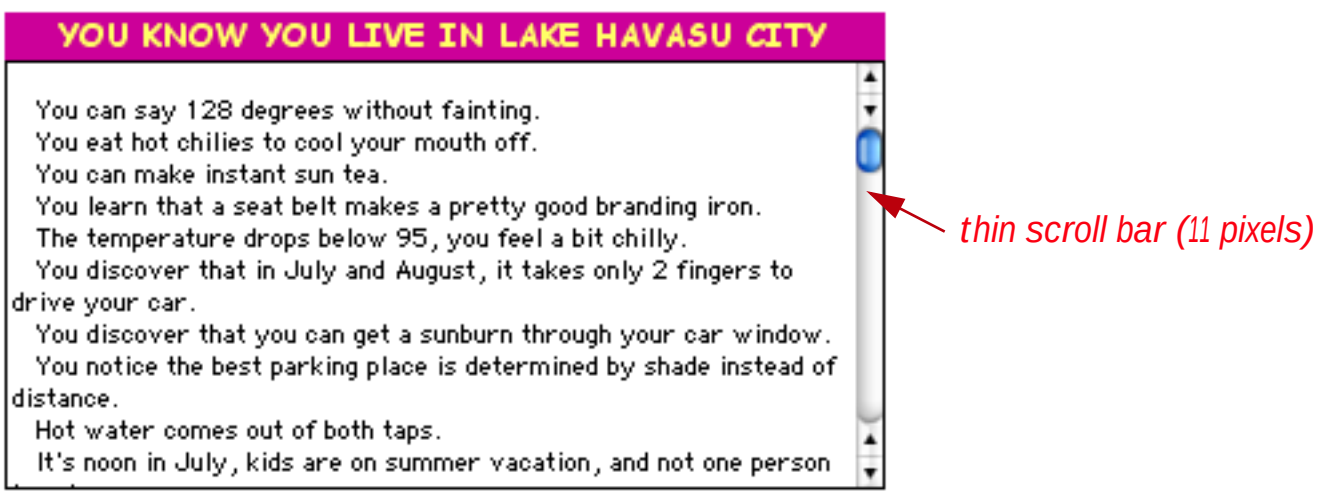
With the Vertical and Grow Box options turned on, Panorama leaves an empty spot for a grow box in the lower right hand corner.



Elsewhere in this manual you can learn how to turn off the form's scroll bars (see "Window Options" on page 226) and how to add your own custom grow box (see "Elastic Forms" on page 912), to make a final window that looks like this.



Thin Scroll Bar: Scroll bars are normally 16 pixels wide. When this option is checked the scroll bar will be only 11 pixels wide.



Align: This area contains nine radio buttons, allowing you to control the position of the text within the object.



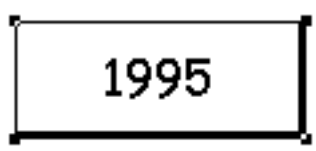
You have the choice of left, center, or right and top, middle, or bottom. (You can also choose left, center, or right from the **Text** menu.) This table illustrates the nine different alignment options.

TOP LEFT	TOP CENTER	TOP RIGHT
MIDDLE LEFT	MIDDLE CENTER	MIDDLE RIGHT
BOTTOM LEFT	BOTTOM CENTER	BOTTOM RIGHT

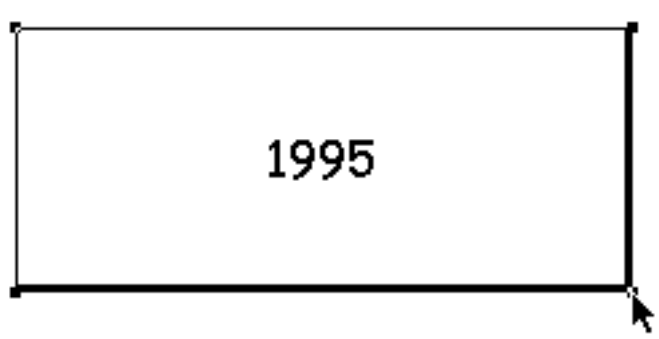
Note: If you have enabled the vertical scroll bar, you must choose one of the top three alignment buttons. If you have enabled the horizontal scroll bar, you must choose the top left alignment button.

Scale Text Size: Usually the Text Display SuperObject displays text in a fixed size, controlled by the **Text** menu or the Graphic Control Strip (see "Text Size" on page 531). However, if the Scale Text Size option is turned on, the text size will be proportional to the height of the Text Display SuperObject. The taller the object, the bigger the text This option can be very useful when used with a form that adjusts as the window size changes.

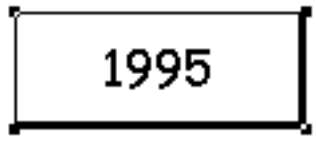
To illustrate this option, let's first look at an ordinary object with this option turned off.



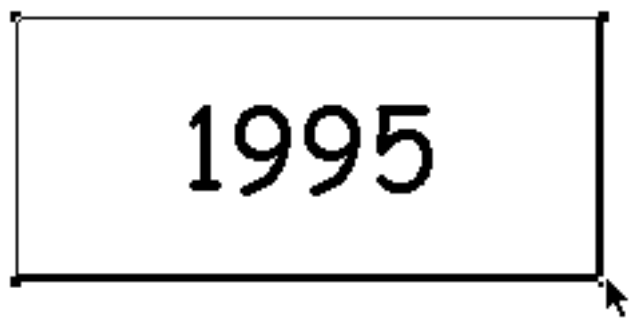
No matter how much we expand or shrink this object, the text size will always be 18 points.



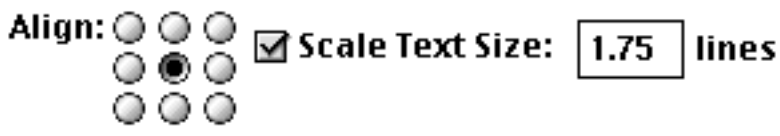
Now let's turn on the Scale Text Size option. At a small size, the object looks pretty similar to the way it did before.



Check out what happens now when the object is expanded! The text size expands also, automatically and in exact proportion to the new height of the object.

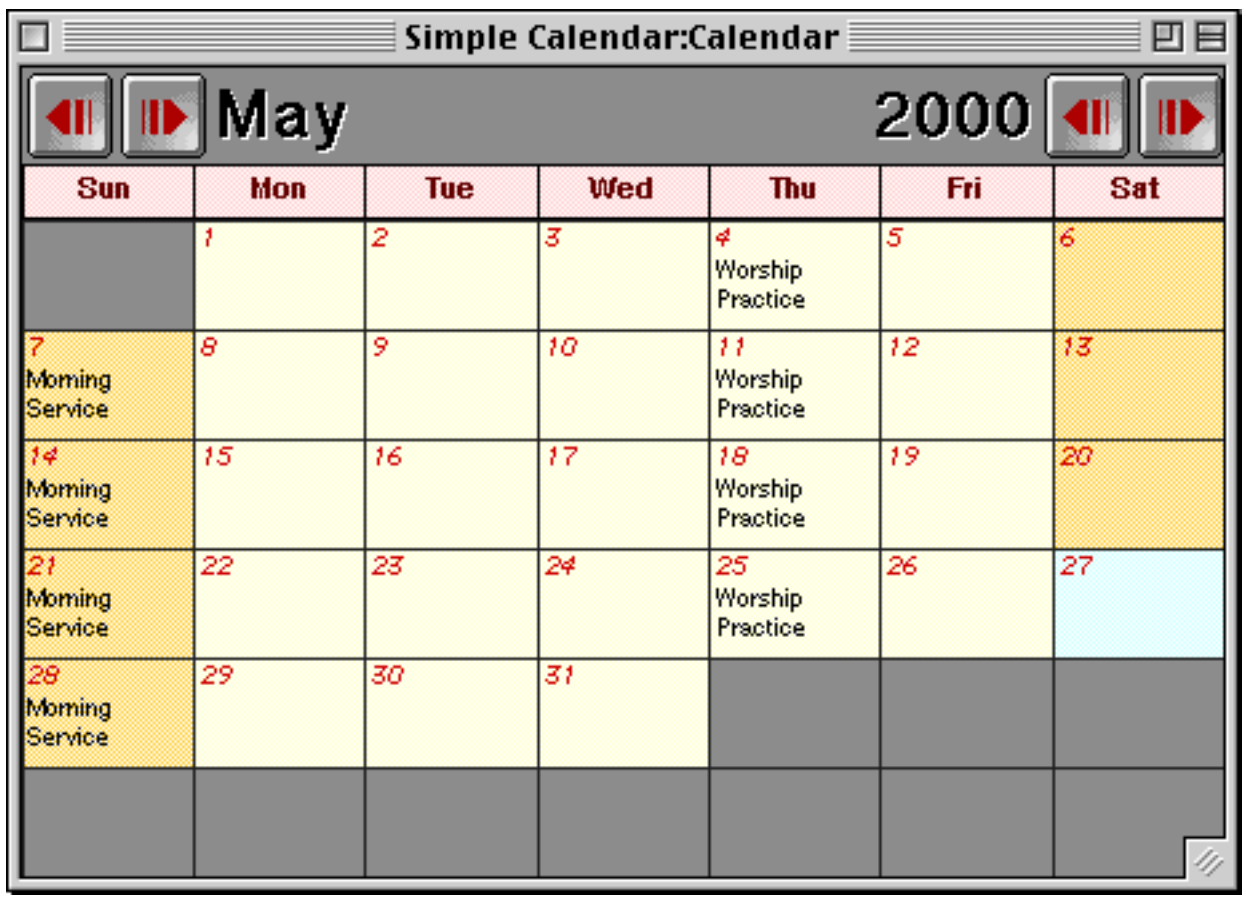


When the Scale Text Size option is enabled, Panorama ignores the font size setting. Instead the value in the Lines box tells Panorama what ratio to use for scaling the text. If the value is 2, the text height will be one half of the object height. If the value is 3.5, the text height will be 1/3.5 of the object height.



You may need to play with the value to get the effect you want, and the value may need to change depending on the font you select. (Remember that when you use the Scale Text Size option, the text may be displayed in any size, so you should select a font that will look good in any size—either a True Type font or a PostScript font with ATM (Adobe Type Manager).)

As an example of a practical use for the Scale Text Size option, here is a calendar that uses this option so that the text gets bigger as the window get bigger. Here's the calendar viewed at minimum size.



When the window expands, all of the text in the calendar expands also.



In addition to **Text Display SuperObjects** this form also uses a **SuperMatrix** object (see "Super Matrix Objects" on page 929) and **Auto Grow SuperObject** (see "Elastic Forms" on page 912).

Controlling Text Display Color and Style on the Fly

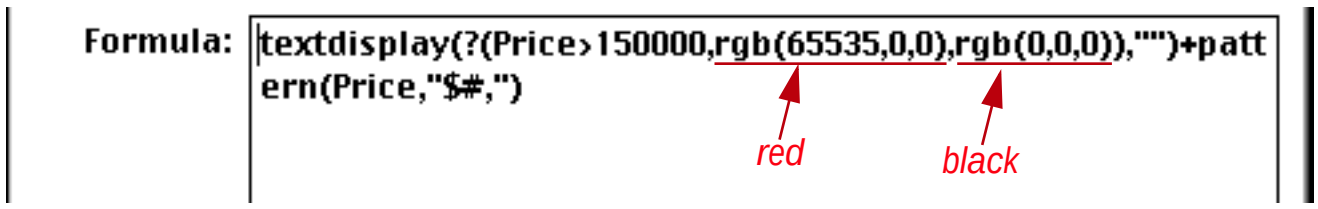
Display text normally appears in the color and style you have selected for the object (see "Color" on page 526 and "Text Style" on page 532). If you are using a Text Display SuperObject, however, it is possible for the formula to change the color and style of the displayed text on the fly. For example you could set up the object so that positive numbers are black and negative numbers are in red. Or, you could make high priority items display in ***bold italic***.

To control the color and style, you must use the `textdisplay(` function (see "TEXTDISPLAY(" on page 5852 for detailed information on this function). Warning: The `textdisplay(` function must be the very first item in the formula! If the `textdisplay(` function is not the first item in the formula, the display will be incorrect. (Advanced Note: The `textdisplay(` function actually generates a special header that is intercepted and removed by the Text Display SuperObject. This header contains special control characters that the object uses to determine what style and color to use.)

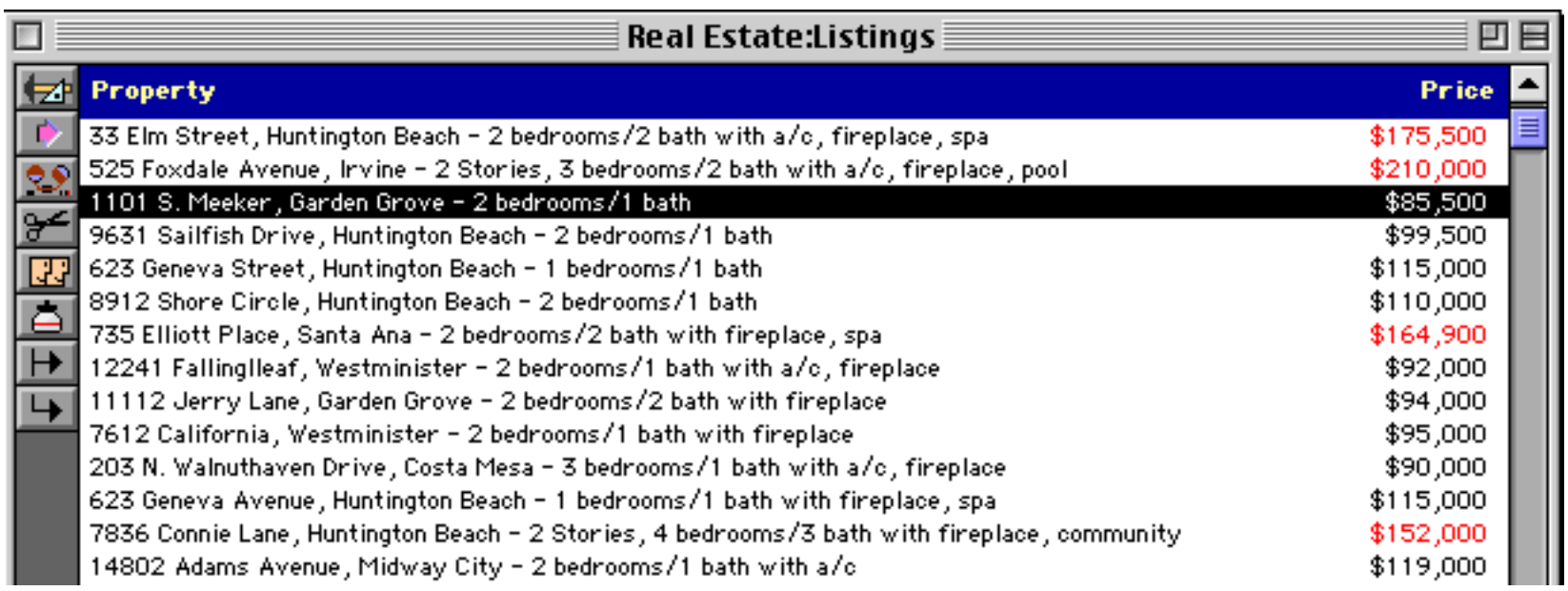
The `textdisplay(` function has two parameters: `color` and `style` The color must be the result of the `rgb(` function (see "Colors" on page 1294 for detailed information on the use of color within Panorama formulas). Some useful colors are listed in the table below.

Formula	Color
`rgb(0,0,0)`	Black
`rgb(65535,65535,65535)`	White
`rgb(65535,0,0)`	Red
`rgb(0,65535,0)`	Green
`rgb(0,0,65535)`	Blue

Here is an example of the `textdisplay(` function in a formula.



This formula causes Panorama to display any price over $150,000 in red. Lower prices are in black.



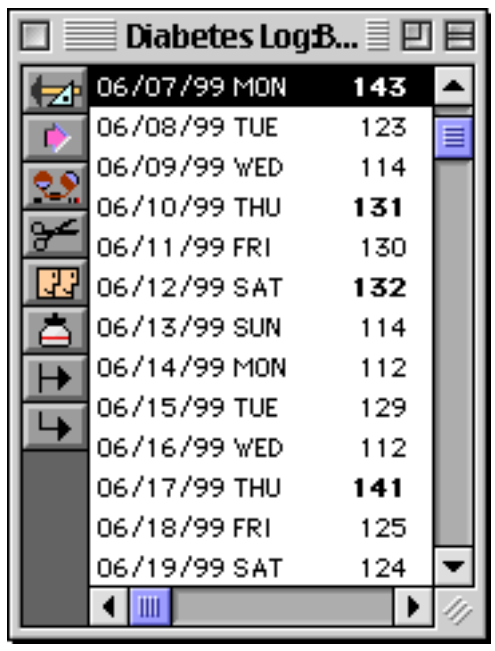
The style parameter is a number that controls the style of the displayed text. For simple styles, simply use the name of the style: `"Plain"` `"Bold"` `"Italic"` `"Underline"` `"Outline"` or `"Shadow"`. If you want to combine multiple styles together, you must specify the style numerically. Add up the number for the styles you want from the table listed below. For example, for bold italic text the style should be 3.

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4
Outline	8
Shadow	16

This example uses the `textdisplay(` function to display numbers greater than 130 in bold.

```
Formula: textdisplay("",?(BBLevel>130,"Bold",""))+str(BBLevel)
```

The formula is being used as part of a blood sugar log for a diabetes patient. Values over 130 are abnormally high and possibly indicate the need for a change in treatment.

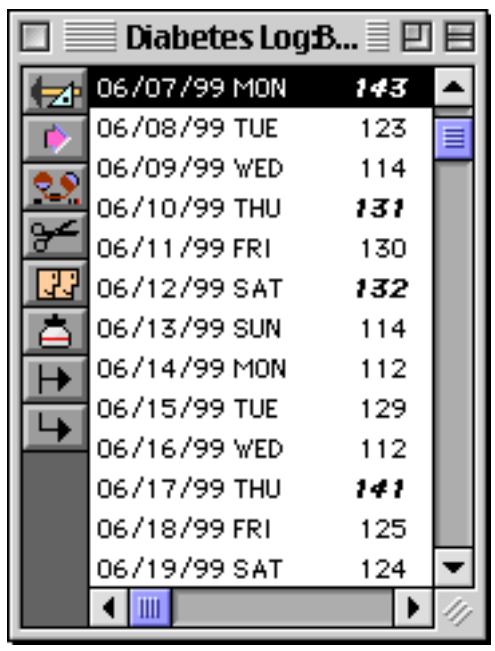


With a slight change this formula will display the abnormal values in bold italic.

```
Formula: textdisplay("",?(BBLevel>130,3,""))+str(BBLevel)
```

1 (bold) + 2 (italic) = 3 (bold italic)

Here's the final result.



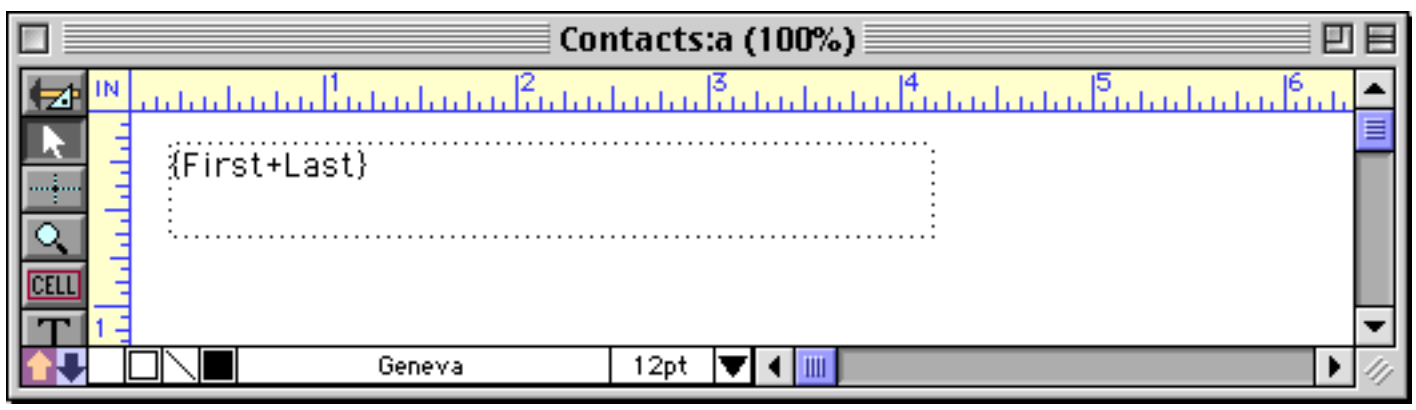
Using Formulas to Display Text

Panorama uses formulas to manipulate numbers and text. Using an auto-wrap text object (see "Displaying Formulas in Auto-Wrap Text" on page 602), a Text Display SuperObject (see "Text Display SuperObjects™" on page 608) or a Text Editor SuperObject (see "Text Editor SuperObject" on page 639) you can display the result of a formula on a form (and, since forms are used to produce reports, on a printed report).

There are an infinite number of ways to combine fields, variables and functions into useful formulas. In the following sections we will explore some of the more common types of formulas used to display information in forms.

Combining Multiple Text Items Into One

Many times you'll need to combine several different fields or variables together, usually with captions and punctuation (carriage returns, commas, spaces etc.) In a formula two text items can be combined with the + operator.



This formula probably isn't what you had in mind, because the result (seen below in Data Mode) doesn't have a space between the first and last name.

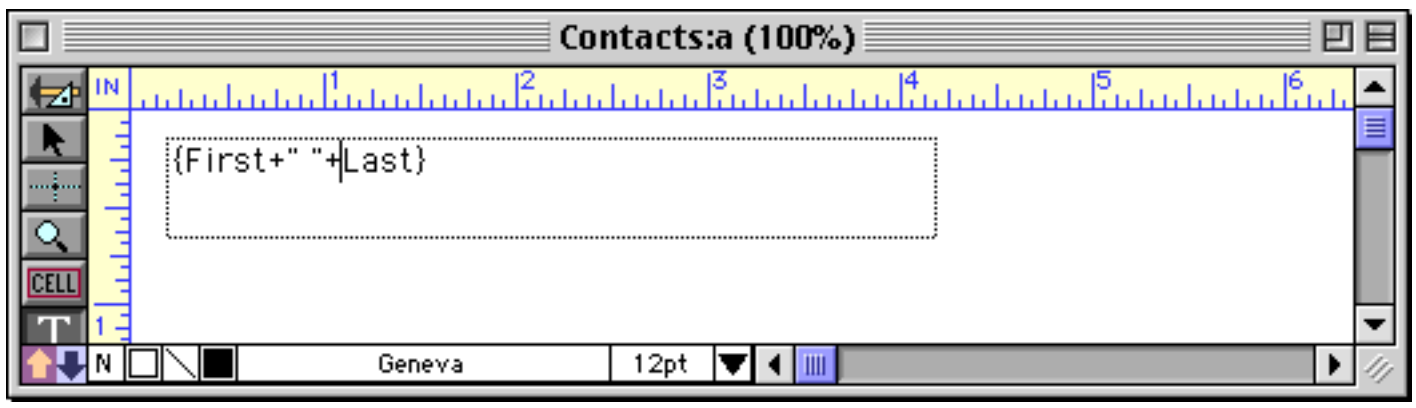


Fixed text items (like captions, spaces and other punctuation) must be enclosed in quotes. Panorama allows several different kinds of quotes, as shown in this table.

Type	Open	Close	Example
Double Quote	"	"	"January"
Single Quote	'	'	'Tuesday'
Curly Braces	{	}	{San Francisco}
Smart Double Quote	“	”	Gothic
Smart Single Quote	‘	’	Bohemian

Curly braces cannot be used to quote text in an auto-wrap text object, because they are used to surround the entire formula. Other than that you can use any one of these pairs of quote characters whenever you want. See “Constants” on page 1195 for more information about quoting text.

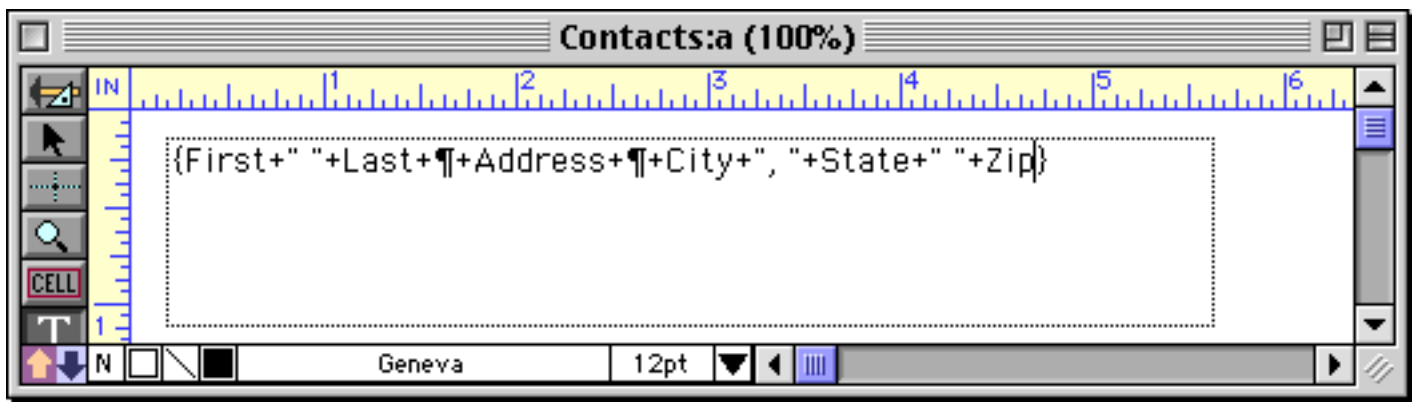
Now that we know how to quote a fixed text item we can add a space between the first and last names.



Switch to Data Mode (see “Form Modes: Data Access vs. Graphic Design” on page 485) and voila! The correctly formatted name appears.



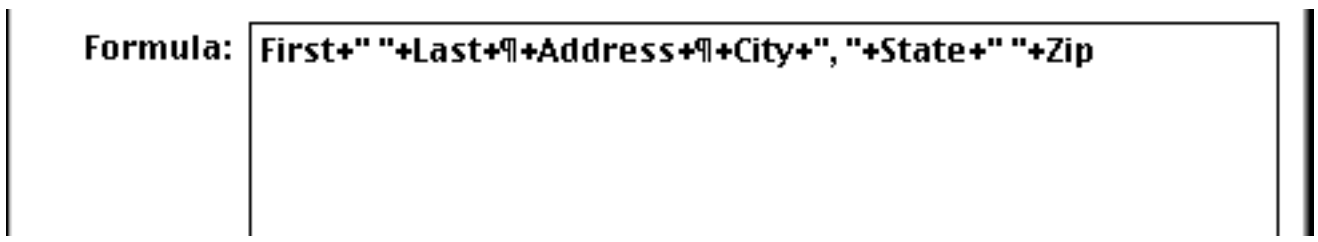
In a formula a carriage return is represented by the ¶ symbol. On the Macintosh you can enter this by typing **Option-7**. On Windows systems press **Alt-0182**. We can use this symbol to help build a complete address label.



The formula appears all on a single line, but switching to Data Mode shows the finished label on three lines.



Our example used an auto-wrap text object, but the exact same formula could be used with a Text Display SuperObject. In Data Mode this object will look exactly like the previous example. (Of course, using a Text Display SuperObject would give you more options for aligning and scaling the text, see “Text Display Options” on page 611).

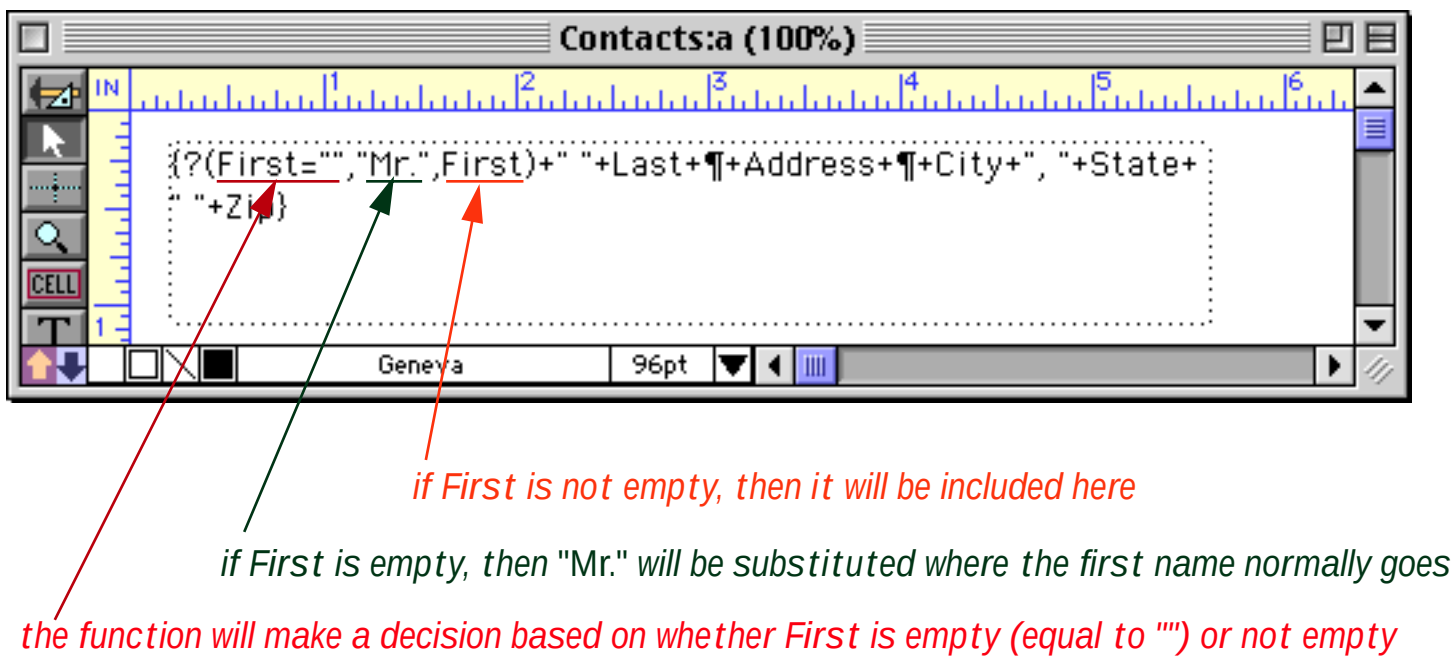


Note: The alert reader will have noticed that it is possible to create a label like this using data merge in an auto-wrap text object, without using all of these quotes and ¶ and + symbols (see “Displaying Data in Auto-Wrap Text” on page 595). The data merge is simpler, so why bother with a formula? In this case there is no reason except to illustrate the ability to combine text items together. In the following sections, however, we will expand on this example to show applications that can only be done with a formula.

Creating a Smart Formula

In the real world, data often doesn't fit into neat little boxes. Some people will enter their middle initial, some won't. Some motels have off peak rates, some don't. Some countries measure temperature in Fahrenheit, some in Celsius. It takes a bit of work, but using the ?(and sandwich(functions you can set up formulas that display data correctly under changing, sometimes opposite circumstances.

The ?(function allows a formula to make a yes/no, either/or decision. For example, consider the address label created in the previous section. Suppose the first name is missing? Using the ?(function a formula can be constructed that substitutes Mr. for the missing first name.



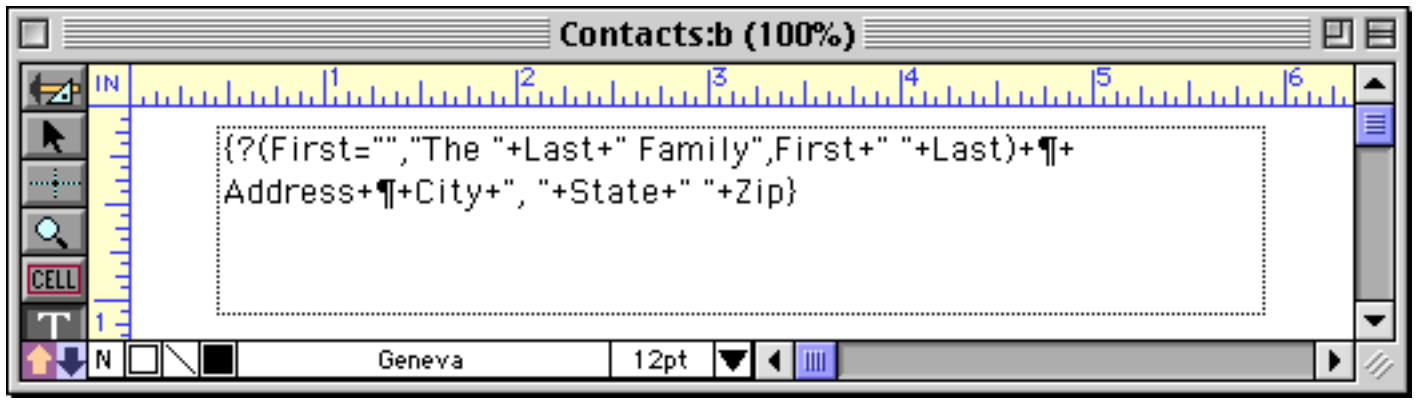
In data mode, anyone with a first name will simply display a standard label including the first and last names.



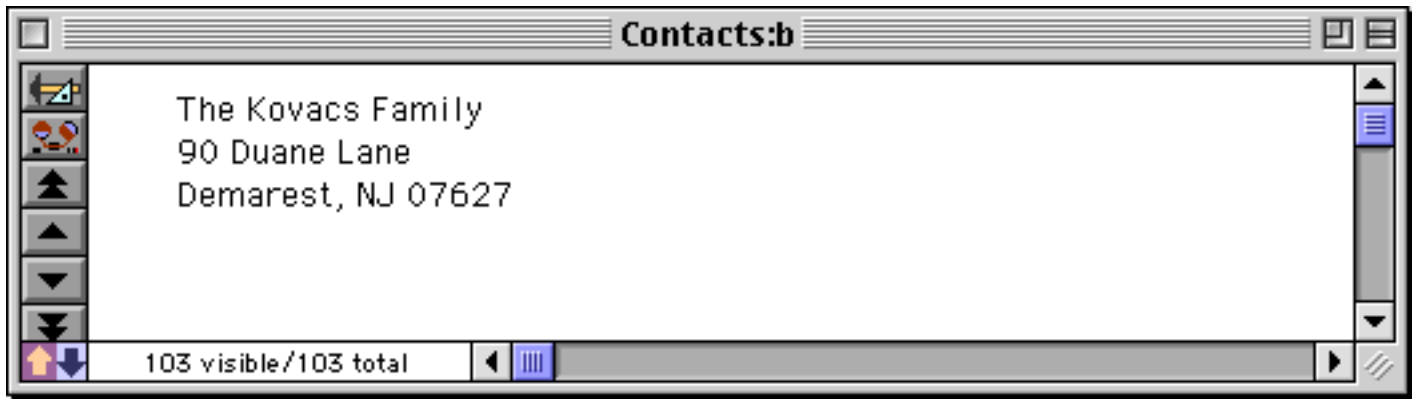
But if the first name is missing, Mr. will be substituted.



There are many ways you can use the ?(function. Here is a slight re-arrangement of the previous example.



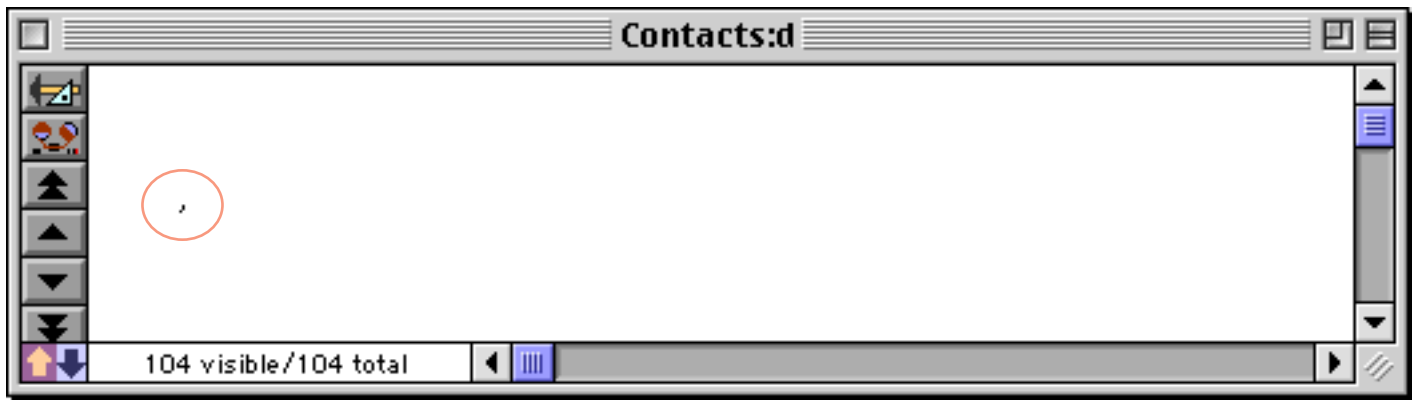
Here's what this formula produces if the first name is missing.



The ?(function is a simple but very powerful tool. See "The ? Function" on page 1271 for more detailed information about this function.

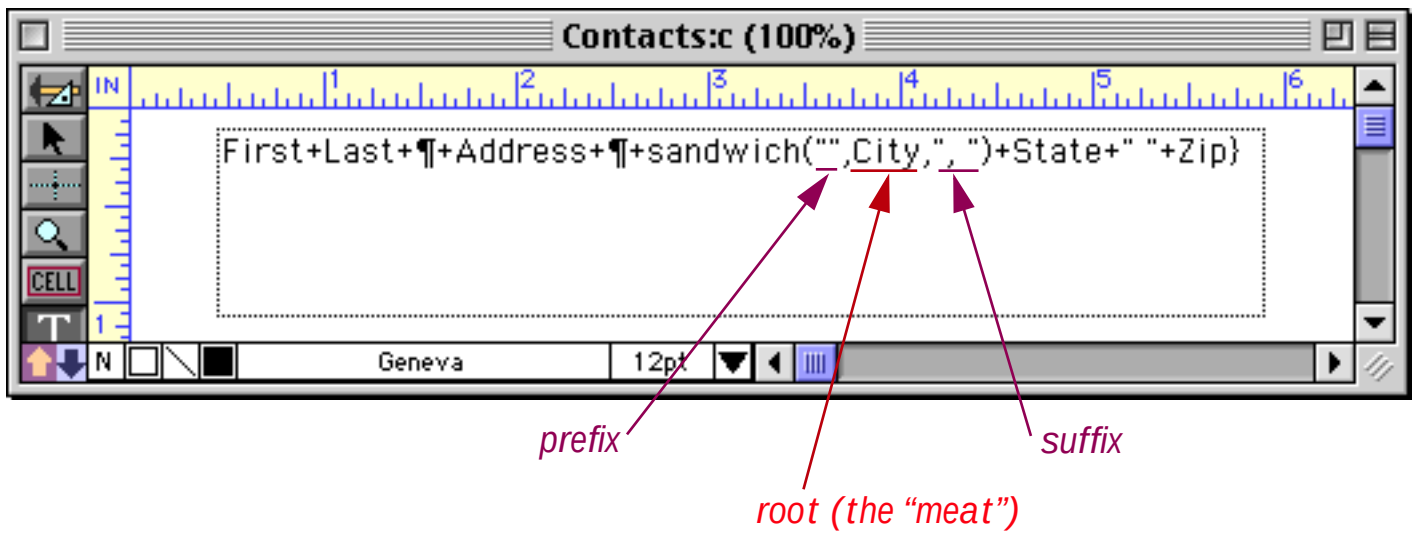
Eliminating Unnecessary Punctuation and Blank Areas With the Sandwich Function

Yes, Panorama actually has a function named sandwich! If an item of data is missing, you'll usually want to eliminate any punctuation that is associated with that item. For example, if the middle name is missing, you won't want to include the extra space. If the city is missing from an address, you'll want to leave off the comma afterwards, instead of leaving a comma hanging in the air like this.

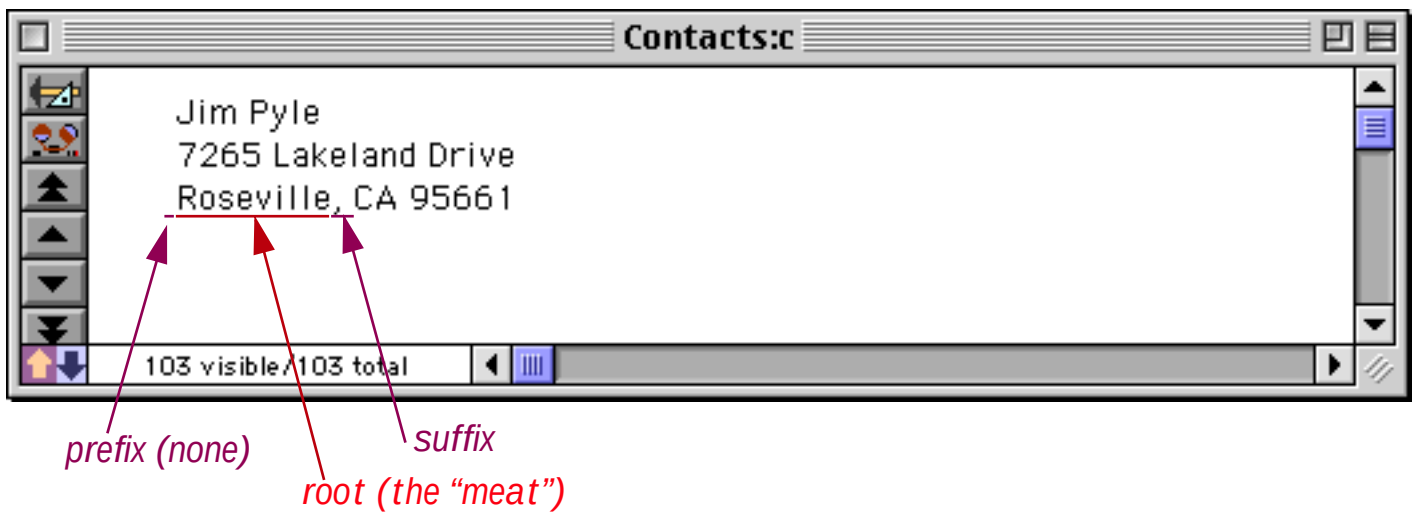


If the company name is missing from an address, you'll want to leave off the following carriage return so there won't be a blank line. All of these tasks can be performed with the ?(function, but there's also an easier way: the sandwich(function.

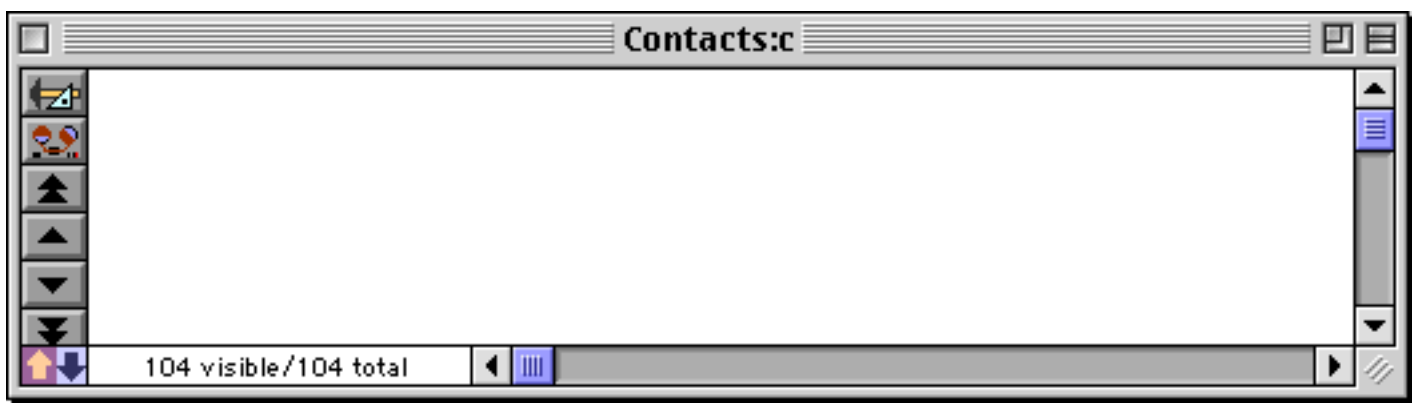
The sandwich(function has three parameters: prefix, root and suffix. The root is the main item of text you want to display. The sandwich(function will add the prefix and suffix to the beginning and end of the root, kind of like slapping bread around a slice of salami. However, if the root is empty, the sandwich(function won't "slap on the bread."



The results of this function depend on whether or not the City field contains any text. If it does, Panorama adds the prefix (which in this case is empty) and the suffix.



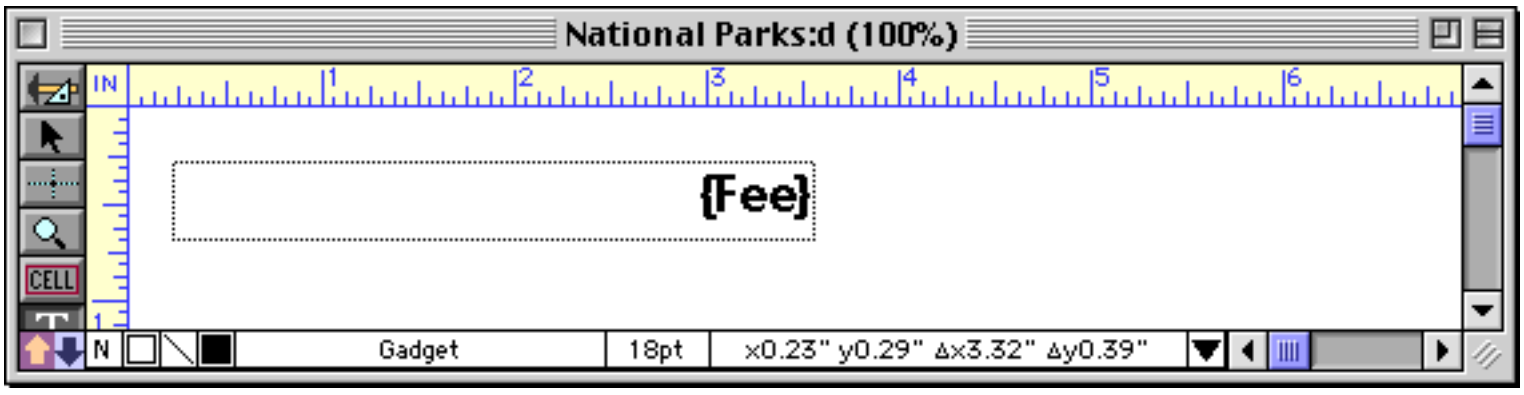
If the City field is empty, Panorama leaves out the prefix and the suffix also. Here's our empty record again, but this time, no comma hanging in the middle of the air!



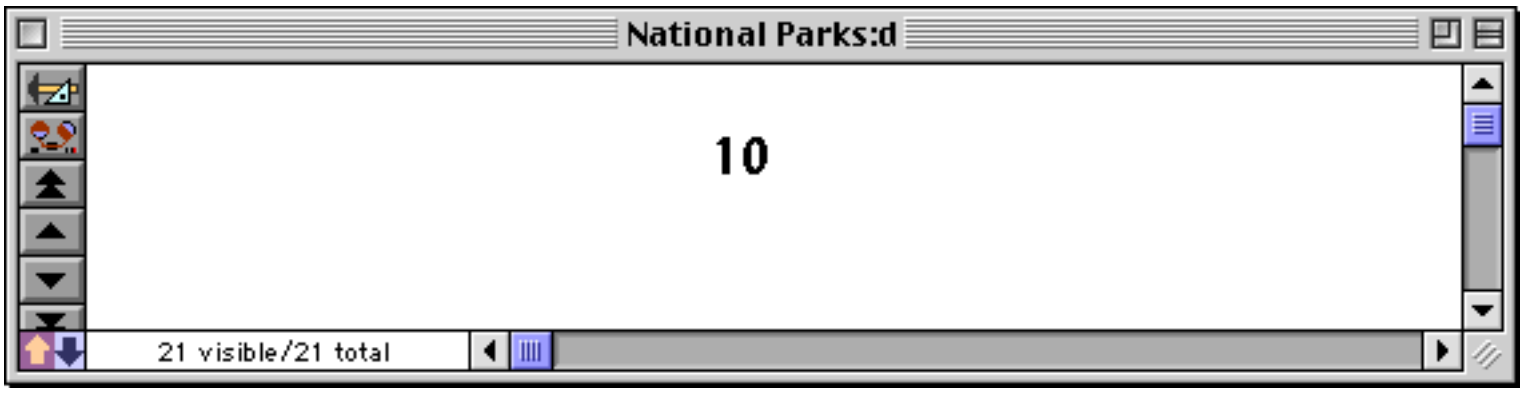
You'll find that the sandwich(function is very delicious any time you need to conditionally include spacing or punctuation around a field that might be blank. (Sorry, couldn't resist.)

Combining Numbers with Text

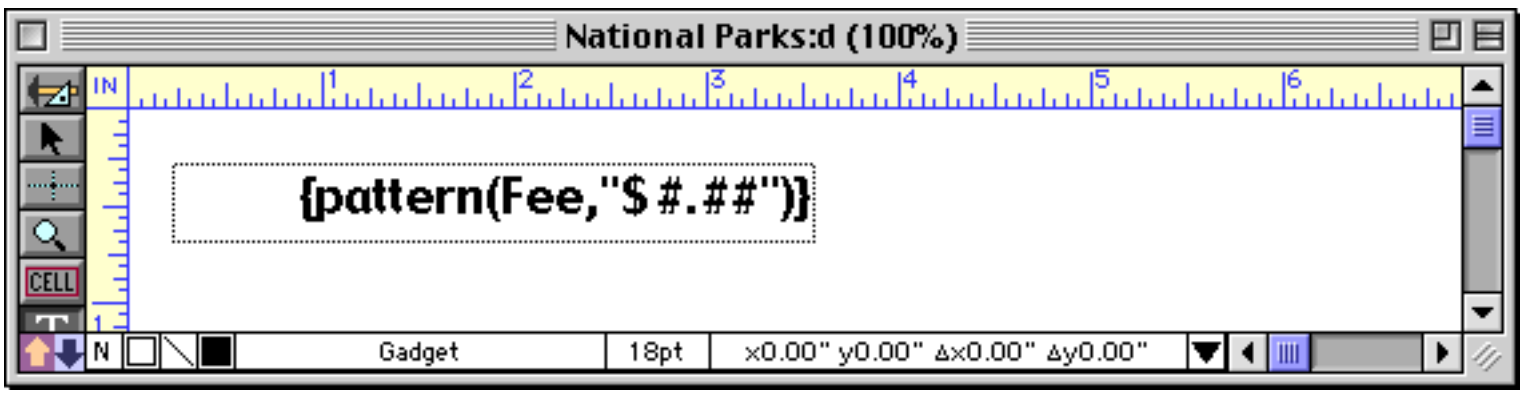
If a formula contains nothing but a single numeric value, Panorama will automatically convert the value to text for you, like this.



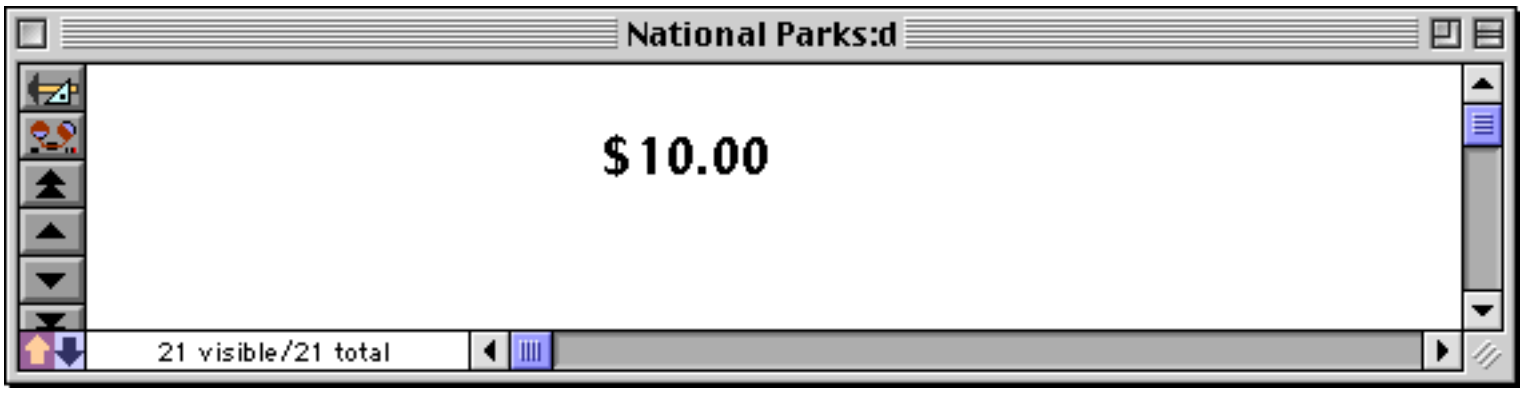
Panorama will decide for itself what format to use for the number.



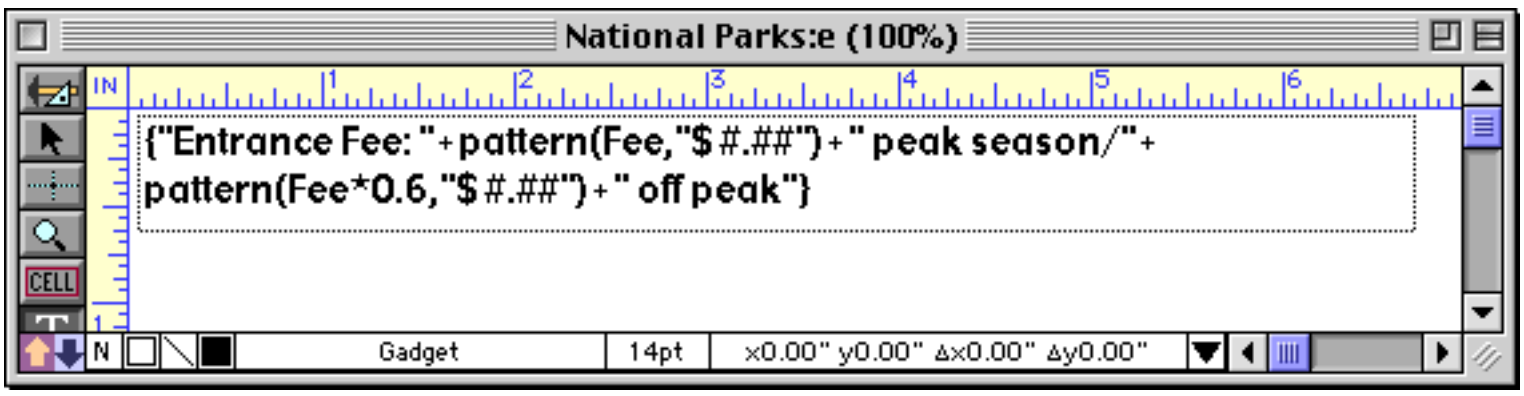
If you don't like the format that Panorama chooses you can use the `pattern(` function to specify the exact format you want to use.



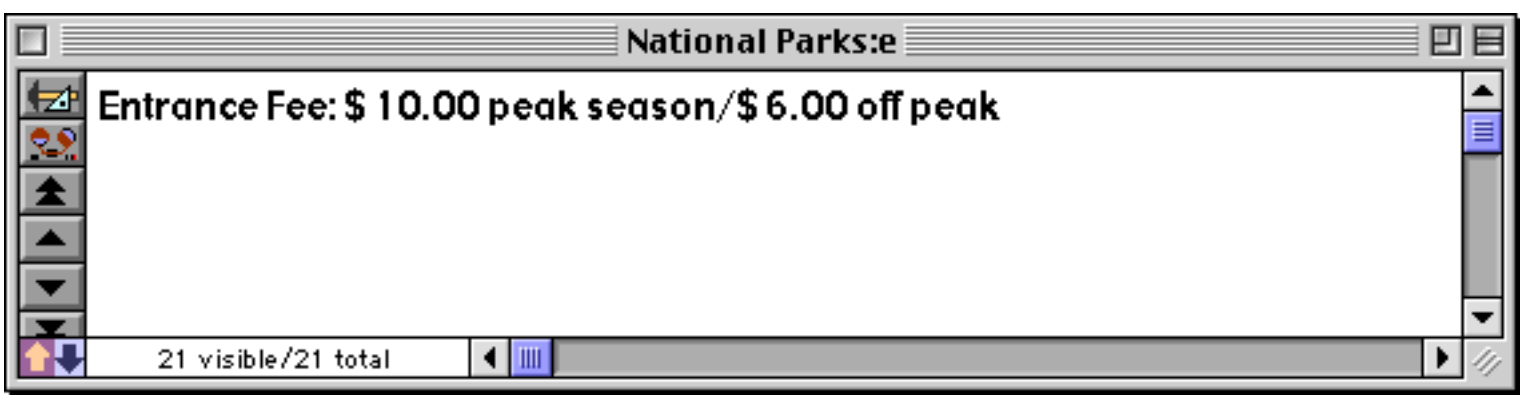
The `pattern(` function gives you total control over the format of the final number. See "Converting Between Numbers and Strings" on page 1227 for a complete description of this function.



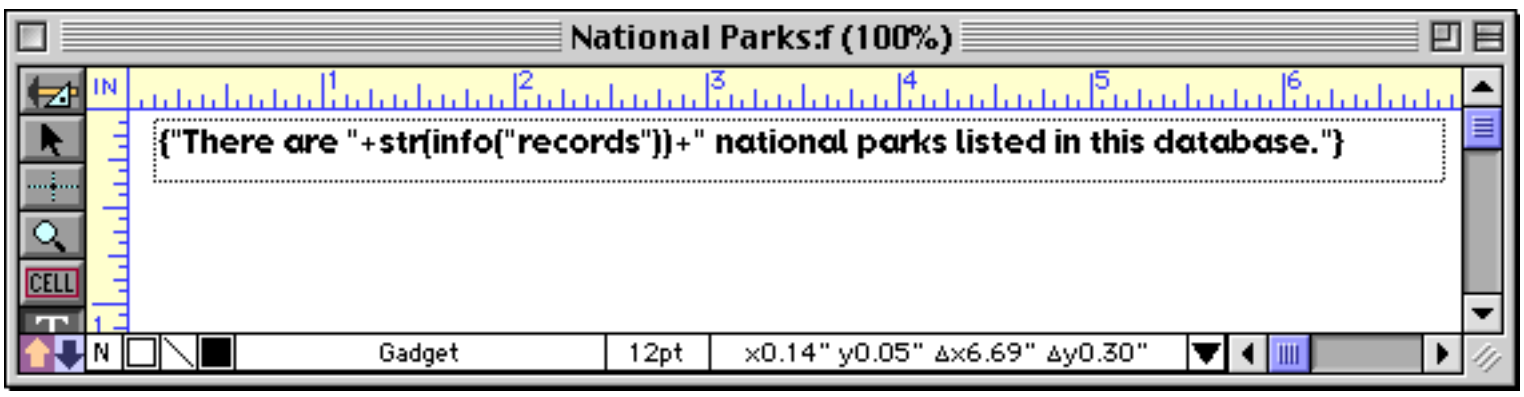
If your formula results in more than a single number (for example two numbers or text and a number) you must convert the numbers to text before they can be used in the formula. This must be done with the `str(` or `pattern(` functions as shown in this example.



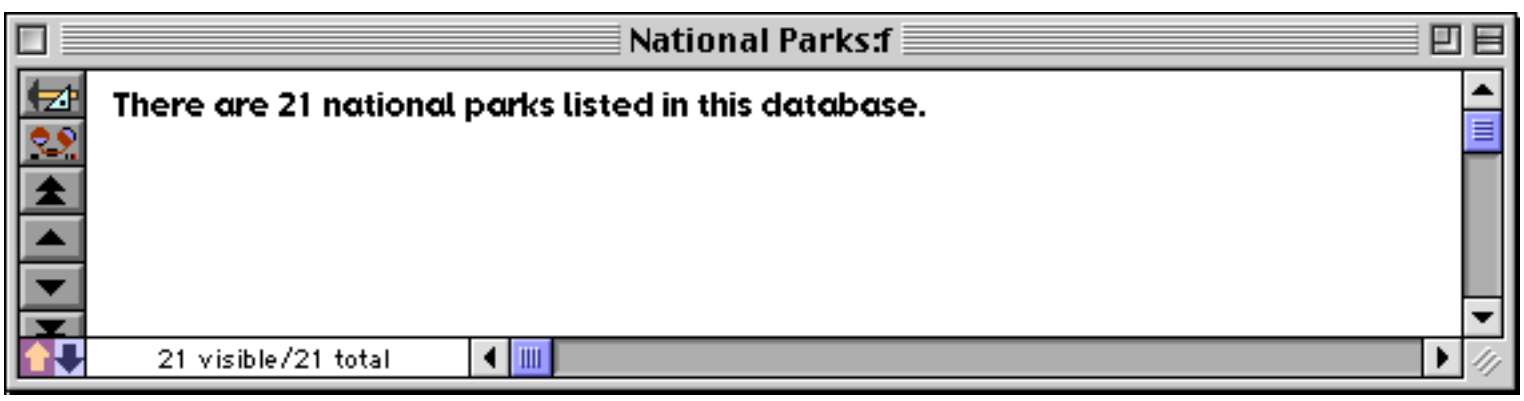
Here's the finished result in Data Mode.



The `pattern(` function gives you total control over the format of the number. Use the `str(` function if you are content to let Panorama decide what format to use.



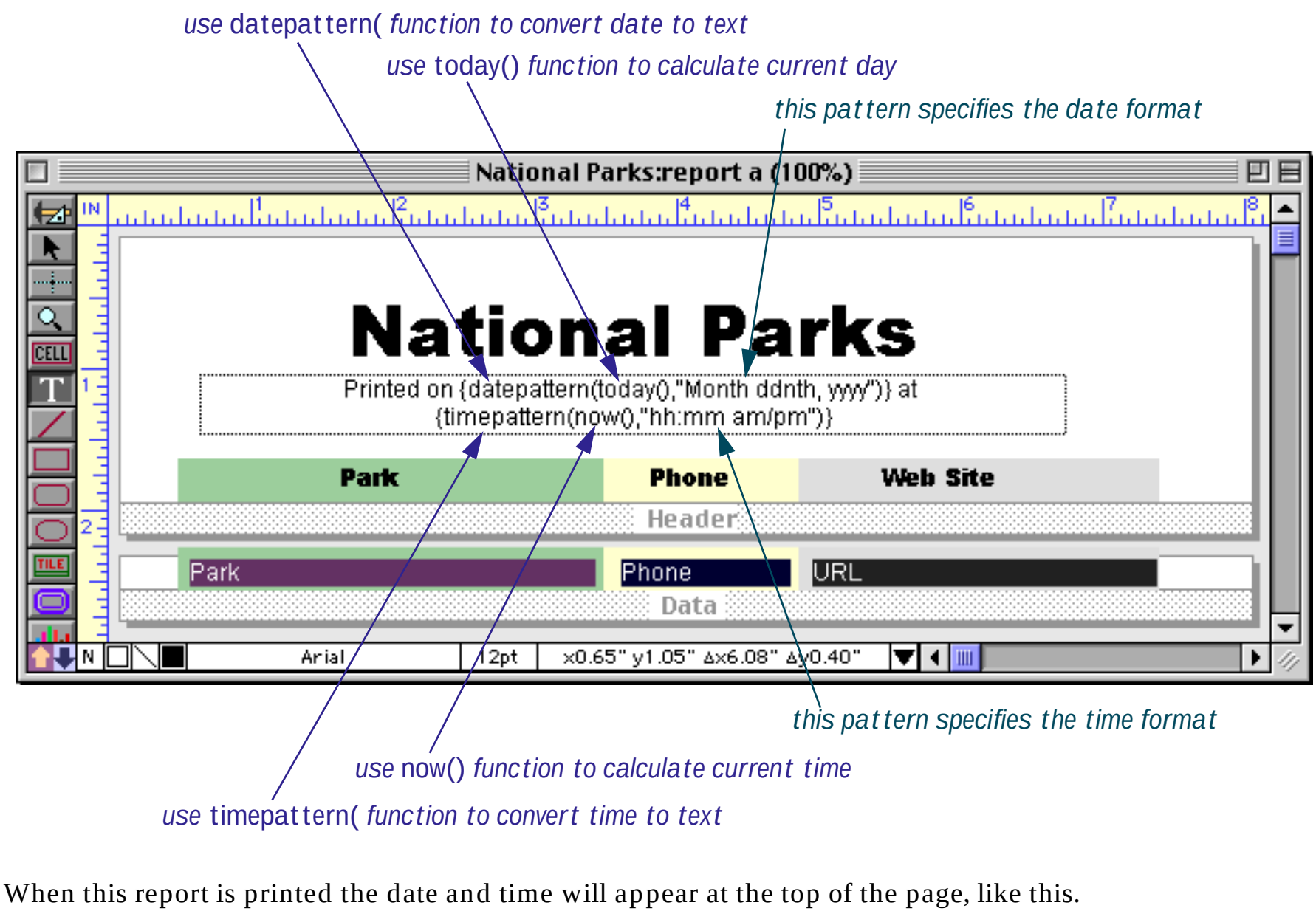
In this case Panorama chose a simple integer format.



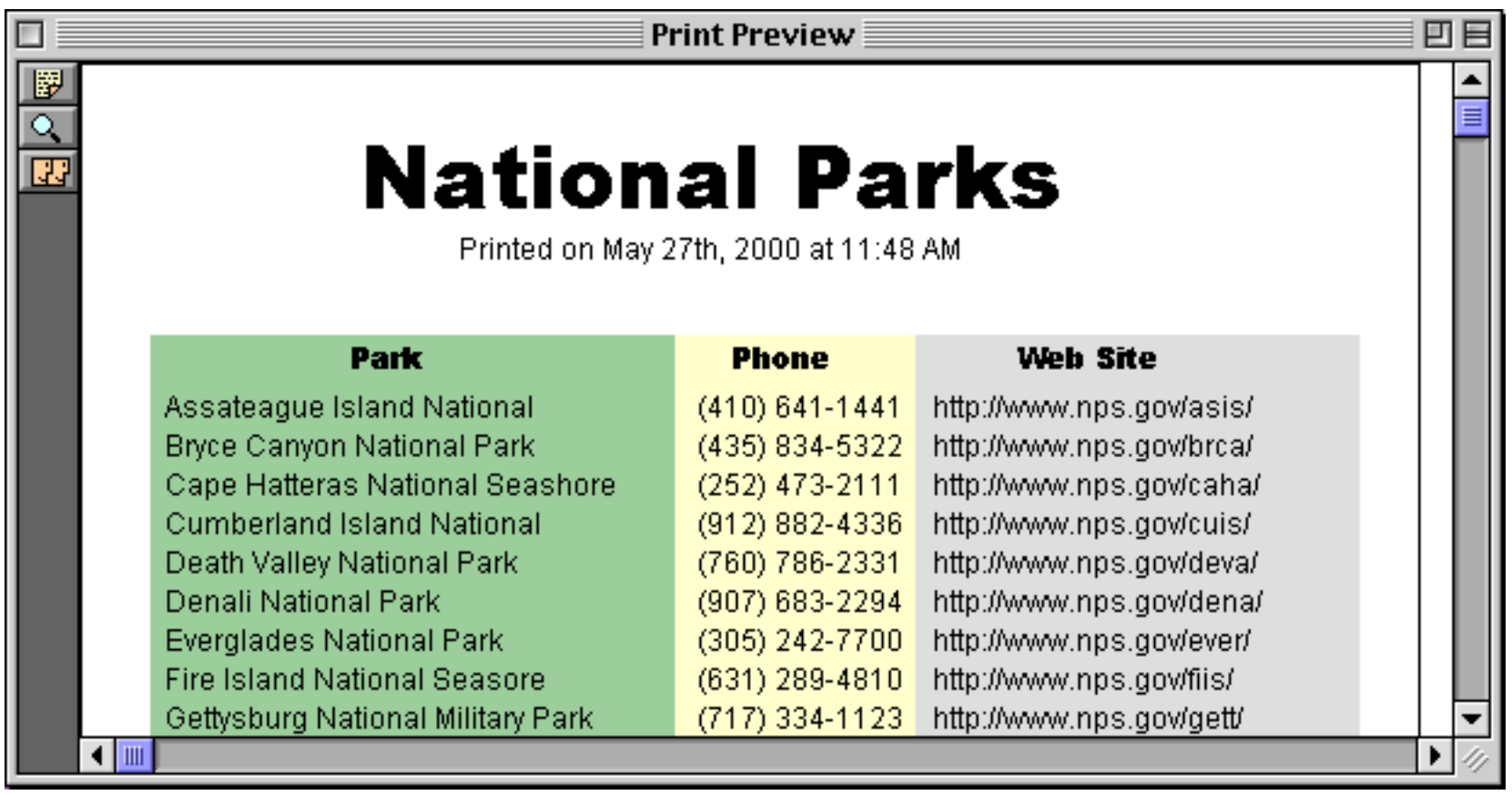
By the way, in case you haven't guessed, the `info("records")` function calculates the total number of records in the database. See "INFO("RECORDS")" on page 5410 for the complete details on this function. See "Converting Between Numbers and Strings" on page 1227 to learn more about the `str(` function.

Displaying Dates

To display a date in a field or variable you must convert that date to text with the `datepattern(` function (see "Converting Between Dates and Text" on page 1249 for all the gory details). Here's a simple example that prints the current date and time on the top of each page of a report.



When this report is printed the date and time will appear at the top of the page, like this.



Park	Phone	Web Site
Assateague Island National	(410) 641-1441	http://www.nps.gov/asis/
Bryce Canyon National Park	(435) 834-5322	http://www.nps.gov/brca/
Cape Hatteras National Seashore	(252) 473-2111	http://www.nps.gov/caha/
Cumberland Island National	(912) 882-4336	http://www.nps.gov/cuis/
Death Valley National Park	(760) 786-2331	http://www.nps.gov/deva/
Denali National Park	(907) 683-2294	http://www.nps.gov/dena/
Everglades National Park	(305) 242-7700	http://www.nps.gov/ever/
Fire Island National Seasore	(631) 289-4810	http://www.nps.gov/fiis/
Gettysburg National Military Park	(717) 334-1123	http://www.nps.gov/gett/

This example was created with an auto-wrap text object and two embedded formulas. You can create the same effect with a Text Display SuperObject, but in that case you must use a single formula like this.

```
"Printed on "+datepattern(today(),"Month ddnth, yyyy")+
" at "+timepattern(now(),"hh:mm am/pm")
```

The end result is the same either way. (However, with the Text Display SuperObject you would have the option to center the text vertically or to scale the text automatically. See “Text Display Options” on page 611.)

Merging Images Into Text

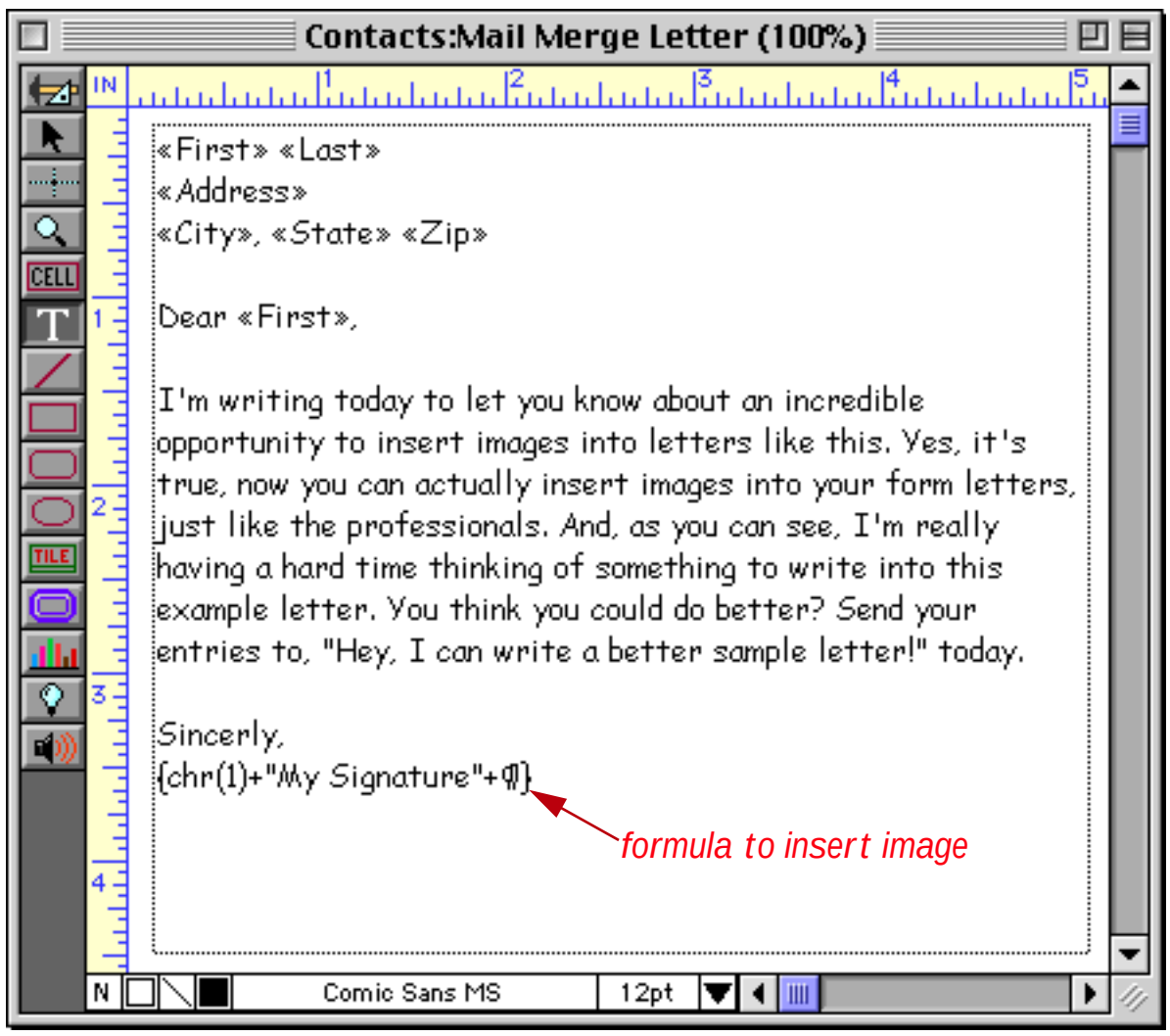
Panorama allows an image from the **Flash Art Gallery** (either stored in the database file itself or on disk) to be merged into the middle of an auto-wrap text object or Text Display SuperObject (see “Flash Art™” on page 748 for more information about this gallery). The image cannot be merged into the middle of a line, but simply replaces one line of text. The line will expand to the full height of the image. For example, this feature could be used to insert a logo or a signature in a letter. To illustrate this feature we’ll assume you have an image in your Flash Art Gallery named My Signature.



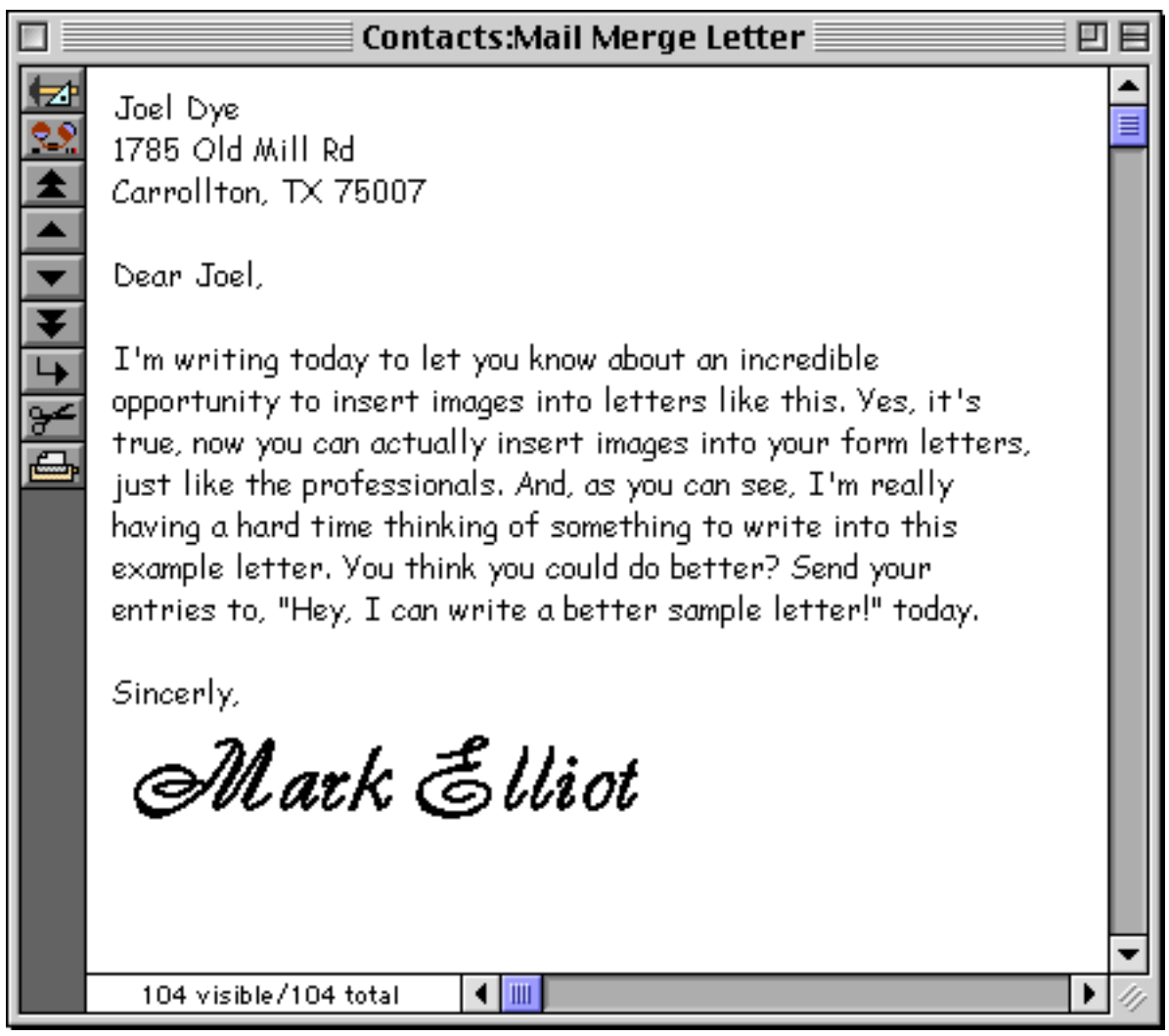
To insert an image into an auto-wrap text object you must use a special formula. The formula looks like this (Of course, you should insert the actual name of your image instead of My Signature).

```
{chr(1)+"My Signature"+¶}
```

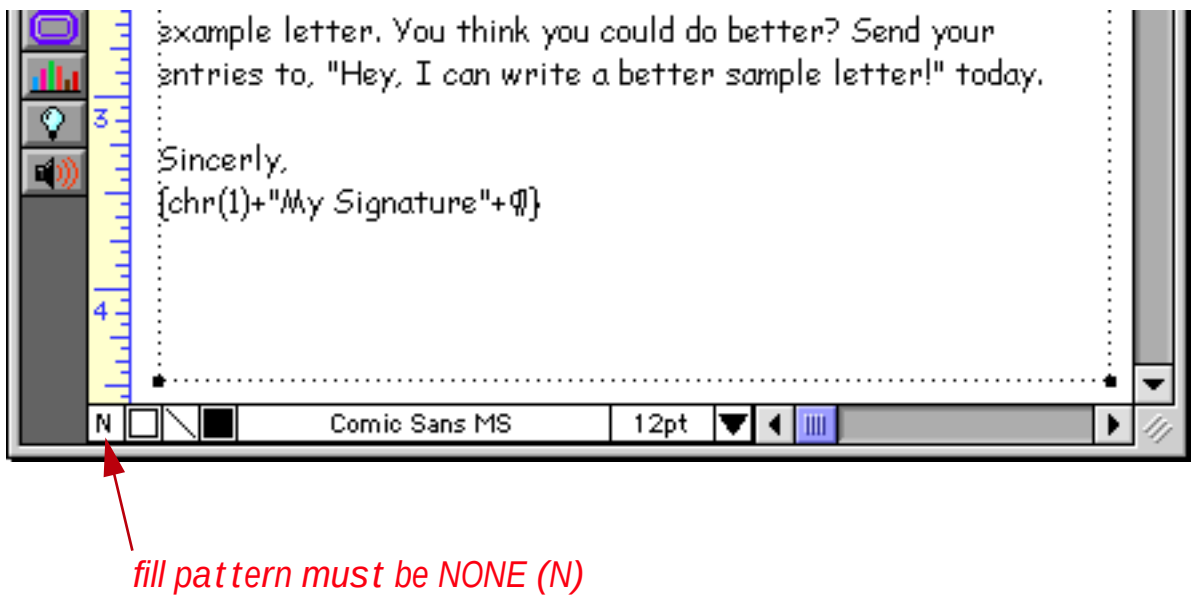
Here is a complete letter with the special formula for inserting the image at the bottom. The formula must be on a separate line all by itself, with nothing else on the line.



Switch to Data Access Mode to see the finished product.



Panorama can be a little bit picky about displaying an image in an auto-wrap text object. Remember, the image cannot be mixed in the middle of a line of text, but must be on a line all by itself. In addition, this feature only works if the auto-wrap text object is filled with NONE (see “Fill Pattern” on page 521). If the text object is filled with white or any other color, the image will not appear. Since NONE is the default for new text objects you probably won’t have a problem, but if you do have difficulties getting the image to appear this is the first thing to check.



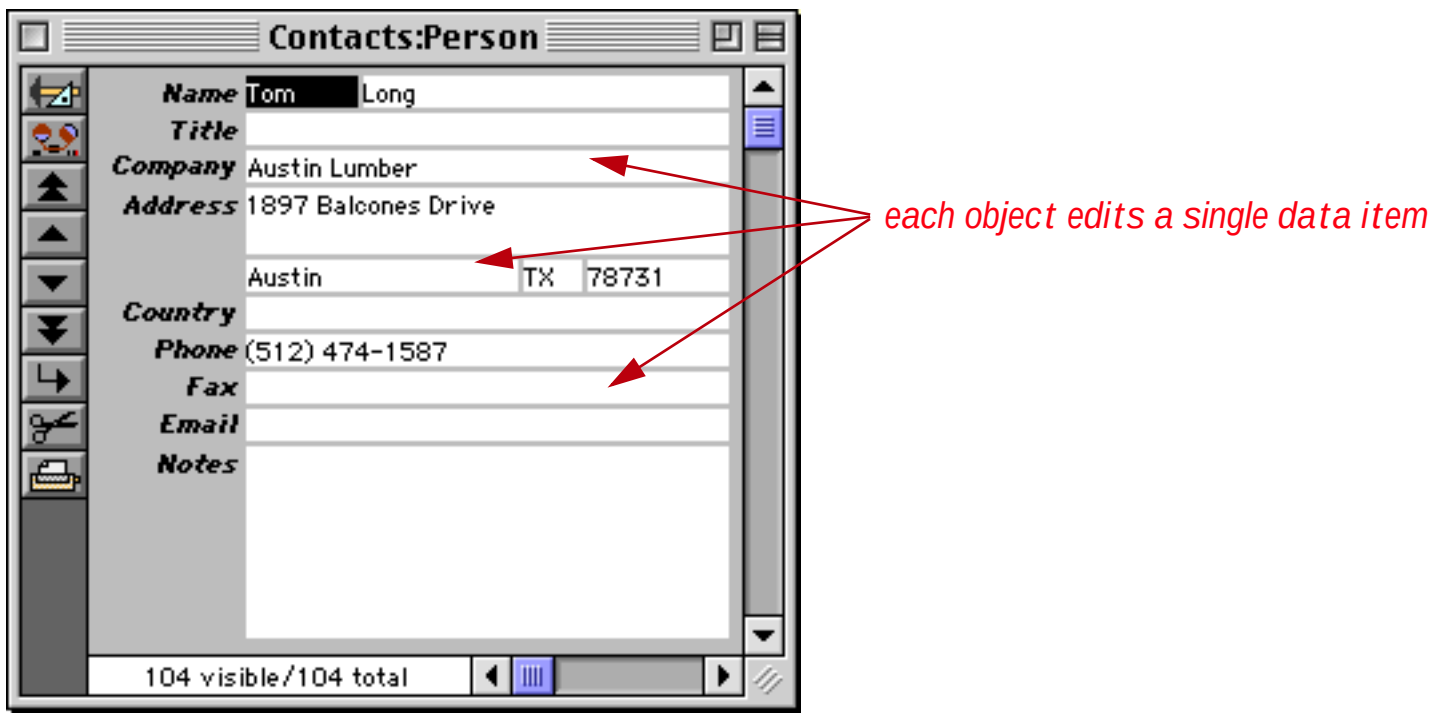
One final tip — this feature works with either an auto-wrap text objects or a Text Display SuperObject. (However, it does not work with Text Editor or Word Processor SuperObjects.)

Editing Text

Most data entry and editing is done with the keyboard. The rest of this chapter shows how to set up a form objects for editing text.

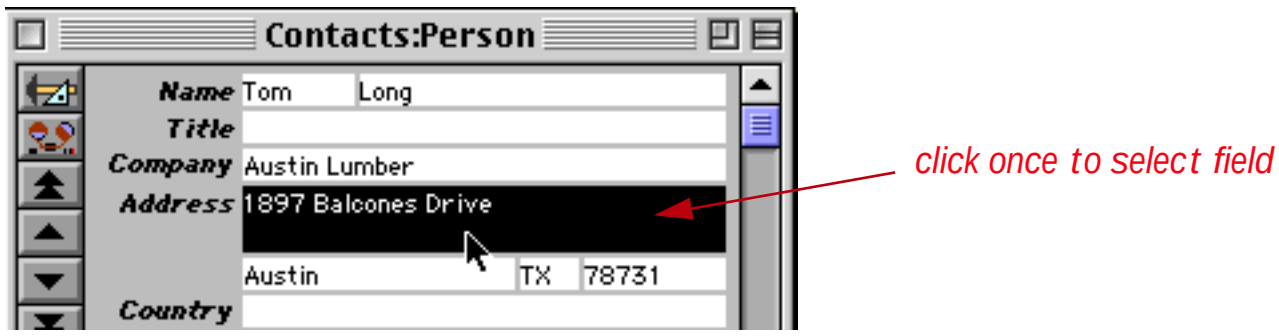
Types of Data Editing Objects

In a form data entry is done through objects (just like everything else in a form!). Each object allows a specific item of data to be edited (for example a person’s first name or a phone number). A collection of data editing objects is assembled to create a complete data entry form.

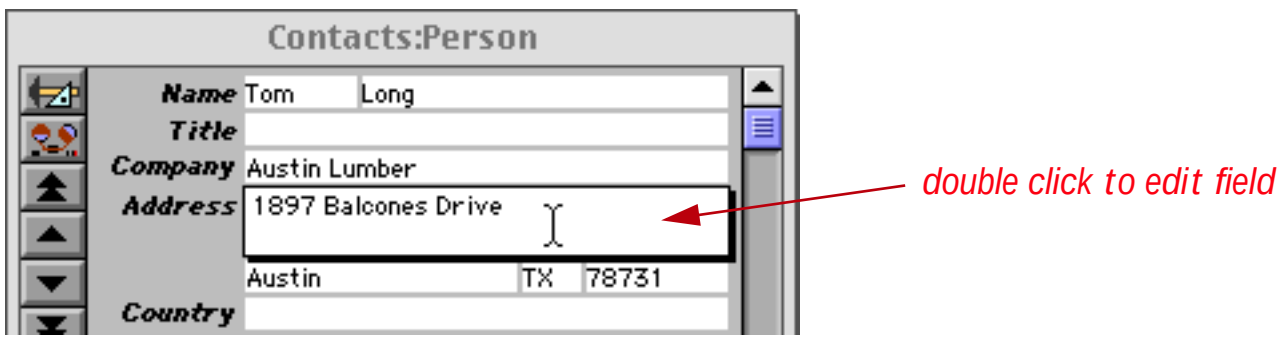


Panorama has two primary types of objects for editing text: **data cell objects** (shown above) and **Text Editor SuperObjects**. You can mix these two types on a single form, but usually you'll want to choose one type per form and stick with it.

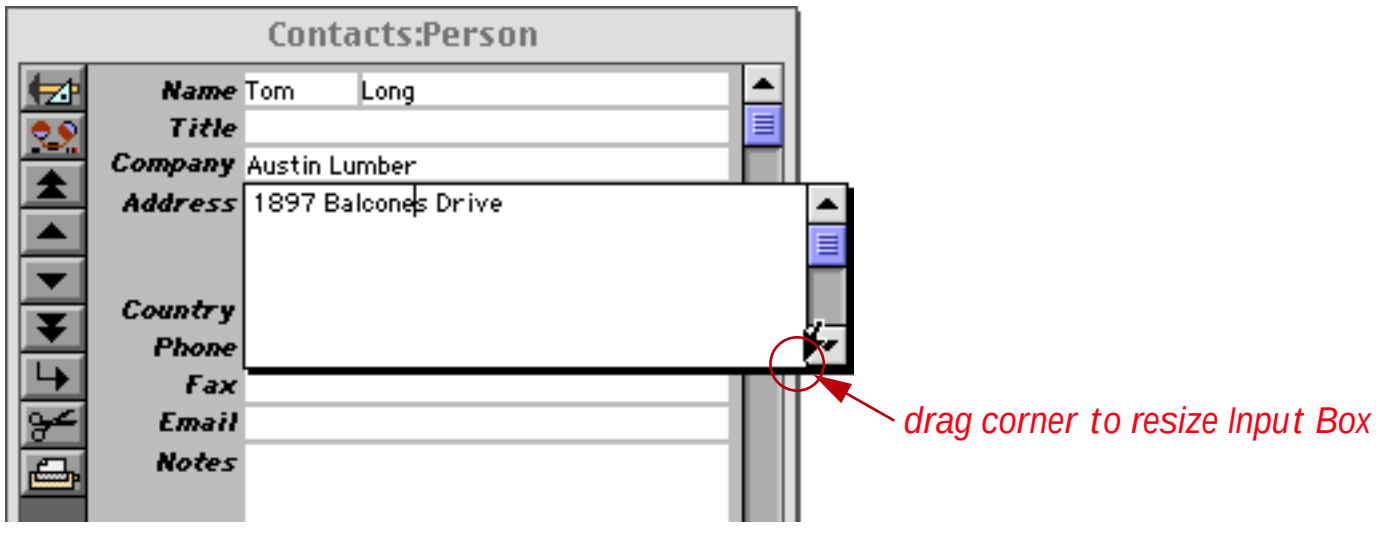
Data cell objects are the "classic" way to edit data in a Panorama form. In early versions of Panorama (before version 3) this was the only kind of text editing object available. Data cell objects are designed to mimic the way Panorama works in the data sheet. In Data Mode, clicking once on a data cell object selects the field, but does not open the field for editing.



Clicking twice on the object opens the pop-up data editing Input Box you are familiar with from the data sheet (see "The Input Box" on page 310).



Just as in the data sheet, the data cell object's Input Box can be expanded by dragging on the lower right hand corner (see "Expanding the Input Box" on page 311).



To learn how to add data cell objects to your form see "Working with Data Cell Objects" on page 635.

As an alternative to data cells, a form may be designed with **Text Editor SuperObjects**. Text Editor SuperObjects allow you to edit text right in the form window—no double click is required. You can simply click or drag on the text to begin editing. Press **Enter** when you are finished. The illustration below shows the effect of double clicking on the word Harmony. As you can see, instead of opening an Input Box this selects the word for editing.



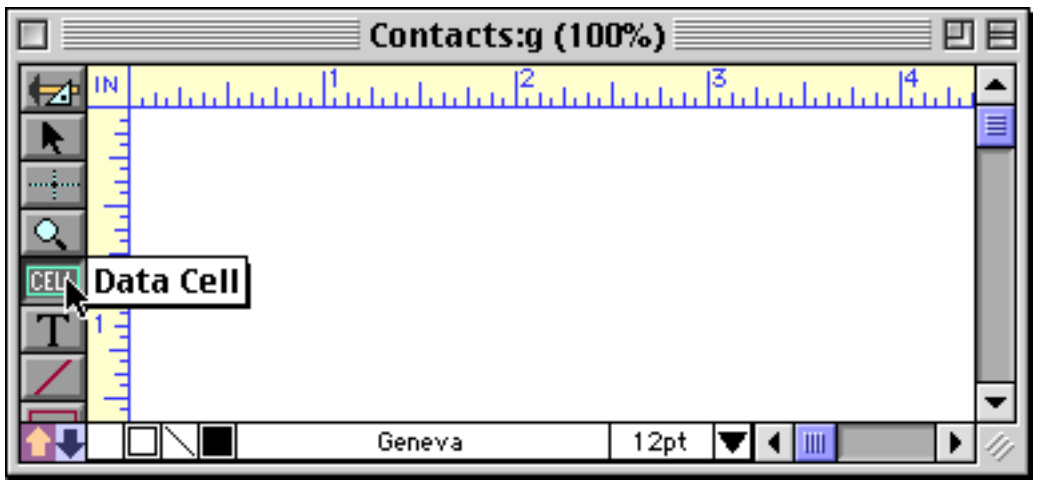
Since the Text Editor SuperObject doesn't use an Input Box, you cannot expand the size of the editing area "on-the-fly" the same way you can with data cells. The editing area must be defined in advance. On the other hand, the Text Editor SuperObject doesn't require the extra double click, and works more like other standard applications you may be used to. See "Creating and Modifying Text Editor SuperObjects" on page 639 to learn how to create a form with Text Editor SuperObjects.

The table below summarizes the differences between data cell objects and Text Editor SuperObjects. For many applications, either type will work all depending on your personal preferences. Some advanced features (for example editing variables, see next section) do require the use of Text Editor SuperObjects.

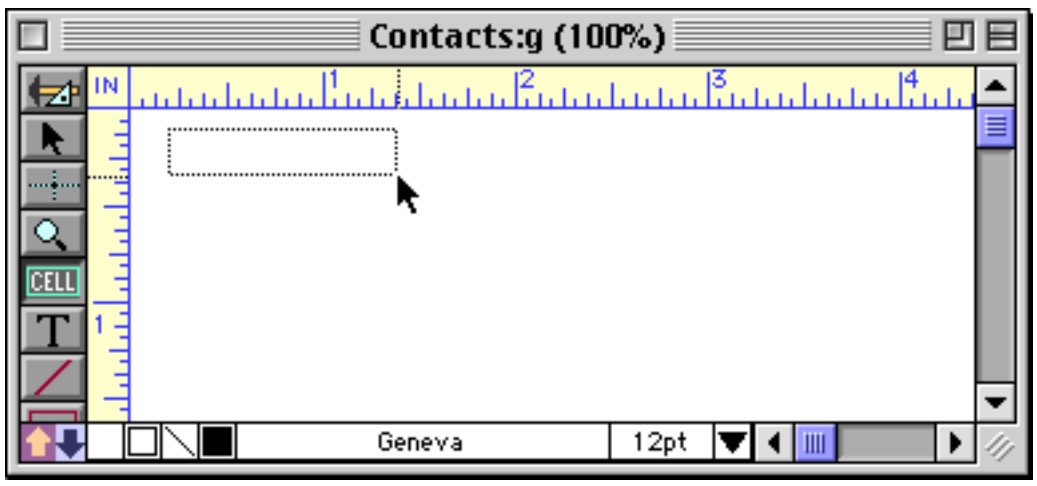
Feature	Data Cell	Text Editor SuperObject
Operation	Edit in pop-up Input Box (similar to data sheet)	Edit directly in form window
Expandable Editing Area?	Yes	No
Double Click before Editing?	Yes	No
Edit Fields?	Yes	Yes
Edit Variables?	No	Yes
Optional Borders?	No	Yes
Custom Object Pattern?	Yes	No

Working with Data Cell Objects

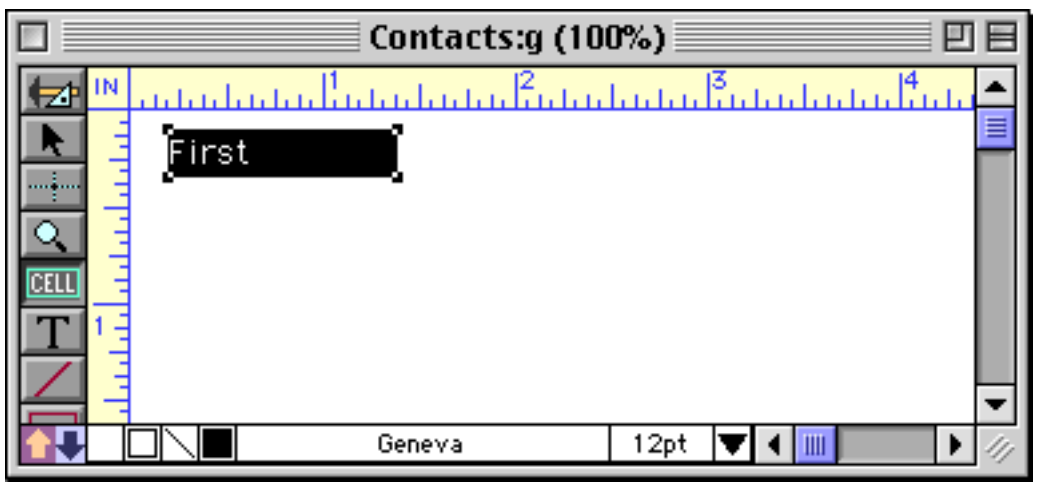
Data cells are created with the **Data Cell** tool. To create a data cell, start by selecting this tool.



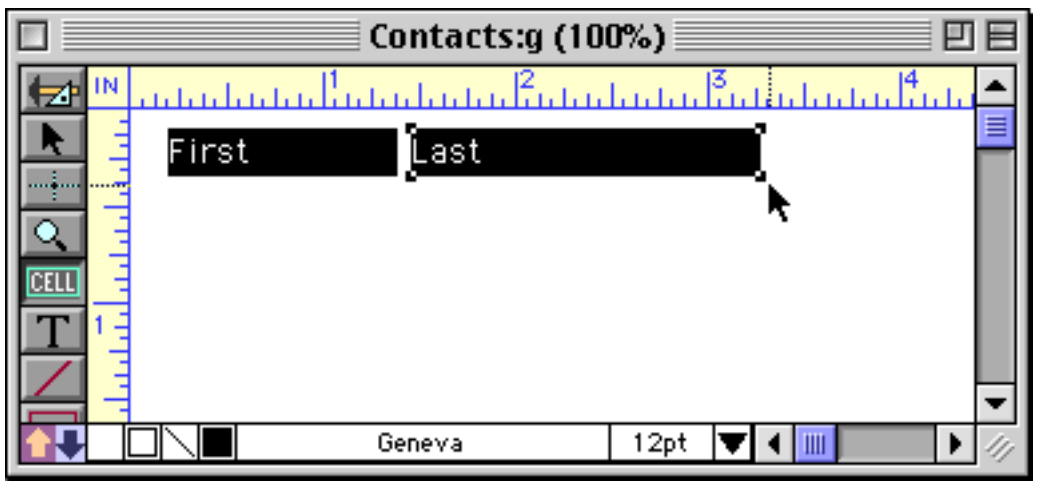
Next drag the mouse across the surface of the form. It's just like creating a rectangle.



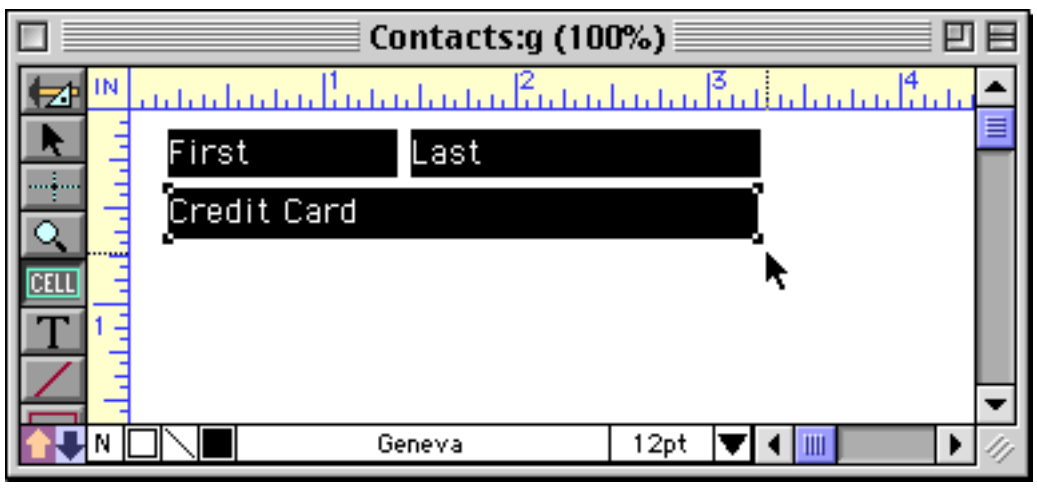
When you release the mouse, Panorama automatically assigns the first field from the database to the new data cell.



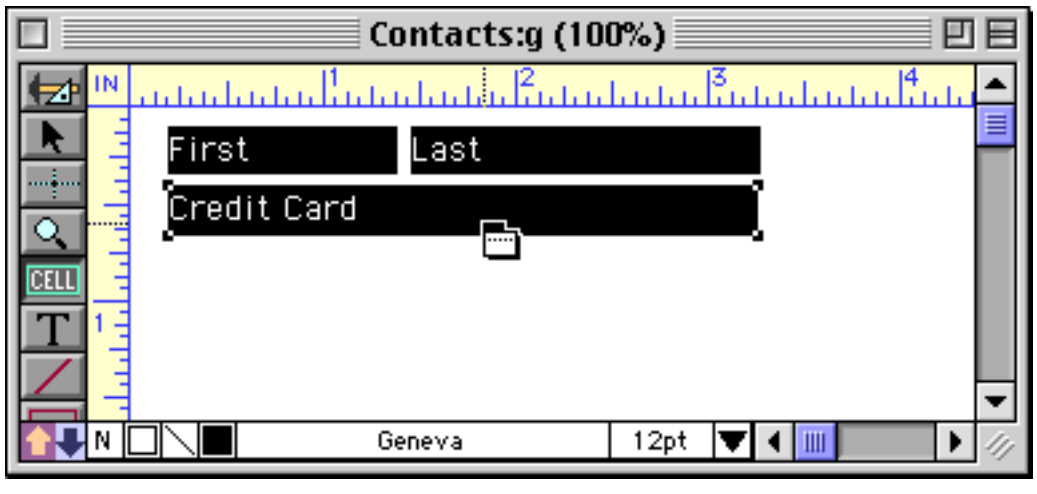
As you create additional cells, each cell is automatically assigned to the next field (using the same order that the fields appear in the data sheet). In this case the second field in the database is named Last.



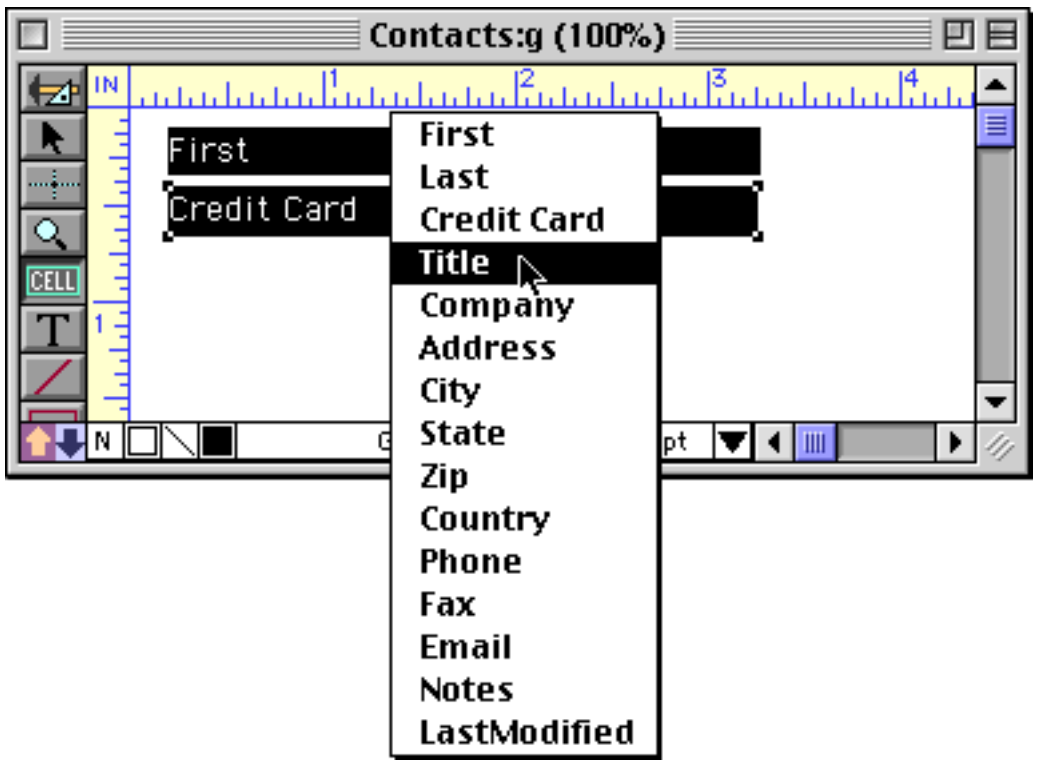
You can continue to add cells to the form. The next field in this database is Credit Card.



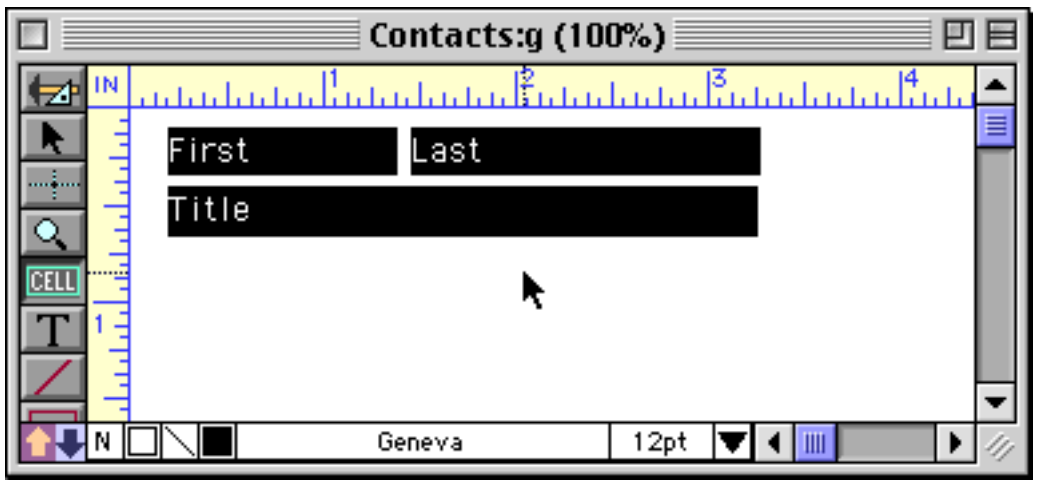
To change the field assigned to a data cell, move the mouse over the cell. The mouse arrow will change to a mini-menu icon.



When you see the mini-menu icon, press the mouse to activate a pop-up menu showing all the fields in the database.

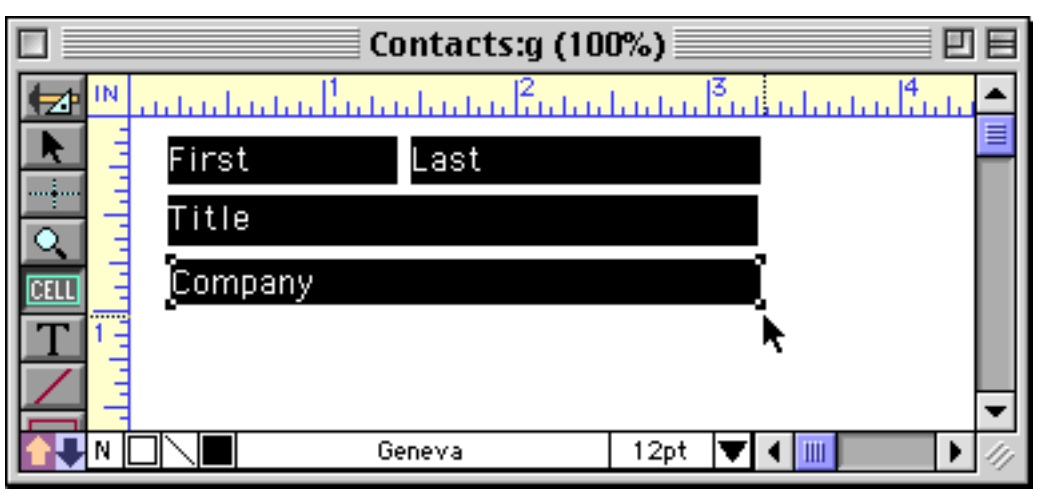


Select the field you want assigned to this data cell and release the mouse.

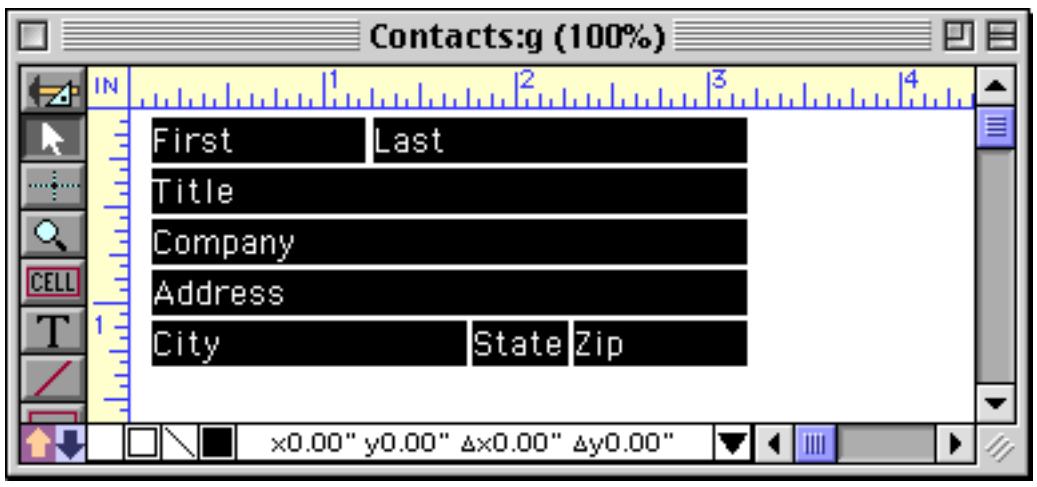


You can use this technique to change the field assigned to any data cell at any time. Remember, however, that you must have the **Data Cell** tool selected. You cannot change the field assignment when any other tool (including the **Pointer**) is selected.

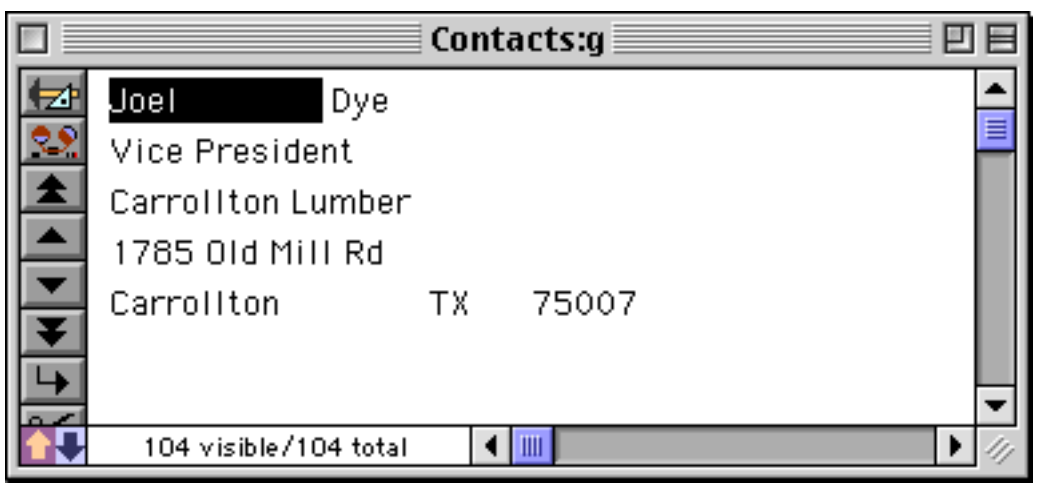
At this point we can continue making additional data cells. The next data cell will be assigned the next field after Title (in this case Company).



When you create data cells by hand like this, the result is likely to be a bit messy. After creating some more cells, we cleaned up this form using a combination of the Dimension dialog (see "Setting Exact Dimensions of Multiple Objects" on page 550) the **Align** dialog (see "Aligning Objects" on page 553), the **Spacing** dialog (see "Adjusting Spacing Between Multiple Objects" on page 556) and nudging with the arrow keys (see "Nudging an Object (or Objects)" on page 509).



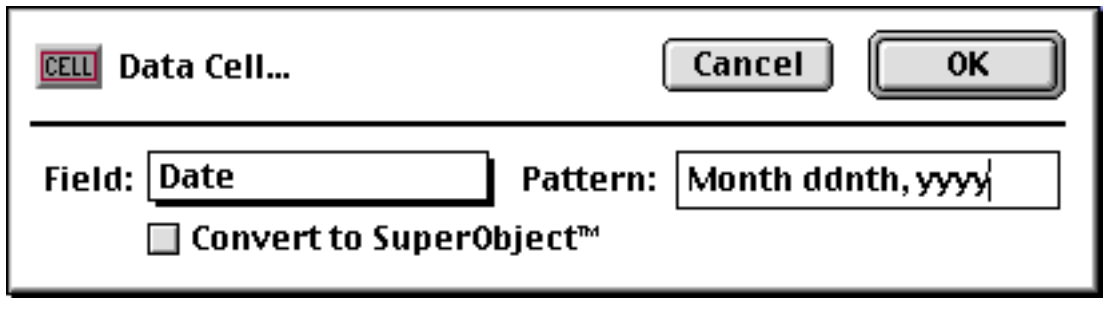
These data cells are ready to use. Simply click on the **Switch To Data Access Mode** tool and you are ready to start typing.



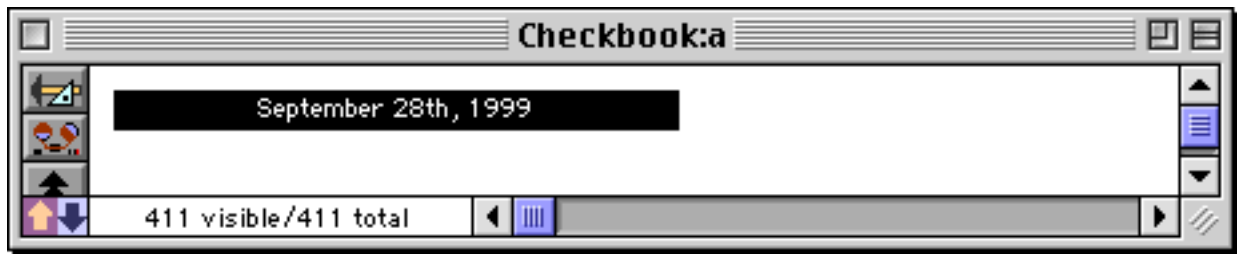
If you need to revise the data cells (or any other form object) later, click on the **Switch to Graphic Design Mode** tool.

Data Cell Custom Output Patterns

Numeric and date data cells are normally formatted using the master output pattern specified in the design sheet (see "Field Properties" on page 250). If you wish, you can override the design sheet output pattern for an individual data cell in the form. To do this, use the pointer tool to select the cell and choose **Output Pattern** from the Text menu. (Alternately, you can simply double click on the data cell.) Type the new output pattern into the dialog.



The data cell will now display the date using the output pattern you typed in.



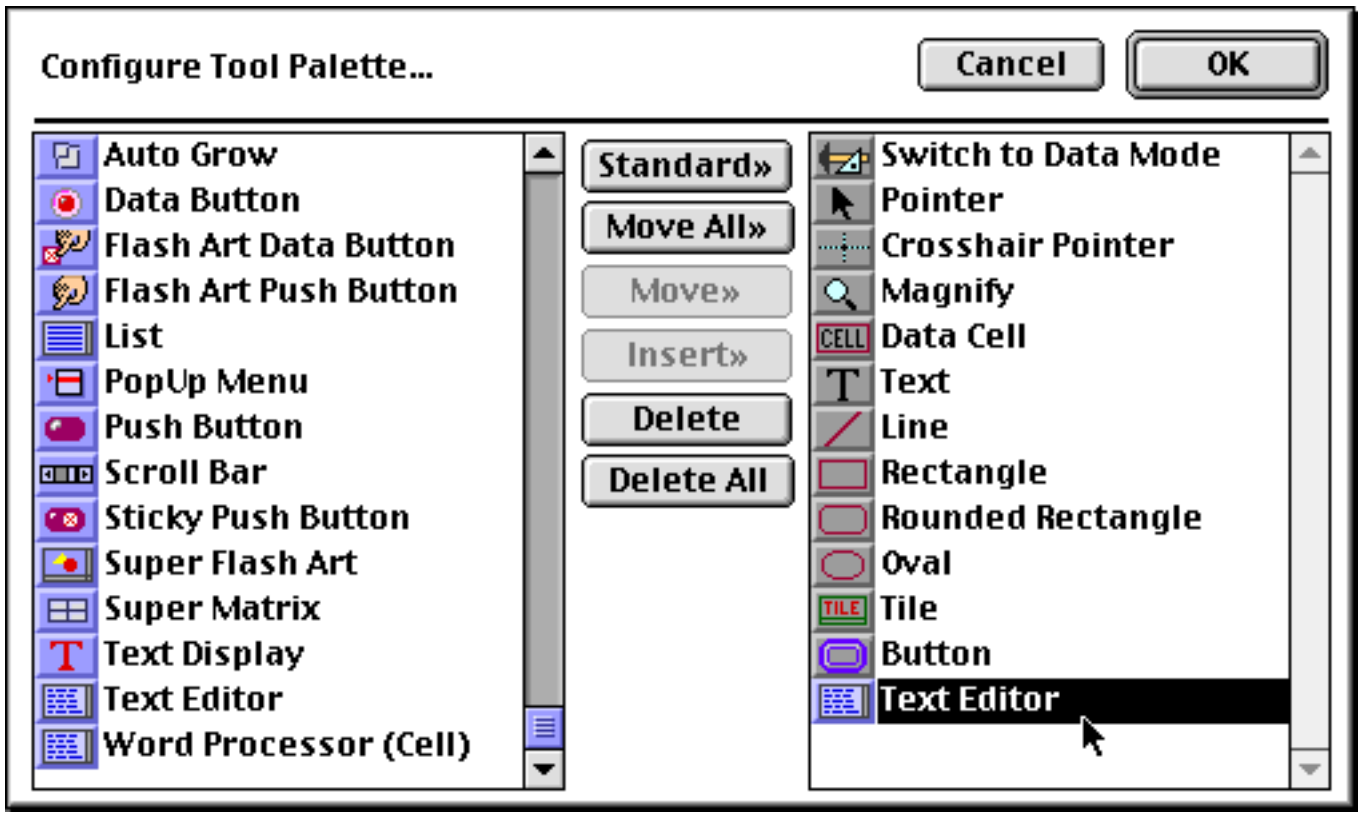
To review output patterns, see “Numeric Output Patterns” on page 290 and see “Date Output Patterns” on page 295.

Text Editor SuperObject

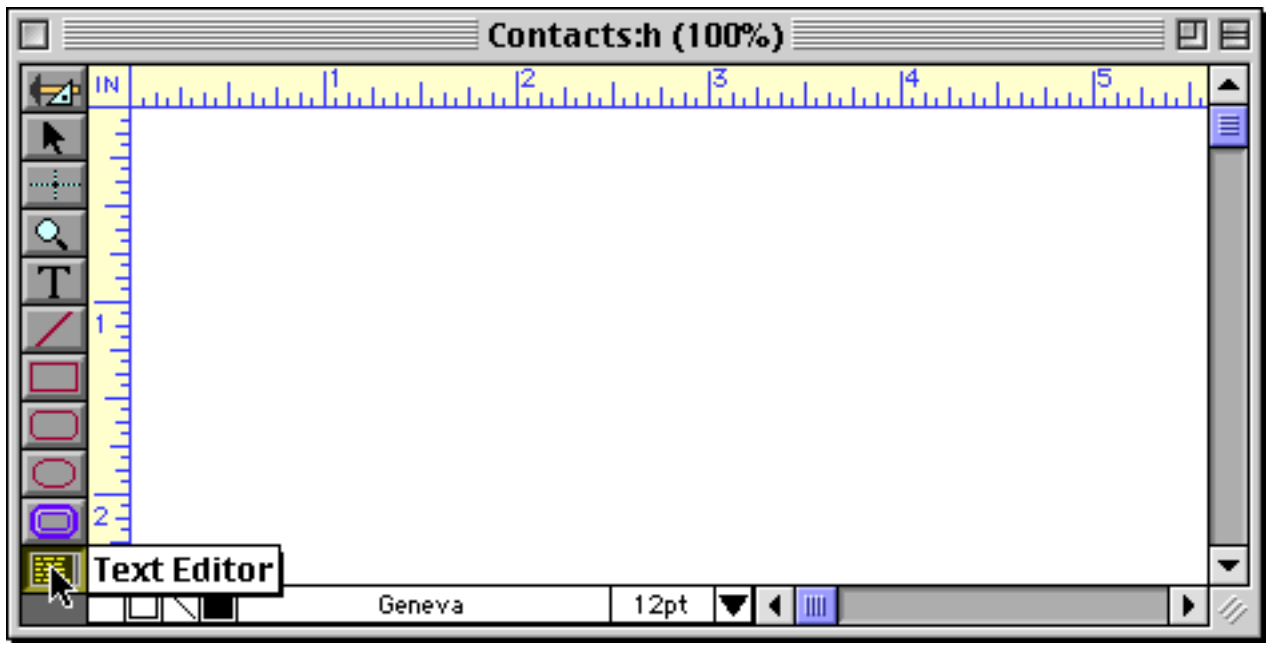
The Text Editor SuperObject is used to edit text in a field or variable. Unlike a data cell, the Text Editor SuperObject does not use a temporary pop-up window for editing. Instead, the user simply clicks and edits the text right in the form window, just as they do with most other applications (see “Types of Data Editing Objects” on page 632). (Of course the down side of this is that the area available for editing is fixed and can't be expanded except by changing the form layout in graphic mode, or with an Auto Grow SuperObject). Another difference from data cells is that Text Editor SuperObjects can edit variables as well as fields. They can automatically draw borders and include one or two scroll bars (or none). (If you wish, you can mix Text Editor SuperObjects with standard data cells objects on the same form).

Creating and Modifying Text Editor SuperObjects

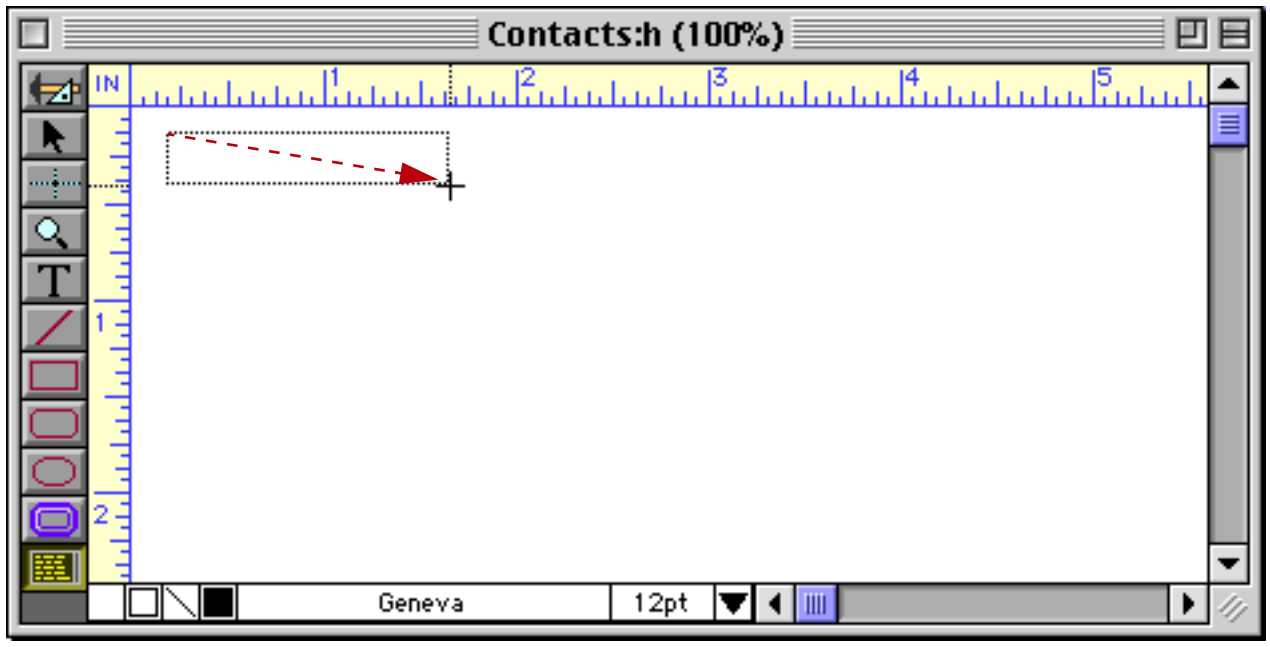
The Text Editor SuperObject tool is not in the default tool palette, so you’ll need to use the **Tool Palette** dialog to add this tool to the palette if it is not already there (see “Customizing the Tool Palette” on page 497).



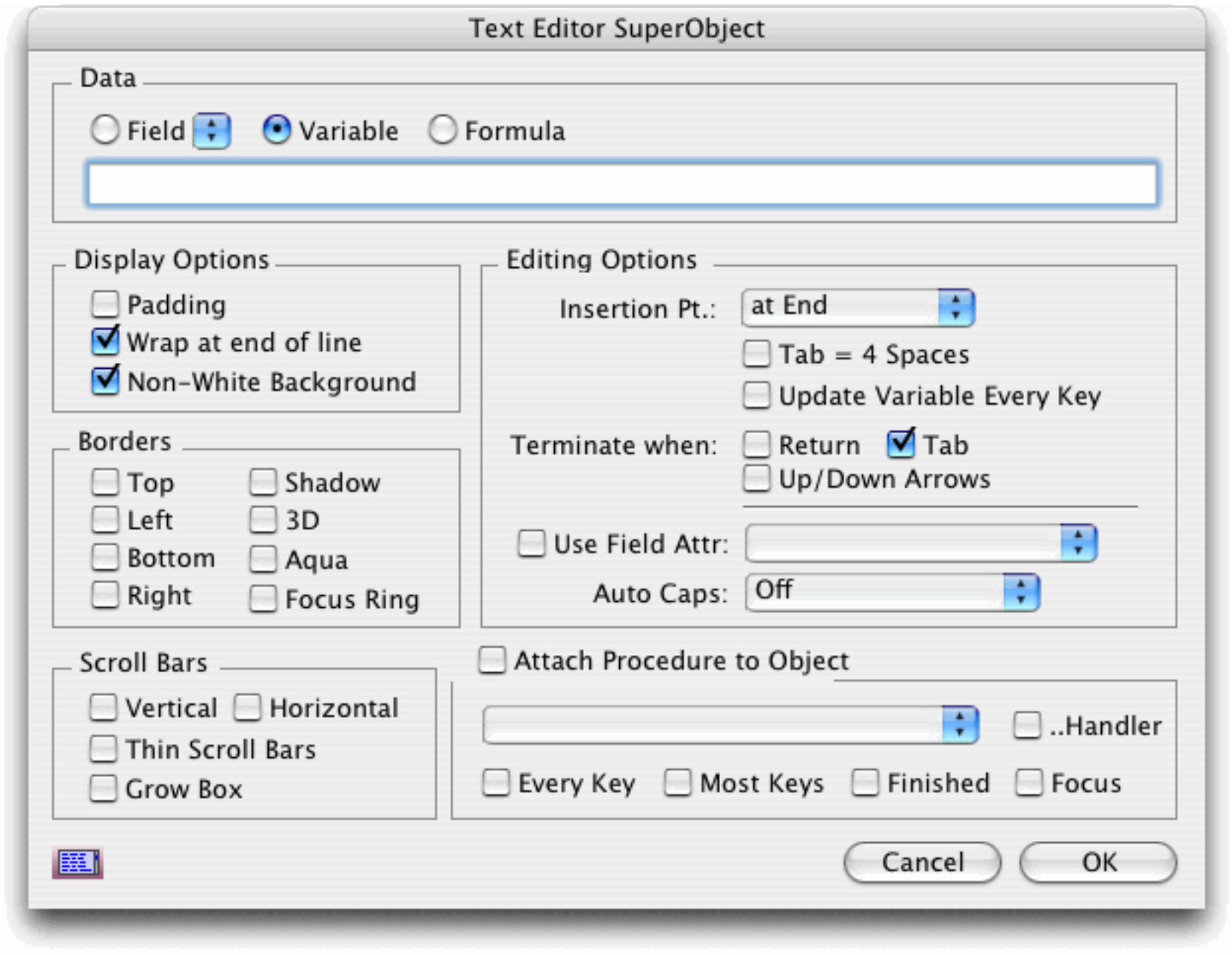
Now that the tool is added to the palette you can select it.



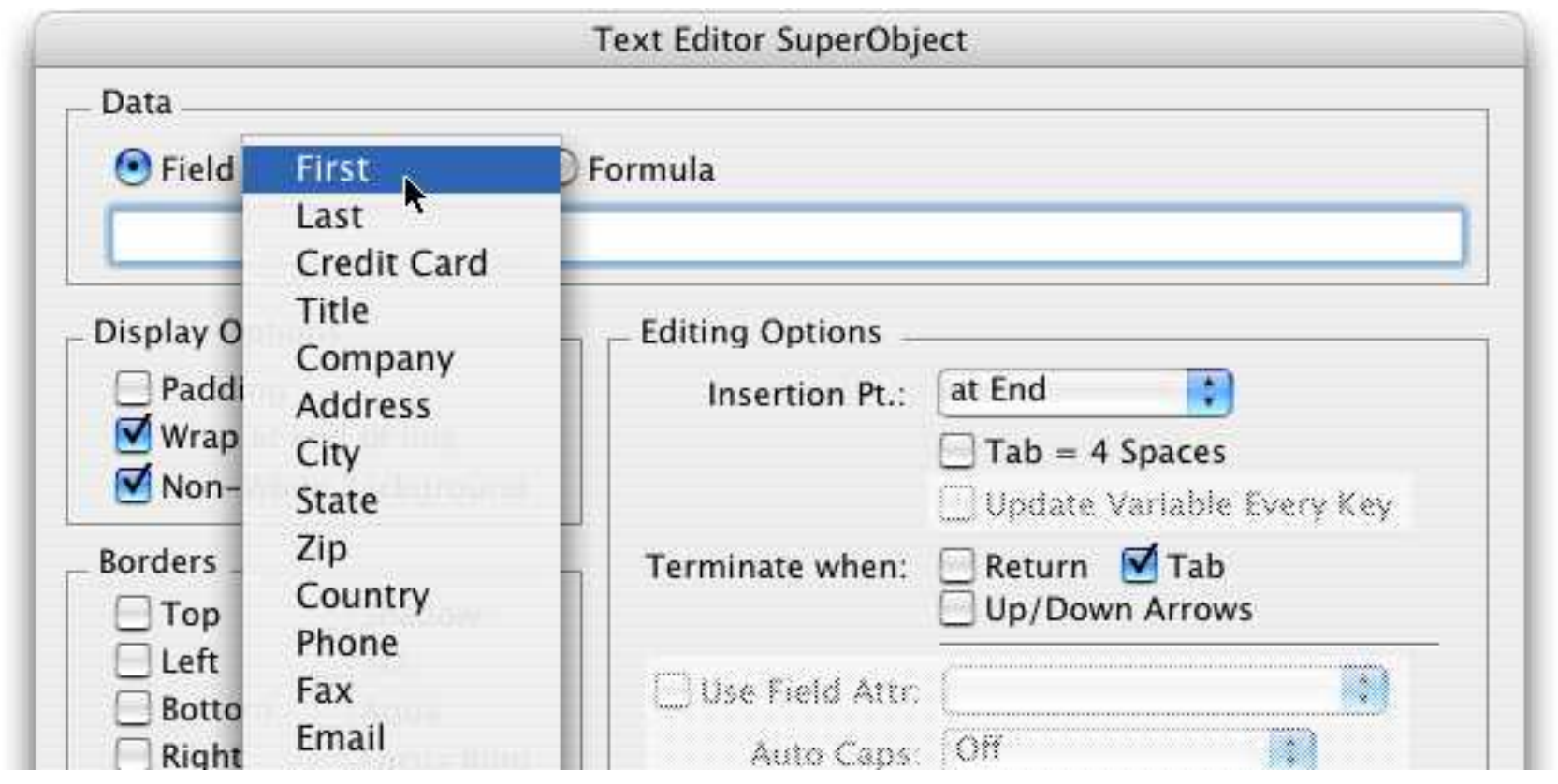
Once the tool is selected, drag the mouse across the form in the location where you want to create the Text Editor SuperObject.



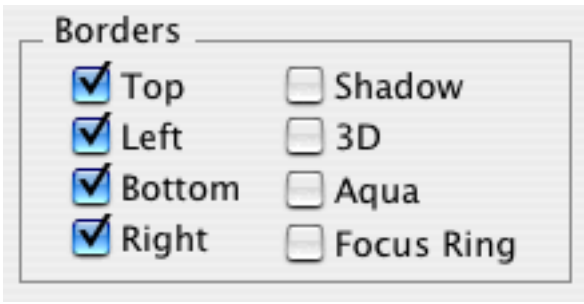
When you release the mouse, the Text Editor SuperObject configuration dialog will appear.



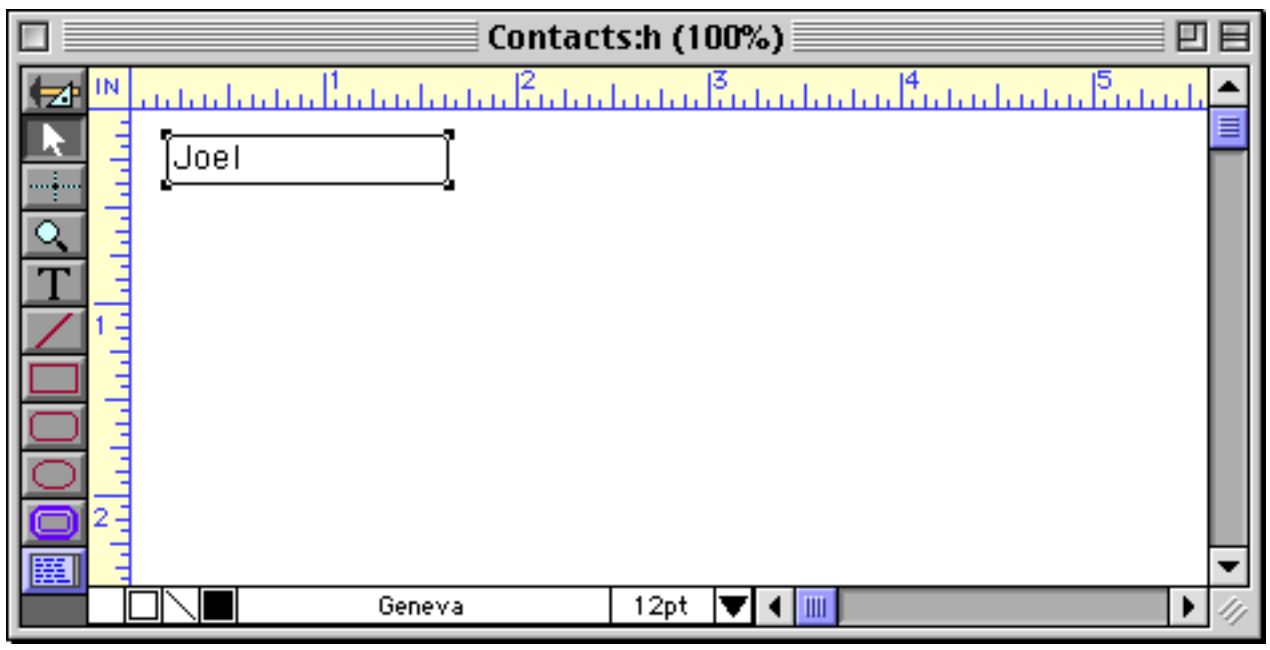
At a minimum you must enter a field name, variable or formula into the dialog. You can use the pop-up menu to select a field.



For this example we've also turned on the Borders options (all of the available options are discussed in detail in the next section).



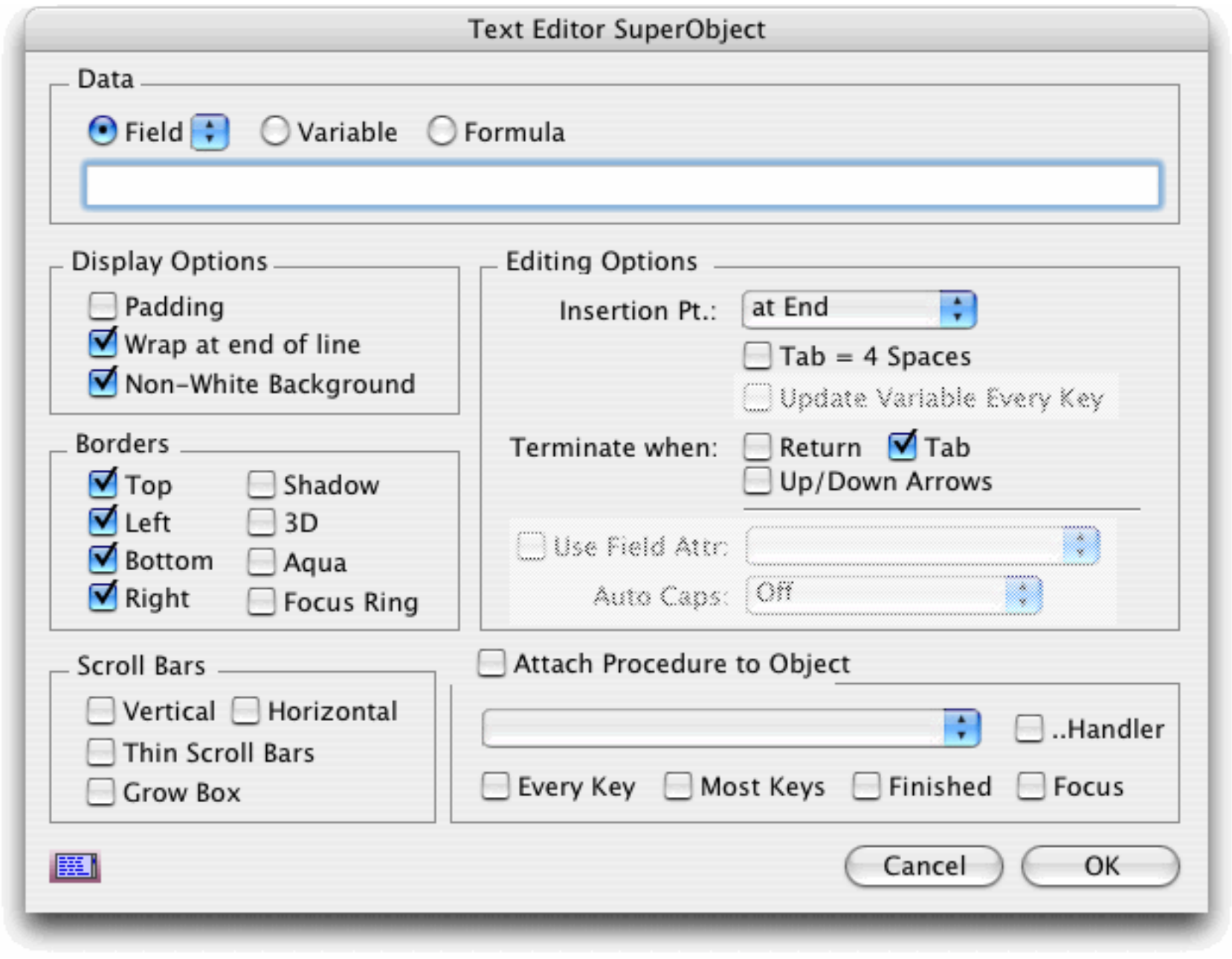
When the OK button is pressed the new object appears. (Notice that unlike the data cell object, the Text Editor SuperObject shows the actual data in both Graphics Mode and Data Access Mode, not just Data Access Mode.)



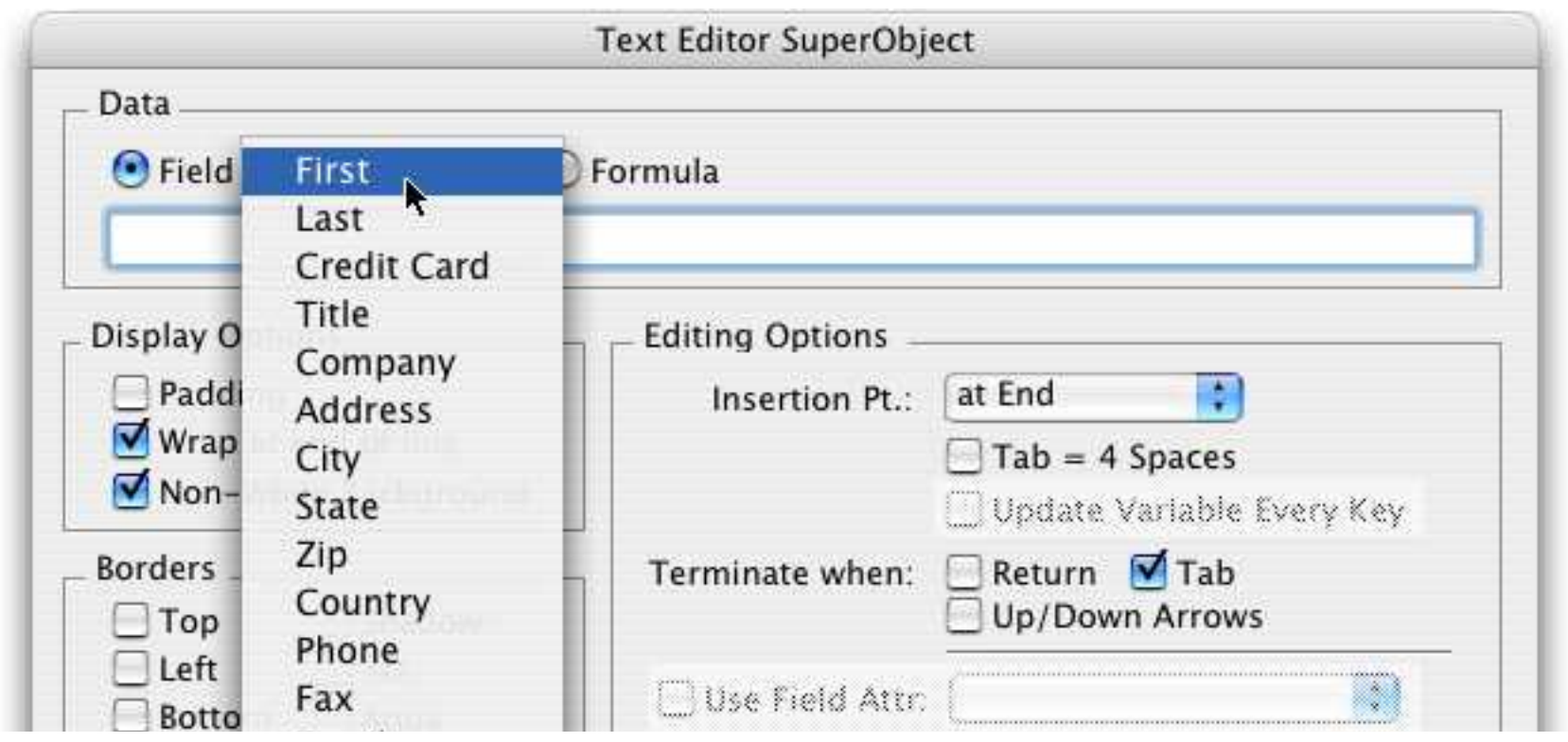
After it has been created you can modify the location, size, font, style and color of a Text Editor SuperObject just like any other object. To change any of the object attributes (scroll bars, border, formatting etc.) select the **Pointer** tool and double click on the object. The configuration dialog will appear again. Make your changes and press the **OK** button.

Text Editor Options

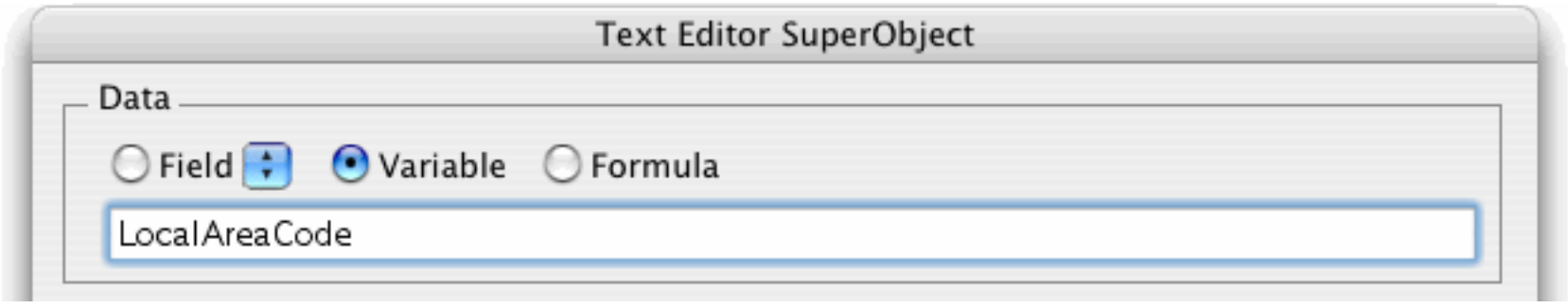
The SuperObject Text Editor Properties dialog is divided into several sections.



Data: Each Text Editor SuperObject edits a single data item, which may be a database field, a variable, or a formula. To edit a field, type in the name of the field or select the field name from the pop-up menu.



A **variable** is a place to store information independently of any database. The primary use for variables is as temporary storage for procedures (see “Variables” on page 1199 and “Variables” on page 1370). A variable can also be created and edited by a Text Editor SuperObject. Simply click on the Variable option and type in the name of the variable.

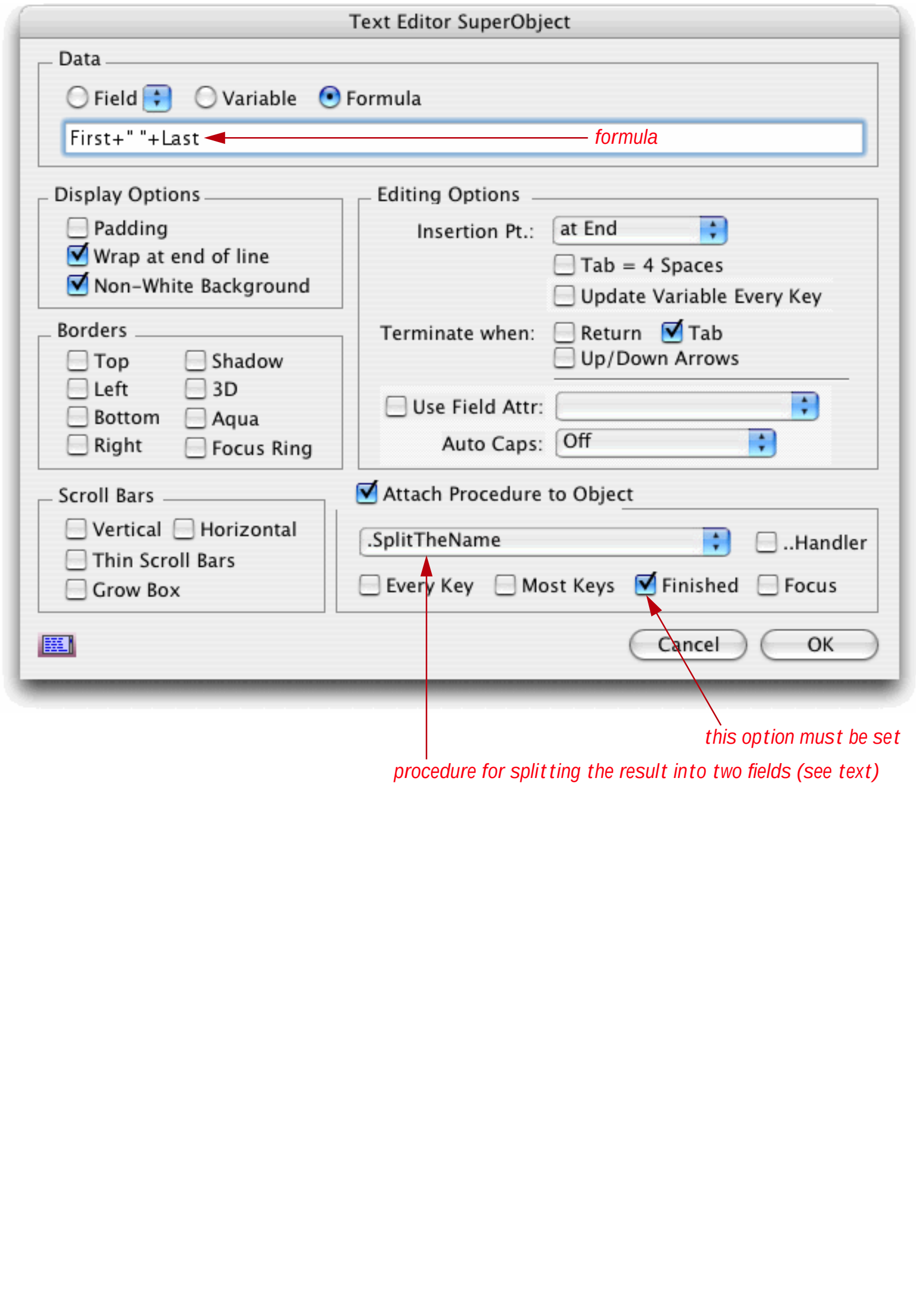


If you specify a variable that has not already been created with a procedure, Panorama will automatically create a global or fileglobal variable with this name the first time the form is displayed. (The default is a global variable unless the FileGlobal Variables option is set in the **Form Preferences** dialog, see “Creating Variables with a SuperObject” on page 1374.) Once it has been created this variable can be used in formulas and procedures, just like any other variable. For example, suppose that your database has fields named AreaCode and PhoneNumber. The short procedure listed below will check the area code against the local area code entered by the user in the Text Editor SuperObject. If it is a local call, only the local number is dialed. If the number is out of the local area, Panorama will dial 1 plus the area code plus the phone number. (See “Writing a Procedure from Scratch” on page 1352 to learn how to create a procedure. See “The ? Function” on page 1271 and “DIAL” on page 5167 for more information about the specific statements and functions used in this procedure.)

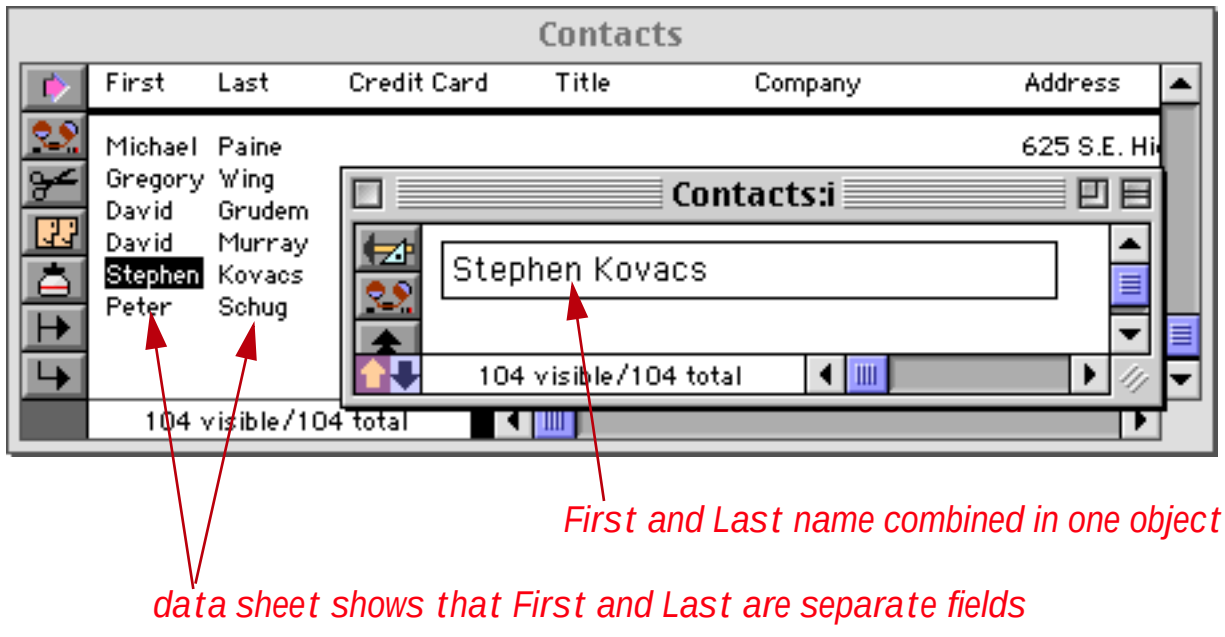
```
dialdigits ?(AreaCode=LocalAreaCode,PhoneNumber,"1"+AreaCode+PhoneNumber)
```

For an application like this you would probably want to make LocalAreaCode a permanent variable so that you don’t have to re-enter it every time you open the database. See “Creating a Variable” on page 1370 to learn how to create a permanent variable.

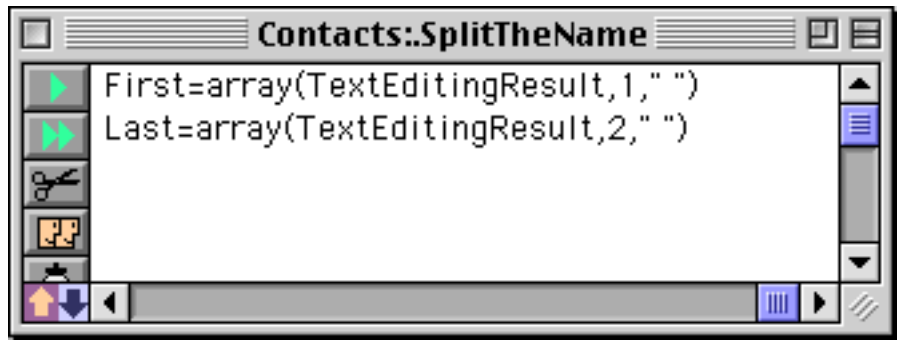
The final data option is Formula. If you select this item the Text Editor SuperObject is not editing a real data item, but a temporary data item created "on-the-fly" using a formula. To illustrate this capability we'll create an object that edits two database fields (first and last names) in a single Text Editor SuperObject. Select the Formula option and type in the formula — in this case a formula that combines the first and last names.



This illustration shows the finished object (in the front window). As you can see, the object combines the two separate fields into just one object.

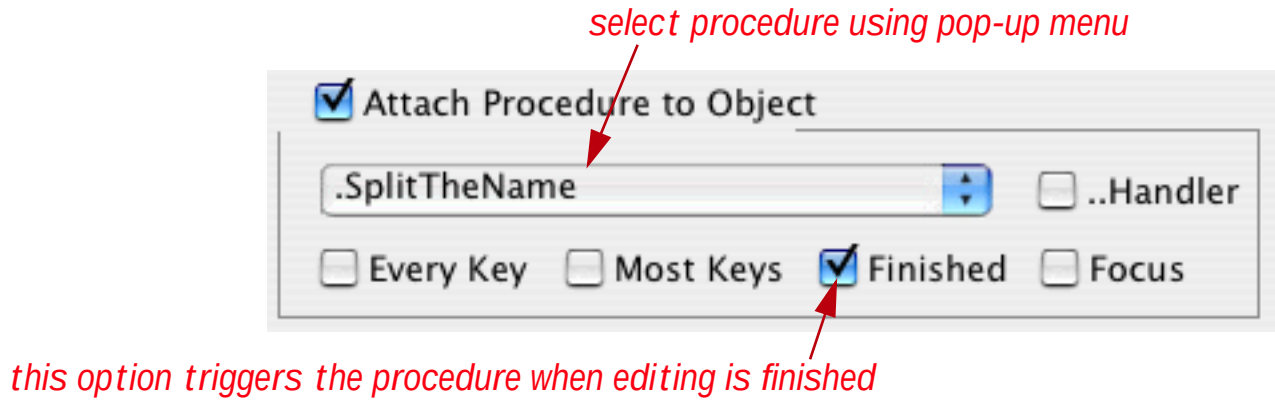


There's only one problem — if you edit the name, how does Panorama know where to put the edited text? You can't store text in a formula! Since it doesn't know where else to put it, Panorama places the edited text in a variable called TextEditingResult. If you want to store this text somewhere else you'll need to create a procedure that takes this variable and stores the data somewhere (see "Writing a Procedure from Scratch" on page 1352 to learn how to create a procedure). The procedure can have any name you like. For this example we have created a procedure called `.SplitTheName`.

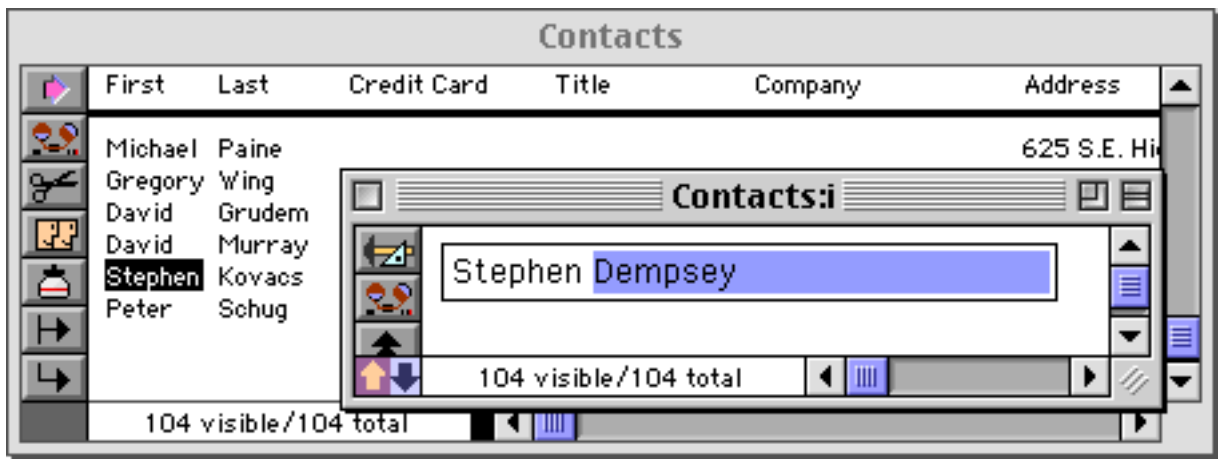


This procedure uses the `array(` function (see "Text Arrays" on page 1236) to take the edited text and split it back into separate first and last names. Panorama has several statements and functions that are very handy for splitting data into multiple components, see "Natural Data Display" on page 1653, "Taking Strings Apart (Text Funnels)" on page 1214 and "Text Arrays" on page 1236.

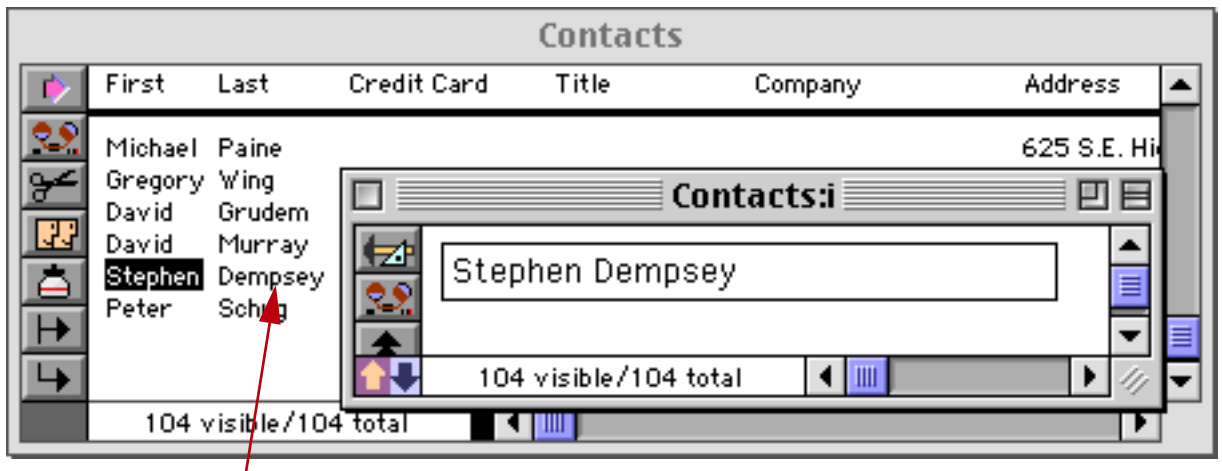
Once the procedure has been created you must select it in the Text Editor SuperObject configuration dialog (see below). You also need to make sure that the Finished option is turned on. The Finished option tells Panorama to trigger the procedure when editing is finished (when the **Enter** key is pressed or when you click on another object).



Once the procedure is set up and configured in the dialog you can use the Text Editor SuperObject to edit the name. In this case the last name is being changed from Kovacs to Dempsey.

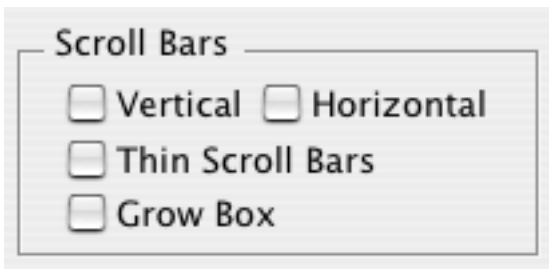


When the **Enter** key is pressed the First and Last name fields in the database are updated (in this case only the last name actually changed).



As you can see, it takes a bit of work to set up the Formula option, but it does give you the power to set up a user interface that is independent from the data sheet structure.

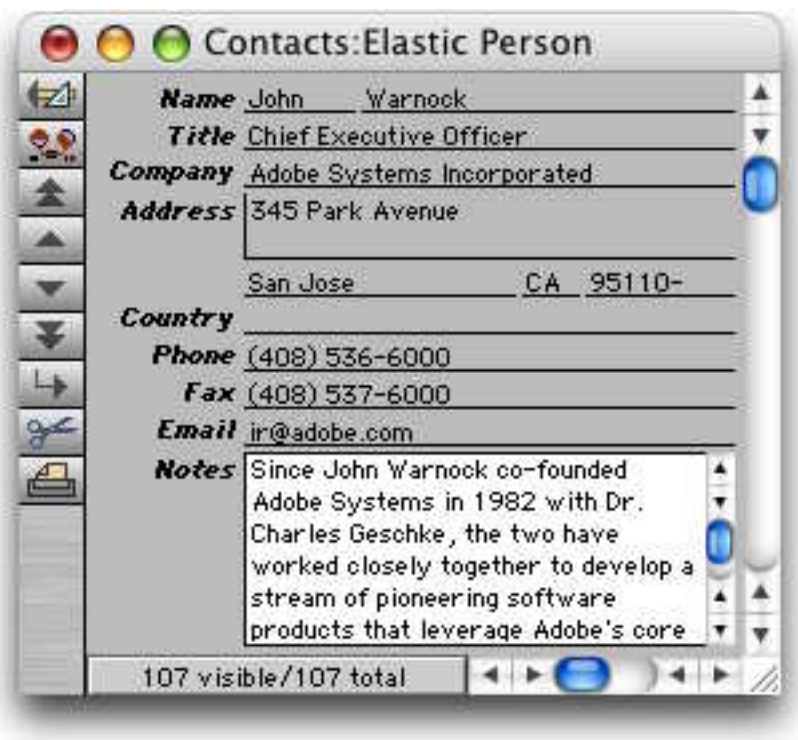
Scroll Bars: This section controls what scroll bars (if any) will be available when editing this text.



Here is a Text Display SuperObject with the Vertical Scroll Bar enabled.



The Thin Scroll Bar option makes the scroll bar (or bars) 11 pixels wide instead of 16.



Here is the same object with both Vertical and Horizontal scroll bars.



With the Vertical and Grow Box options turned on, Panorama leaves an empty spot for a grow box in the lower right hand corner.



Elsewhere in this manual you can learn how to turn off the form's scroll bars (see "Window Options" on page 226) and how to add your own custom grow box (see "Elastic Forms" on page 912), to make a final window that looks like this.



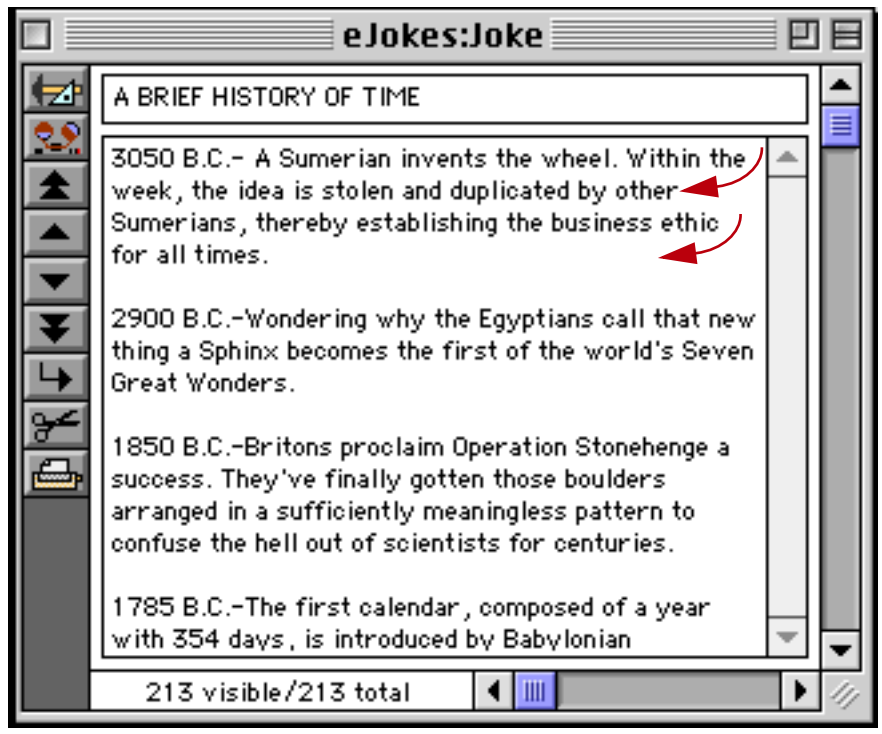
Padding: When this option is checked extra padding appears around the top, left and right sides of the text. Here is an example of an object **without** padding.



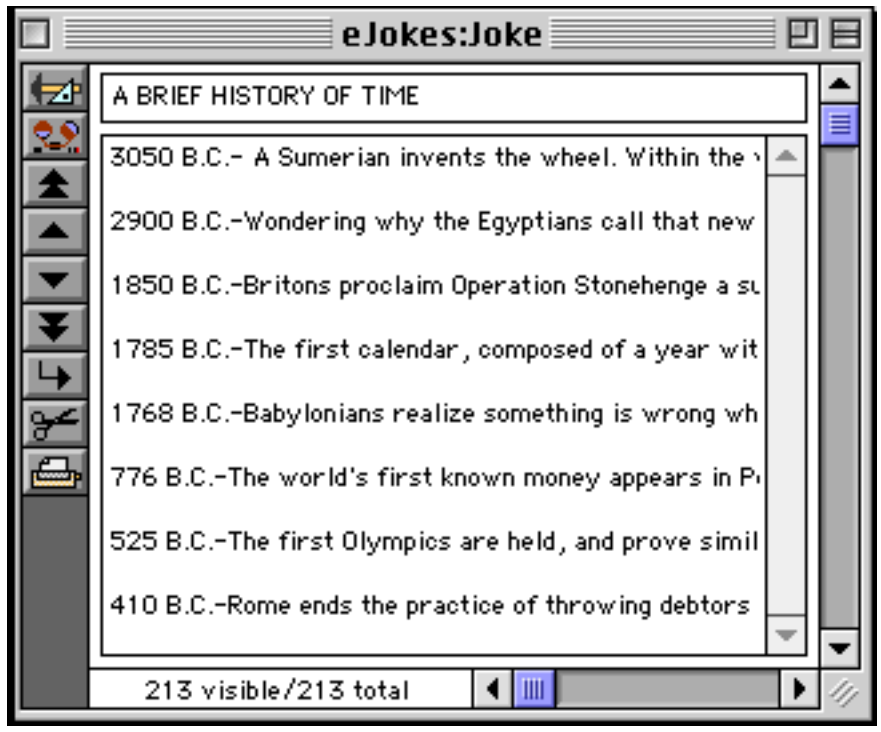
Here is the same object **with** padding.



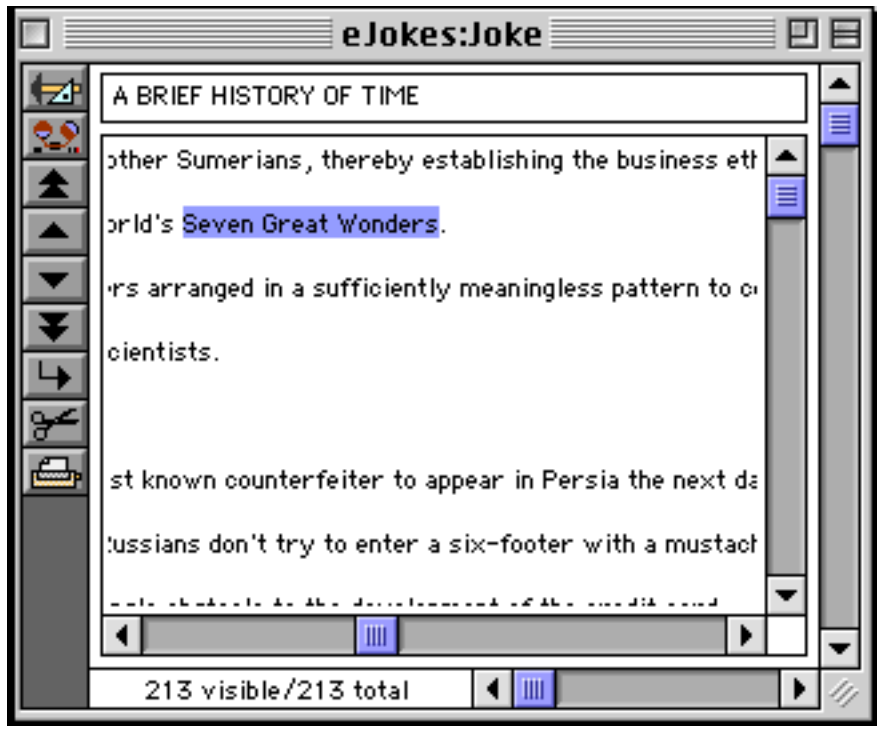
Wrap at End of Line: If text being edited is too long to fit on a single line, it will usually "wrap" around to the next line.



However, if the Wrap at End of Line checkbox is turned off, the text will not wrap. Instead, the text will continue off the right edge.



If the horizontal scroll bar is enabled, you can scroll over to see the rest of the text.



Non-White Background: We recommend that you use this option if the Text Editor SuperObject is placed over a color (non-white) background. If this option is turned on, Panorama will temporarily display a white background behind the text while it is being edited.



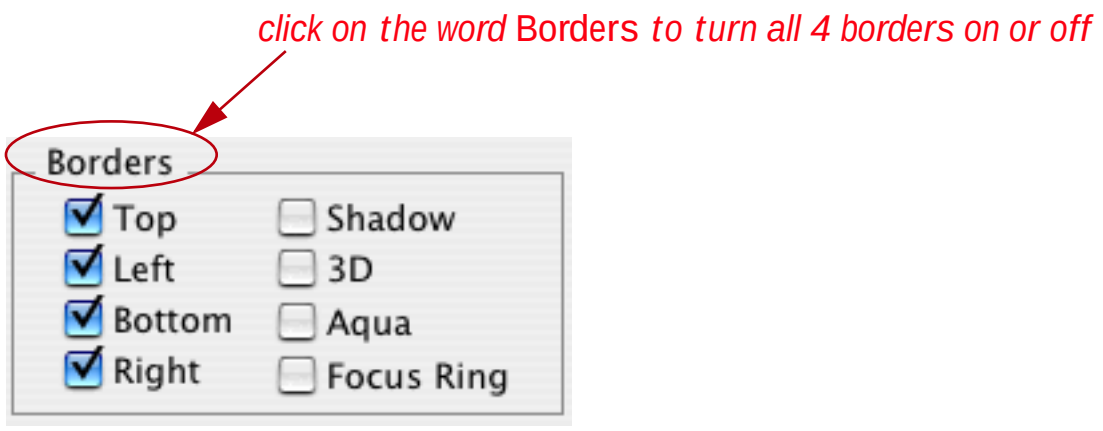
If you don't use this option, you'll find that portions of the background will turn white as you edit anyway. the result is ugly and possibly confusing.



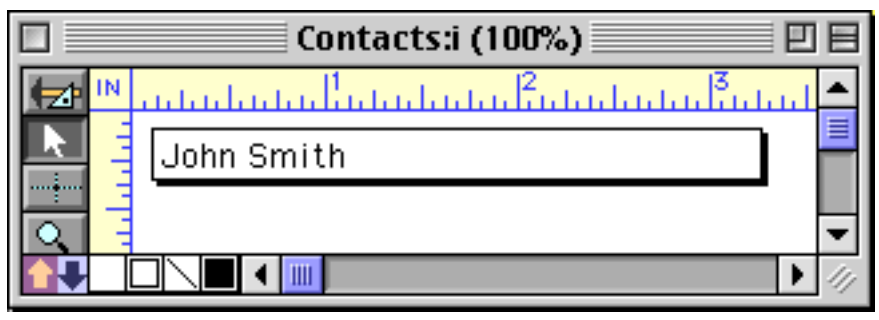
I think you can see why we recommend using the Non-White Background option!

Terminate When: You can always press the **Enter** key when you have finished editing a data item. Depending on the options set in this dialog other keys may also cause editing of this cell to finish, including the **Return**, **Tab**, **Up Arrow**, or **Down Arrow** keys. (Note: If you want to be able to tab from this item to the next, be sure to select the **Tab** key as one of the keys that causes termination. On the other hand, if you want to be able to use the **Tab** key inside this field or variable, Tab should not be checked.)

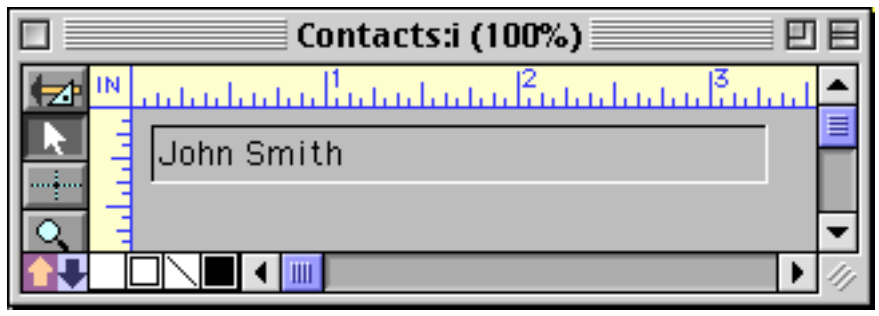
Borders: The options in this section control the borders that are displayed around the text (if any). You may separately control the top, bottom, left, and right borders or click on the word **Borders** to turn all four on or off at once.



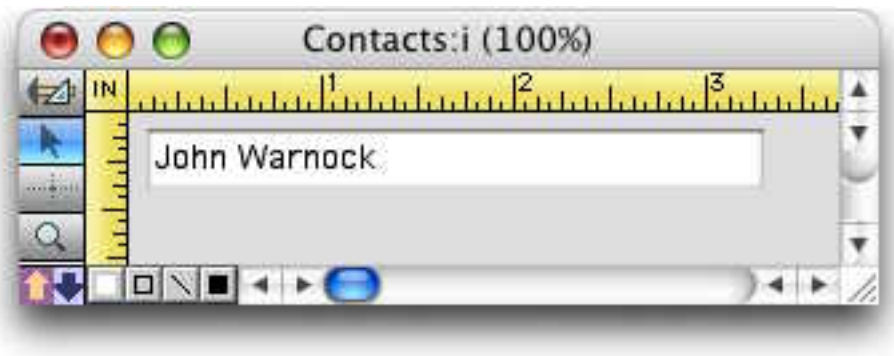
The Shadow option makes a drop shadow appear.



The 3D border effect works best with a light gray background.



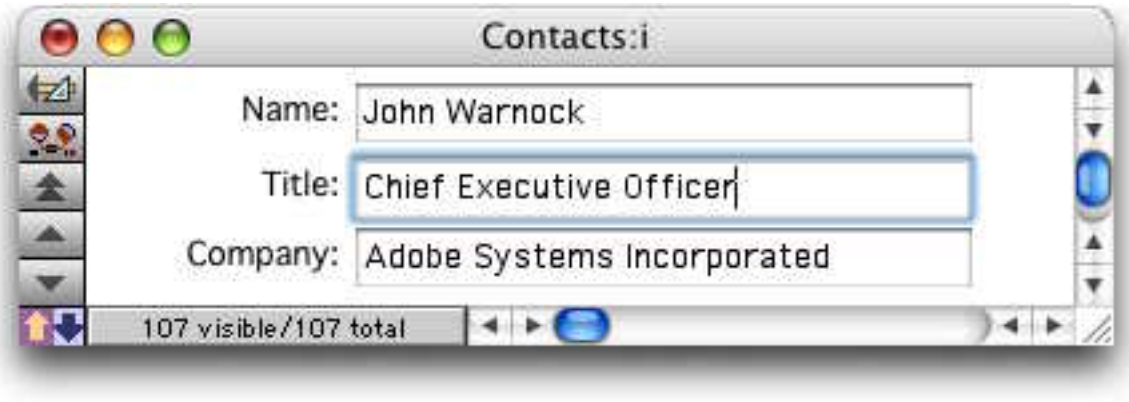
The Aqua border effect displays the same soft 3D borders that are used by most OS X applications. This option can work with a light gray or white background.



When the Focus Ring option is checked, Panorama will display a blue ring around the object when it is being actively edited.

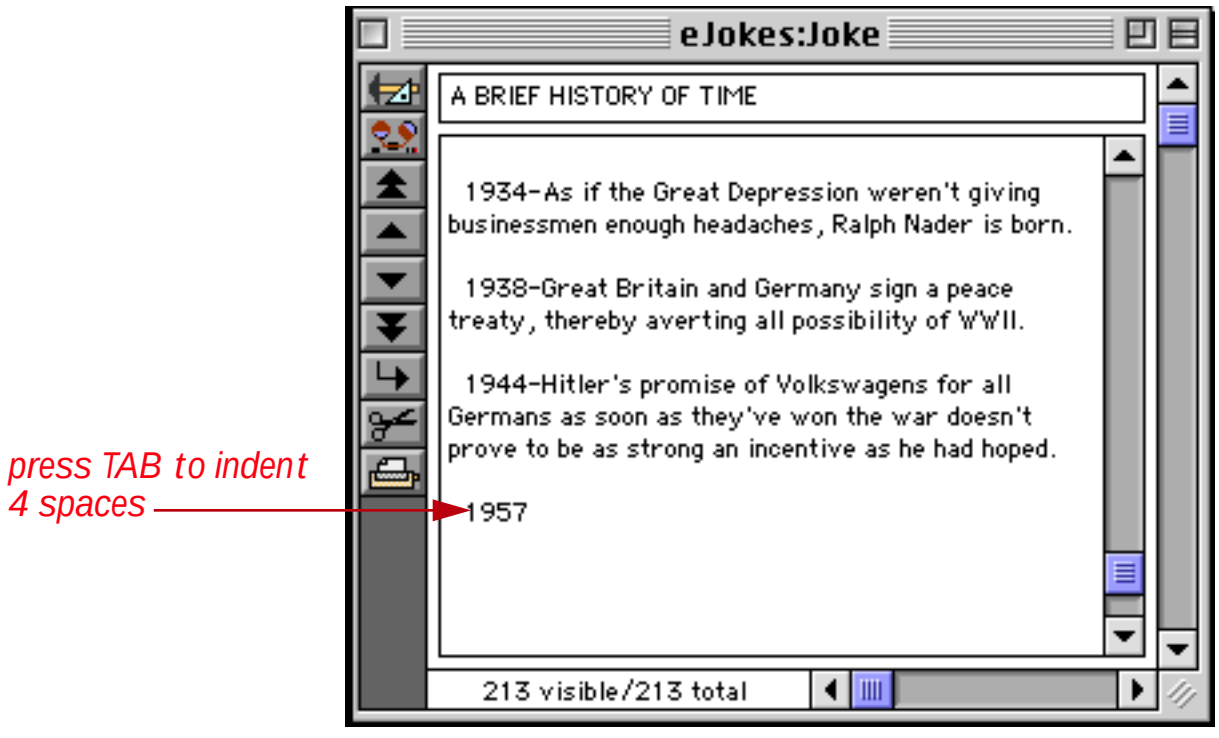


When you shift to editing a different object, the blue ring will move to this new object.



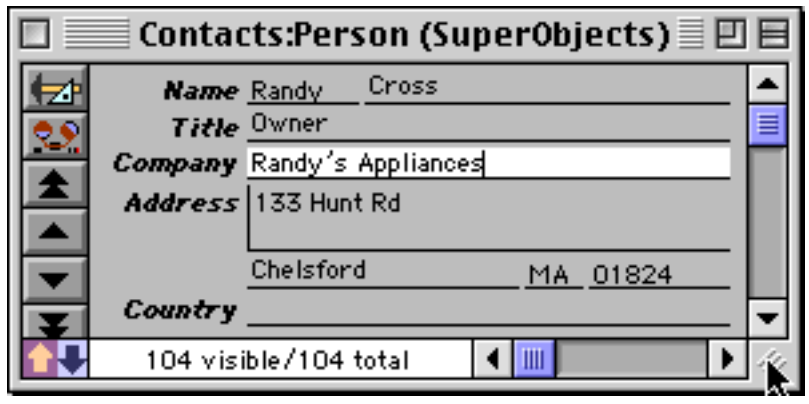
(Note: You can control the background color for the entire form with the **Form Preferences** command in the Setup menu. See “Form Background Colors” on page 582.)

Tab = 4 Spaces: If this option is enabled, pressing the **Tab** key will be the same as pressing the **Space Bar** four times.

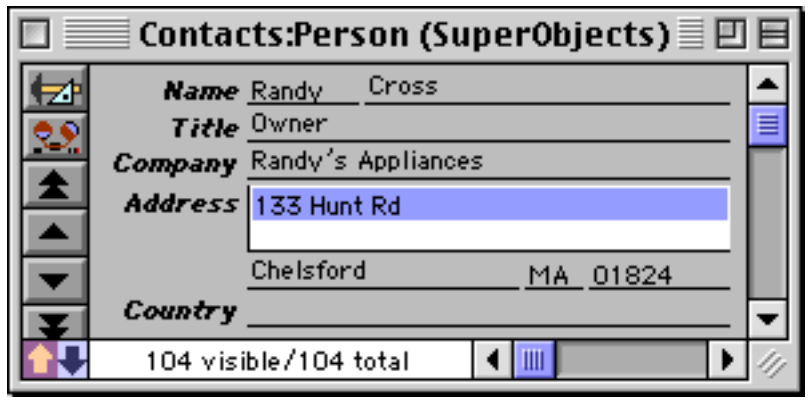


If this option is selected, the Tab checkbox (part of the Terminate line) should be turned off.

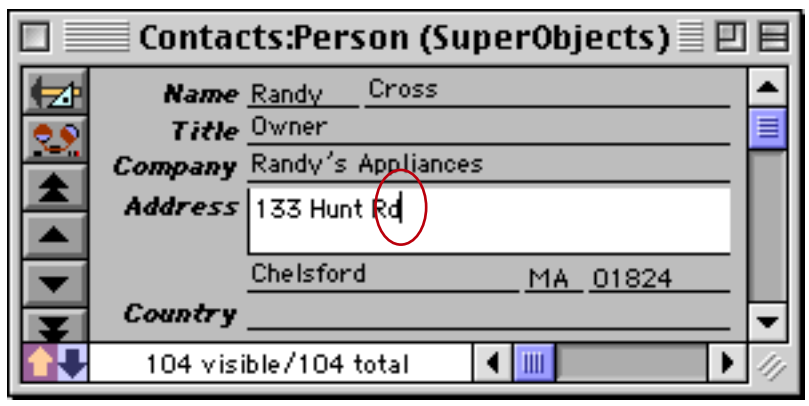
Insertion Point: This option gives you the choice of what text should be selected when you tab into this Text Editor SuperObject. (Of course when you click into a Text Editor SuperObject, the insertion point goes where you click.) The three options are: At End, At Start, and ALL (which selects all the text). In the example above, the Address object has the Insertion Point option set to ALL.



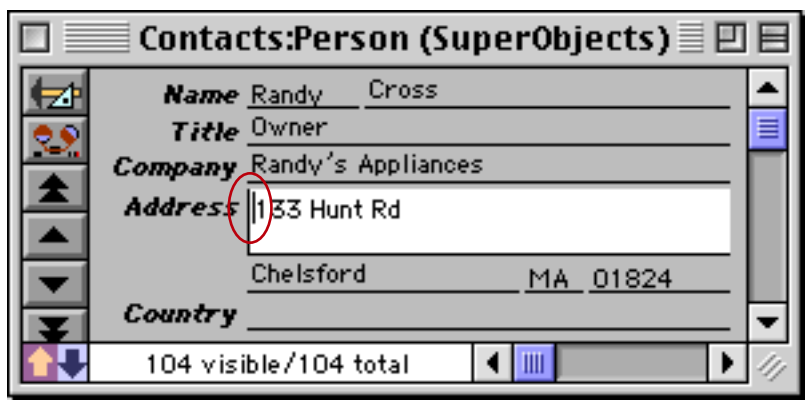
When you **Tab** from the Company into the Address field, all of the text is selected.



If the At End option had been selected, the insertion point would be at the end of the address.

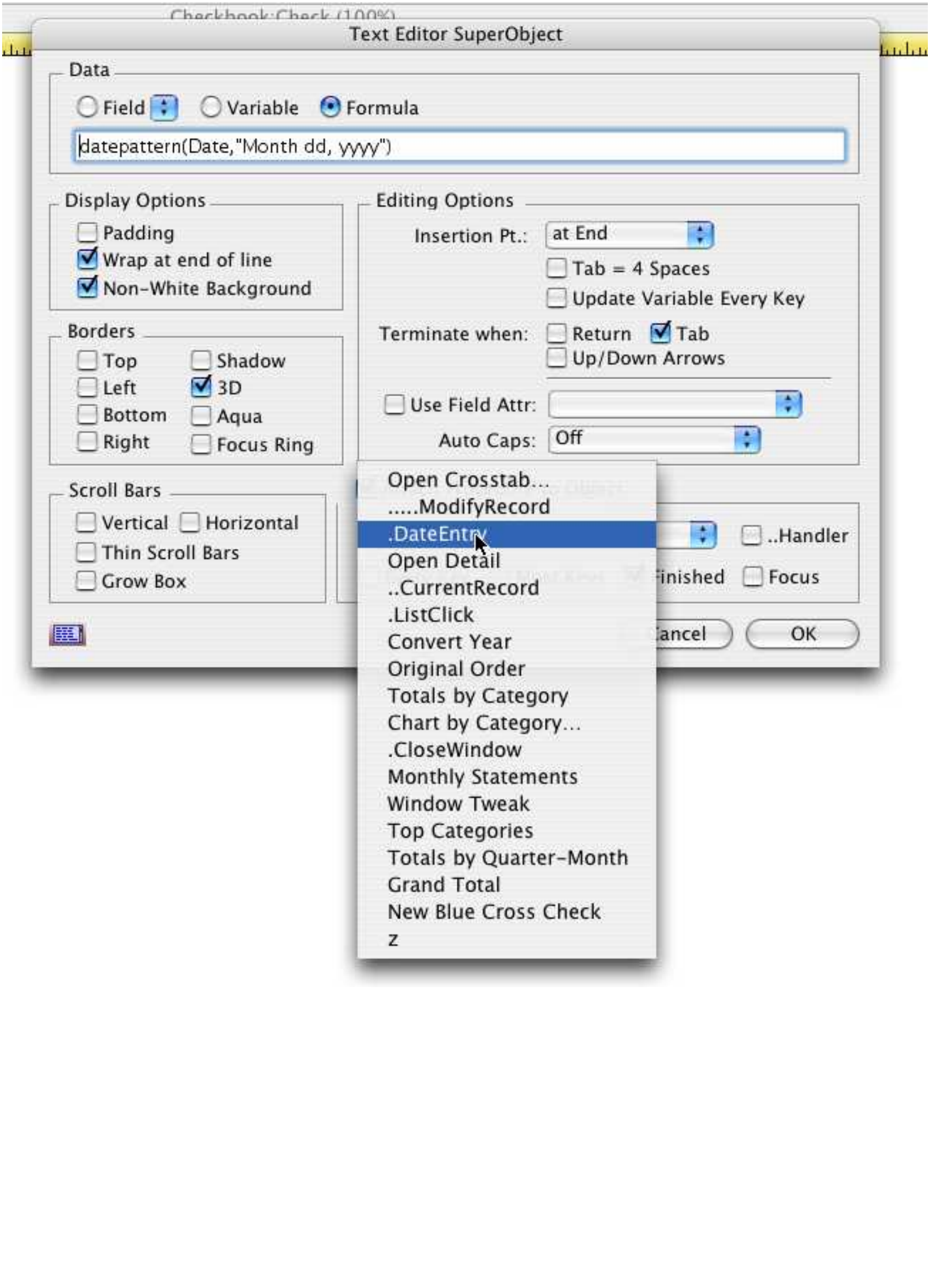


If the At Start option had been selected, the insertion point would be at the start of the address.

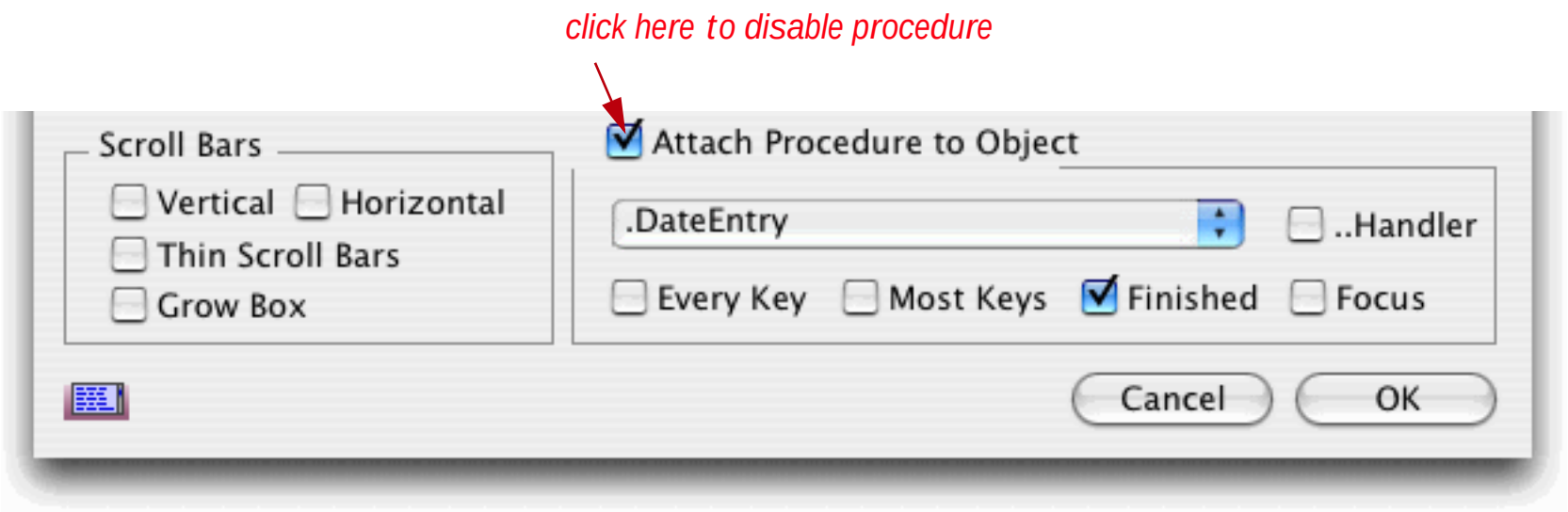


Update Variable Every Key: This option only works when editing variables, not fields. If this option is enabled, the Text Editor SuperObject will update the value of the variable immediately after every key is pressed. Otherwise the variable will not be updated until editing is finished. This mode is especially useful when the Text Editor SuperObject is set up to trigger a procedure after every key or most keys (see next section).

Procedure: This section specifies what procedure is associated with this Text Editor SuperObject (if any) and when that procedure will be triggered. To select or change the procedure associated with this object, use the pop-up menu.



If you later decided to disable the procedure, click on the checkbox (or simply select another procedure).

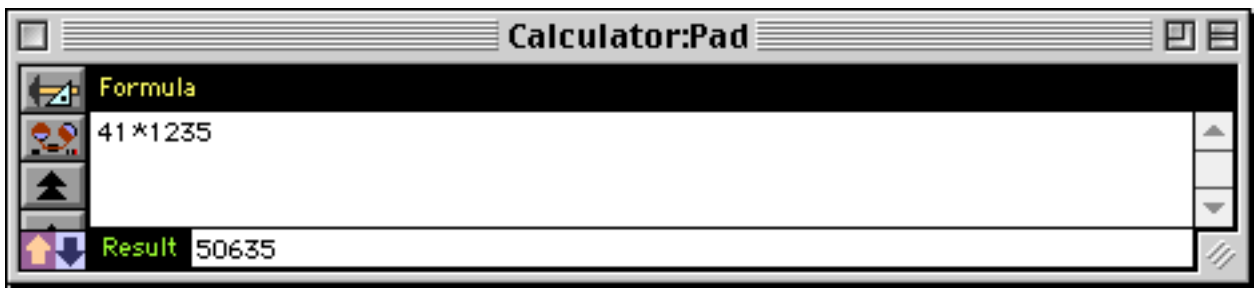


There are four choices for triggering a procedure: Every Key, Most Keys, Finished and Focus. **Finished** simply means that the procedure will be triggered when the user signals that he or she has finished editing the text by pressing **Enter**, **Return**, **Tab**, etc. (Note: If the Text Editor SuperObject is associated with a field, a procedure may also be triggered even if no procedure is assigned in the configuration dialog. If the field has a procedure assigned to it in the design sheet, it will be triggered (see "Data Entry Triggers" on page 1515). If there is no Finish procedure selected and no design sheet procedure, the **.ModifyRecord** procedure (if any) will be triggered (see ".ModifyRecord" on page 1512).)

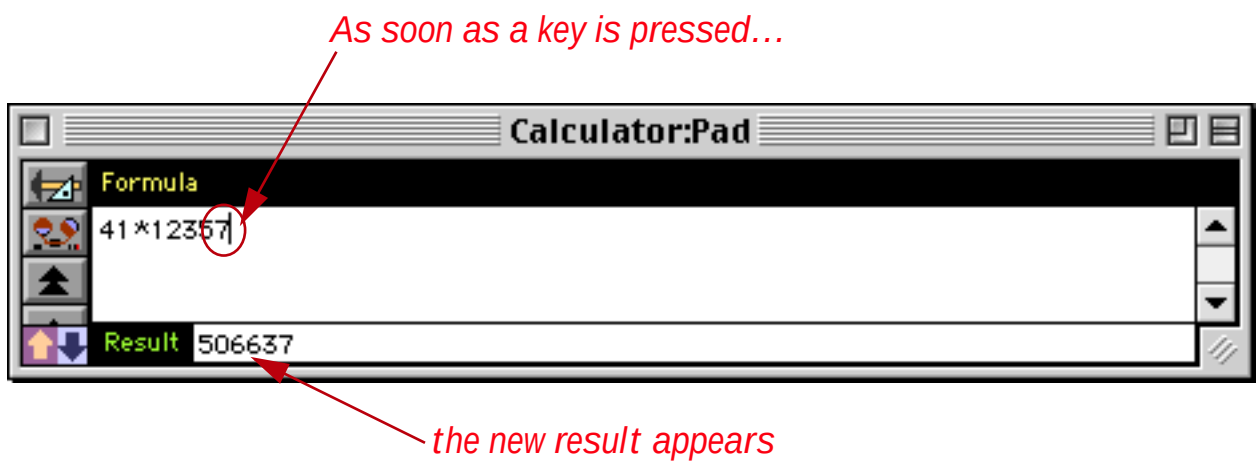
If the **Every Key** option is checked, the procedure will be triggered every time the user presses a key. For example, you might use this option if you wanted to count the user's key strokes.

If the **Most Keys** option is checked, the procedure will be triggered after every key when the user types slowly, but will not be triggered for each key when the user types several characters quickly in a row. The procedure will not be triggered until the user pauses in his or her typing. This option often works as well as the Every Key option but usually appears much smoother and faster to the user because the procedure is not being triggered as frequently while the user types. Possible applications for the Most Keys options include counting the characters or words being edited, performing a calculation (metric conversion, for example), checking spelling, etc.

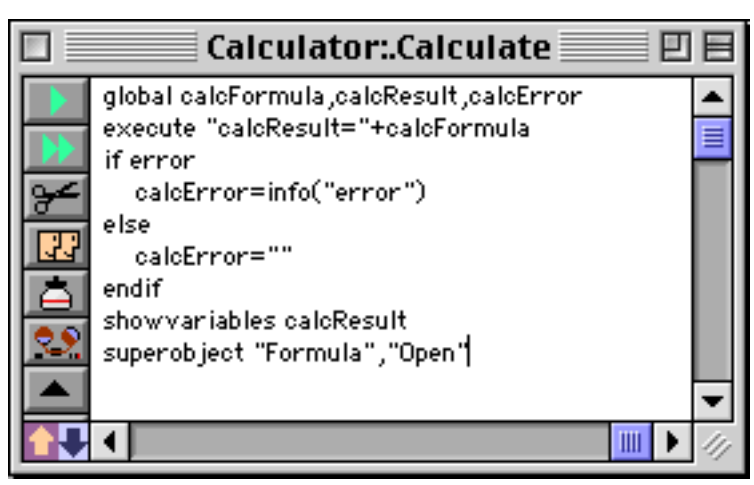
In this illustration the Text Editor SuperObject containing the formula has been set up with the Most Keys option.



When a key is pressed (most times) a procedure is triggered. This procedure calculates and displays the result of the formula. The new result appears immediately, as soon as the key is pressed. (If the Finished option was used instead of Most Keys, the result would not appear until you pressed the **Enter** key.)



If you want to duplicate this example yourself, here is the `.Calculate` procedure (see "Procedures" on page 1339). This procedure relies on the `execute` statement to perform the calculation. See "Building Subroutines On The Fly (The Execute Statement)" on page 1399 to learn about this statement.



The last line of this procedure refers to the Text Editor SuperObject by the name `Formula`.

```
superobject "Formula","Open"
```

For this to work the SuperObject must be given this name with the Object Name dialog. See "Object Type / Object Name" on page 533 to learn how to give an object a name.

If the Focus option is checked, the procedure will be triggered every time editing starts to happen in this object. In other words, when you click on the field or tab into the field, the procedure will be triggered. It's called Focus because this option is triggered whenever this object becomes the focus of attention.

One use for a focus procedure is to implement Undo for editing. Here is a procedure that saves the data in the field as the editing begins.

```
if info("trigger") beginswith "Focus."
   undoCell=«»   /* «» is the current field */
   undoField=info("fieldname")
endif
```

The Undo procedure would look like this.

```
if undoField<>""
   set undoField,undoCell
endif
```

For completeness you may wish to add the following line to your **.CurrentRecord** procedure (see ".CurrentRecord" on page 1510). This line ensures that you cannot undo after moving to a different record.

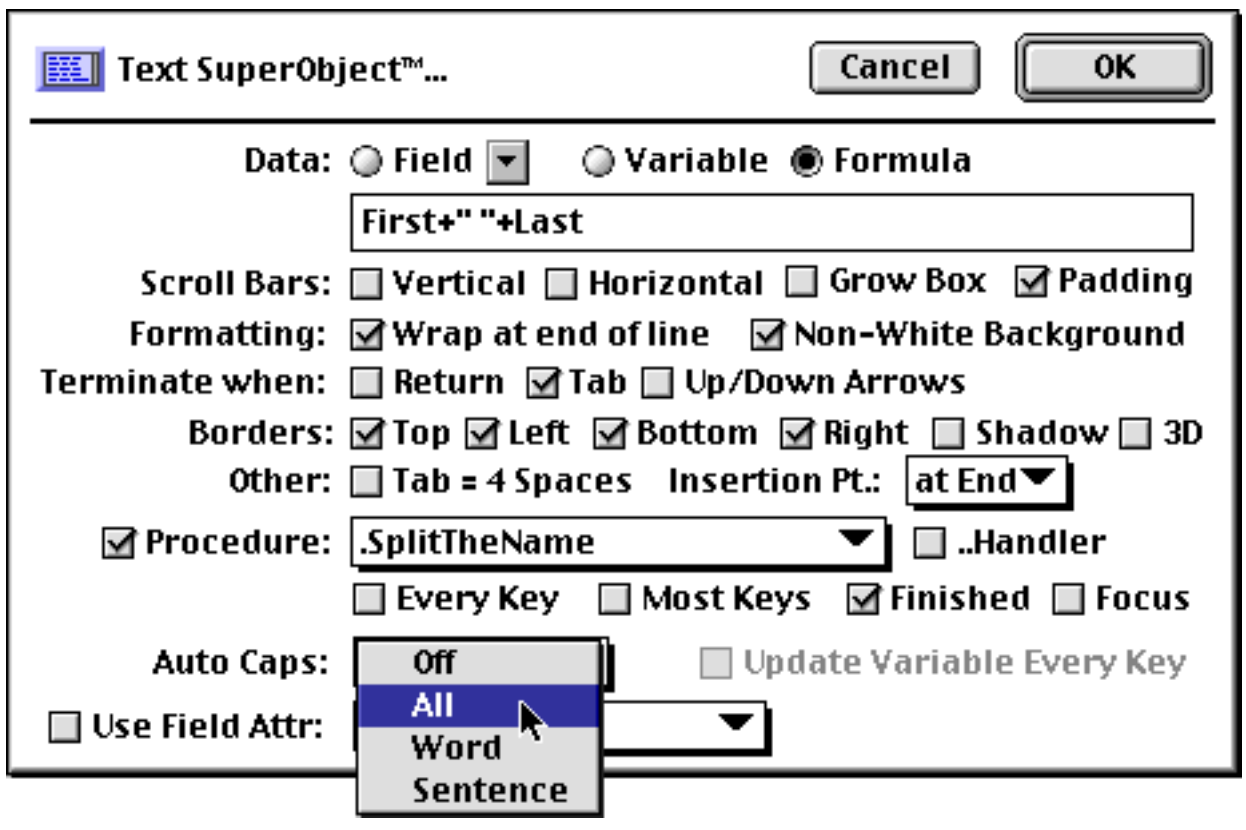
```
undoField=""
```

Another use for the Focus procedure is to memorize the selection point when editing was terminated and reset the selection when editing resumes again. This example assumes that the database has two numeric fields named textStart and textEnd.

```
if info("trigger") contains "focus"
   activesuperobject "setselection",textStart,textEnd
else
   activesuperobject "getselection",textStart,textEnd
endif
```

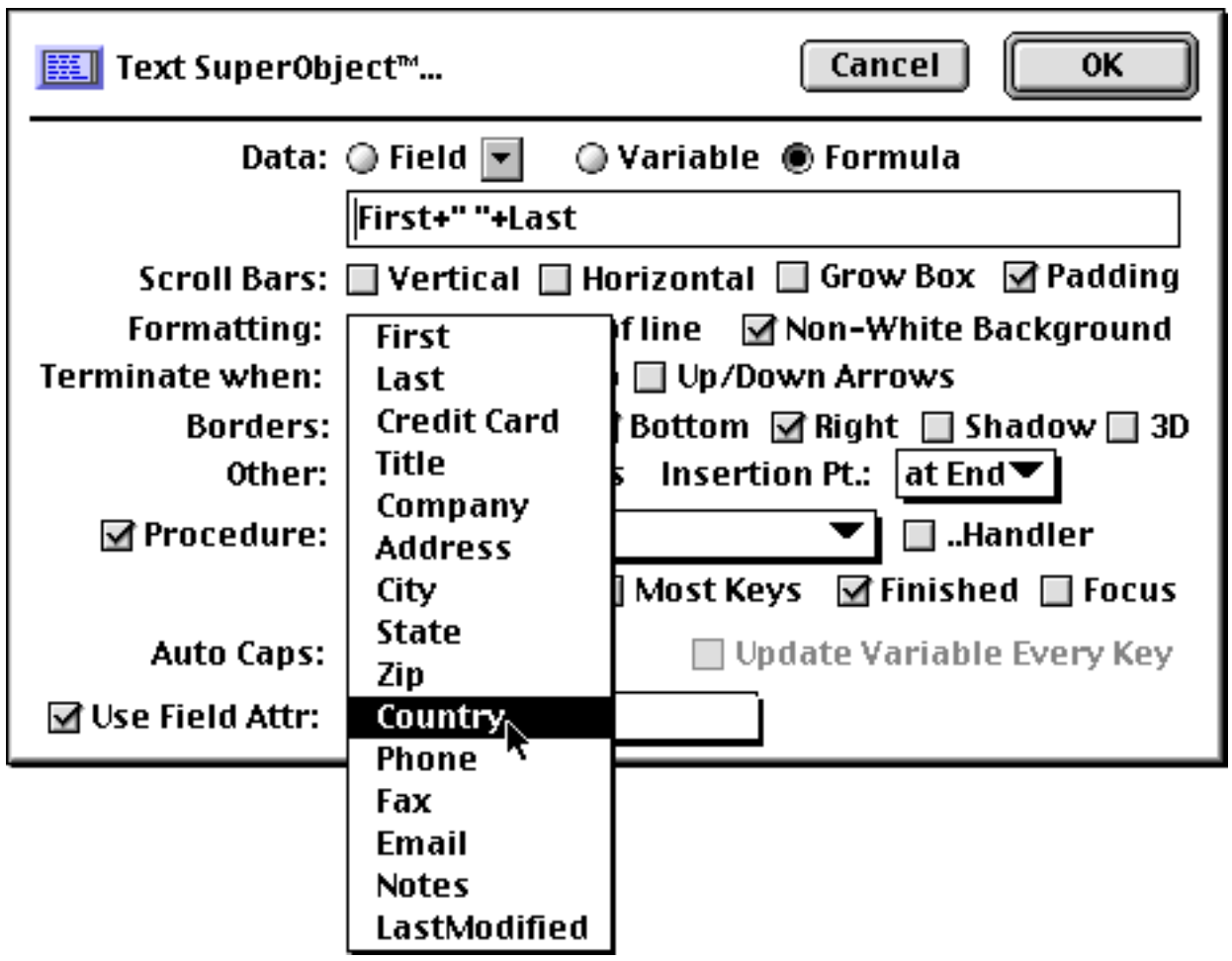
The Focus option must be used with the ..Handler option turned on. When the ..Handler option is turned on, all procedures triggered by the Text Editor SuperObject are treated as event handler procedures (see "Event Handler Procedures" on page 1522). The benefit of using event handler procedures is that these procedures are guaranteed to trigger and work properly under all conditions, no matter how the user started or stopped editing and whether or not another procedure is currently running. The only downside is that event handler procedures cannot open or close windows. To retain compatibility with databases created with earlier versions of Panorama you are allowed to turn the ..Handler option off. To learn more about this type of procedure see "Event Handler Procedures" on page 1522.

AutoCaps: This option only appears if the Text Editor is associated with a variable or formula.



Use the pop-up menu to control automatic capitalization of the text as it is entered. You can force the text to all upper case (ABC), word caps (Abc), or capitalization of the first letter of each sentence. If the Text Editor SuperObject is associated with a field, the Text Editor SuperObject will automatically use the Auto Caps setting for that field (set in the design sheet or the Field Preferences dialog, see "Field Properties" on page 250.)

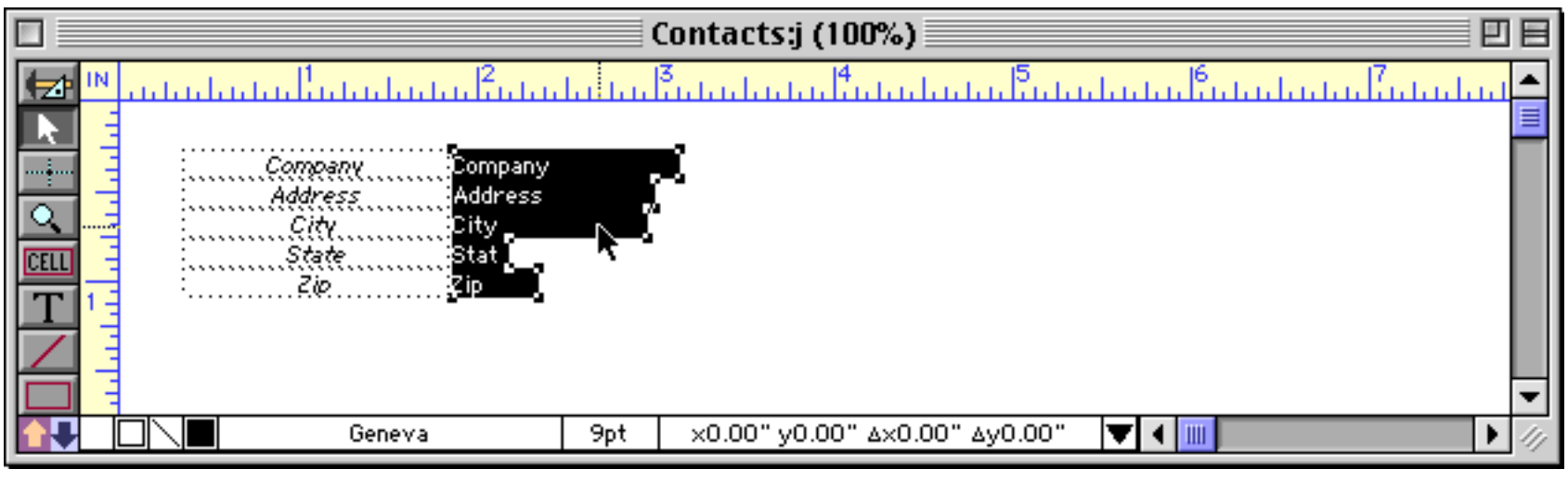
Use Field Attributes: This section only appears if the Text Editor SuperObject is associated with a variable or formula. You can use this if you would like the Text Editor SuperObject to use the attributes assigned to one of the fields in your database.



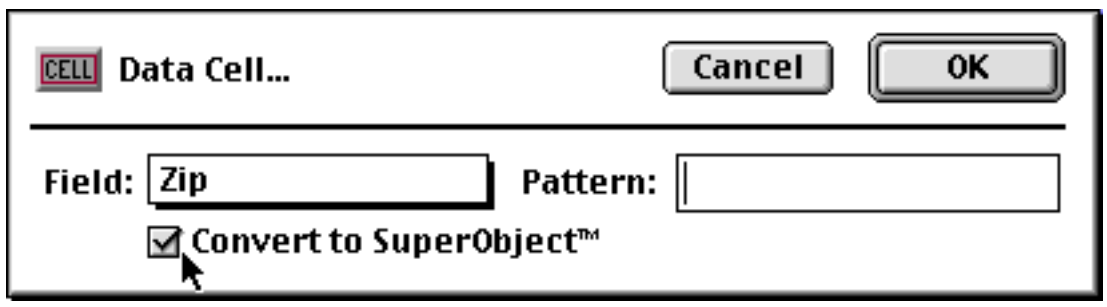
When you select a field with the pop-up menu, the Text Editor SuperObject will use that field's settings for Input Pattern, Range, Clairvoyance, Space Bar Tab, and Duplicates. (Of course, if the Text Editor is associated with a field, the text editor will always use the attributes of that field (set in the design sheet or the Field Preferences dialog.)

Converting Data Cells into a Text Editor SuperObjects

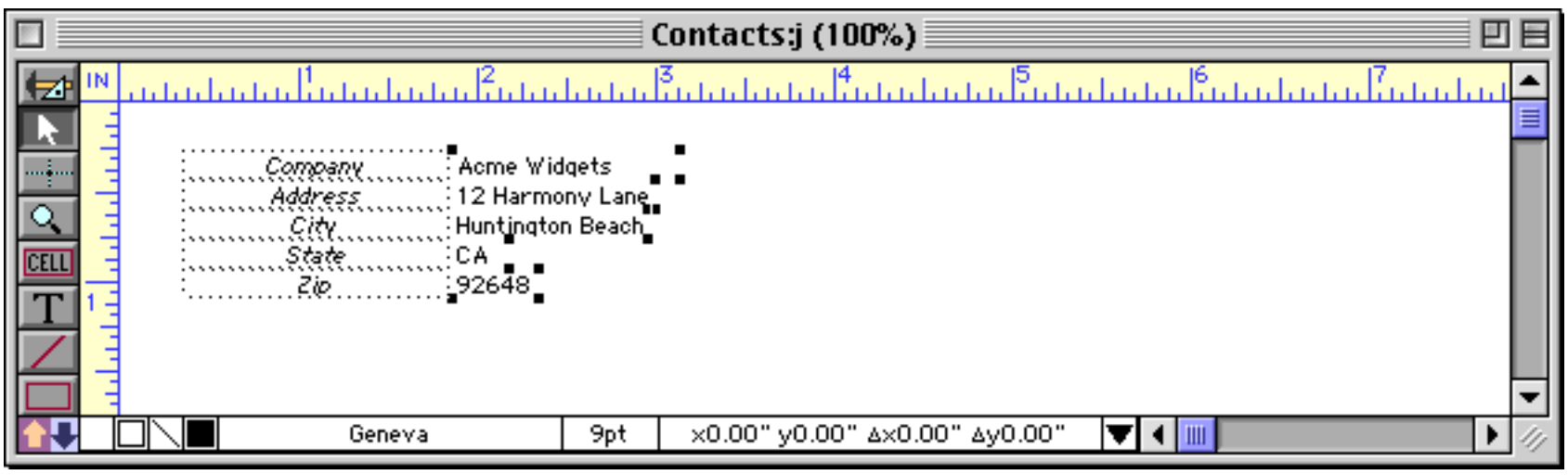
To convert one or more data cells into Text Editor SuperObjects, start by selecting the cells. Then double click on one of the selected cells.



When you double click the dialog shown below will appear.



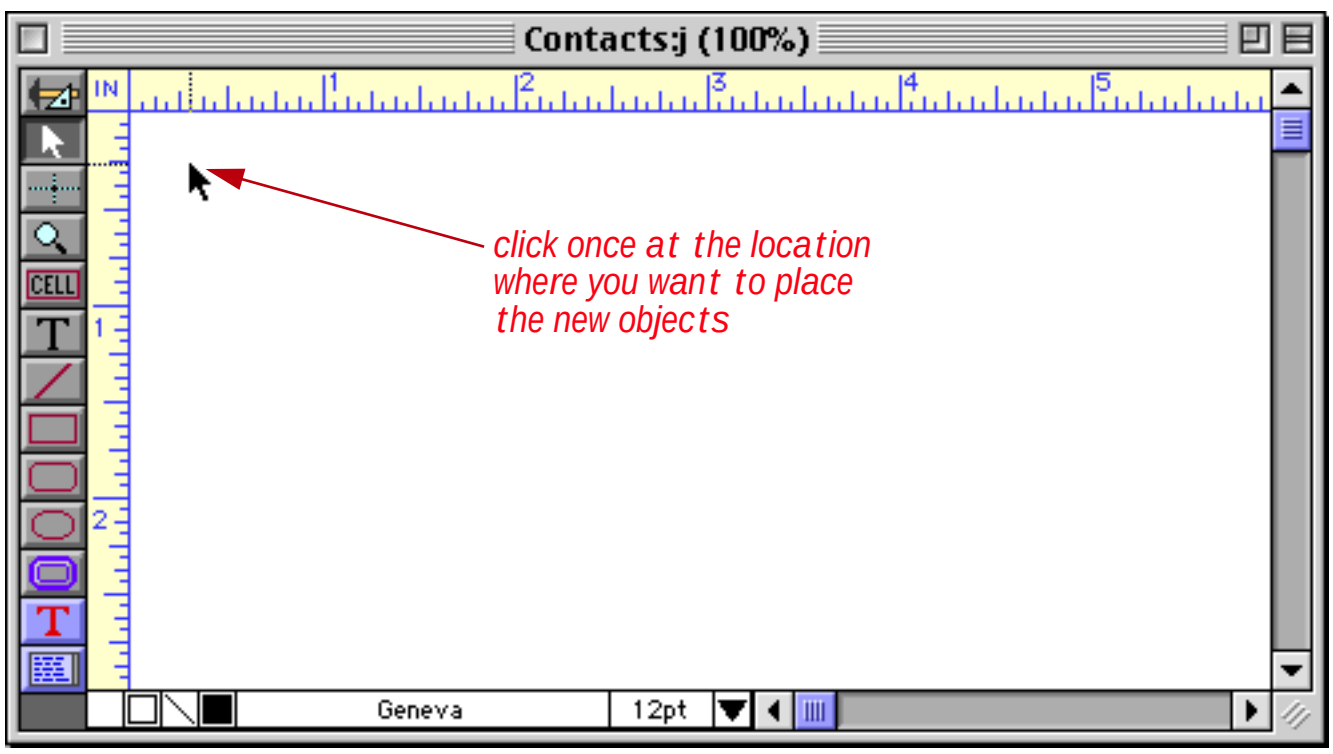
Check the Convert to SuperObject box and press **OK** to convert the selected data cells into Text Editor Super-Objects.



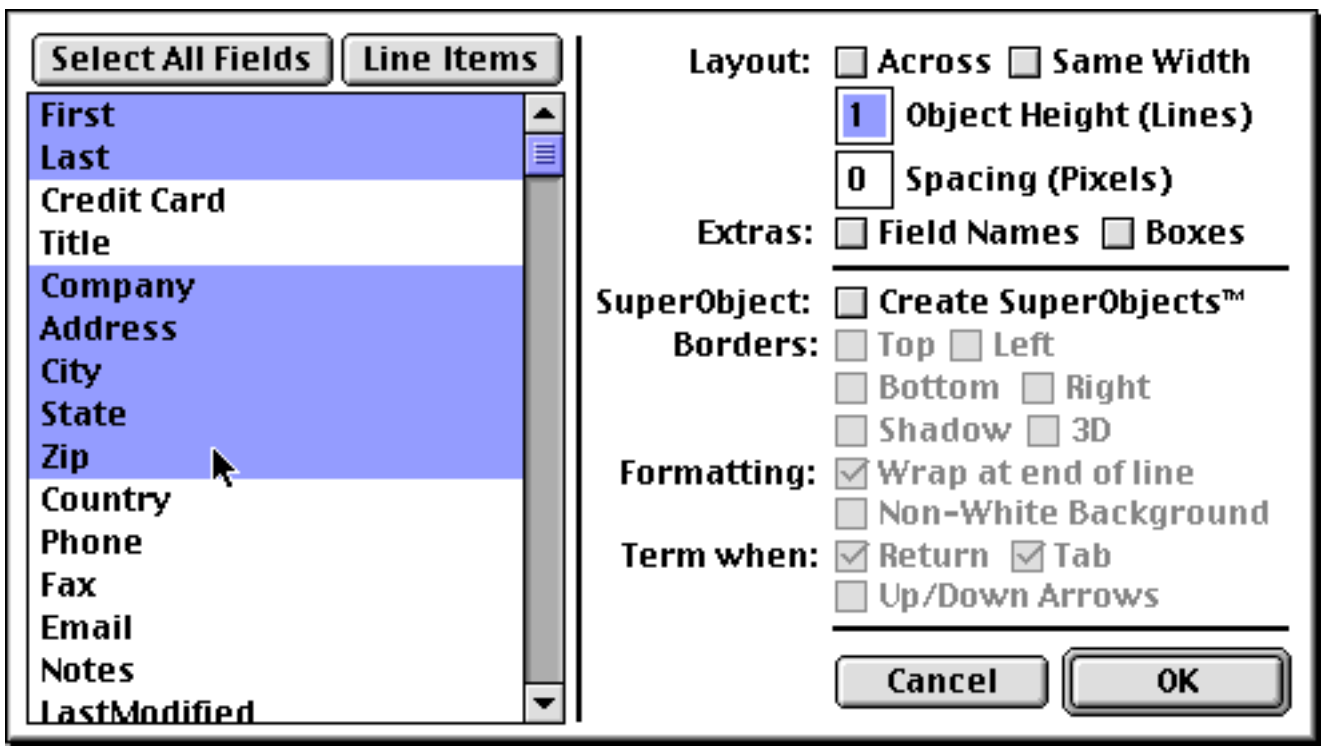
If you want to change the attributes of any of these new SuperObjects you must double click each object and set the attributes individually.

Automatically Creating Rows or Columns of Data Cells or Text Editor SuperObjects

Earlier in this chapter you learned how to create data cell objects and Text Editor SuperObjects one at a time. The **Auto Cell Layout** command (in the Arrange menu) can automatically generate an entire row or column of these objects. To use this command, first pick the font and size you want to use from the Graphic Control Strip or the Text sub-menus (see “Font” on page 529 and “Text Size” on page 531). Once the font is set, make sure the **Pointer** tool is selected and click on the spot where you want to place the new objects.

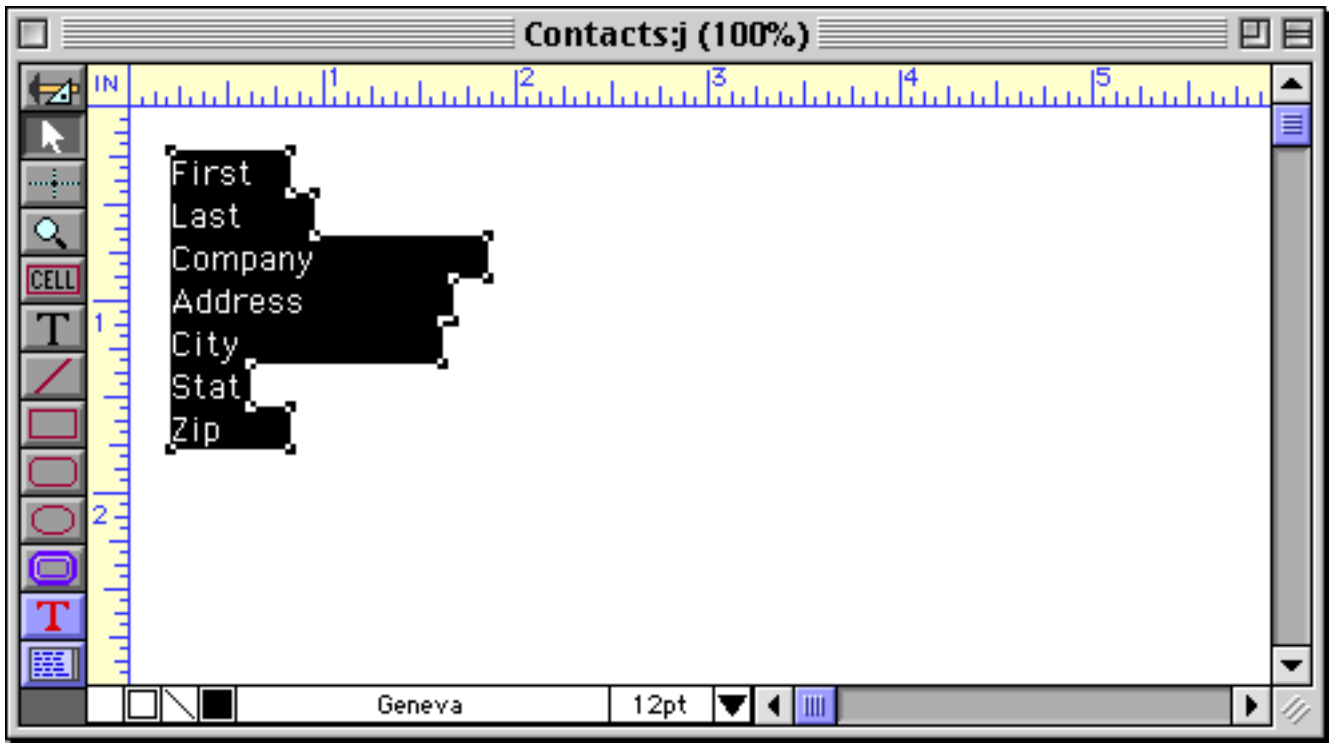


Then open the **Auto Cell Layout** dialog using the Arrange menu. This dialog allows you to choose the fields and arrangement you want.



The box on the left of the dialog lists all the fields in the database. Select each field you want to place in the form. There are several ways to select fields. You can select individual fields by clicking on them. You can select several fields at once by dragging the mouse across them. You can select all the fields by pressing the **Select All Fields** button. If you change your mind, you can de-select a field by clicking on it again.

Once you have selected the fields you want to create, press the **OK** button to place the fields into the form.

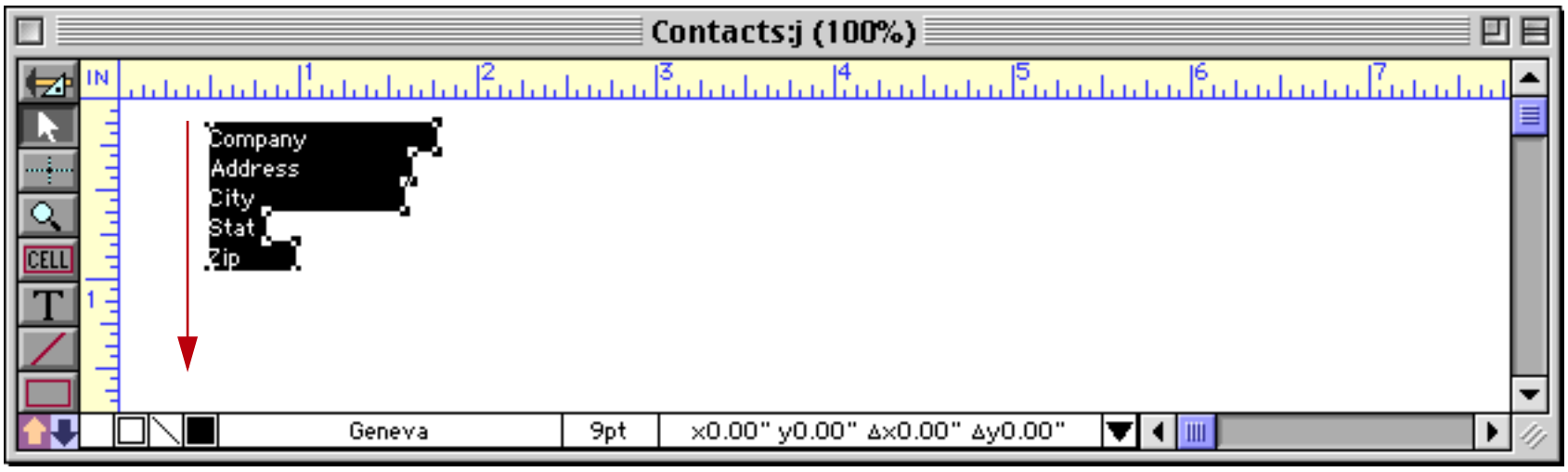


The new objects appear just below and to the right of the spot you originally clicked on.

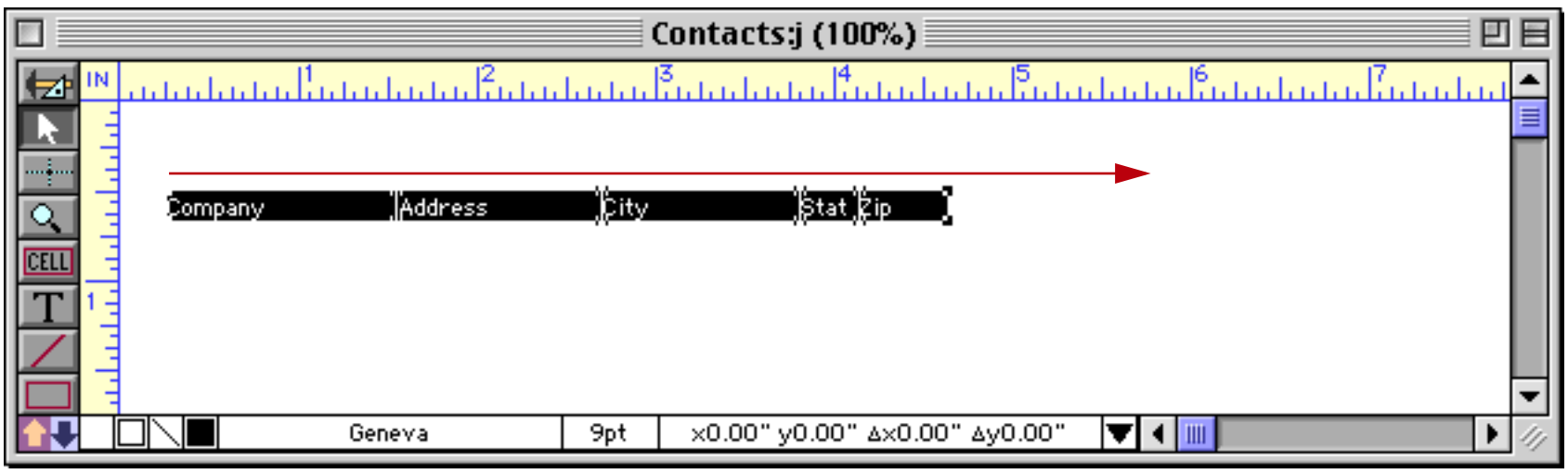
Automatic Layout Options

The right hand side of the Auto Cell Layout dialog contains options for varying the arrangement of objects that are created.

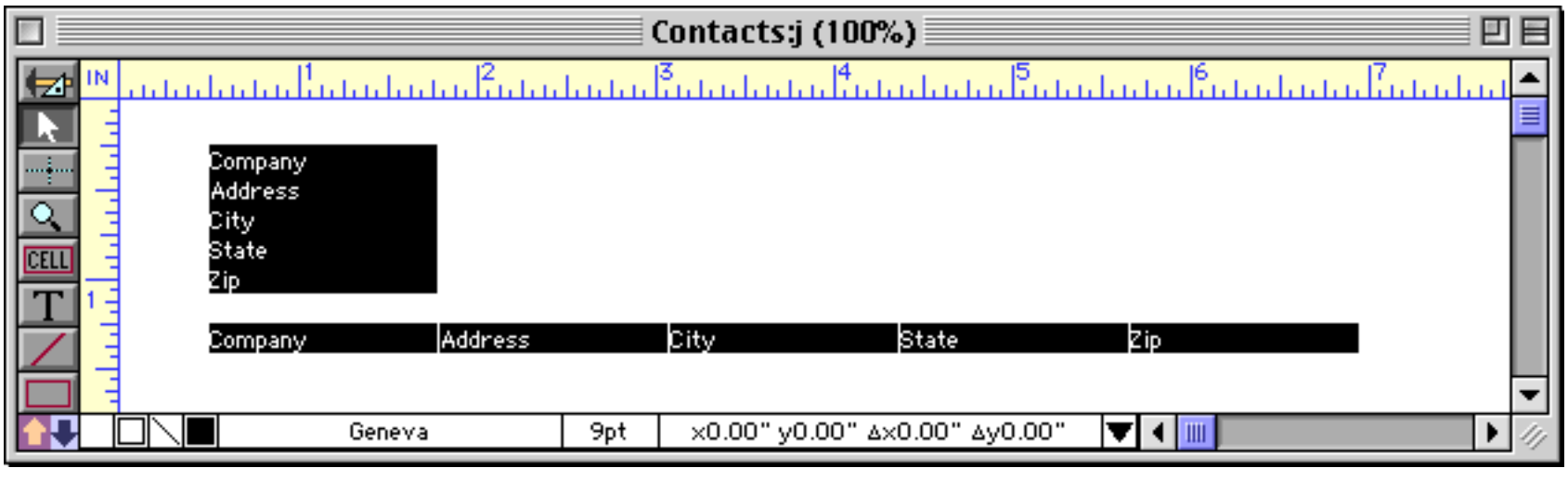
The Across option controls the direction of the generated objects. The normal direction is down.



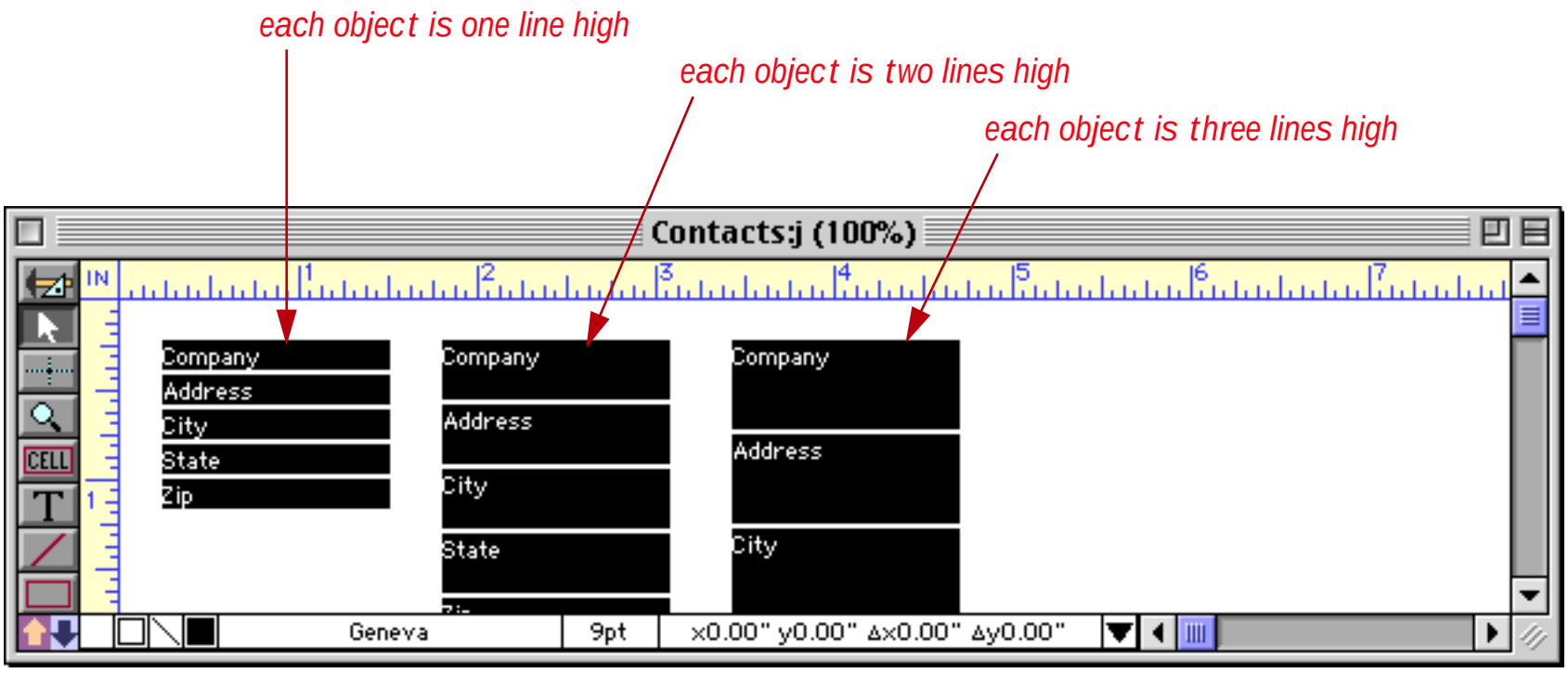
If the Across option is enabled the objects are generated horizontally, in a row.



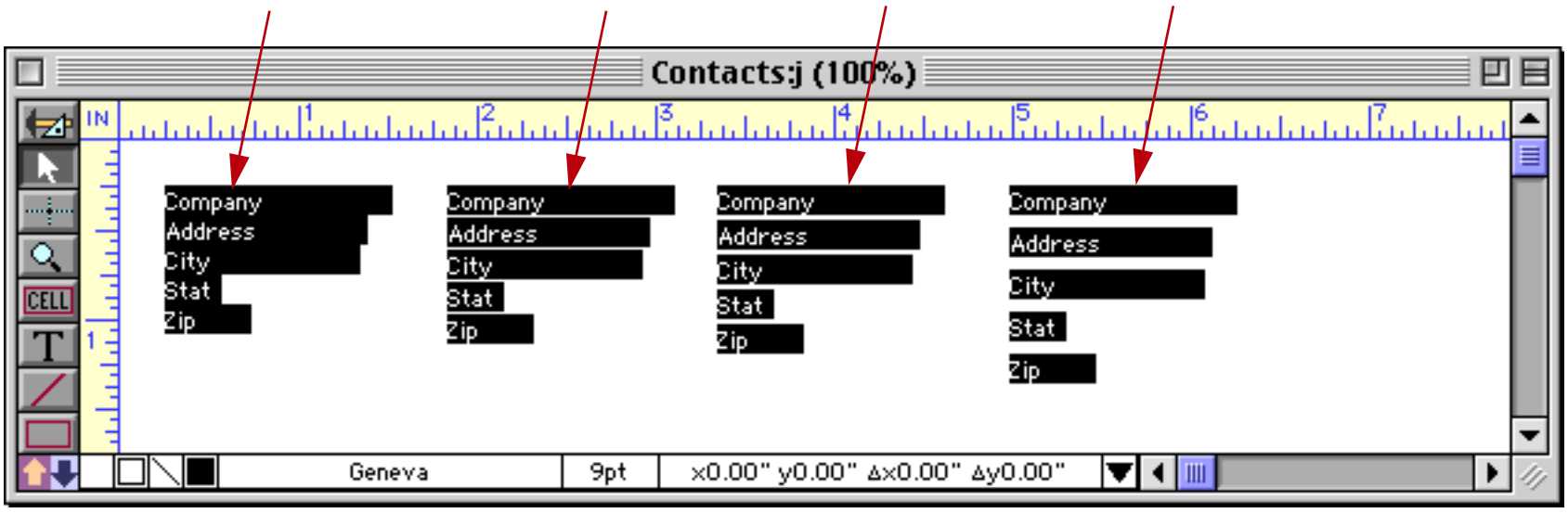
The Same Width option controls the width of the generated objects. If this option is off, the width of each object will be the same as the width of the corresponding column in the data sheet (see “Changing the Width of a Field” on page 251). If the Same Width option is enabled, all of the objects will have the same width. (Of course you can always change the width of any object after it has been generated — see “Changing the Size of a Single Object” on page 513.) This illustration shows what the end result of this option looks like both in normal mode and with the Across option enabled.



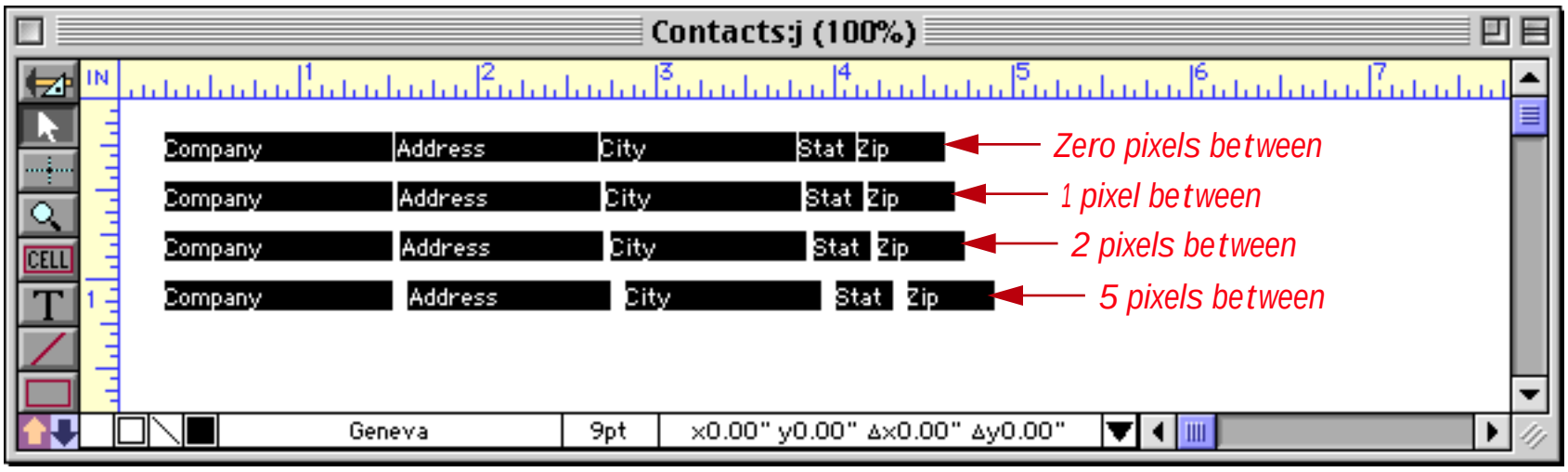
The Lines High option specifies the height of each object. Normally each object is one line high in the current font, but you can generate objects that are two lines high, three lines high, or more. Of course you can also change the height of any object after it has been generated (see "Changing the Size of a Single Object" on page 513).



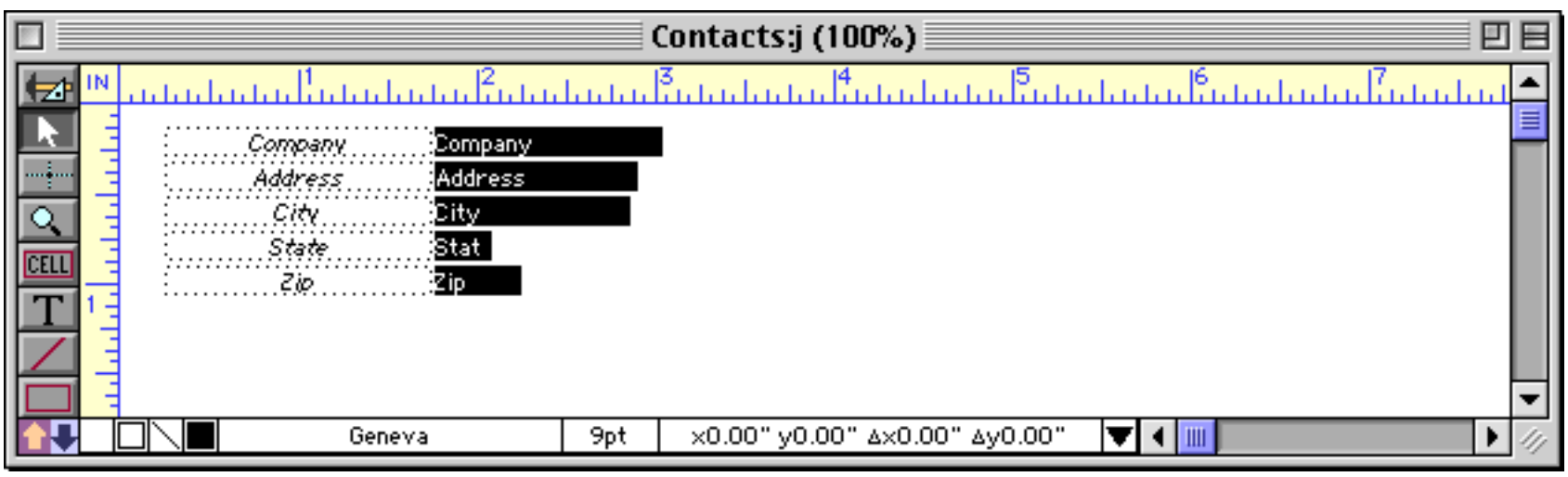
The Pixels Between option specifies the spacing between adjacent data cells. (A pixel is one dot on the screen at 100% magnification, or 1⁄72 inch.) When you create a column of data cells, this option specifies the vertical spacing between the cells.



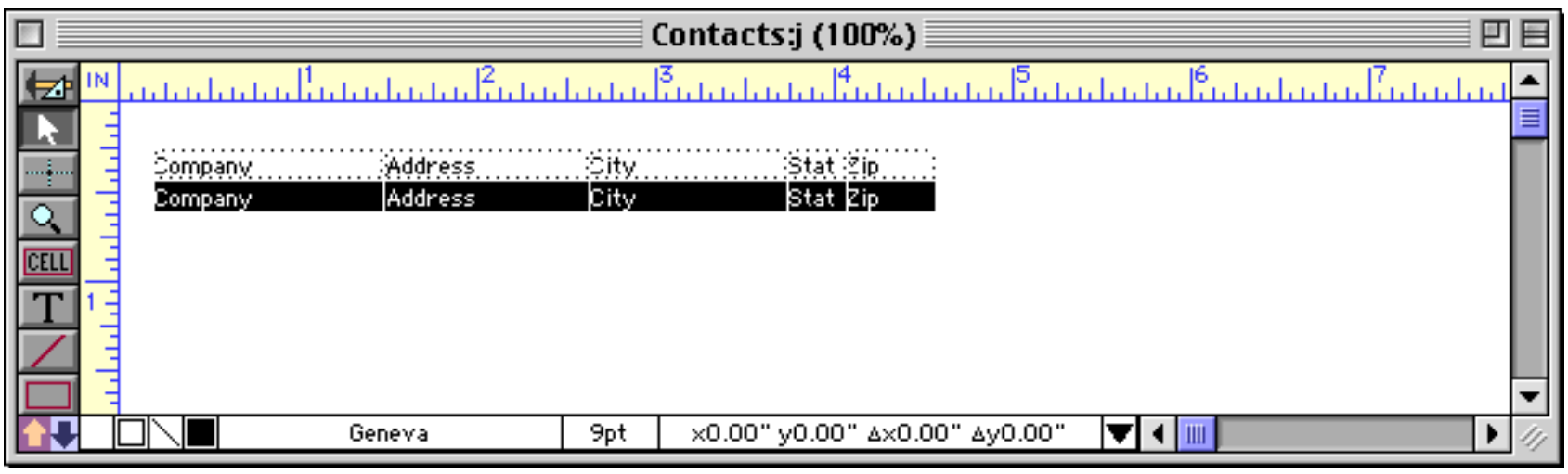
When you create a row of data cells (Across), this option specifies the horizontal spacing between the cells.



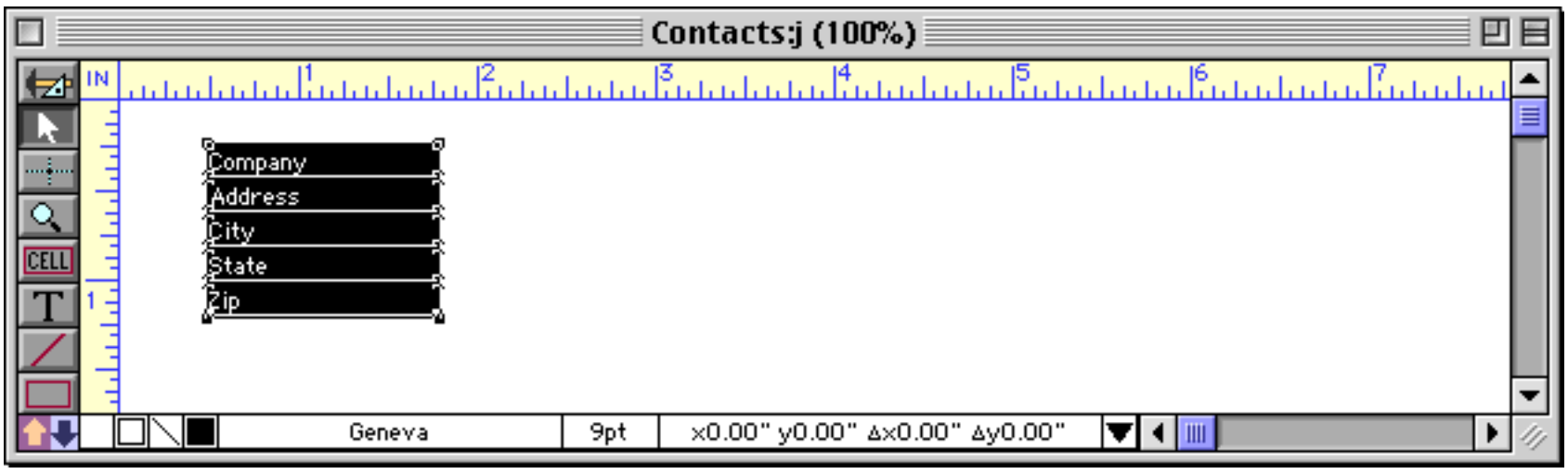
The Field Names option tells Panorama to create a field name next to each generated object. If you create a column of objects, the field names will be placed to the left.



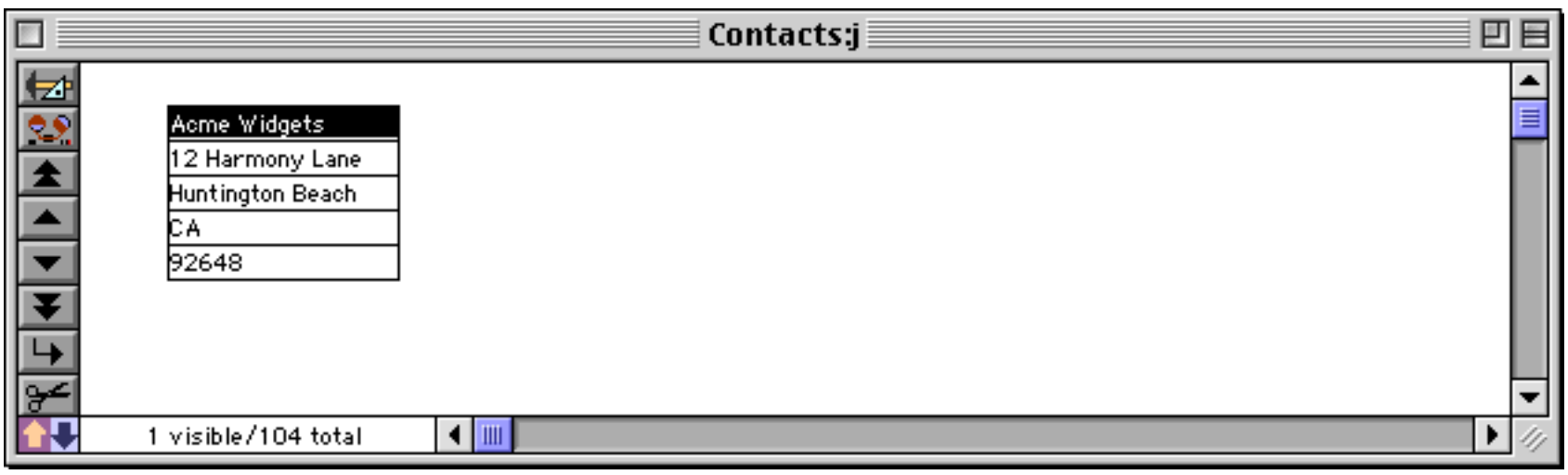
If you create a row of data cells (Across), the field names will be placed above the objects, like this.



Use the Boxes option if you want Panorama to draw a box around each generated object. This usually makes sense only for data cells, since for Text Editor SuperObjects you can generate borders as part of the object itself. This illustration shows what the boxes look like when used with data cells.



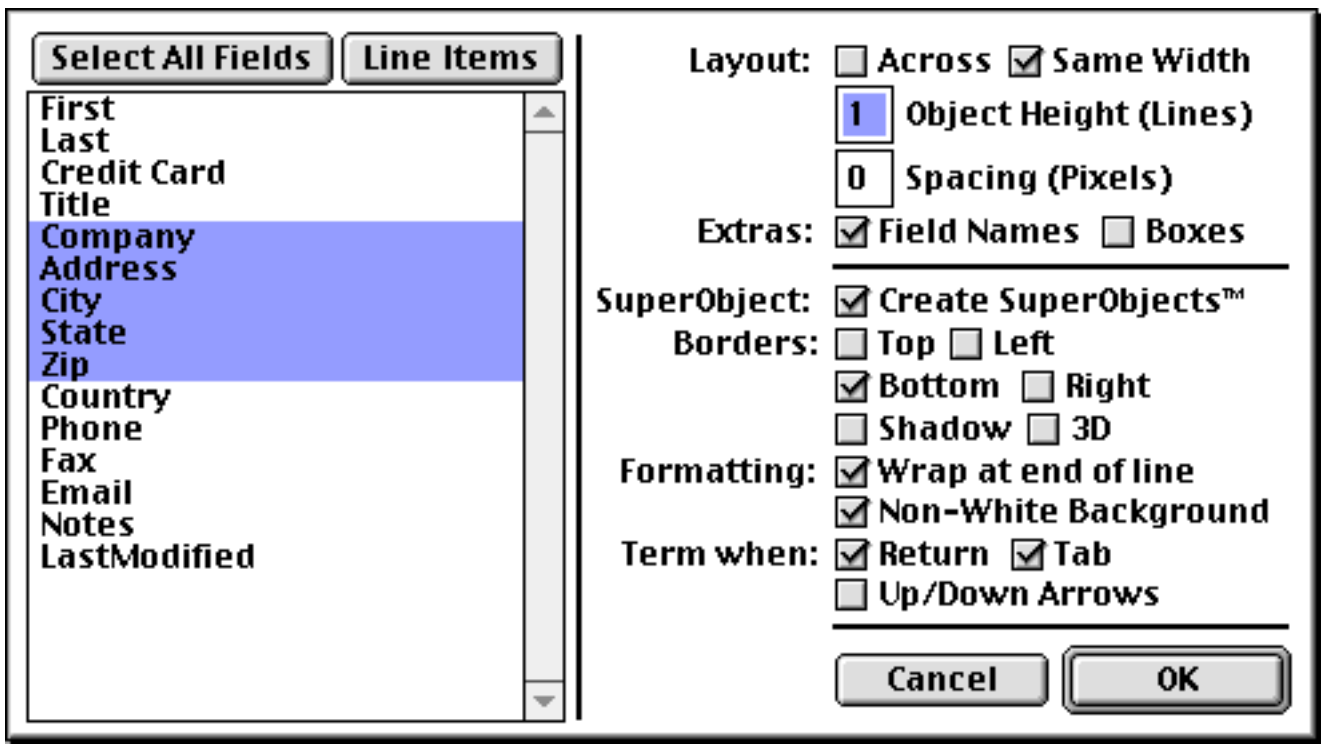
It's a bit easier to see what the boxes look like in Data Access Mode (see "Form Modes: Data Access vs. Graphic Design" on page 485).



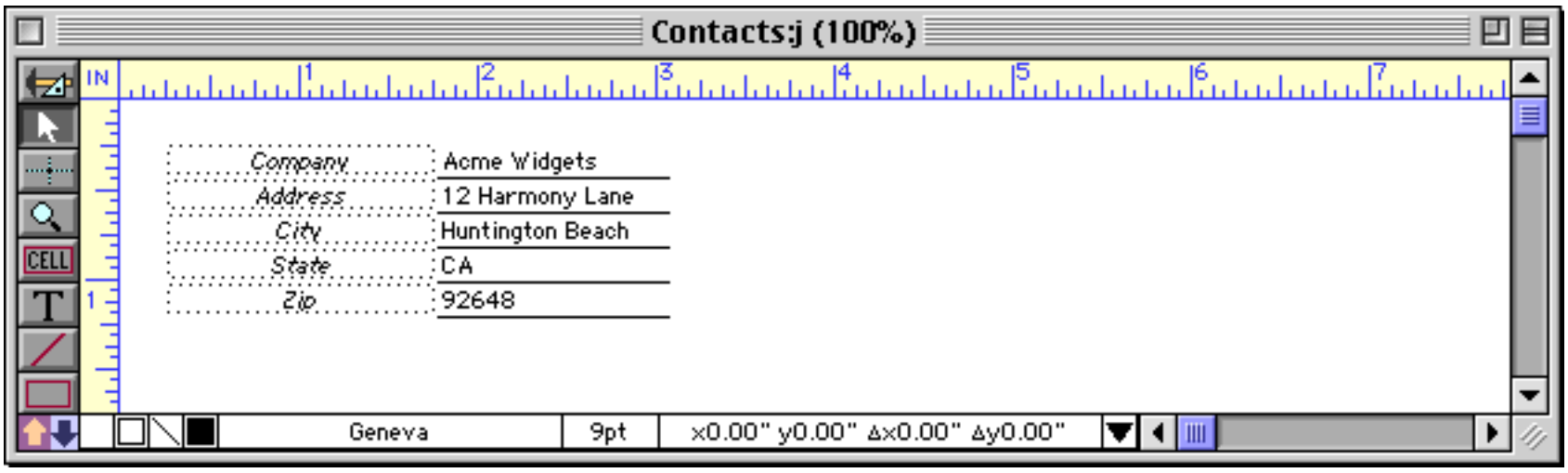
If Field Names option is checked Panorama will draw a box around the field names as well.



To create Text Editor SuperObjects instead of data cell, check the Create SuperObjects option. Once this option is turned on, you can select the options for the Text Editor SuperObjects you want to create — borders, formatting, etc. These options are the same as the options in the Text Editor SuperObject configuration dialog (see "Text Editor Options" on page 643).



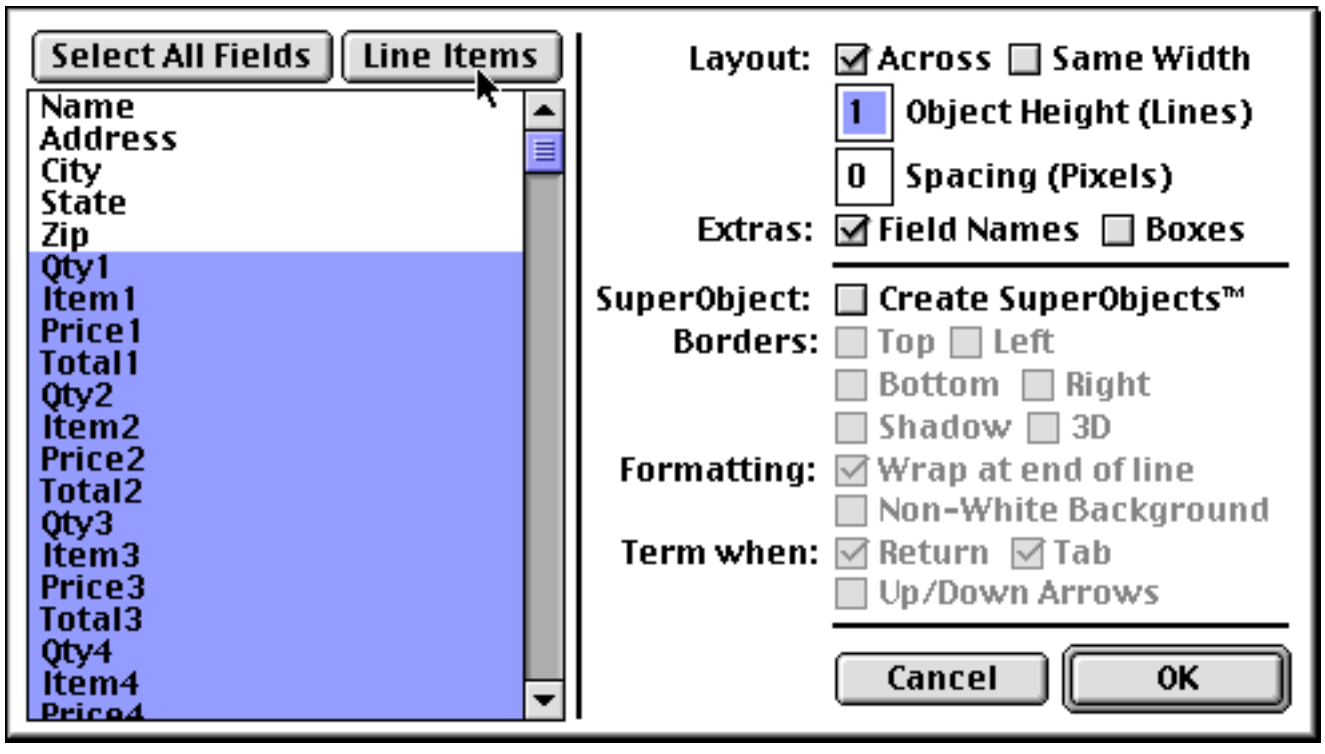
Pressing **OK** generates a column of Text Editor SuperObjects (instead of data cells).



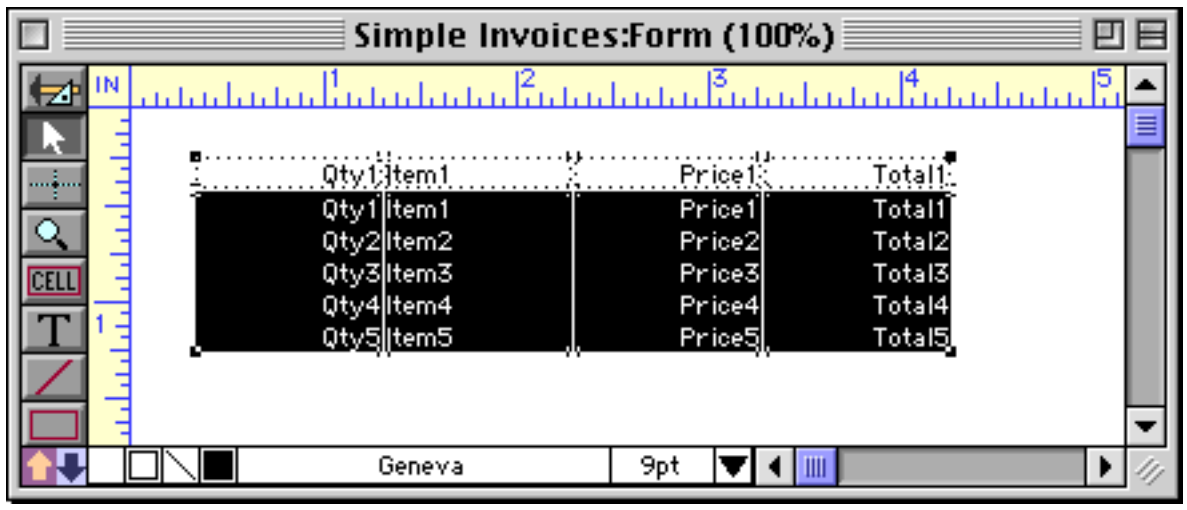
Clicking on the word **Borders** will turn all four borders on or off. If you use the Text Editor SuperObject borders your probably will not want to turn on the Boxes option, which adds an additional box around each object (see above).

Line Items in a Form

Line items are used for repeating items within a record (see "Repeating Fields (Line Items)" on page 262). The **Auto Cell Layout** command makes it easy to create a table of line items within a form. With the **Pointer** tool selected, click on the upper left hand corner of the spot where you want the line items to appear. Next choose the font and style you want to use. Now open the **Auto Cell Layout** dialog and press the **Line Items** button.



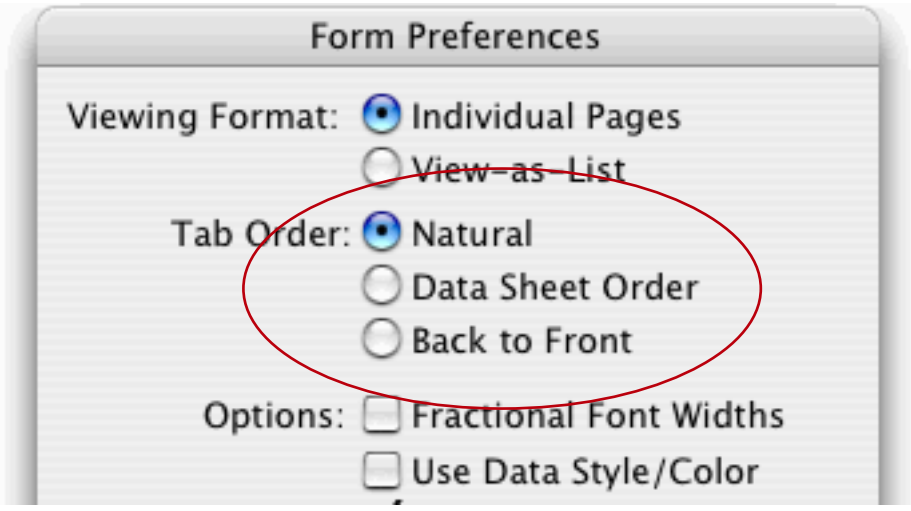
Pressing the **Line Items** button selects all the line item fields, and also checks the Across and Field Names options. You may also want to check the Boxes option and / or the Create Superobjects option (see "Automatic Layout Options" on page 663). When you press the **OK** button, Panorama will automatically create a table of line items formatted into rows and columns.



To learn how to adjust the width of an entire column in this table see "Cluster Resize" on page 541. To learn how to change the font size of this table (or the spacing) see "Adjusting Spacing Between Multiple Objects" on page 556.

Tab Order in Forms

Panorama has three tab order options for forms—**data sheet order**, **back to front order**, and **natural order**. Use the **Form Preferences** command (Setup menu) to specify the tab order option you want to use. (The form must be in graphic design mode.)

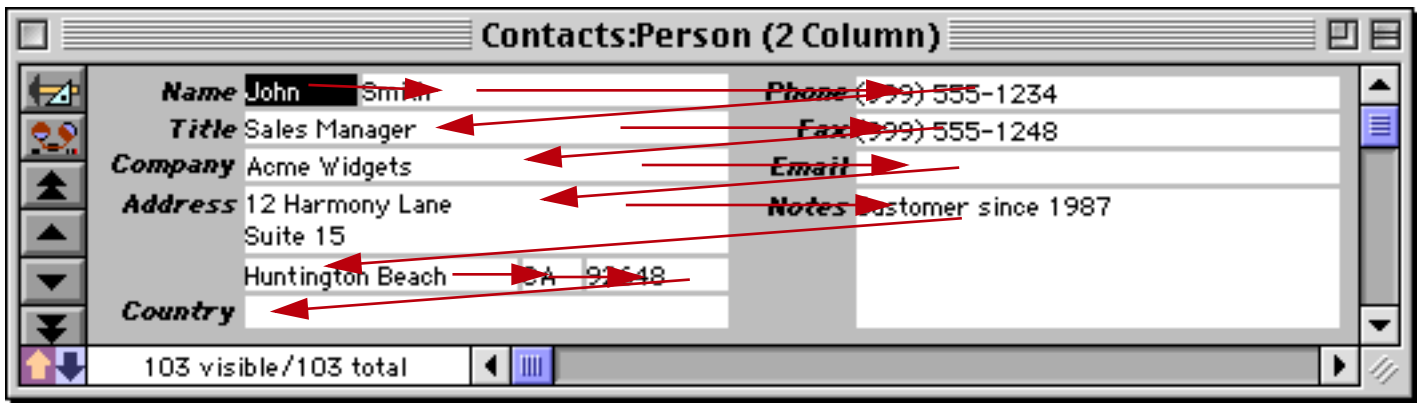


Data sheet order is exactly that—the **Tab** key moves from cell to cell in the same order as it would in the data sheet. However, data sheet order will not work if your form contains one or more variables in addition to fields to be edited (See "Text Editor Options" on page 643).

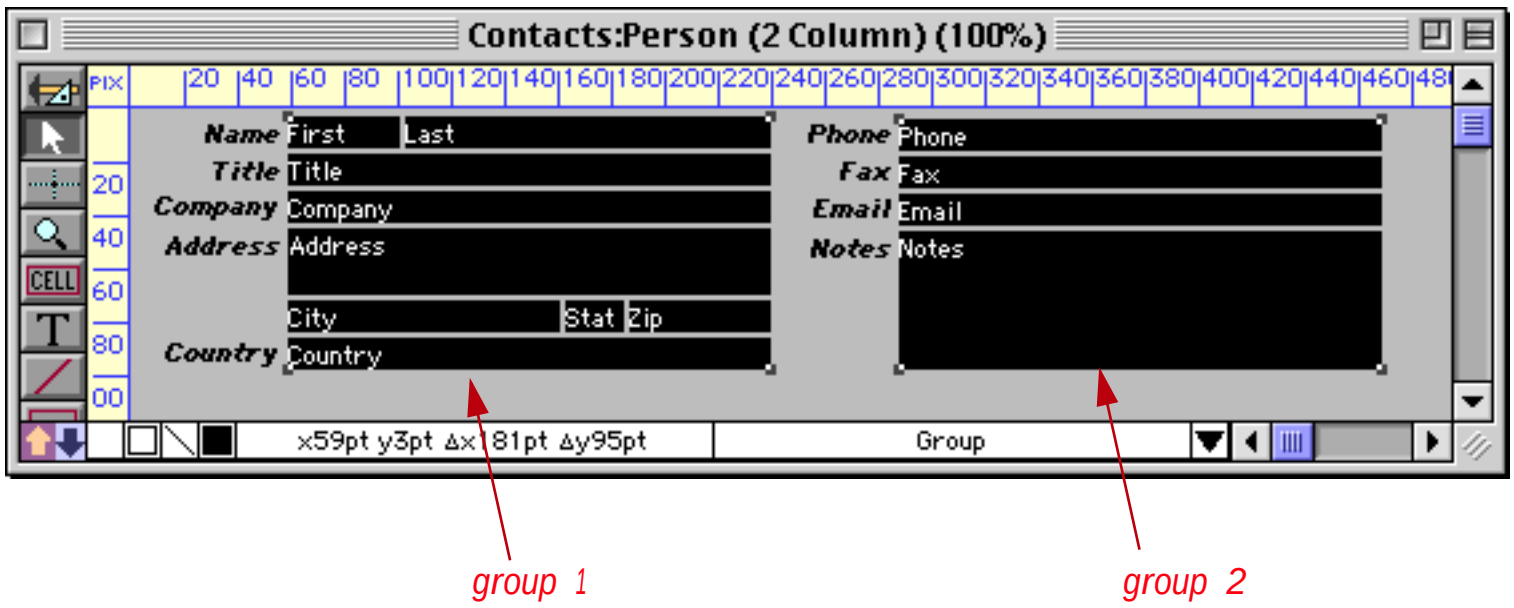
Natural order causes the **Tab** key to move from left to right, then from top to bottom.



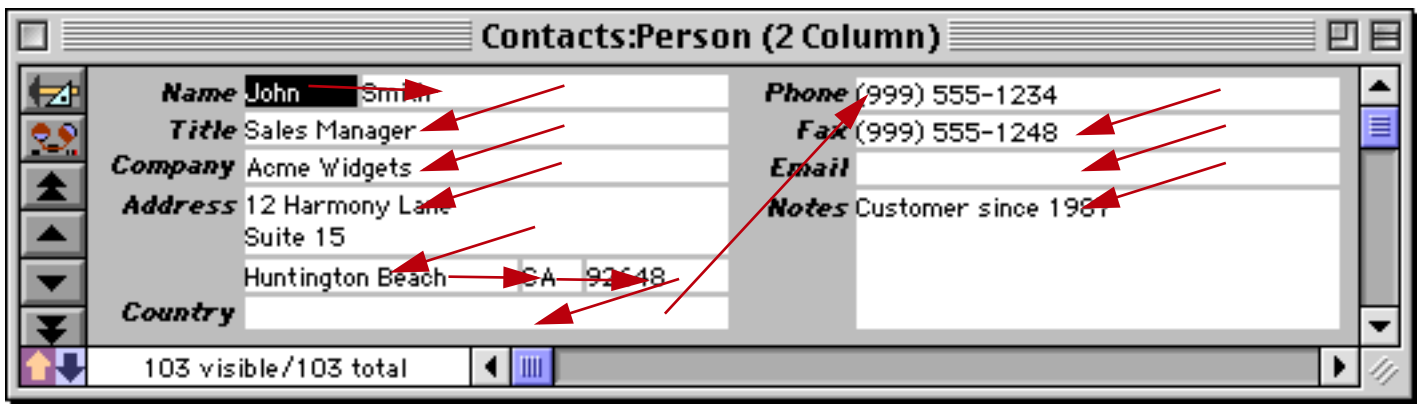
This usually works well (and is the default option), but in some cases isn't really what you want. This is especially true in forms with side by side columns of data.



To fix this you can alter the natural order by grouping data cells together–the **Tab** key will move through all the cells in the group of objects (in natural order) before it moves to the next cell. In this case the data cells need to be brought together into two groups using the **Group** command (see "Grouping Objects Together" on page 536.)



Now that the cells have been grouped together the tab order will tab through all of the cells in the left hand column before moving to the right hand column.



Back to Front order gives you the most control, but also takes the most work to set up. When this option is enabled the tab order depends on the back to front layering of the data cell objects in the graphic design mode. Use **Send to Back** to bring a data cell to the start of the tab order, and **Bring to Front** to send it to the end of the tab order. See "Changing the Stacking Order" on page 569 for more information on these commands.

For example, suppose your form contained three fields A, B, and C and you wanted to tab from field to field in the order B › A › C. To set up this order click on field B and use **Bring to Front** (the form must be in graphic design mode). Then click on field A and use Bring **to Front**. Finally click on field C and use **Bring to Front**.

Tab Order for Variables

The **Form Preferences** dialog (Setup menu) allows you to choose from three options — Natural, Data Sheet Order, and Back to Front (see previous section). All three options will work fine if you editing only fields. However, if a form allows the user to edit variables, the Data Sheet Order option will not work (because the data sheet does not contain variables, so they have no order!) Any form that contains Text Editor SuperObjects for variables should use Natural or Back to Front tab order.

Field Setup in Graphics Mode

As you are building a form, you may realize that the database needs another field for the form. You can use the Setup menu to add new fields to the database (or modify existing field properties) without having to leave graphic design mode.

To add a new field to the database from within graphic design mode, use the **Add Field** command in the Setup menu. Once the field is added to the database itself, you can create new data cell or Text Editor SuperObjects using the new field.

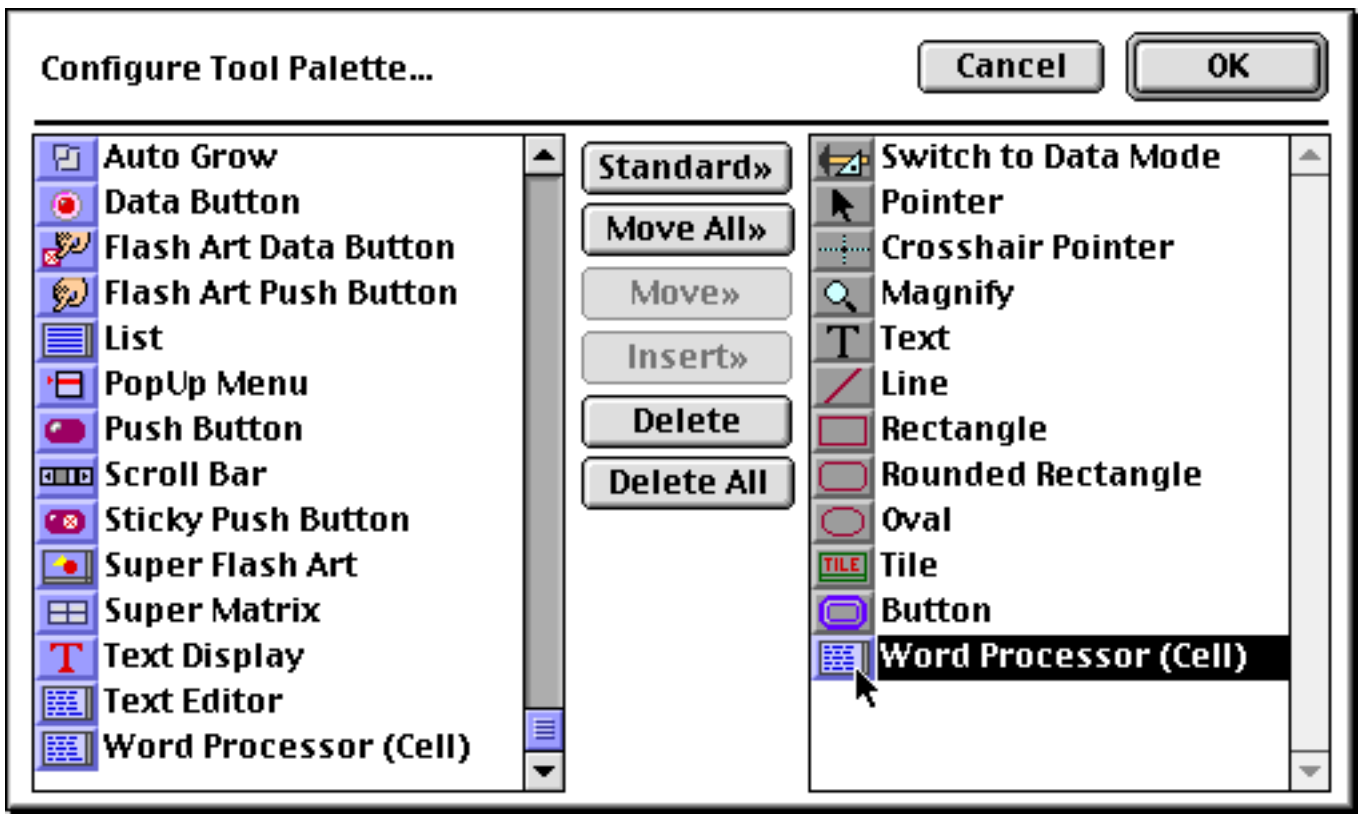
To modify an existing field, first use the **Pointer** tool to select a data cell assigned to the field you want to modify (only data cells work — you cannot click on a Text Editor SuperObject). Then use the **Field Properties** command (Setup menu) to modify the field. For more information on setting up fields see “Field Properties” on page 250.

Word Processor SuperObject

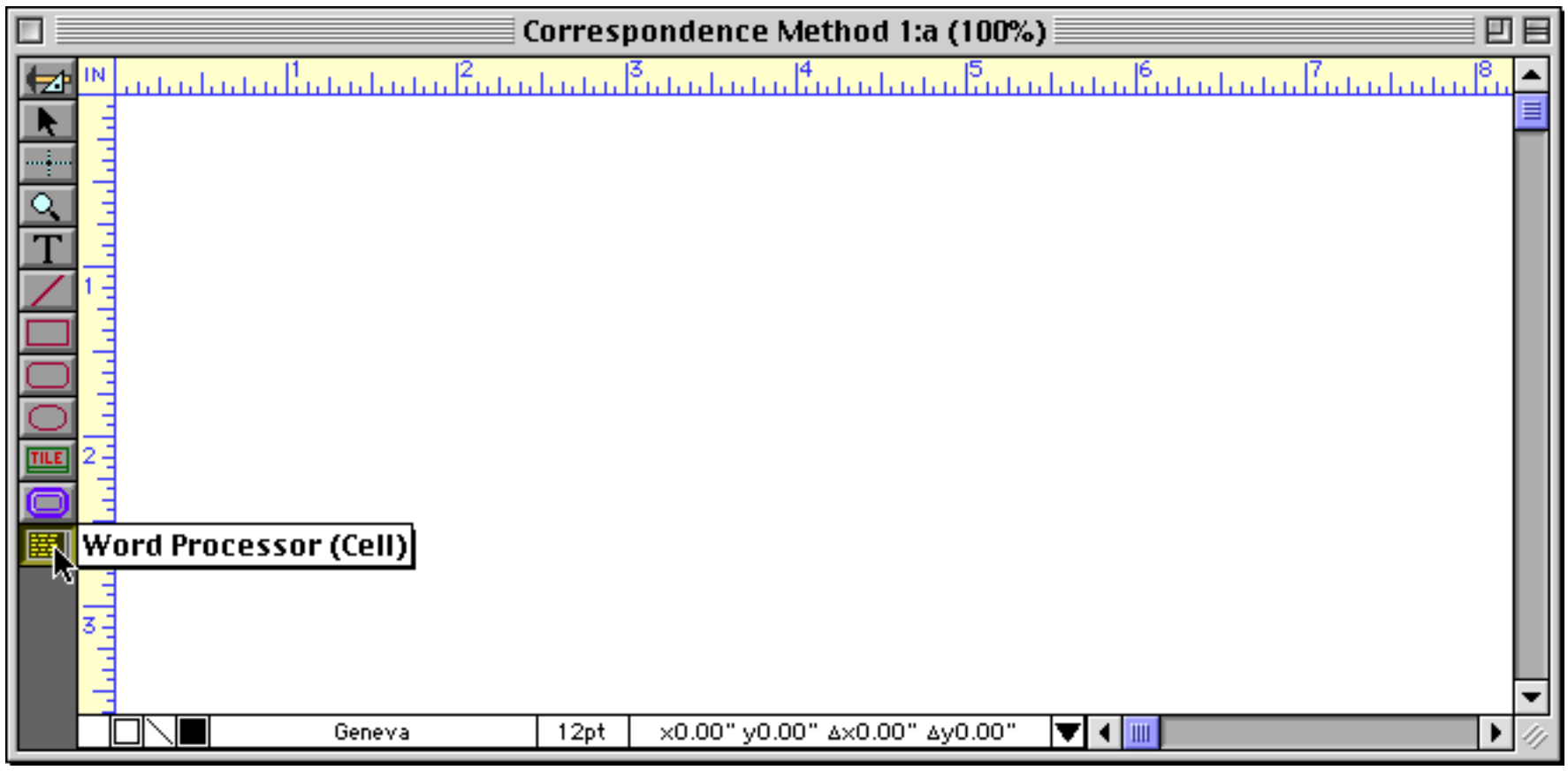
The Word Processing SuperObject™ allows you to include a complete word processor as part of a form. The word processor allows you to mix different fonts, sizes, styles (16 different styles) and colors in a single paragraph. Left, Right, and First Line margins may be set up separately for each paragraph, and you may set up left justified, right justified, center justified and decimal tabs with optional tab leaders. The merge option allows you to merge information from the database (or complete formulas) into the word processing text. The word processor SuperObject also includes most of the options available with the Text Editor SuperObject, including borders and a vertical scroll bar.

Creating and Working With Word Processor SuperObjects

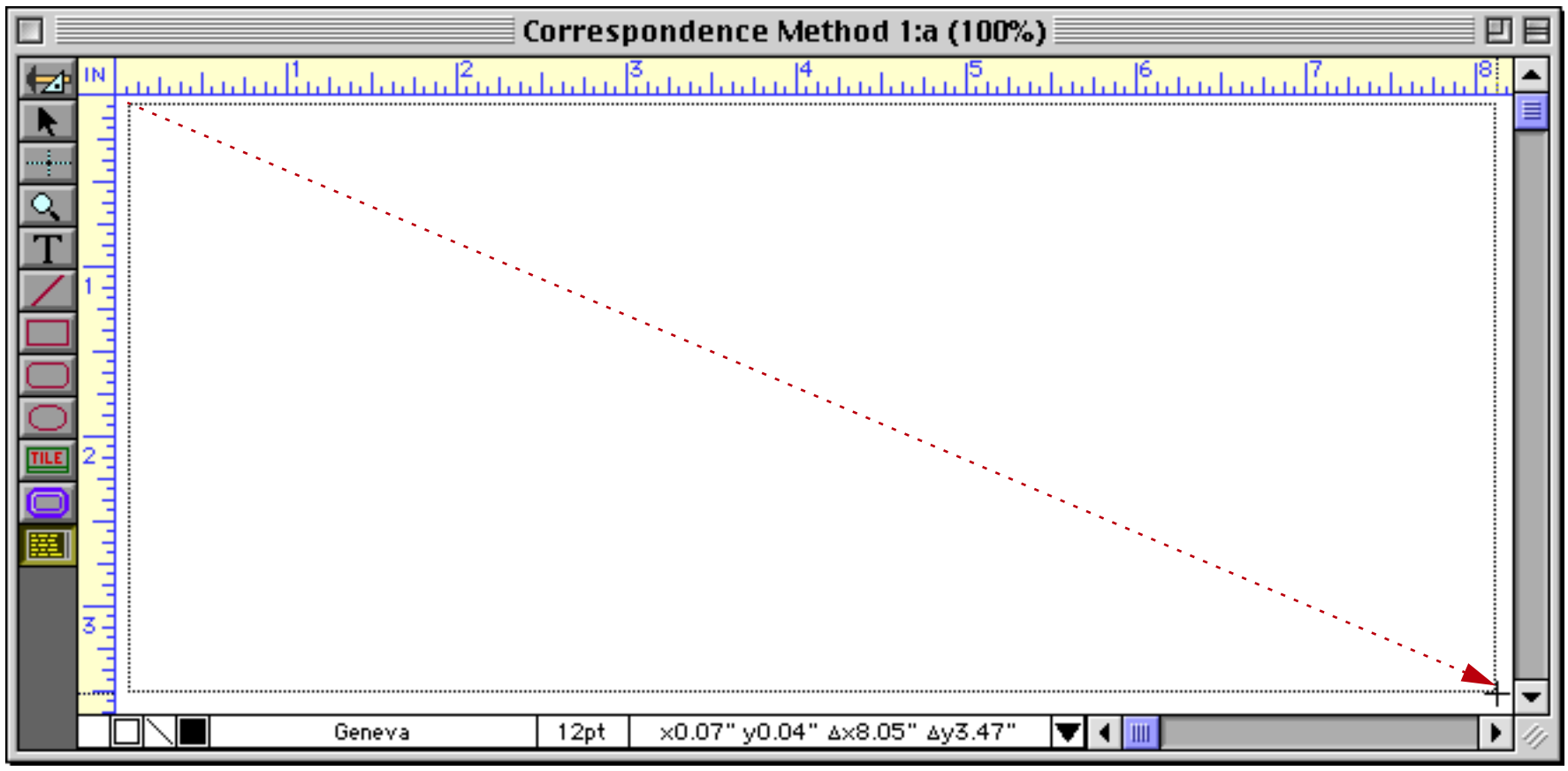
The Word Processor SuperObject tool is not in the default tool palette, so you'll need to use the Tool Palette dialog to add this tool to the palette if it is not already there (see "Customizing the Tool Palette" on page 497).



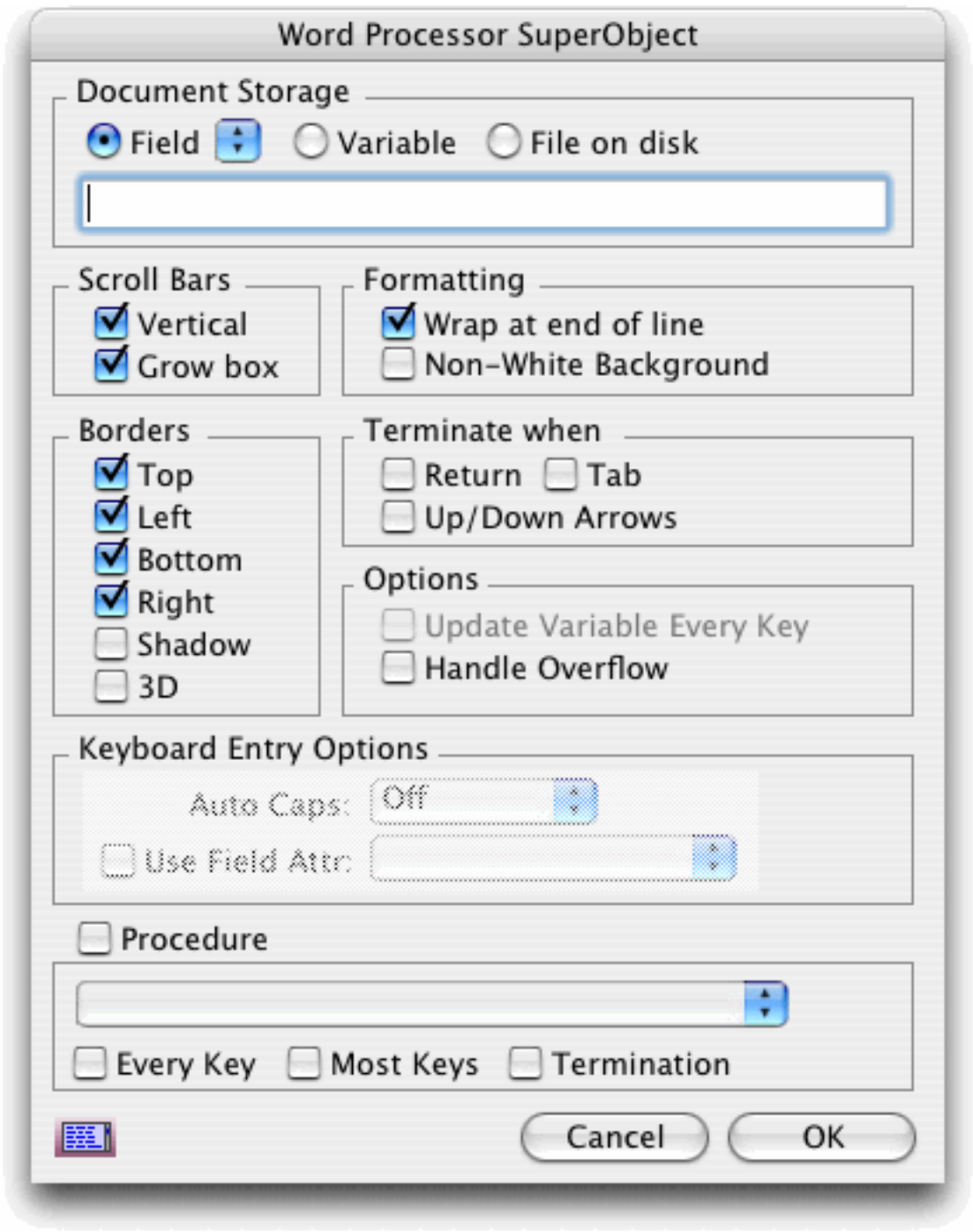
Now that the tool is added to the palette you can select it.



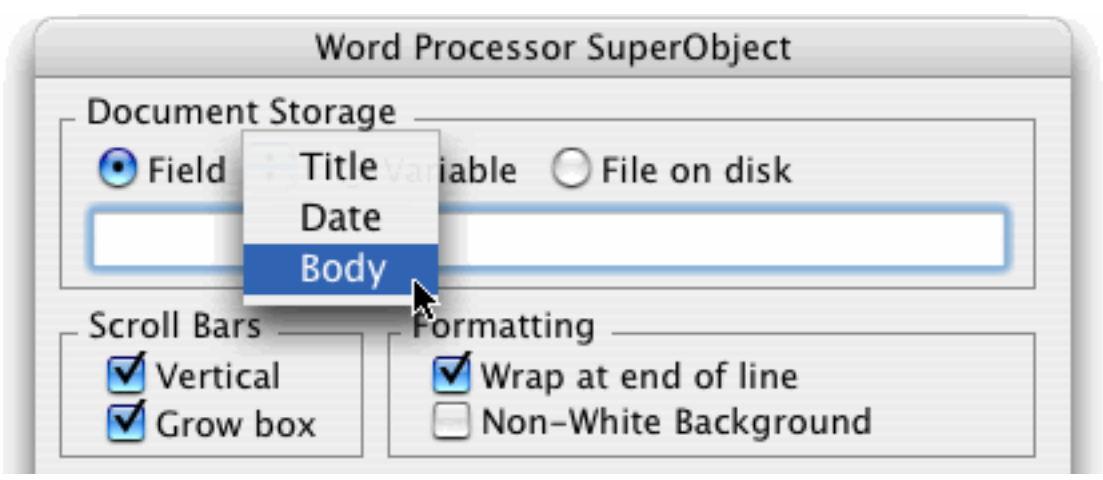
Once the tool is selected, drag the mouse across the form in the location where you want to create the Word Processing SuperObject.



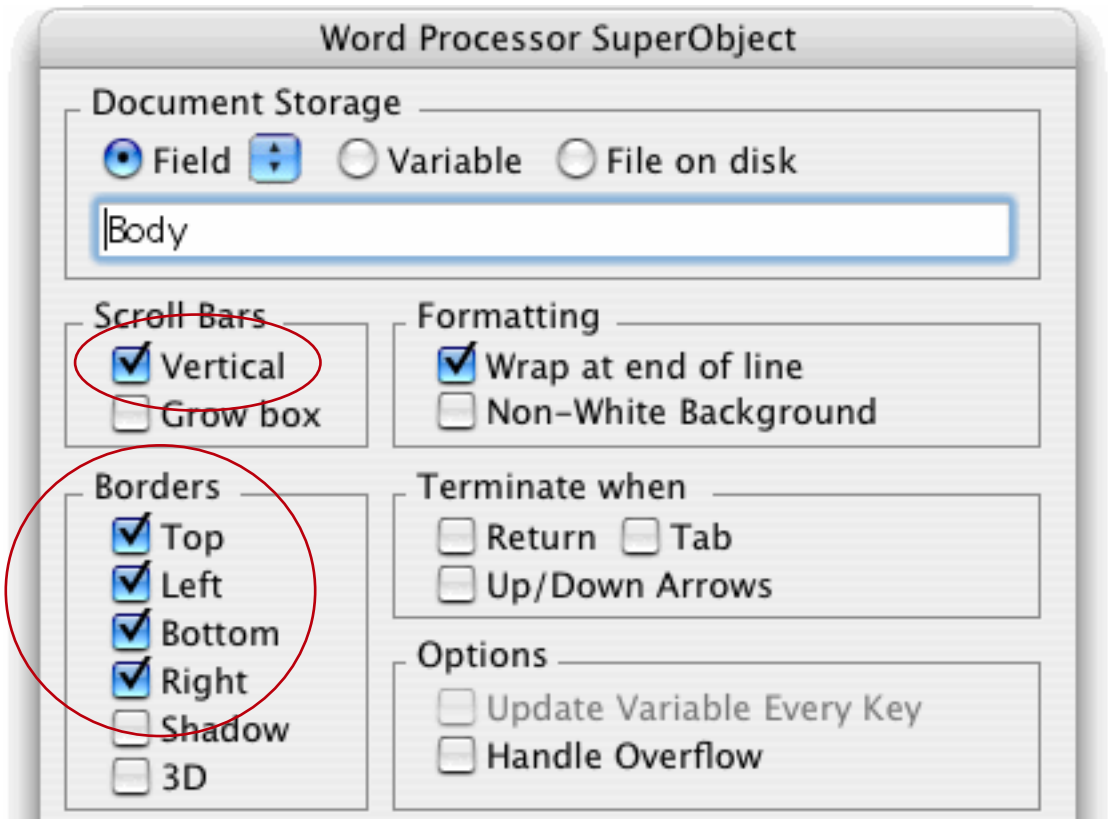
When you release the mouse, the Word Processor SuperObject configuration dialog will appear.



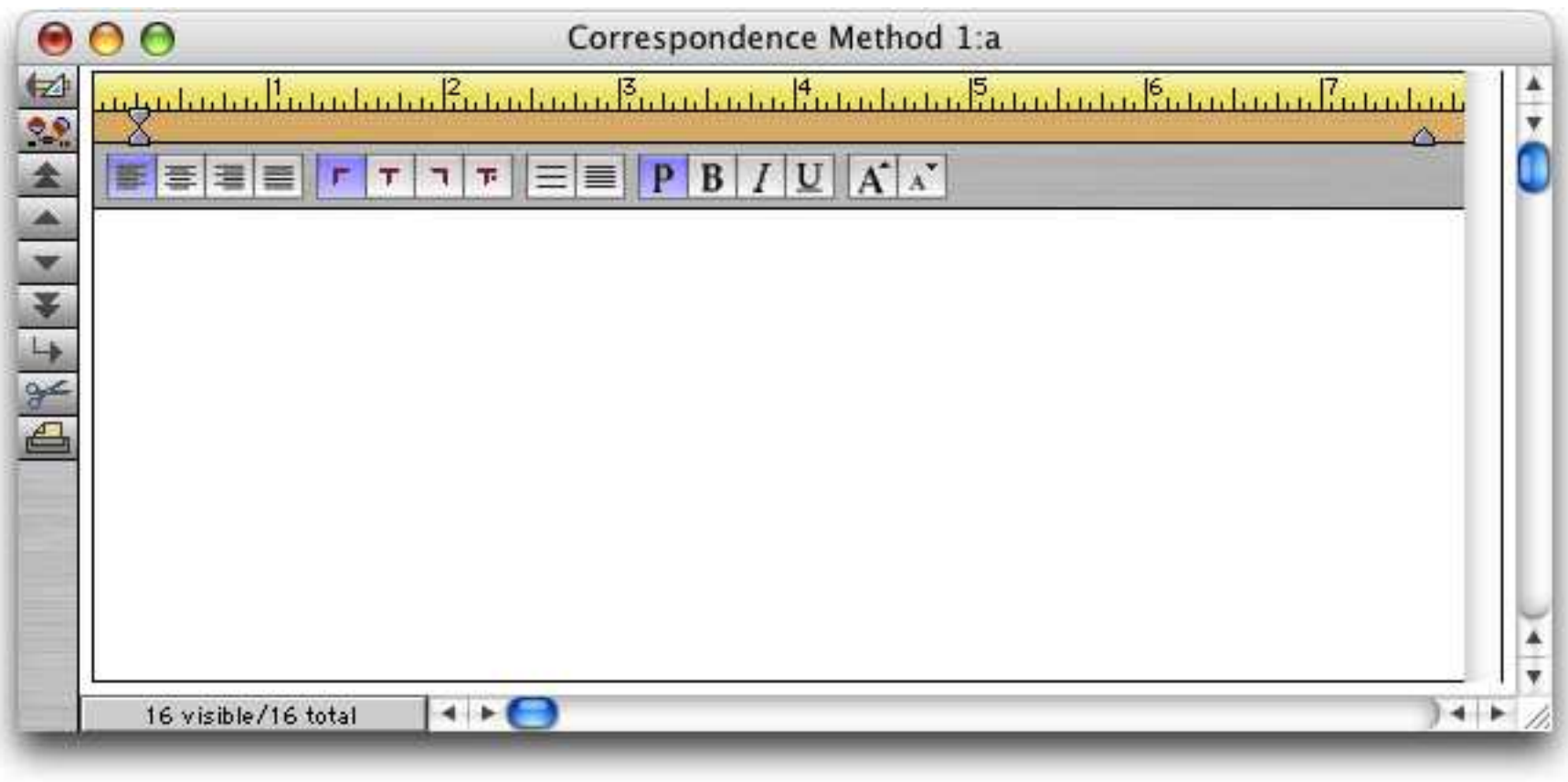
At a minimum you must enter a field name, variable or formula into the dialog. You can use the pop-up menu to select a field. In this database we've created a field, called Body, to hold the word processing information.



For this example we've also turned on the Vertical Scroll Bar and Borders options (all of the available options are discussed in detail later in this chapter).



When the OK button is pressed the new object appears. The top portion of the object contains a ruler that allows you to set tabs, alignment, line spacing and style.

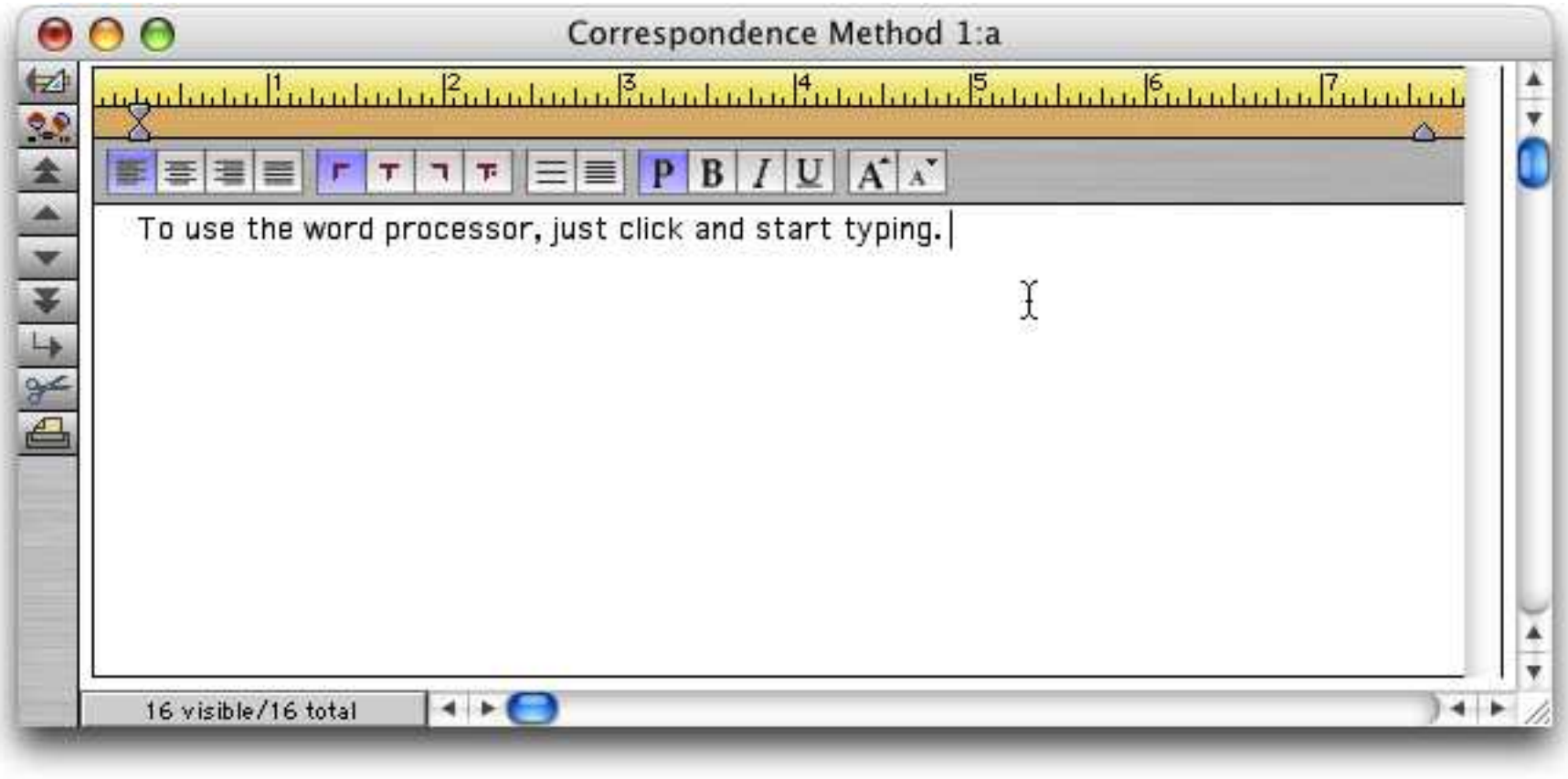


After it has been created you can modify the location and size of a Word Processing SuperObject just like any other object. To change any of the object attributes (scroll bars, border, formatting etc.) select the **Pointer** tool and double click on the object. The configuration dialog will appear again. Make your changes and press the **OK** button.

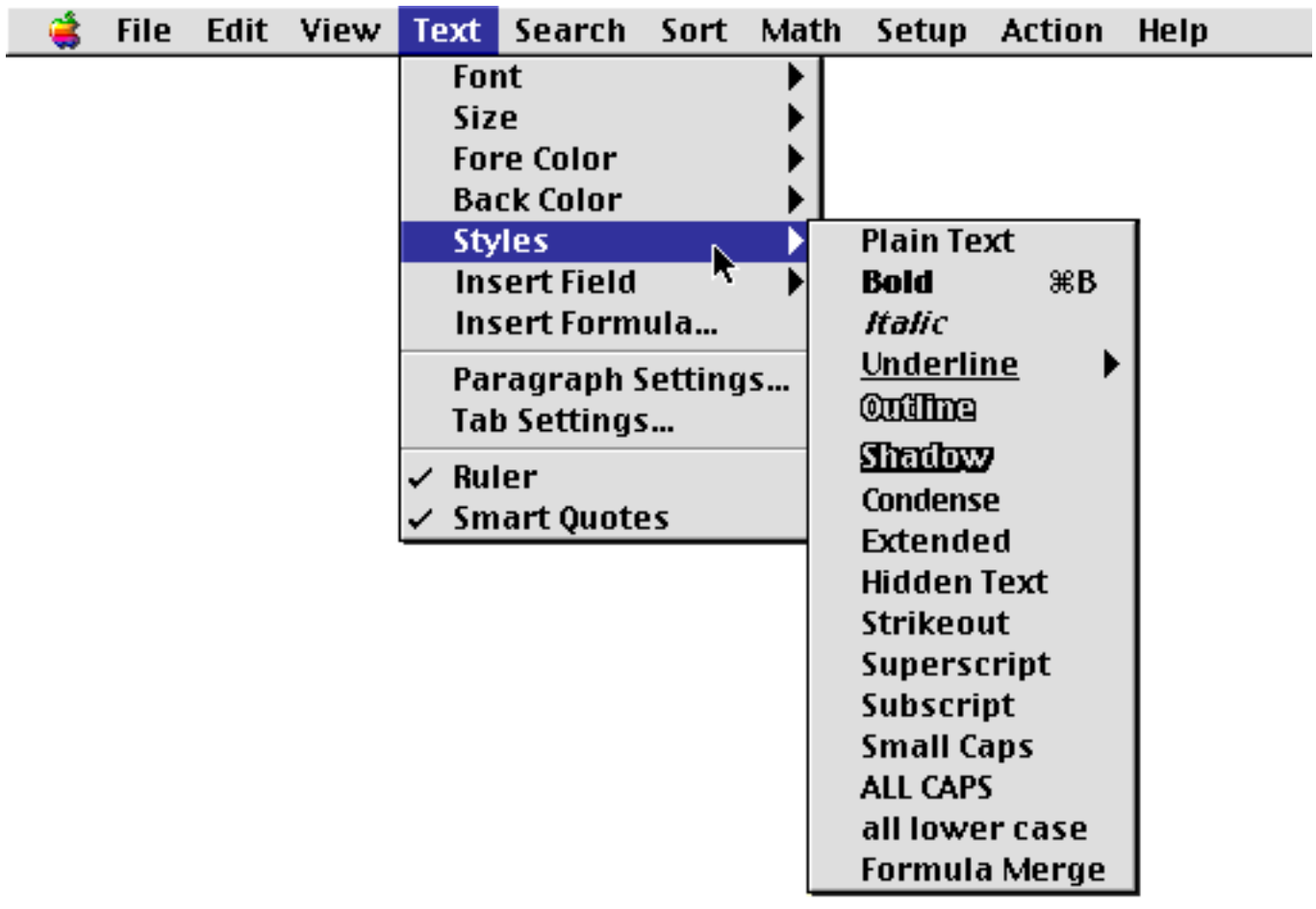
To actually use the word processor you need to switch to Data Access Mode (see "Form Modes: Data Access vs. Graphic Design" on page 485). To learn more about how to configure the Word Processor for your specific application see "Configuring the Word Processor" on page 696.

Using the Word Processor

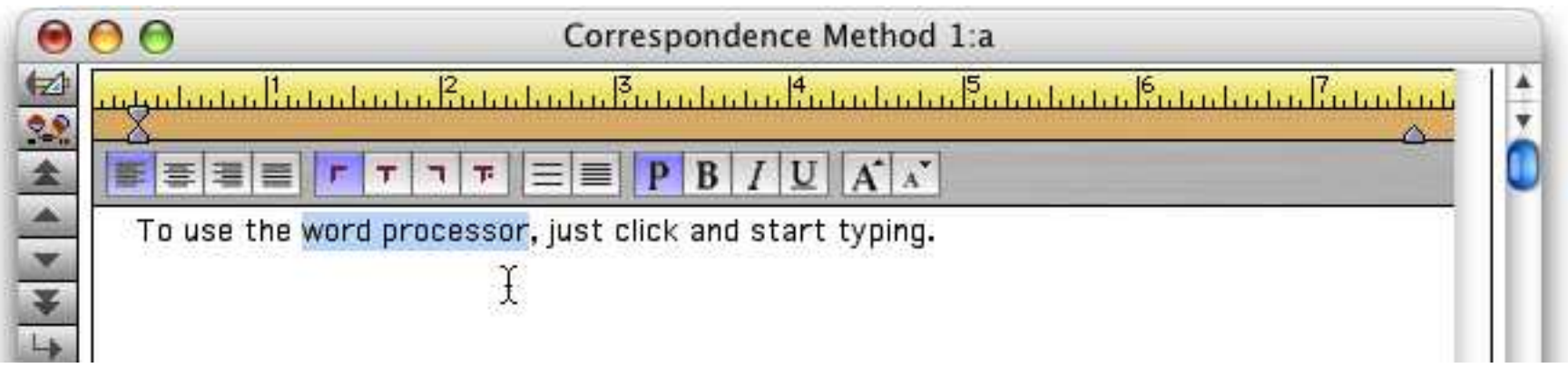
Once the word processing object is set up you can switch the form to Data Access Mode (see “Form Modes: Data Access vs. Graphic Design” on page 485) and start using the word processor. To start editing, simply click in the word processor object and start typing.



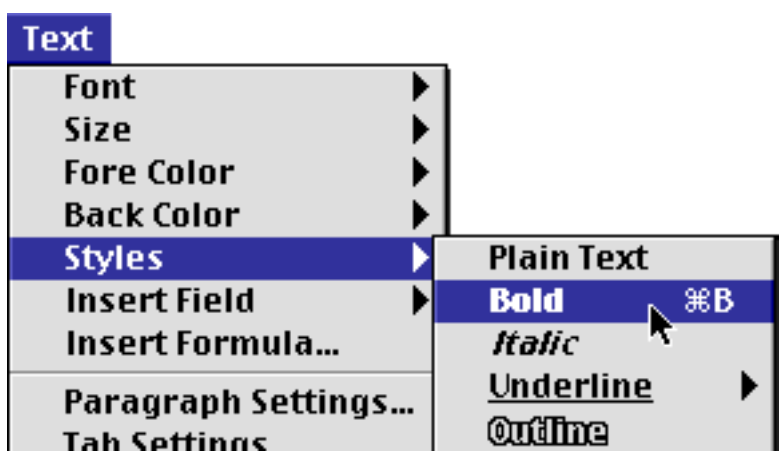
Whenever the word processor is active, an extra menu appears in the menu bar: the **Text** menu. This menu contains the options for formatting text and controlling the word processor.



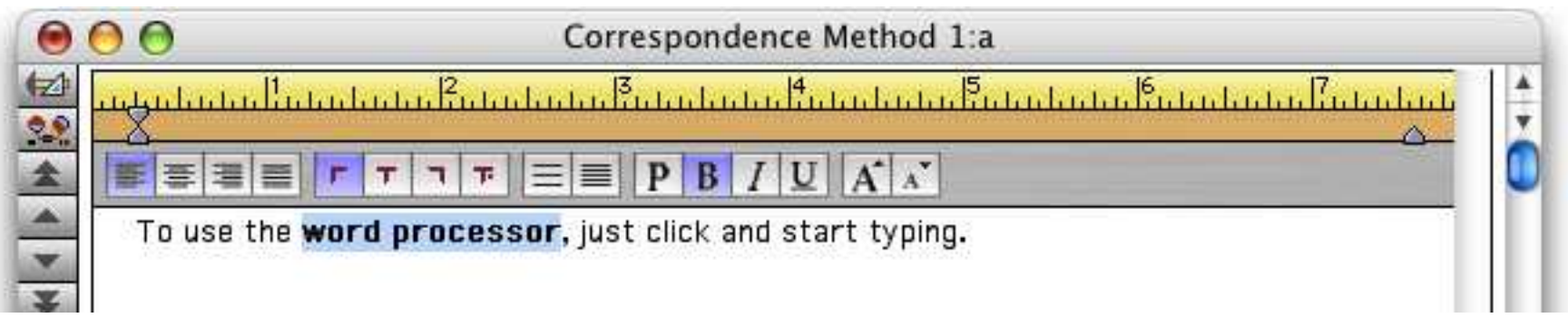
At the top of the Text menu are five submenus: **Font**, **Size**, **Styles**, **Fore Color**, and **Back Color**. To change the appearance of a section of text, start by selecting the text.



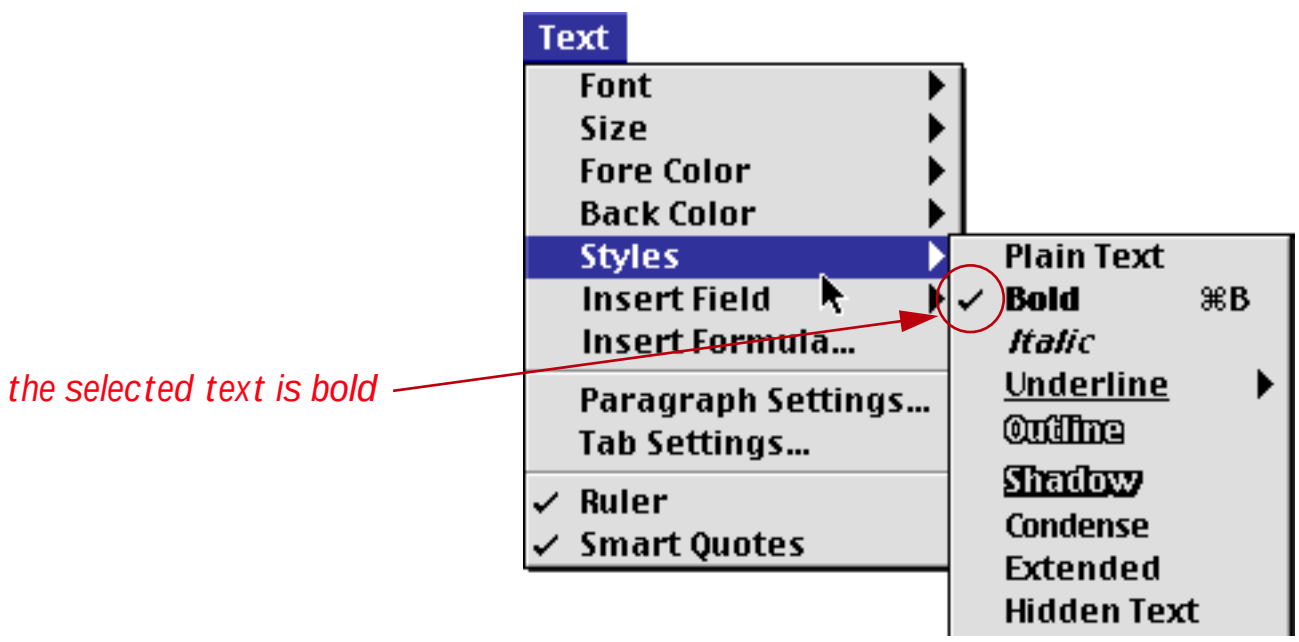
Next, choose the desired options from the **Text** menu.



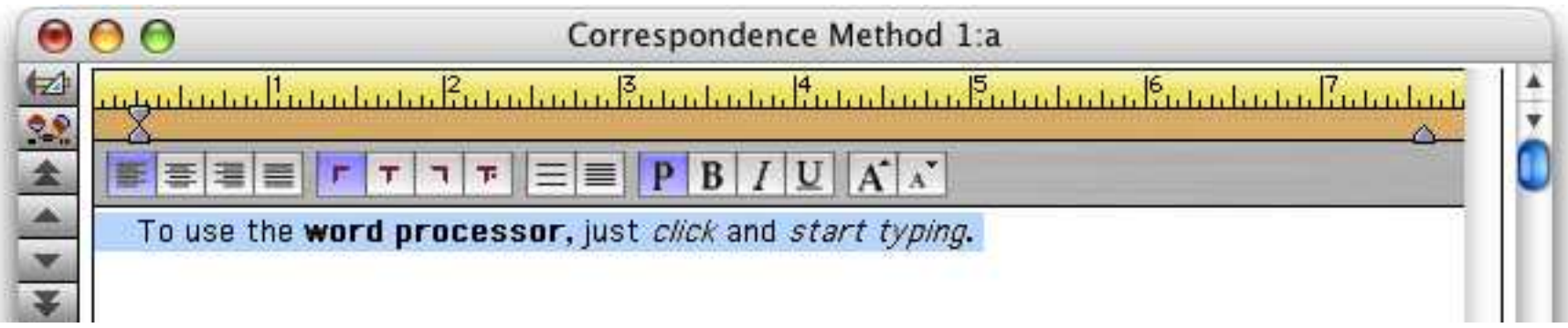
The selected text will take on the new font, size, color or style (in this case **bold**).



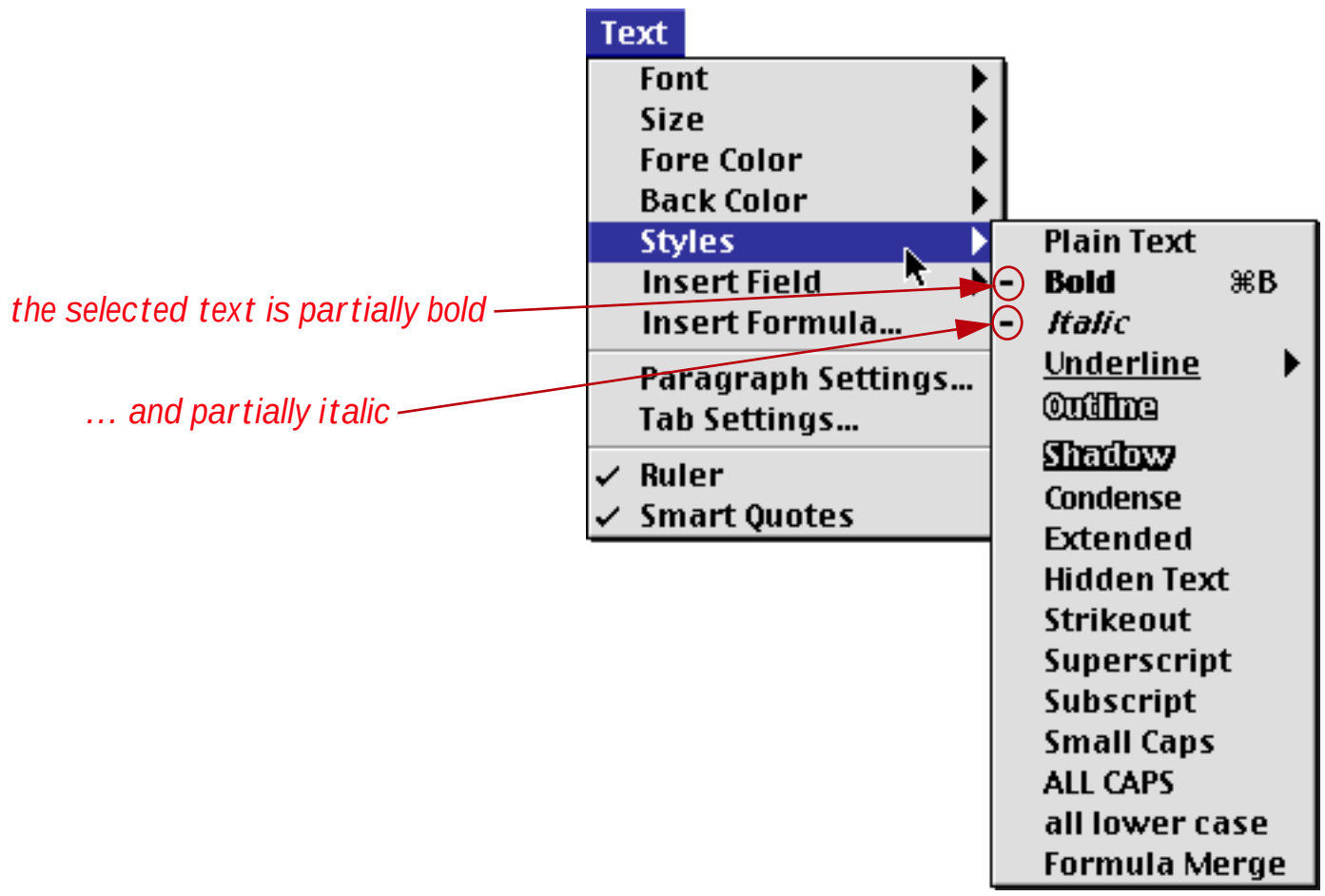
The submenus also display the status of the selected text. If the text is all the same, a checkmark will appear next to the font, size, or style.



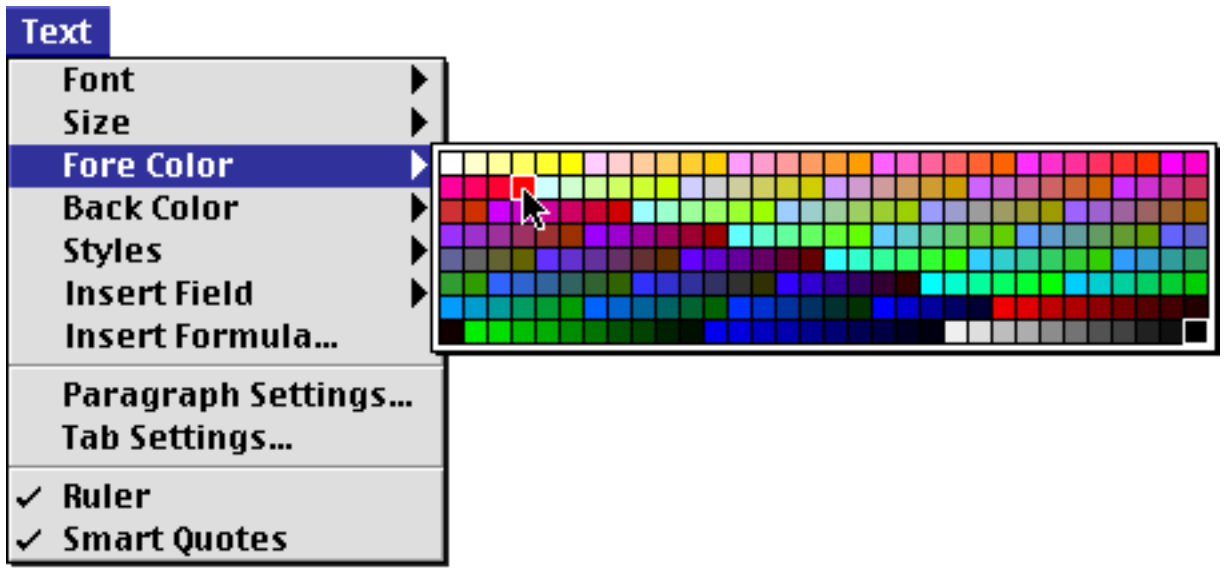
If the text contains several different styles in different parts of the text a dash will appear next to each font, size or style that is contained within the selection. For example, the selected text in the illustration below contains both bold and italic text.



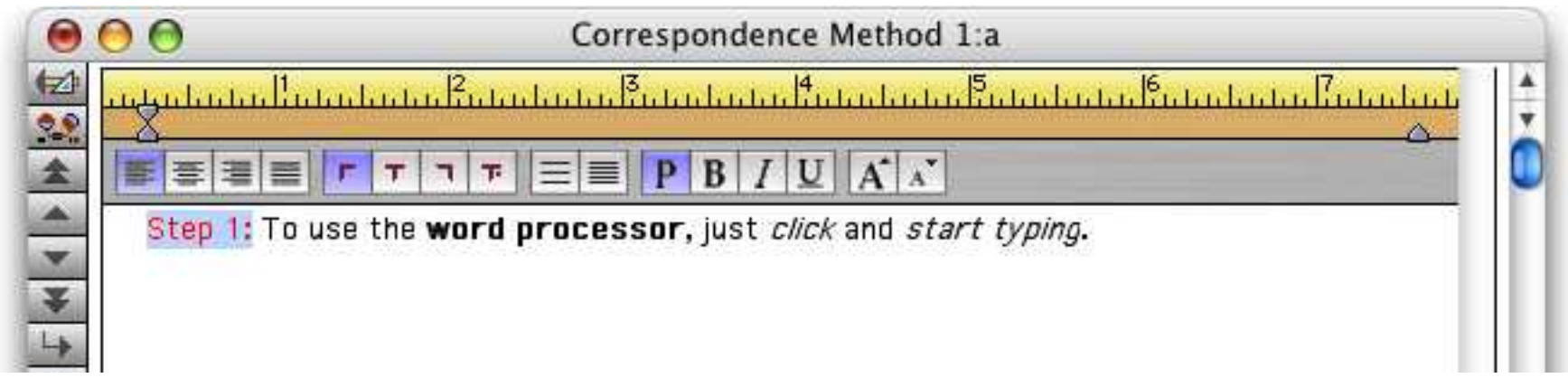
Pulling down the **Text** menu shows all of the different styles in the current selection. (The **Font** and **Size** submenus will also show all of the options used in the selection.)



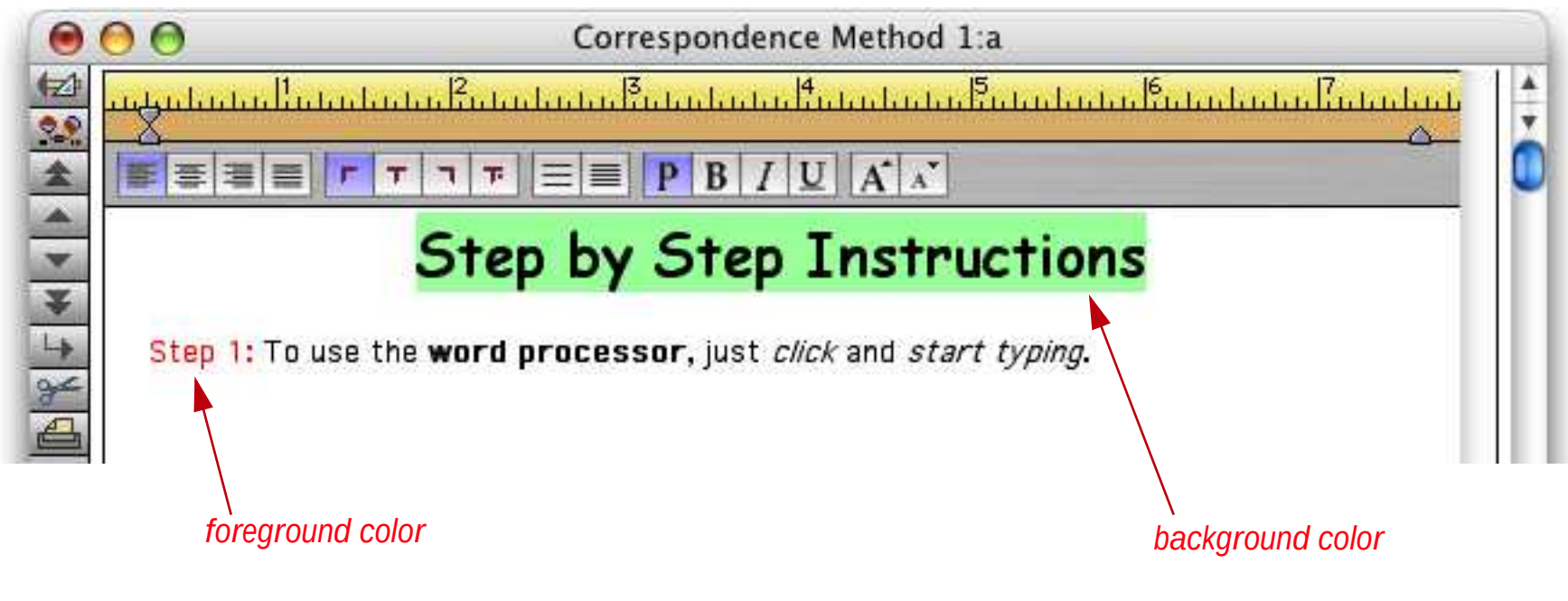
The **Fore Color** submenu changes the color of the text.



For example, you can make the selected text red. You can choose any color in the 256 color palette.



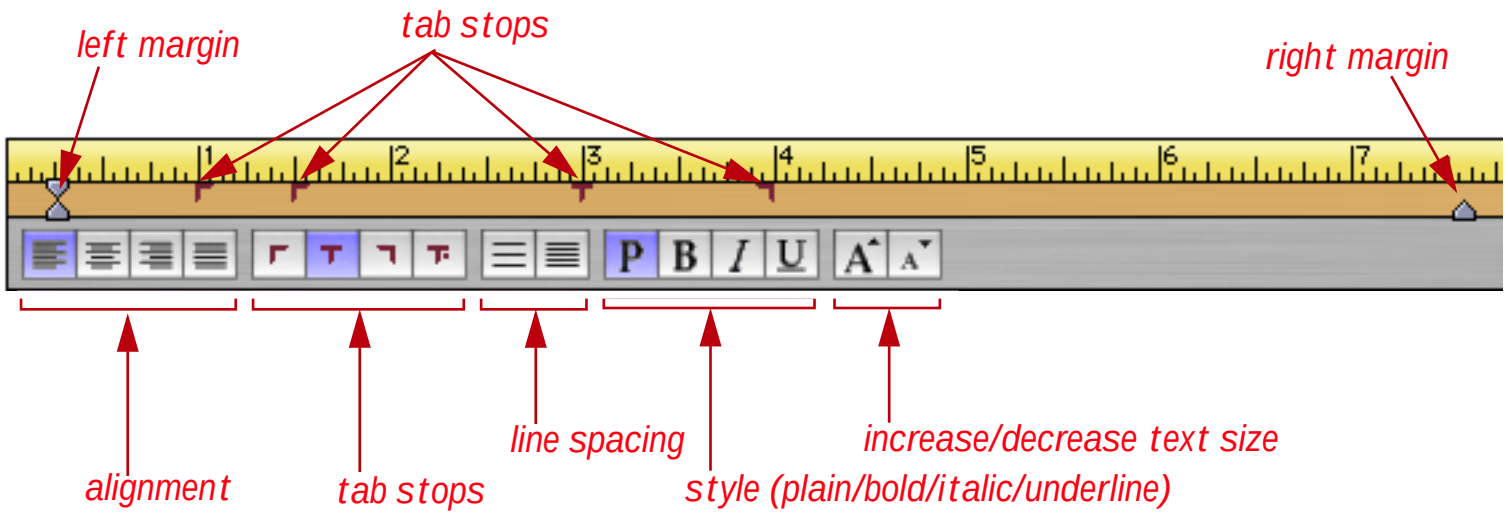
The **Background Color** submenu works the same way, but changes the background color of the text.



The **Font** and **Size** menus work the same way as the corresponding menus for other text objects (see "Font" on page 529 and "Text Size" on page 531). However, the Word Processor SuperObject allows multiple fonts and sizes within a single object (as shown in the illustration above). In addition, the Font and Size are set when in Data Access Mode, not Graphics Mode.

The Ruler

The ruler controls how paragraphs are formatted. The ruler allows you to set margins, tab stops, alignment, line spacing and style.



Since the ruler controls how paragraphs are formatted, the changes you make to the ruler are always applied to entire paragraphs. If you click on a paragraph and make a change to the ruler, the change will affect the entire paragraph. If you want to change the formatting of multiple paragraphs, select at least one character in all of the paragraphs you want to change before adjusting the ruler.

The word processing ruler always uses the same measurement system used by the form containing the word processor. If the form is set to inches, the word processing ruler will also work in inches. If the form is set to centimeters, the word processing ruler will also work in centimeters. To change the measurement system, switch into Graphics Mode (see "Form Modes: Data Access vs. Graphic Design" on page 485) and click on the box in the upper left hand corner of the ruler (see "Rulers" on page 506), then switch back to Data Access Mode.

To turn the ruler on or off, use the **Ruler** command in the Text menu (you must be in Data Access Mode and editing the text within the object to use this command).

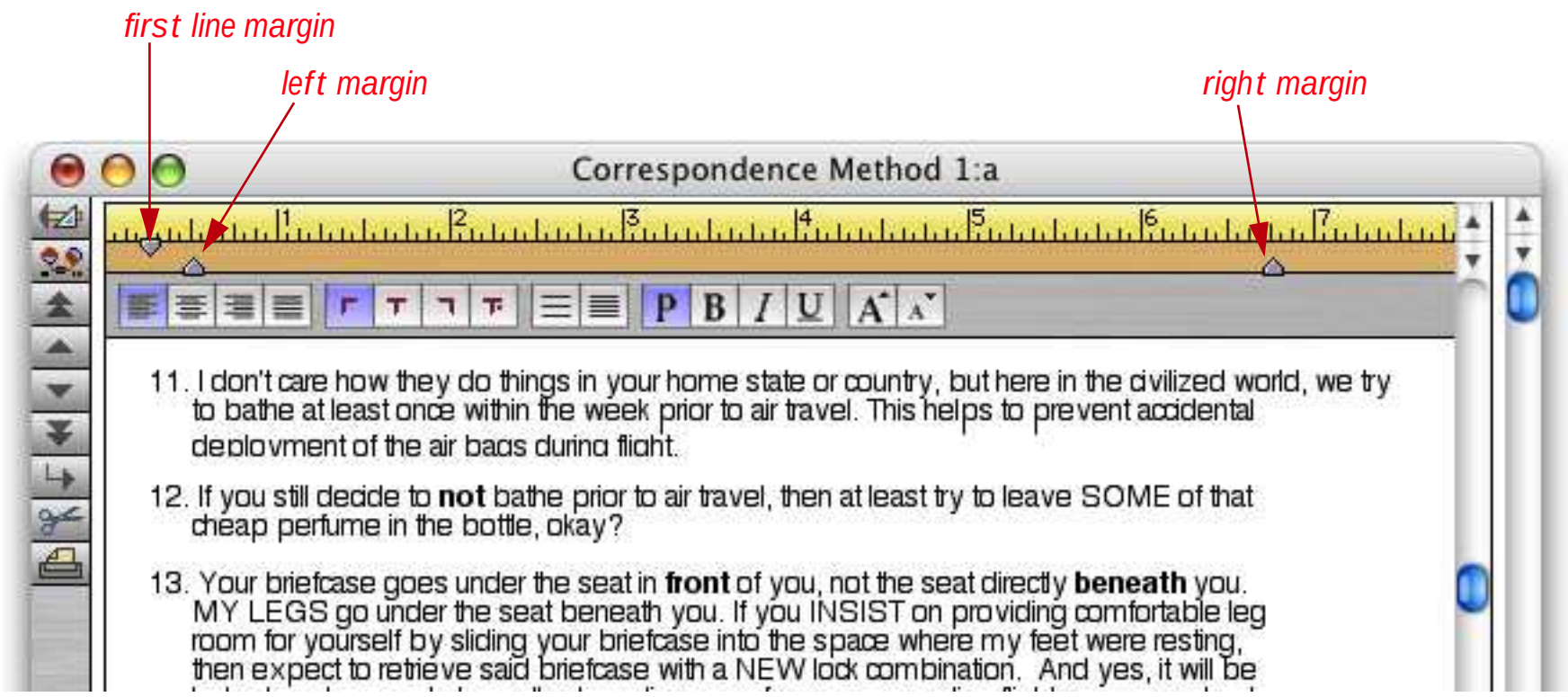


Here's what the word processor looks like with the ruler turned off.

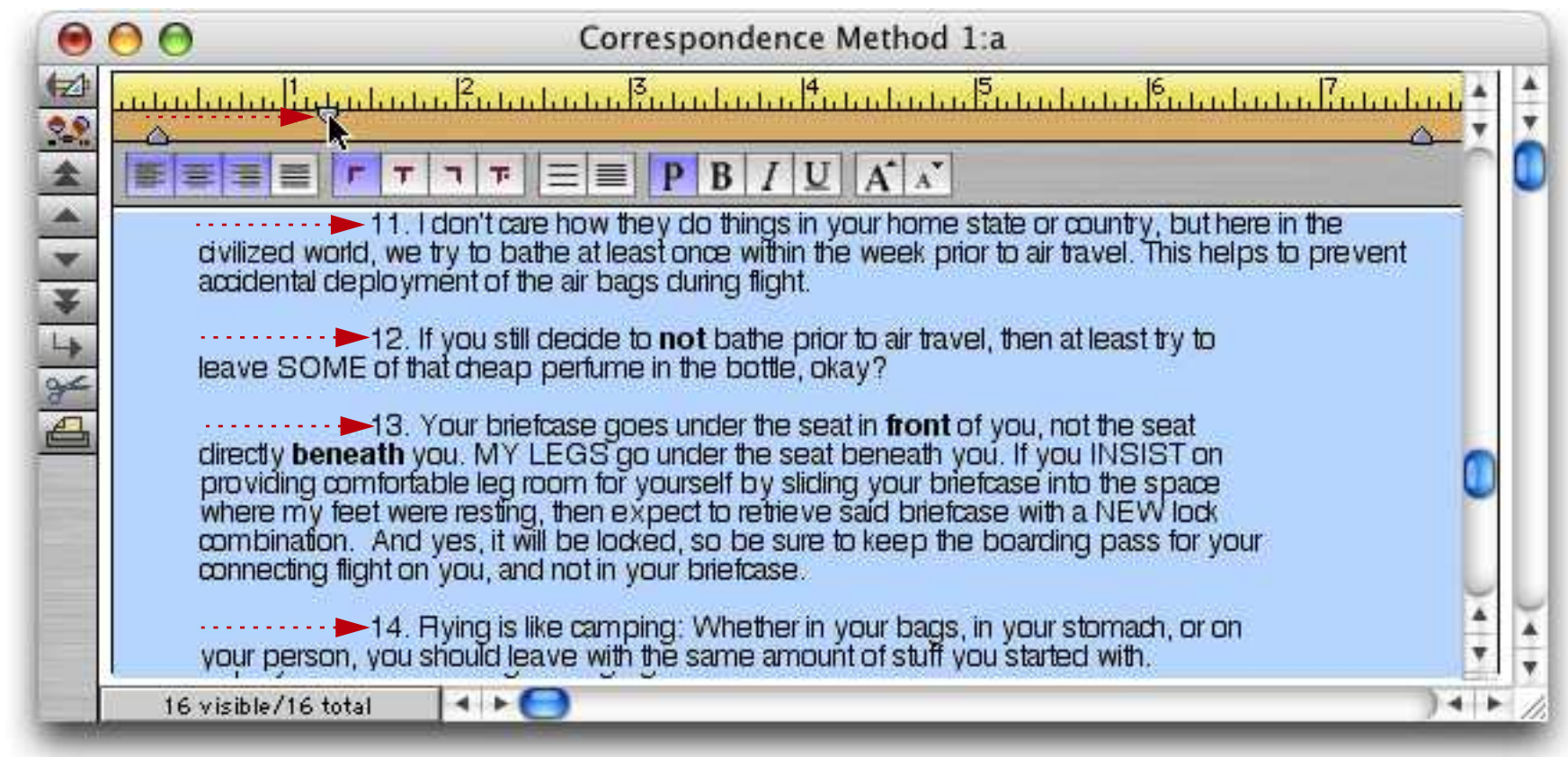


Margins (Indents)

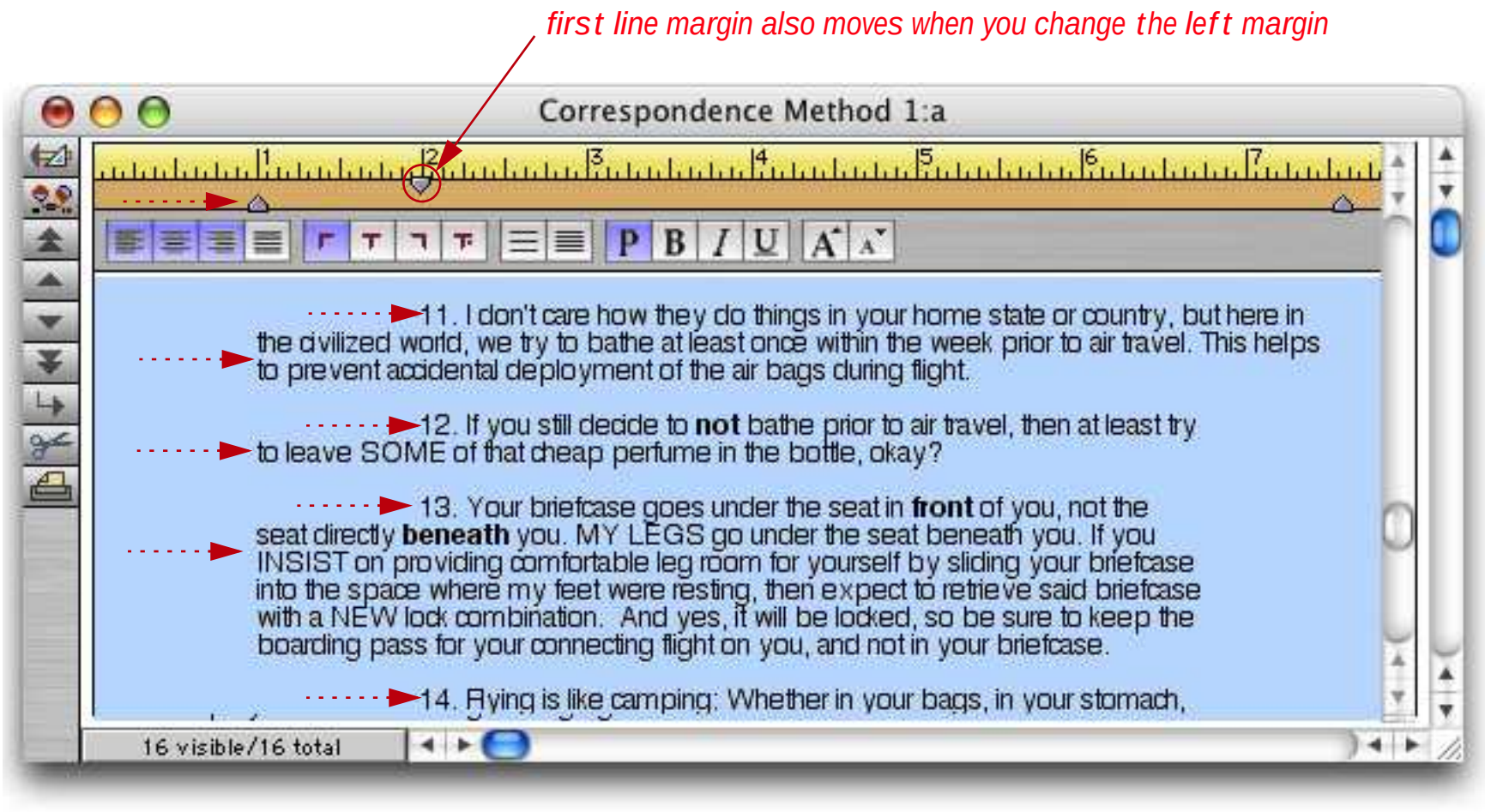
Each paragraph has three margins: **First Line**, **Left**, and **Right**.



The **First Line** margin is the distance from the left side of the word processing object to the beginning of the first line in the paragraph. Drag the upper margin triangle to adjust the first line margin. The text itself will adjust dynamically as you drag.

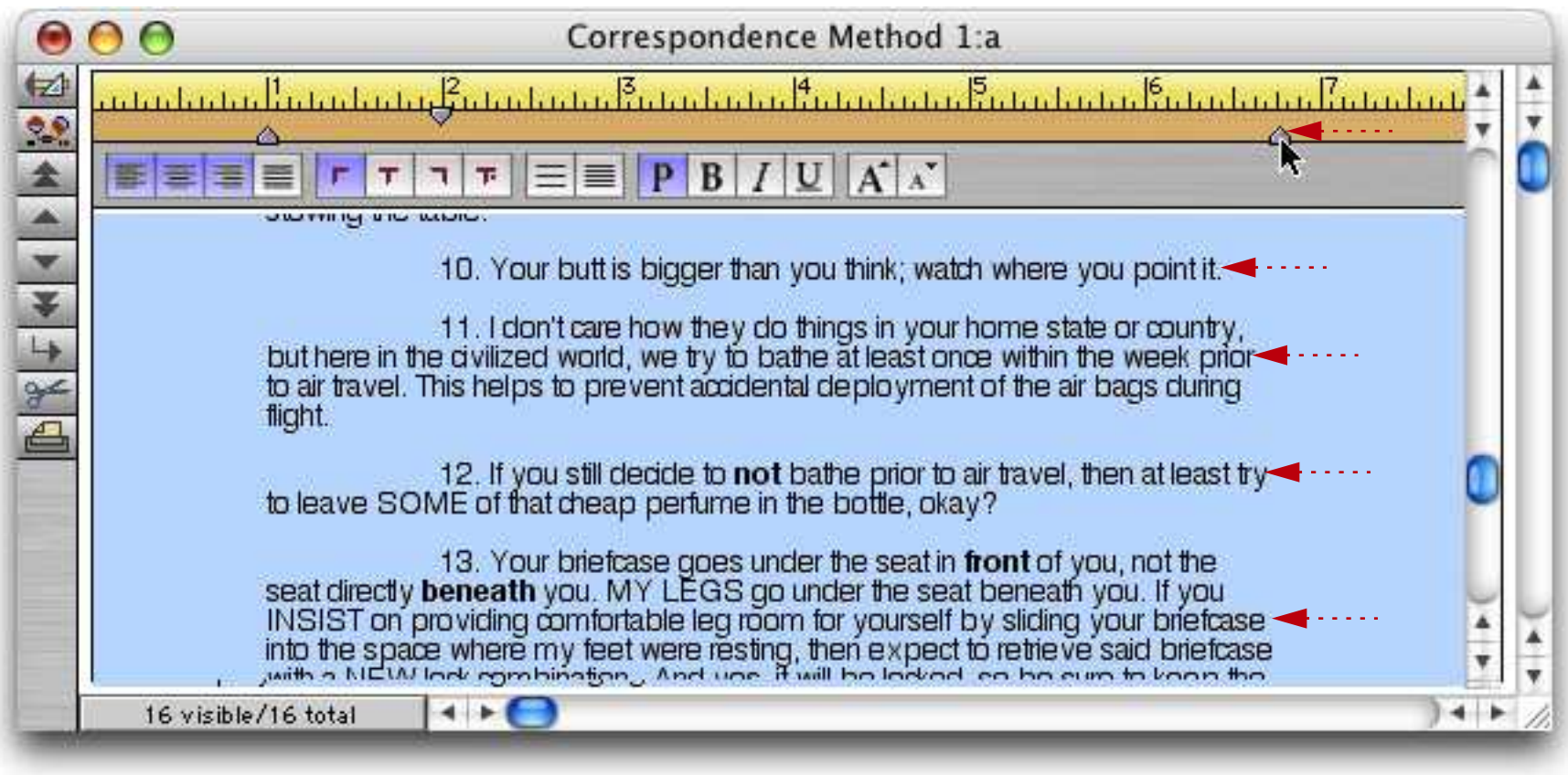


The **Left** margin is the distance from the left side of the word processing object to the beginning of the rest of the lines in the paragraph. Drag the lower margin triangle to adjust the left margin.

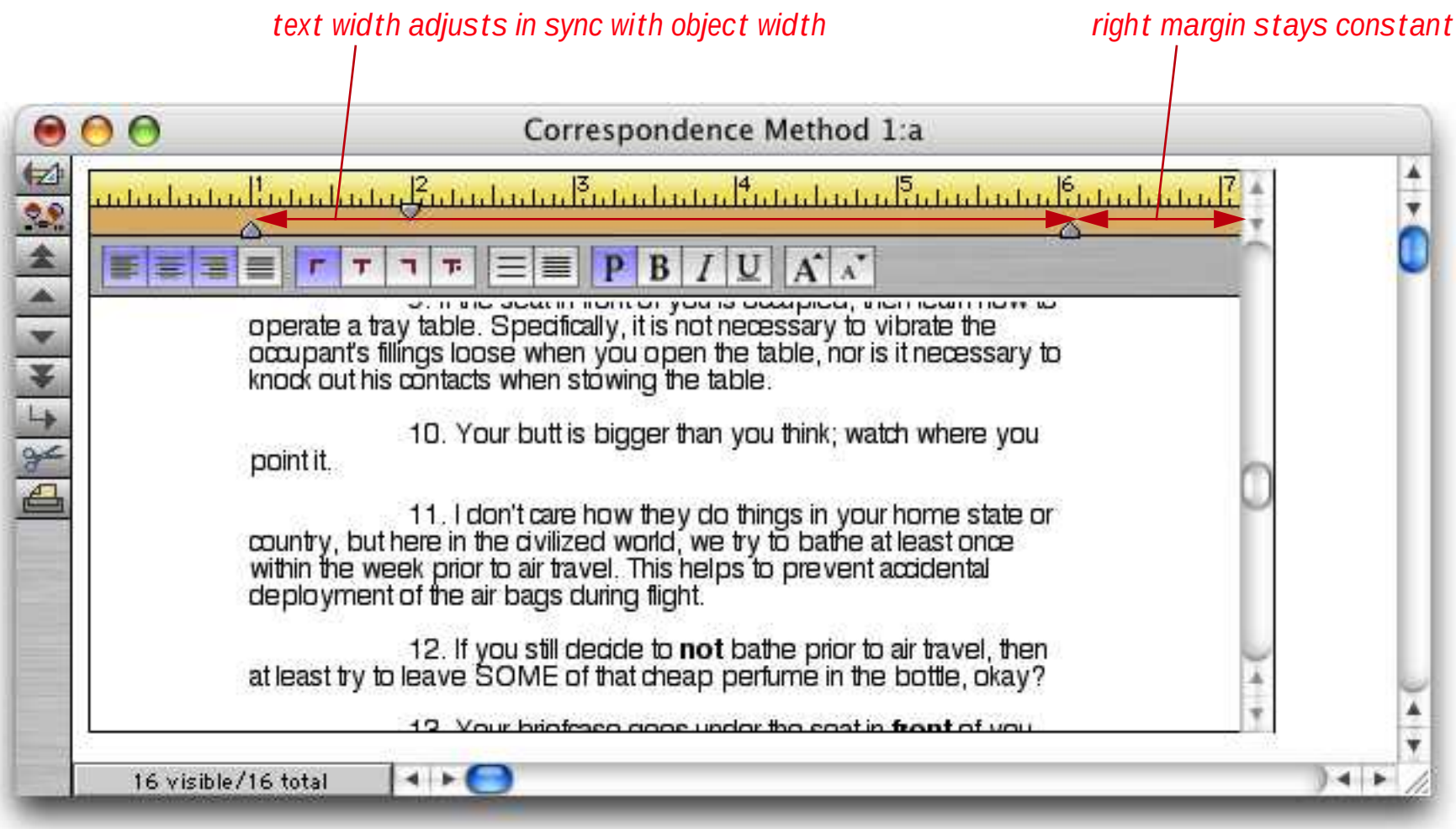


When you drag the left margin, the first line margin also moves to maintain the same spacing between the two margins as shown in the illustration above. It's usually best to set the left margin first, then set the first line margin.

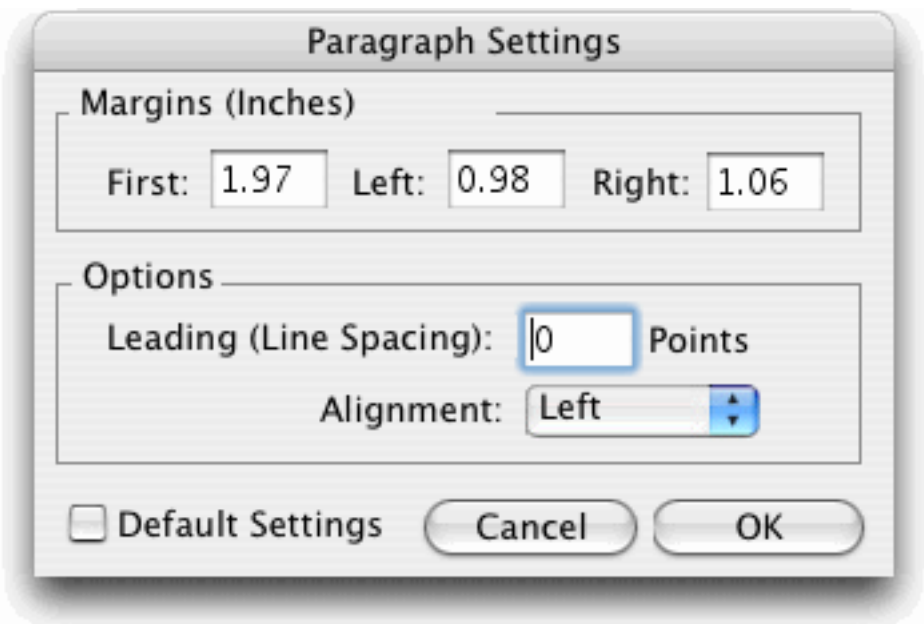
The **Right** margin is the distance from the right side of the word processing object to the end of the each line in the paragraph. Drag the triangle symbol to adjust the right margin.



If the width of the word processing object changes (for example if the form adjusts when the window size is changed or if the document is printed using a different form) the width of the paragraph may change. However, the margin from the right edge will not change.



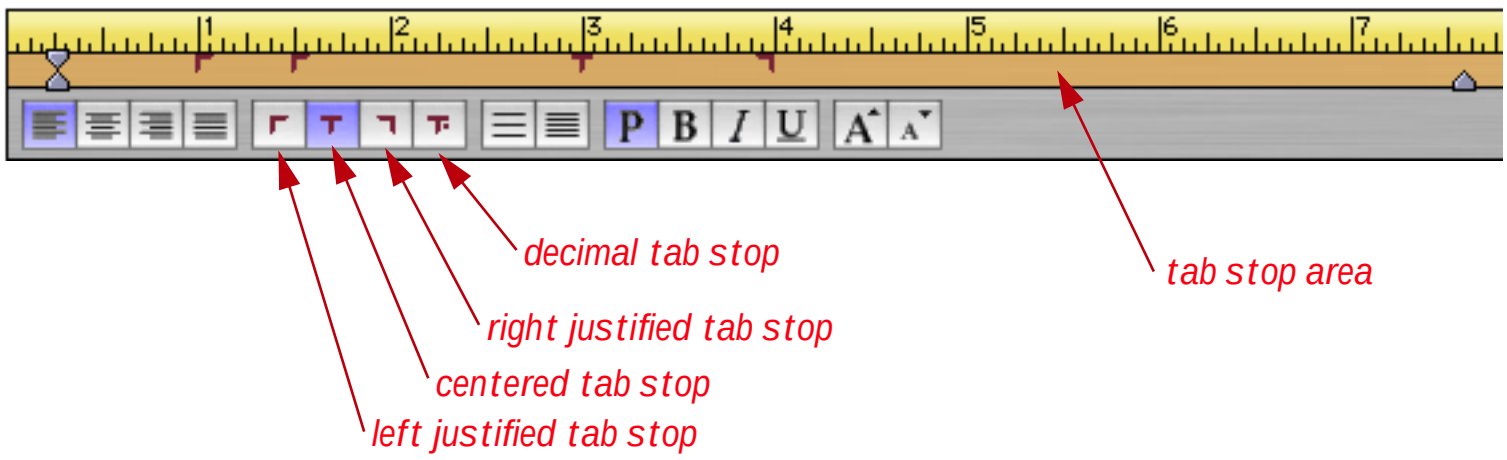
In addition to setting the margins with the ruler you can also set them numerically using the **Paragraph Settings** dialog. You can open this dialog from the Text menu or by double clicking anywhere in the ruler (except on a button or tab stop).



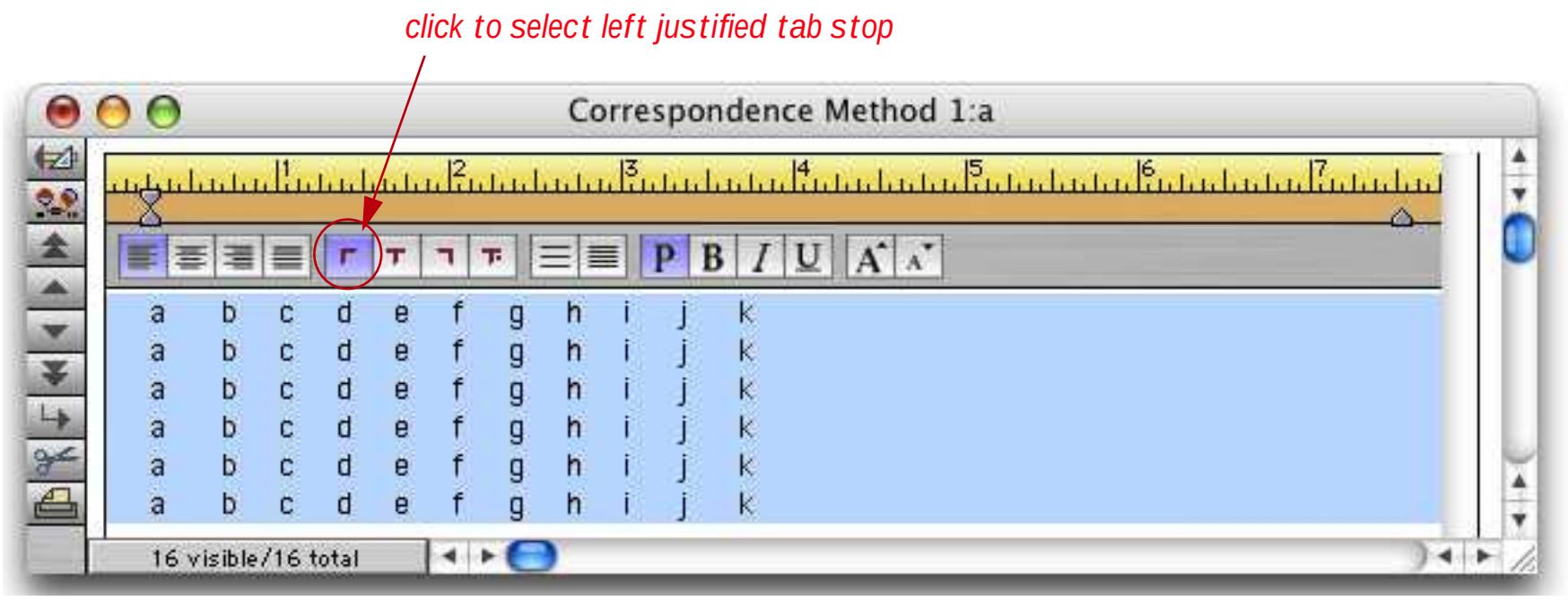
Simply enter the numeric values for each indent and press **OK** to change the paragraph formatting.

Tab Stops

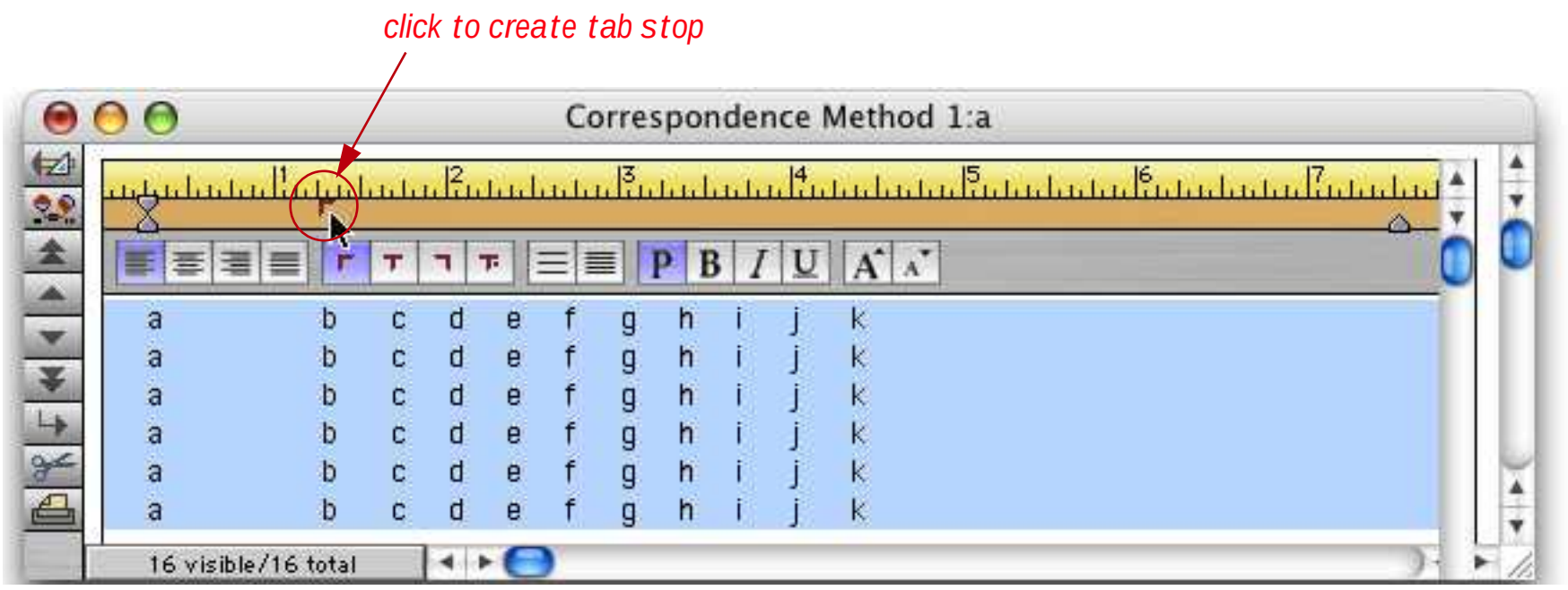
The word processor supports four types of tabs: **left**, **center**, **right**, and **decimal** tabs. There are four buttons in the ruler for selecting the type of tab you want to create.



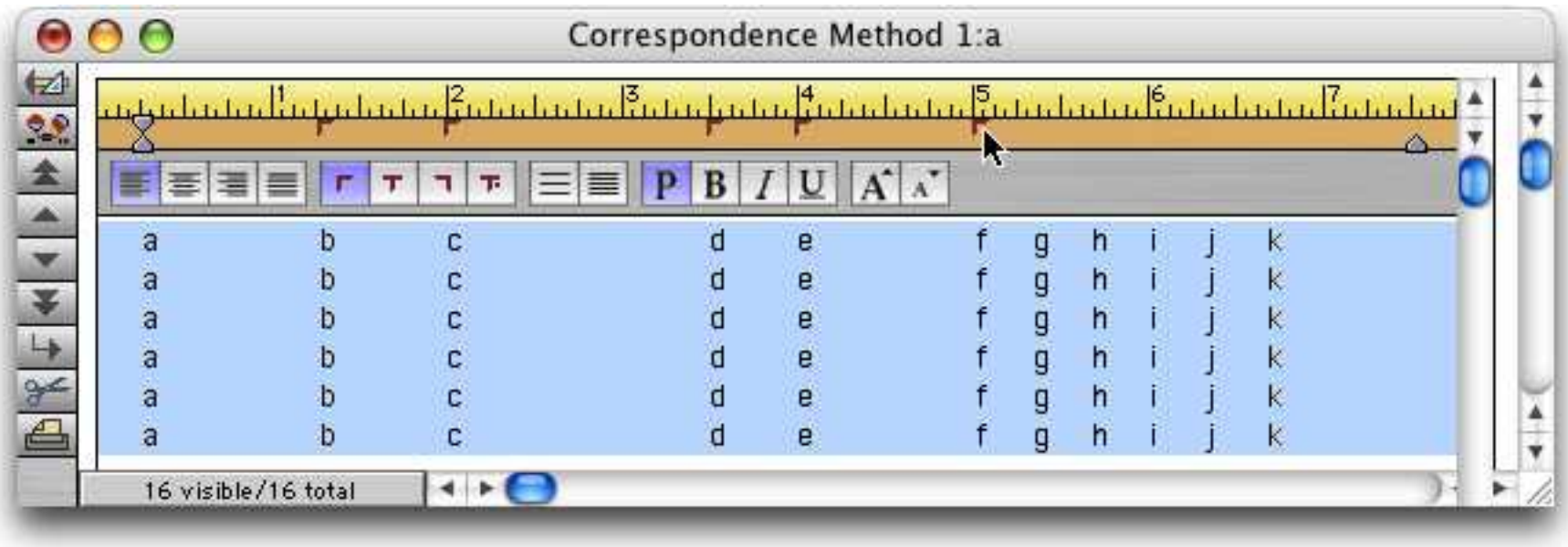
If there are no tab stops at all Panorama automatically creates 3 tab stops per inch, as shown below. To create a new tab stop, first select the type of tab stop by clicking on one of the four tab stop buttons.



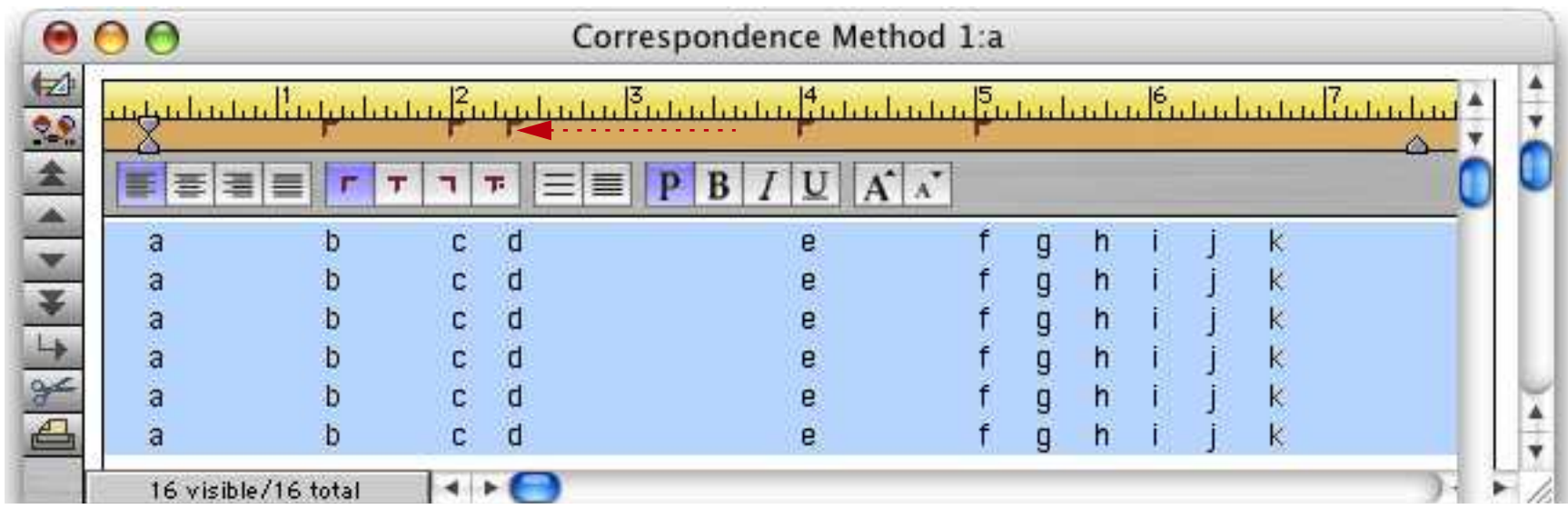
To create a tab stop, click anywhere in the tab stop area (just below the ruler itself).



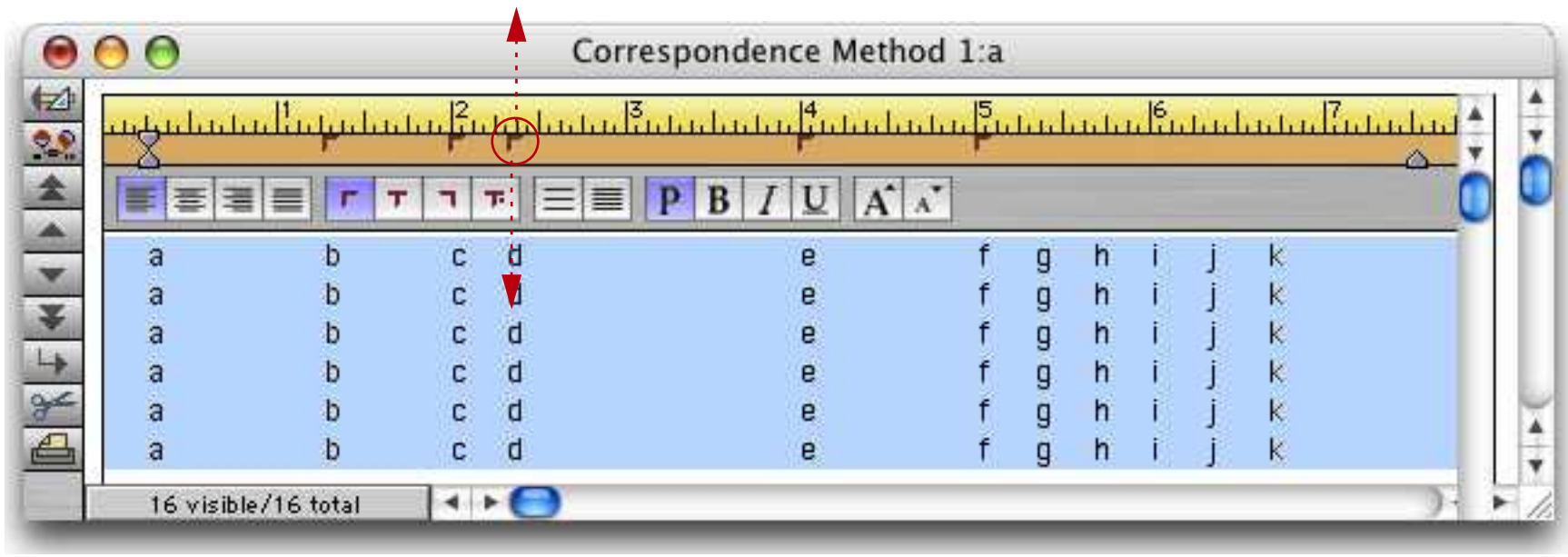
You can click additional times to create additional tab stops. Tab stops may be created in any order (not just left to right).



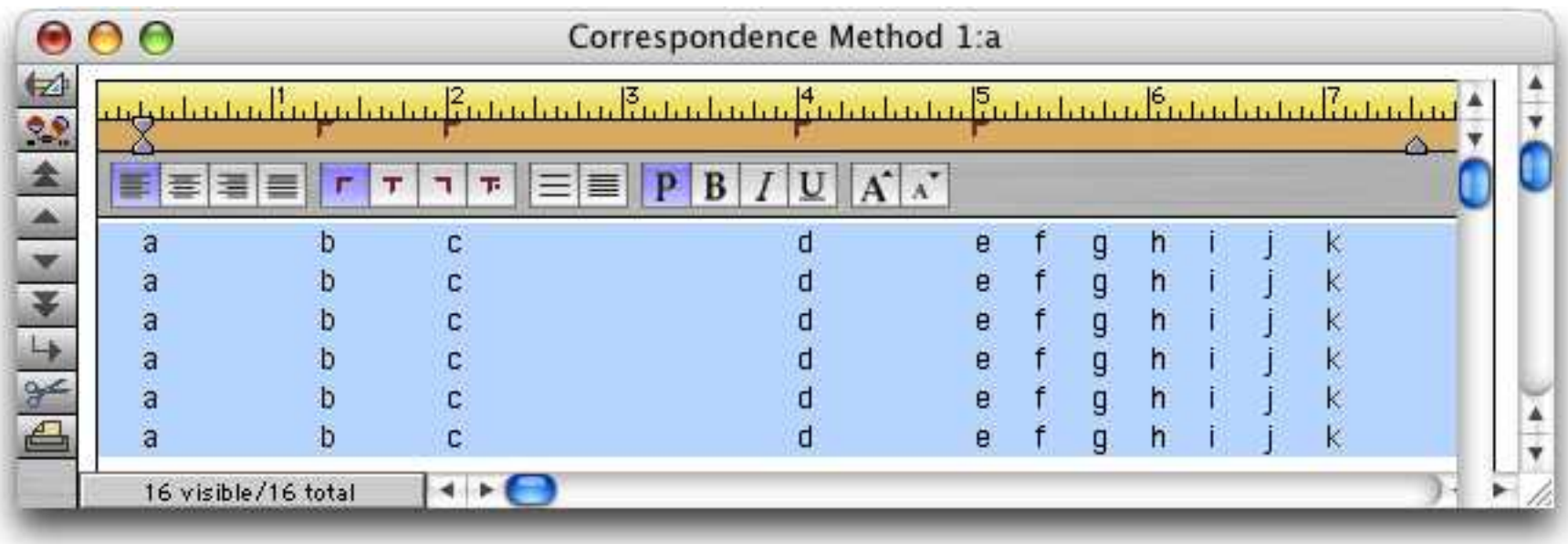
To move a tab stop, simply drag it to a new position.



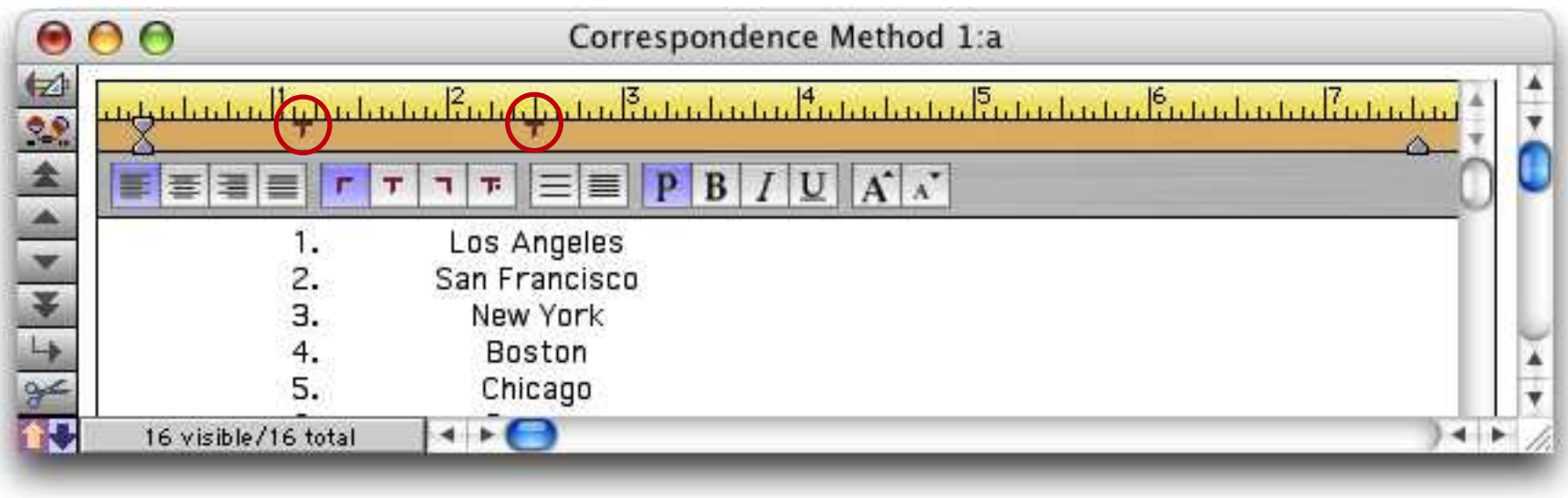
To delete a tab stop, drag it out of the tab area (either above or below).



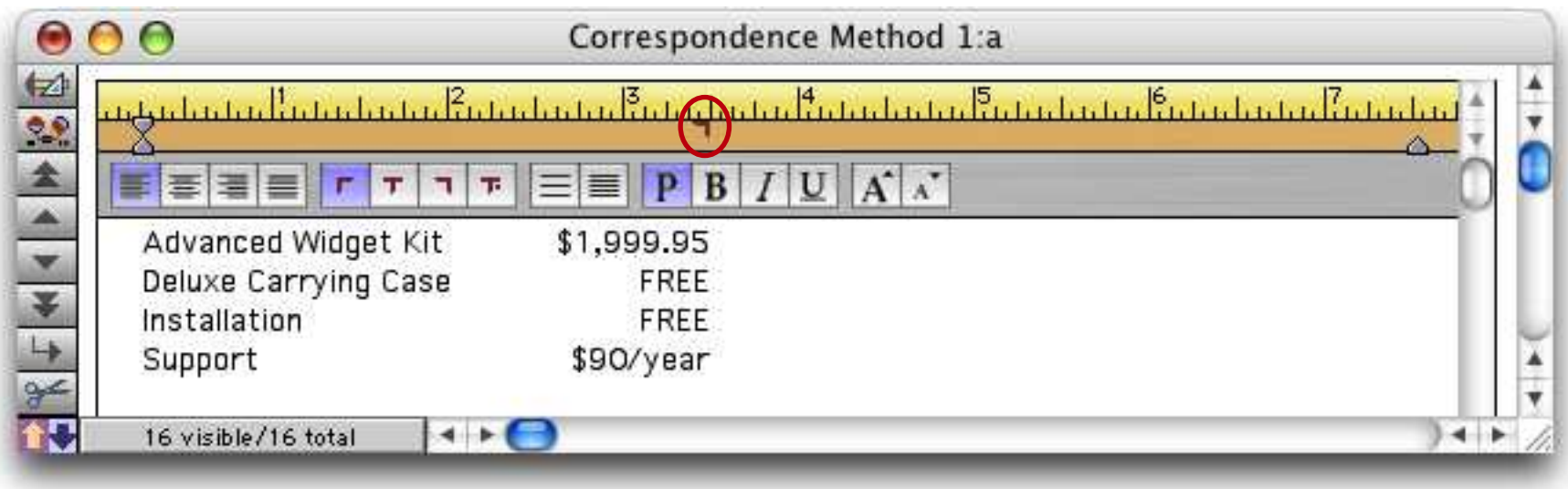
The tab stop disappears and the columns shift into their new positions.



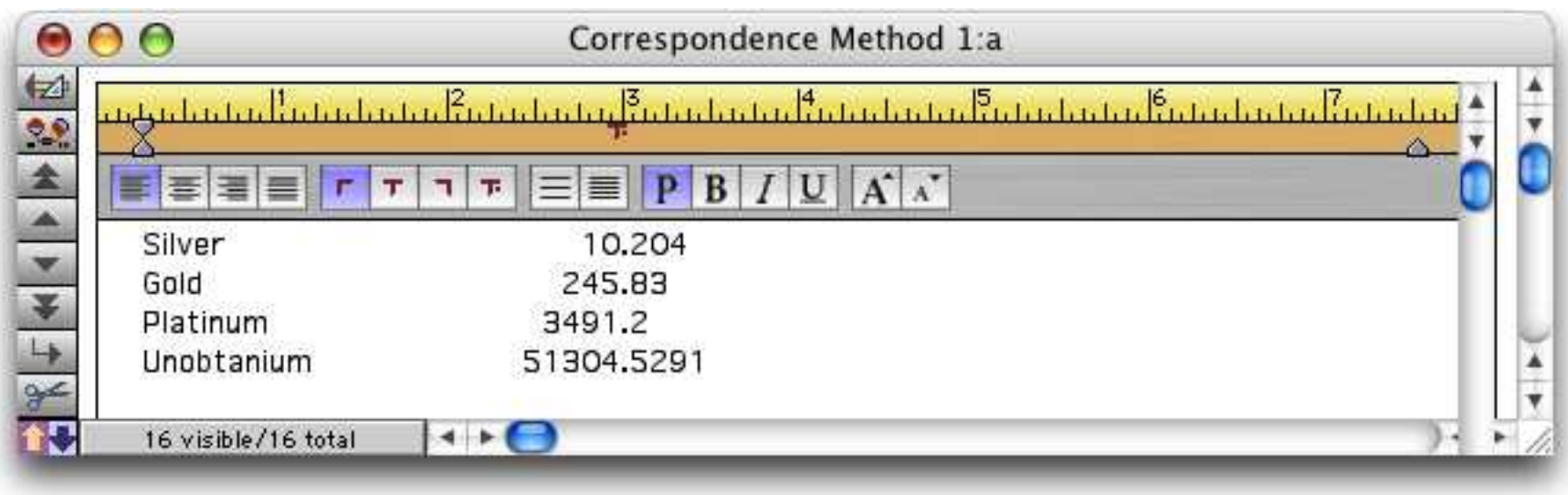
This illustration shows the use of a **centered tab**.



This illustration shows the use of a **right justified tab**.



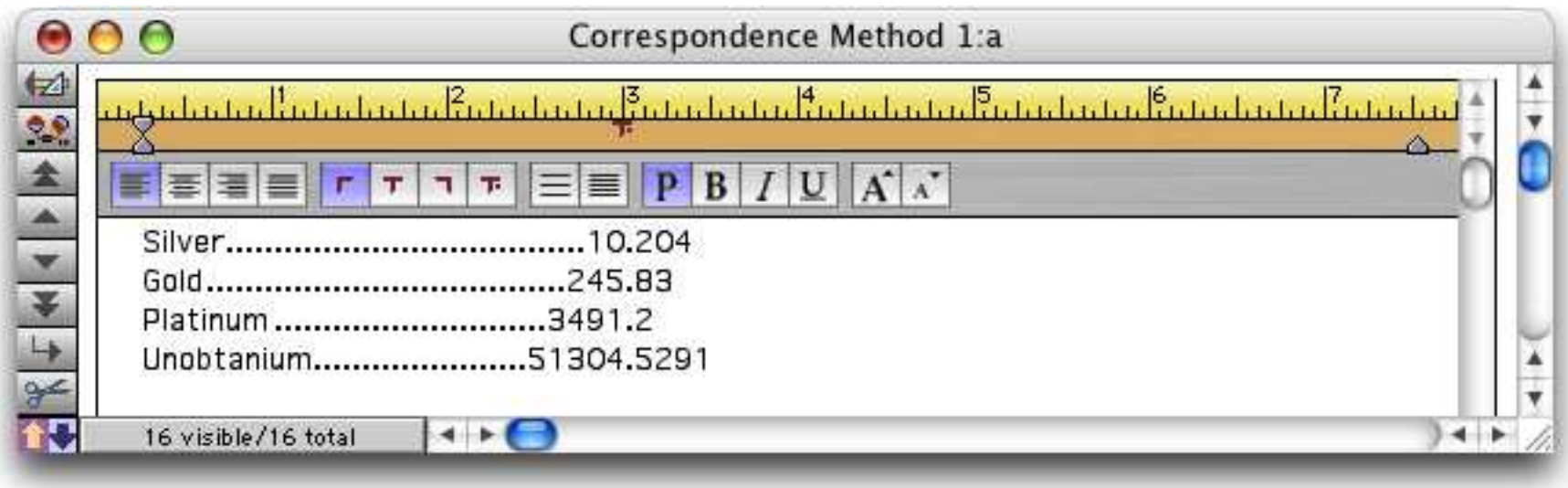
And finally, a **decimal tab stop**, so named because the decimal points line up no matter how many digits are to the left or the right.



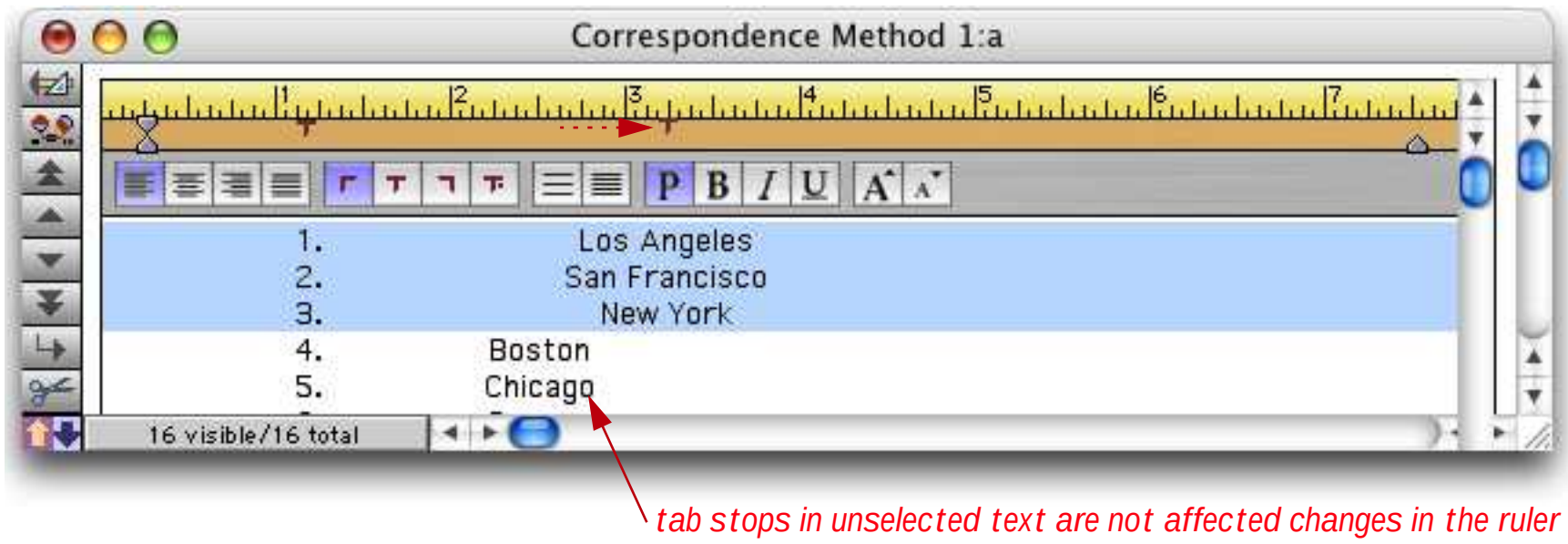
For precise tab adjustments you can use the **Tab Settings** dialog. You can open this dialog three ways: 1) by double clicking on a tab stop, 2) by holding down the Command key (Mac) or Control key (PC) and clicking on a tab stop, or 3) using the **Tab Settings** command in the Text menu (in which case it will operate on the last tab that you adjusted). This dialog allows you to set the precise position of the tab, the type of tab (this is the only way you can change the type of an existing tab) and the tab leader.



The tab leader is a character that will automatically repeat to fill the space leading up to the tab. The tab leader helps draw the eye across the page. Common tab leaders include periods (shown below), dashes, and asterisks (but you may use any single character you want). Use the **Tab Settings** dialog to set the tab leader character. In this example the tab leader has been set to a period.



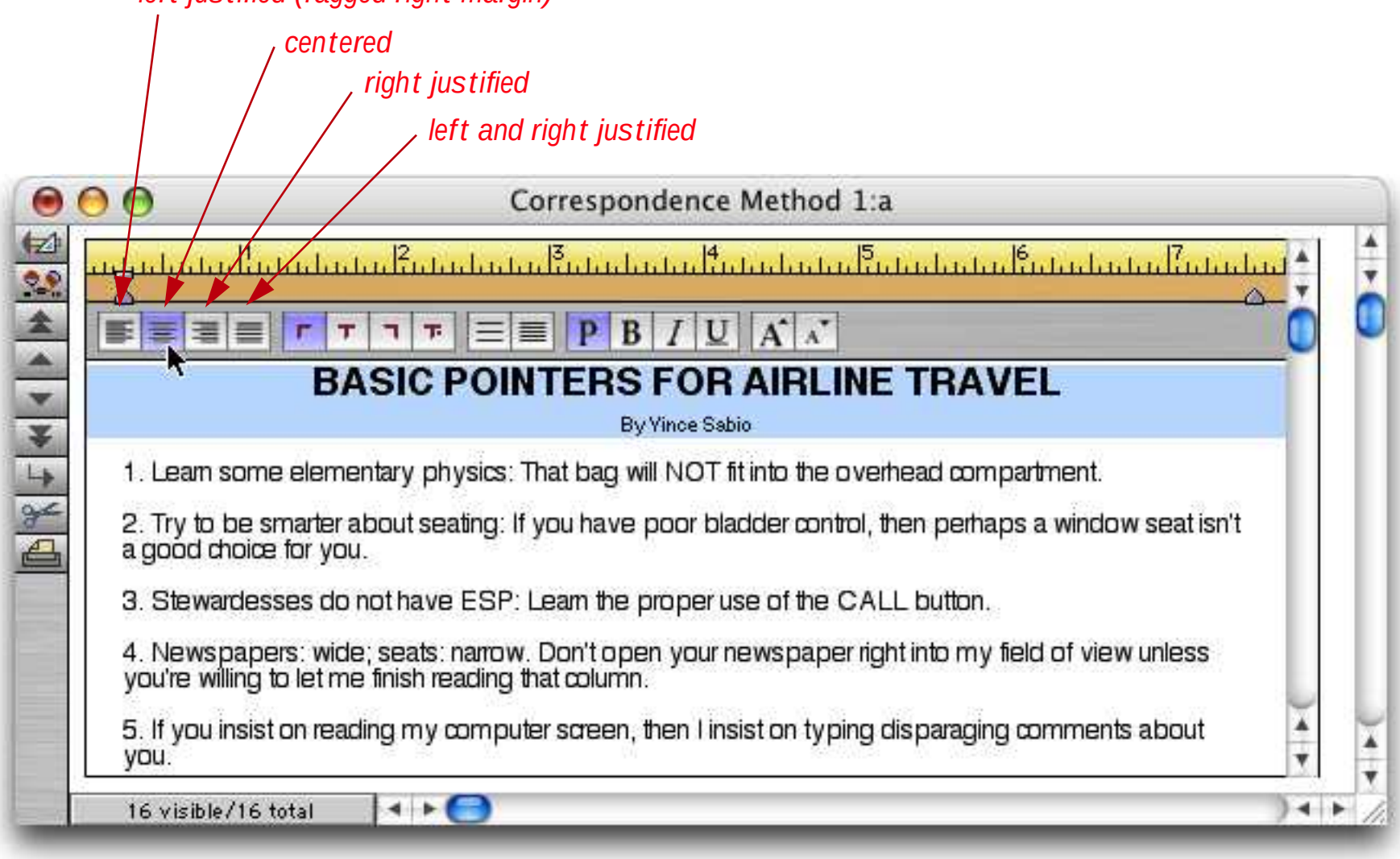
When you adjust a tab stop (either by dragging or with the **Tab Settings** dialog) don't forget that only selected text is affected by the change. Be sure to select the text you want to adjust before changing the tab.



The most common mistake is to click on the text instead of selecting it. In that case, only the line clicked on will change. Be sure you have selected all of the text containing the tab stops you want to modify.

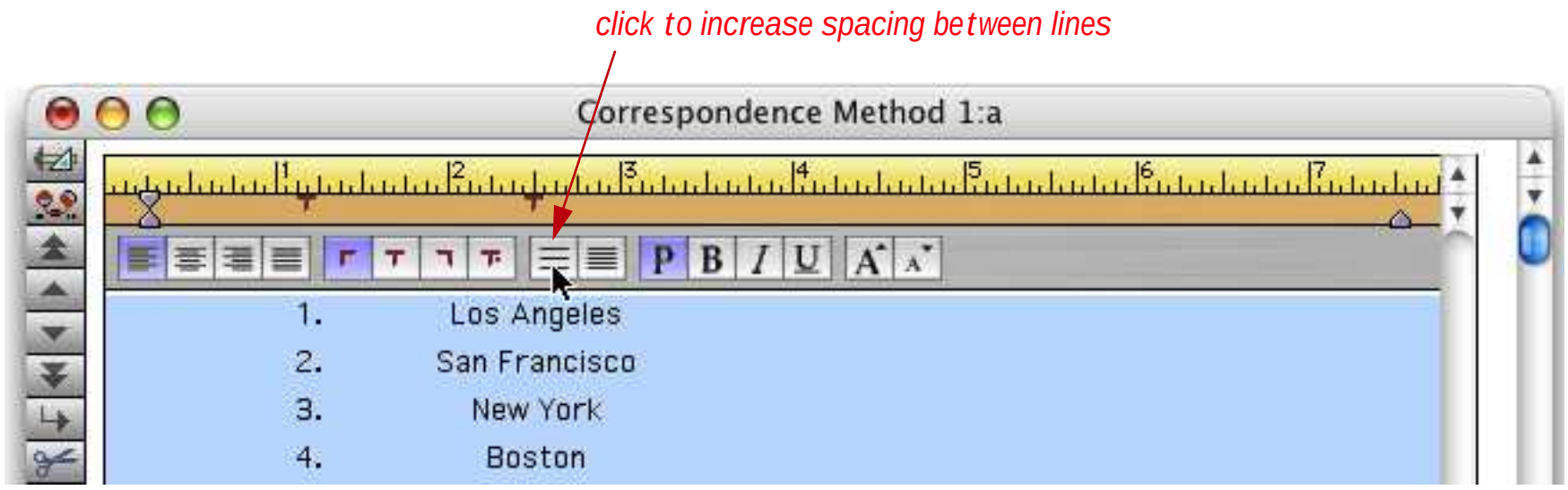
Alignment

For each paragraph, the alignment can be left flush, centered, right flush, or full (left and right flush). The four buttons on the left side of the ruler control the horizontal alignment of the text. For example to center one or more lines first select the lines, then click on the **center** button.

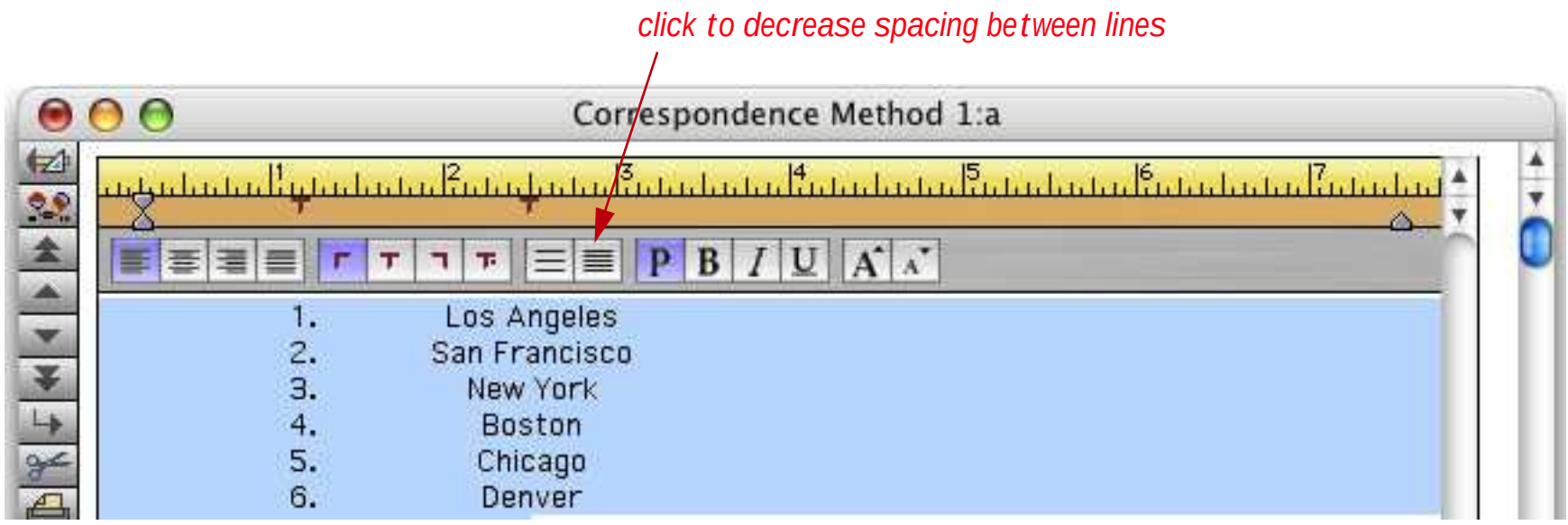


Line Spacing

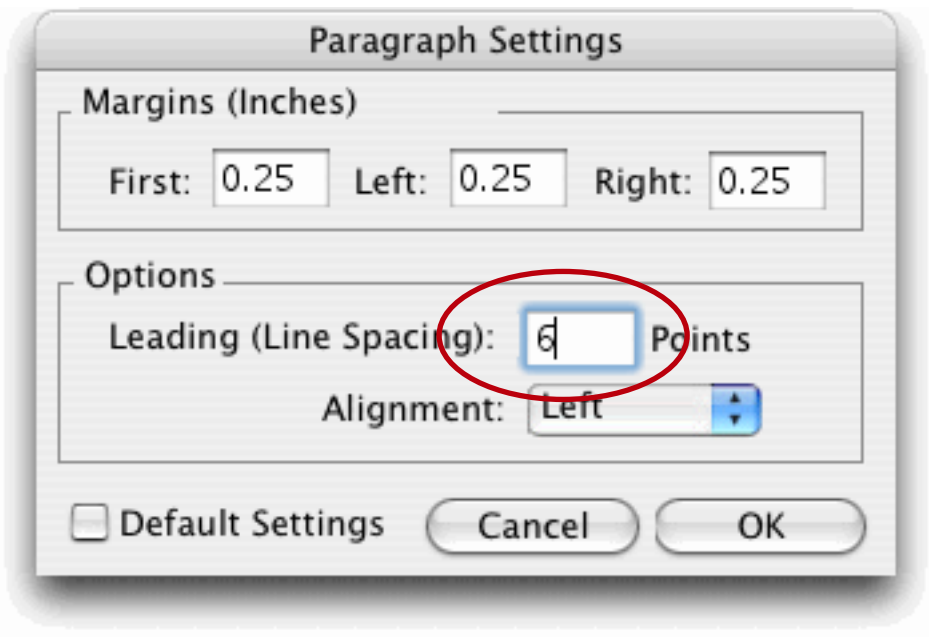
The spacing between each pair of lines is called the **leading**. (This is an old word going back to movable type days, when an actual piece of lead was inserted between each pair of lines.) The word processor normally sets the spacing for you automatically. However, you can adjust this spacing for unusual effects (double spaced text, for example). To increase the spacing between lines, press the button on the left.



To decrease the spacing between lines, press the button on the right.



You can also adjust the spacing with the Leading setting in the **Paragraph Settings** dialog.

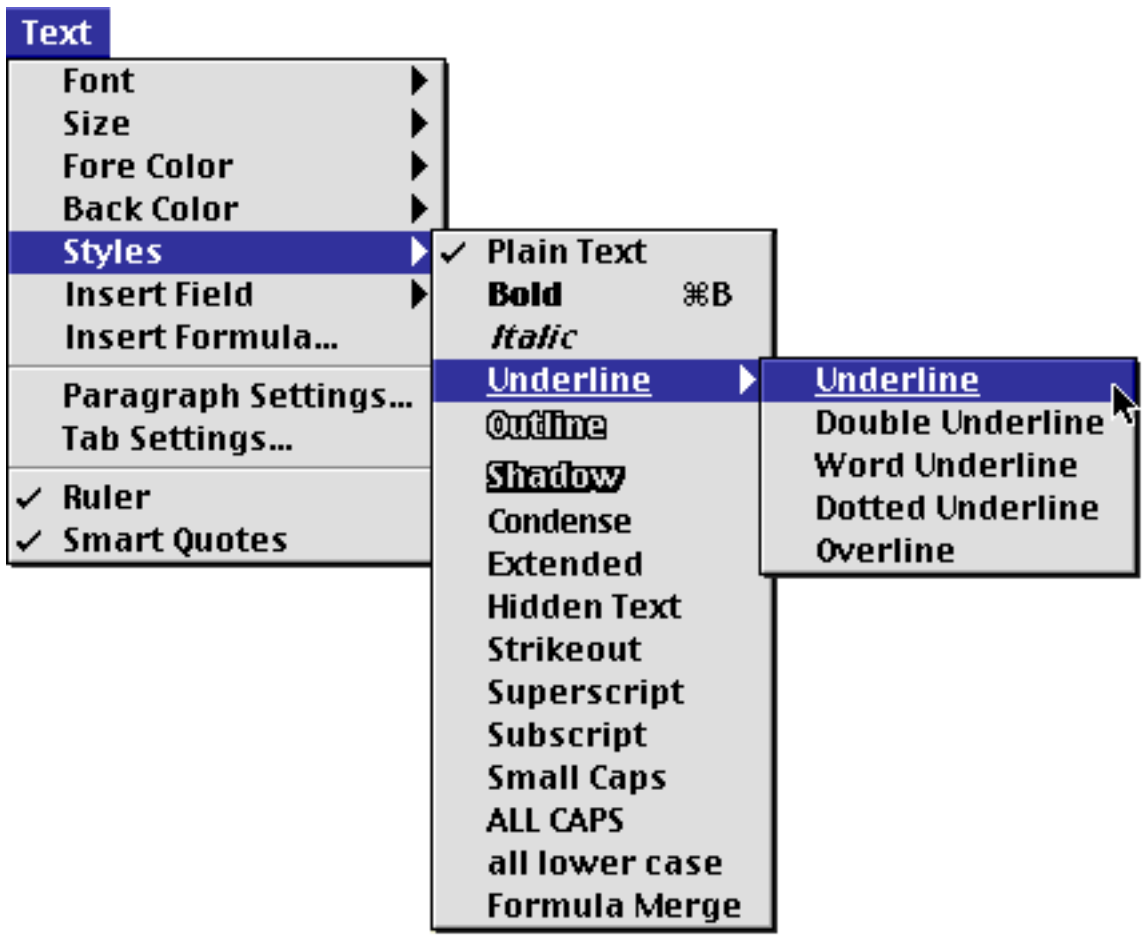


This setting specifies the amount of extra spacing desired between each line. For example, to add 6 extra points between each line, set the leading value to 6. (This value must be 0 or greater.)

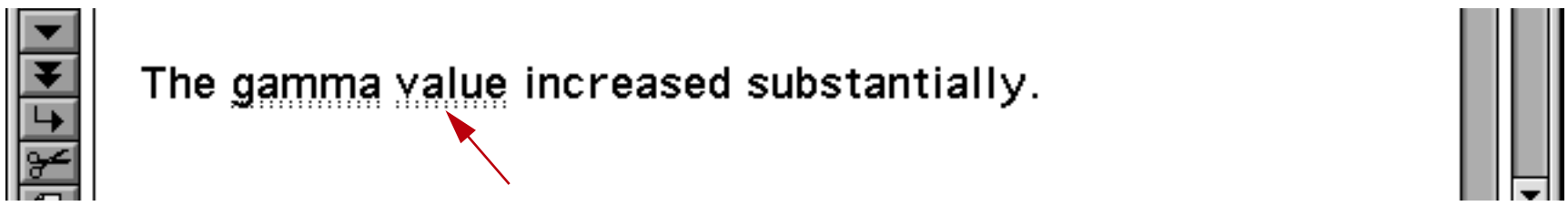
Styles

The Panorama word processor supports twenty different styles of text. Most of these styles are self explanatory, but a few unusual styles require some additional explanation.

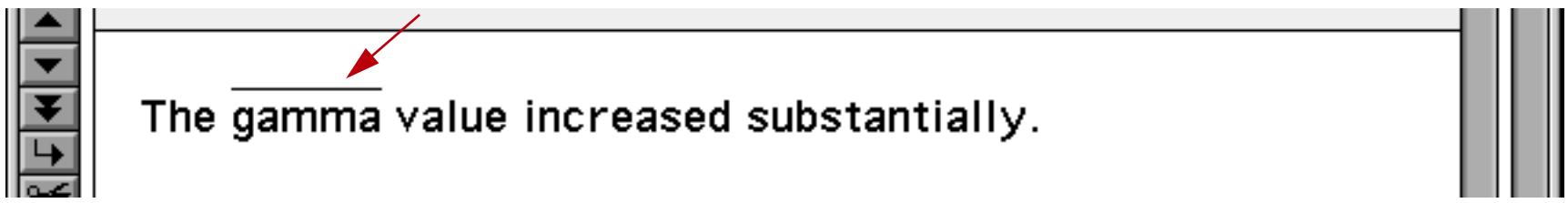
The Underline style can be modified with four different options: **Double Underline**, **Word Underline**, **Dotted Underline**, and **Overline**.



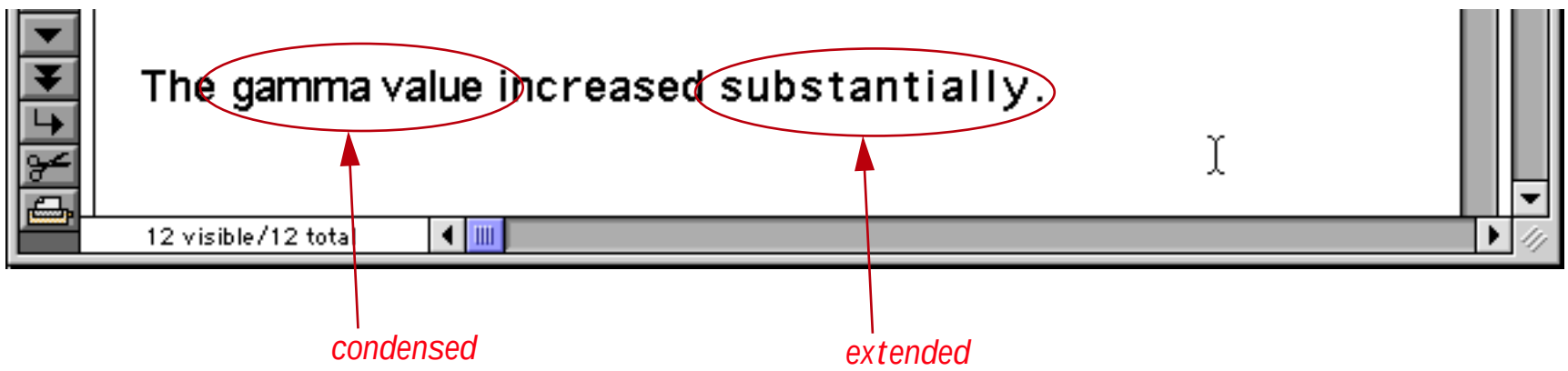
You can combine these options to produce 16 different underline styles, for example **Double Word Dotted Underline**.



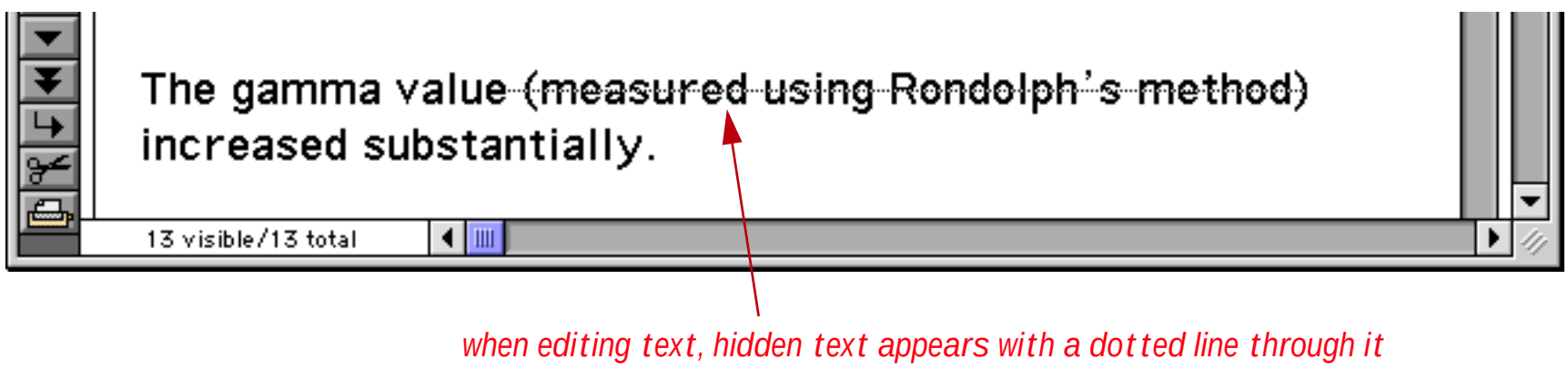
The **Overline** option draws the line above the text, instead of underneath.



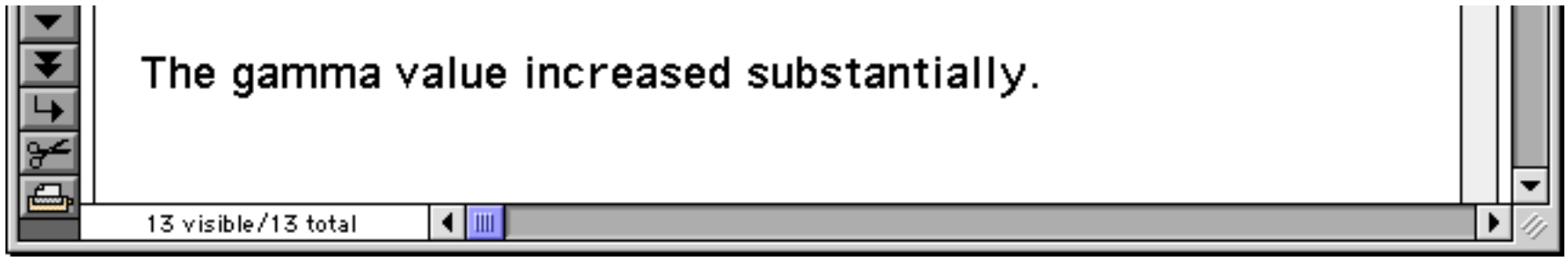
The **Condensed** and **Extended** styles change the letter spacing between the characters. The letters themselves do not change size. The condensed style pushes the letters closer together, while extended spreads them apart.



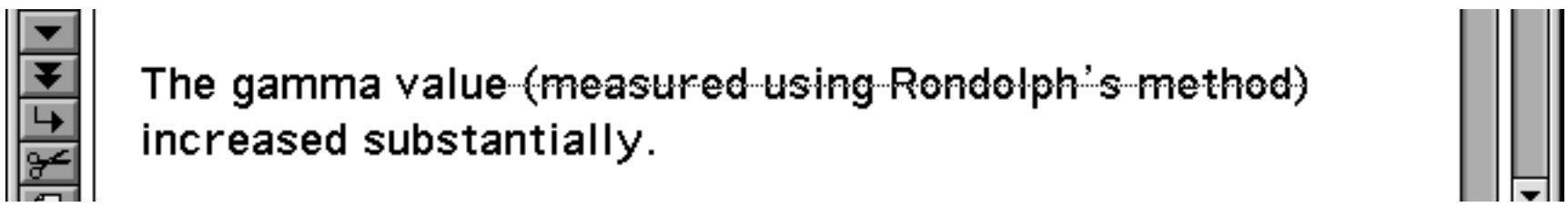
The **Hidden Text** style creates text that only appears when you are actually editing the document.



When the document is not being edited (only displayed or printed) the hidden text disappears.

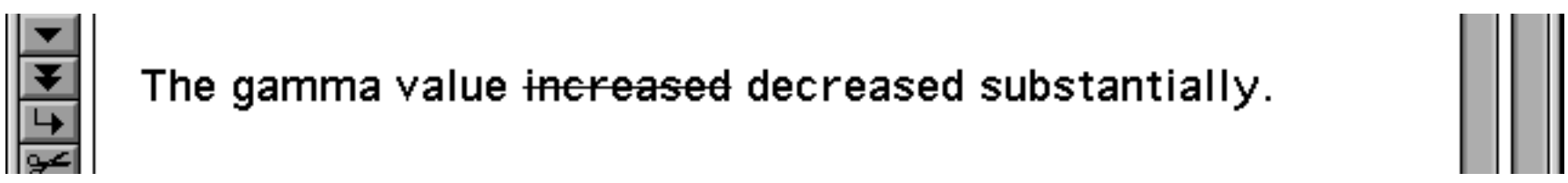


When you re-edit the document (by clicking on it), the hidden text re-appears.

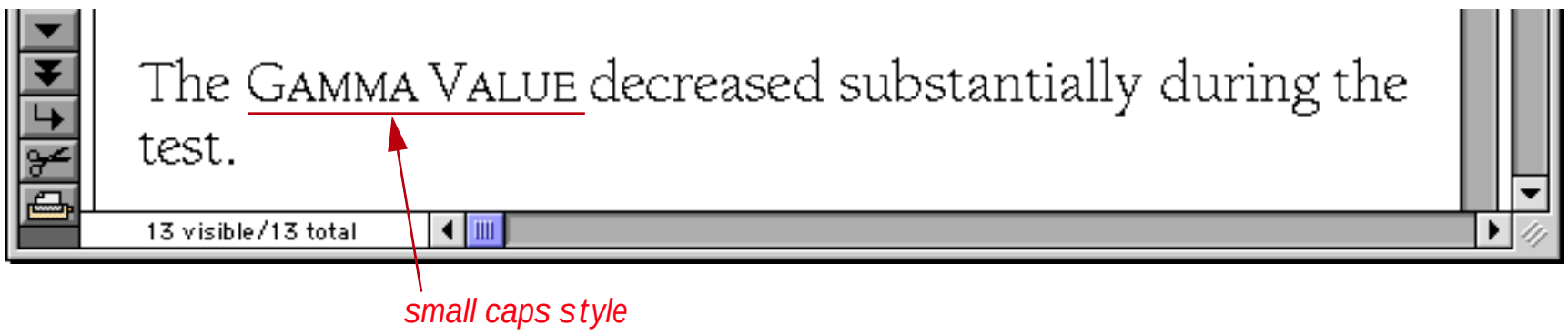


To remind you that this text is hidden, Panorama draws a dotted line through the text. **Tip:** If you have a line of hidden text and you want the entire line to disappear without leaving a blank line, make sure that the carriage return at the end of the line is part of the selection before selecting the **Hidden Text** style.

The **Strikeout** style draws a line through the middle of the text.



The **Small Caps** style draws lower case letters as miniature upper case letters.



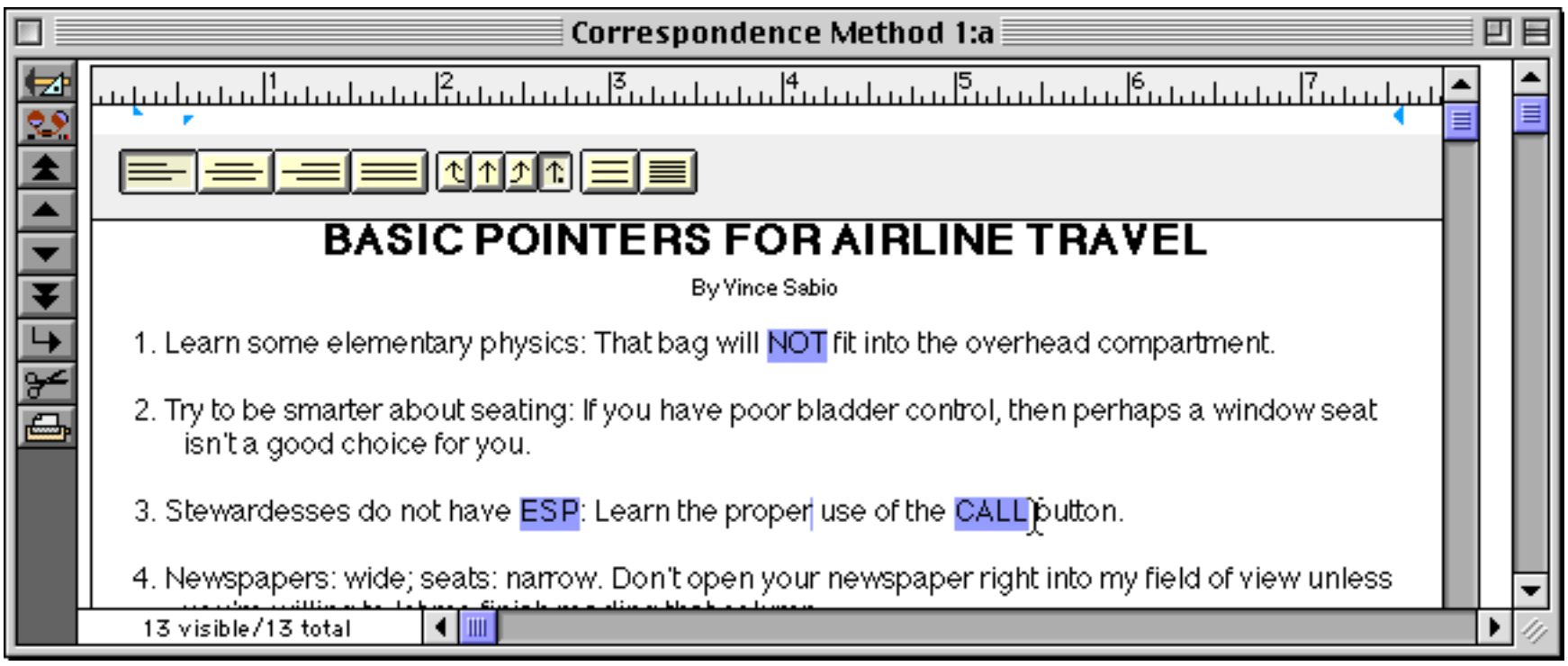
The **All Caps** style draws all text in UPPER CASE even if it was entered in lower case. The text is not actually modified, so you can cancel this style later to display the text normally. The All Lower Case style draws all text in lower case even if it was entered in UPPER CASE. Like the all caps style, the text is not actually modified, so you can cancel this style later to display the text normally.

See "Merging Data into Word Processing Documents" on page 707 for a description of **Formula Merge**, the final style option.

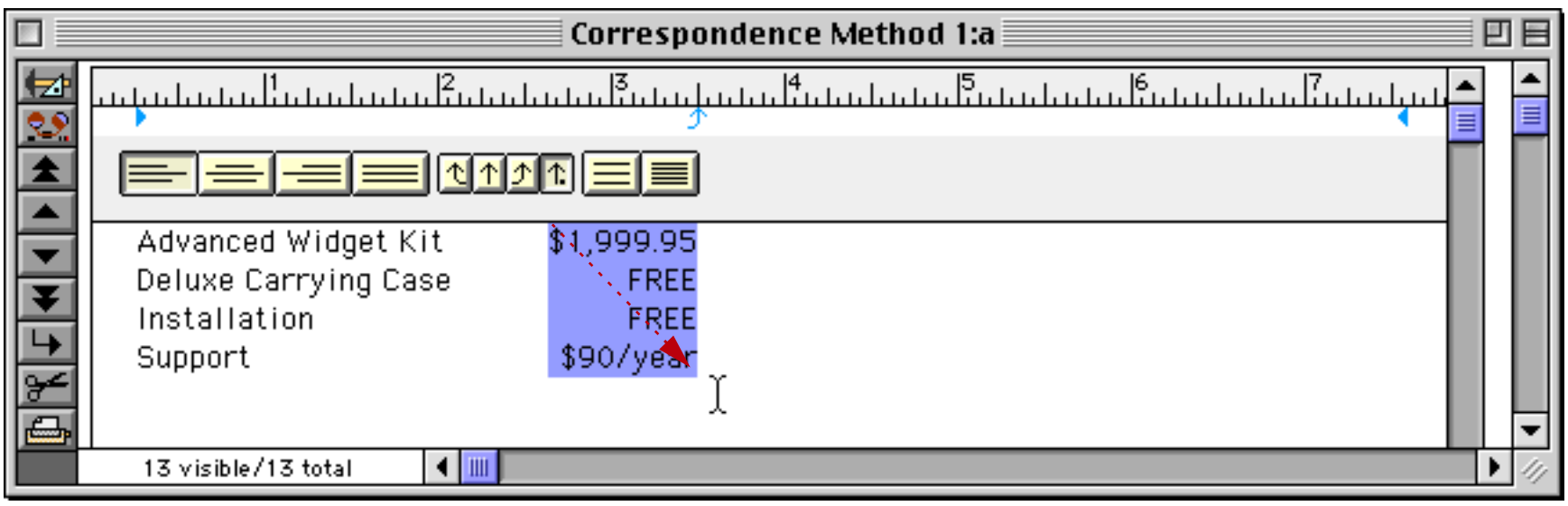
Selecting Text

The word processor allows you to select text by clicking, dragging, double clicking (to select a word), and triple clicking (to select a line). You can also select text by holding down the **Shift** key and using the arrow keys (hold down the **Shift** and **Option**(Mac)/**Alt**(PC) keys to select a word at a time).

To select non-contiguous selections, hold down the **Shift** key and the **Command** key (Mac) or **Control** key (PC) while you drag each selection (or double or triple click). Once you have multiple non-contiguous selections, you can change the appearance of all of the selections at once with the font, size, style, and color submenus.



To select a rectangular area on the Macintosh, hold down the **Control** key while you drag over the area you want to select. On the PC you can do this by holding down the right mouse button as you drag.



This option is especially handy for selecting one or more columns from a table. Once the columns are selected, you can change the appearance with the font, size, style, and color sub-menus.

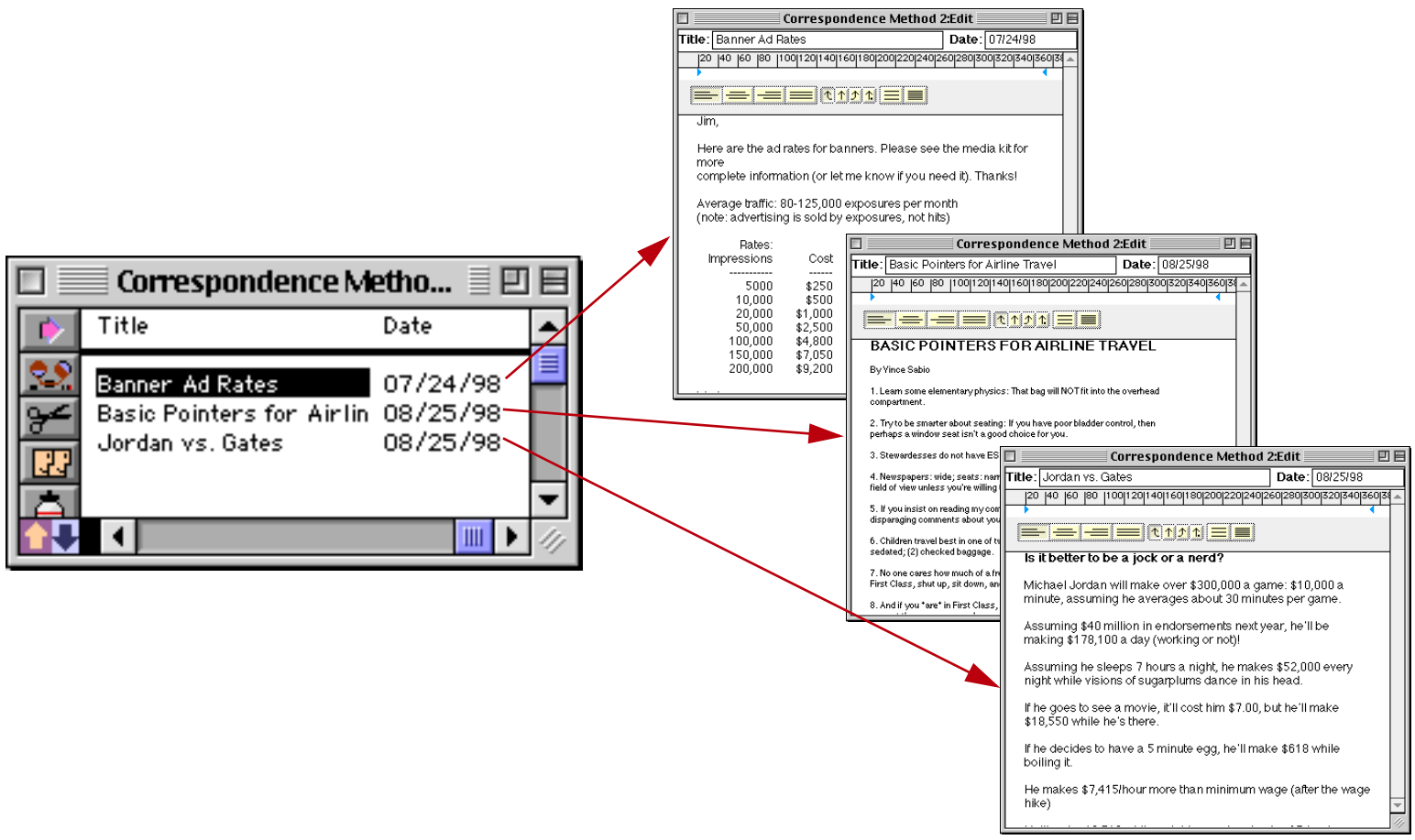
Configuring the Word Processor

The word processor can serve many uses — to store correspondence, to generate custom mail merge "bulk mail," to create catalogs, etc. Different applications require different setups. The following sections explain the different configuration options that are available.

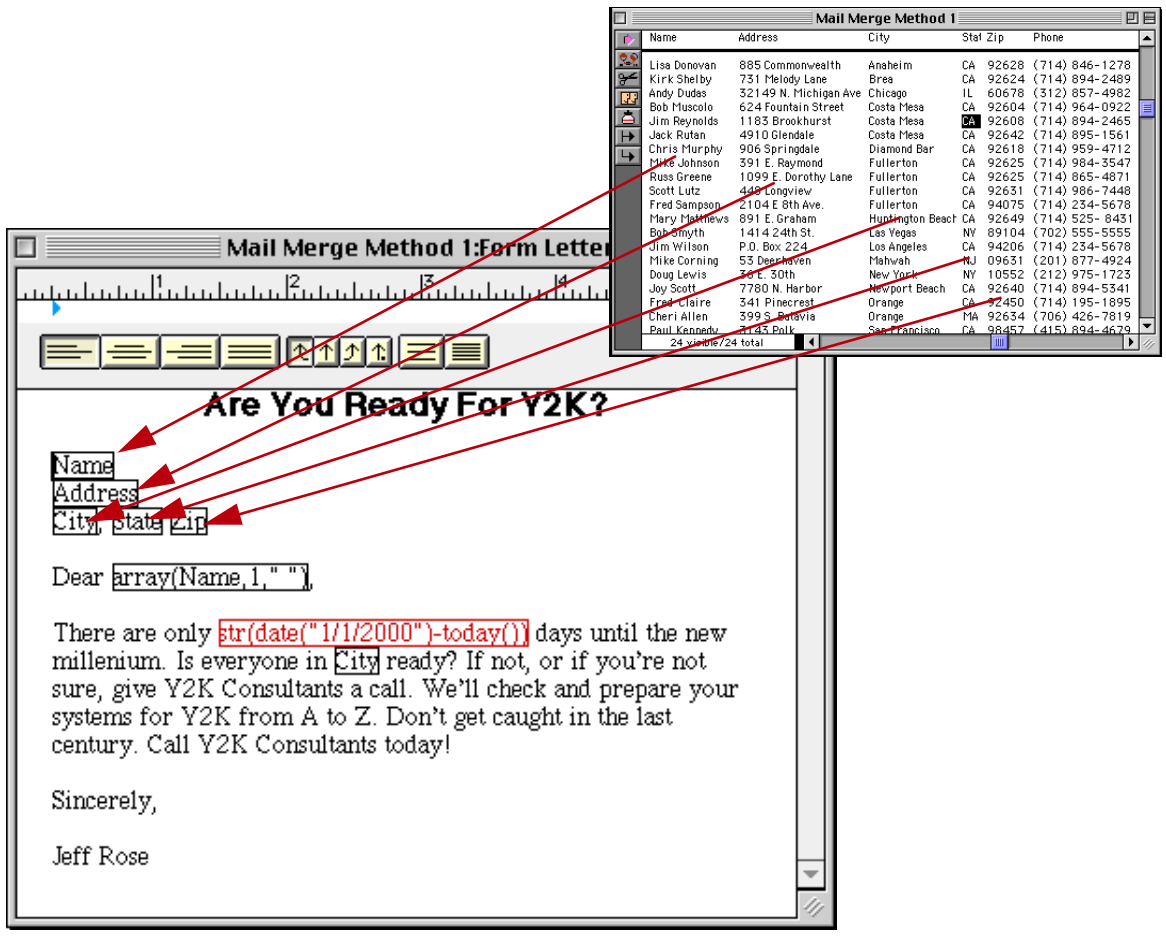
Word Processor Document Storage Strategies

Traditional word processors (Microsoft Word, WordPerfect, Nisus, etc.) store each word processing document in a separate self contained disk file. In Panorama word processing documents are not self-contained but part of a database. Panorama gives you flexibility in choosing how the document is related to the database.

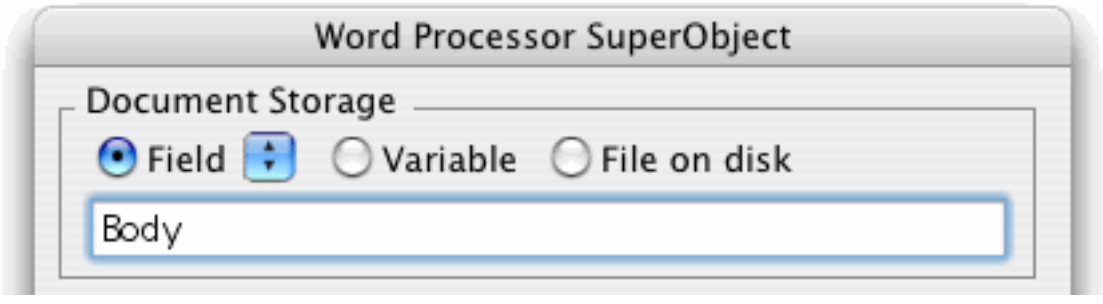
In Panorama the word processor can be used in two basic ways. The first way is to use it to create and organize a collection of documents. For example you might have a correspondence database that keeps all your letters and memos organized. When the word processor is used this way you will usually have one document for every record in your database. In addition to the documents themselves, the database might contain additional information like the date the letter was written, the subject, who it was written to, etc. Here is an example of a correspondence database that contains three records and three separate documents (pieces of correspondence.



The second way to use the word processor is as a template. When used this way there is a single document for the entire database. Panorama will combine this “template” document with information from the database to produce the final documents. For example, you can use this technique to send customized “form” letters to a group of people in your database.

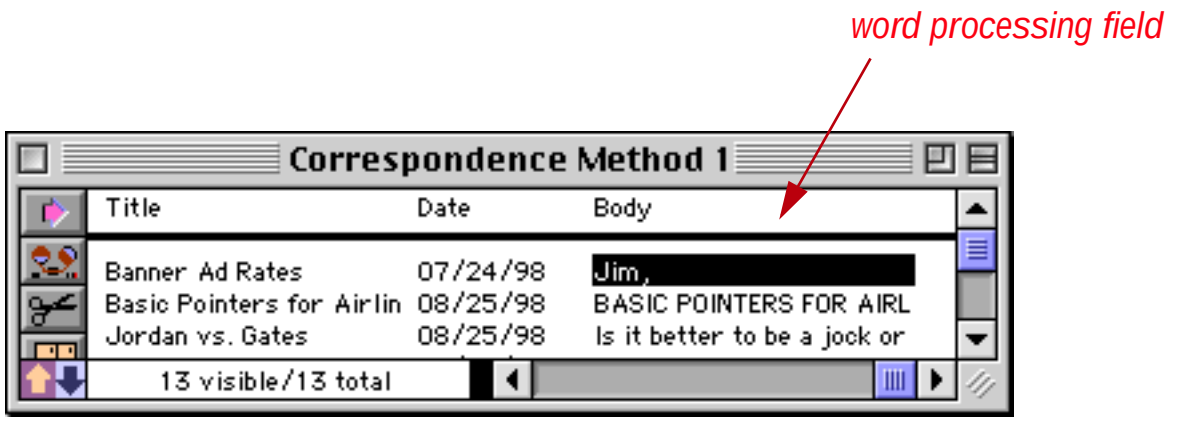


The Panorama word processor allows documents to be stored in three different places: 1) in a database field, 2) in a variable, or 3) in a disk file (see illustration below). Which one of these you choose will depend on several factors, including whether you need a collection of documents or a template, the number of documents involved, the amount of memory on your machine and whether or not the documents need to be searchable.



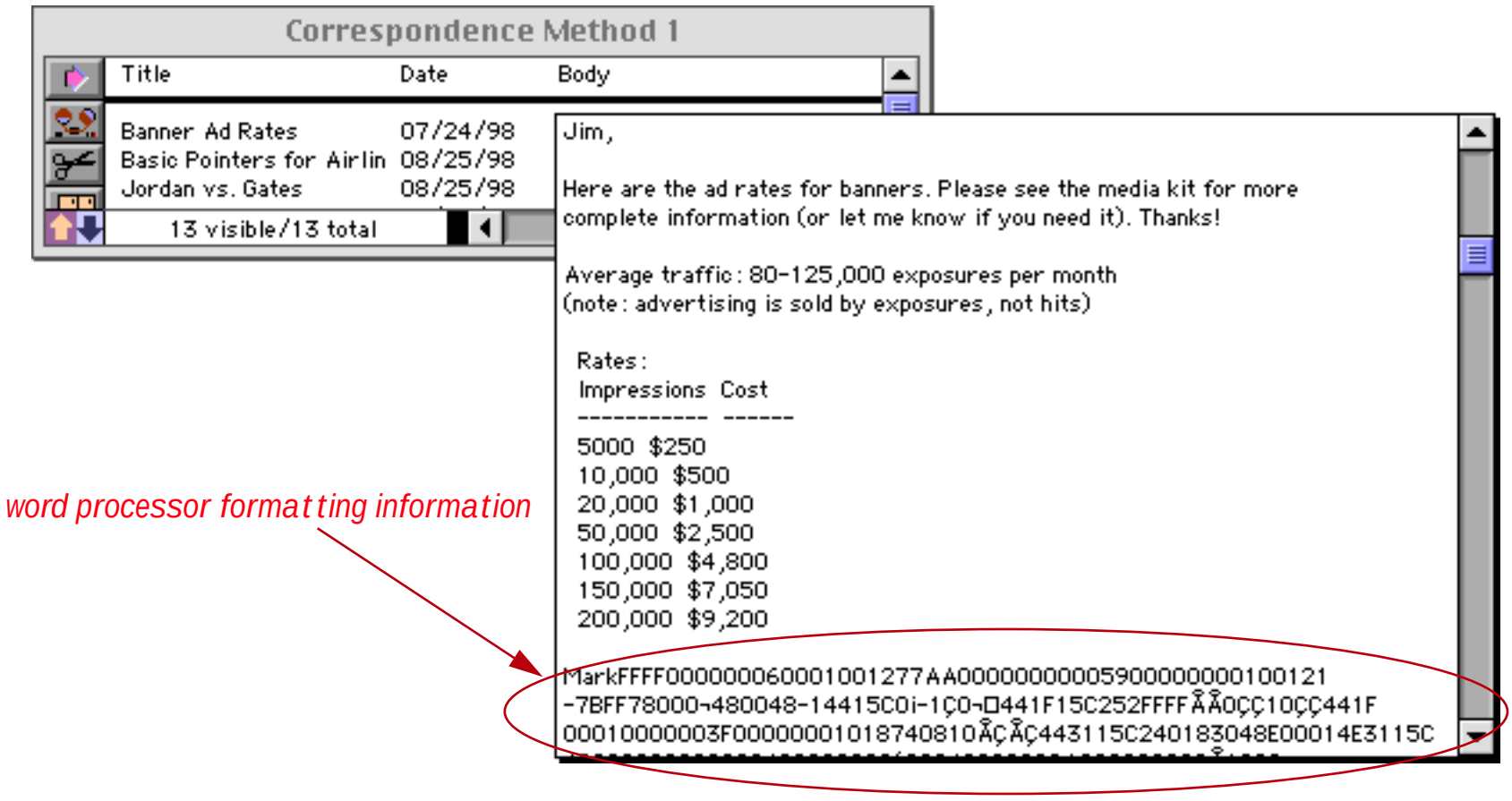
Storing a Collection of Documents

If you need to store a collection of documents the simplest approach is to store the documents in a database field. You'll need to set up a text field specifically for the word processor. In the database below a field name Body has been created for this purpose (the actual name isn't important, you can pick any name you like).

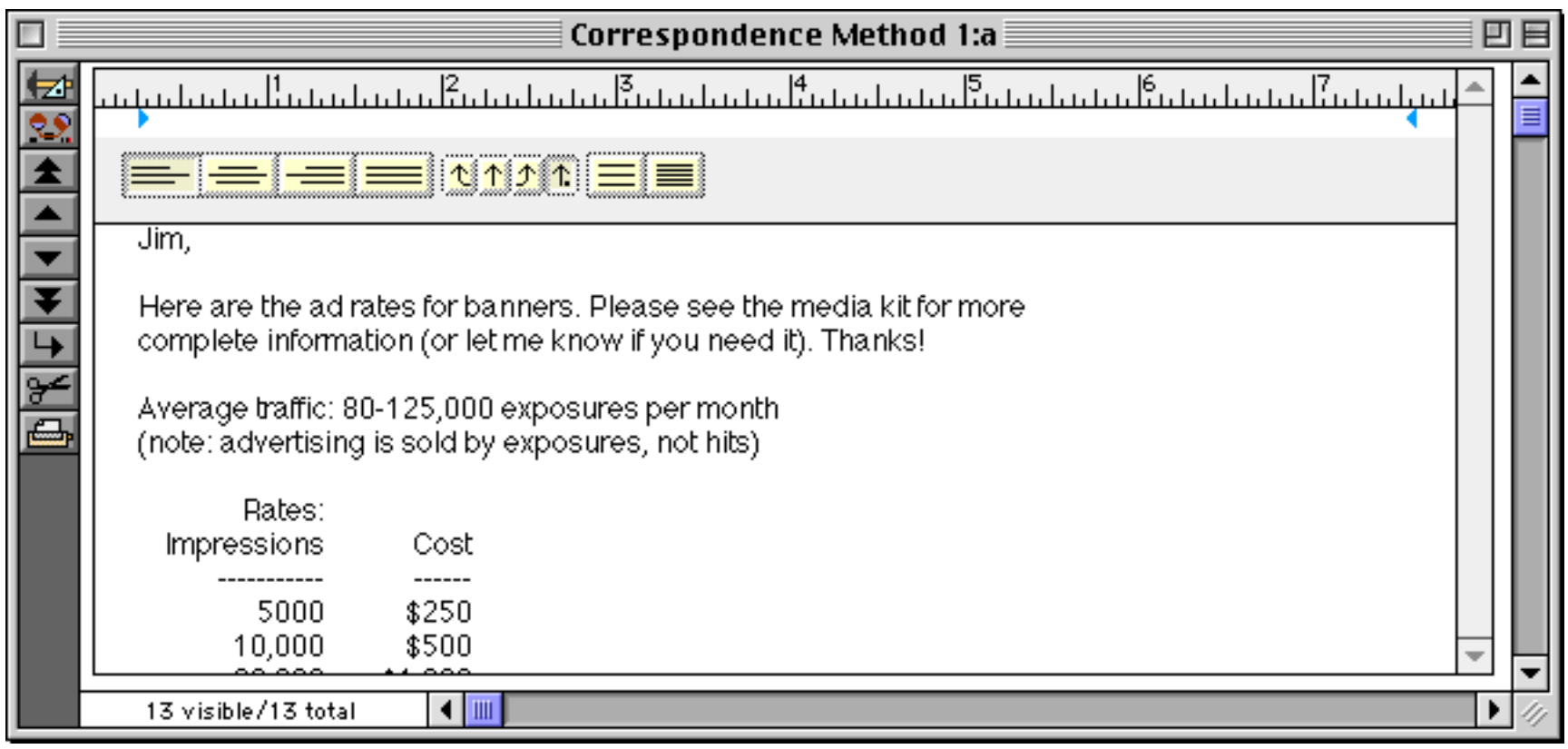


As you can see, the field you set up for the word processor is visible in the data sheet, and you can even double click on the field in the data sheet to edit it. However, you should **not** do this. If you scroll to the end of the Input Box you will see the special formatting information used to keep track of the font, style and other special word processing information. If you edit the field in the data sheet you will corrupt this information and the field will no longer format properly when used in the word processor.

Do Not Edit Word Processor Fields in the Data Sheet!

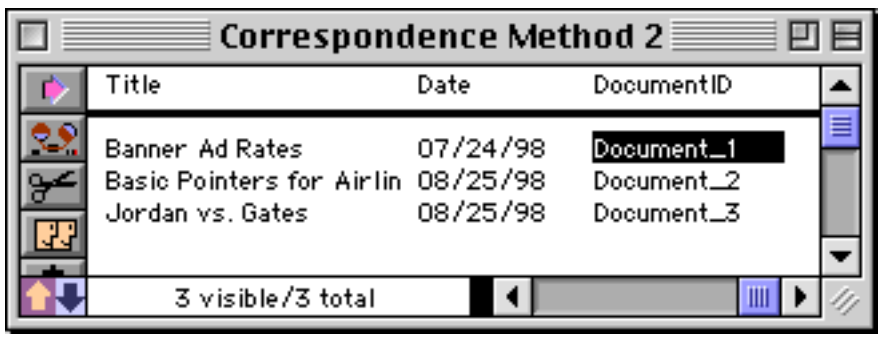


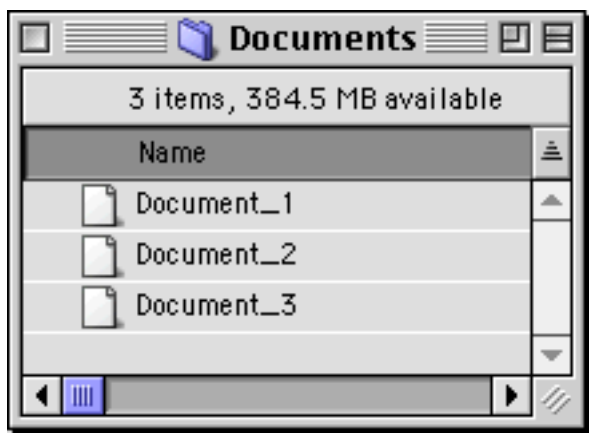
The correct way to display and edit this field is to create a form with a Word Processor SuperObject (see "Creating and Working With Word Processor SuperObjects" on page 673).



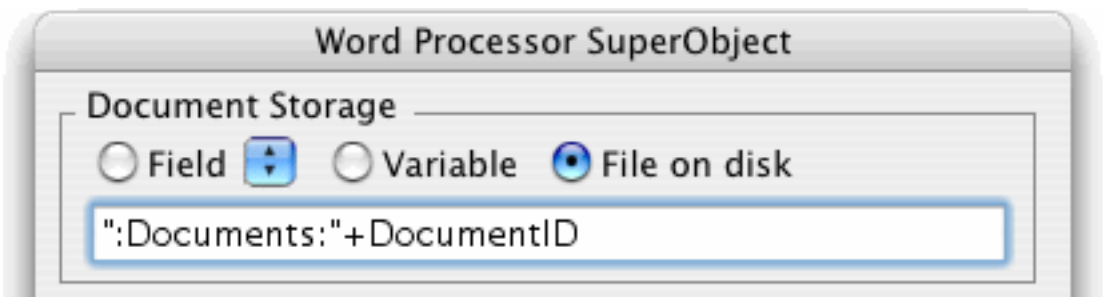
Storing the documents in a field is simple, but if you have a lot of documents (thousands or tens of thousands) these documents can consume a lot of memory. This isn't as much of a problem as it used to be, but if it does become a problem Panorama has an alternative — you can store the documents in separate disk files. If you do this you'll still need a field for the word processor, but instead of holding the entire document this field will only contain the name of the file that actually contains the document.

Here is a typical data sheet for a database with word processing documents stored in external files.

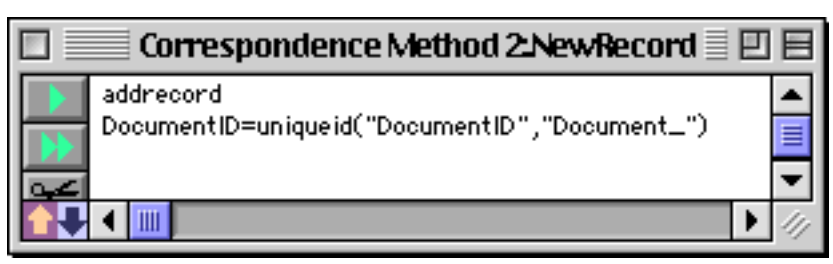




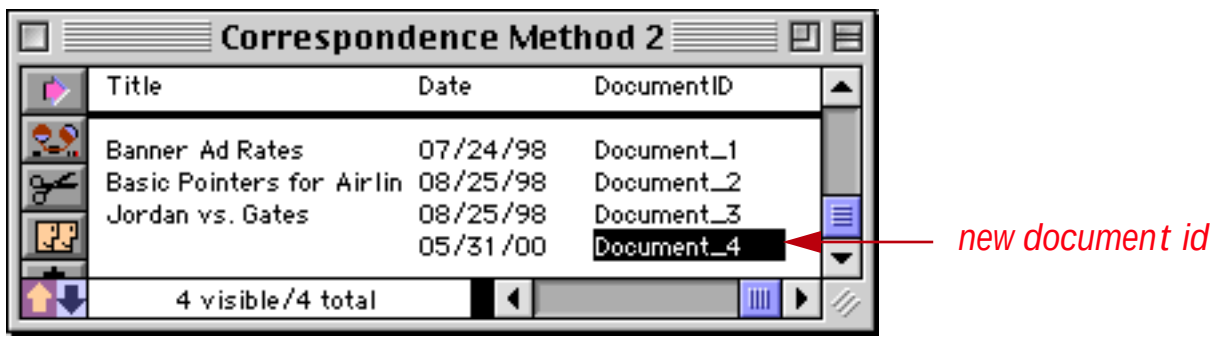
The DocumentID field contains a link to the external document. You must set up a formula in the configuration dialog that creates the actual name of the external file from the value in this field. In this example we have set up a formula that tells Panorama that the document is stored in the Documents subfolder (a folder within the folder containing the database). (On PC systems `.pwp` will automatically added to the end of each file name if no other extension is supplied, for example `Document_1.pwp`.)



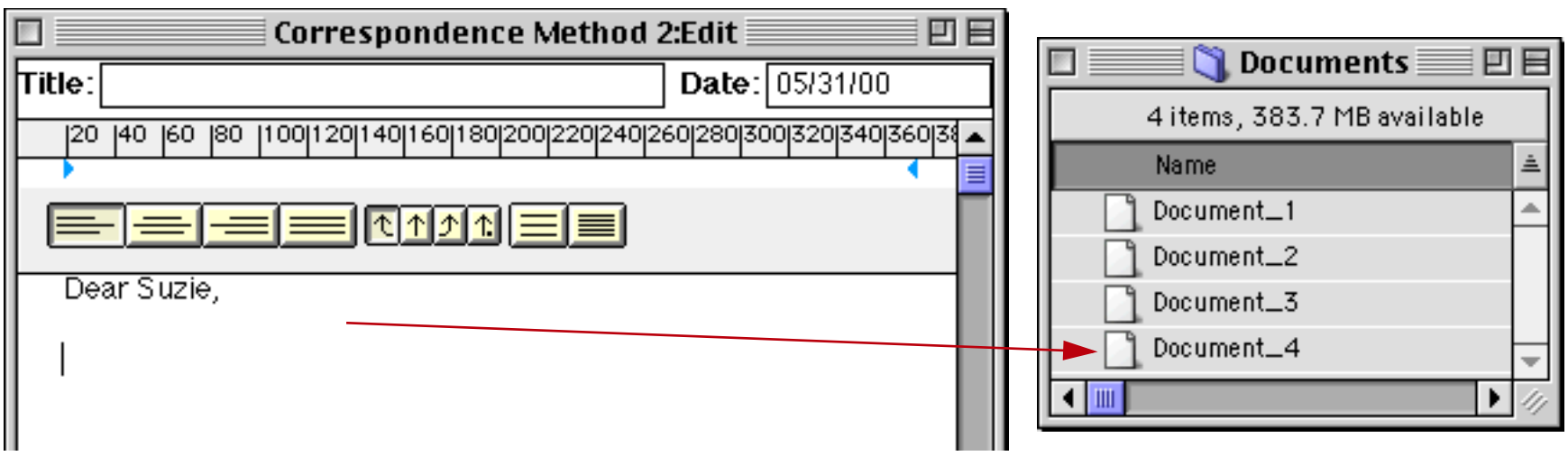
The DocumentID field must contain a unique value for each record, and that value must be a legal file name. In some cases you may be able to use data already in your database, for example the customer name. In this particular example file the unique value is created by a procedure each time a new record is created. See ".NewRecord" on page 1513 to learn about the automatic `.NewRecord` procedure, which is triggered whenever a new record is added to a database. The `uniqueid(` function is described in "UNIQUEID(" on page 5870.



When a new record is added to the database, the `.NewRecord` procedure automatically assigns it a new Document ID.



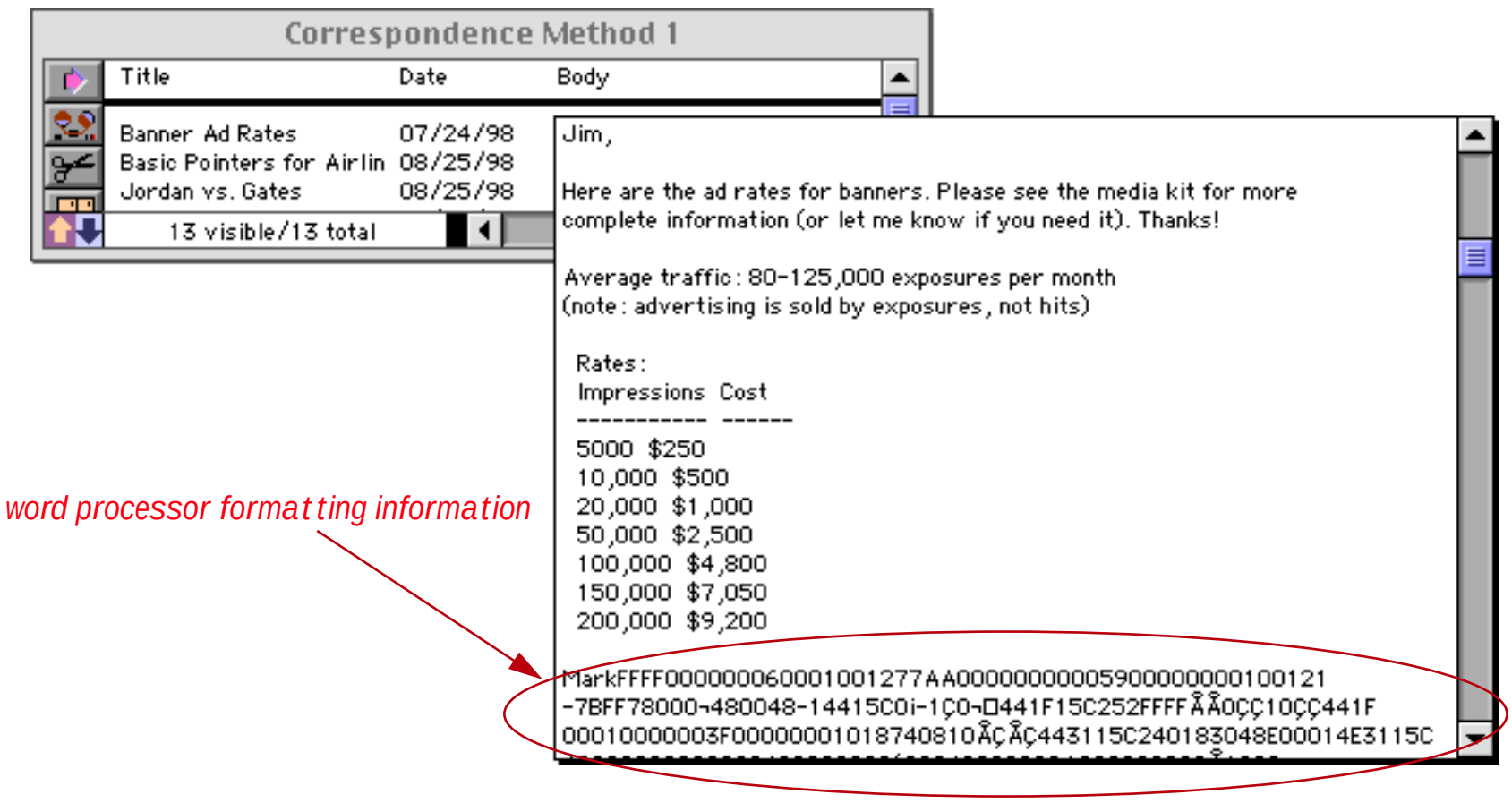
As soon as you start typing into the word processing document Panorama automatically creates the new file on the disk.



Storing the word processing documents allows you to build a large collection of documents without creating a huge memory hogging database. On the other hand, since the documents are not part of the Panorama database, they cannot be searched with the **Find/Select** command (see "The Find / Select Dialog" on page 369).

Searching for Text Within a Collection of Documents

If the documents are stored in a field (see previous section) you can perform normal database operations on them, including searching for text within the documents. However, word processor documents combine text with style information, as you can easily see if you edit a field containing a word processor document with the data sheet (or with an ordinary data cell in a form).



Remember, you should not edit any part of a word processing document with the data sheet or a data cell. Editing a word processing document this way will cause the document to lose its style information. Always use a word processing object to edit word processing documents.

Sometimes you may want to use the text contained in the word processor in a formula. For example, you may want to count the number of letters or words in a document, or search the document to see if it contains a particular word or phrase. To use the text in a formula, use the `documenttext(` function. This function has one parameter, a word processing document. The result is the text of the document with all of the style information removed. For example, suppose your database has a field called Letter that contains word processing documents. You can use the formula shown below with the **Formula Find/Select** command (see "The Find / Select Dialog" on page 369) to select all records with documents containing the word Yosemite.

```
select documenttext(Letter) contains "Yosemite"
```

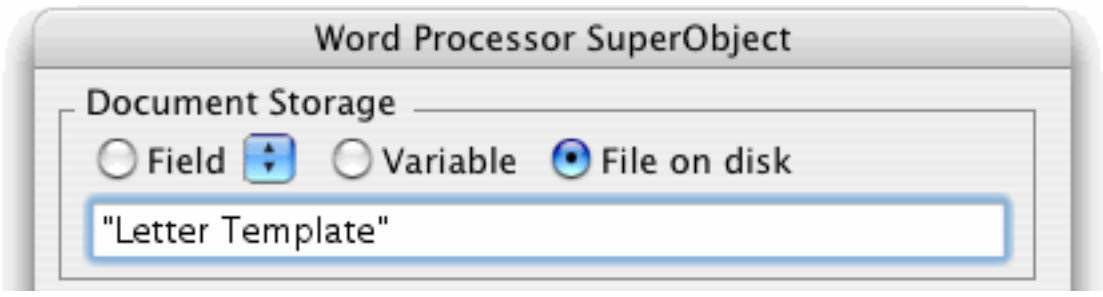
Here is another example that calculates the number of words contained in a letter.

```
message "This letter contains "+str(arraysize(documenttext(Letter)," ")+" words."
```

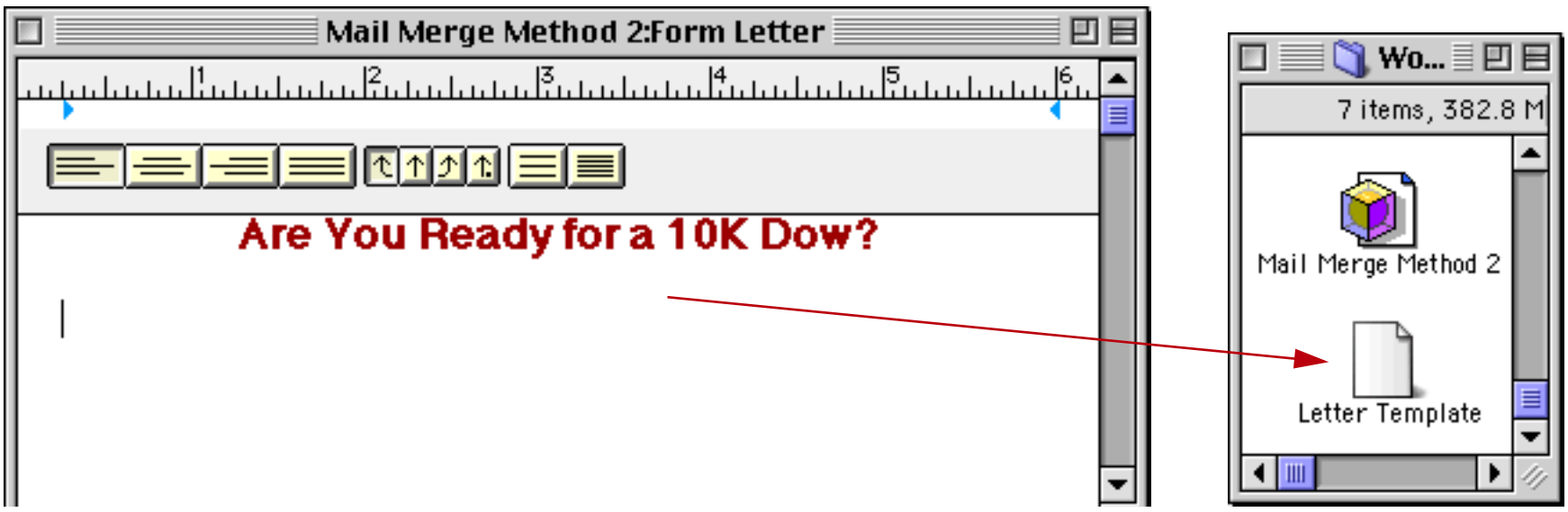
The word processor can also work with standard text. If you give the word processor standard text with no style information, it will use the default font, size, indents, tabs and line spacing until you specify otherwise.

Setting up Storage for a Template Document

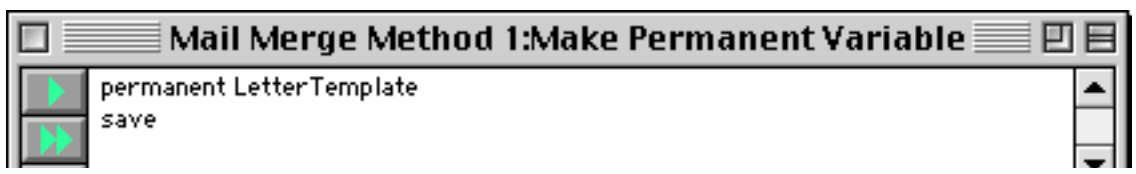
If you need a mail-merge template document the simplest approach is to store the document in a separate disk file.



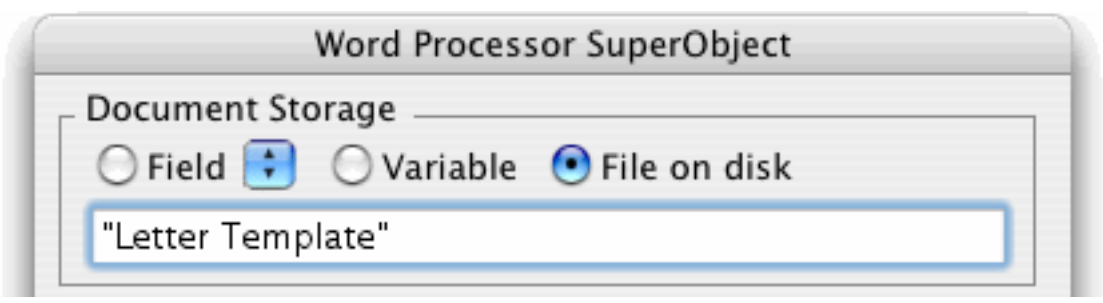
As soon as you begin typing into the word processor Panorama will automatically create a file named Letter Template (or Letter Template.pwp on PC systems). The file will be created in the same folder as the database.



An alternate method is to store the document in a variable. A permanent variable is recommended so that the document will be saved automatically when the database is saved (see "Long Life Variables" on page 1372). Here is a procedure that creates a permanent variable named LetterTemplate (see "Writing a Procedure from Scratch" on page 1352 to learn how to create procedures). You only need to run this procedure once to create the permanent variable (that's why it's called permanent!).



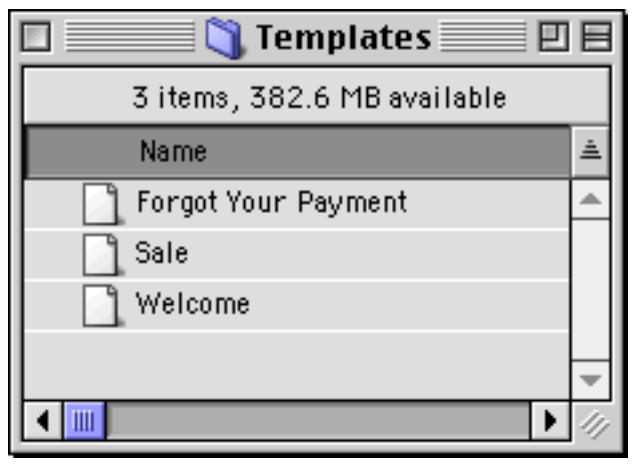
Once the permanent variable has been created you can go ahead and create the Word Processing SuperObject (see "Creating and Working With Word Processor SuperObjects" on page 673). Select the Variable option, and enter the name of the permanent variable you have created.



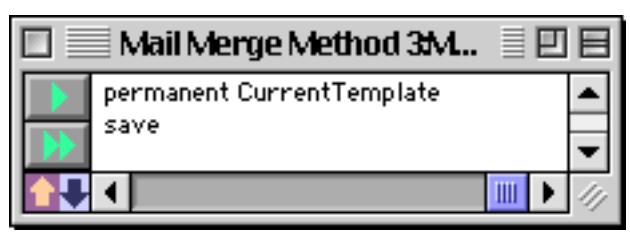
When you press the **OK** button your Word Processor SuperObject is ready to go. The document will be stored in the permanent variable, which is automatically saved as part of the database each time you use the **Save** command.

Setting up Storage for Multiple Template Documents

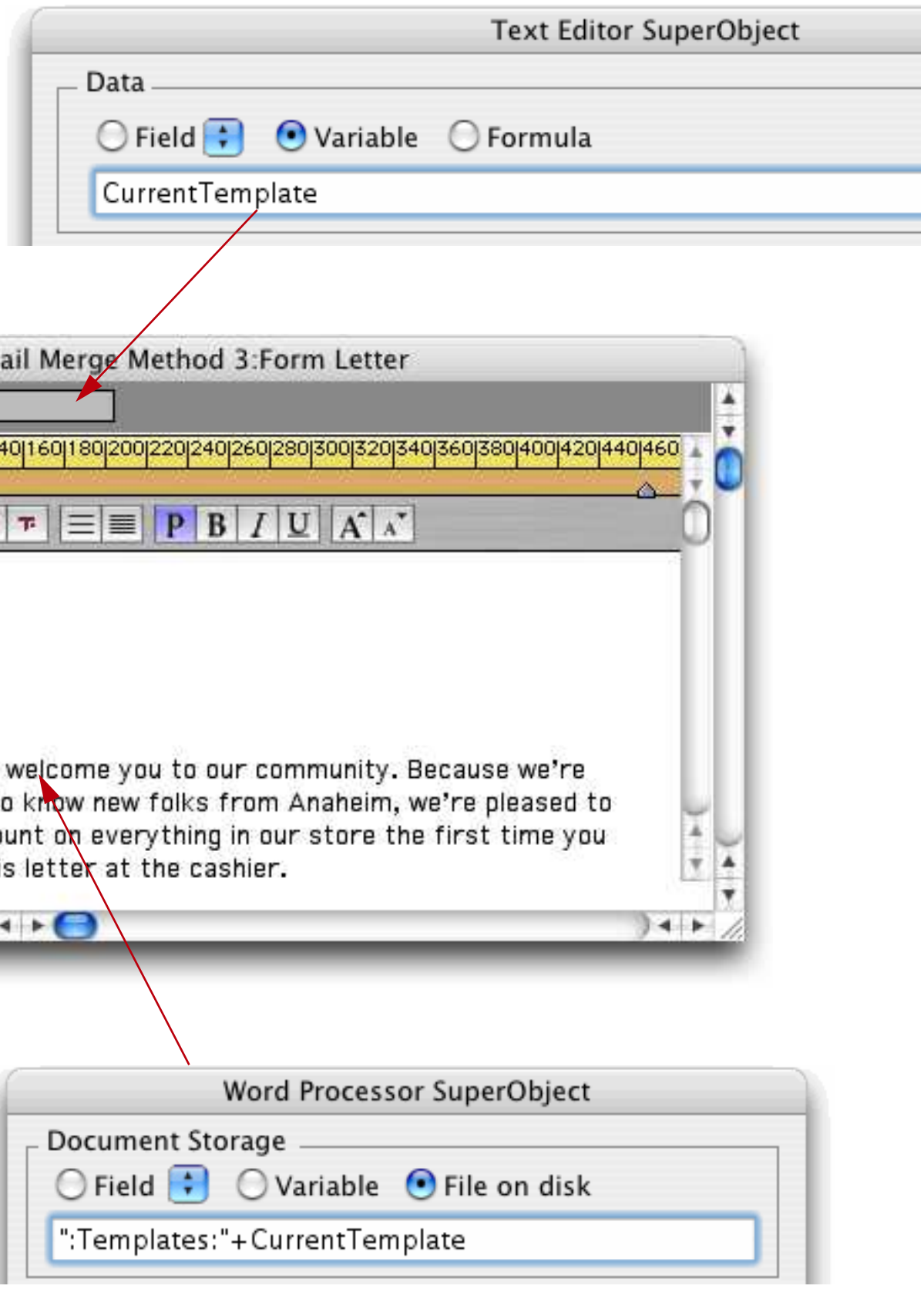
The techniques described in the previous section are fine if you want to use a single mail-merge template. In this section we'll show how to set up multiple mail-merge templates for a single database. For example, you might have different "form" letters that you want to send out at different times — to welcome a new customer, to announce a sale, or to ask about a missed payment. Each one of these templates will be stored in a separate file on the disk, like this.



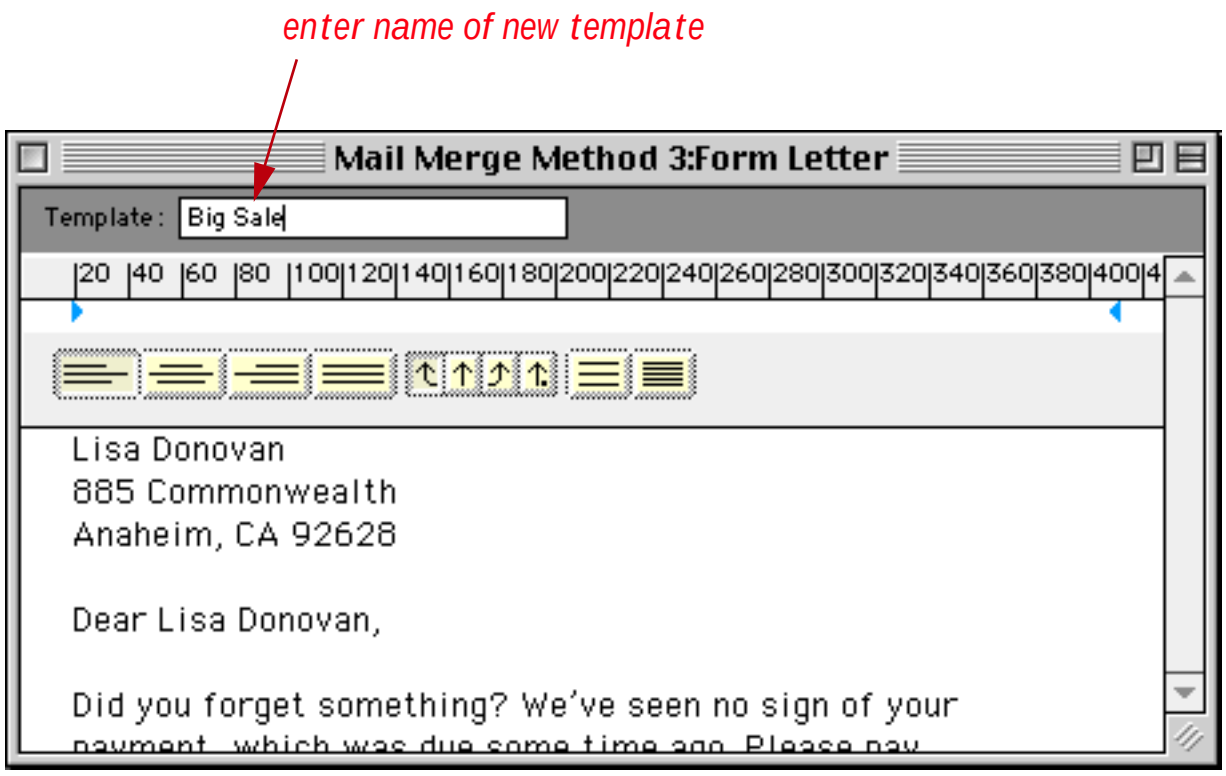
To start with you'll need to create a permanent variable. However, instead of using this variable to store the entire template document we'll only use it to store the name of the current template document. Here is a procedure that can create the permanent variable for us. Be sure to run this procedure at least once before you continue with the next step.



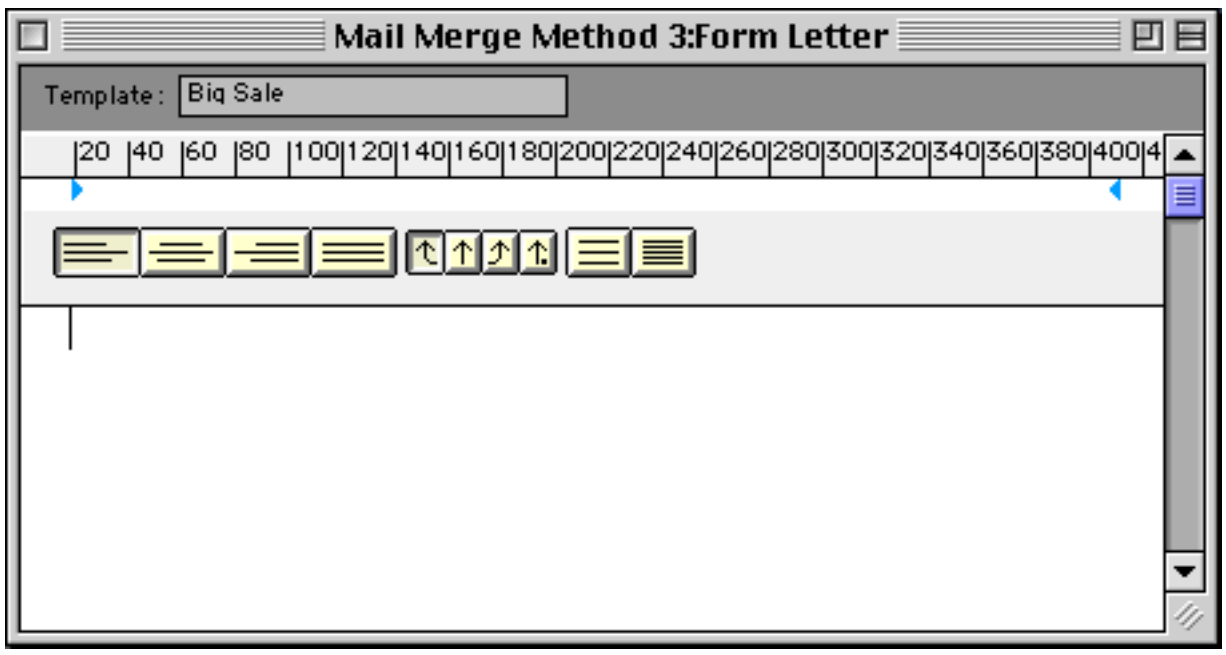
The form for this example requires two editing objects. First, you'll need a Text Editor SuperObject that will allow the name of the current template to be changed (see "Creating and Modifying Text Editor SuperObjects" on page 639). Of course a Word Processing SuperObject is also necessary. The text is stored as a File on disk, with a formula that tells Panorama to look in the folder Templates and to that the permanent variable CurrentTemplate contains the file name. (You'll need to create the Templates folder if it doesn't already exist. It should be in the same folder as the database itself.)



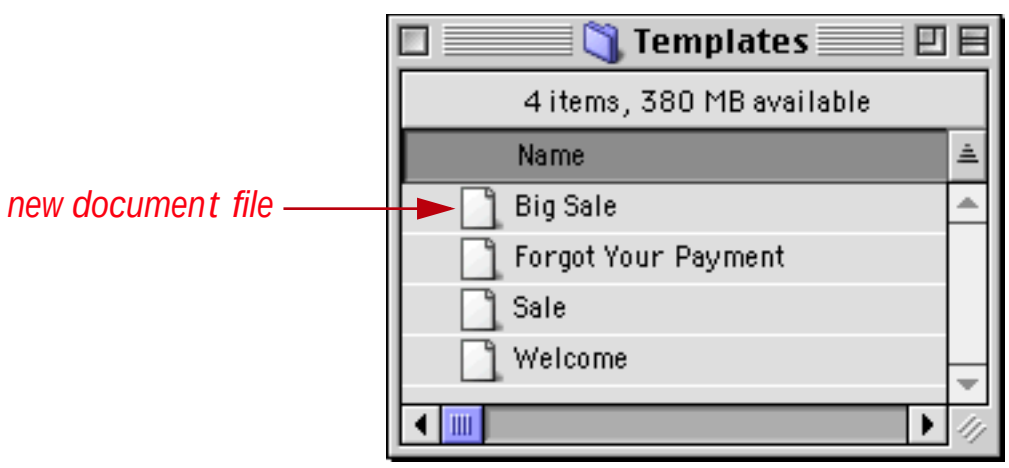
Once these objects are set up you can switch to Data Access Mode and start building templates. Start by entering the name of the new template.



After you have entered the name of the new template, click on the word processor. Panorama will immediately create the new, blank template, which you can start filling in.



If you look on the hard disk you'll see that the new template has been created.



At any time if you want to re-use a previous template, just type the name of that template into the Text Editor SuperObject. As soon as you press the **Enter** key, the previous template is saved and the requested template is loaded in.

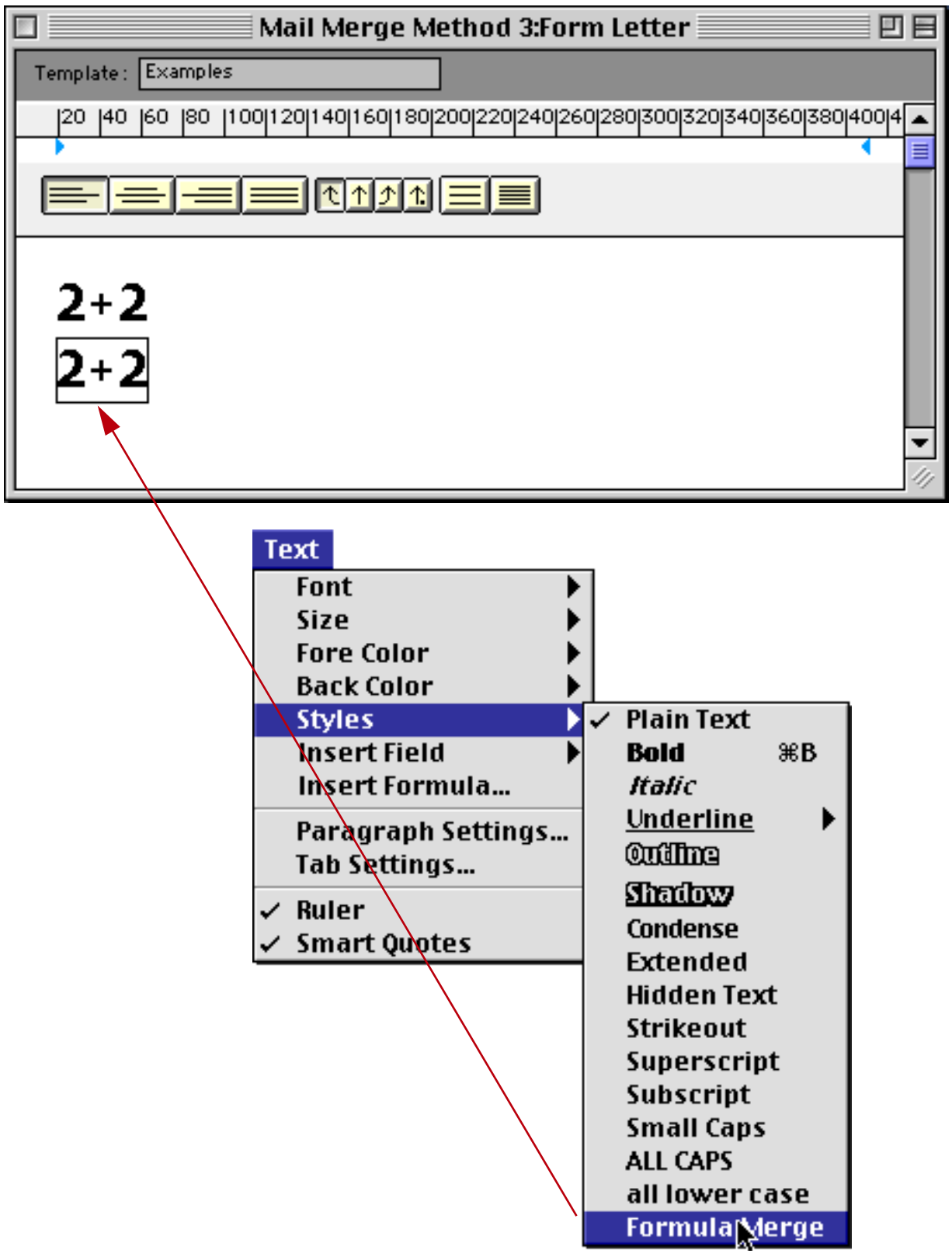


This simple example requires you to type in the name of the template you want to use. A more advanced database could use a pop-up menu (see "Pop-Up Menus" on page 850) or a scrolling list (see "List SuperObjects" on page 869) in combination with the `listfiles(` function (see "Disk Files and Folders" on page 1304) to select the template.

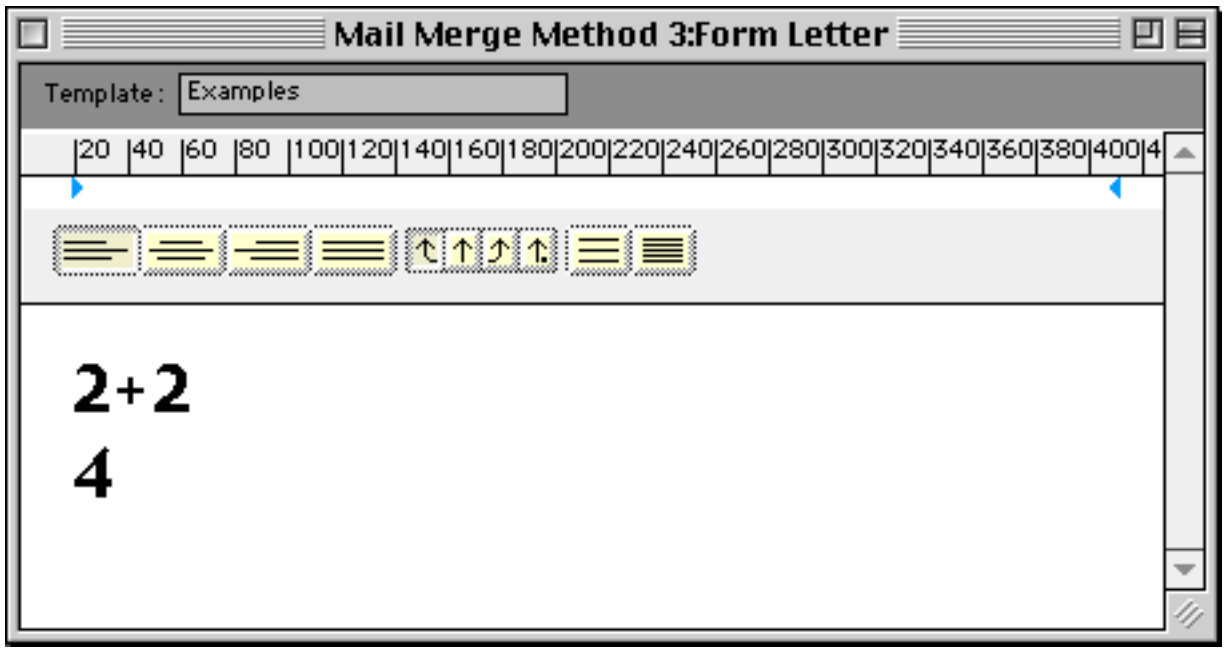
Merging Data into Word Processing Documents

Just as with an auto-wrap text object, Panorama can merge the result of a formula into the middle of a word processor document. The auto-wrap text editor does it with special characters around the formula (see "Displaying Formulas in Auto-Wrap Text" on page 602). The word processor does it with a special style, the **Formula Merge** style.

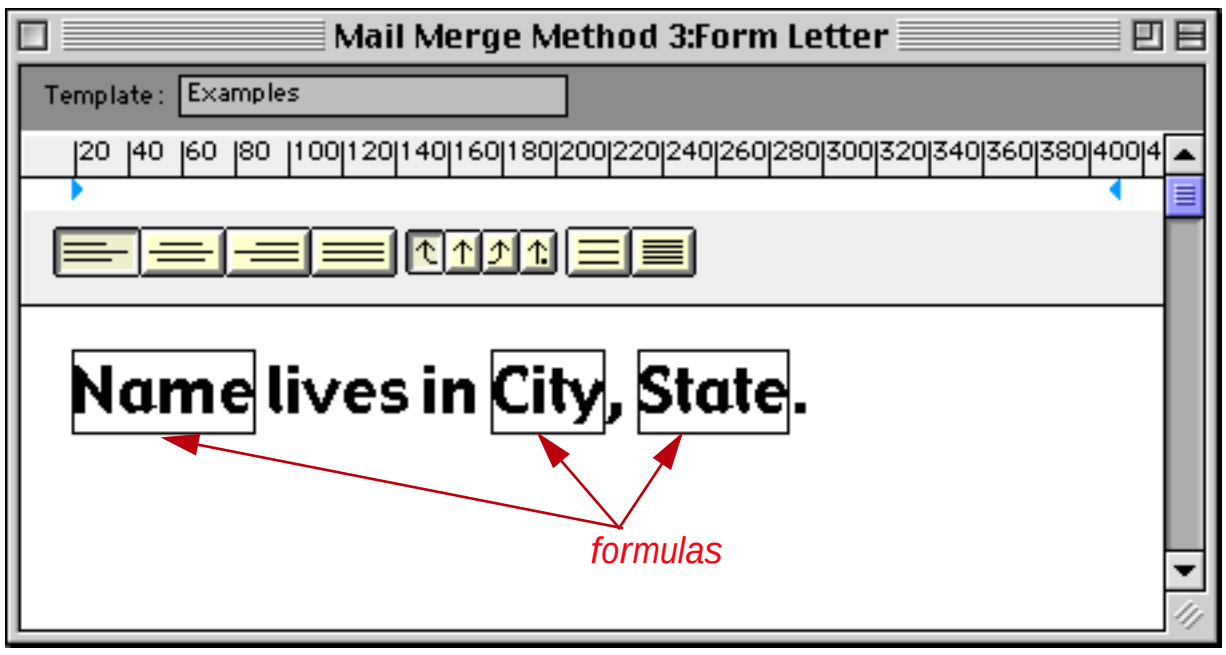
Let's start with a simple example: 2+2. We'll type this formula in twice, the first time using the Plain style. The second time we'll select the text and apply the **Formula Merge** style. Notice the box that appears around the formula.



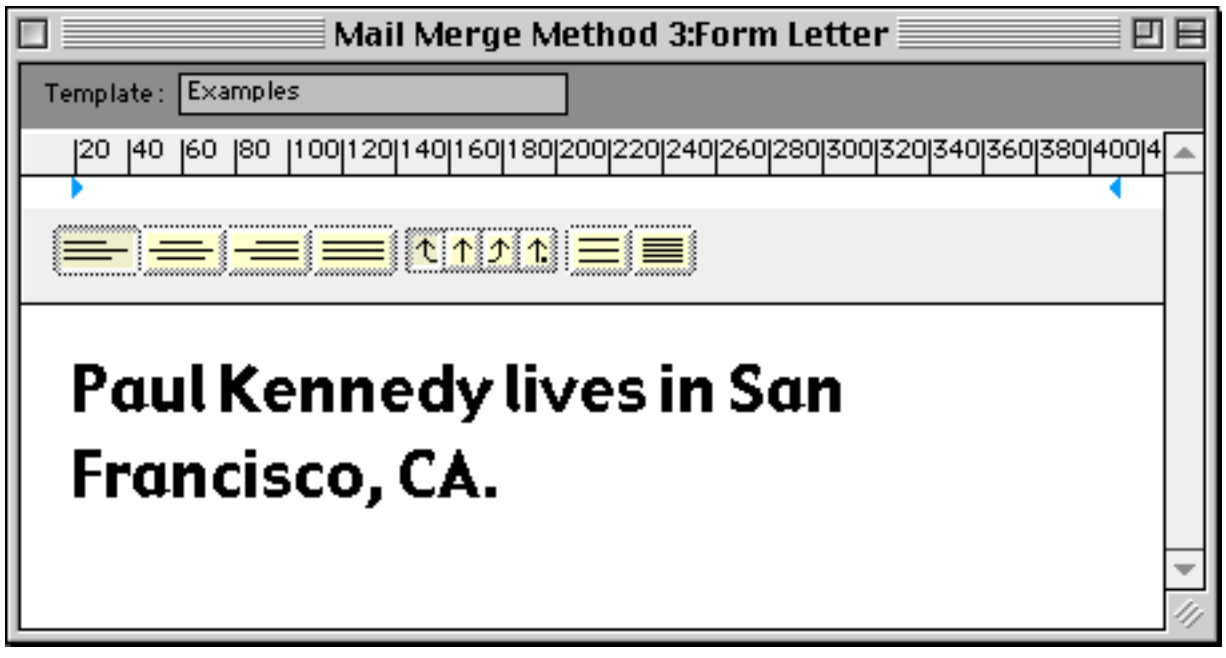
When the **Enter** key is pressed, the display changes. The second formula is now replaced by the result of the formula: 4.



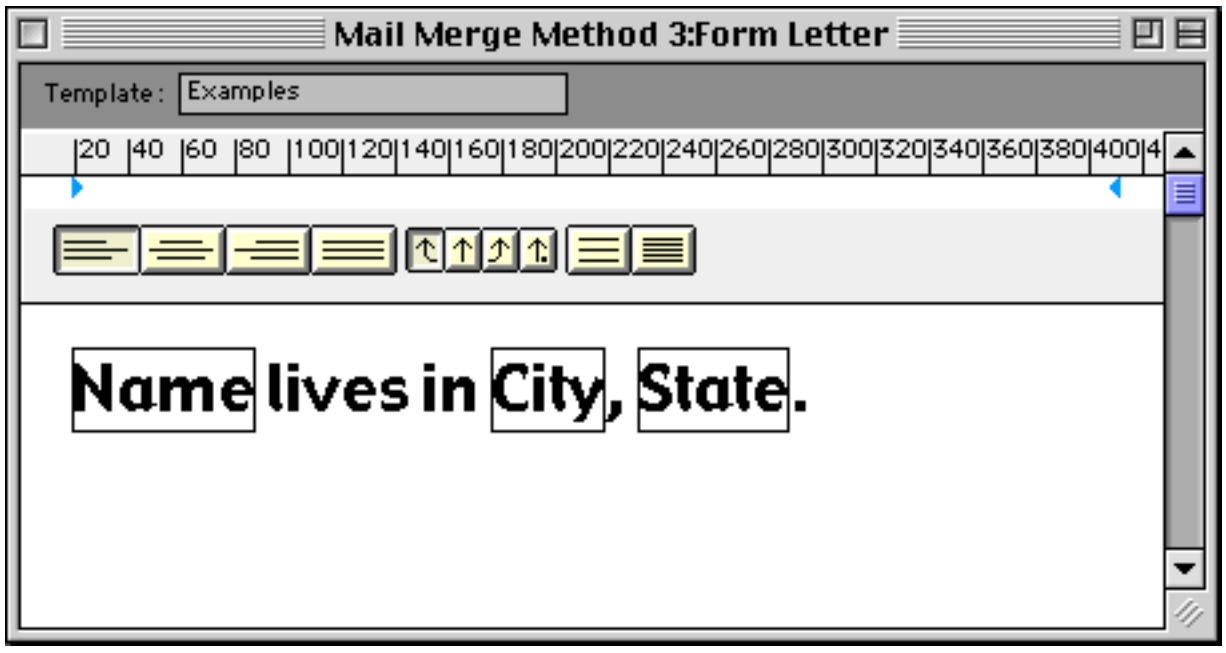
Any valid Panorama formula may be used. To merge in a data field, just type in the name of that field and apply the **Formula Merge** style.



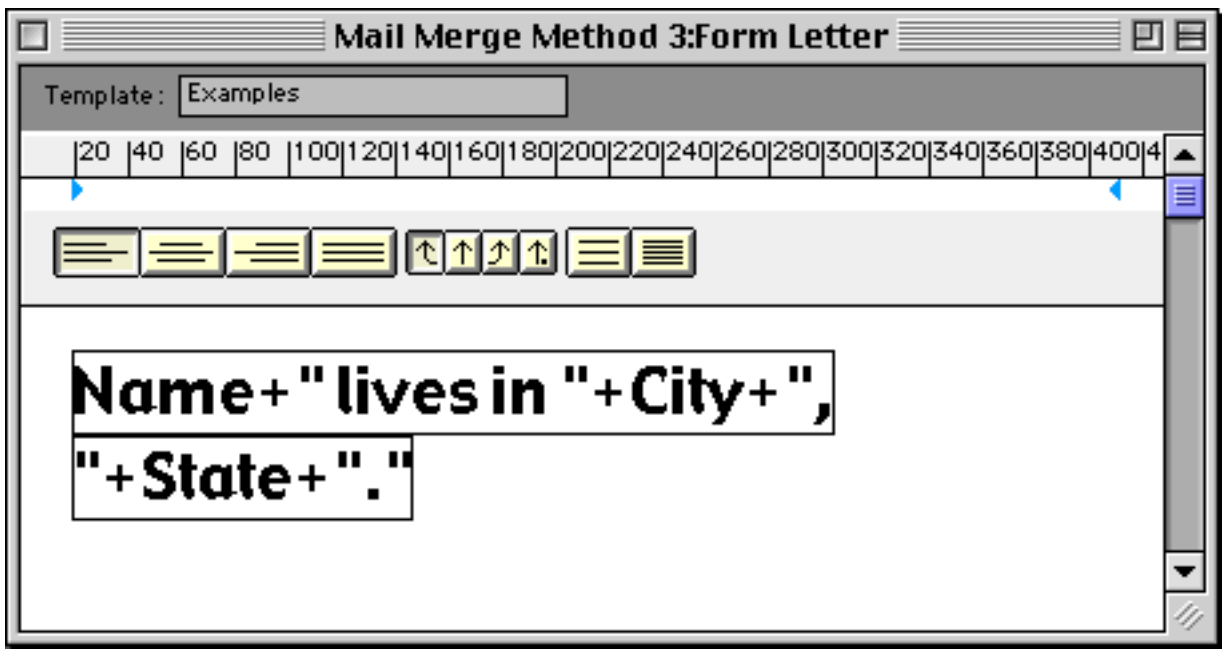
When the **Enter** key is pressed, the actual data appears.



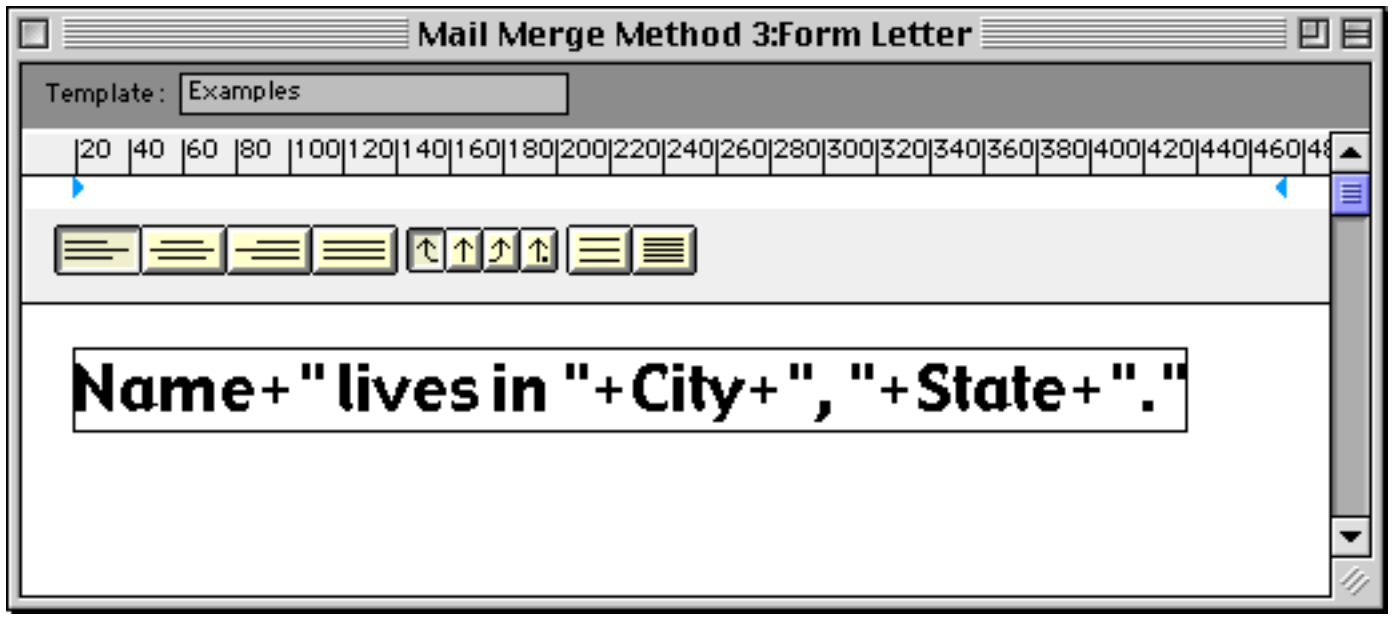
Click on the word processor to see the formulas again.



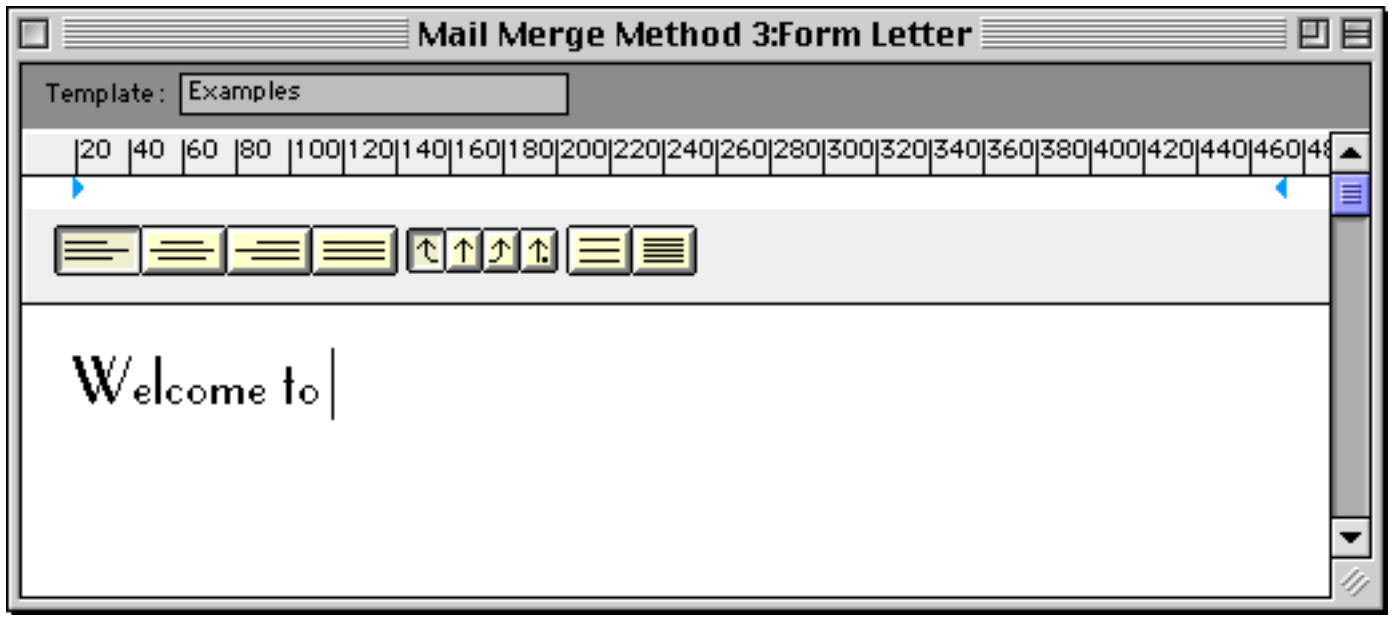
The previous example used three formulas, but it could be rewritten to use a single formula.



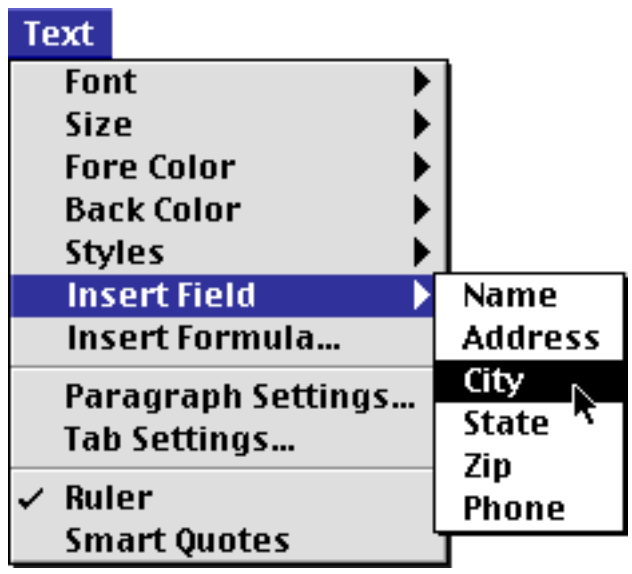
Since this formula wraps over two lines it looks like two formulas, but it is really one, as you can see by making the window wider.



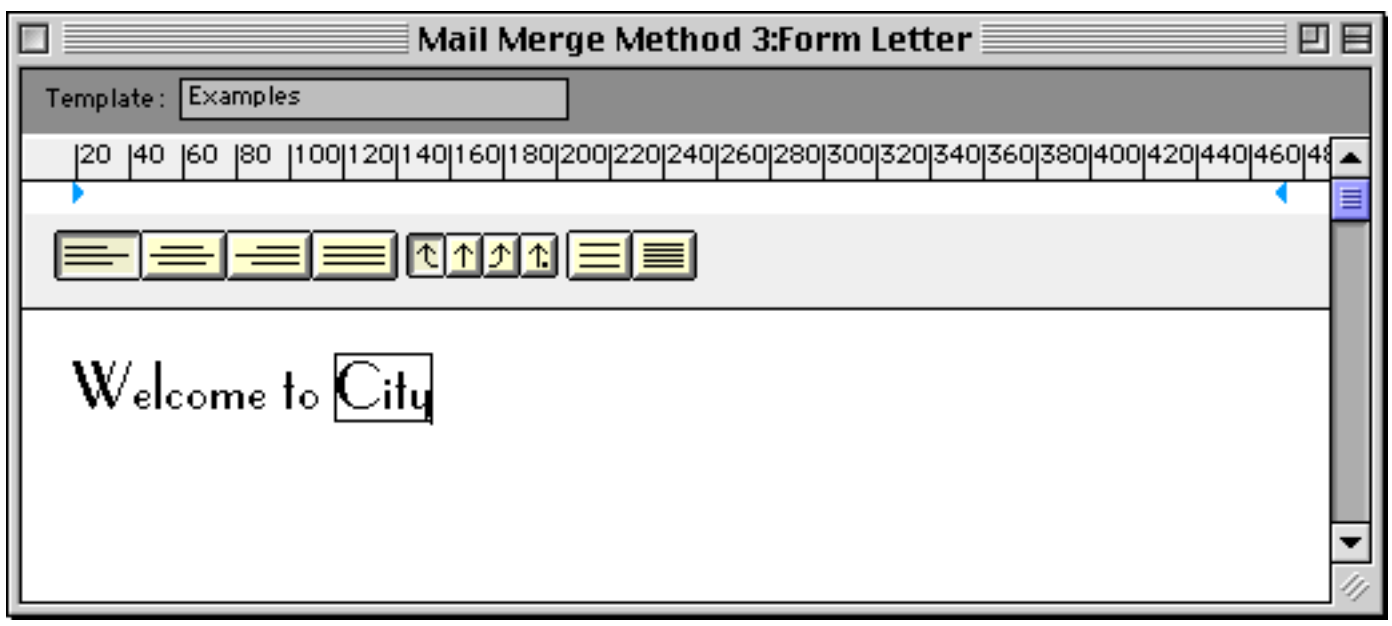
The **Text** menu contains two commands that can help you build formulas. The **Insert Field** submenu types in a field name and automatically turns on the **Formula Merge** style. For example, you can be typing in regular text, like this.



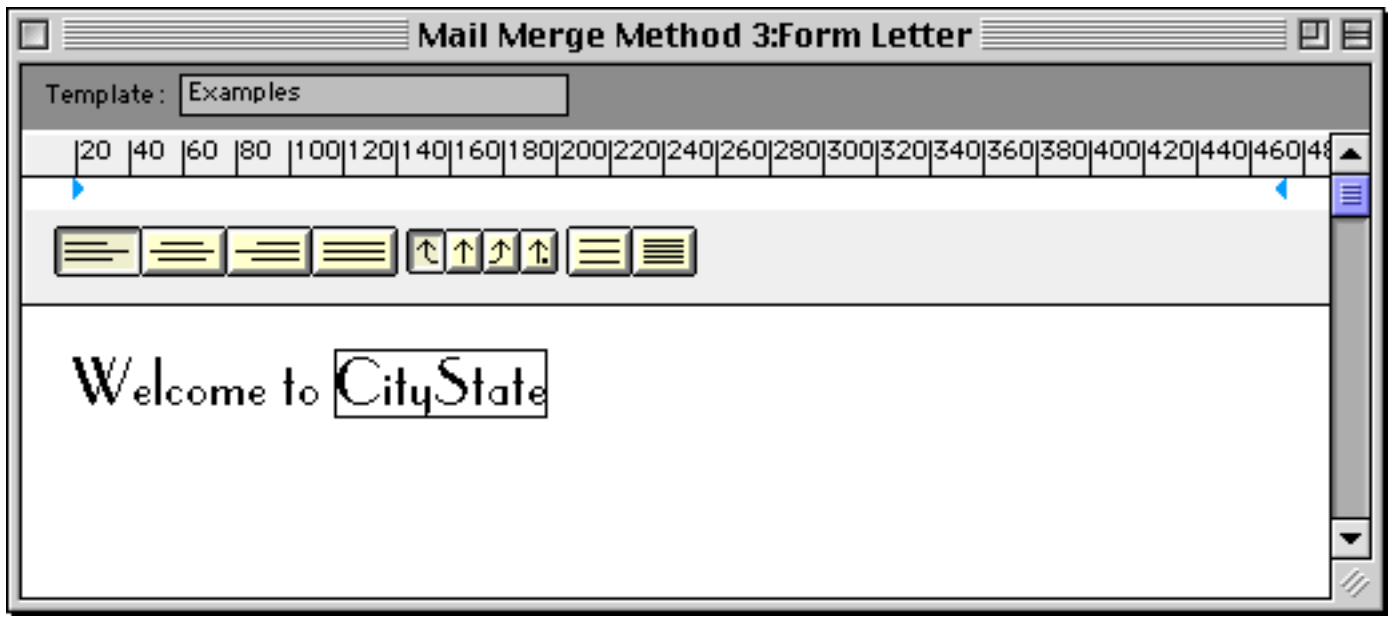
When you want to merge in a field, simply choose it from the menu.



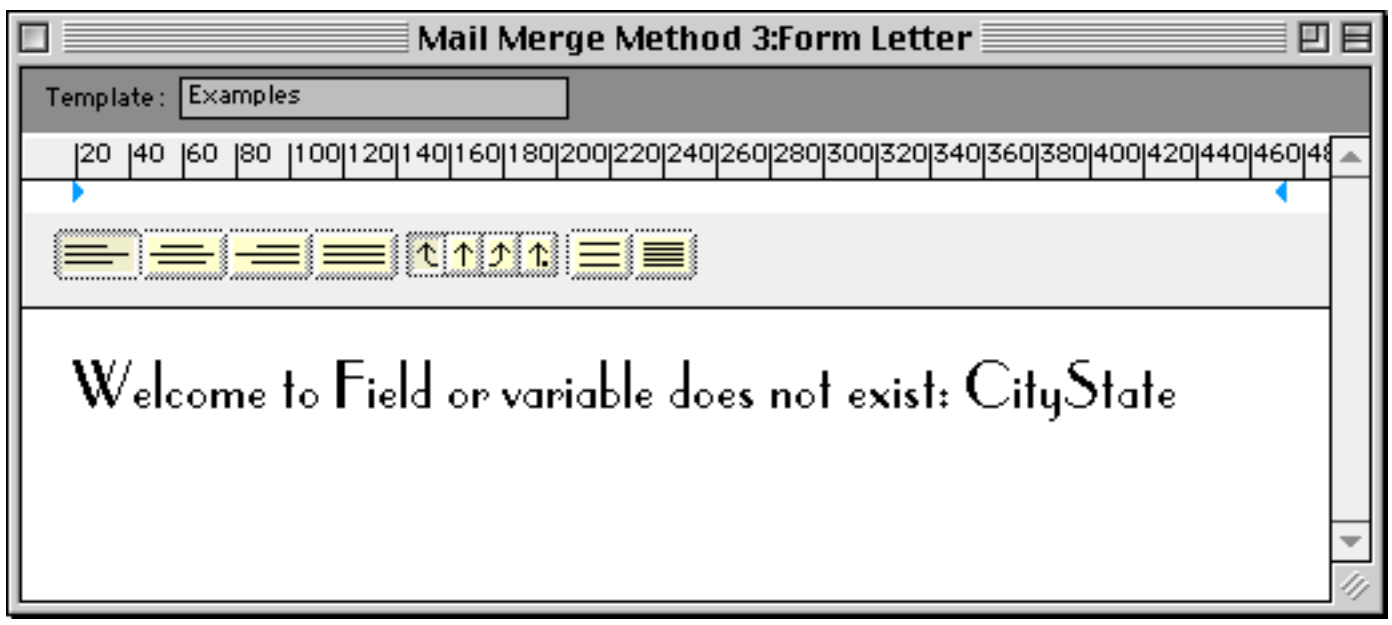
Panorama automatically inserts the field and selects the **Formula Merge** style.



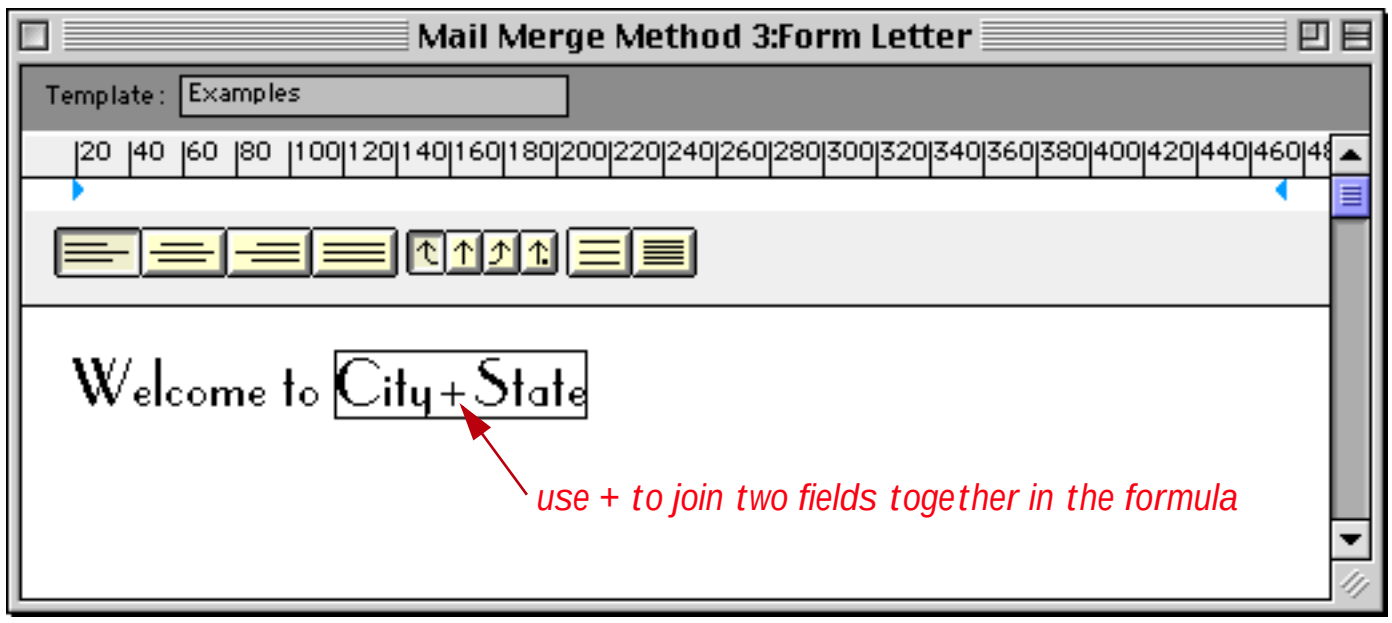
When you use the **Insert Field** menu you need to be careful not to insert two fields right next to each other. If you do, the two field names will merge together.



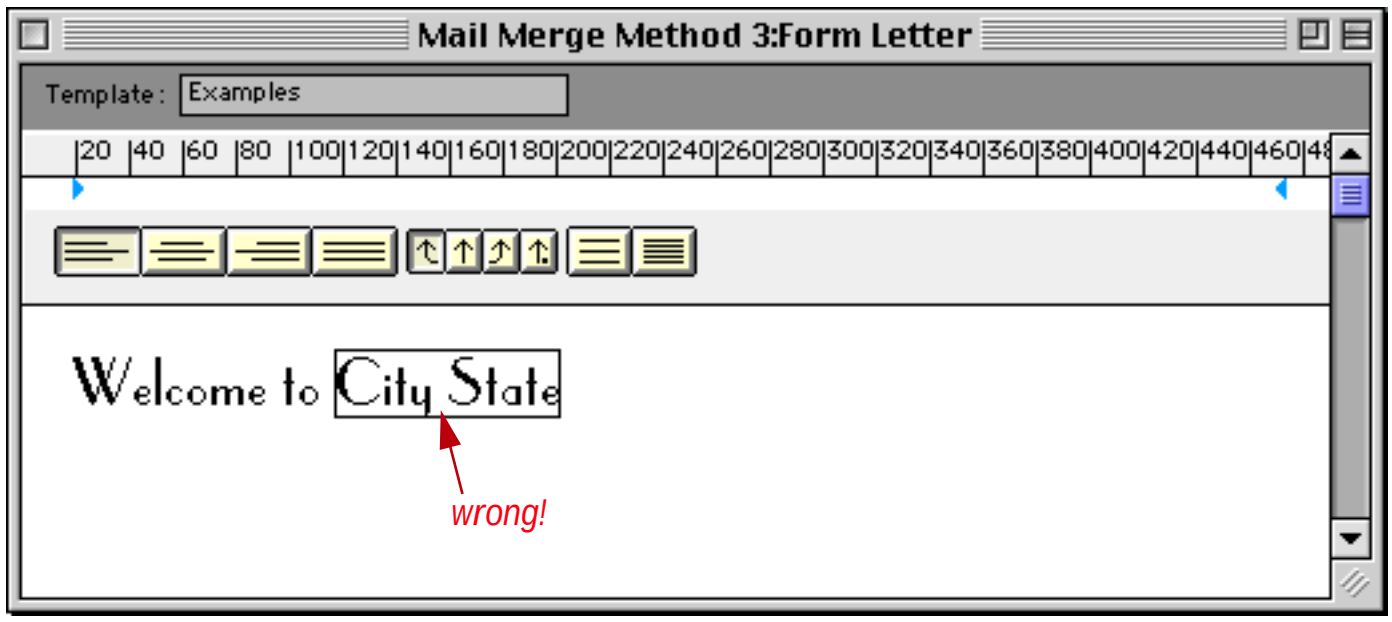
Since there is no field named CityState, this will produce an error when you press the **Enter** key.



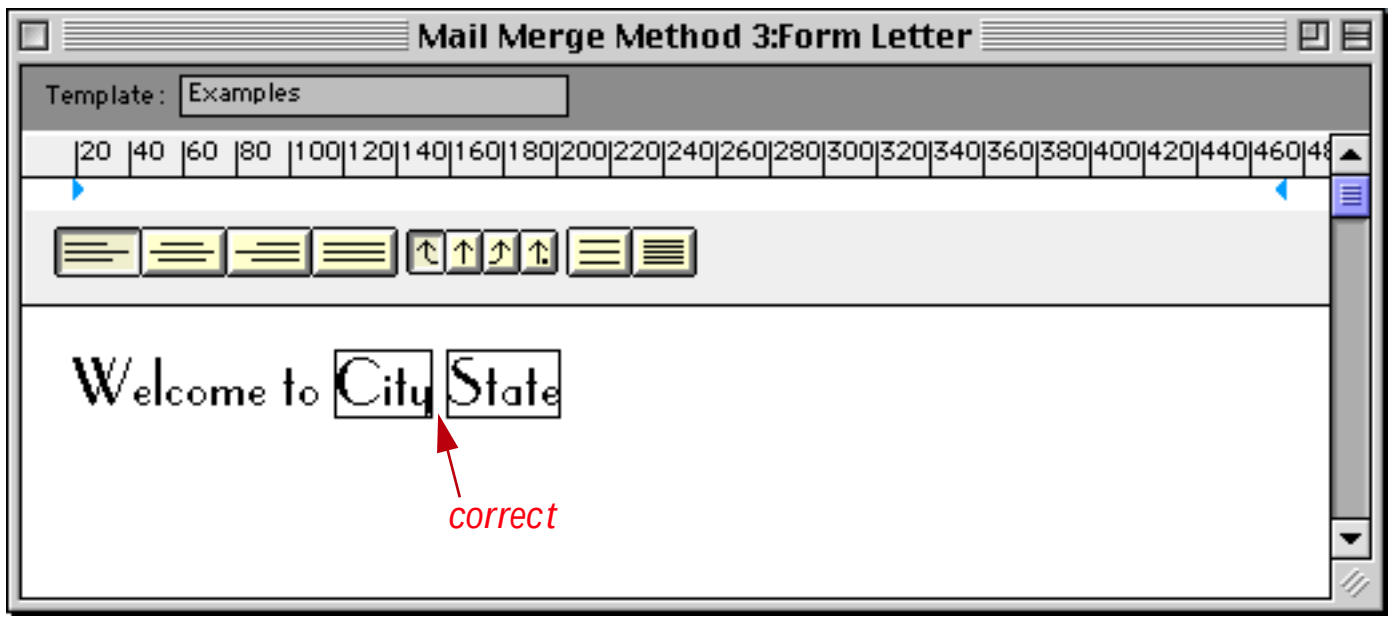
If you need the two fields to be adjacent to each other with no punctuation in between, you must use the + operator to make a valid formula.



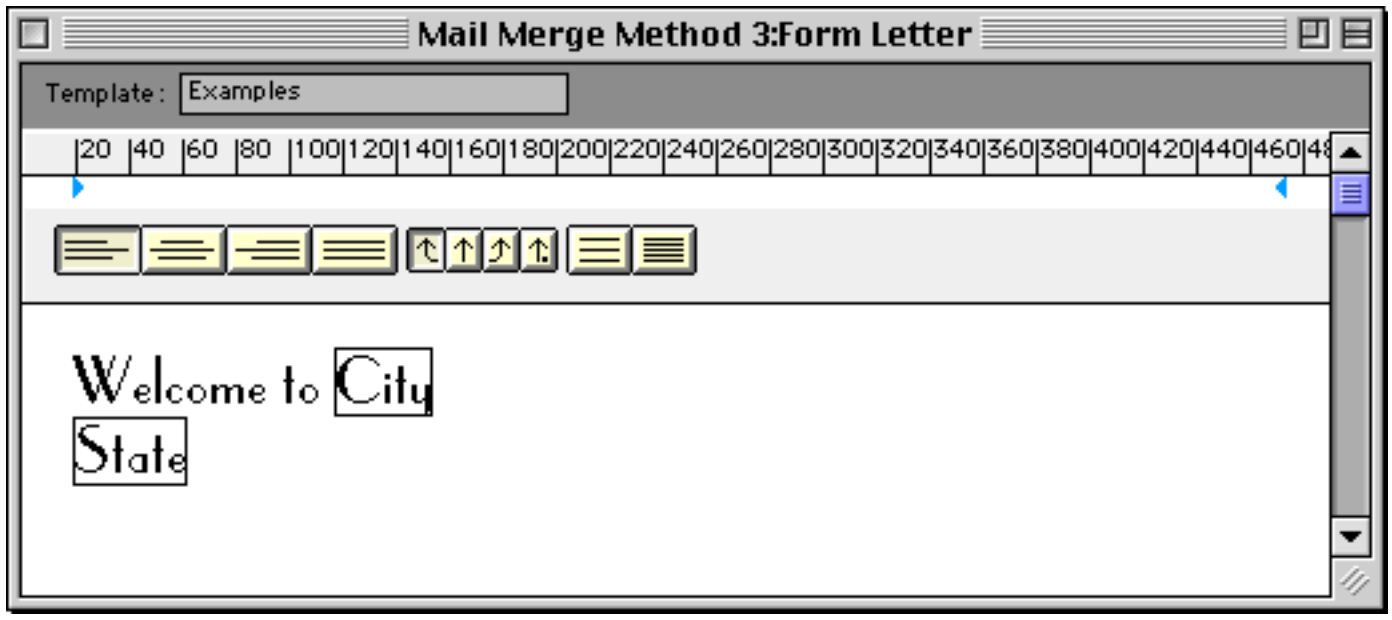
A more common solution is to put a space or other punctuation in between the fields. However, watch out! The punctuation character must not have the **Formula Merge** style selected.



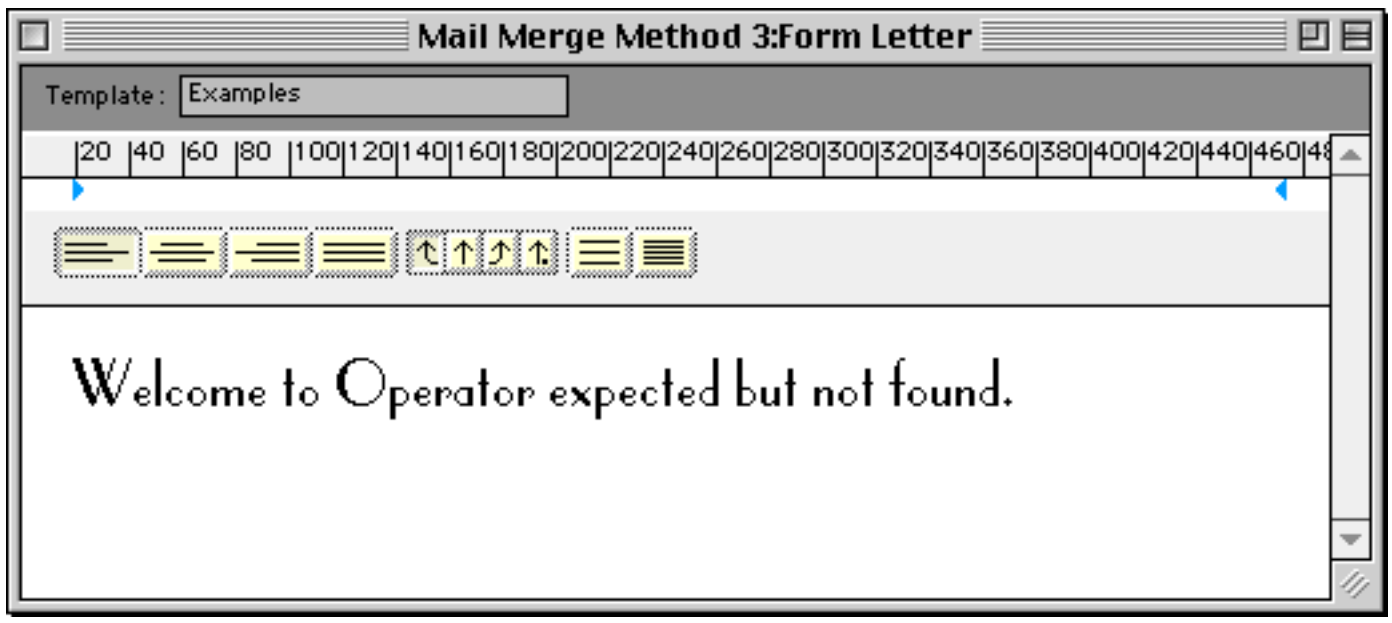
To make sure that the punctuation does not have the **Formula Merge** style selected, select the character and choose **Plain** from the style menu.



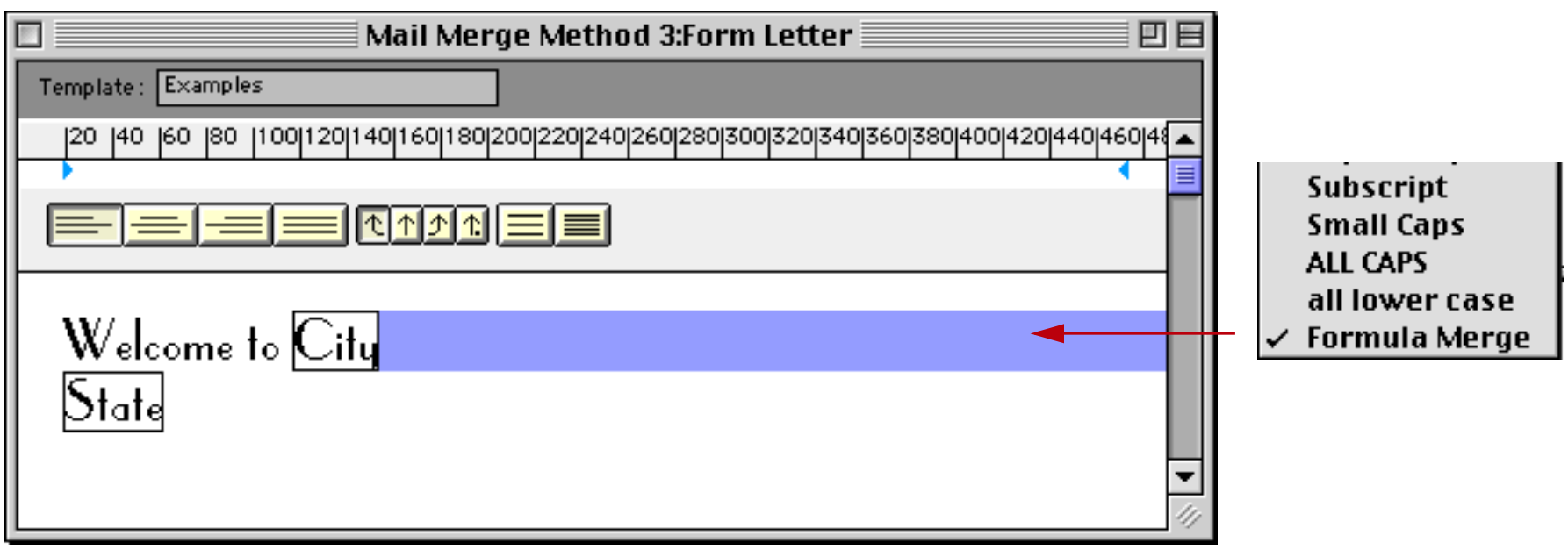
Here's a tricky problem. This looks like two fields in two separate formulas, which should be just fine.



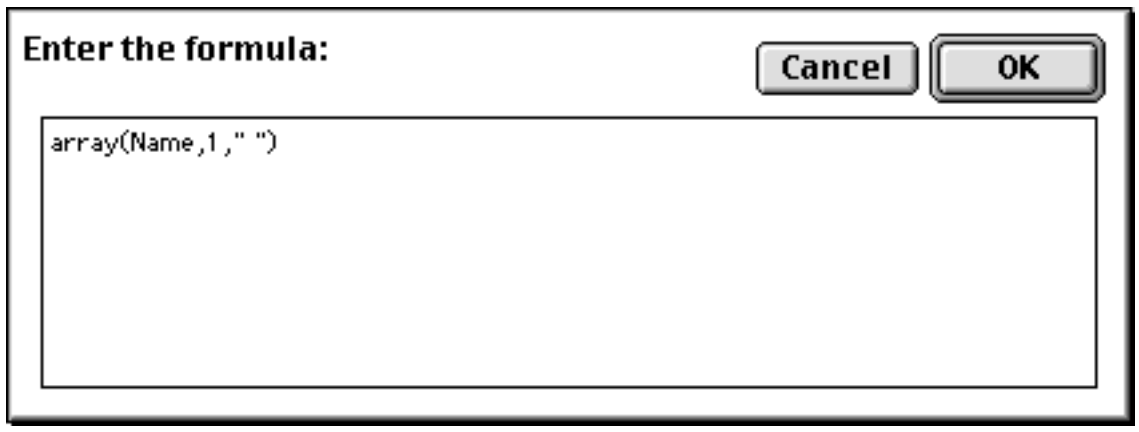
However, when **Enter** is pressed, an error message is displayed.



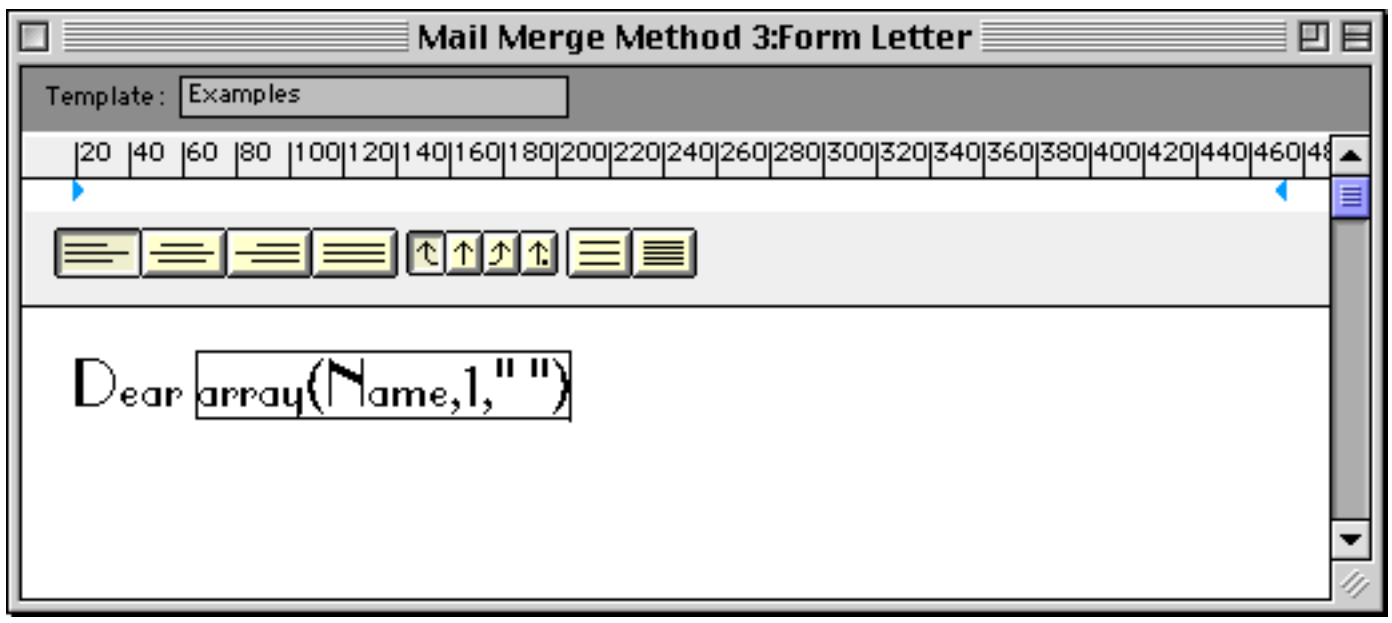
The problem is that it really is just a single formula, because the line break character has the **Formula Merge** style set. To fix this problem, select the line break character (as shown below) and un-check the **Formula Merge** style.



To insert a complete formula into the document, use the **Insert Formula** command in the Text menu. A dialog box appears allowing you to enter a formula.



Type the formula into the dialog. You can use the **Fields** and **Functions** menus to help you enter the formula. When you press **OK**, the formula will appear in the document, and it will already be in the **Formula Merge** style.



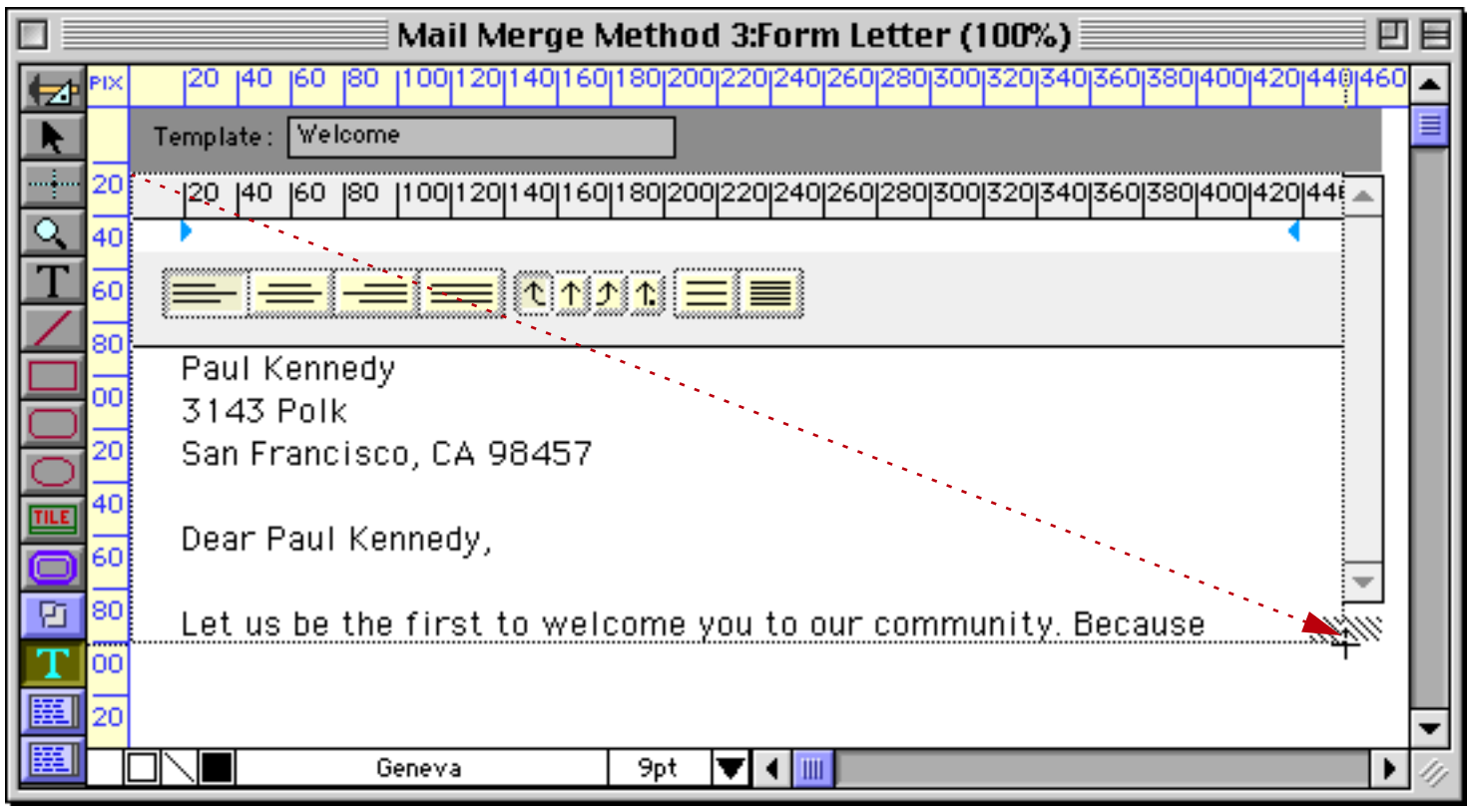
This particular formula extracts the first name from the Name field (to produce, for example, Dear Mary). The formula may use any field or function available in Panorama, including lookups to grab data from other databases. See “Using Formulas to Display Text” on page 621 to learn about some useful formulas for displaying text.

Forcing Merge Data to Update When Moving From Record to Record

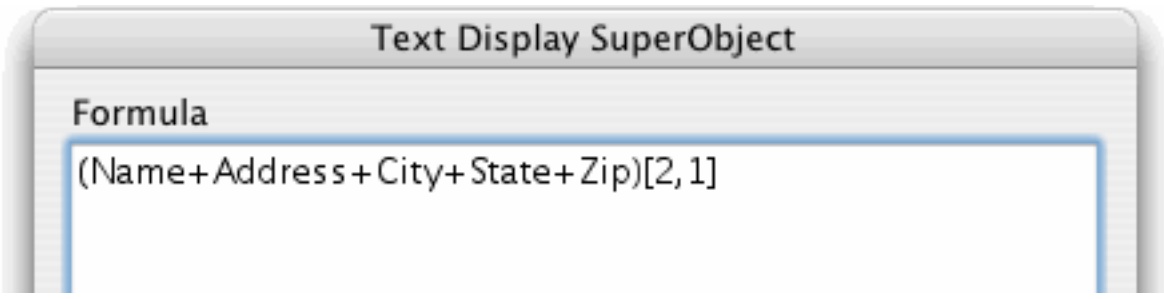
Mail merge documents are usually printed. If you display the document on the screen, you may think that it is broken because the merged data will not update as you move from record to record.

If displaying the merged data is important to you (as opposed to simply printing it) one solution is to overlay a Text Display SuperObject just underneath the Word Processor SuperObject. If it is configured in the way described below, the Text Display SuperObject will update whenever you move to a record. The Word Processing SuperObject will go along for the ride.

Start with the Word Processing SuperObject. Make sure it is positioned where you want it. Then select the Text Display tool and create a Text Display SuperObject on top. The new object should cover the entire Word Processor SuperObject except for the scroll bar.



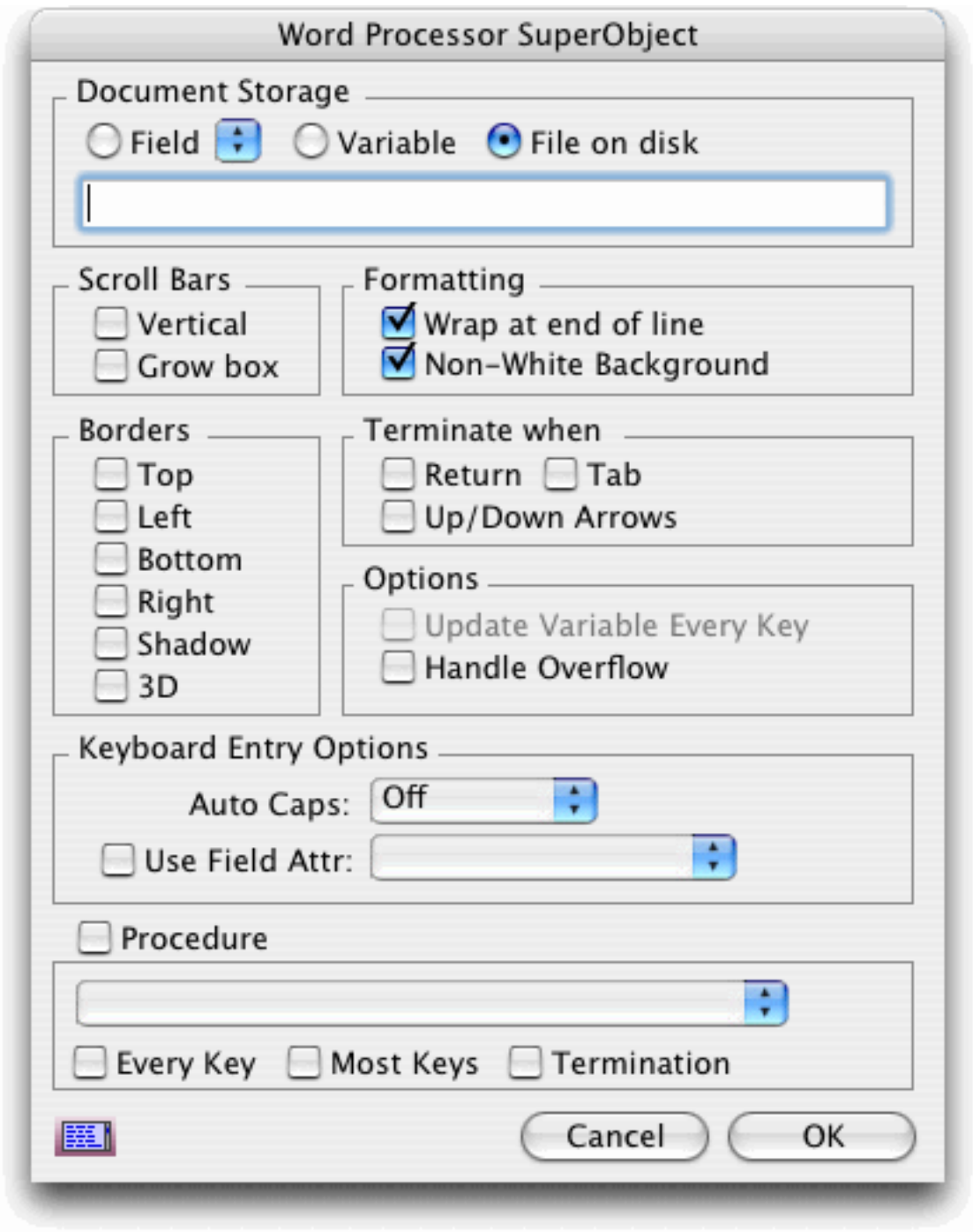
The Text Display formula should be any text field followed by the text funnel `[2,1]`. This text funnel tells Panorama to take the field and extract the text from character 2 to character 1. Since character 2 is after character 1, the result is nothing, nada, zip. So our new Text Display SuperObject won't display anything, but it will try! In the process it will force the Word Processor SuperObject to update also.



There is one more step to complete this mini-project. Select the Text Display SuperObject and use the **Send To Back** command to move it behind the word processor (see "Changing the Stacking Order" on page 569). That's it. Now go to Data Access Mode and see that the data merged in the word processor updates as Panorama moves from record to record.

Word Processor Options

The SuperObject™ Word Processor Properties dialog is divided into several sections.



Document Storage: This section of the dialog specifies whether the document will be stored in a field, variable, or separate file that will be edited by this object. See “Storing a Collection of Documents” on page 698 and “Setting up Storage for a Template Document” on page 702 for more information about setting up these options.

Scroll Bars: This section controls whether the vertical scroll bar is displayed, and whether space is reserved for a grow box in the lower right hand corner of the object. The word processor object does not support a horizontal scroll bar. Instead, the margins automatically adjust to the width of the object.

Wrap at End of Line: If text is too long to fit on a single line, it will usually “wrap” around to the next line. However, if this checkbox is turned off, the text will not wrap. Instead, the text will be continue off the right edge.

Non-White Background: We recommend that you use this option if the Word Processor object is over a color (non-white) background. If this option is turned on, Panorama will temporarily display a white background behind the text while it is being edited. (If you don’t use this option, you’ll find that portions of the background will turn white as you edit anyway. The result is ugly and possibly confusing, so that’s why we recommend you use the Non-White Background option.)

Terminate When: This section controls which keys indicate that you have finished editing this item of text. The **Enter** key always indicates you are finished editing. You may also specify that the **Return**, **Tab**, **Up Arrow**, or **Down Arrow** keys terminate editing. Usually these options will be left disabled so that these keys may be typed into your word processing documents.

Borders: This section controls any borders that are displayed around the document. You may separately control the top, bottom, left and right borders, or click on the word **Borders** to turn all four on or off at once. If the Shadow option is checked a drop shadow will appear. If the 3D option is checked a "three dimensional" border will appear around the object. The 3D border effect works best with a light gray background. (Note: You can control the background color for the entire form with the **Form Preferences** command in the Setup menu, see "Form Background Colors" on page 582.)

Update Variable Every Key: This option only works when the document is stored in a variable, not in a field or disk file. If this option is enabled, the Word Processor will update the value of the variable after every key is pressed. This is especially handy when the Word Processor is set up to trigger a procedure after every key or most keys (see next section).

Overflow: This option is used in conjunction with an overflow tile for printing documents that are more than one page long. See "Printing Multiple Page Documents" on page 724 for more information about this option.

Procedure: This section specifies what procedure is associated with this Word Processor object, and when that procedure will be triggered. To select or change the procedure associated with this object, use the pop-up menu. To disable the procedure, clear the checkbox.

There are three choices for triggering a procedure: Every Key, Most Keys, and Termination. The Termination option simply means that the procedure will be triggered when the user signals that he or she has finished editing the document by pressing the **Enter**, **Return**, **Tab**, **Up Arrow**, or **Down Arrow** keys. (Note: If the Word Processor is associated with a field, a procedure may be triggered even if no procedure is assigned in the Word Processor dialog. If the field has a procedure assigned to it in the design sheet, it will be triggered (see "Automatically Triggering a Procedure" on page 350). If there is no Termination procedure and no design sheet procedure, the **.ModifyRecord** procedure [if any] will be triggered. See ".ModifyRecord" on page 1512.)

If the Every Key option is checked, the procedure will be triggered every time the user presses a key. For example, you might use this option if you wanted to count the user's keystrokes.

If the Most Keys option is checked, the procedure will be triggered after every key when the user types slowly, but will not be triggered for each key when the user types several characters quickly in a row. The procedure will not be triggered until the user pauses in his or her typing. This option often works as well as the Every Key option but usually appears much smoother and faster to the user because the procedure is not being triggered as frequently while the user types. Possible applications for the Most Keys option include counting the characters or words being edited, performing a calculation (metric conversion, for example), checking spelling, etc.

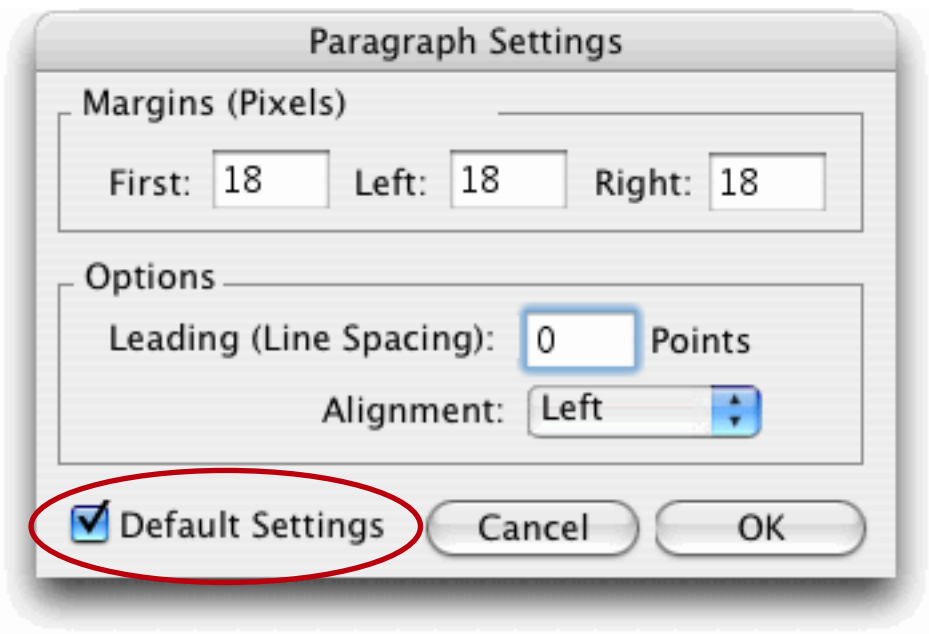
Auto Caps: This section only appears if the Word Processor is associated with a variable or file on disk. Use the pop-up menu to control automatic capitalization of the text as it is entered. You can force the text to all upper case (ABC), word caps (Abc), or capitalization of the first letter in each sentence. If the Word Processor is associated with a field, the Word Processor will use the Auto Caps setting for that field (set in the design sheet or the Field Preferences dialog.)

Use Field Attributes: This section only appears if the Word Processor is associated with a variable or file on disk. You can use this if you would like the Word Processor to use the attributes assigned to one of the fields in your database. If you select a field with the pop-up menu, the Word Processor will use that field's settings for Input Pattern, Range, Clairvoyance, Space Bar Tab and Duplicates. Of course if the Word Processor is associated with a field, the Word Processor will always use the attributes for that field (set in the design sheet or the Field Preferences dialog.)

Default Font and Text Size for New Documents

When you create a new document, the font, size, indents, tabs and line spacing are all set to default values. The default values for the font and text size are set by setting the font and size of the object in graphics mode. For example, if you want new documents to default to 14 point Palatino, go to graphics mode, select the word processor object, and select Palatino from the **Font** submenu and 14 from the **Size** submenu (or use the Graphic Control Strip, see "Font" on page 529).

To set the default indents, tabs and line spacing, first set up these properties the way you want them (using the ruler). Once the indents, tabs and line spacing are set up, open the Paragraph Settings dialog (in the Text menu). Check the Default Settings checkbox, then press the **OK** button.



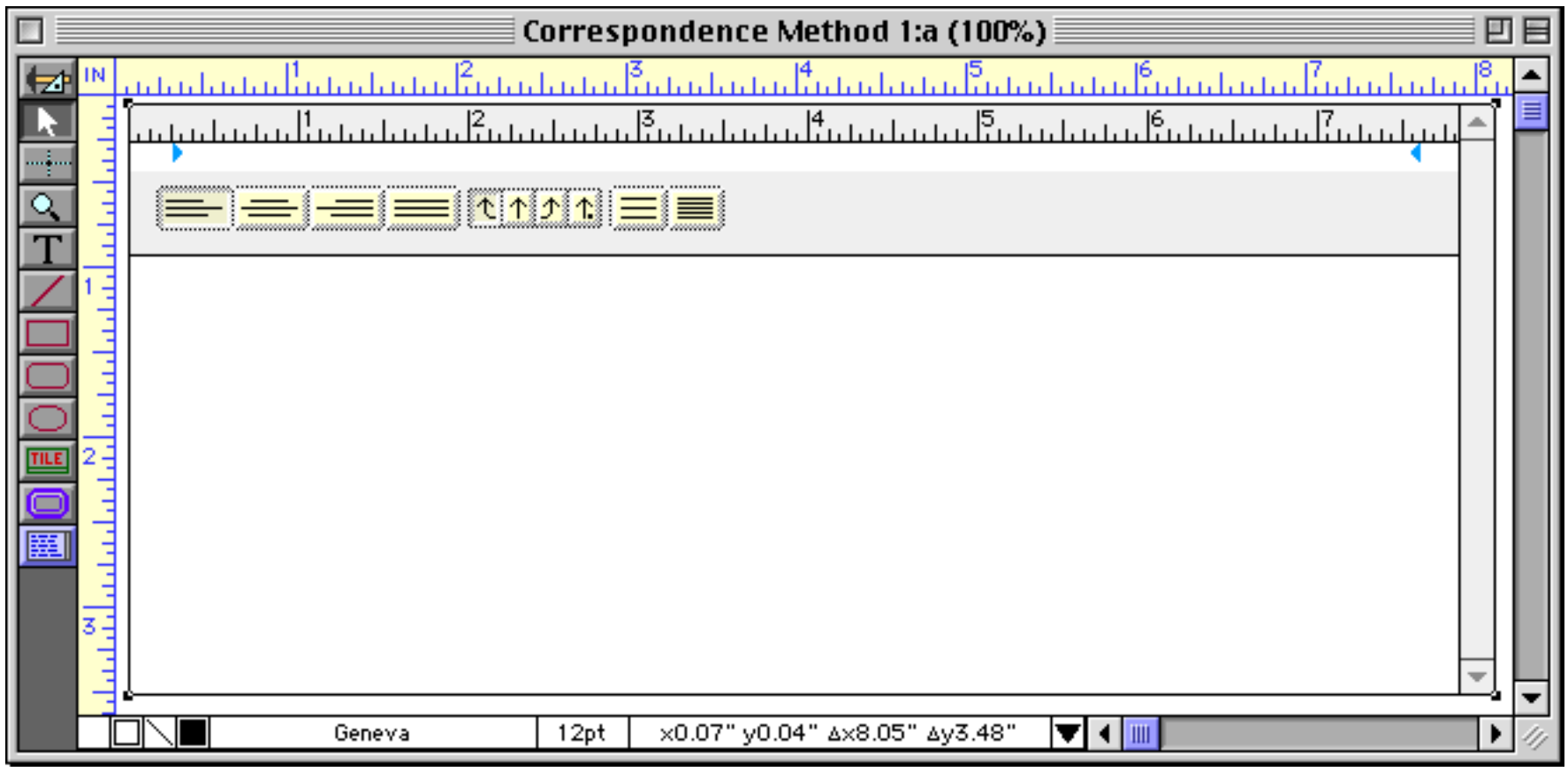
Note: If a document consists entirely of text using the default settings (no font changes, style changes, indent changes, etc.), Panorama will save only the text without any style information. This makes the document much smaller. If you later change the default settings, this document will reflect the new defaults the next time it is displayed or opened for editing.

Printing Word Processor Documents

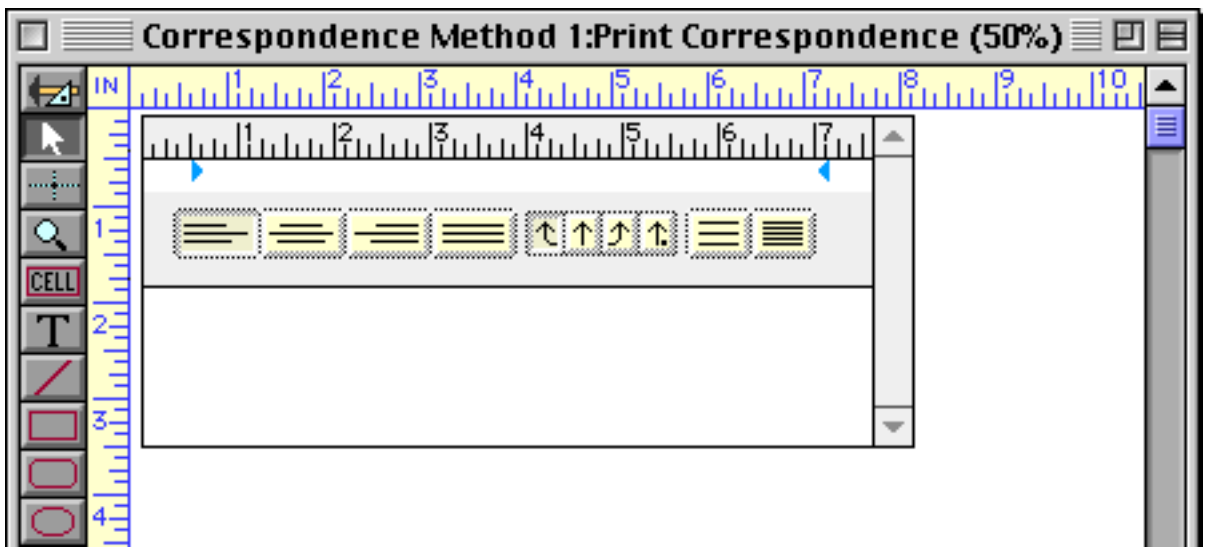
The primary use for the word processor is usually to create printed documents — letters, memos, correspondence etc. In this section we will cover some tips for printing these documents.

Normally when a document is printed you won't want any extra doo-dads — no ruler, no borders, no scroll bar. When you are displaying and editing the document, however, you need these. You could change these settings each time you want to print. A better method is to create two separate forms: one set up for editing documents, and one for printing.

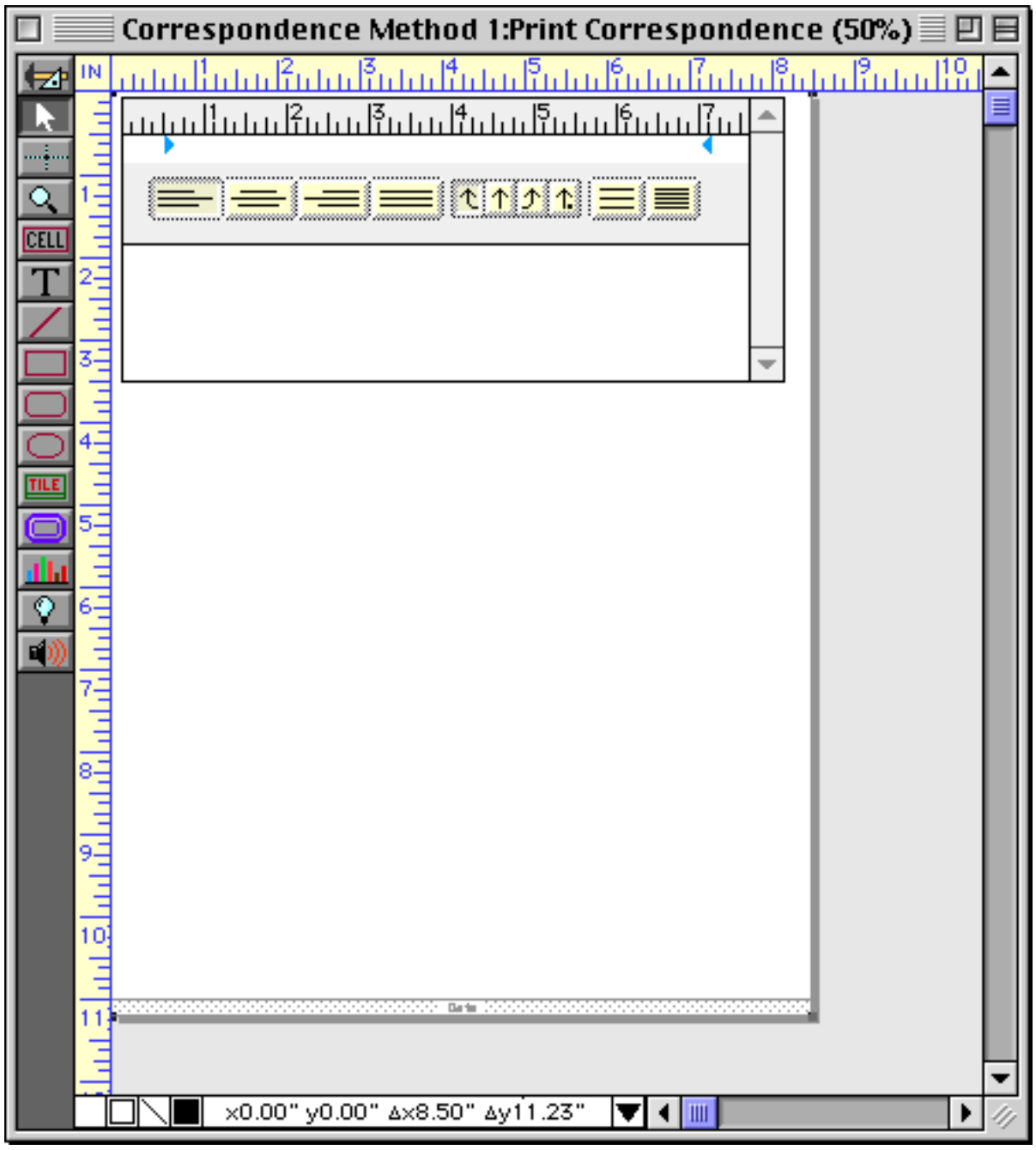
To illustrate this idea we'll start with a Word Processor SuperObject that has been set up for editing a document (see "Creating and Working With Word Processor SuperObjects" on page 673).



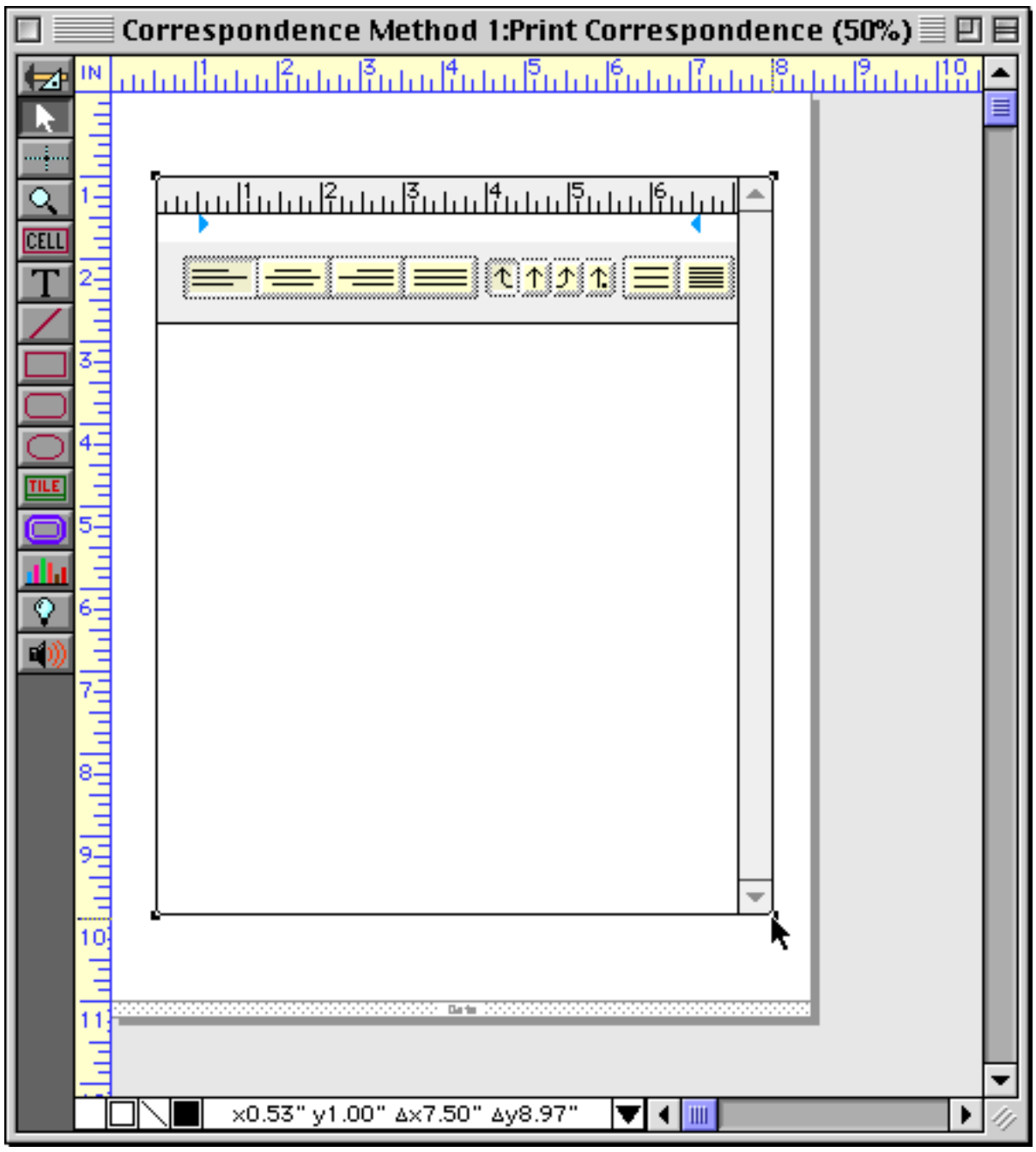
Select the object (as shown above) and use the **Copy** command to copy it onto the clipboard (see "Cut, Copy, and Paste" on page 566). Then use the **New Form** command in the View menu to create a new form (see "Creating a New Form" on page 487), and **Paste** the Word Processor SuperObject into the new form. Since you'll probably want the printed correspondence to fill an entire page, use the **Magnify** tool to zoom out and make the an 8 1/2 by 11 inch area visible (see "Magnification and Reduction" on page 579).



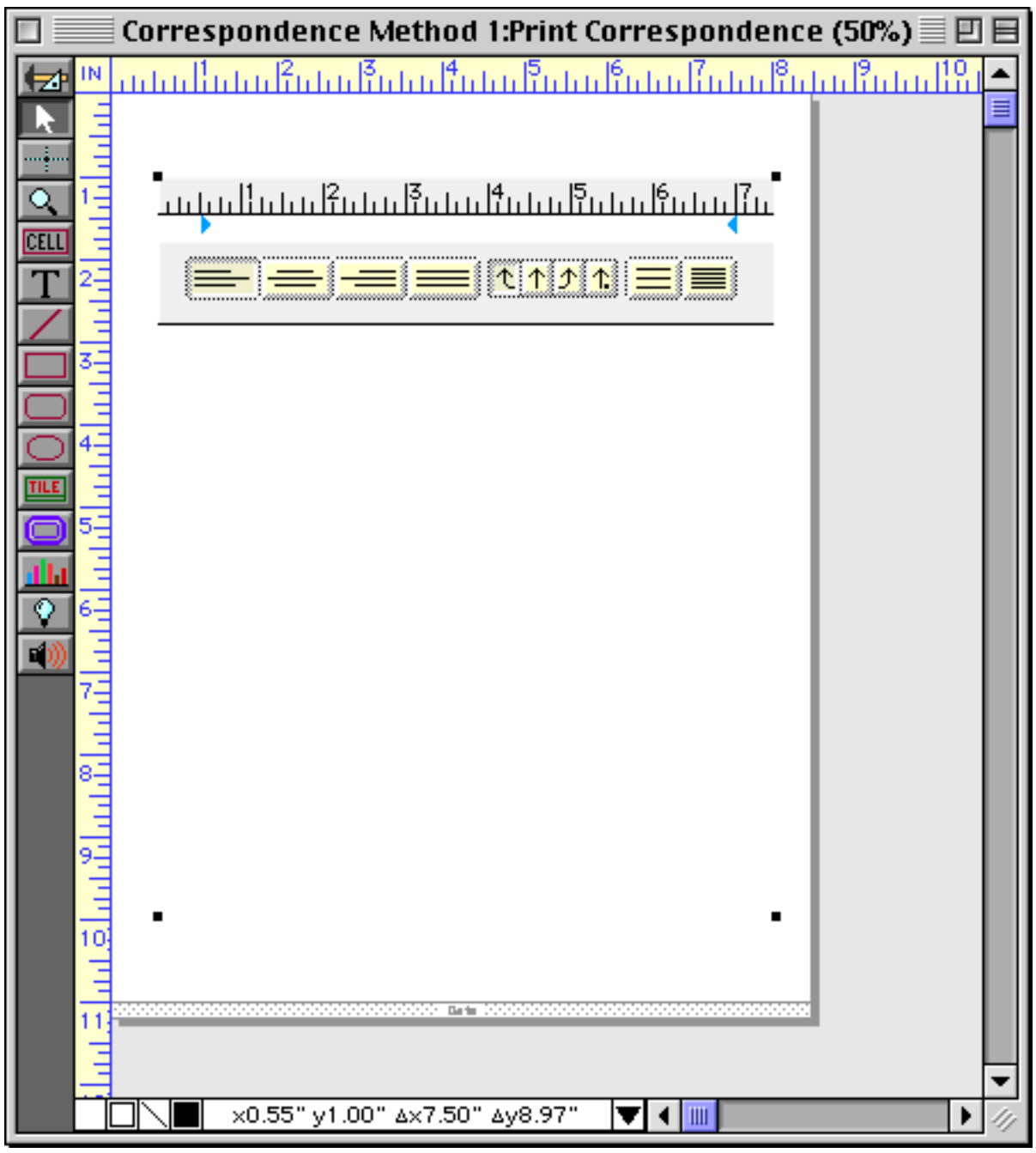
Next, use the **Tile** tool to create a data tile that is 8 1/2 by 11 inches (see "Working with Tiles" on page 1050). You can use the Dimensions dialog to help you set the exact dimensions (see "Viewing and Setting Exact Object Dimensions" on page 512). (You may need to adjust these dimensions later depending on your printer.)



Now expand and position the Word Processor SuperObject on the data tile just as you would like the final correspondence to appear on the printed page.



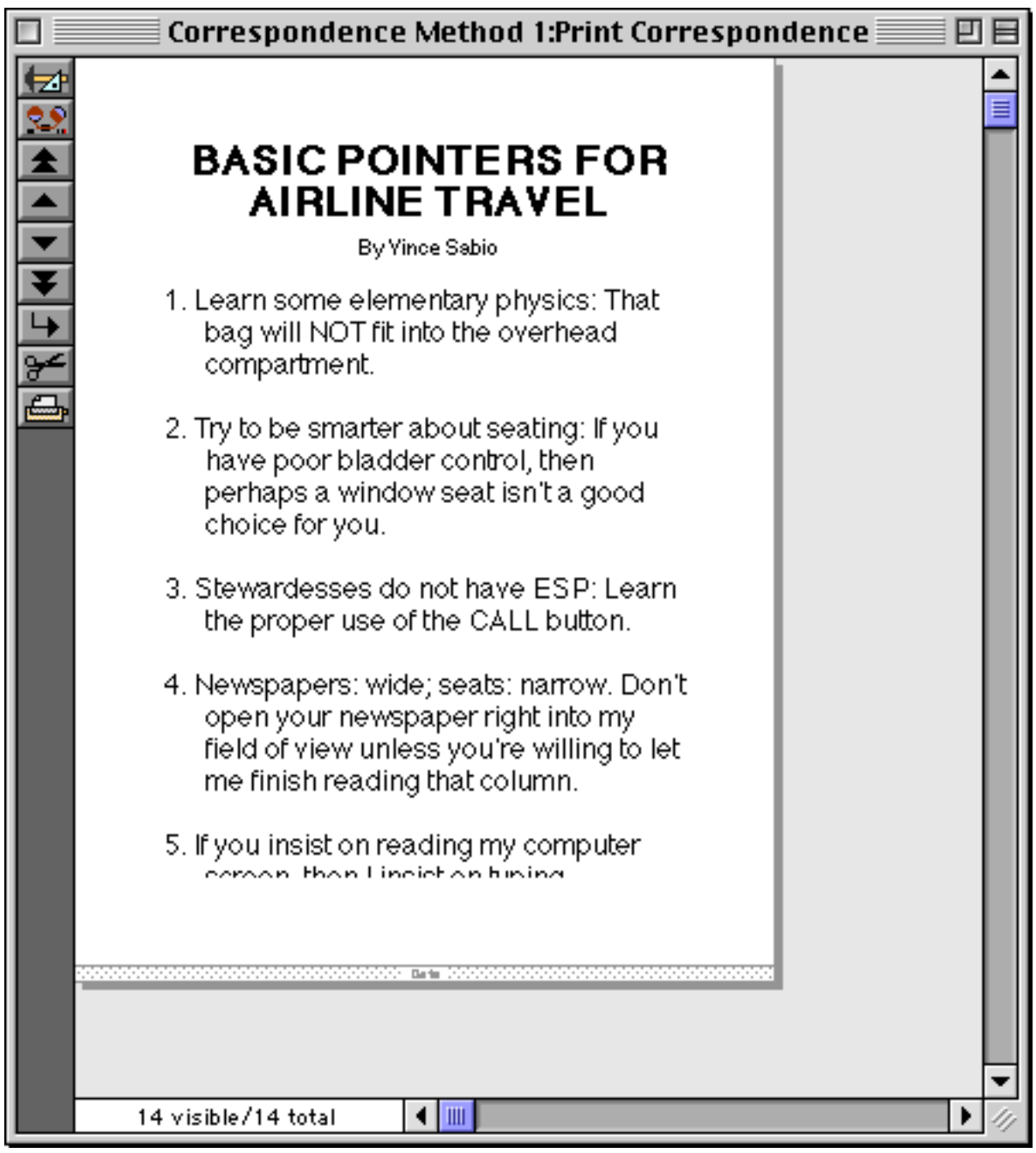
The next step is to double click on the word processor to open the configuration dialog. Use the dialog to turn off the borders and the scroll bar (see "Word Processor Options" on page 716).



The final step is to turn off the ruler. To do this, switch into Data Access Mode. Click on the word processor to edit the text, then use the **Text** menu to disable the ruler.



Voila! The form is now ready to print.



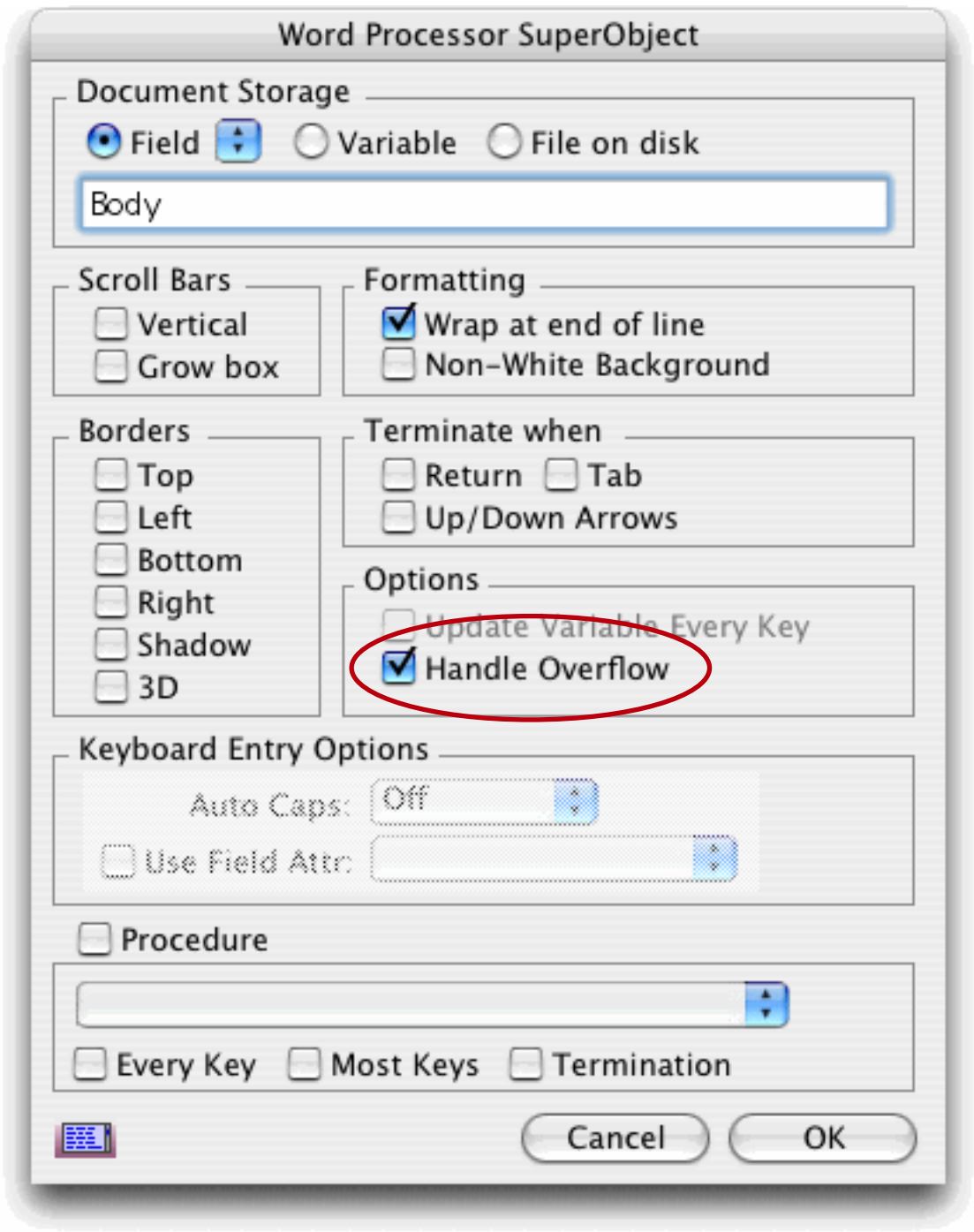
Edit all your documents in the original form, which still includes the ruler and scroll bar. When you need to print switch to the new form, which prints without these extra items. If you like, you can add a procedure to your database to automatically switch forms and print. Here's one way this procedure could be written.

```
local windowOne
windowOne=info("windowname") /* remember where we came from */
goform "Print Correspondence"  /* switch windows */
printonerecord dialog   /* or print, if you want to print all selected records */
goform windowOne    /* switch back to original window */
```

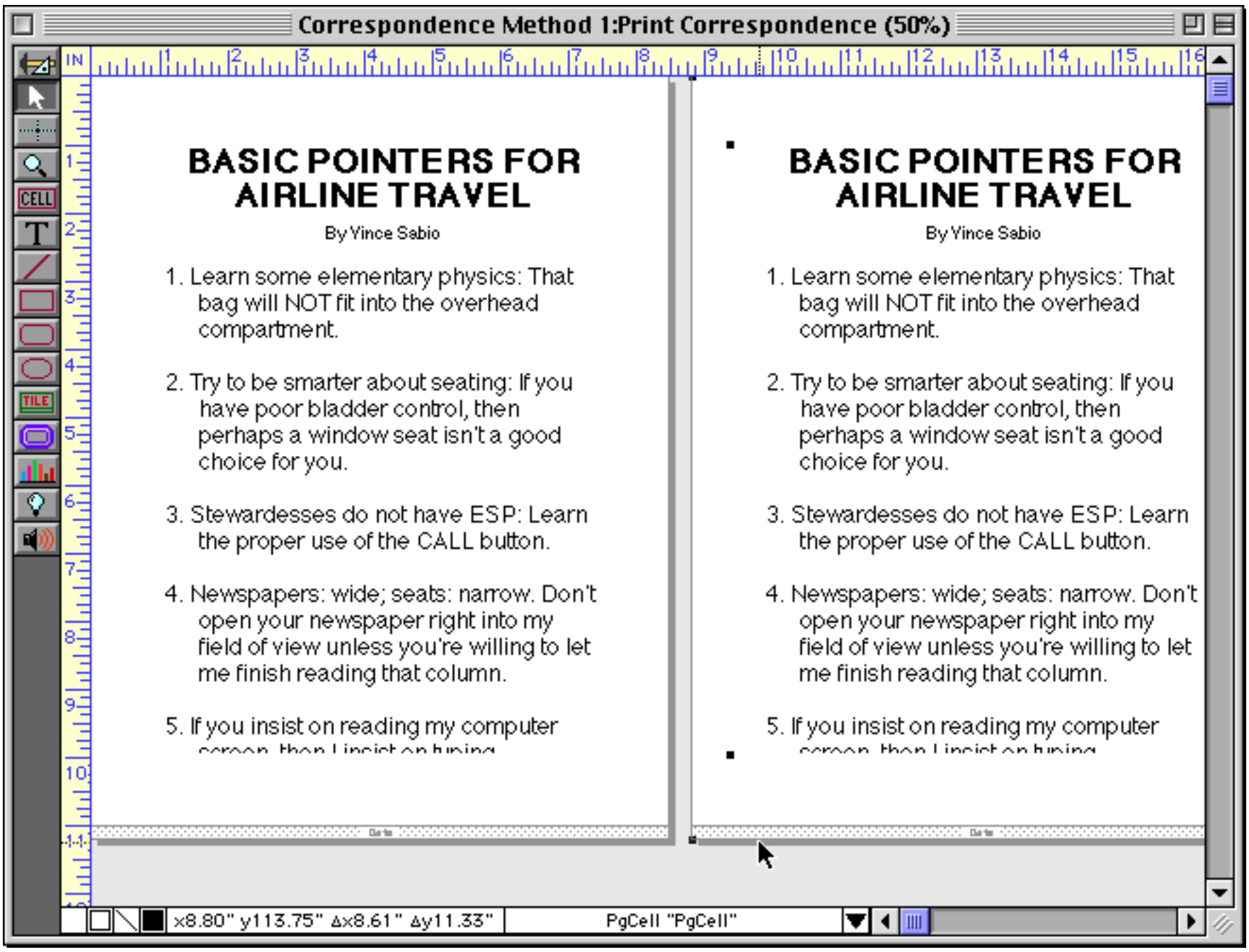
Printing Multiple Page Documents

The technique described in the previous section works fine for single page documents. Of course, many documents are more than one page long. In this section we will extend the example to allow the printing of multiple page documents.

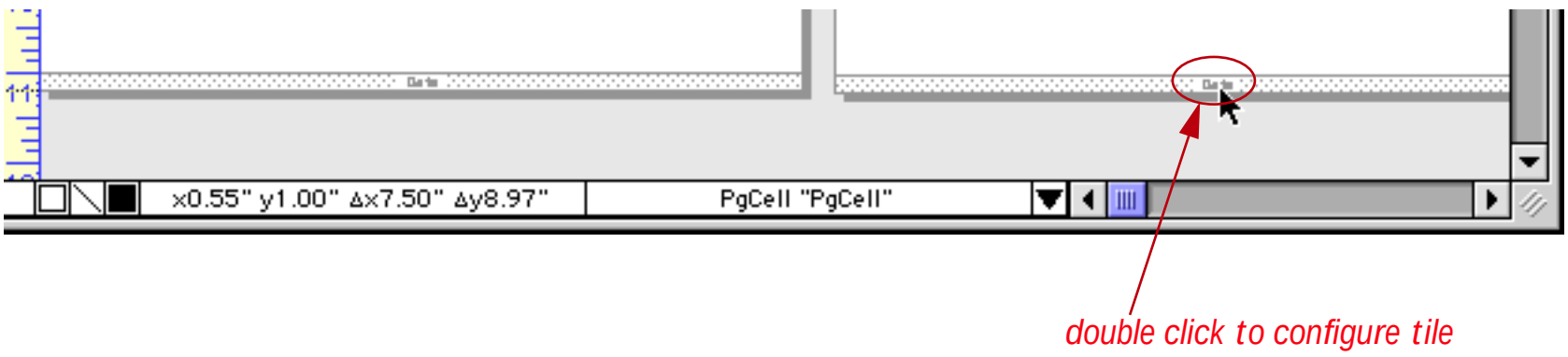
Start with the form created in the previous example, and switch back to Graphic Design Mode. Double click on the word processor. In the configuration dialog, enable the Handle Overflow option, then press **OK**.



Now choose **Select All Objects** from the Edit menu. While holding down the **Option** key (Macintosh) or **Alt** key (Windows), drag a copy of all the objects on the form to an empty spot (see "Drag Duplicating" on page 561).



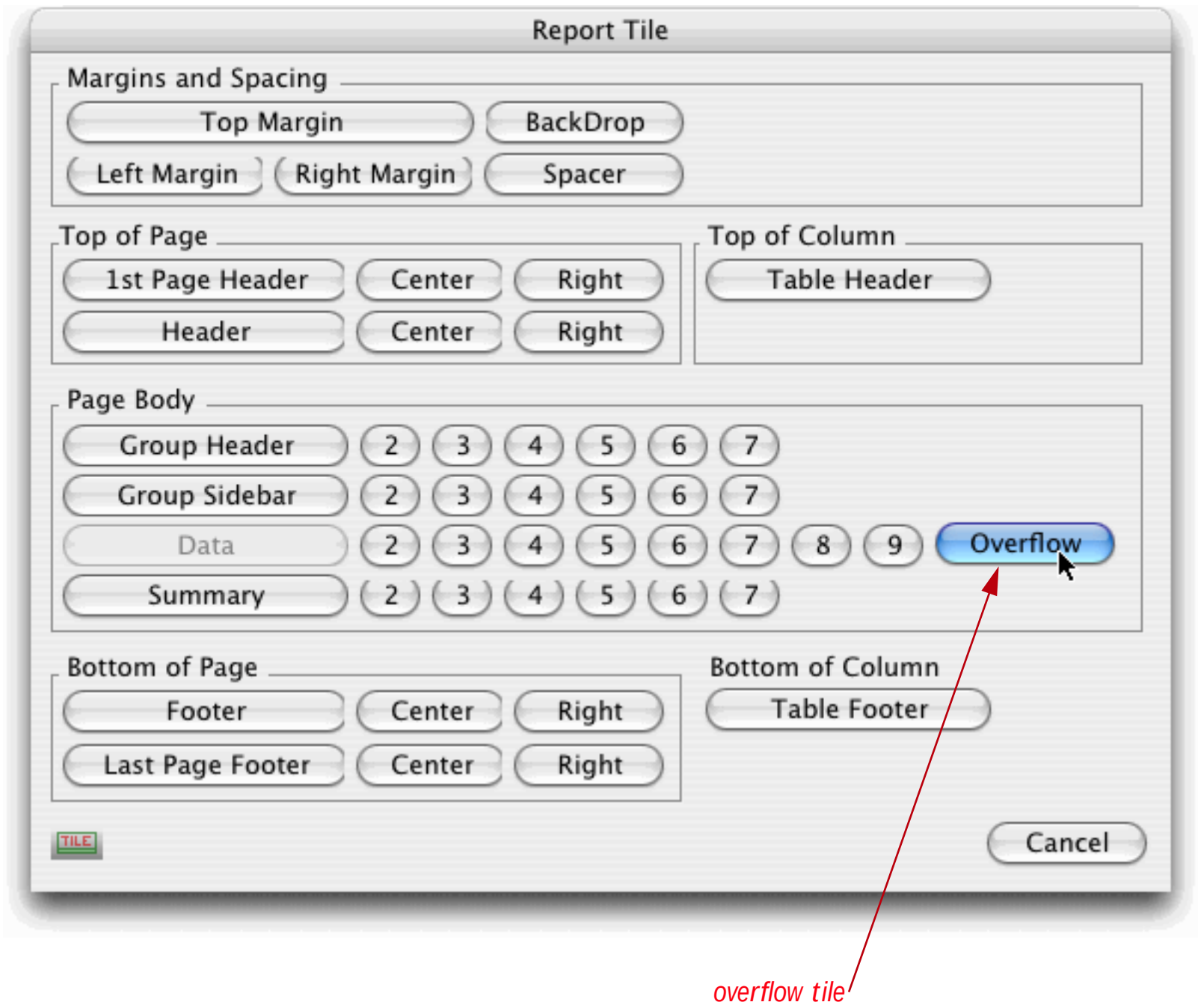
Now we need to turn the duplicate Data tile into an Overflow tile (see "Printing Data that Overflows a Page" on page 1104). To do that, double click on the word Data in the middle of the drag bar at the bottom of the tile.



This opens the tile configuration dialog (see “Working with Tiles” on page 1050).



Click on the **Specialized Tiles** button to open a larger dialog. This dialog has over 40 different kinds of tiles to choose from. Pick the **Overflow** tile, which is activated with the button shown in the illustration below.



That’s all there is to it. The form will now automatically print as many pages as the document requires — 1, 2, 3, or 1,000! For more information on overflow printing see “Printing Data that Overflows a Page” on page 1104.

Using the Mini Correspondence Wizard

Panorama includes a pre-built database for handling general word processing chores — the **Mini Correspondence** wizard. This database may be used for general correspondence (letters, memos, etc.) and to create mail merge letters that are customized and sent to a group of recipients. You can use the **Mini Correspondence** wizard as is or you can modify it or even take components of the wizard and include them in your own databases.

Creating a New Letter

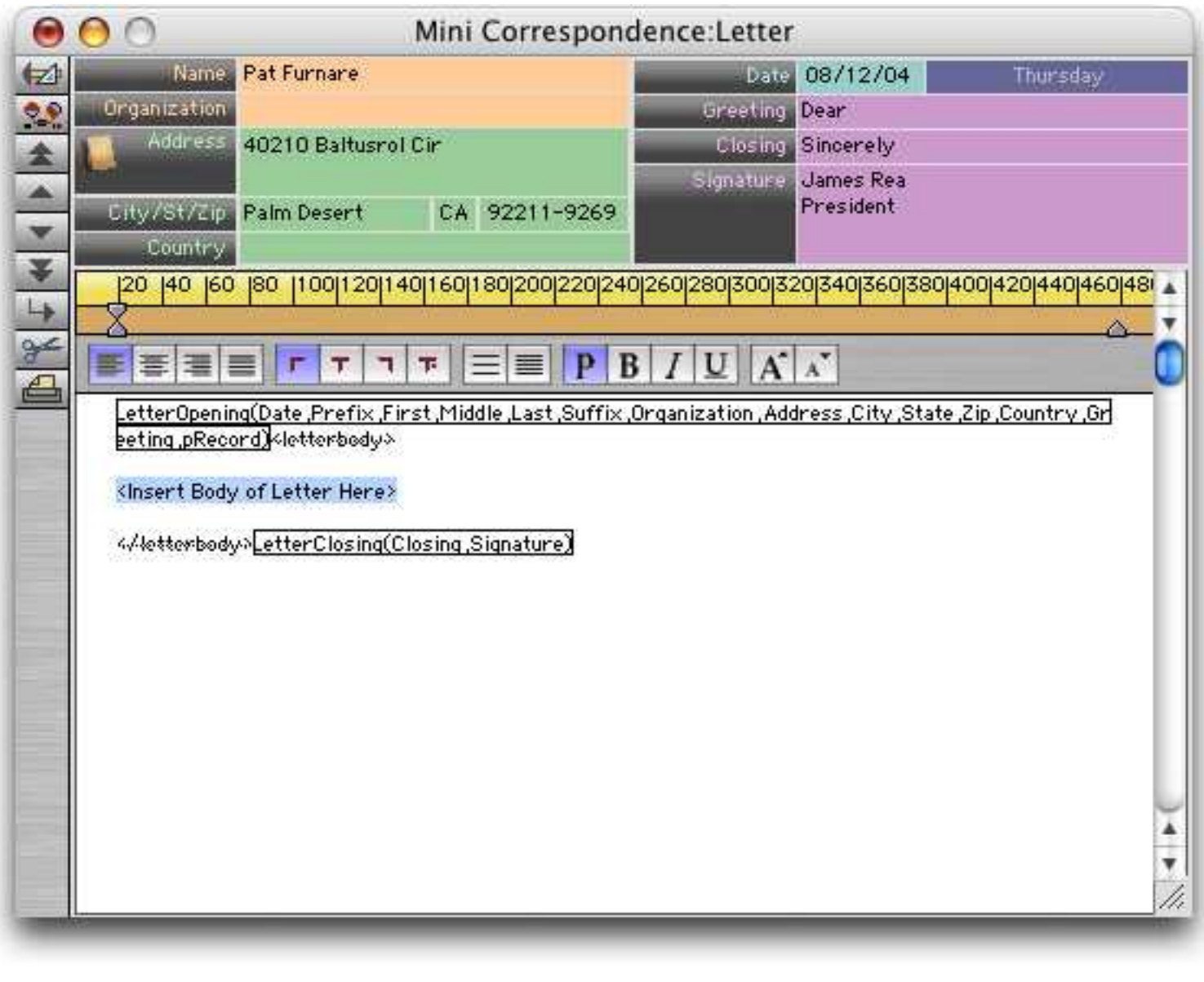
One of the most powerful features of the **Mini Correspondence** wizard is that using "Generic Fields" it can be linked with other databases that contain names and addresses (see ""Generic" Fields" on page 270). For example, suppose that you have looked up Pat Furnare using the **White Pages** wizard (see "White Pages" on page 35).



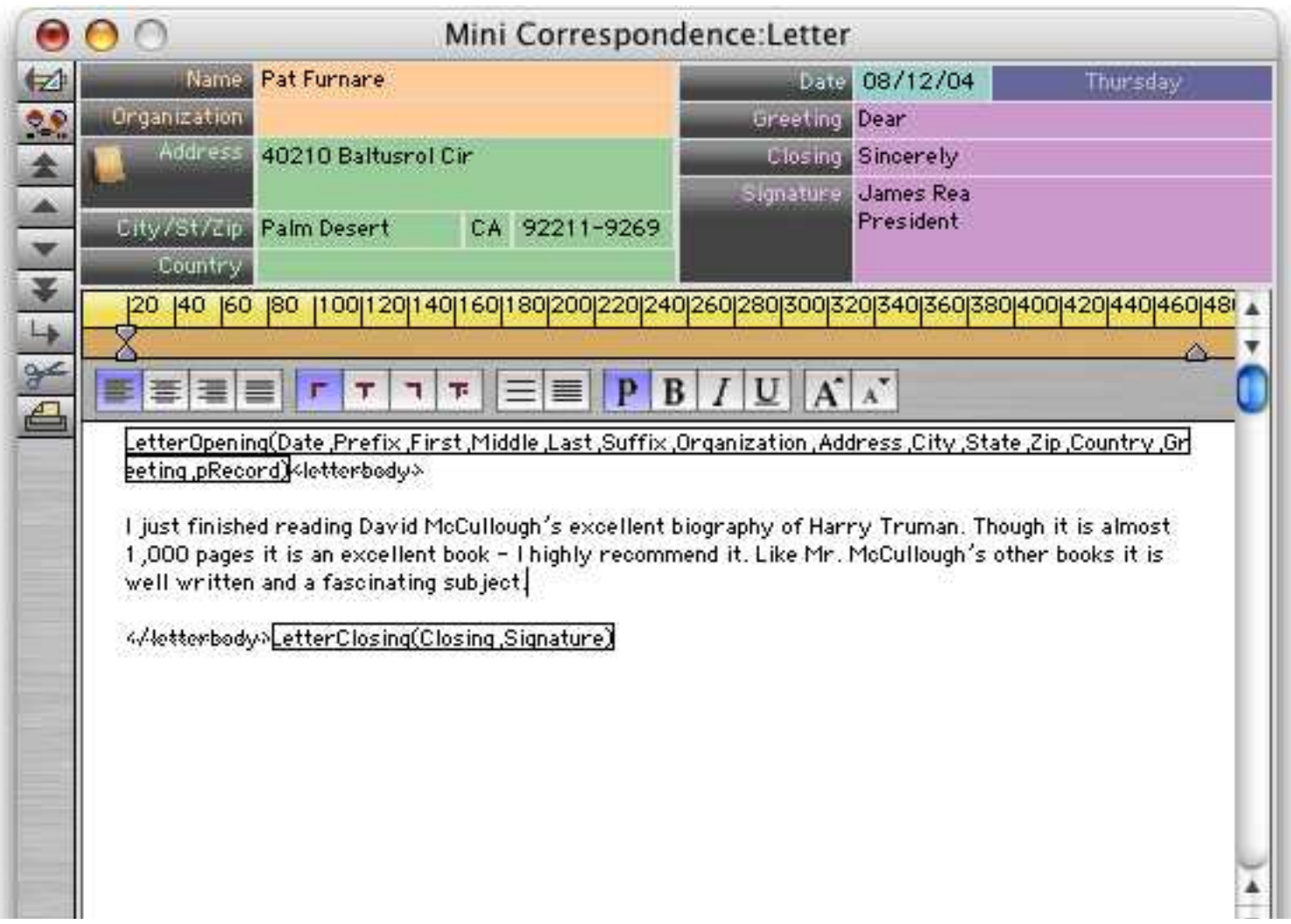
To write a letter to Pat, click on his entry in the **White Pages** window. Then choose **Mini Correspondence** from the Mini Applications submenu of the Wizards menu. The wizard will open and as you if you want to write a letter to this person.



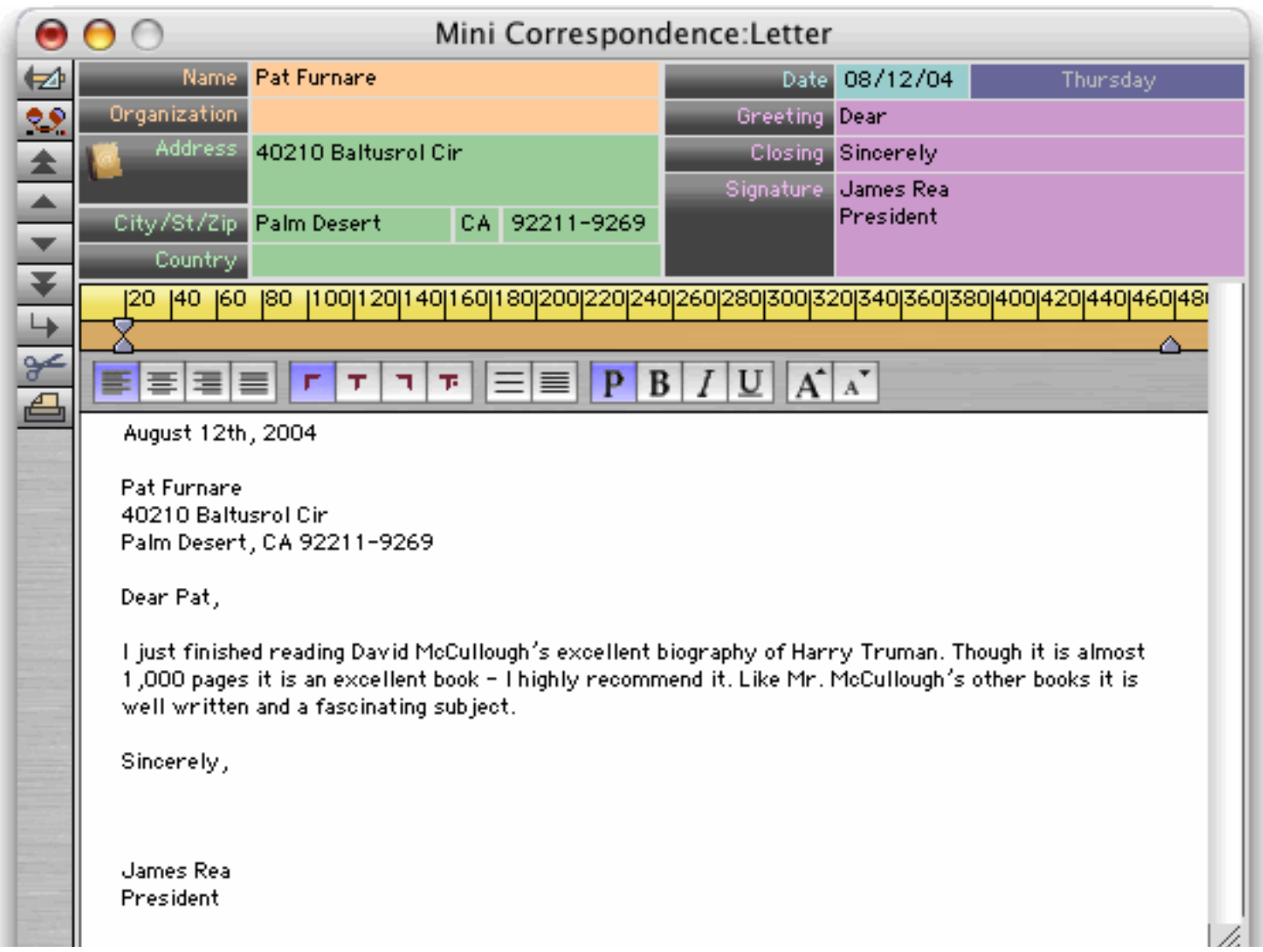
When you press **Yes**, the wizard will create a new letter with Mr. Furnare's name and address.



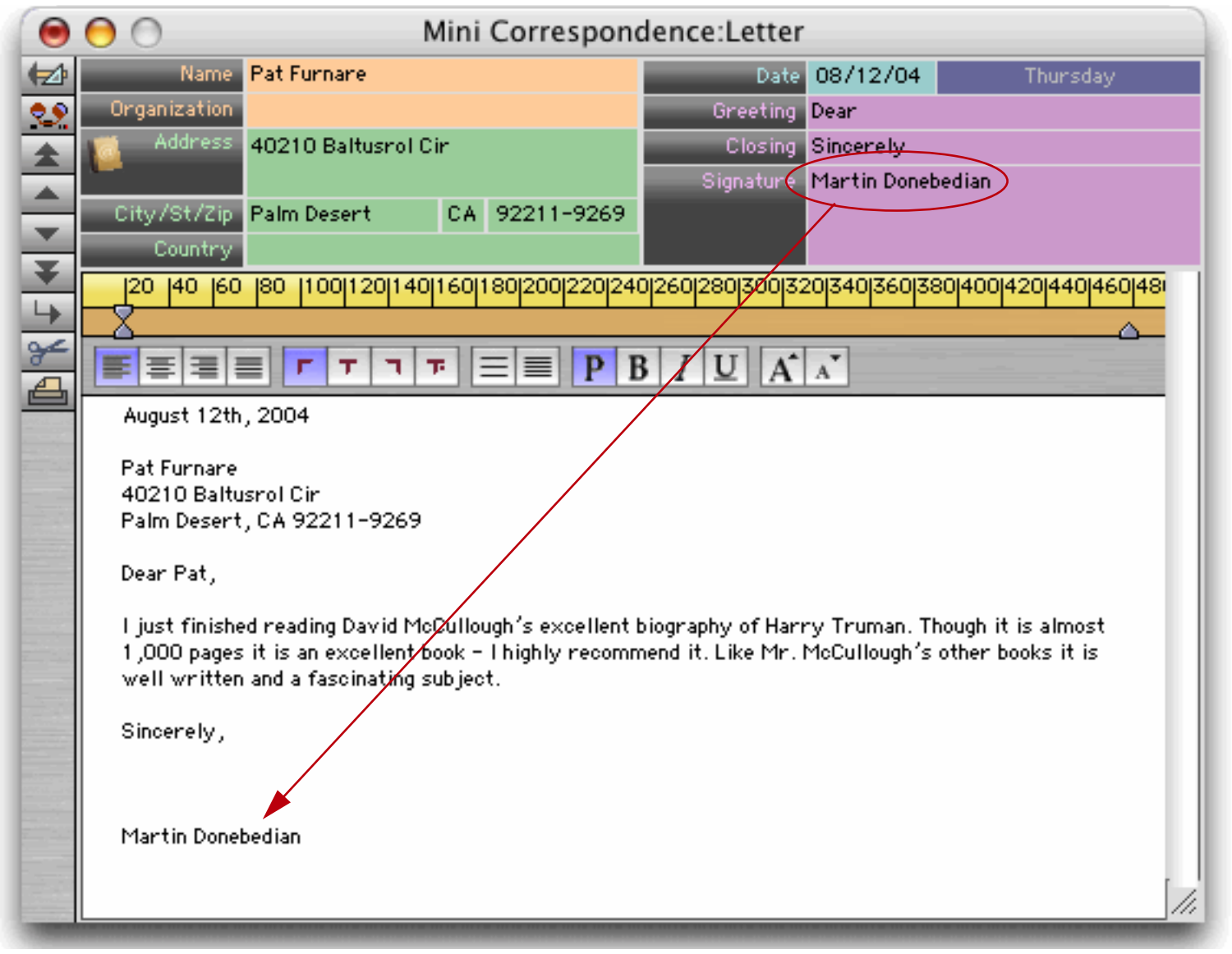
The opening and closing sections of the letter have been created with some incomprehensible looking formulas — just ignore those for now and start typing in the body of the letter.



When you press the **Enter** key you'll see the finished letter.

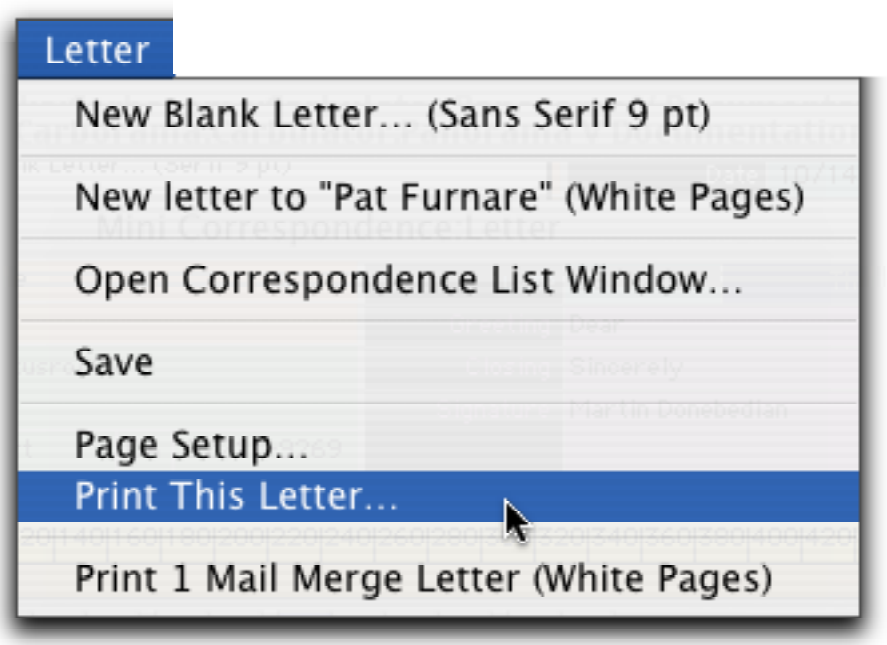


If this is the first letter you have created you'll probably want to customize the signature. When you enter the signature at the top of the window the wizard will automatically change the letter itself (in fact changing any of the fields at the top of the window will change the letter).

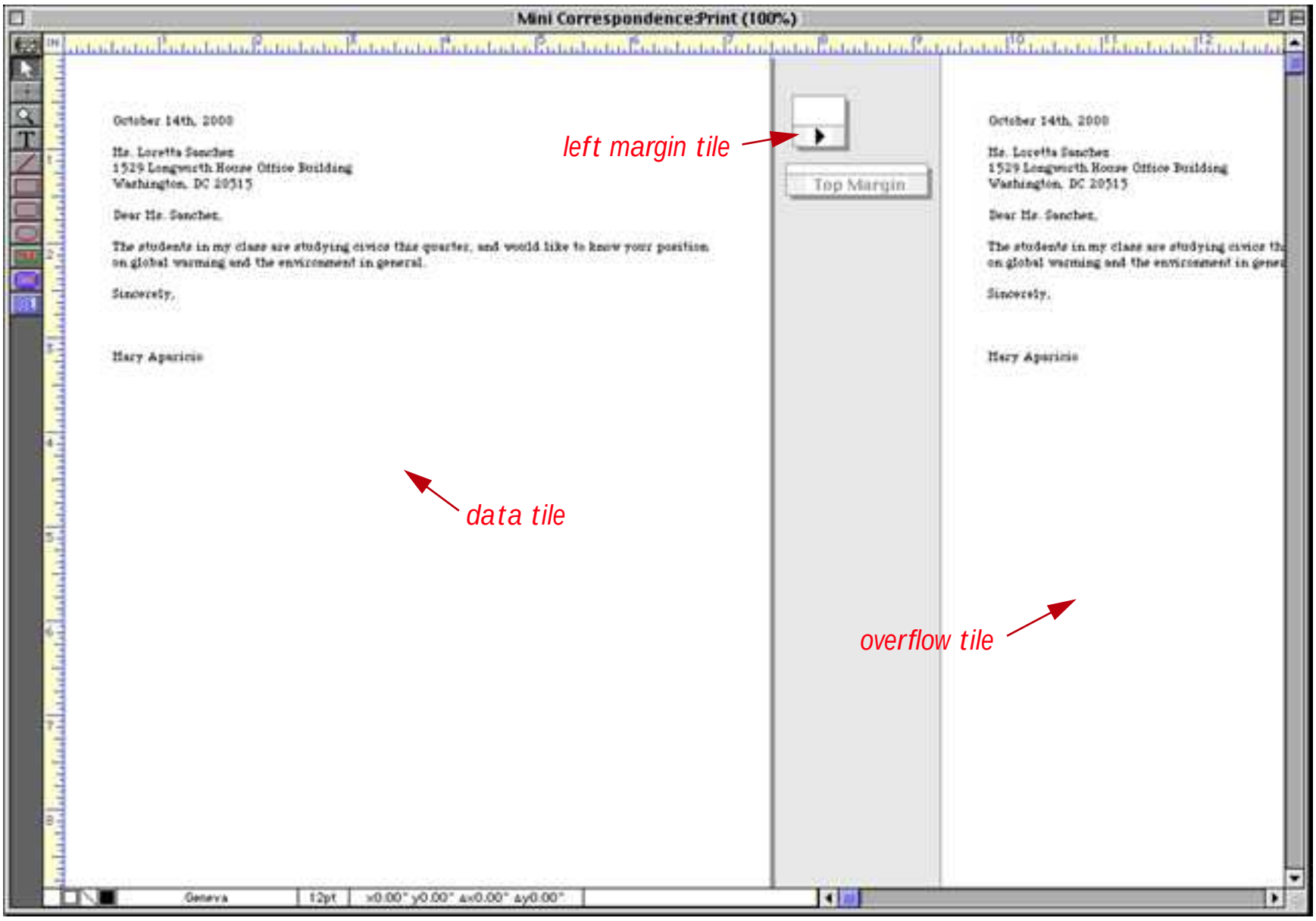


Printing a Letter

To print the letter choose **Print This Letter…** from the Letter menu. (You'll probably also want to save the letter at this point. You can use the **Save** command in the Letter menu or in the regular File menu. The **Save** command saves all of the letters in the entire database, not just the current letter.)



The wizard uses a form called Print to print the letter. You can customize this form by opening it with the **View** menu or by choosing **Edit Print Form** from the Setup menu. The Print form has a data tile and overflow tile to allow it to print multi page letters (see See "Printing Multiple Page Documents" on page 724). By editing this form you can adjust the margins and/or add your logo to the printed letter. When you are done be sure to close the form and save the database.

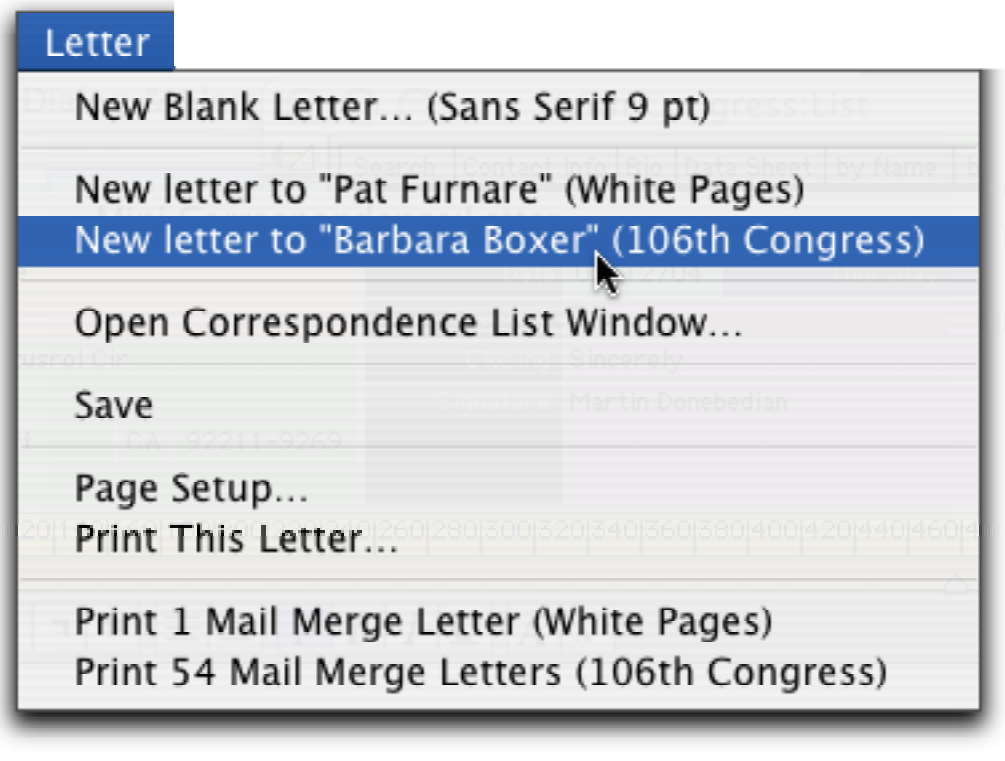


Printing a Mail Merge Letter

The wizard can also print a single letter over and over again for sending to multiple recipients — for example we could send a personalized letter to every congressional member from California. For this example we'll use the **106th Congress** database that is included in the Panorama example files. The first step is selecting the members of congress from California (see "The Find / Select Dialog" on page 369).As you can see there are 54 members in the California delegation.



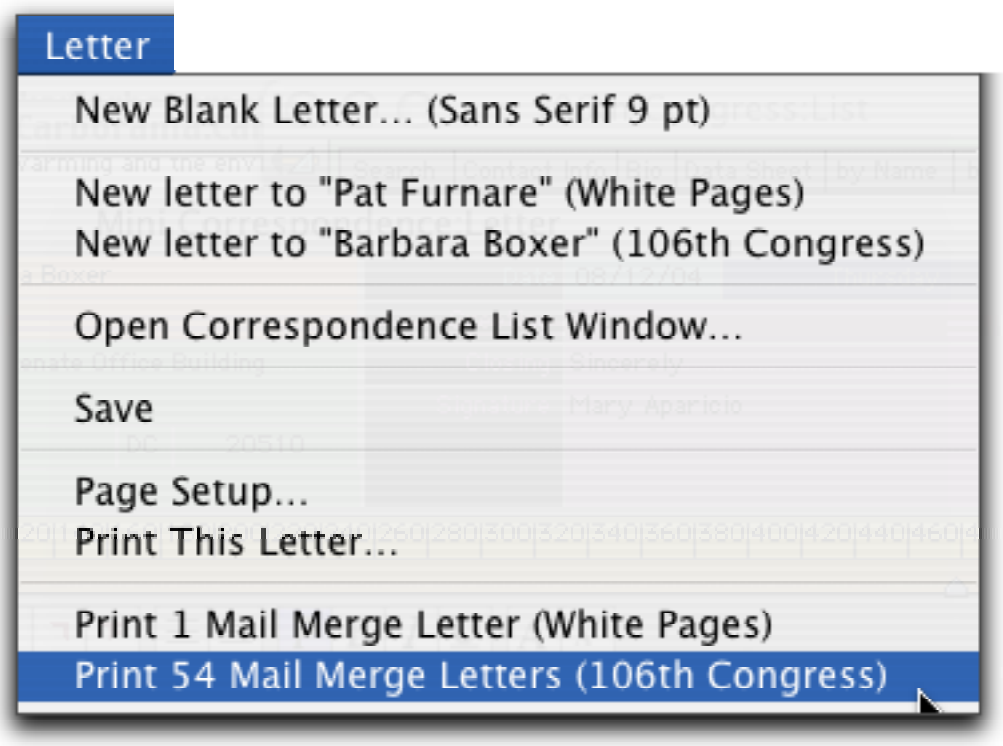
Now go back to the **Mini Correspondence** wizard, then pull down the **Letter** menu and create a new letter to Barbara Boxer.



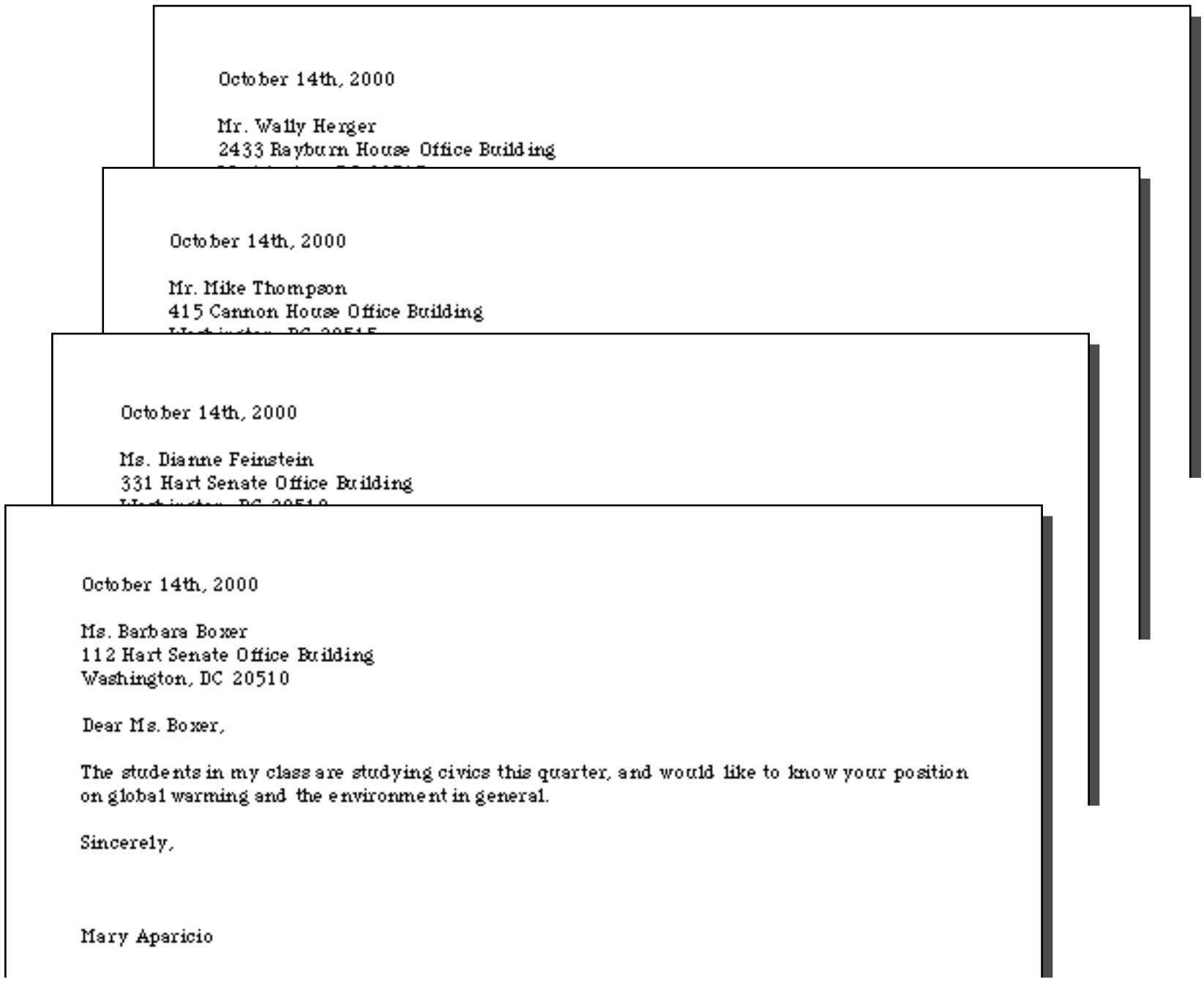
Type in the body of the letter and press **Enter**. The result should look like this:



The last item in the **Letter** menu will print a customized letter to each selected person in the 106th Congress database.



The wizard will print 54 personalized letters.



October 14th, 2000

Mr. Wally Herger
2433 Rayburn House Office Building

October 14th, 2000

Mr. Mike Thompson
415 Cannon House Office Building

October 14th, 2000

Ms. Dianne Feinstein
331 Hart Senate Office Building

October 14th, 2000

Ms. Barbara Boxer
112 Hart Senate Office Building
Washington, DC 20510

Dear Ms. Boxer,

The students in my class are studying civics this quarter, and would like to know your position on global warming and the environment in general.

Sincerely,

Mary Aparicio

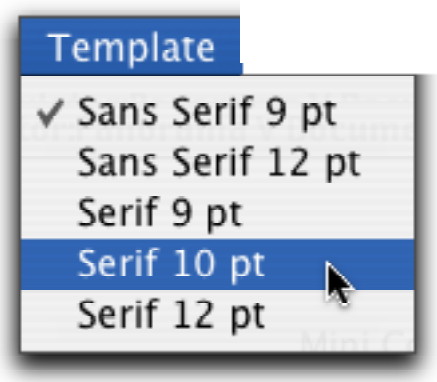
Viewing a List of Letters

The **Mini Correspondence** wizard normally displays only one letter at a time. To see a list summarizing all of your correspondence choose **Open Correspondence List Window...** from the Letter menu. This window shows a list of the letters you have created.



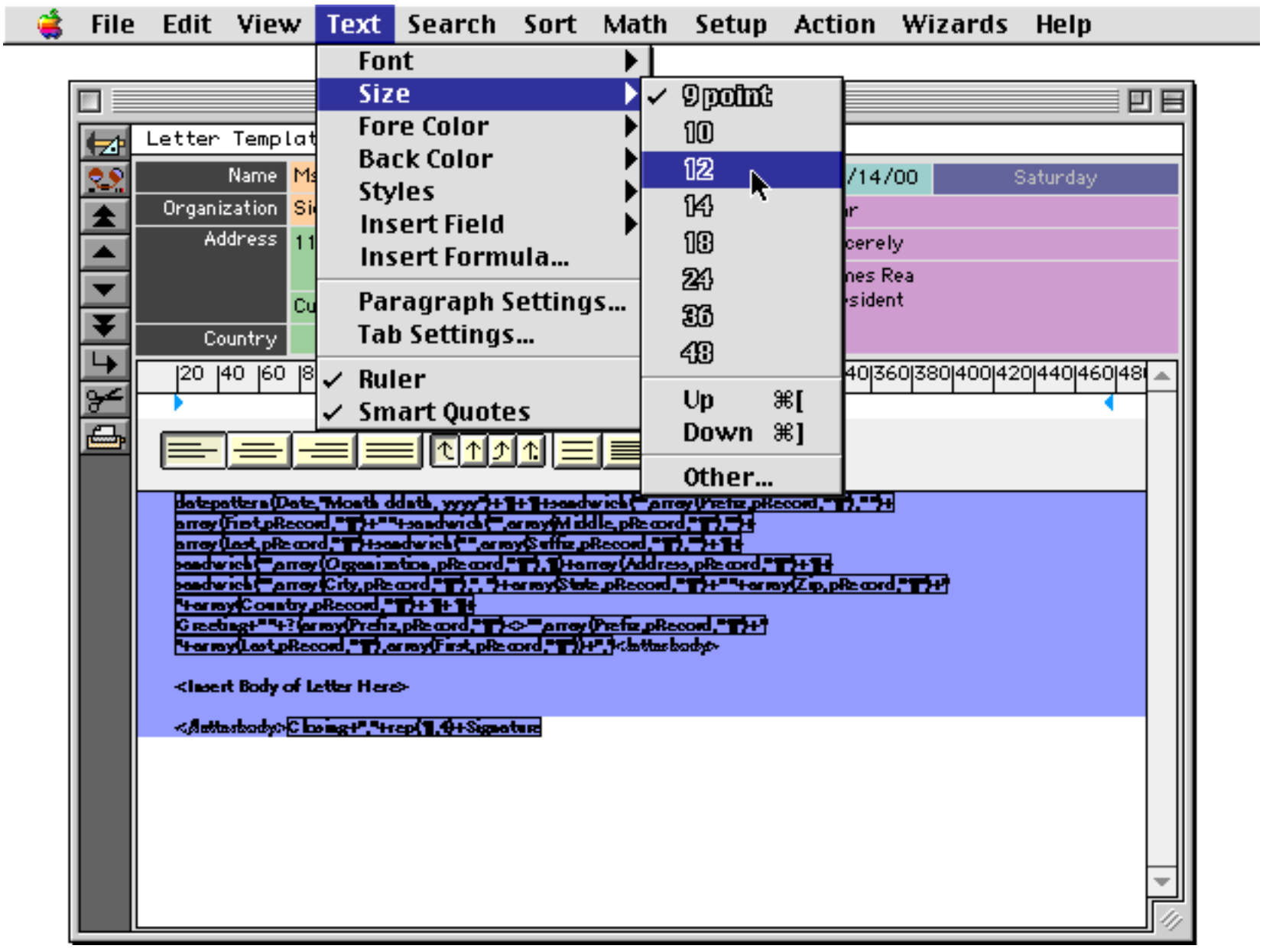
Correspondence Templates

When you create a new letter the wizard uses a template to set up the heading, closing, and font. You may choose from pre-built templates and you can also create your own templates. After you create a new letter you can use the **Template** menu to change the template used for this (and subsequent) letters.



The wizard updates the letter with the new template, which in this case uses a different font and font size. (Warning: If you change the template after you have written the letter, the body of the letter will be changed to the template body! Be sure to change the template *before* you write the letter body.)

To create your own template you should first create a new letter with one of the existing templates. Then select all of the text and adjust the font and size (see “Styles” on page 692).

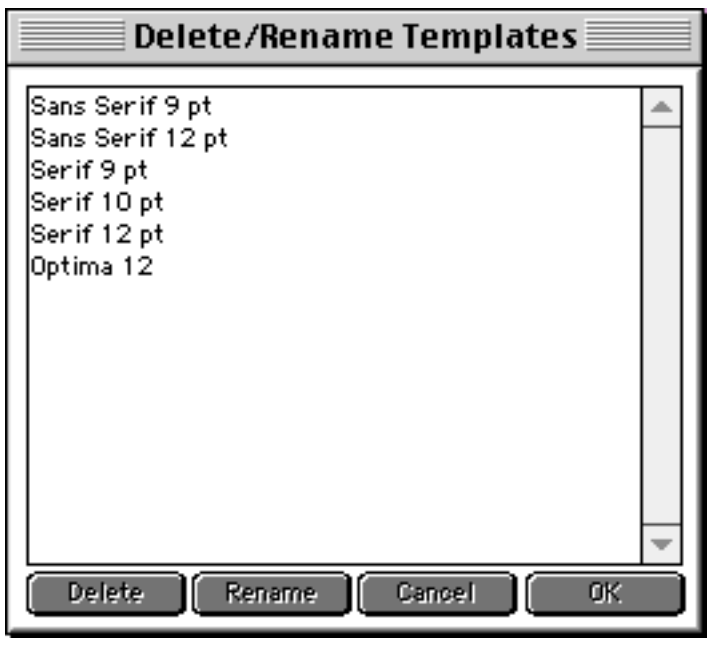


Once you have the font and size you want select the phrase <Insert Body of Letter Here>, then choose **Save Template** from the Setup menu inside the window. The wizard will prompt you to enter a name for the new template.



The new template is added to the **Template** menu.

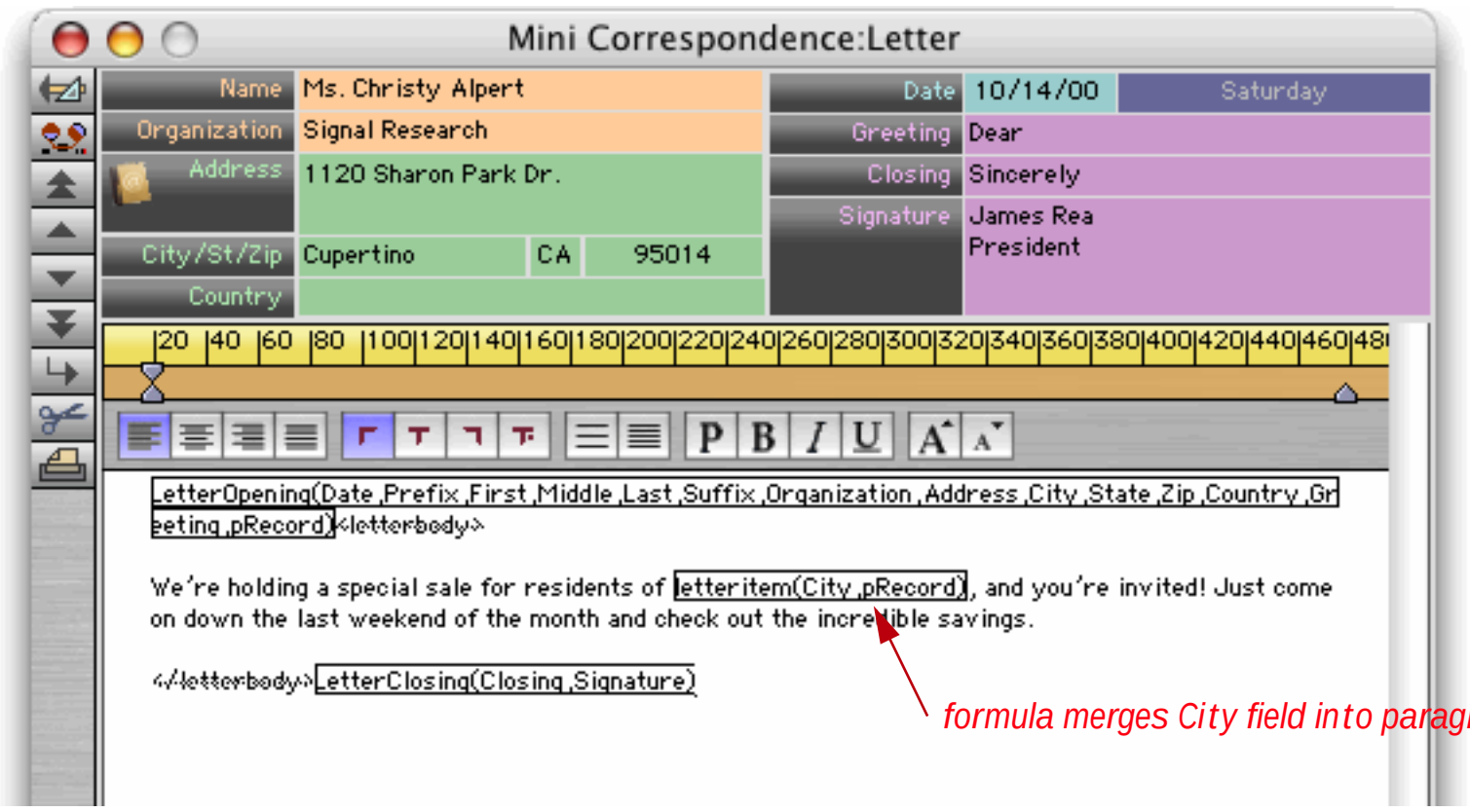
If you later decide you want to rename or delete a template use the **Delete/Rename Template...** command in the Setup menu. This command opens this dialog.



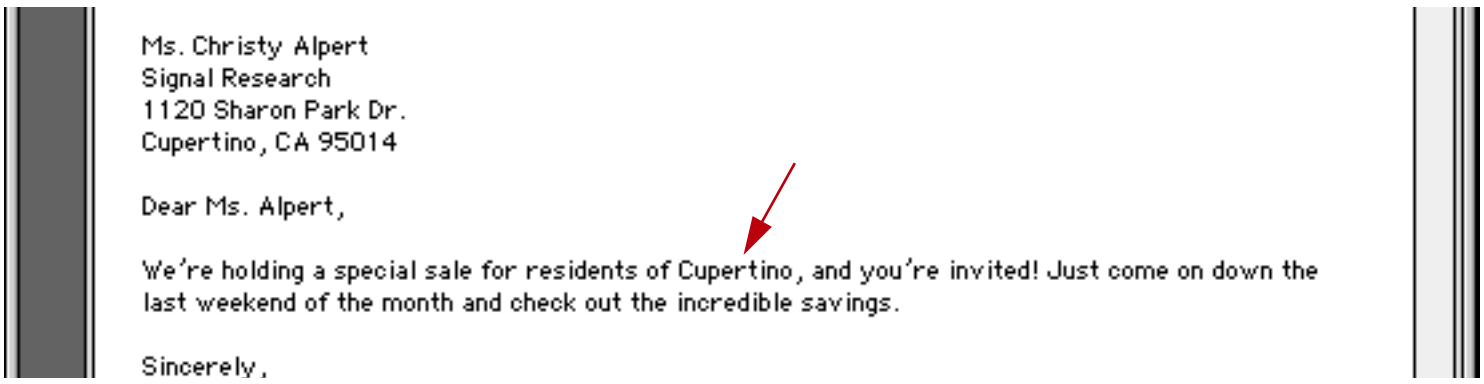
To delete a template simply select it and press the **Delete** button. To rename a template select it and press the **Rename** button. The wizard will prompt you for the new template name. When you are finished press OK to finalize all of the changes.

Merging Data into the Body of a Letter

In addition to setting the default font and text size the letter templates also contain all of the formulas needed to merge the database information into the header and closing section of the letter (see “Merging Data into Word Processing Documents” on page 707). Using the Formula Merge style you can also merge data into the body of the letter itself (see “Merging Data into Word Processing Documents” on page 707). To merge data into the body of the letter you must use the `letteritem(` function. The first parameter of this function is the name of the item you want to merge. The possible choices are: Prefix, First, Middle, Last, Suffix, Organization, Address, City, State, Zip or Country. The second parameter must be pRecord. This is an internal variable used by the wizard for printing. Once you type in this function, you must set the style to Formula Merge (use the Style menu). The example below shows how to merge the City into the body of the letter.



When you press the **Enter** key you'll see the actual data merged into the body of the letter.



This technique will work correctly even if you print multiple letters using the mail merge feature.

(Advanced Note: If you are curious about the pRecord variable it is used for printing mail merge letters. When the wizard prints a mail merge letter it temporarily saves the name and address stored in the current record. It then scans the linked database and builds an array in each of these fields. For example, the City field will contain an array that contains the city name for each person that will be receiving this letter. The wizard prints each letter using the `printonemultiple` statement (see “Printing Data in an Array” on page 1844) and it increments the pRecord variable as it does it (1, 2, 3, etc.). If only a single letter is being printed the formula still works because pRecord is set to 1 so the `array(` function simply grabs all of the text in the field.)

Chapter 16: Images & Movies



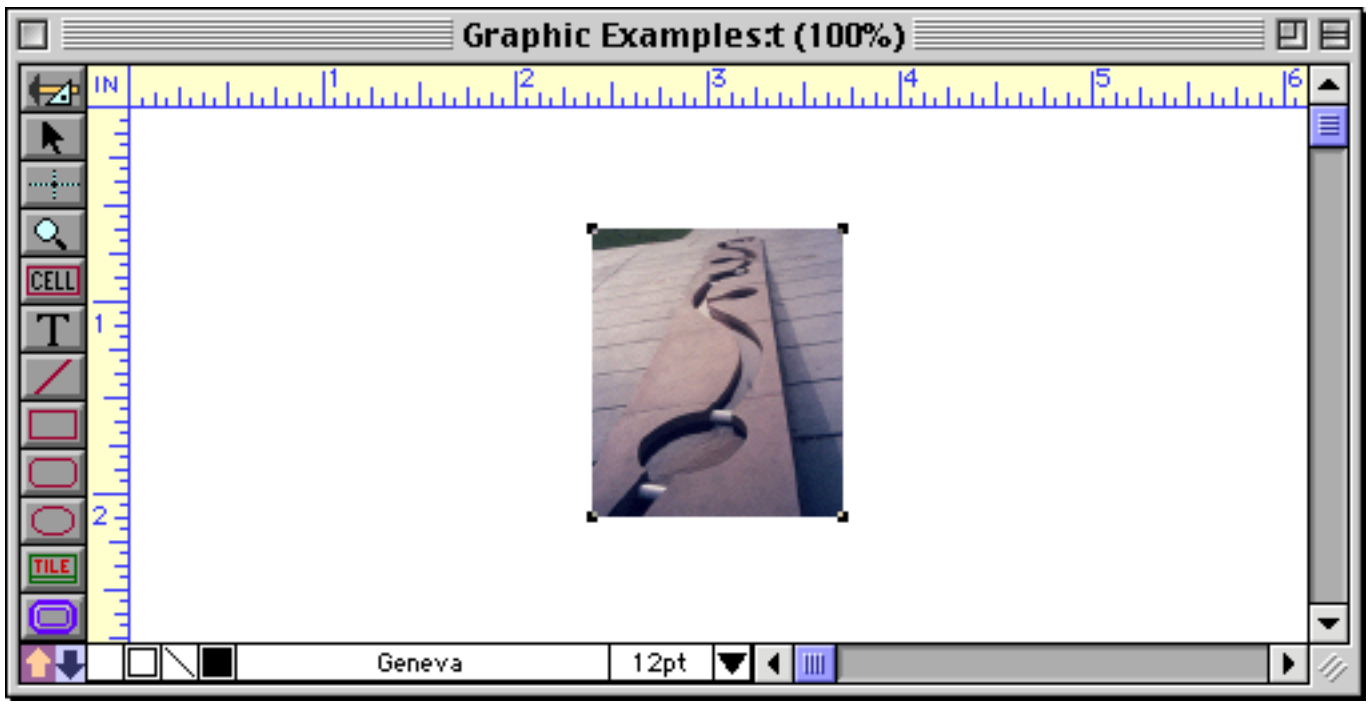
A Panorama form can display images and movies from a wide variety of sources. An image may be fixed (for example a logo or background) or variable (changing from record to record - for example personnel photos or maps associated with individual records). Variable images may be included in the database or (more commonly) displayed directly from files on the disk. Images that are displayed from disk are called **Flash Art™**. Flash Art allows any image to be displayed according to its name.

Fixed Images

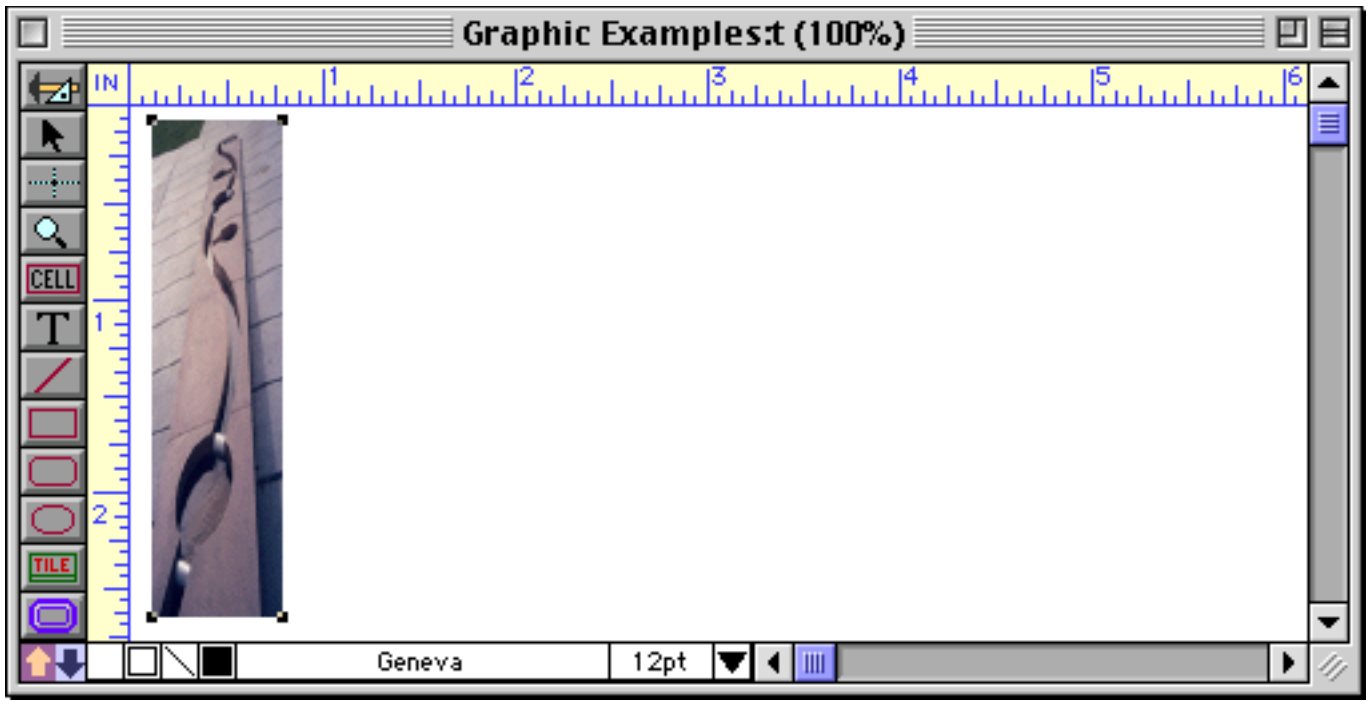
Fixed images are imported into Panorama with the **Paste** command. Before you use the **Paste** command, the graphic picture you want to import must be placed on the clipboard. Most paint and drawing programs allow you to select the graphic elements you want to use and then use the **Copy** command to place the graphics on the clipboard. This illustration shows an image being transferred to the clipboard in Adobe Photoshop.



Once the image has been transferred to the clipboard, switch to Panorama. The form you want to place the image into must be open and in Graphic Display Mode (see “Form Modes: Data Access vs. Graphic Design” on page 485). Then choose **Paste** from the Edit menu to import the image.

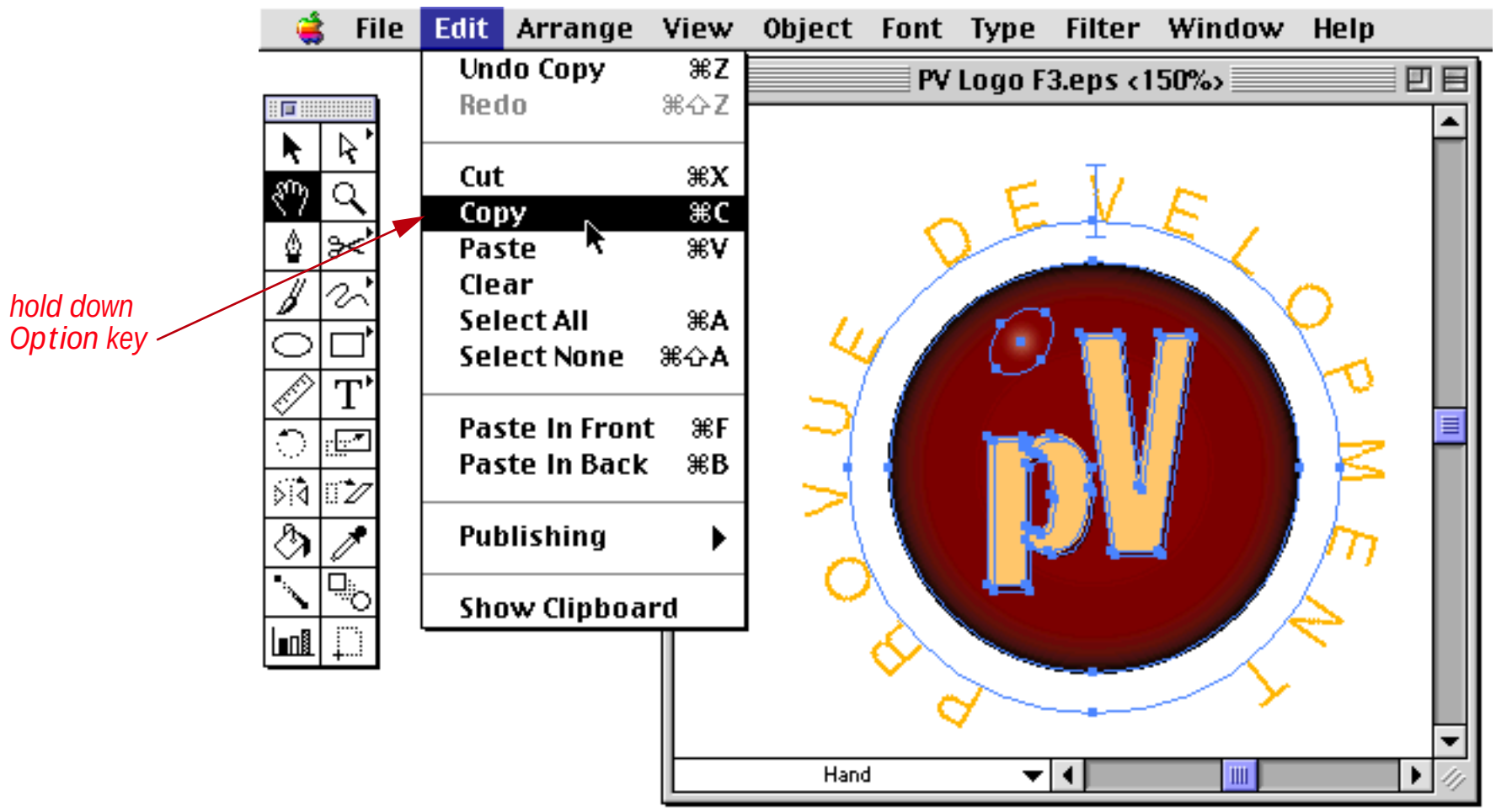


The image will be imported into the center of the window. Like any other object, you can move the picture by dragging it or adjust the size by dragging the handles on the corners (or with the **Dimensions** or **Scale** dialogs).

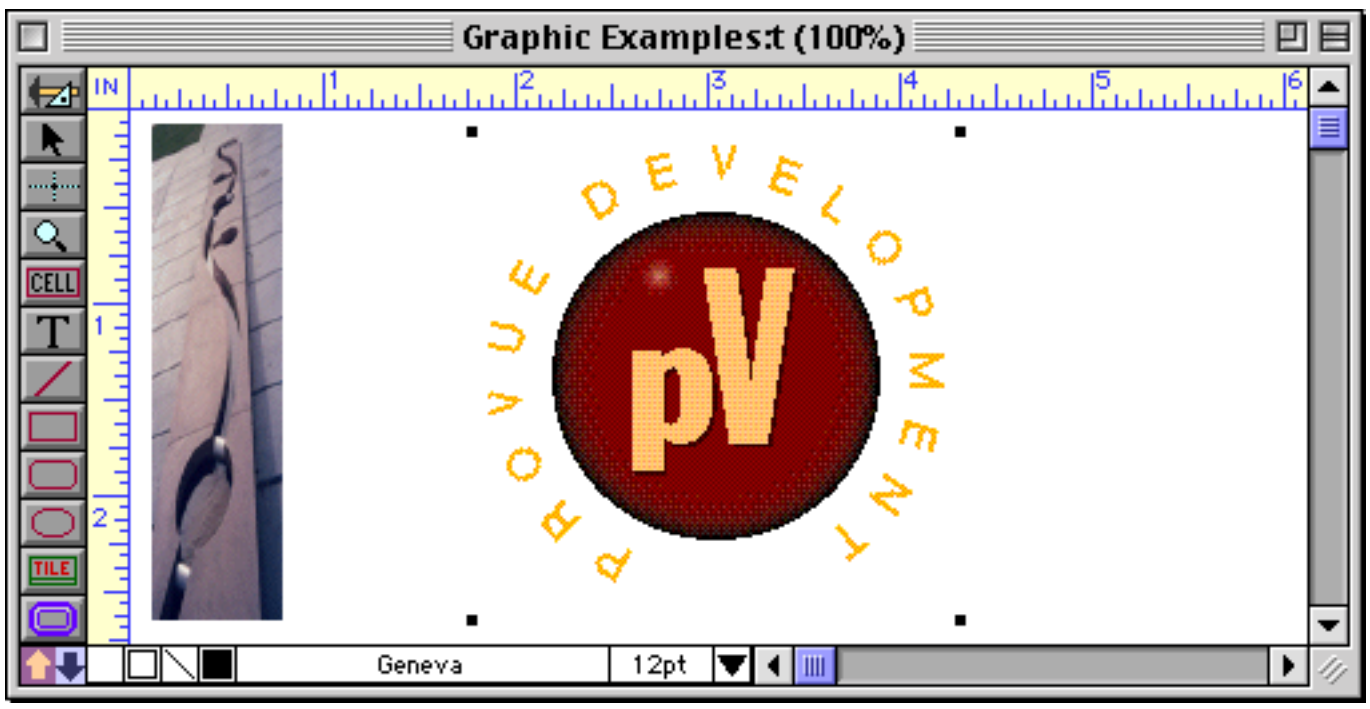


Displaying and Printing EPS Images

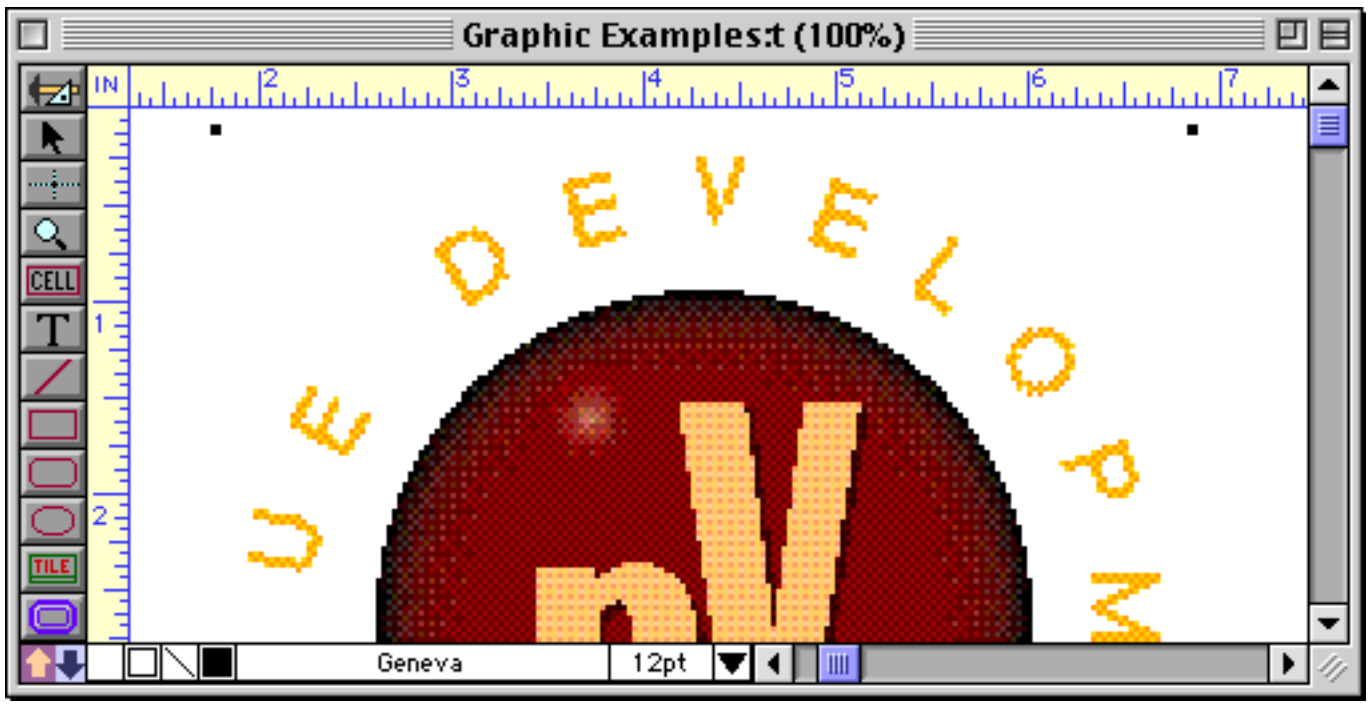
On Macintosh systems you can place EPS images on the clipboard when using Adobe Illustrator, Macromedia Freehand and many other programs. To place an EPS image on the clipboard, hold down the **Option** key while you use the **Copy** command.



Once the EPS image has been placed on the clipboard you can switch to Panorama and **Paste** it into your form, just like any other image.



You may notice that the image appears a bit “grainy.” You’re actually seeing a bitmap image that was created by Illustrator when you placed the image on the clipboard. If you enlarge the image it will get even more “jaggy.”



Although the image appears jaggy on the screen, it won’t look jaggy if you print the form on a PostScript™ printer. When printing, Panorama uses the EPS and ignores the jaggy bitmap so you get a nice clean image on the paper.

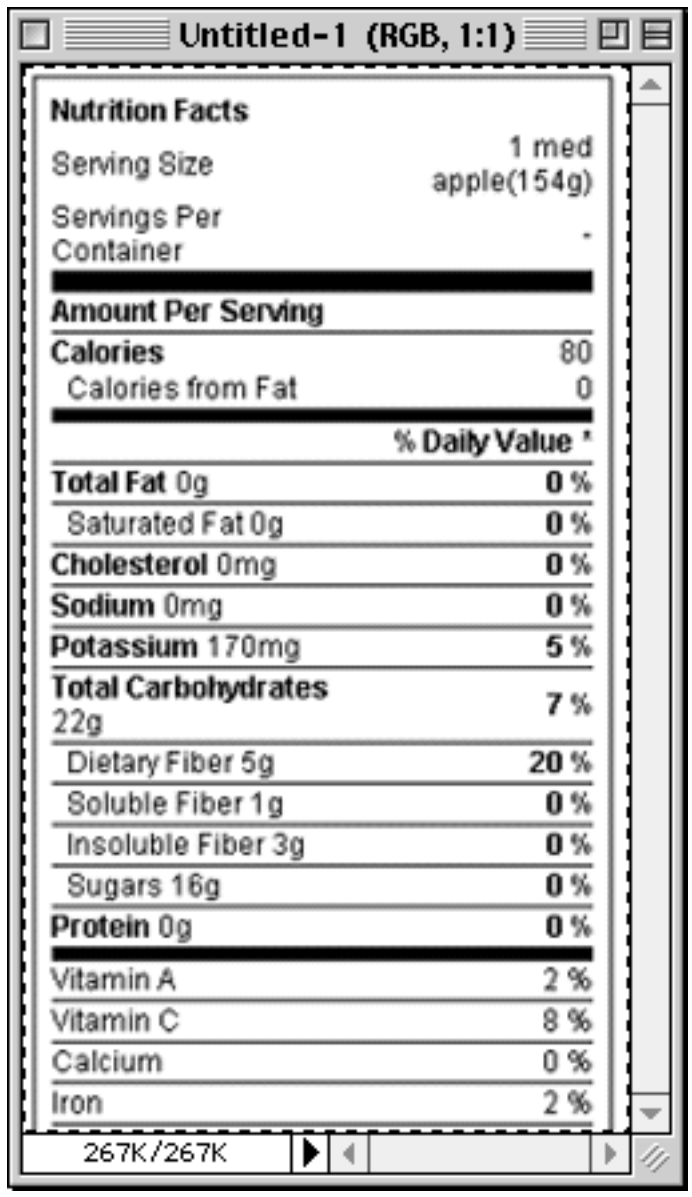
Memory Requirements for Large Images

Pictures can be real memory hogs. A large image may need up to several megabytes of memory. On Mac OS X and PC based systems you don’t have to worry about this. On MacOS 9 based systems, however, you may need to adjust Panorama’s **Scratch Memory** setting. Panorama uses scratch memory as temporary memory for many tasks, including loading fonts, displaying images, sorting, printing, etc. Panorama normally reserves 650,000 bytes (650K) for **scratch memory**. If you try to display or print very large pictures, Panorama will warn you that scratch memory is full. If that happens, you will have to increase the scratch memory size. See “Changing Scratch Memory Size (Macintosh OS 9)” on page 192 to learn how to do this.

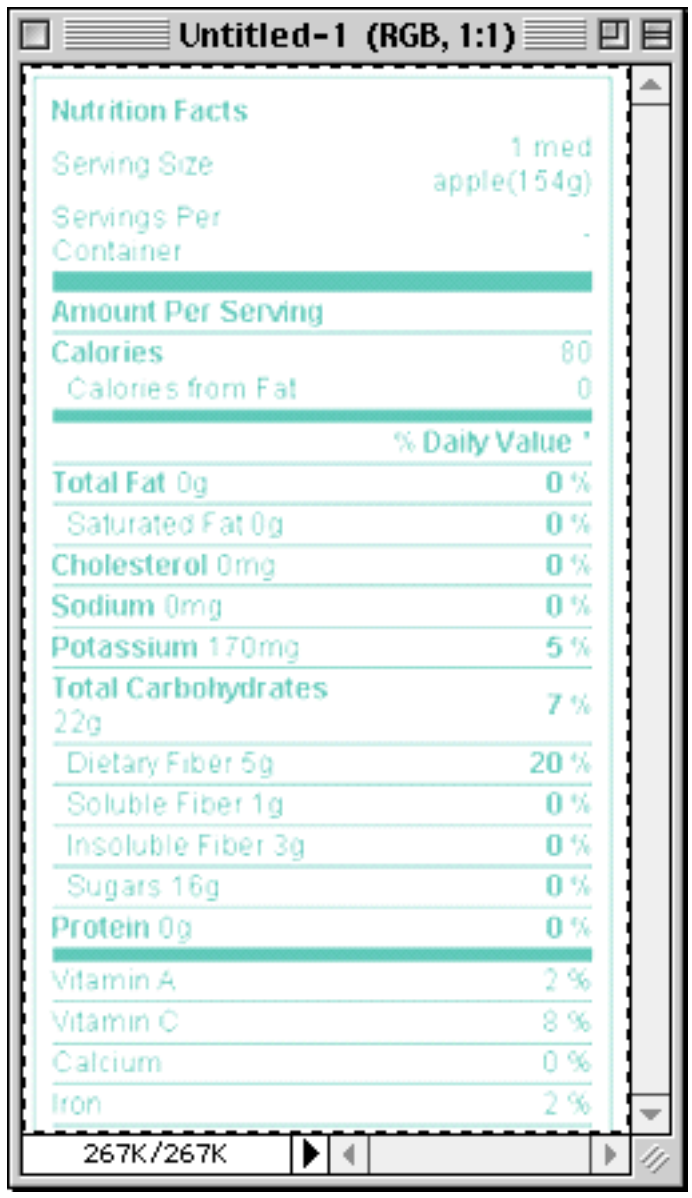
How much memory do you need to allocate for scratch memory? That depends on the images you want to display. For uncompressed images (PICT, TIFF, etc.) you simply need to assign slightly more memory than the largest image you want to display. For compressed images (JPEG) you’ll need to allocate enough memory for an uncompressed version of the image. It’s impossible to know from the file size exactly how much memory you’ll need, but a good rule of thumb would be to allocate 5 to 10 times the size of the largest image.

Tracing a Scanned Form

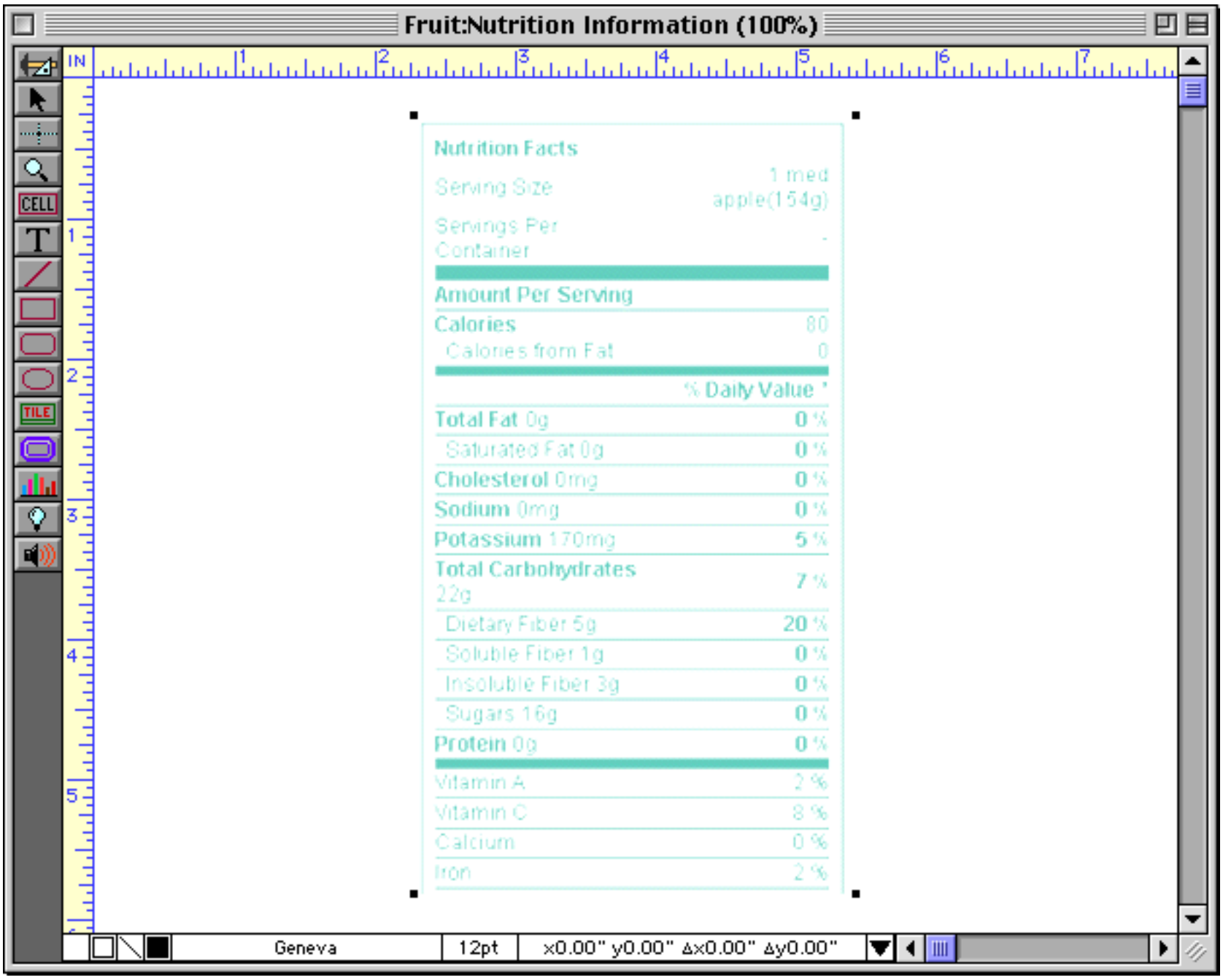
You can use a scanner to digitize a form and then paste the image into Panorama. The scanned image can either be used as is, or you can use it as a template to create a form using Panorama graphic objects. (If you trace the image using Panorama's box and line tools, the form will print faster and with better quality.) Start by scanning the form you want to duplicate. For example, we've scanned our Nutrition Information form into Adobe PhotoShop.



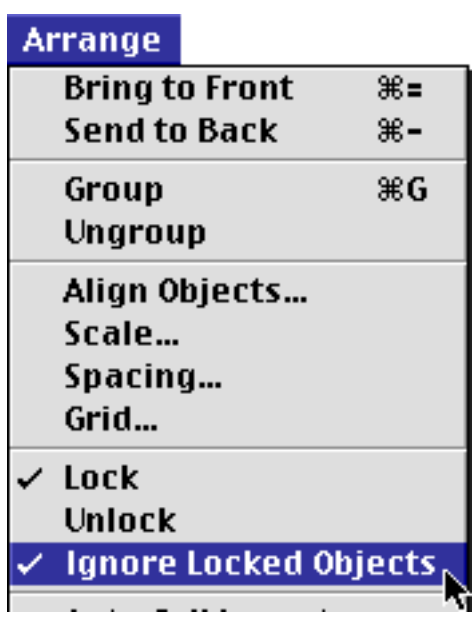
To make it easier to use this form as a template for tracing, we'll increase the brightness, reduce the contrast, and change the color. See the documentation for your photo editing software to learn how to do this.



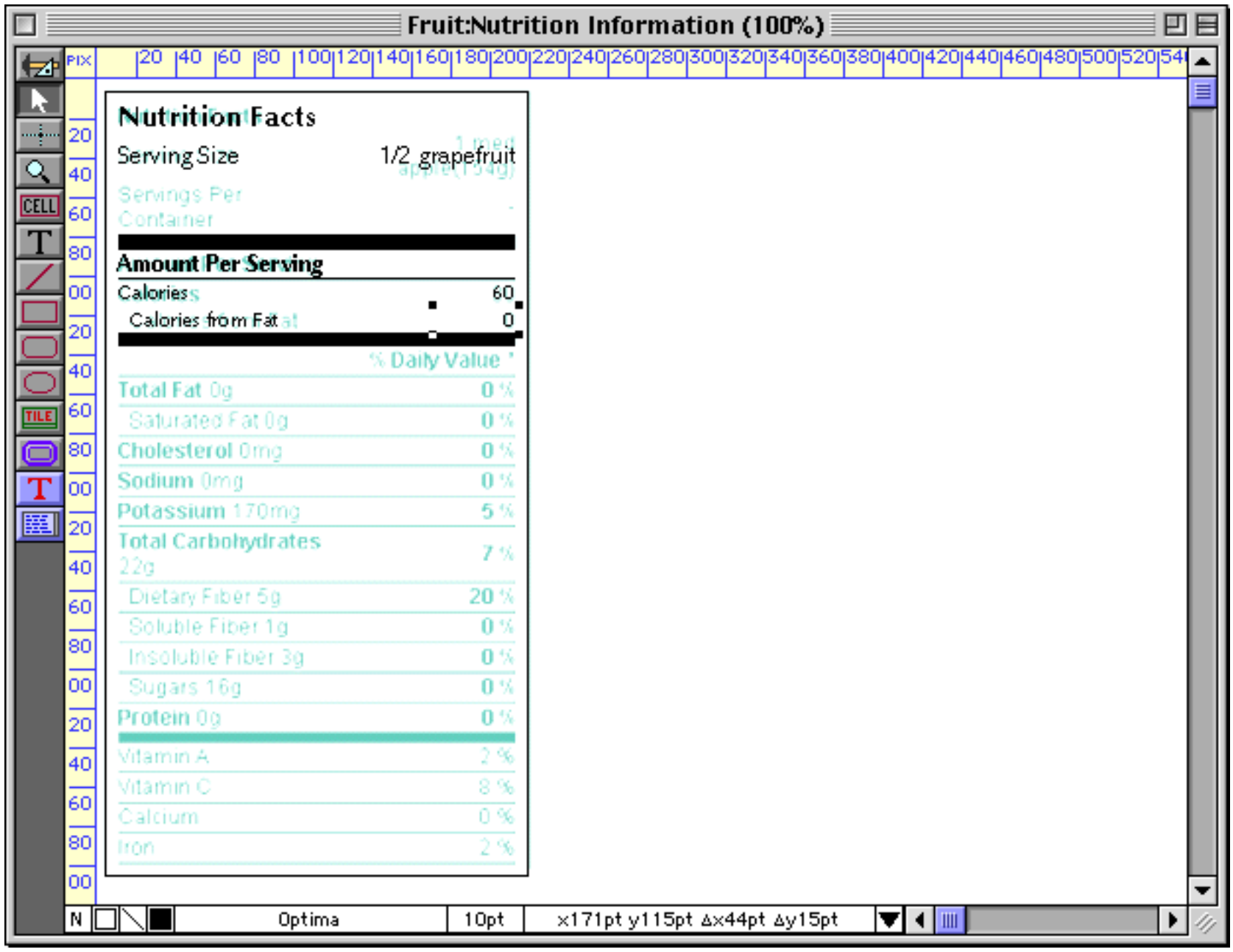
Now the image can be copied into your Panorama form. Simply **Copy** the image in your photo editing application, then switch to Panorama and **Paste** into the form.



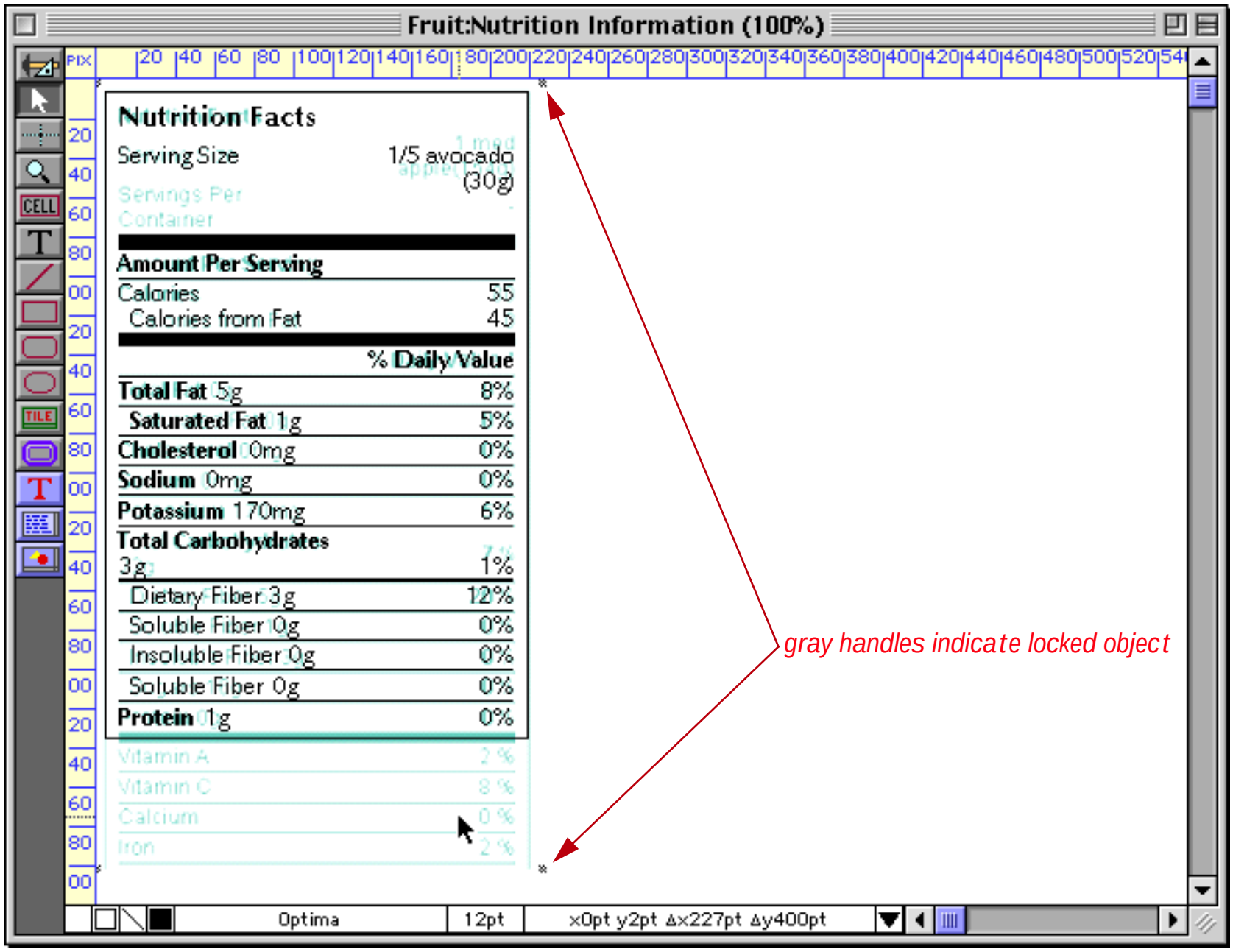
Move the template into position, then freeze the template with the **Lock** command in the Arrange Menu (see "Locked Objects" on page 575). Once the image is locked, check the **Ignore Locked Objects** menu option (see "Ignoring Locked Objects" on page 577). This option allows you to work on top of the scanned image without having to worry about accidentally clicking on and dragging the scanned image.



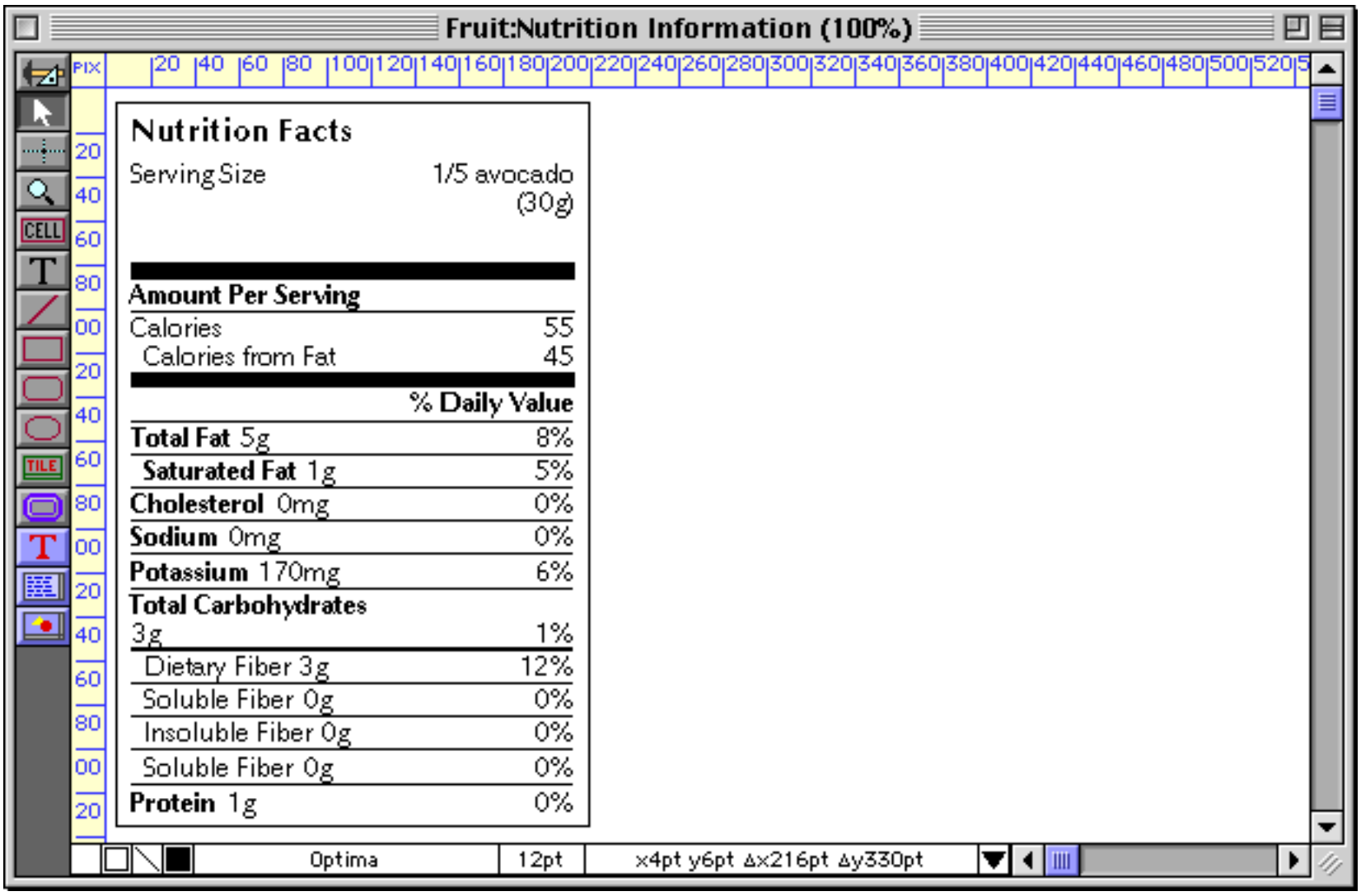
Now you can begin to trace over the scanned template to create Panorama objects like rectangles and text objects.



When the tracing is complete, the final step is to delete the scanned image you used as a template. Start by disabling the **Ignore Locked Objects** option. Then click on the scanned image.



Once the object is selected, choose the **Unlock** command to unlock the image (see "Locked Objects" on page 575). Once the object is unlocked you can use the **Cut** or **Clear** commands to delete the image from your form, leaving only the new objects you have created.



Flash Art™

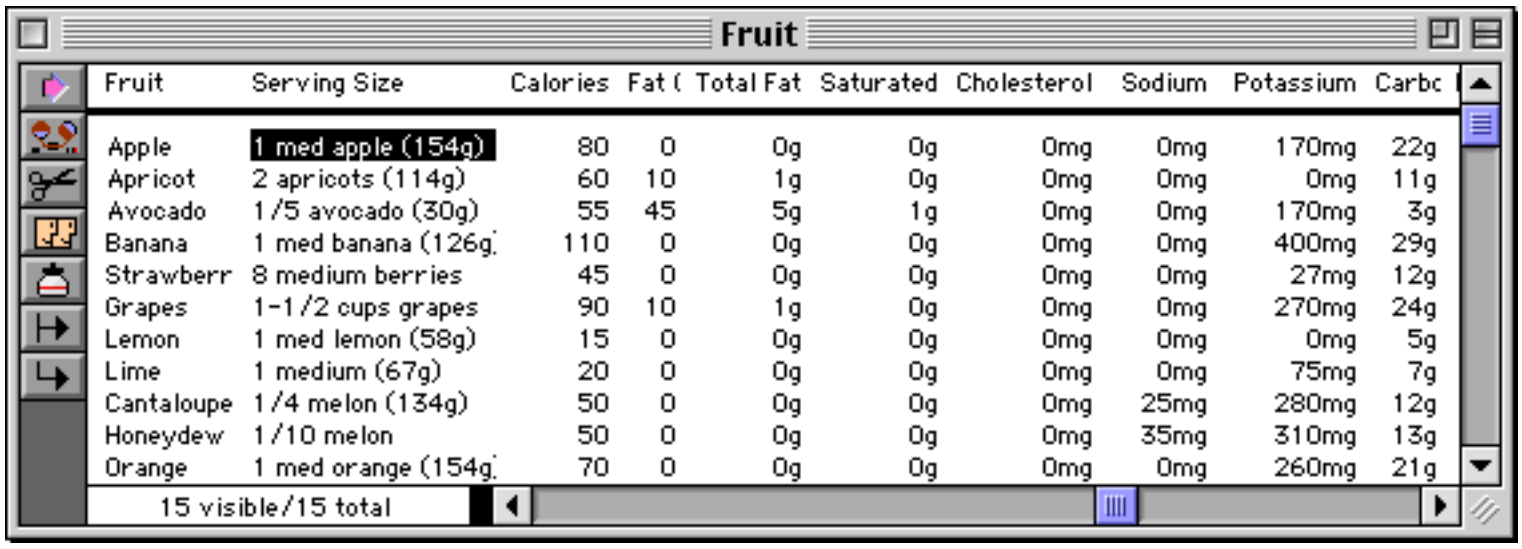
Panorama employs a technique called **Flash Art™** to display non-fixed images. Using Flash Art you can display images that are not fixed but change depending on circumstances—for example photos of each person in a personnel database, maps in a contact database or product photos in a catalog database.

When Flash Art displays an image, it does so by name. The actual image may be stored in the database itself (in the Flash Art Gallery, see "The Flash Art Scrapbook (Gallery)" on page 762), or in separate disk files (see "Displaying Images Directly From Disk Files" on page 767). You set up a formula that controls what image to display. When the formula matches the name of an image, that image is displayed.

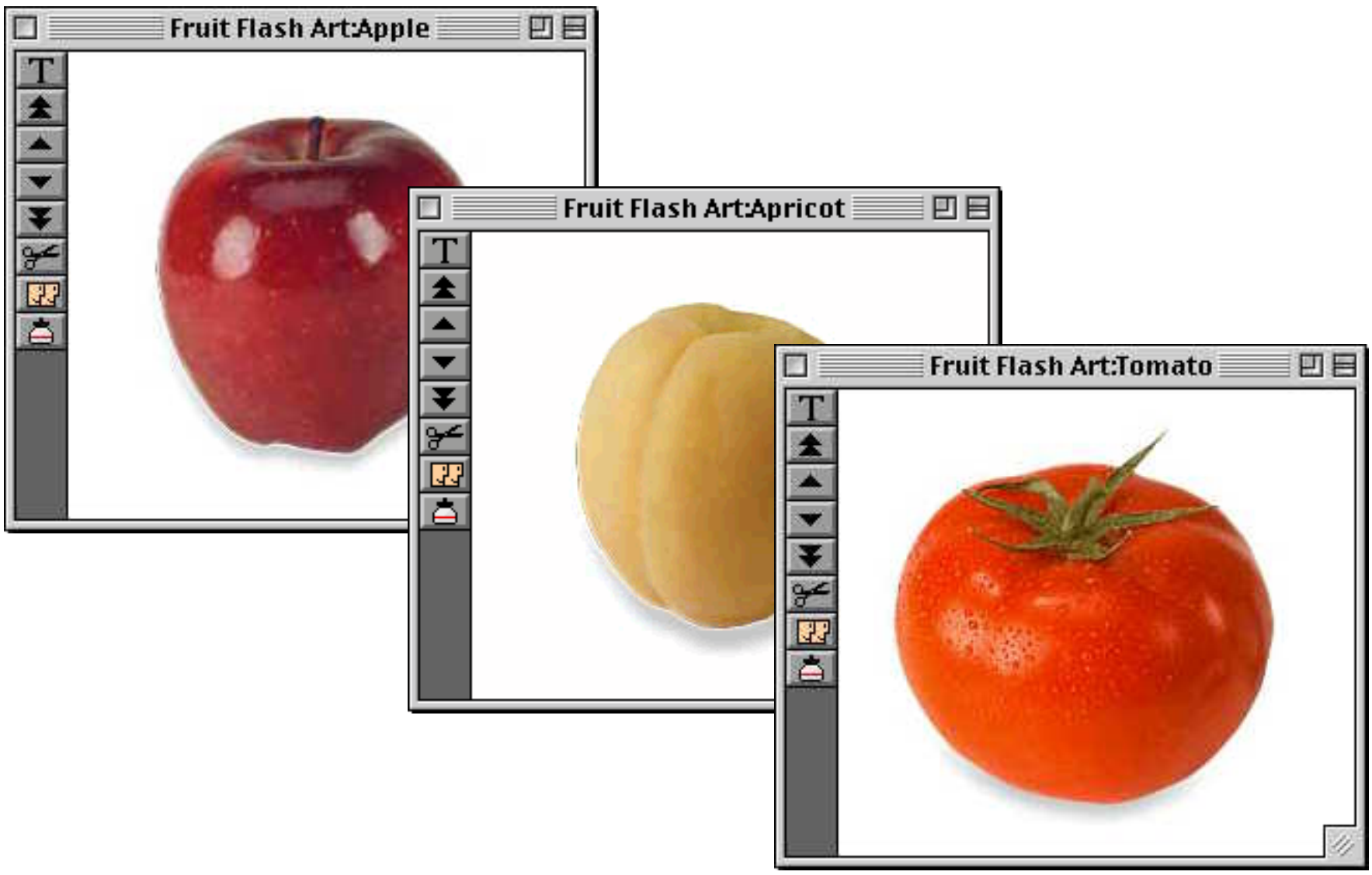
Panorama actually has several different types of objects for displaying images by name. The standard Flash Art object (see ""Classic" Flash Art Objects" on page 806) has been standard from the earliest versions of Panorama, and is retained for compatibility with older databases. The newer Super Flash Art object (see "Creating Super Flash Art Objects" on page 749) is more customizable (more alignment and scaling options, scroll bars, etc.) and should usually be used for new applications. In addition, the Flash Art Push Button (see "Flash Art™ Push Button SuperObjects™" on page 823) and Flash Art Data Button (see "Flash Art Data Button SuperObjects™" on page 842) allow you to create custom buttons from any image.

Creating Super Flash Art Objects

To illustrate the creation of a Super Flash Art object within a form we'll use a database of nutritional information for fruits. This database contains about a dozen different fruits, including the name of the fruit and the FDA nutritional information.

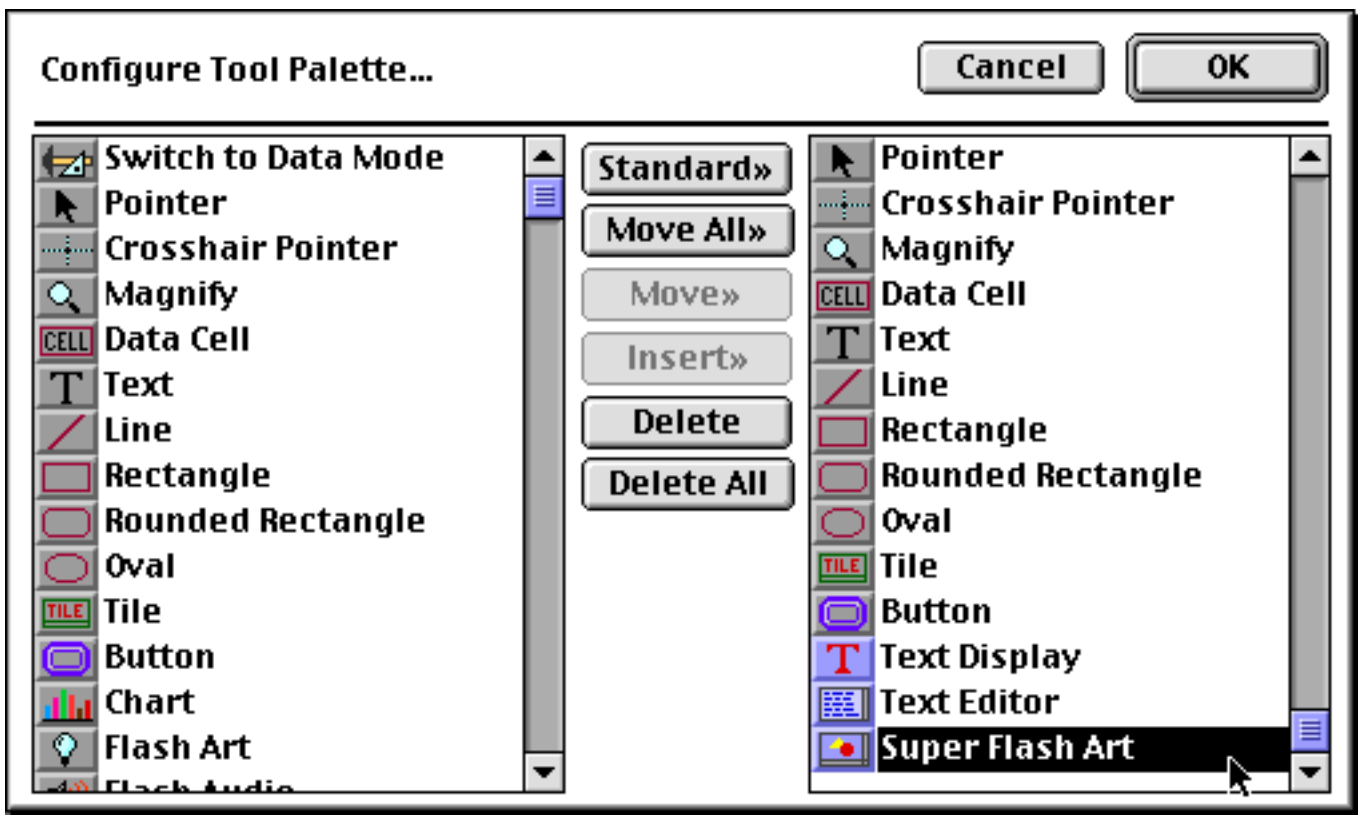


We've also prepared a collection of photographs of fruit. Each photograph has a name, and the photo name matches the name of the fruit in the database, from Apple to Tomato.

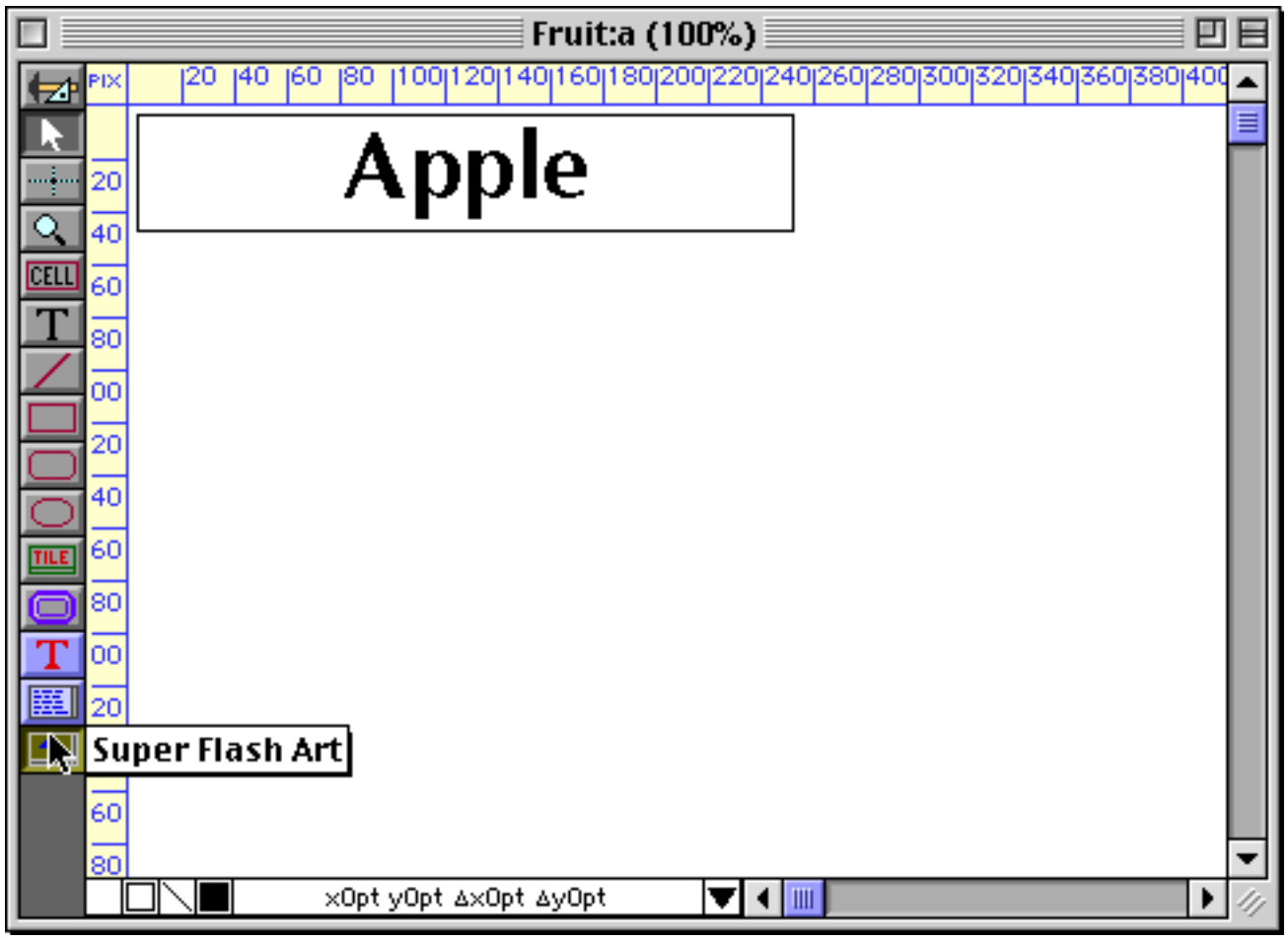


In this case the images are stored in the database in the Flash Art Gallery (see "The Flash Art Scrapbook (Gallery)" on page 762), but they could also be stored in files on the disk (see "Displaying Images Directly From Disk Files" on page 767). For now, don't worry about how the images are set up, let's just continue to see how the images can be used in a form.

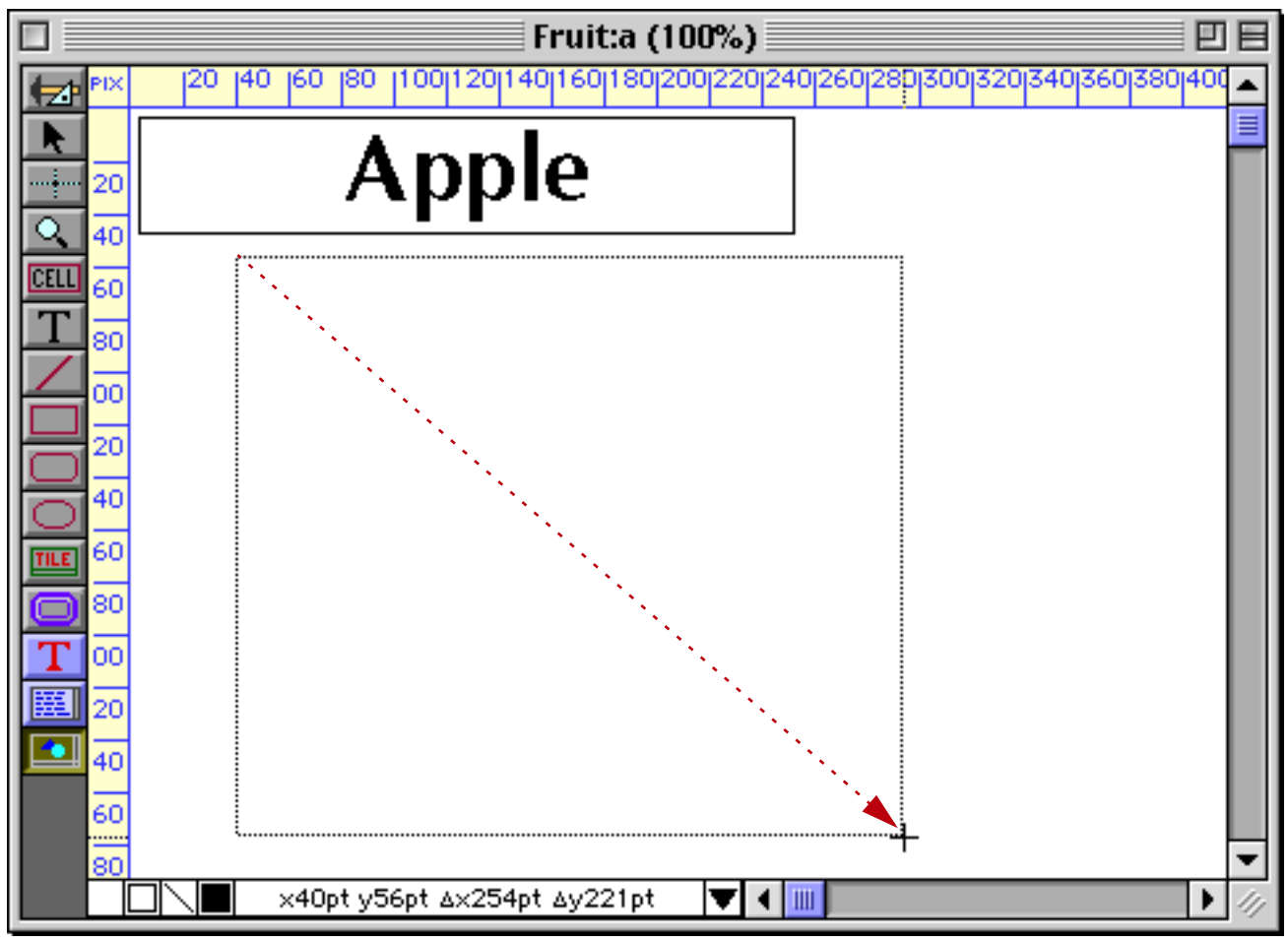
The Super Flash Art tool is not in the default tool palette, so you'll need to use the **Tool Palette** dialog to add this tool to the palette if it is not already there (see "Customizing the Tool Palette" on page 497).



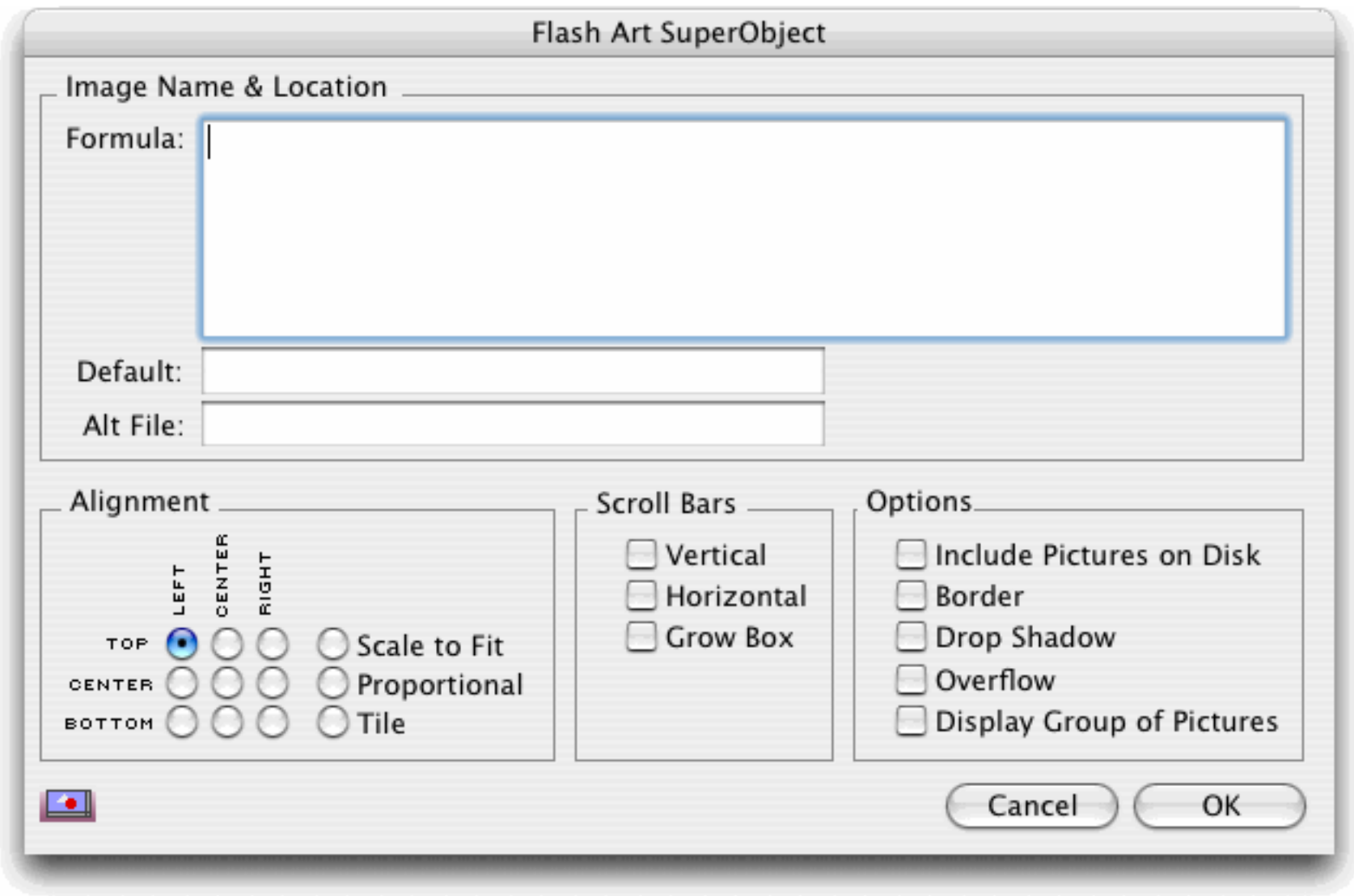
Now that the tool is added to the palette you can select it. Notice that we've set up a Text Editor SuperObject (see "Text Editor SuperObject" on page 639) to allow the name of the fruit to be displayed and edited.



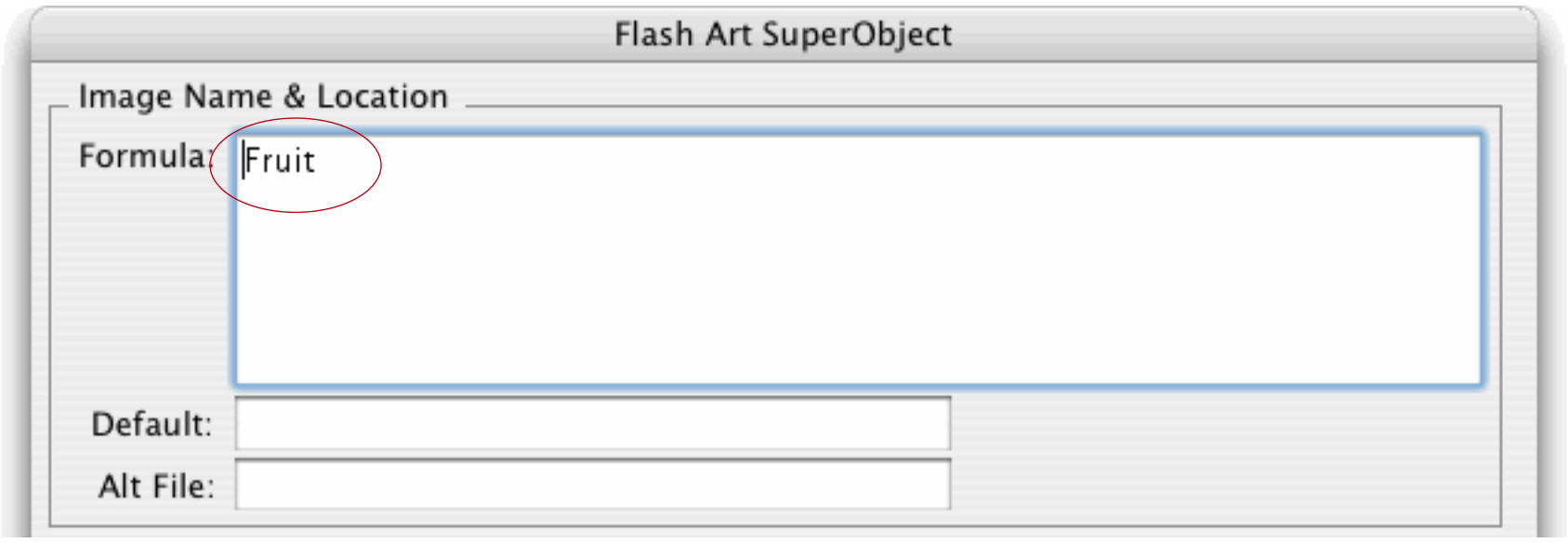
Once the tool is selected, drag the mouse across the form in the location where you want to create the Super Flash Art object.



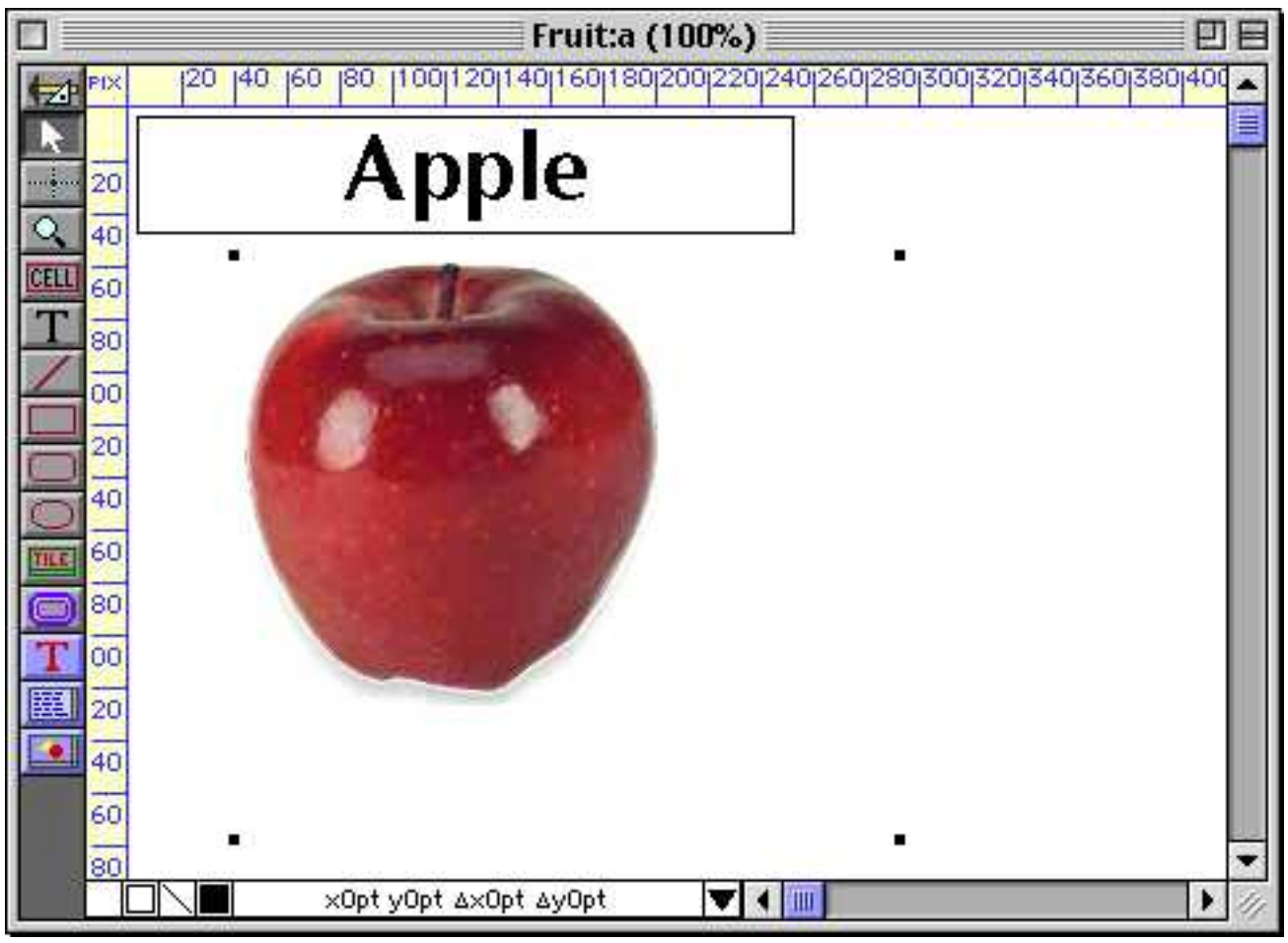
When you release the mouse, the Super Flash Art configuration dialog will appear.



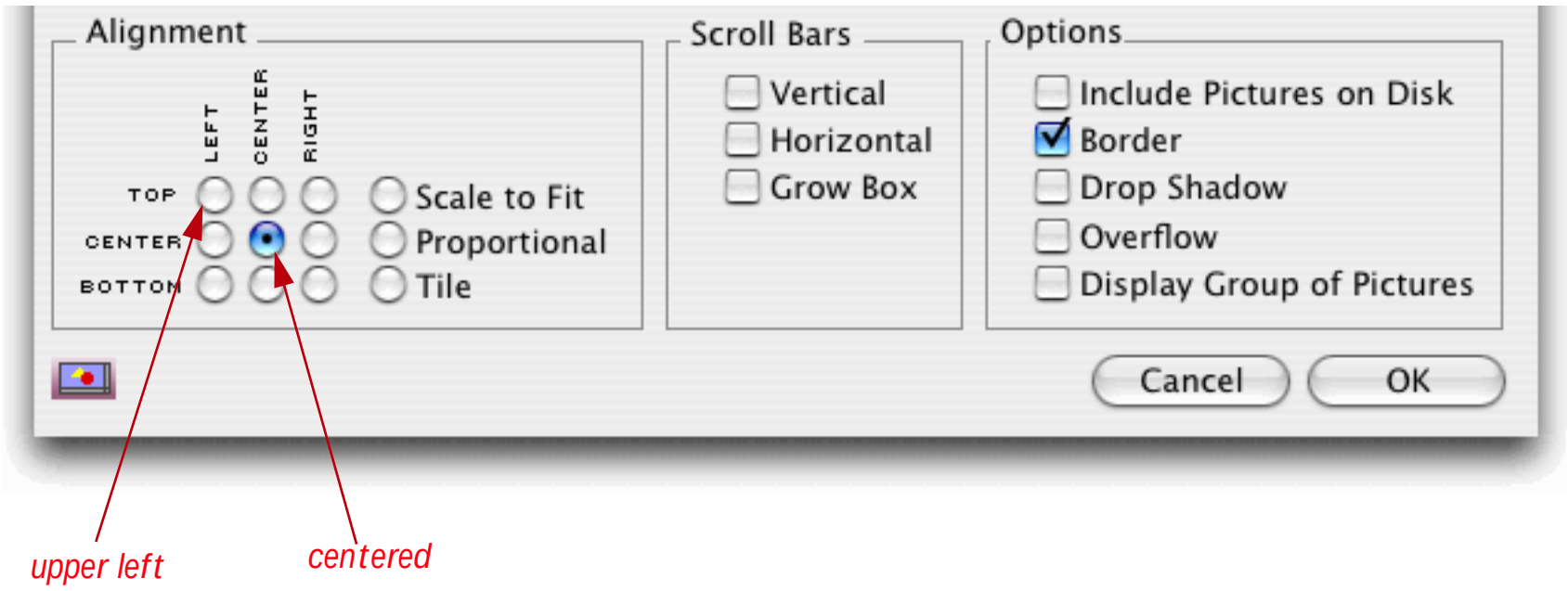
At a minimum you must enter a valid formula into the dialog. In this case the formula is simply `Fruit`.



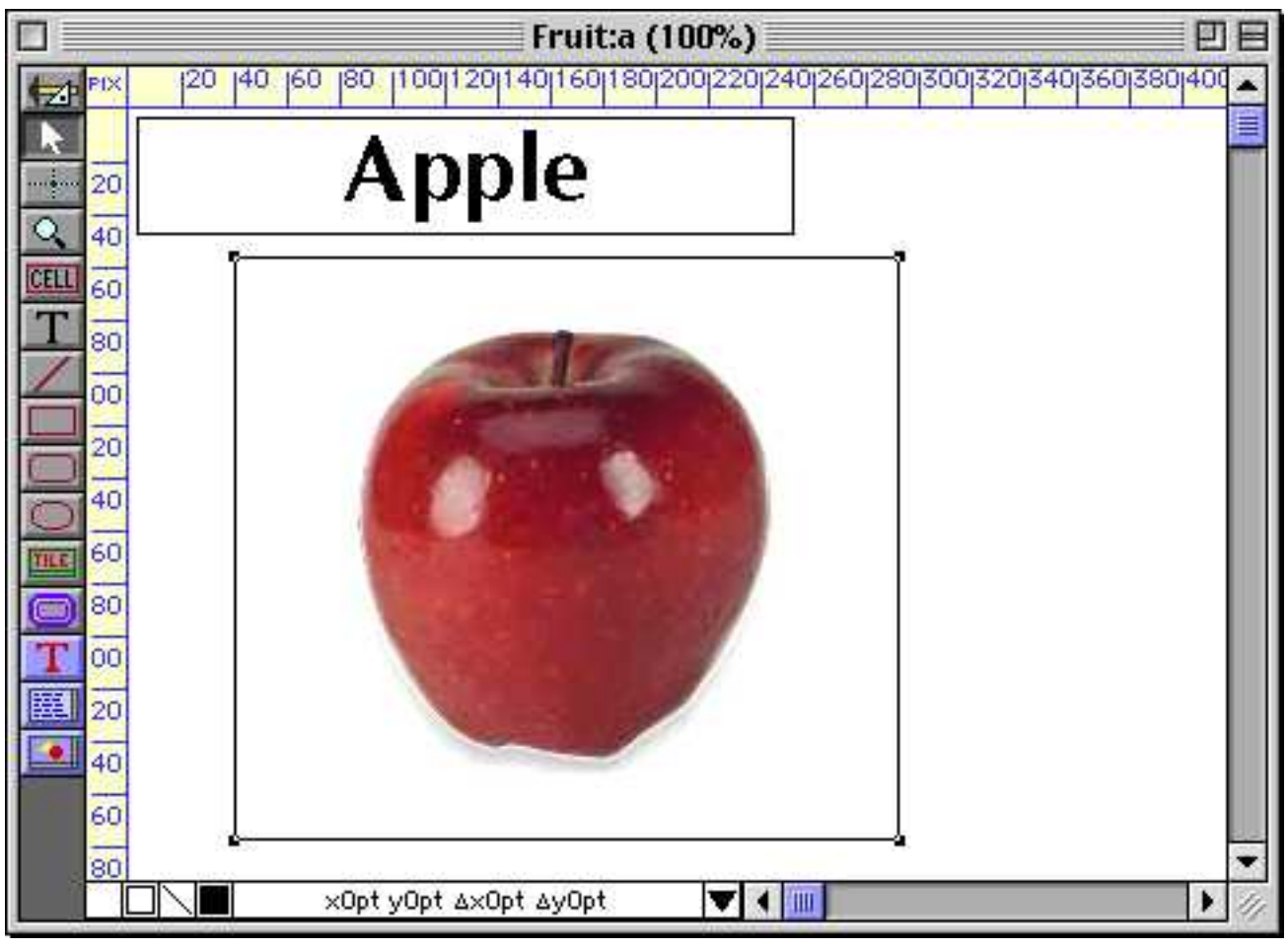
Since the Fruit field contains the name of each fruit, this will automatically display the image for the correct fruit, in this case an apple.



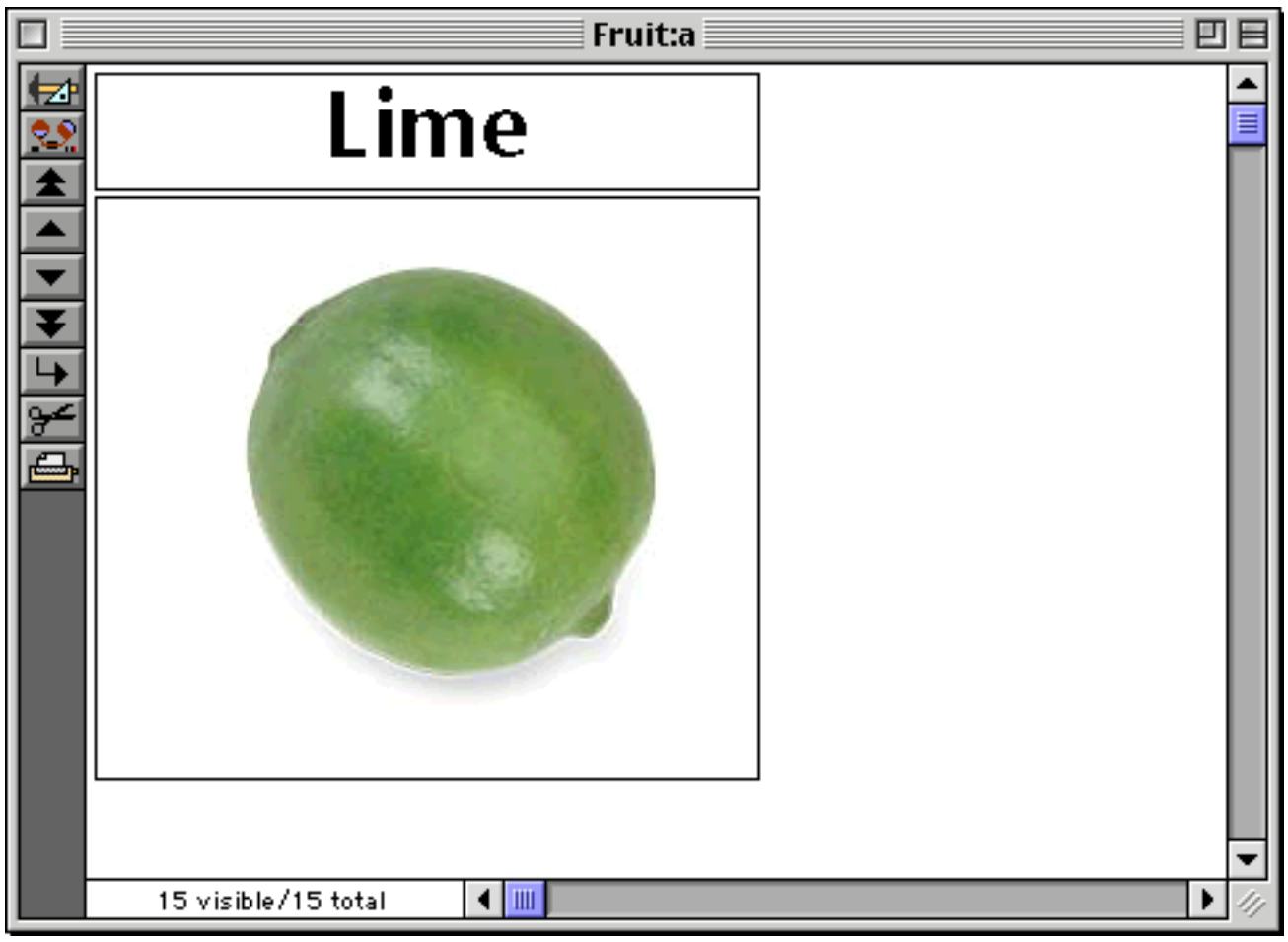
Notice that the apple is displayed in the upper left hand corner of the Super Flash Art object. You can also center the image. To do this, first click on the **Pointer** tool, then double click on the object. This re-opens the configuration dialog, allowing you to change the alignment.



We've also turned on the border for this object.



With a few adjustments the form is ready to use. As long as you have a photo with the correct name available, the correct photo will appear for each fruit.



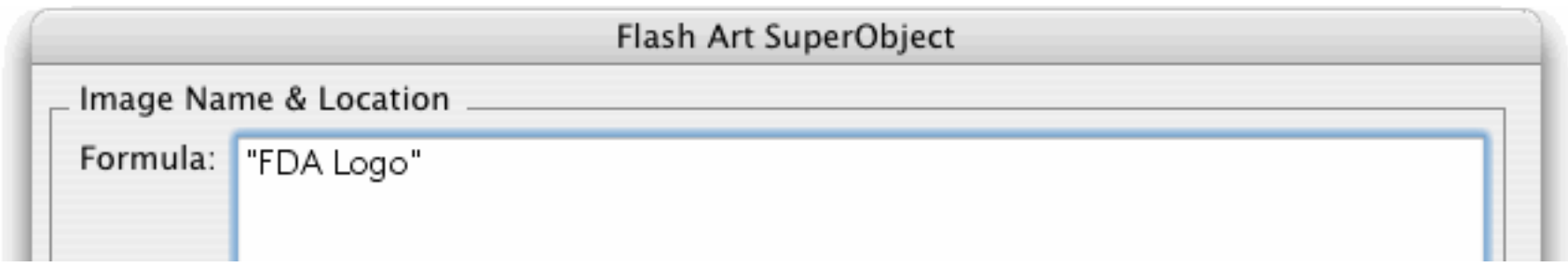
Using Flash Art to Display a Fixed Image

Flash art is usually used to display an image that changes, like the fruit in the previous example. However, it can also be used to display a fixed, unchanging image like a logo or a background. Of course you can also use a standard picture object (see “Fixed Images” on page 739) to display a fixed image, so why use Flash Art? The primary advantage is that using Flash Art a single image can be displayed over and over again without taking up any additional memory or disk space. Only one copy of the image is needed, which can be used multiple times in the database (or even multiple databases).

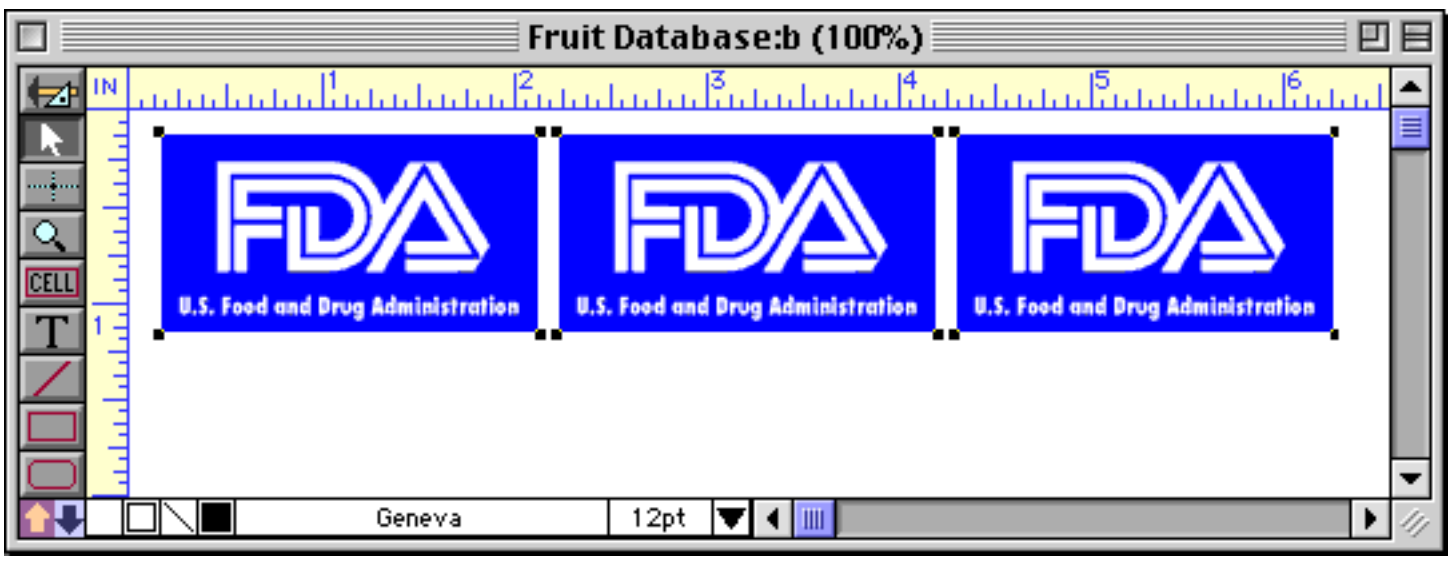
To display a fixed, image, simply enter the name of the image into the Flash Art formula, surrounded by quotes ("). For example, suppose you have an image named FDA Logo.



You can display this image in any form or report by using this formula in a Flash Art or Super Flash Art object.



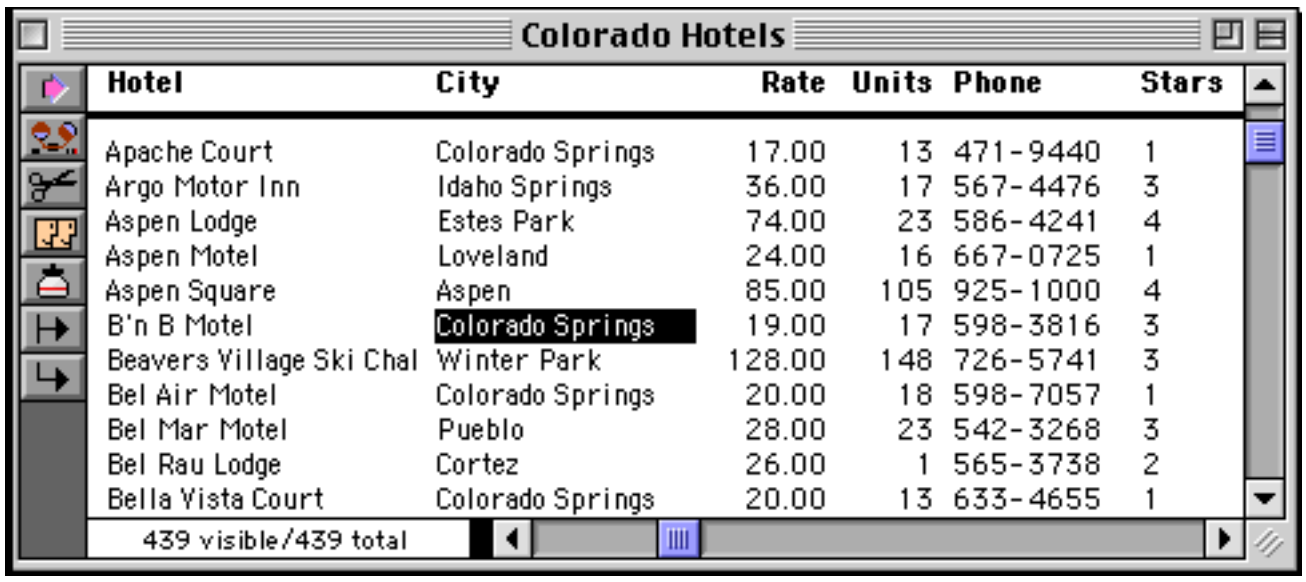
You can use this logo as many times as you like within a single form or across multiple forms.



As an added bonus, if you ever need to modify the logo it only needs to be modified in one place.

Using Flash Art to Display a Smart Background

Flash Art isn't just for displaying pictures. It can also be used to display a border or background that changes depending on the data in the database. To illustrate this we'll use this database of hotel listings.

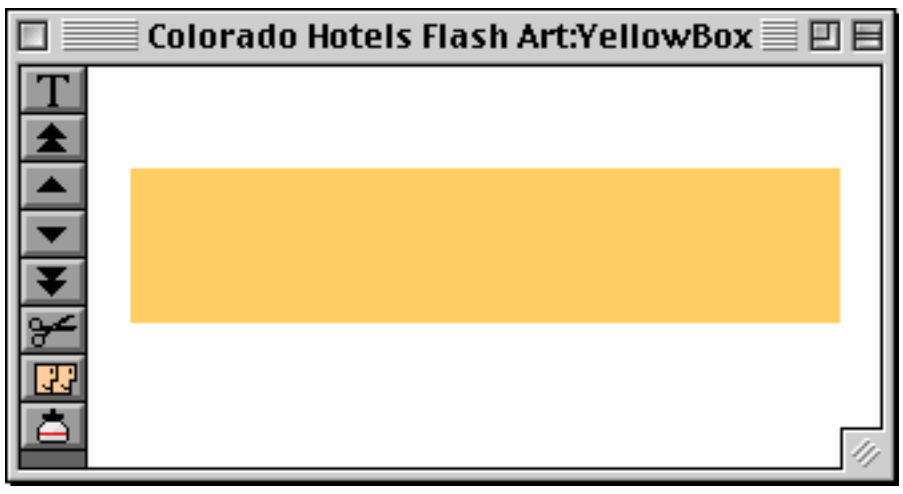


Hotel	City	Rate	Units	Phone	Stars
Apache Court	Colorado Springs	17.00	13	471-9440	1
Argo Motor Inn	Idaho Springs	36.00	17	567-4476	3
Aspen Lodge	Estes Park	74.00	23	586-4241	4
Aspen Motel	Loveland	24.00	16	667-0725	1
Aspen Square	Aspen	85.00	105	925-1000	4
B'n B Motel	Colorado Springs	19.00	17	598-3816	3
Beavers Village Ski Chal	Winter Park	128.00	148	726-5741	3
Bel Air Motel	Colorado Springs	20.00	18	598-7057	1
Bel Mar Motel	Pueblo	28.00	23	542-3268	3
Bel Rau Lodge	Cortez	26.00	1	565-3738	2
Bella Vista Court	Colorado Springs	20.00	13	633-4655	1

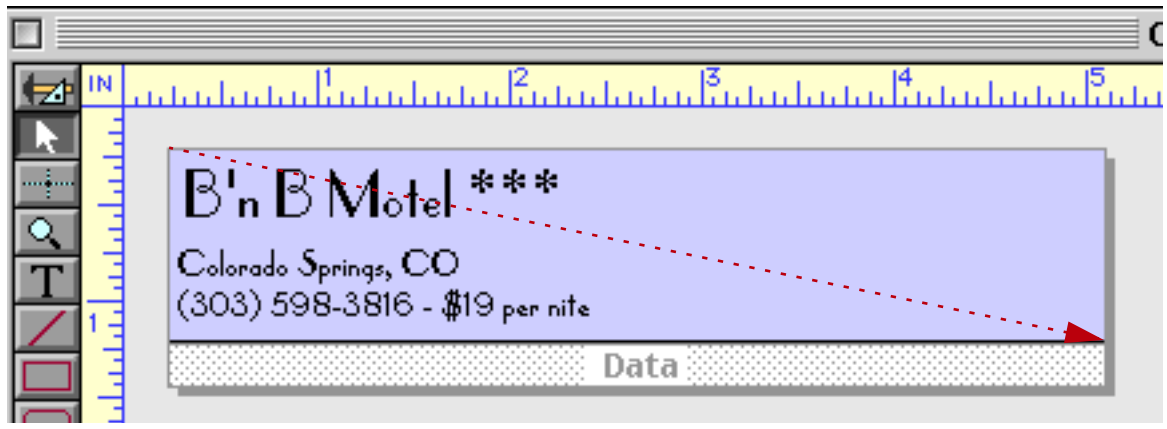
We've created a view-as-list form to display this data (see "View-As-List Forms" on page 889).



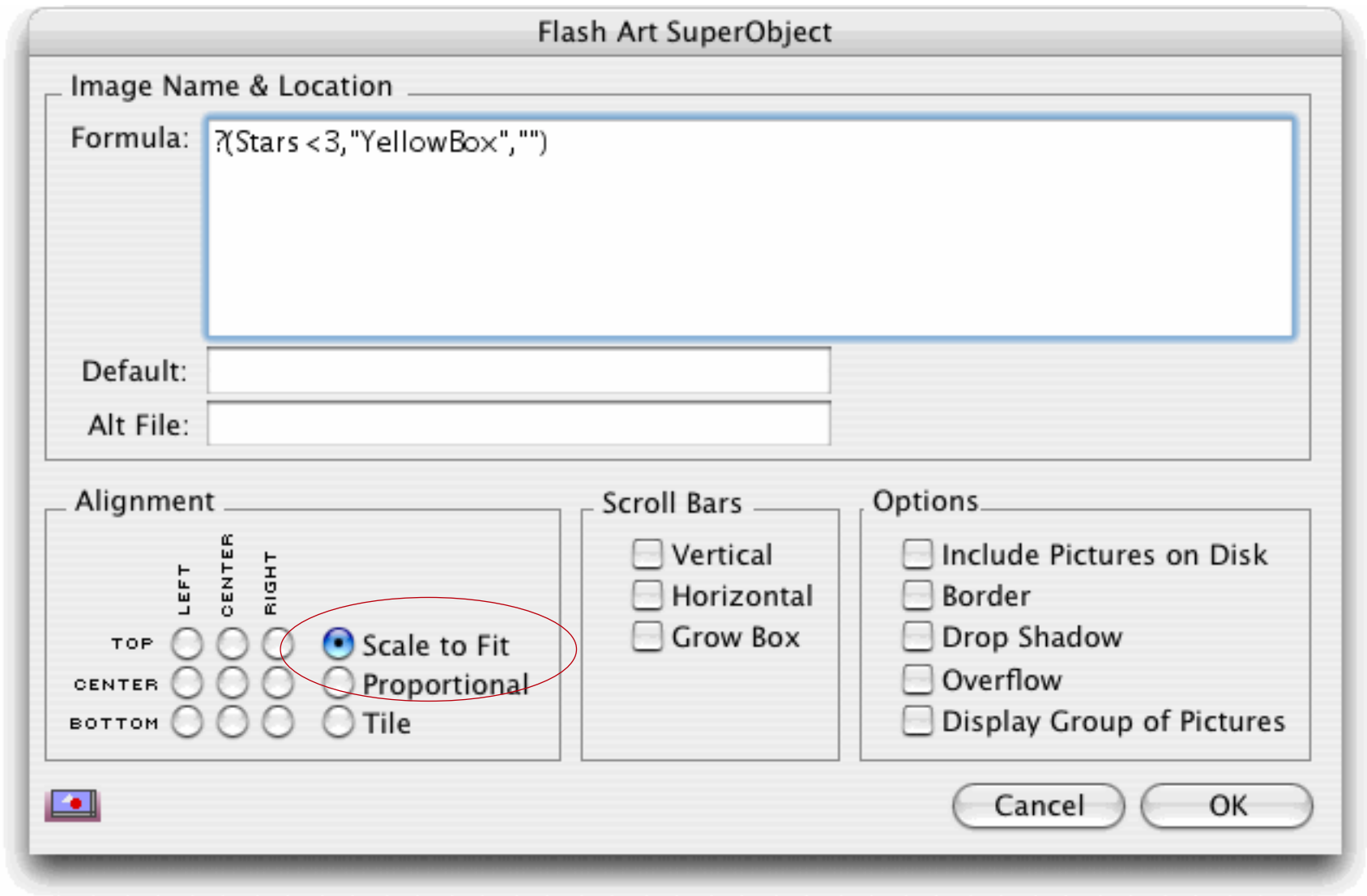
Our goal will be to display a yellow background if the hotel has a rating of 1 or 2 stars. The first step is to create a yellow box. You can do this with a graphics program or with Panorama itself ("Creating a Graphic Object" on page 494). Paste the yellow box into Panorama's Flash Art Scrapbook (see "Adding a New Image to the Scrapbook" on page 763) with the name YellowBox.



Now switch to Graphics Mode and create a Super Flash Art object that exactly covers the data tile. You may need to nudge the object precisely into place after you have created it (see "Nudging an Object (or Objects)" on page 509 and "Nudging the Size of an Object" on page 513).



Set up the Flash Art formula so that the yellow box will be displayed only if the number of stars is 1 or 2 (in other words, less than 3). You'll also want to make sure that the Scale to Fit option is checked. This ensures that the yellow box fills the entire area.



Once the object is created, use the Send To Back command to move it behind the other objects on the data tile (see “Changing the Stacking Order” on page 569). Finally, switch back to Data Access Mode to see the finished result. Voila! Hotels with less than 3 stars have a yellow background, while 3 stars and up have a blue background. (The first line, the B’n B Motel, is actually blue, but is currently highlighted to indicate that it is the current record. If you move to another record you will see that this record is actually blue also.)

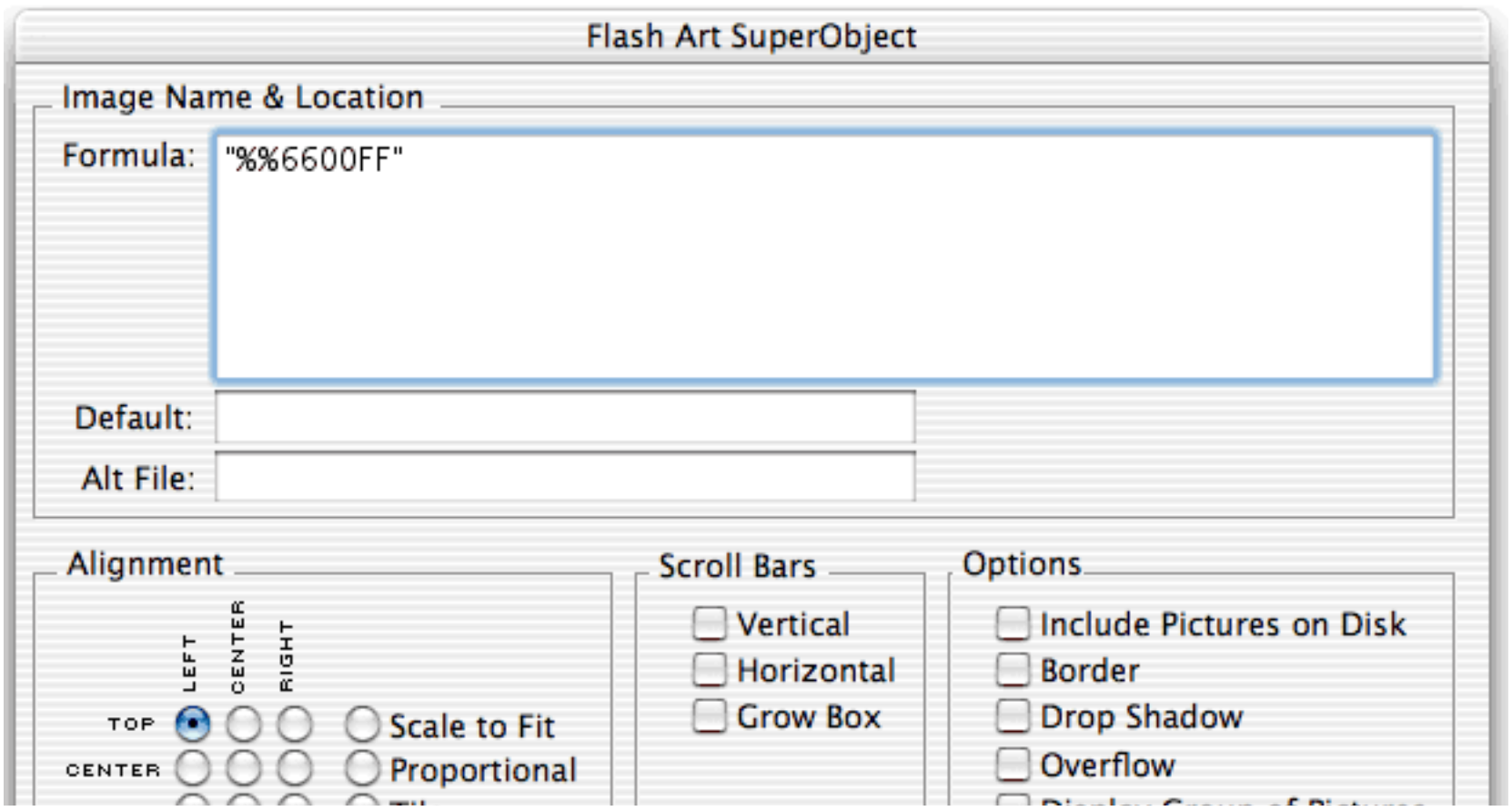


The ?(function is the heart of this formula. See “The ? Function” on page 1271 to learn more about it.

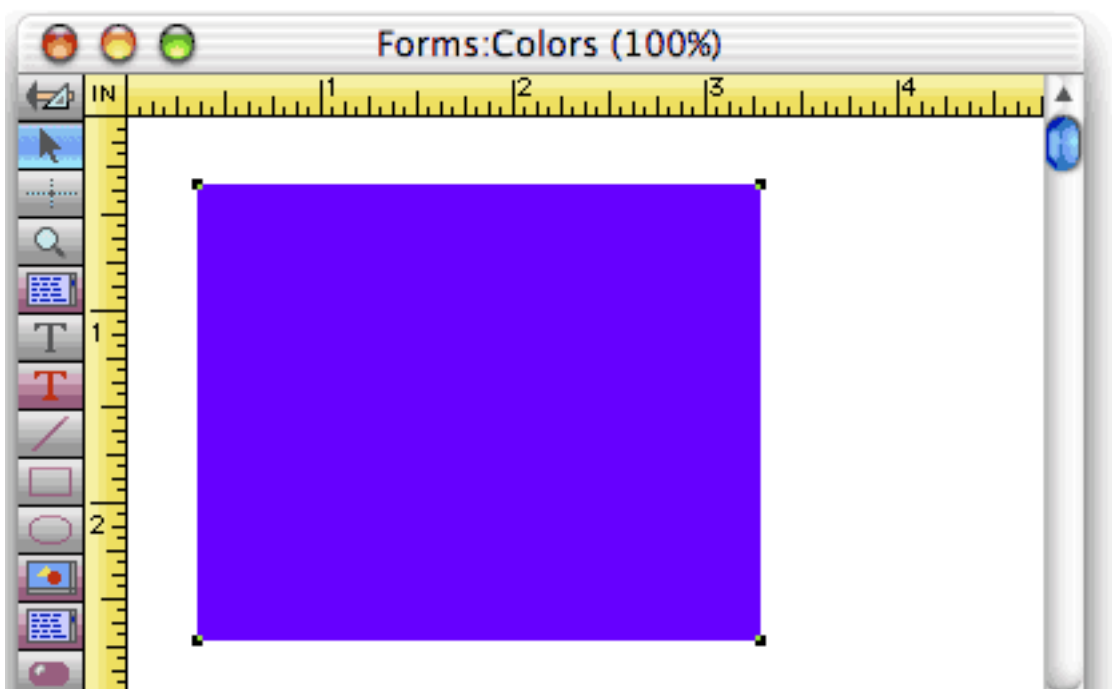
Using Flash Art to Display a Color

In the previous example the Super Flash art object was used to display a simple color. Instead of preparing a color object in advance, the Super Flash art object can also simply display any color you want on the fly. No image is required - the object generates the color on it's own.

To display a color the formula must calculate a text value that starts with two percent signs in a row (%%). This is followed by six hexadecimal characters that specify the color exactly the same way that colors are specified in HTML. The first two hexadecimal digits specify the red component of the color (from 00 to FF), the next two digits specify the green component, and the final two digits specify the blue component. For example, %%FFFFFF is white, %%000000 is black, %%00FF00 is green, etc. Of course if you are displaying a fixed color you'll need to include quotes around these values so that the formula will work correctly, in other words "%%FFFFFF", "%%000000" and "%%00FF00" etc. Here is an example of how to set up a solid color.



In graphics mode, this object will appear as a purple box.



Although this feature can be used to display virtually any fixed color, what makes this feature really interesting is the ability to change the color on the fly. For example, you can set up a Super Flash Art object with the formula `"%%"+SuperColor`. You can set this object to red with a procedure, like this:

```
fileglobal SuperColor
SuperColor="FF0000"
showvariables SuperColor
```

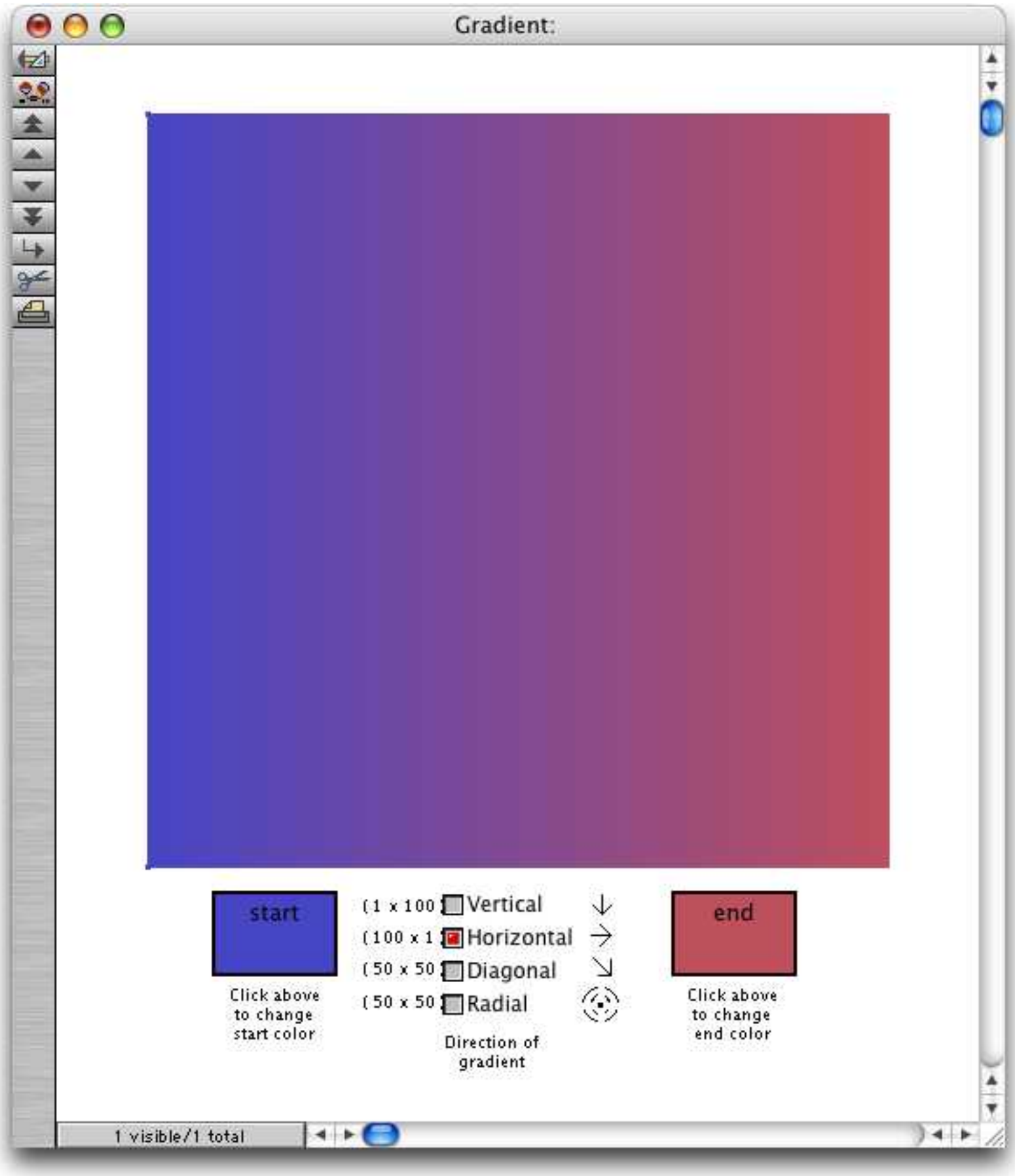
Later if you want to change the color to yellow this can be done on they fly with another procedure.

```
SuperColor="FFFF00"
showvariables SuperColor
```

Here is a procedure that gradually fades the object from black to blue.

```
local bluecolor
bluecolor=0
loop
    SuperColor="0000"+radixstr("hex",bluecolor)[-2,-1]
    showvariables SuperColor
while bluecolor<256
```

Panorama customer Gary Yonaites has created a cool demo that uses the solid color feature in combination with a Super Matrix object to create blends. This demo database, called Gradient, has been included with the Panorama example files.



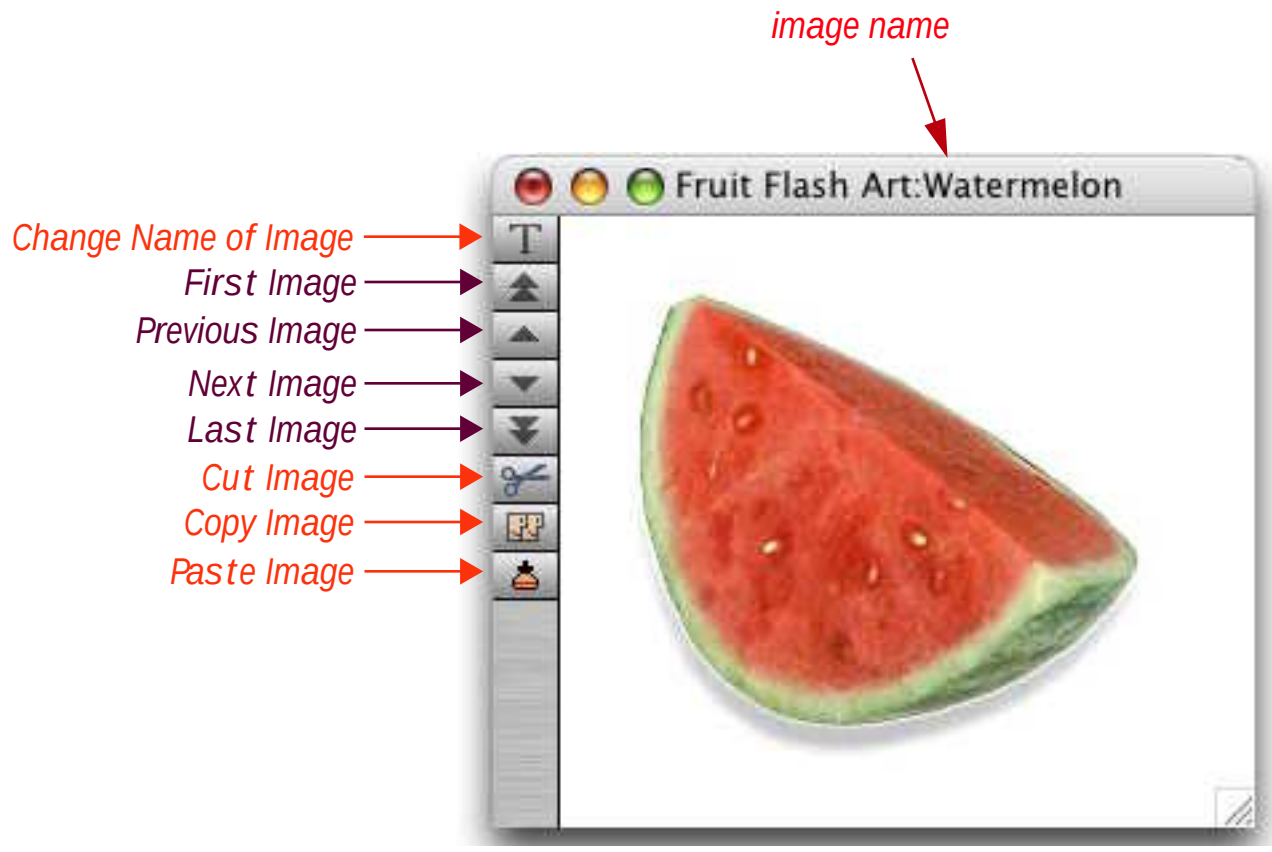
The gradient above was not created in PhotoShop or some other graphics program, but within Panorama itself. By clicking on the check boxes and color selections you can change the gradient on the fly. Thanks Gary!

The Flash Art Scrapbook (Gallery)

Every Panorama database has its own Flash Art Scrapbook, a collection of images that can be displayed within the database. (You can also display images directly from disk files without installing them in the database — see “Displaying Images Directly From Disk Files” on page 767.) Use the **View** menu to open the Scrapbook.



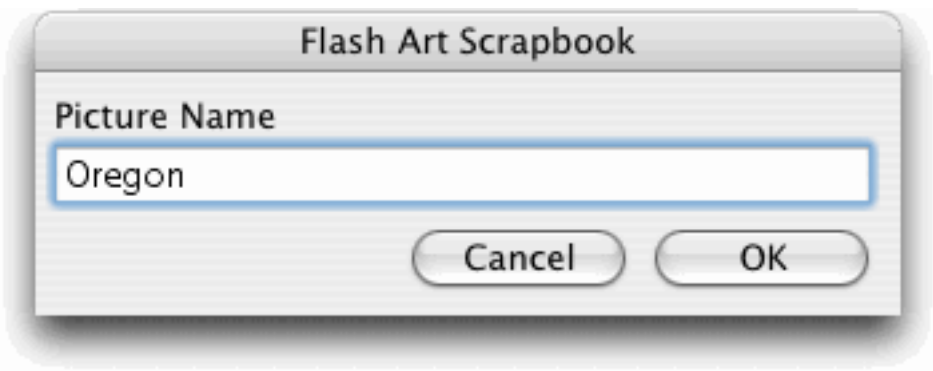
The Flash Art Scrapbook window displays one picture at a time. The name of the picture appears in the windows title bar (for example, the image in the window below is named Watermelon). You can use the **Previous Picture** and **Next Picture** tools to flip through the pictures in the Scrapbook.



You can fill the Flash Art Scrapbook with images you want to use in the database (up to the amount of memory available in your computer). Once a picture is pasted into the Flash Art Scrapbook, it can be used over and over again.

Adding a New Image to the Scrapbook

To add a new picture to the Flash Art Scrapbook, use the **Paste** command or **Paste** tool. Before it adds the picture to the Flash Art Scrapbook, Panorama will ask you for the name of the picture.



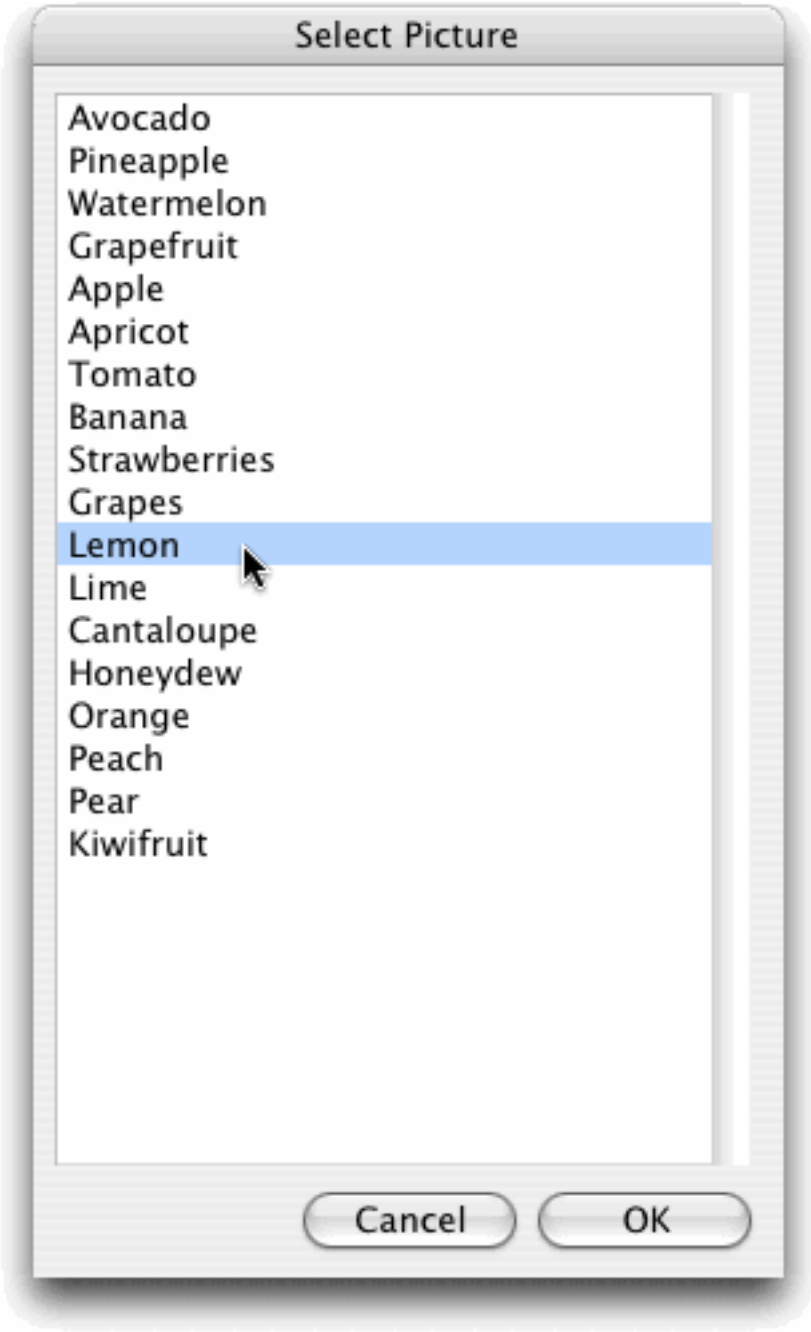
It's a good idea to think about image names before you start. Usually the picture names should correspond to the data in the database. For example if the picture is a map of Oregon, you might name the picture Oregon or OR, depending on how you have entered the state name in your database. If the picture is a photograph of an employee named Bob, you might name the picture Bob or use Bob's initials. Again it depends on how you have entered the in information in the database. Keep in mind that the image names do not have to exactly match the data in the database — but you must be able to write a formula that can convert the database information into the image name. For example, suppose you had an employee database with fields for first, middle and last names. The images could be named according to the employees initials, for example CSL or SLE. In the Flash Art object you would then need to use the formula

```
First[1,1]+Middle[1,1]+Last[1,1]
```

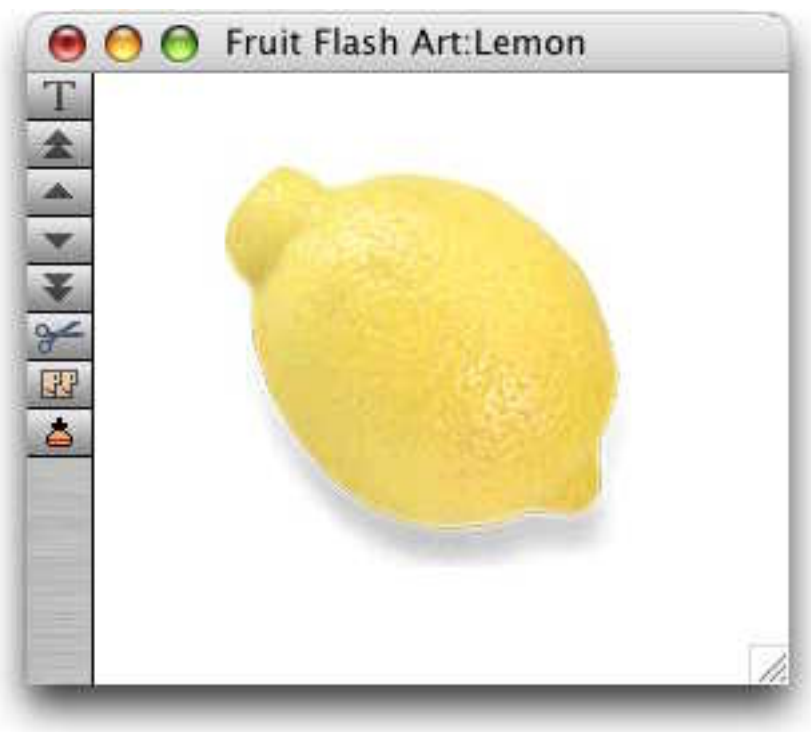
This formulas uses text funnels (see "Taking Strings Apart (Text Funnels)" on page 1214) to extract the initials from the full name, automatically displaying the correct image.

Locating an Image in the Flash Art Scrapbook

The **Find Picture** command (in the Picture Menu) displays a list of all the images in the Flash Art Scrapbook



Choose the image you want to see and press **OK** (or double click on the name) to see the image.

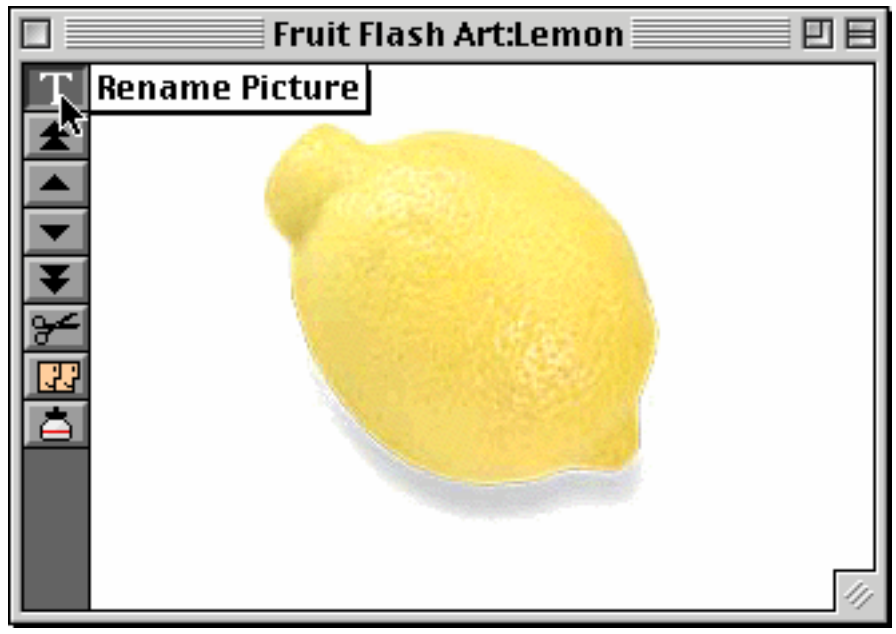


Removing an Image from the Flash Art Scrapbook

To remove an image from the Flash Art Scrapbook, first locate the image and then use the **Cut** tool or choose **Cut** from the Edit Menu.

Renaming an Image

To give an image a new name use the **Rename Picture** tool.



This tool opens a dialog box that allows you to change the name of the currently visible picture.



Re-Arranging the Image Order

You can use the **Re-Arrange Pictures** command in the Picture Menu to change the order of the images within the Flash Art Gallery.



Simply copy the images from the left to the right in the new order. Double click on an image to copy it to the right. (Note: There's really no reason to ever re-arrange the order of the images, since Panorama doesn't care what the order is. The only exception is if you have two images with the same name. In that case, Panorama will always display the first image in the list and ignore the rest.)

Printing the Flash Art ScrapBook

You can use the Print command to print all the pictures in the Flash Art Scrapbook. Panorama will print as many pictures per page as it can fit.

Importing PICT Files into the Flash Art Scrapbook

Many graphic applications can save pictures as PICT files. The **Import PICT** command in the Picture Menu allows you to add these files to the Flash Art Scrapbook directly, without having to go through the clipboard.

To import a single PICT file into the Flash Art Scrapbook, open the **Import PICT** dialog (Picture Menu), then choose the file and press the **Load** button. Panorama will use the name of the PICT file as the picture name.

You can import every PICT file in a folder by pressing the **Load All** button. All the pictures in the folder will be added to the Flash Art Scrapbook, using the names of the files as the names of the pictures.

A Flash Art object can also display PICT files directly from disk without loading them into memory. This saves memory, but is slower than displaying pictures actually loaded into the gallery. To learn how to use PICT files directly see “Displaying Images Directly From Disk Files” on page 767.

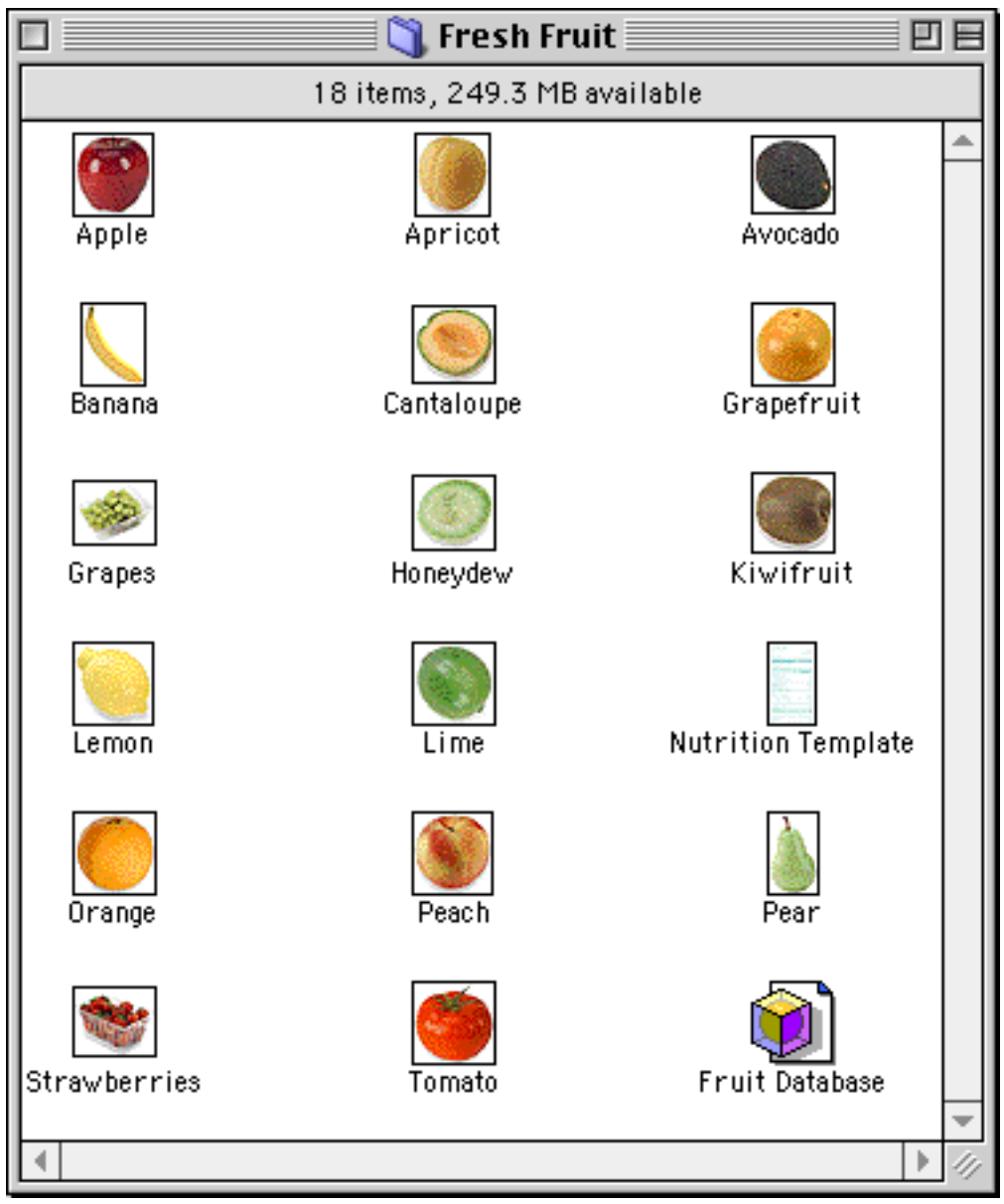
Transferring the Flash Art Scrapbook to Another Database

You can use the **Save Flash Art Scrapbook** and **Import Flash Art Scrapbook** commands to copy the pictures in a Flash Art Scrapbook from one database to another. The **Save Flash Art Scrapbook** command saves the entire Scrapbook in a separate file. The **Import Flash Art Scrapbook** command imports the saved pictures into the current Flash Art Scrapbook.

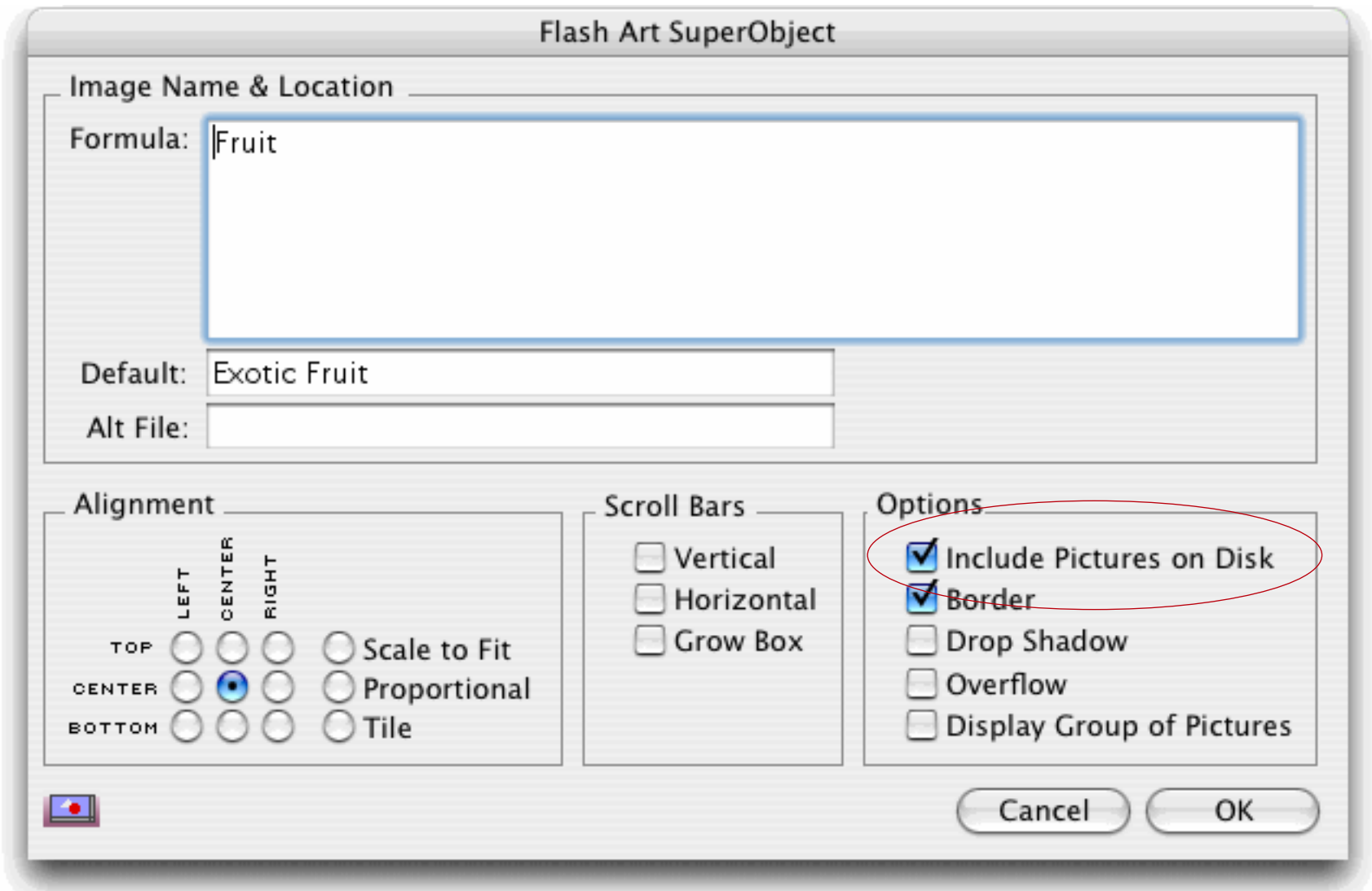
To transfer all the pictures in database A to database B, first open the Flash Art Scrapbook in database A and choose the **Save Flash Art Scrapbook** command (Picture Menu). Type in a name for the exported Flash Art Dialog, and press **Save**. Now open the Flash Art Scrapbook in database B and choose the **Import Flash Art Scrapbook** command. All of the pictures will be added to database B.

Displaying Images Directly From Disk Files

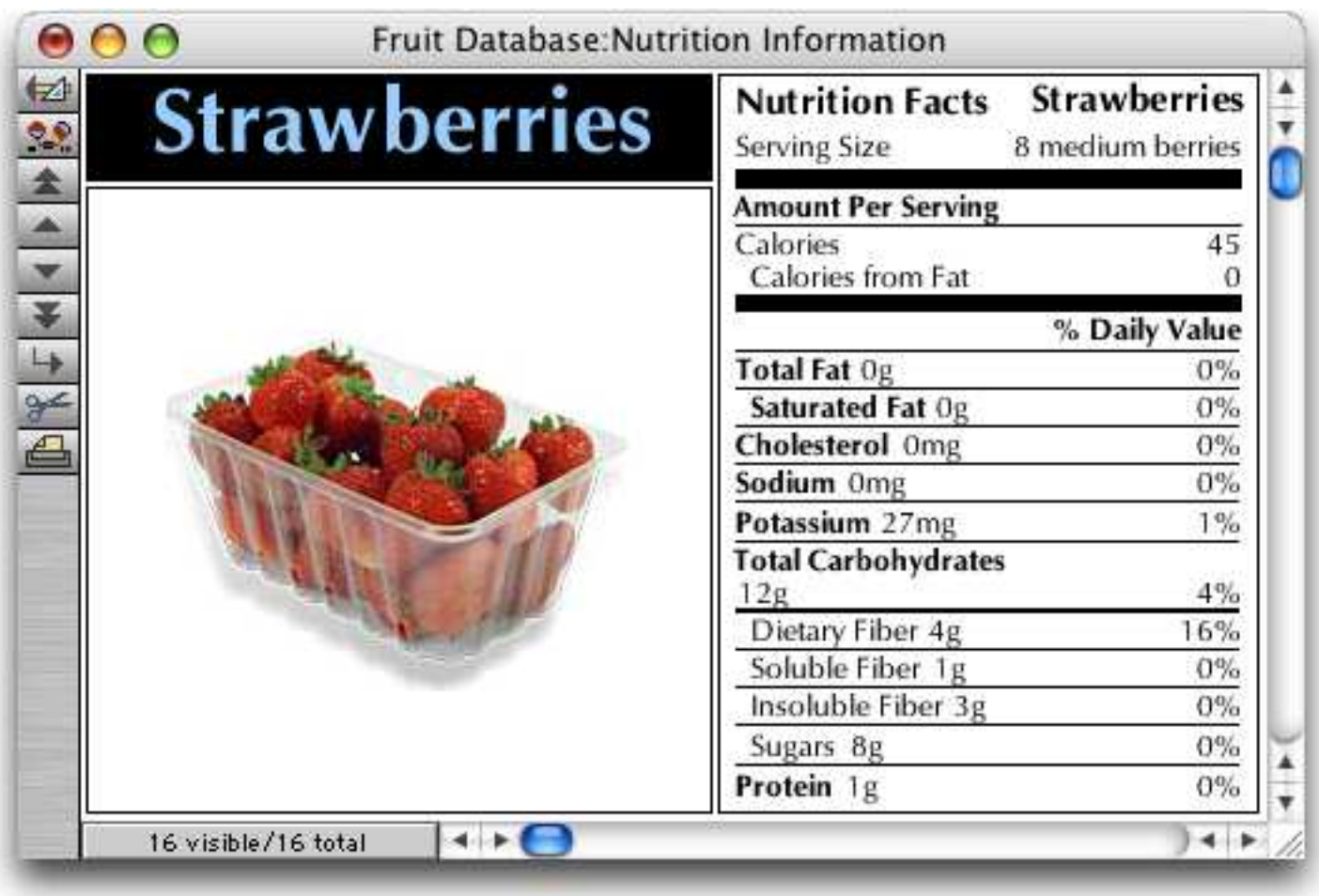
In addition to displaying images from the Flash Art Scrapbook, Panorama can also display images directly from disk files. If the images are large, this can save a tremendous amount of memory. It also makes it easier to edit the images. The downside is that if you move or copy the database to a new location you’ll have to make sure to move or copy all of the images also. Here is a typical folder full of image files.



To allow Panorama to display images directly from disk files you must enable the Include Pictures on Disk option.



Since the images are in the same folder as the database itself, only the image name is needed as the formula (in this case the image name is in the field Fruit). All of the images have been removed from the Flash Art Scrapbook in the database shown below (saving 782 kilobytes of memory - almost a megabyte!), so now the images are displayed directly from the disk files.

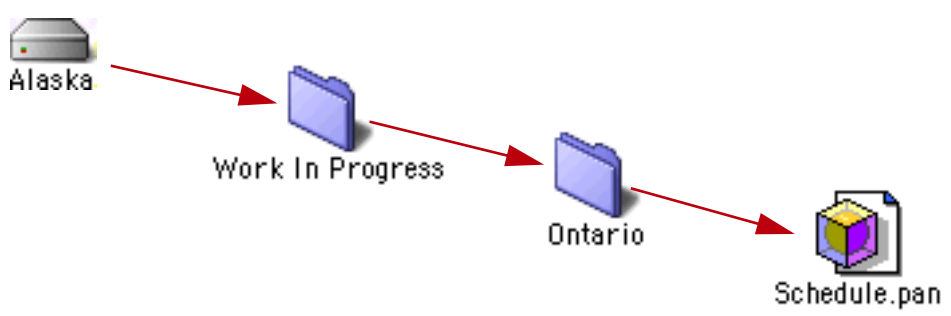


Note: If the Flash Art Scrapbook is not empty, Panorama will look there first when trying to locate an image. If the Flash Art Scrapbook contains an image with the correct name, it will be displayed, and Panorama will ignore any disk file with the same name.

Displaying Images in a Different Folder (Directory)

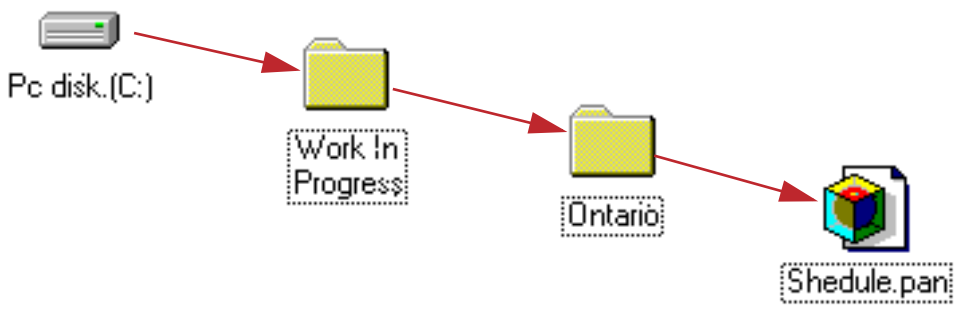
Flash Art images don't have to be in the same folder as the database—they can be in any folder (directory) on your hard disk, or even on a removable media like a CD-ROM or a Zip disk. On the Macintosh, the exact location of any file can be specified by stringing together the name of the volume (disk) and the folders, each separated by a colon.

```
Alaska:Work in Progress:Ontario:Schedule.pan
```



Windows systems are similar, but backslashes (\) are used instead of colons, and drive names always consist of letters followed by a colon (A:, B:, C:, etc.).

```
C:\Work in Progress\Ontario\Schedule.pan
```

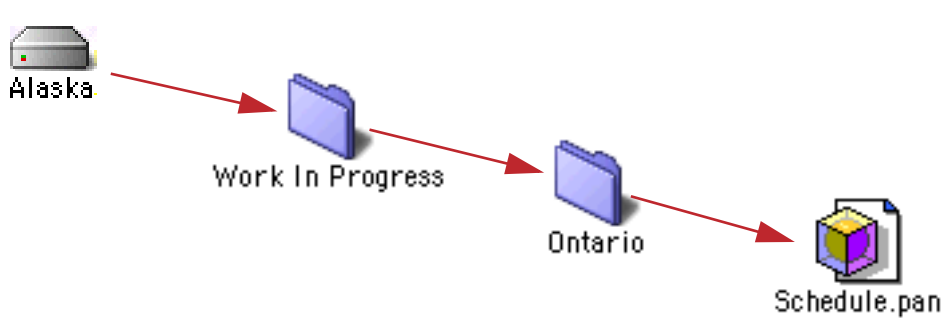


For cross platform compatibility, Panorama also allows you to use colons when using Panorama for Windows, like this:

```
C::Work in Progress:Ontario:Schedule.pan
```

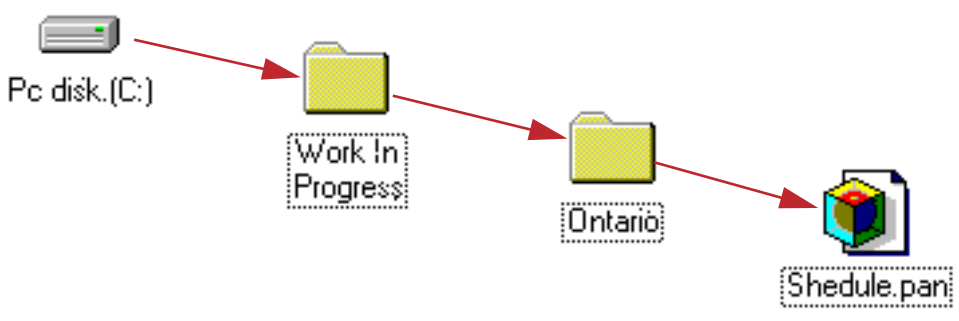
A file's location may also be specified relative to the current database. For example, suppose the current database was in the Work in Progress folder. In that case you could specify the location of the Shedule.pan file by simply leaving off left hand portion of the specification. The specification must begin with a colon or backslash to indicate that it is relative to the current folder and not an absolute location.

```
:Ontario:Schedule.pan
```



On PC systems you can specify this relative location like this:

```
\Ontario\Schedule.pan
```

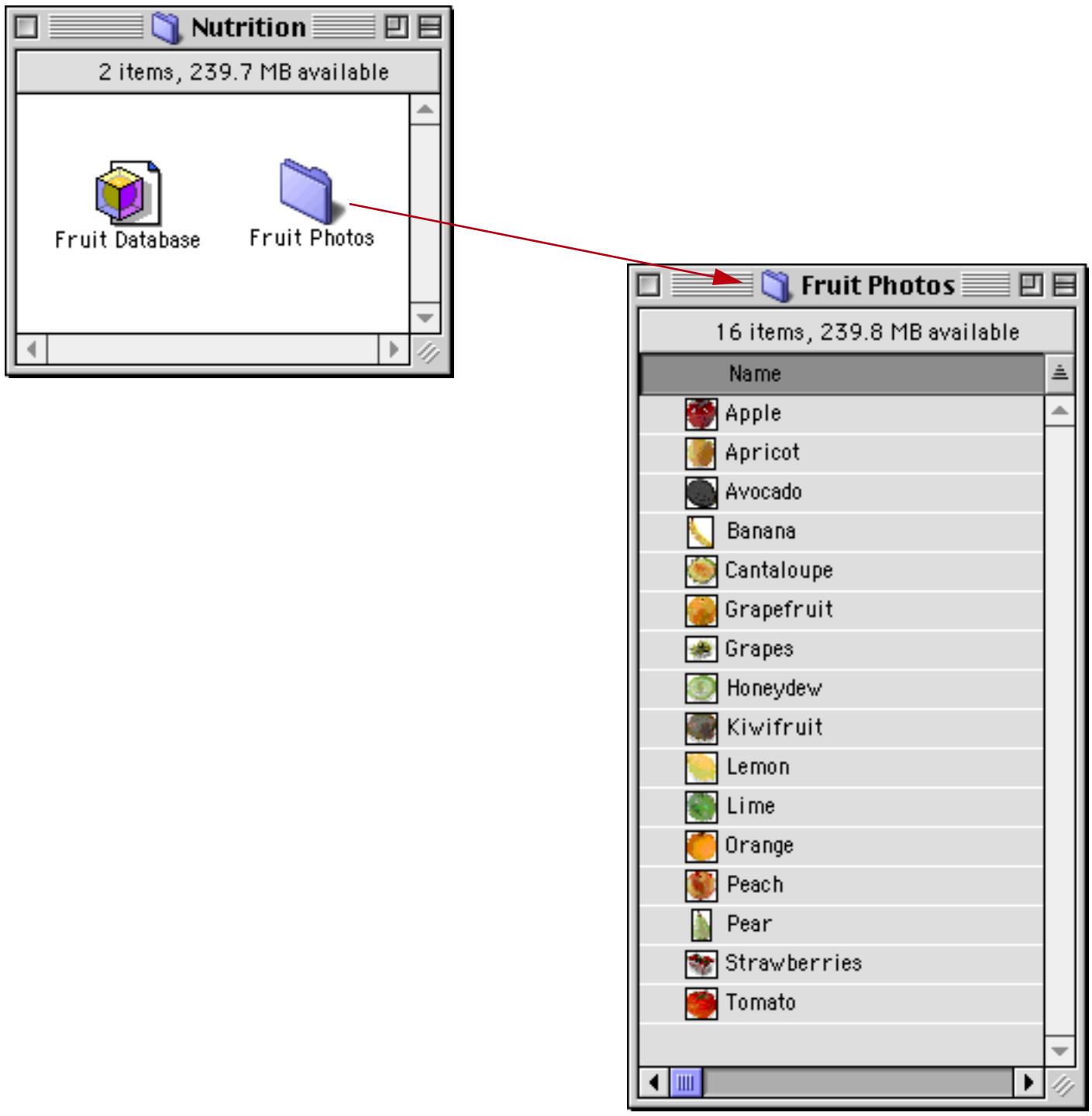


However, keep in mind that on PC systems Panorama will accept : instead of \. Therefore, the specification

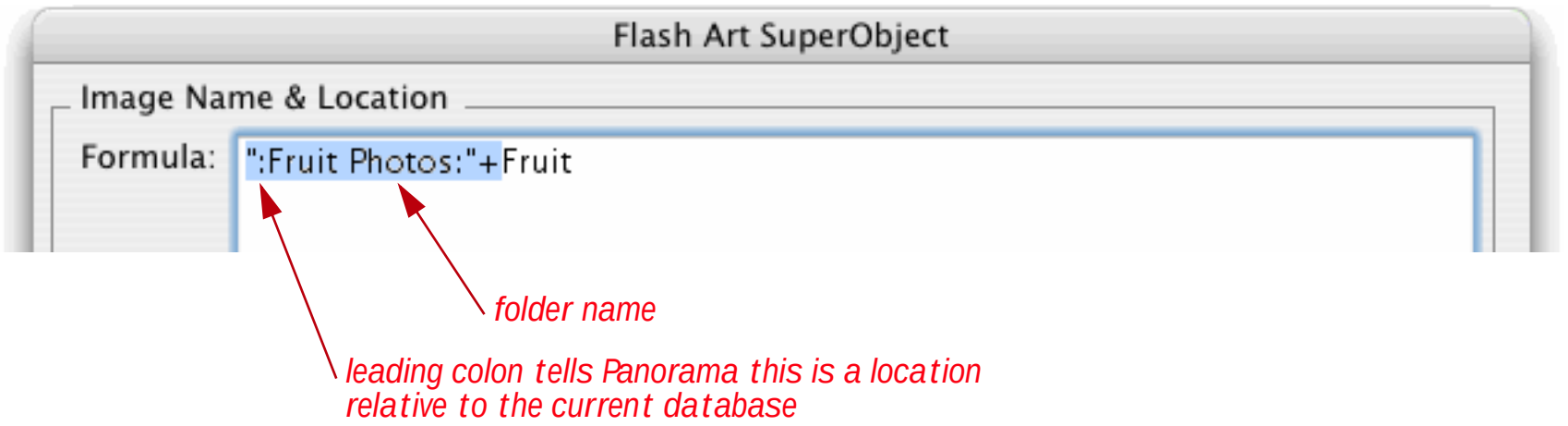
```
:Ontario:Schedule.pan
```

will work on both Windows and MacOS based computers. You should use colons if your database might ever be used on both Macintosh and Windows computers.

To illustrate all of this with a real-world example, let's revise the Fruit Nutrition database from the previous section. Instead of placing all of the image files in the same folder as the database, we will move them to their own folder, with that folder in the same folder as the database itself.

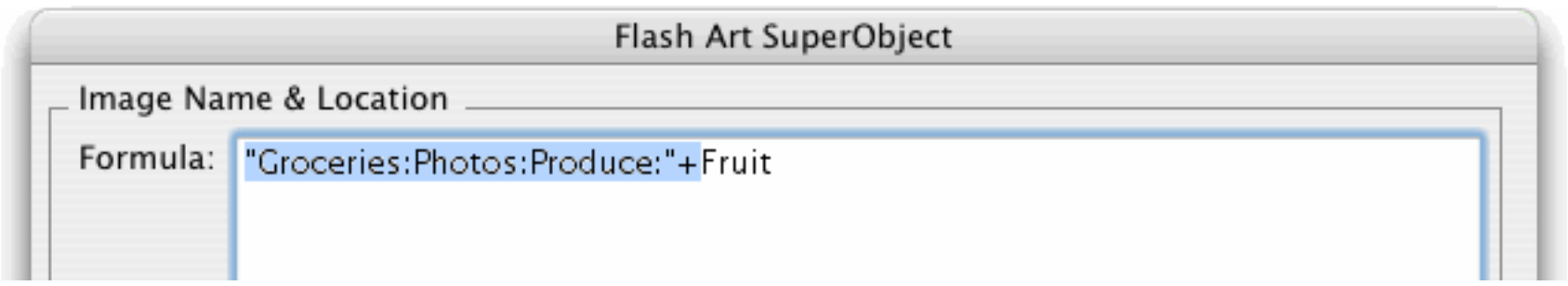


The Fruit Photos folder is said to be **nested** inside the Nutrition folder, which is the folder that actually contains the database. To display the photos, the formula in our Super Flash Art needs to be modified.

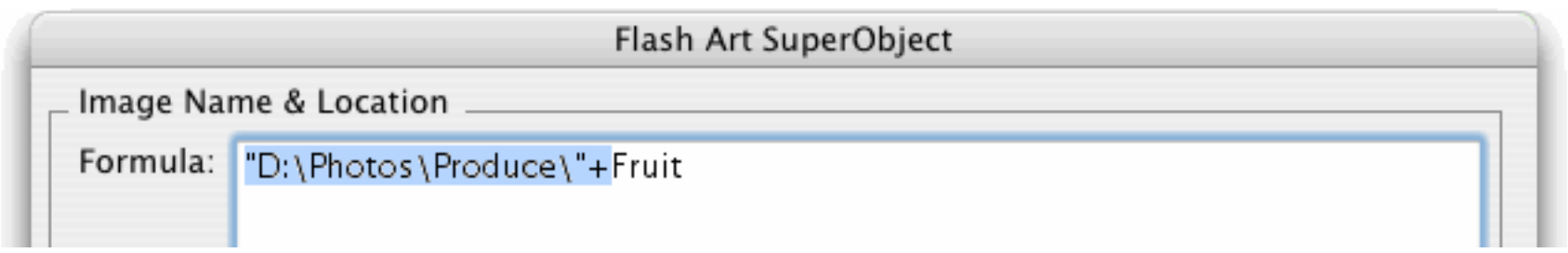


The new formula combines the fruit name with the folder location. For example, if the fruit name is Apricot, the result of the formula will be `:Fruit Photos:Apricot` — the exact location for the image.

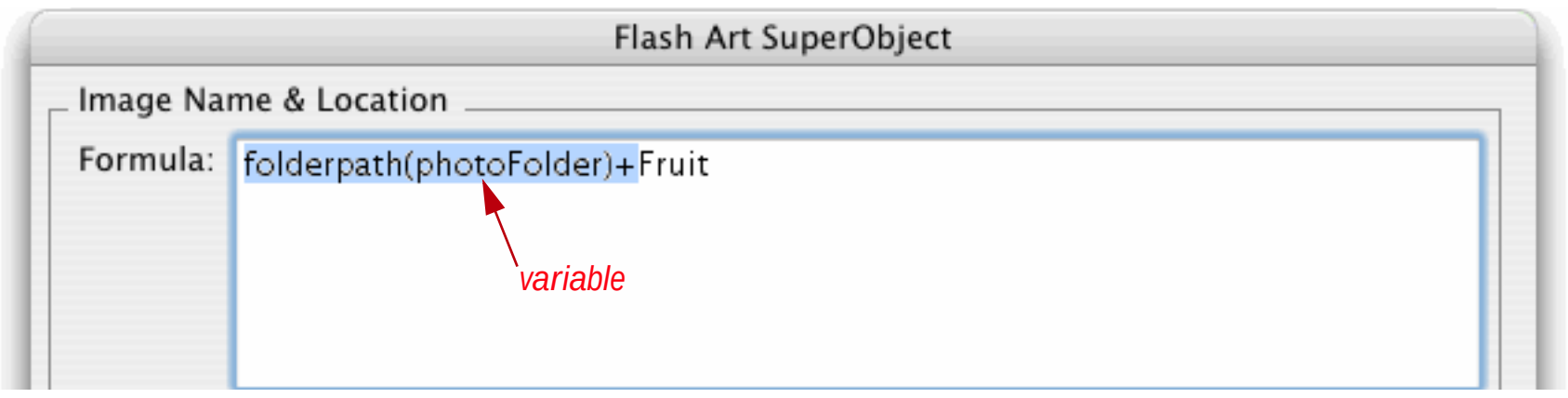
A similar technique can be used to specify an absolute location for the image. Suppose you have a CD-ROM named Groceries that contains the images nested within the Produce folder within the Photos folder. The formula below could be used to display the images.



On the PC the CD-ROM drive is usually D:, so this would be the formula.



The photos do not have to be in a fixed location. Using a variable (see "Variables" on page 1199 and "Variables" on page 1370) and the `folderpath(` function (see "Disk Files and Folders" on page 1304) you can allow the folder to be moved around.



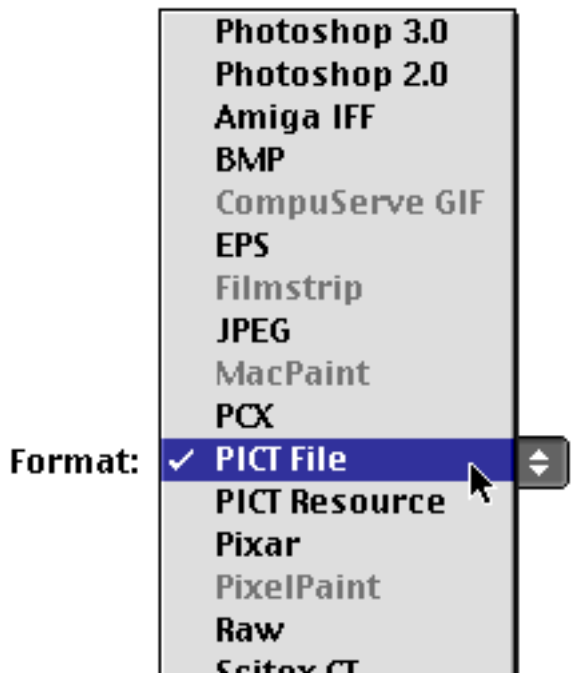
Elsewhere in your database you can include a procedure (see "Procedures" on page 1339) like this to allow the database user to select the folder containing the images.

```
local xFile,xFolder,xType
openfiledialog xFolder,xFile,xType,""
if xFile=""
   stop    /* stop because the Cancel button was pressed */
endif
photoFolder=xFolder   /* update location of photos */
showvariables photoFolder   /* make sure new photos are displayed immediately */
```

All of the previous examples have assumed that the images are all in the same folder. However, that is not necessary. If you wish, the folder location may be included in the database itself, allowing different images to be in different folders. The folder location may be combined in with the image name in a single database field, or they may be stored in separate fields and combined in the Flash Art formula.

Displaying Non PICT Images (Enhanced Image Pack)

The standard configuration allows Panorama to display images in PICT format. This is the standard format for images on Mac OS computers, and is easily created using graphics programs on both Macintosh and Windows computers. This illustration shows how the PICT format may be selected when saving a file in Adobe Photoshop.

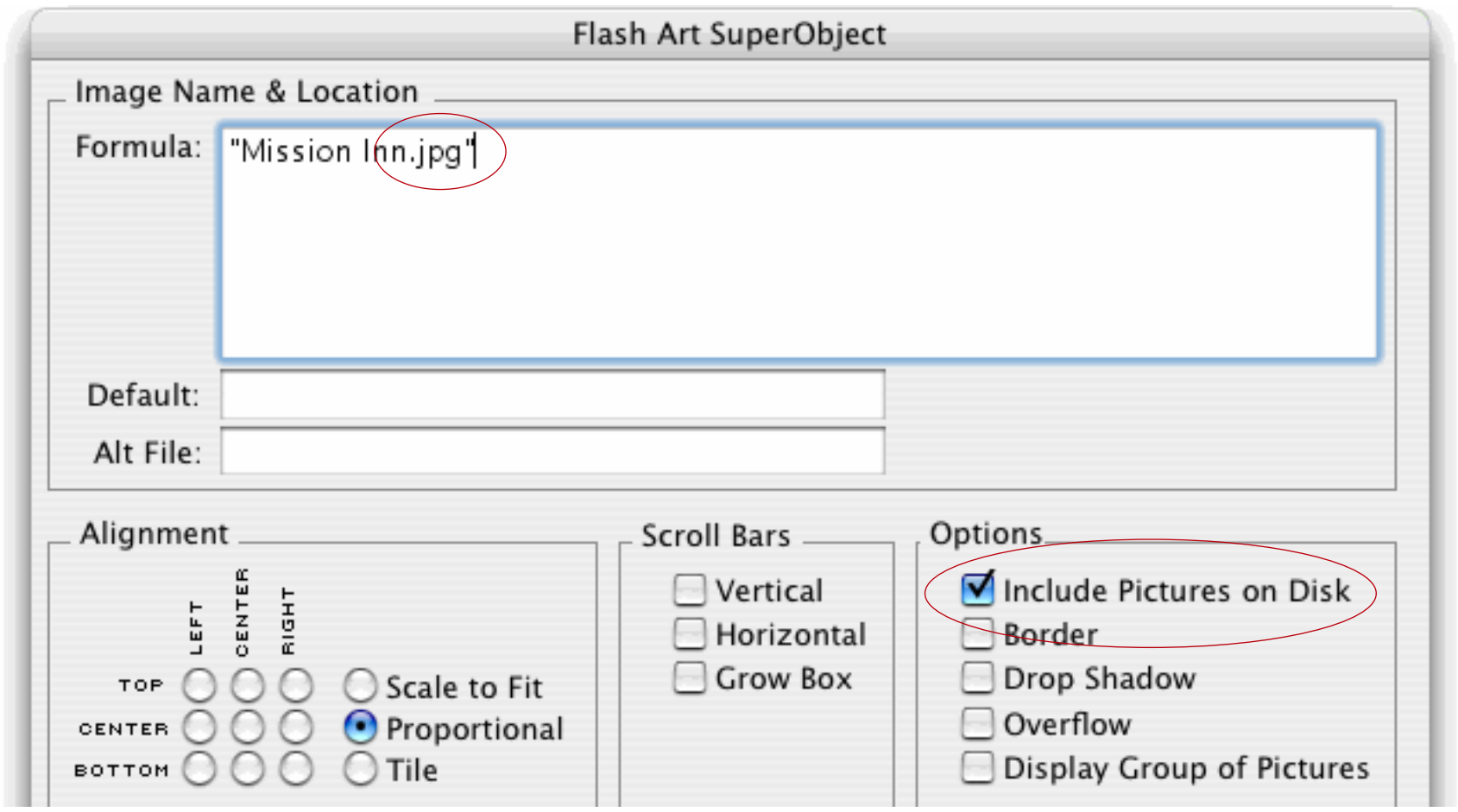


The optional **Enhanced Image Pack** gives Panorama the ability to display a wide variety of other image formats. This table lists some of the most popular image formats that can be displayed with this option.

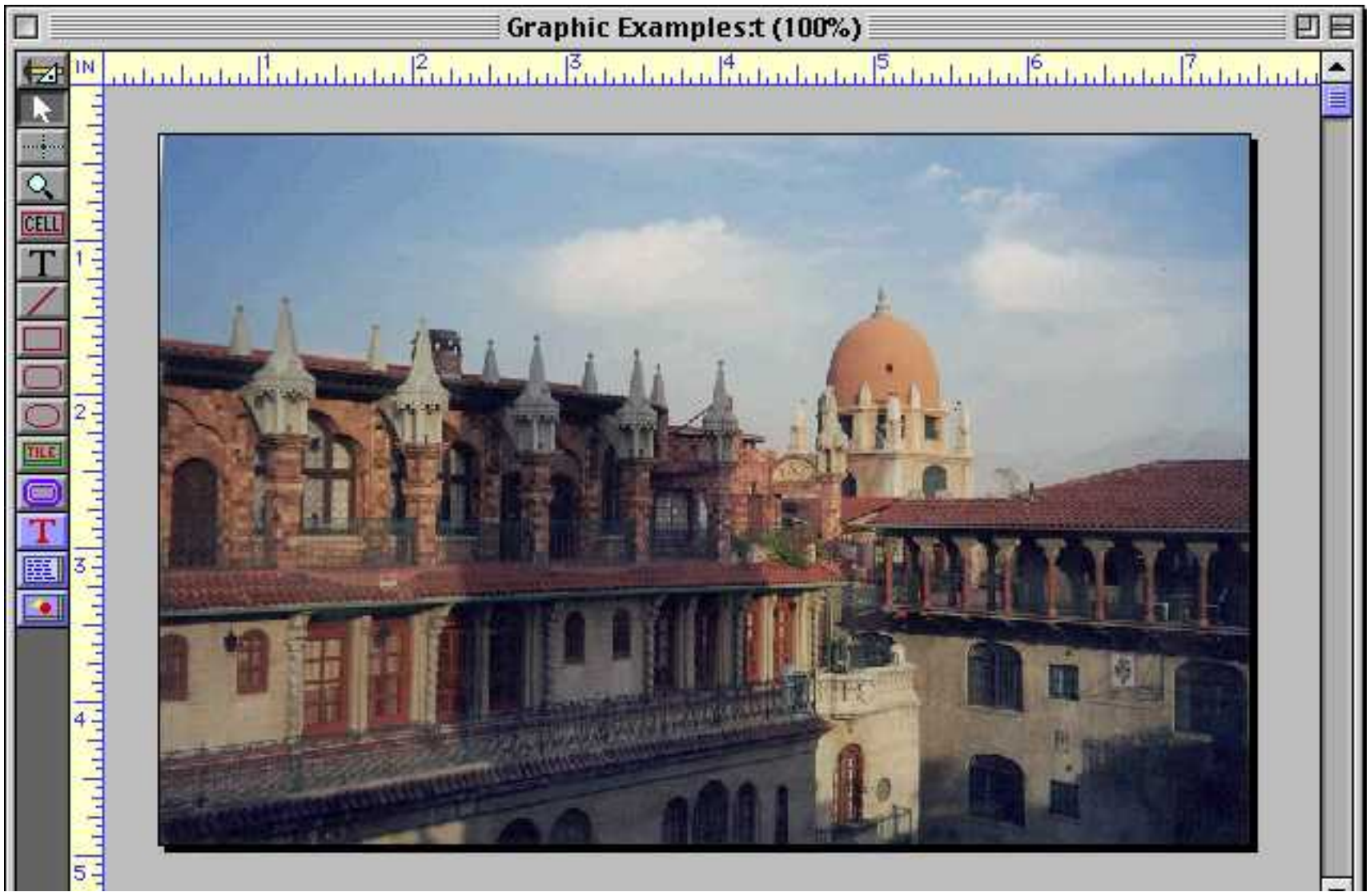
Image Type	PC Extensions	Notes
BMP	.bmp	Windows and OS/2 bitmap
JPEG	.jpg .jpeg	JPEG compressed image
PNG	.png	Portable Network Graphics bitmap
TIFF	.tif .tiff	Tagged Image Format
GIF	.gif	Common web format
PHOTOSHOP	.psb	Adobe Photoshop
FLASHPIX	.fpx	FlashPix bitmap
TARGA	.targa	

The **Enhanced Image Pack** requires that Apple Quicktime 4.0 or later be installed on your computer. If Quicktime is not already installed on your system you can download it from www.apple.com.

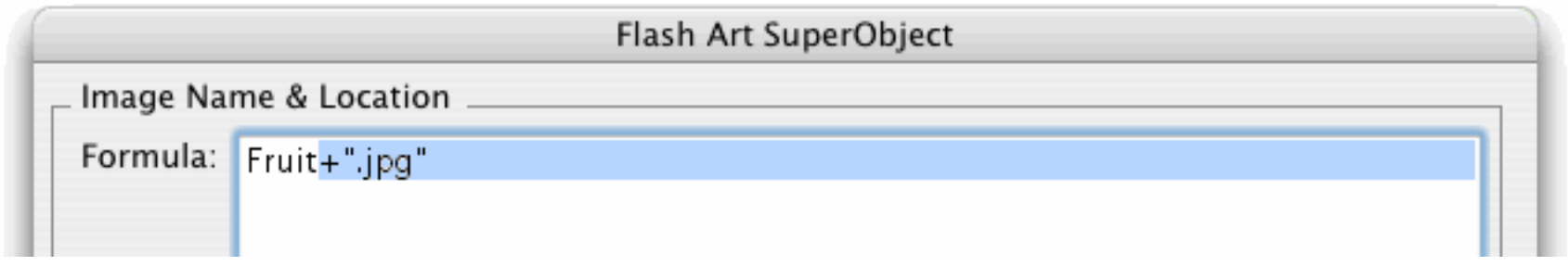
Once the **Enhanced Image Pack** is installed on your system you can display any of these image formats by name, just as with PICT images. This illustration shows how to display a fixed JPEG image.



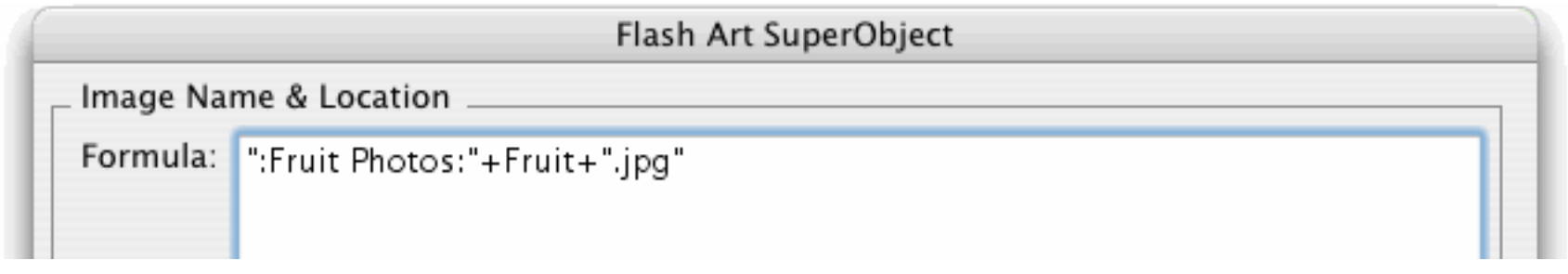
Make sure you enable the Include Pictures on Disk option, as shown above. Here is the actual JPEG image being displayed in the form.



Of course usually the images you display will be variable, not fixed. For example, suppose the fruit images used in the previous examples were JPEG images instead of PICT format. In that case you could display the images using this formula.



Or, if the images were nested in a different folder you could use a formula like this (see “Displaying Images in a Different Folder (Directory)” on page 769).



In addition to displaying images the Enhanced Image Pack can also convert an image from one format into another (see “Converting Between Image Formats” on page 1814). For more information on ordering the **Enhanced Image Pack** visit our website at **http://www.provue.com**.

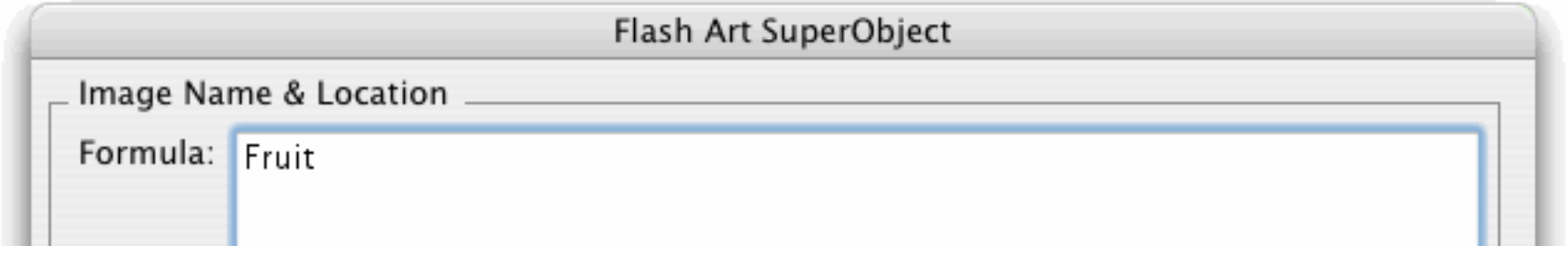
Image File Extensions in a Cross Platform Environment (MacOS and Windows)

On Windows systems all files have a three our four letter “extension” based on the type of data in the file. For example, all Panorama databases end with .pan, all text files end with .txt, all PICT image files end with .pct and all JPEG image files end with .jpg or .jpeg.

On Macintosh systems this extension is not necessary because the file itself contains information about what type of data it contains. (However, for a few file types it is traditional to include the extension anyway, for example JPEG or HTML files.)

For most databases you don’t need to worry about whether or not to use file extensions. If the database will be used on a PC system, you use an extension, if it will be used on a Macintosh, you don’t. But what about cross-platform databases that may be used on both platforms? Panorama is designed so that you can build a single database that will automatically display images correctly on both platforms. Let’s see how it’s done.

As you learned earlier, Panorama uses the Flash Art formula to generate the name of the image to display (see “Displaying Images Directly From Disk Files” on page 767). On Windows systems, this file name **must** include an extension (.pct, .bmp, .jpg etc.). If it doesn’t, Panorama will automatically add .pct to the end of the name. (Remember, .pct is the extension for PICT images, Panorama’s standard image format.) To illustrate this, let’s go back to the fruit example we saw earlier. Here’s the formula:

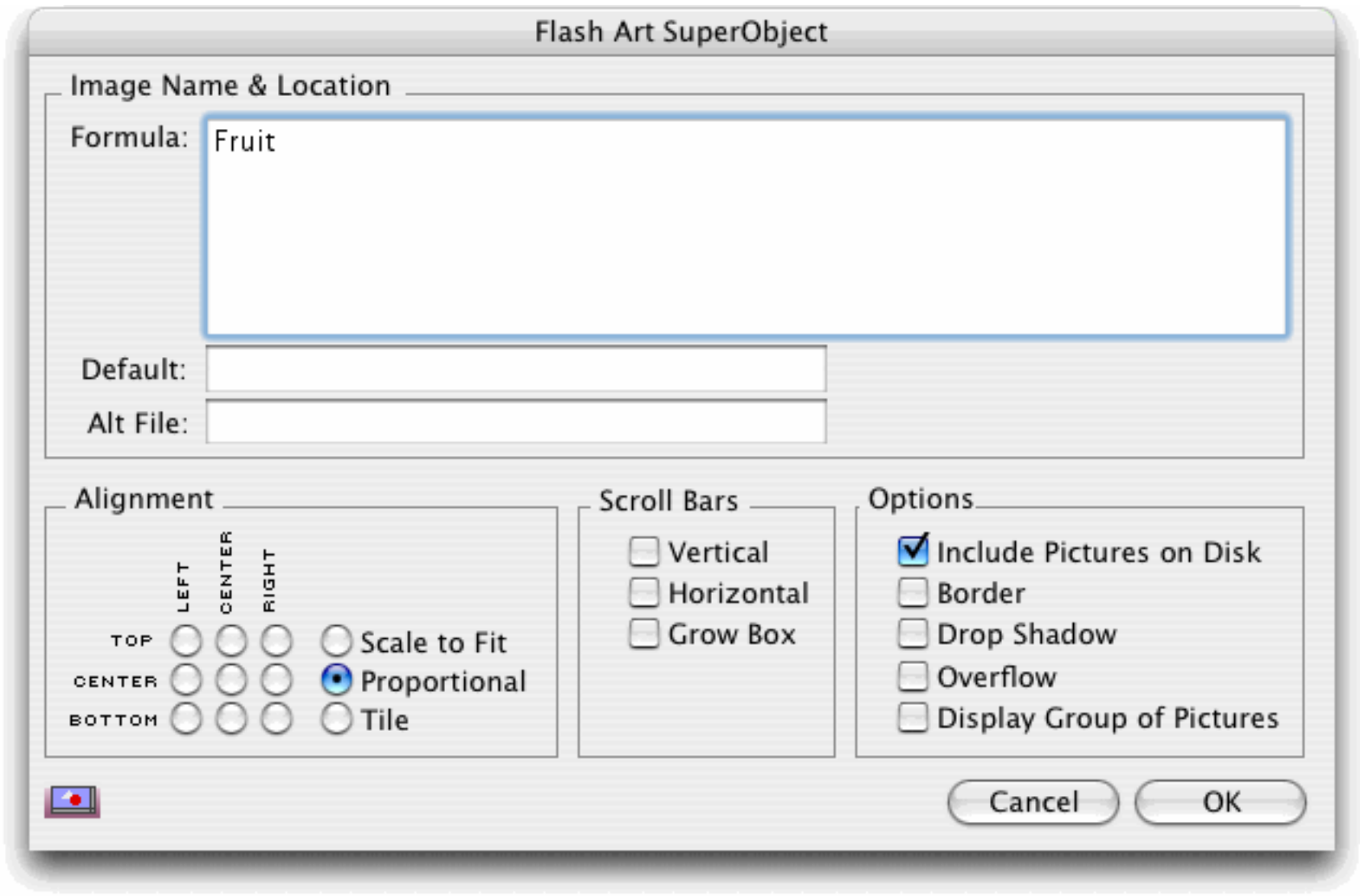


On a Macintosh system this formula will simply generate the names of the fruits: Apple, Apricot, Avocado, etc. But on a Windows computer the Panorama will automatically add the .pct extension: Apple.pct, Apricot.pct, Avocado.pct etc. Therefore the same formula will automatically work on both Macintosh and PC computers. (If you use the **Panorama Platform Converter** to convert your database from Mac to PC it will automatically add the .pct extension to all of your image files fro you, or remove the extension when moving from PC to Mac. See "Platform Converter Wizard" on page 1856 for more information on the Platform Converter.)

When you use Panorama on a Macintosh system it doesn't normally add the .pct extension. However, there is one exception. If the name of the database ends with .pan, Panorama will automatically add the .pct extension to any image file name that does not already include an extension. This allows you to create a database that may be shared on a server between Mac and PC systems. The same database can be used on either system with no conversion. However, this also means that when the database name ends with .pan images that do not have any extension (for example Apple or Tomato) cannot be displayed, only images that do have extensions (for example Apple.pct or Tomato.pct). (This restriction does not apply, however, to images in the Flash Art Scrapbook, see "The Flash Art Scrapbook (Gallery)" on page 762.)

Super Flash Art™ Options

The SuperObject™ Flash Art configuration dialog has numerous options for customizing each object you create.



Formula

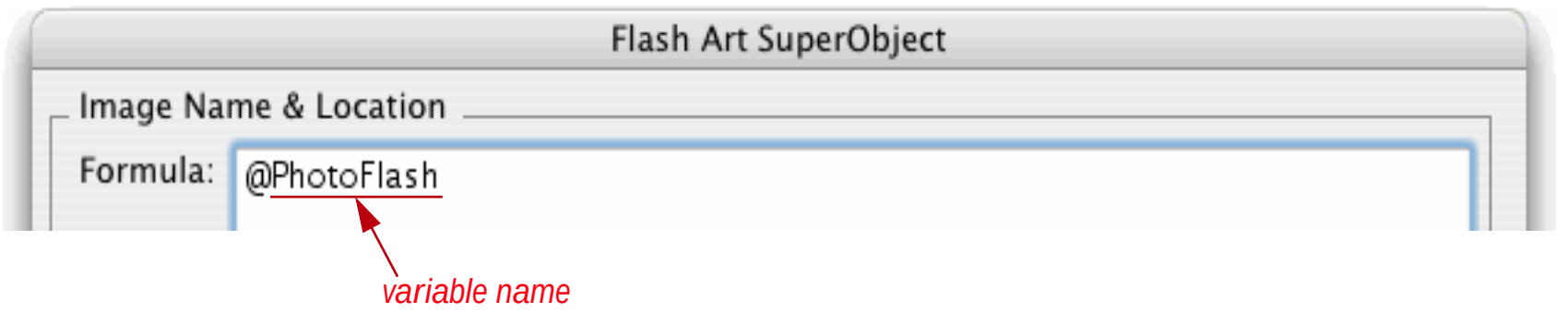
This section of the dialog contains the formula that calculates the image name. The formula may be up to 255 characters long (to create longer formulas, see the next paragraph). Use this formula to combine one or more fields and/or variables into a picture name. See “Formulas” on page 1165 to learn more about Panorama’s formula capabilities.

Formula in a Variable. If the first character of the formula is @ Panorama treats the rest of the line as a variable name instead of a formula. This variable must contain the actual formula for calculating the image name. Using this technique if your formula is over 255 characters (a variable may contain an unlimited number of characters) or if the formula needs to change under different conditions. See “Variables” on page 1199 and “Variables” on page 1370 to learn more about variables.

Before you can use this feature you must set up a variable that contains the actual formula you want to use. Here’s an example that set’s up a variable named PhotoFlash to display JPEG images.

```
fileglobal PhotoFlash
PhotoFlash={PhotoName+".jpg"}
```

The Super Flash Art formula would be set up to use this variable. Remember, to use this option the first character must be the @ symbol.

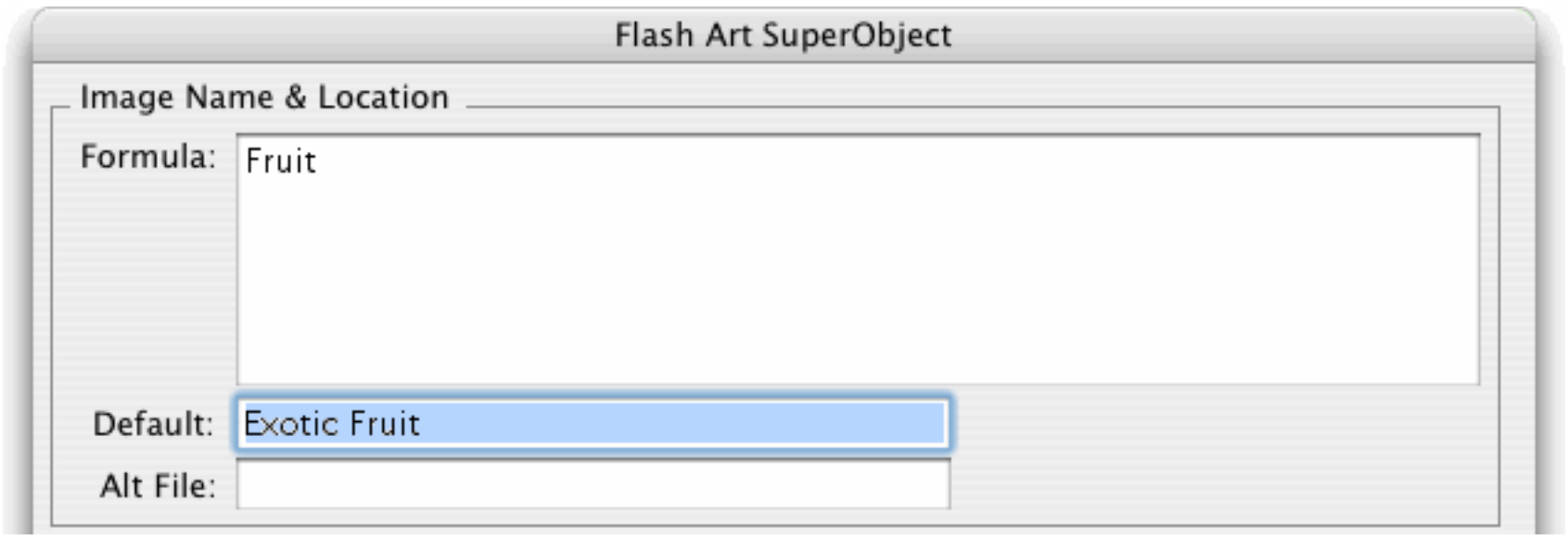


Later you could change the formula to display PICT files like this.

```
PhotoFlash={PhotoName+".pct"}
```

Default

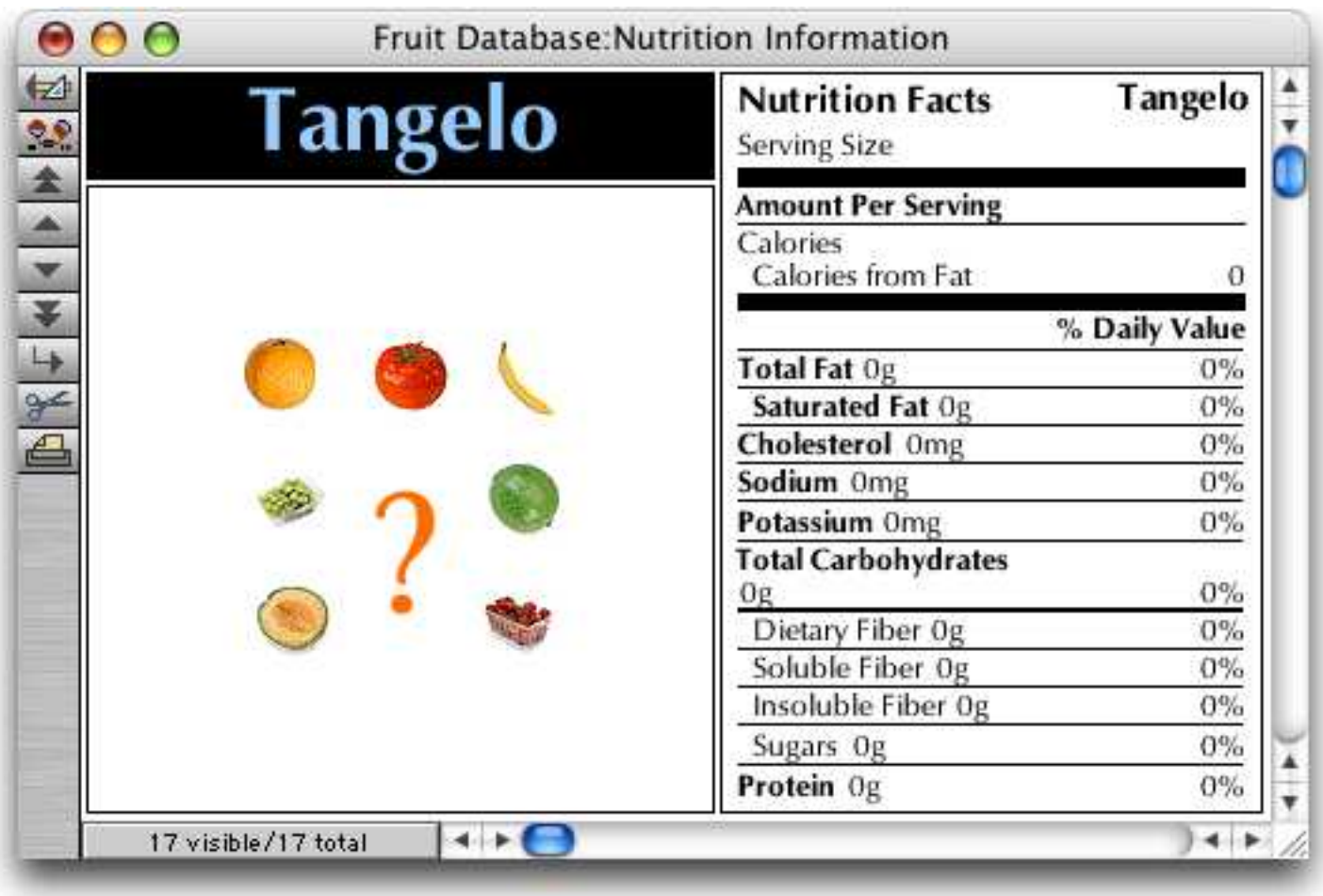
This is the name of the default image that should be displayed if Panorama cannot locate an image with the name specified by the formula. For example, consider the Fruit database used in the previous example. Suppose you encounter a new type of fruit for which you don't have a picture? You could simply leave the image blank. Or, you could create a default image that will be used in these situations, like this.



Now you can specify this image as the default. Notice that no quotes are needed (or allowed!). The default image cannot be changed on the fly — it is not specified by a formula like the main image.



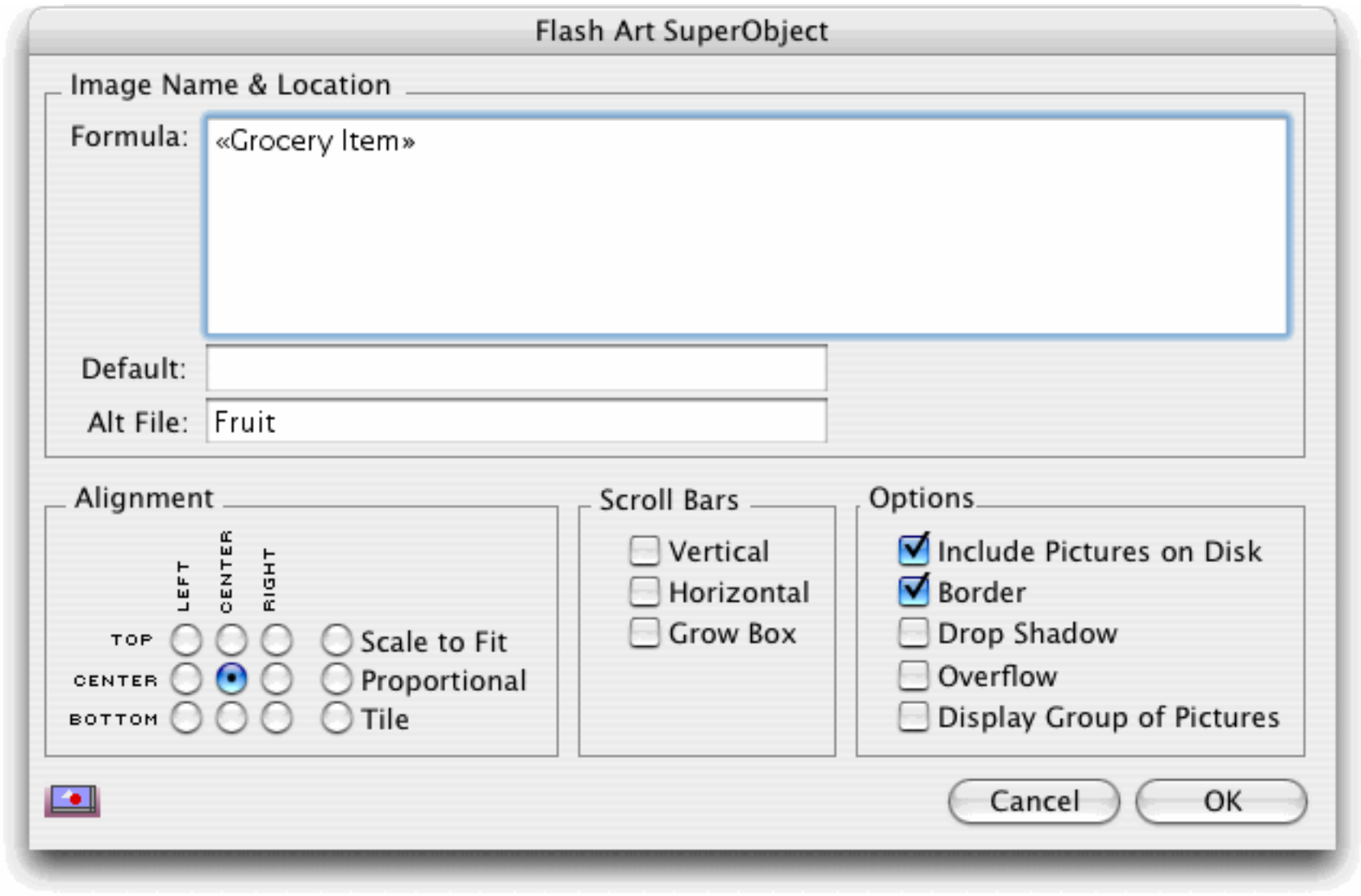
Now if you add a new fruit to the database for which there is no picture, the default image, Exotic Fruit, will be displayed.



Alt File

This option allows one database to share the images in its Flash Art Scrapbook with another database. Both databases must be open for this to work. Usually you would use this option when you have a group of databases that are always used together. All the images could be stored in a single database, saving memory. (This option only applies to images in the Flash Art Scrapbook (see “The Flash Art Scrapbook (Gallery)” on page 762). Images in separate disk files (see “Displaying Images Directly From Disk Files” on page 767) can be accessed by any database at any time.)

For example, suppose you had a Grocery database that was always used with the Fruit database created earlier. A flash art object in the Grocery database can display fruit images in the Fruit database, as long as the Fruit database is open.



Even if the database containing the images is in a different folder, you should only type in the name of the database. Since the database must be open in memory, Panorama doesn't need to know the location of the database on the disk.

Include Pictures on Disk

Check this option if you want images in separate disk files to be displayed. Panorama will check the Flash Art Scrapbook first, then check to see if there is a PICT file with the name specified by the formula. For more information on this option see "Displaying Images Directly From Disk Files" on page 767.

Display Group of Pictures

Enabling this option allows multiple images to be displayed in one spot. The images must be designed to overlay on top of each other. The most common application for this option is to display a map. You start with a base map, then overlay information on top of it. The Display Group option is designed to be used with a summary record (see "3-Step Summarizing" on page 393). It will automatically display all of the images belonging to the data records above the summary record.

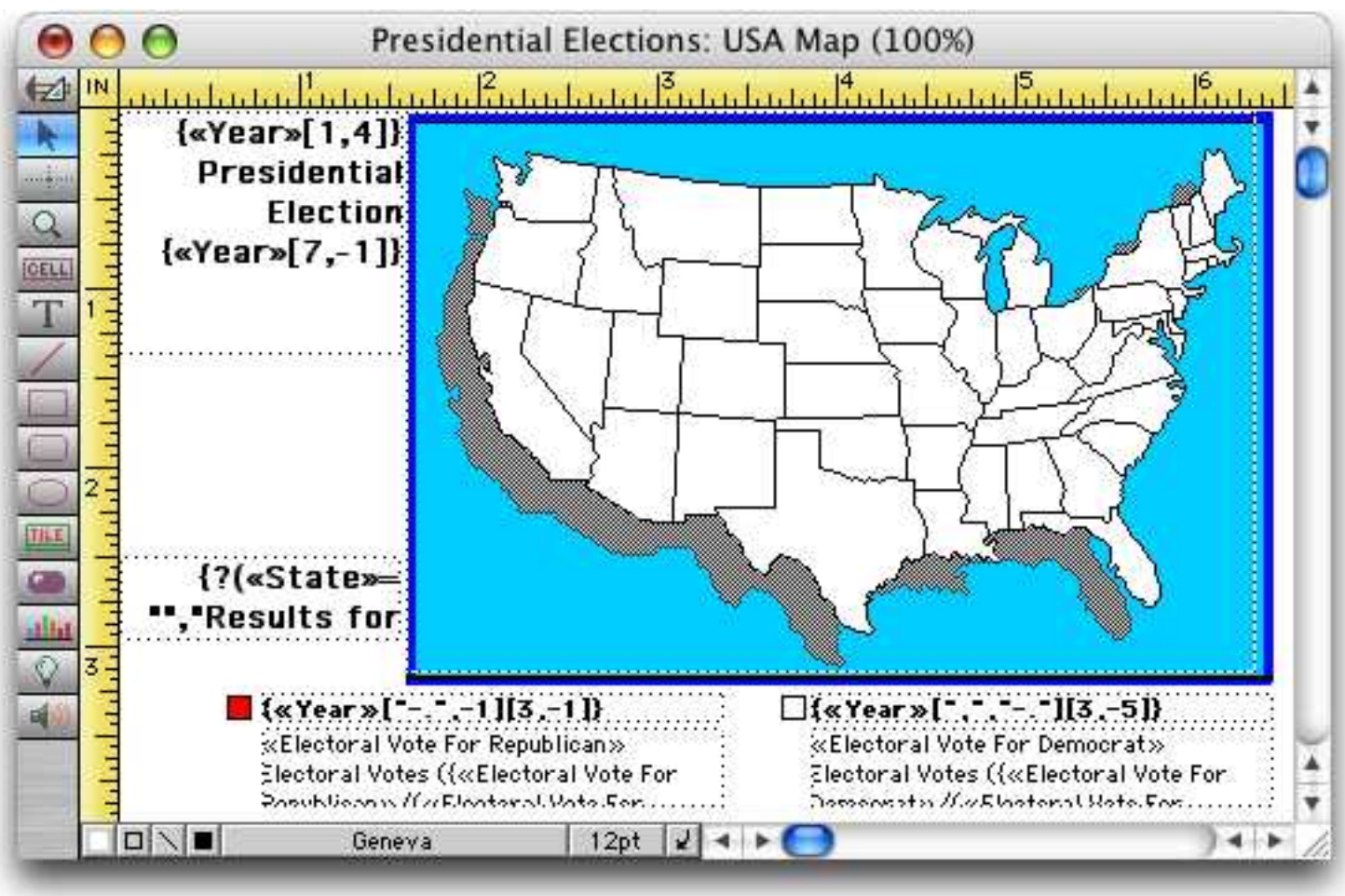
To illustrate this option we'll use a database with presidential election data. The database has been grouped to total up the electoral votes for each state, and for each election.

Presidential Elections

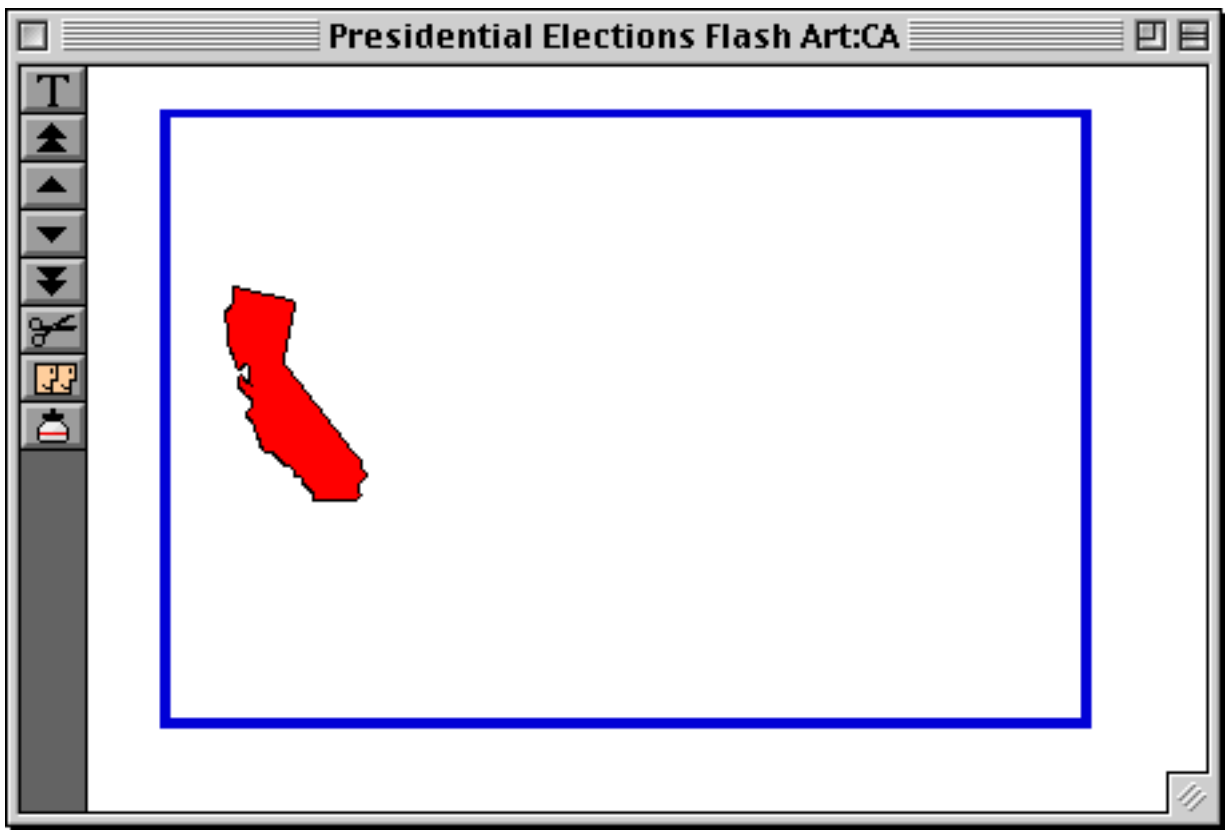
Year	State	Electoral Vote For Democrat	Electoral Vote For Republican	Popular Vote For Democrat	Popular Vote For Republican
1976, Carter vs. Ford	OH	25	0	2,011,621	2,000,505
1976, Carter vs. Ford	OK	0	8	532,442	545,708
1976, Carter vs. Ford	OR	0	6	490,407	492,120
1976, Carter vs. Ford	PA	27	0	2,328,677	2,205,604
1976, Carter vs. Ford	RI	4	0	227,636	181,249
1976, Carter vs. Ford	SC	8	0	450,807	346,149
1976, Carter vs. Ford	SD	0	4	147,068	151,505
1976, Carter vs. Ford	TN	10	0	825,879	633,969
1976, Carter vs. Ford	TX	26	0	2,082,319	1,953,300
1976, Carter vs. Ford	UT	0	4	182,110	337,908
1976, Carter vs. Ford	VA	0	12	813,896	836,554
1976, Carter vs. Ford	VT	0	3	78,789	100,387
1976, Carter vs. Ford	WA	0	8	717,323	777,732
1976, Carter vs. Ford	WI	11	0	1,040,232	1,004,987
1976, Carter vs. Ford	WV	6	0	435,864	314,726
1976, Carter vs. Ford	WY	0	3	62,239	92,717
1976, Carter vs. Ford		**297**	**240**	**40,828,929**	**39,148,940**

104 visible/104 total

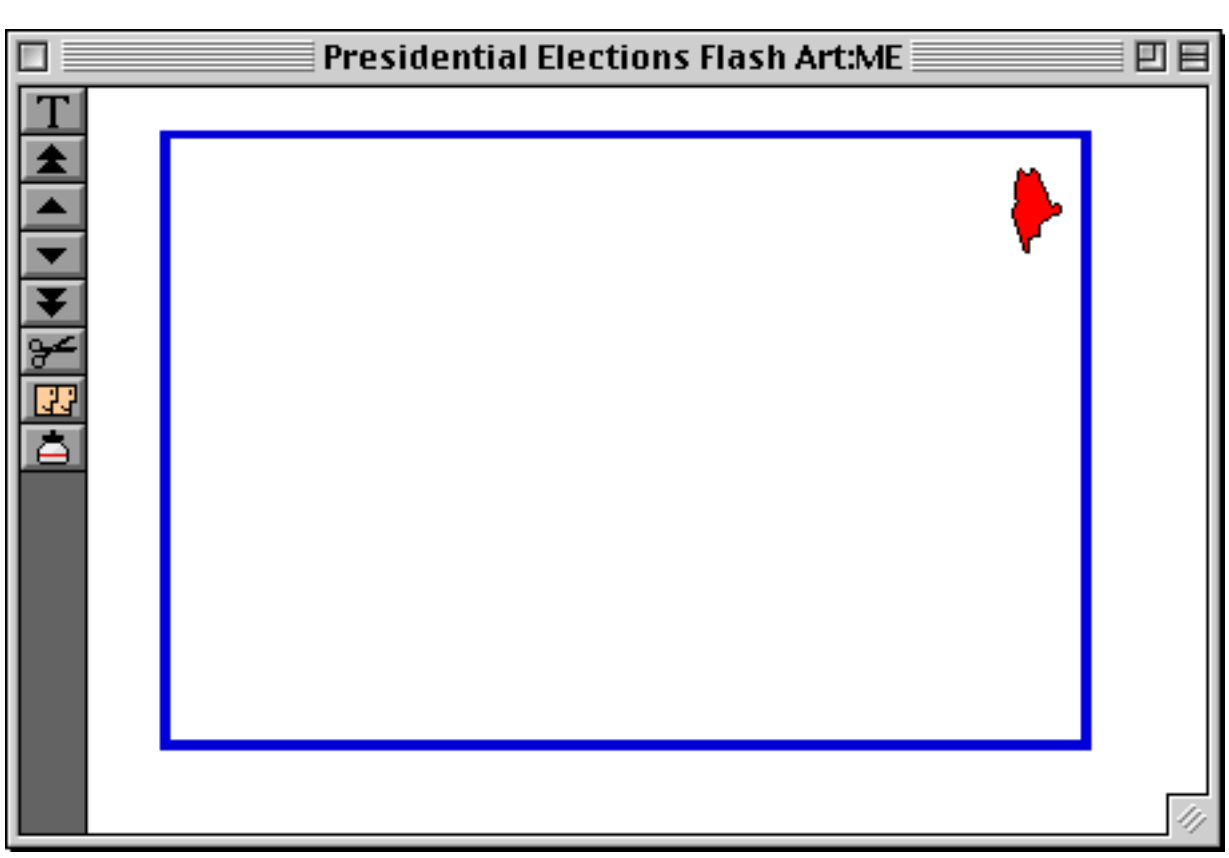
We'll start with a base map of the United States, as shown in the illustration below. This base map is simply a fixed image that has been pasted into the form (see "Fixed Images" on page 739).



To create our final map we'll need an overlay map for each of the 50 states. The overlay map must be designed to fit right on top of the base map. Here's the overlay map for California.

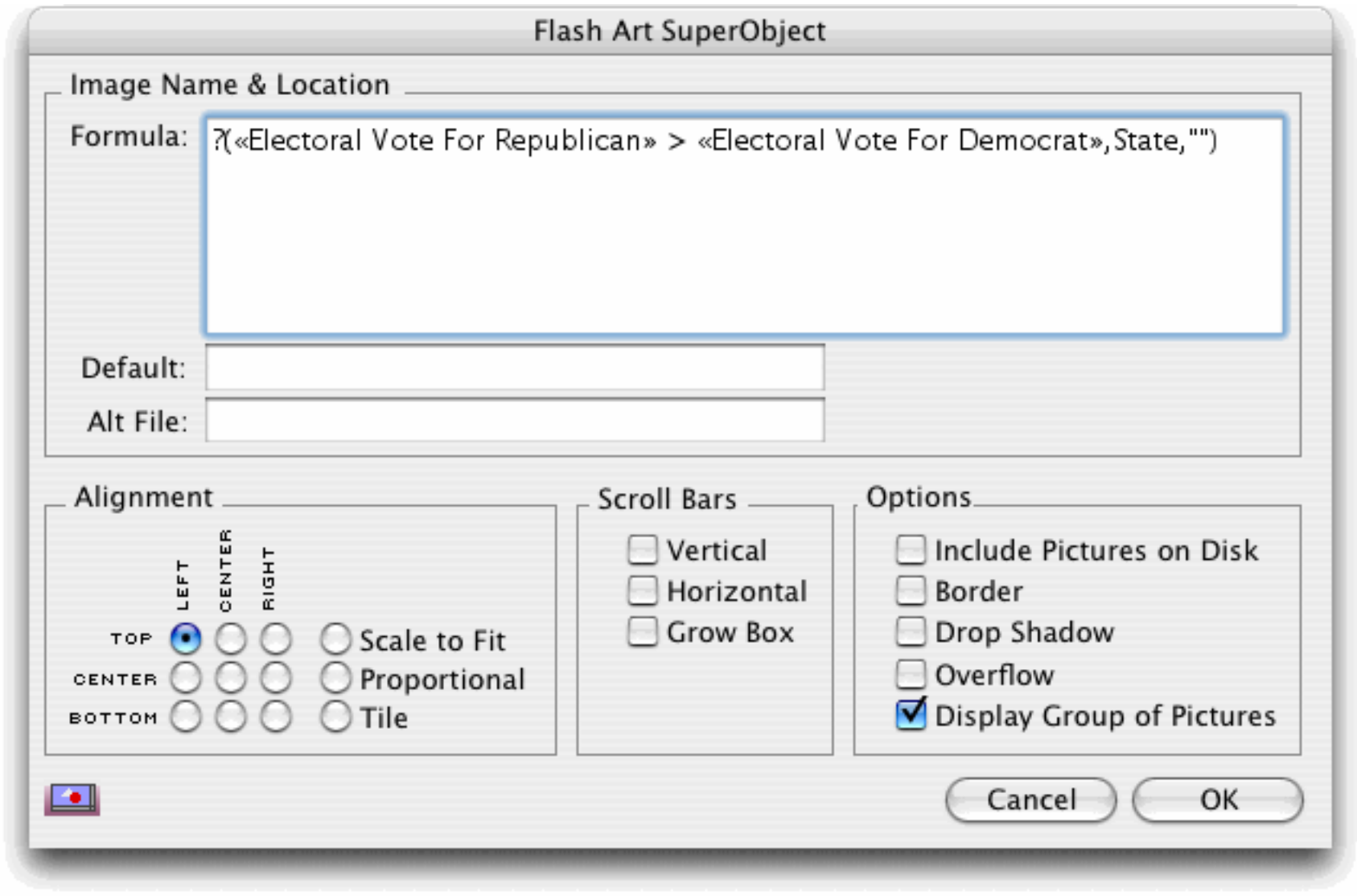


Here's the overlay map for Maine.

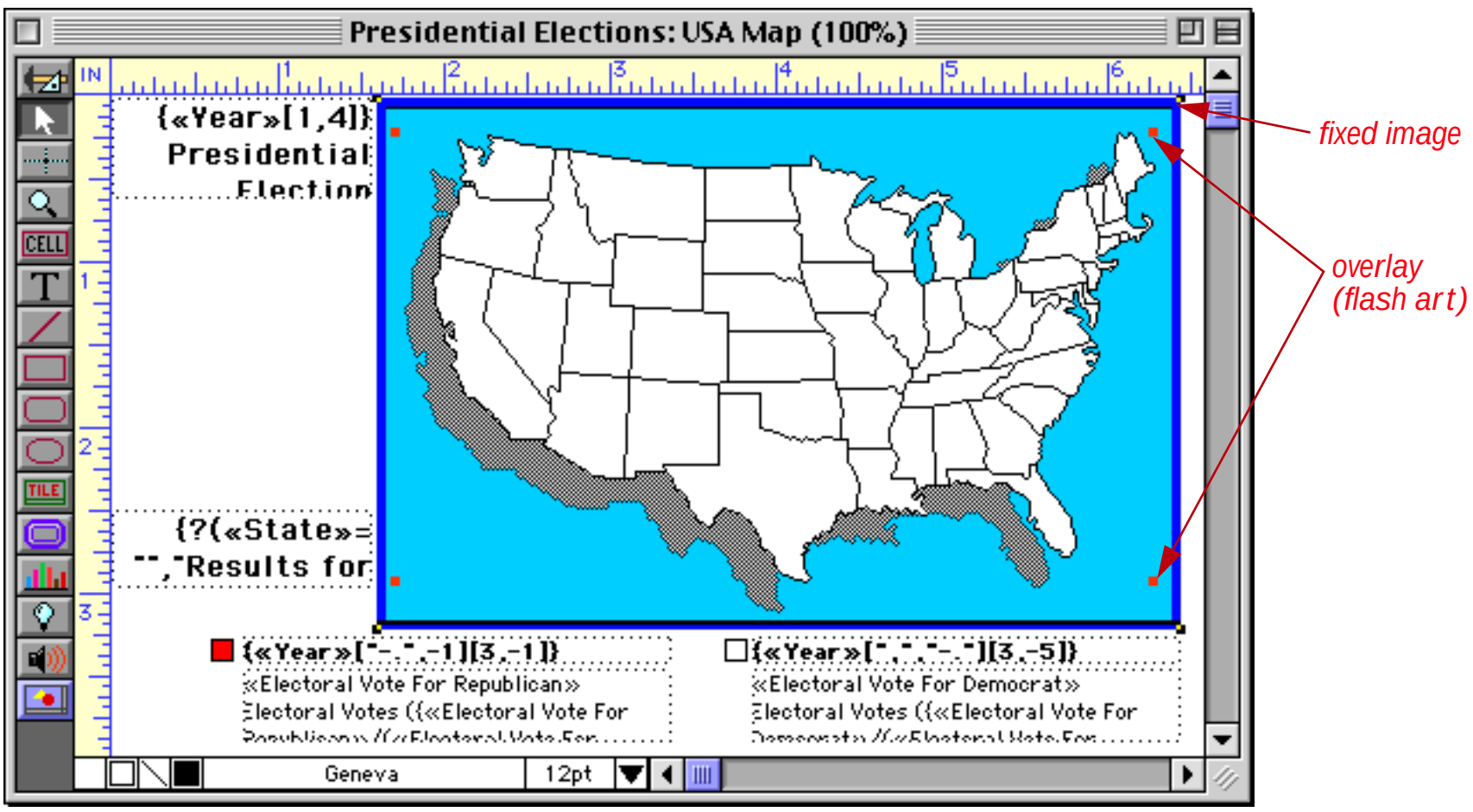


It can take a lot of work to set up the overlay maps! Once they are complete, you are ready to build a composite map — in this case showing the results of the presidential election.

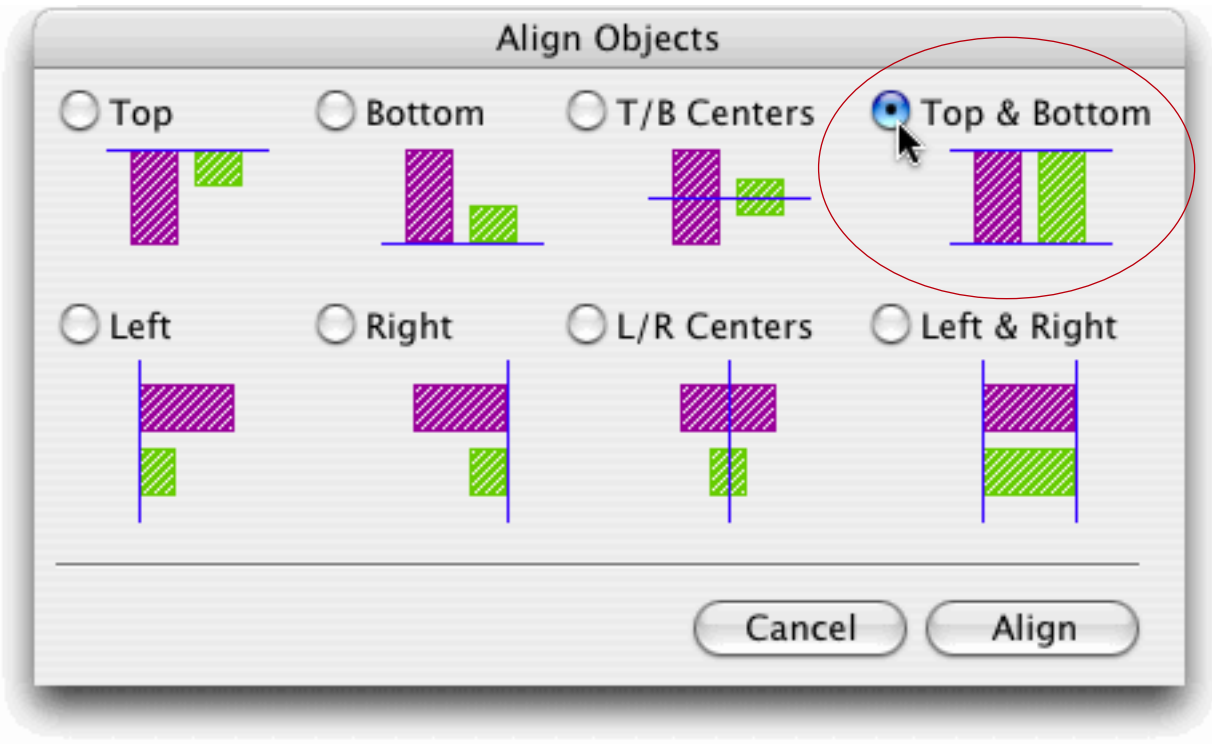
To build the composite map you need to create a Super Flash Art object directly on top of the base map. The Super Flash Art Object must exactly match the position and size of the base map, so that the overlay maps will line up with the base map. Here's the configuration dialog for this Super Flash Art object. The formula uses the ?(function to decide whether or not to display the overlay map for a particular state (see "The ? Function" on page 1271). If the electoral vote for the Republicans is greater than the vote for the Democrats, the overlay is displayed (so in this case the state will be displayed in red). If the Democrat vote is larger, the overlay is not displayed (so the state remains in white).



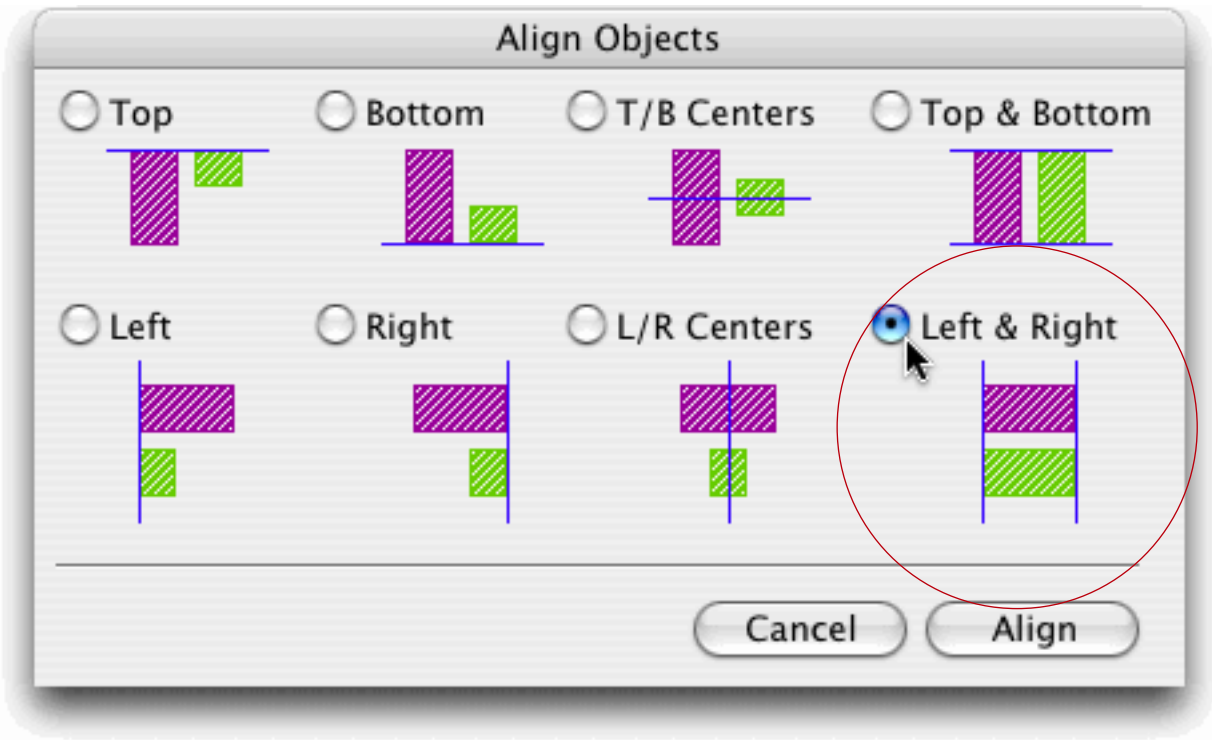
It can be tricky to get the overlay Flash Art SuperObject to line up exactly with the fixed map. Here's an easy way to do it. Start by selecting both of the objects (see "Selecting Multiple Objects at Once" on page 502).



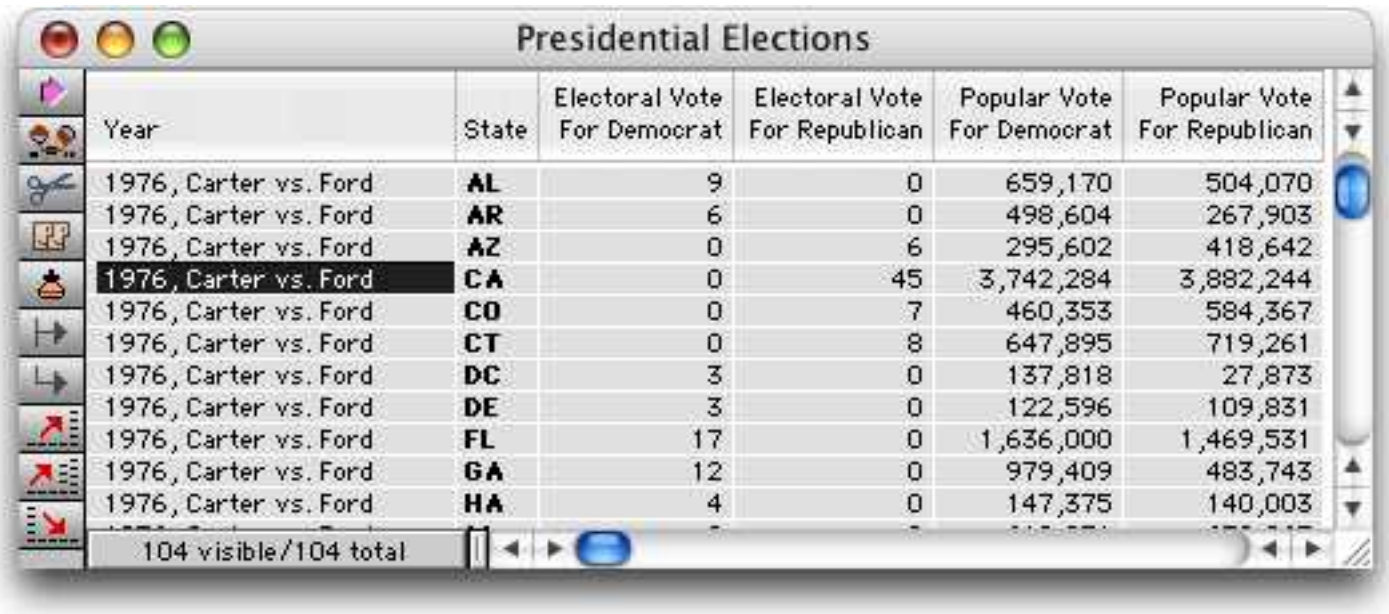
Now use the **Align Objects** command to line up the objects (see "Aligning Objects" on page 553). Start by aligning the top and bottom…



Then repeat the **Align Objects** command to line up the left and right edges.



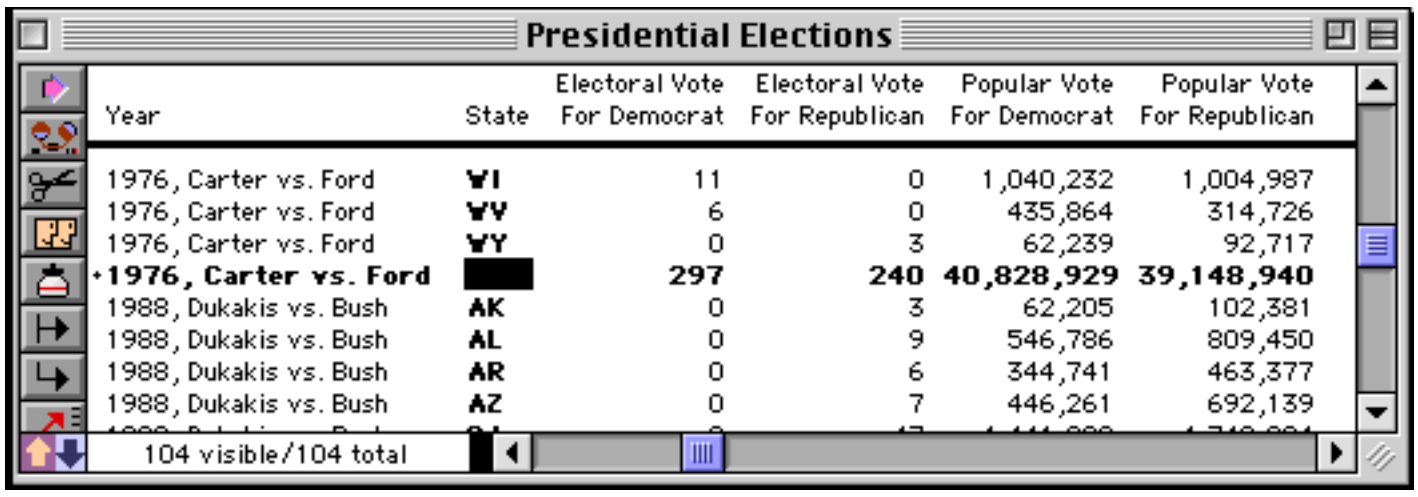
Now let's see what our composite map looks like. Using the data sheet, we move to a state that was won by the Republicans.



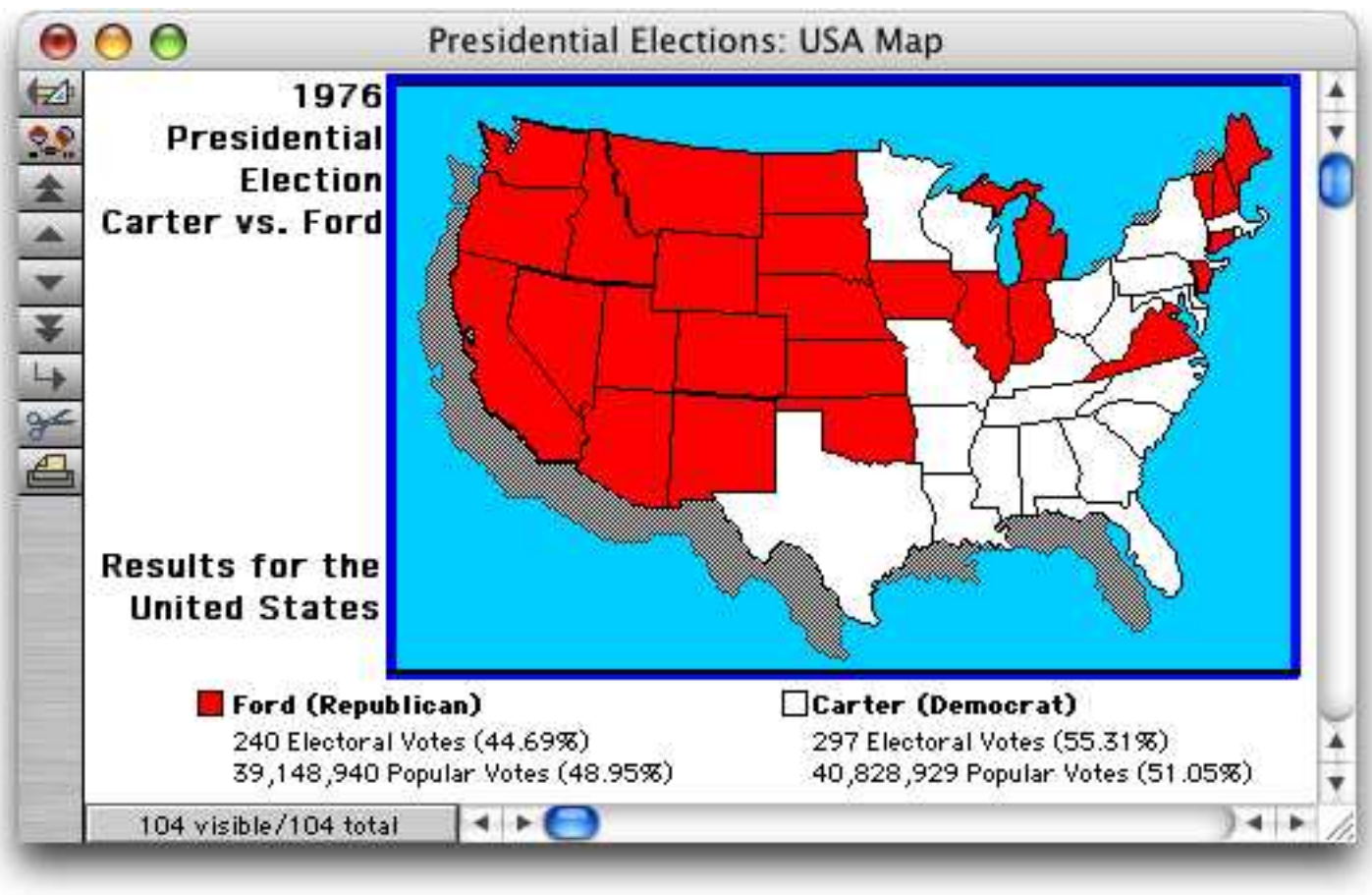
The formula tells Panorama to display the overlay map for this state.



You can move from state to state, viewing the results for each state. When you get to the summary record, however, something different happens. Here's the summary record.



Now here's the map. Because the Display Group of Pictures option is enabled, Panorama displays the overlay map for every state with a Republican victory that year. You can easily see that Gerald Ford swept the west, while Jimmy Carter took the South.



A really powerful combination is to create multiple overlay maps with different colors and/or gray levels, and then select the appropriate color using the formula. For example you could use different colors to indicate different market penetration levels, or pollution levels, etc.

Border

If this option is enabled, Panorama will draw a 1 pixel border around the Super Flash Art object.

Drop Shadow

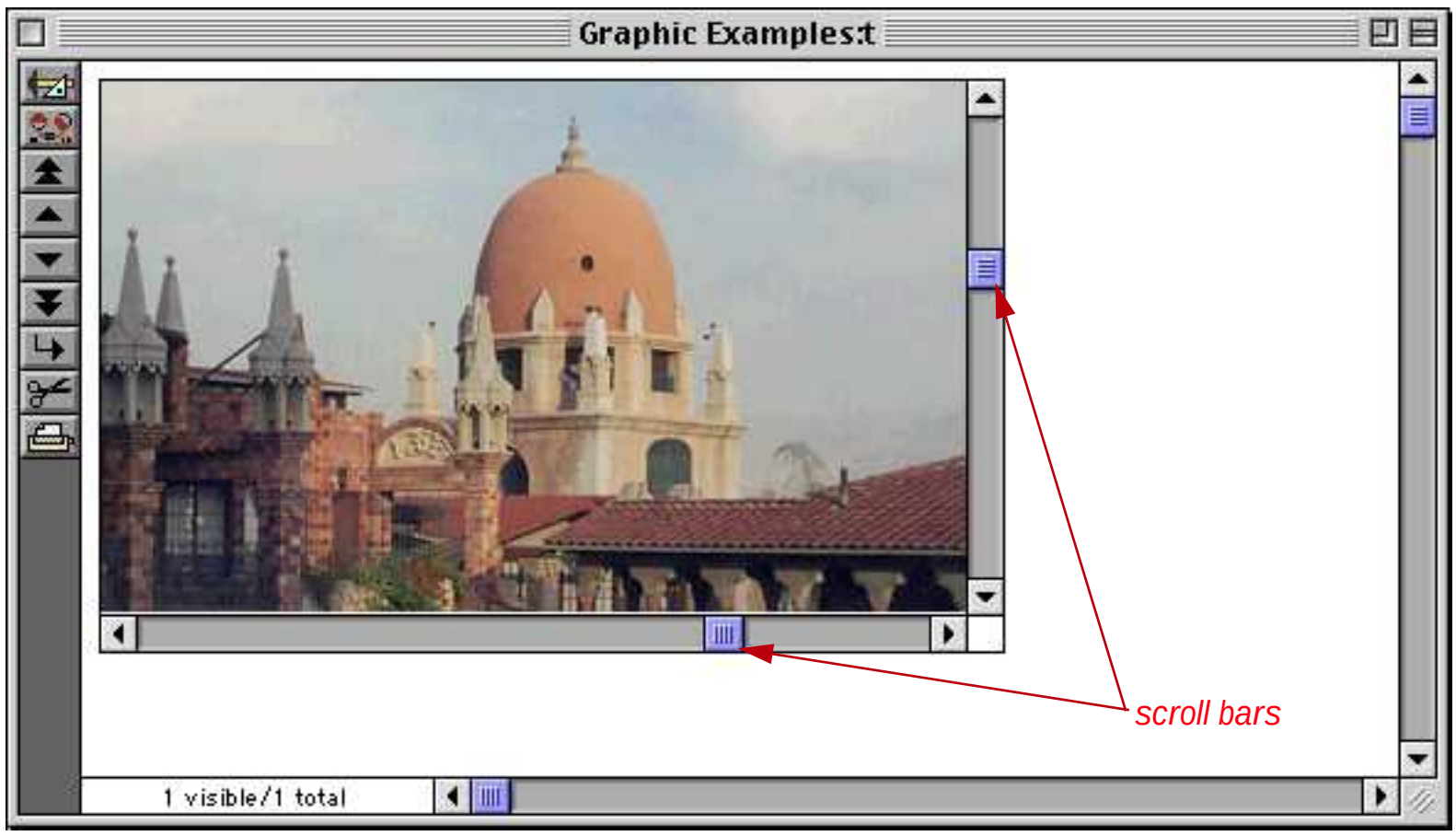
If this option is enabled, Panorama will draw a simulated drop shadow slightly below and to the right of the Super Flash Art object.

Overflow

This option is used in conjunction with an overflow tile for printing images that are more than one page high. See “Printing Data that Overflows a Page” on page 1104 for more information on multi-page overflow printing.

Scroll Bars

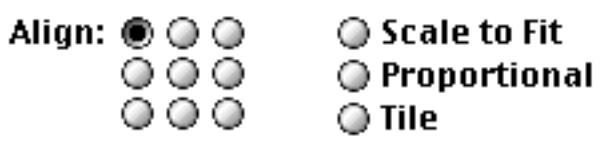
If you expect that the images you will be displaying may be too large to fit into the dimensions of the Super Flash Art object, you can enable scroll bars that will allow the user to shift around and view different parts of the picture. You can enable a Vertical scroll bar, a Horizontal scroll bar, or both.



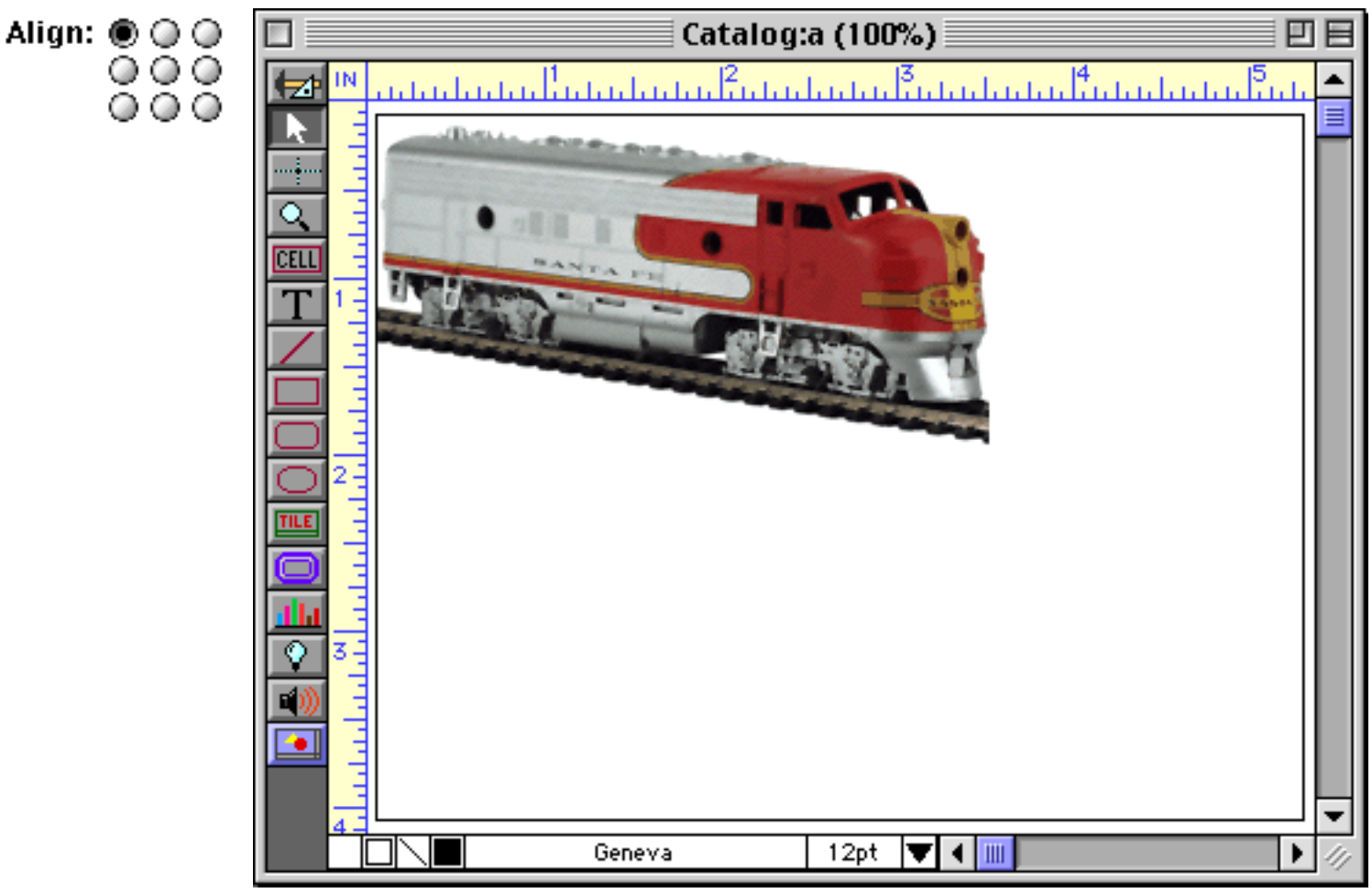
You also have the option of leaving space for a Grow Box in the lower right hand corner of the Super Flash Art object. The Grow Box is simply a 16 by 16 pixel box that is left empty. It’s up to you to draw an icon in this area (if you want). If the bottom right hand corner of the Super Flash Art object is in the same position as the bottom right hand corner of the window, you can disable the window’s normal scroll bars and use the Super Flash Art object’s scroll bars and Grow Box instead (see “Elastic Forms” on page 912).

Align

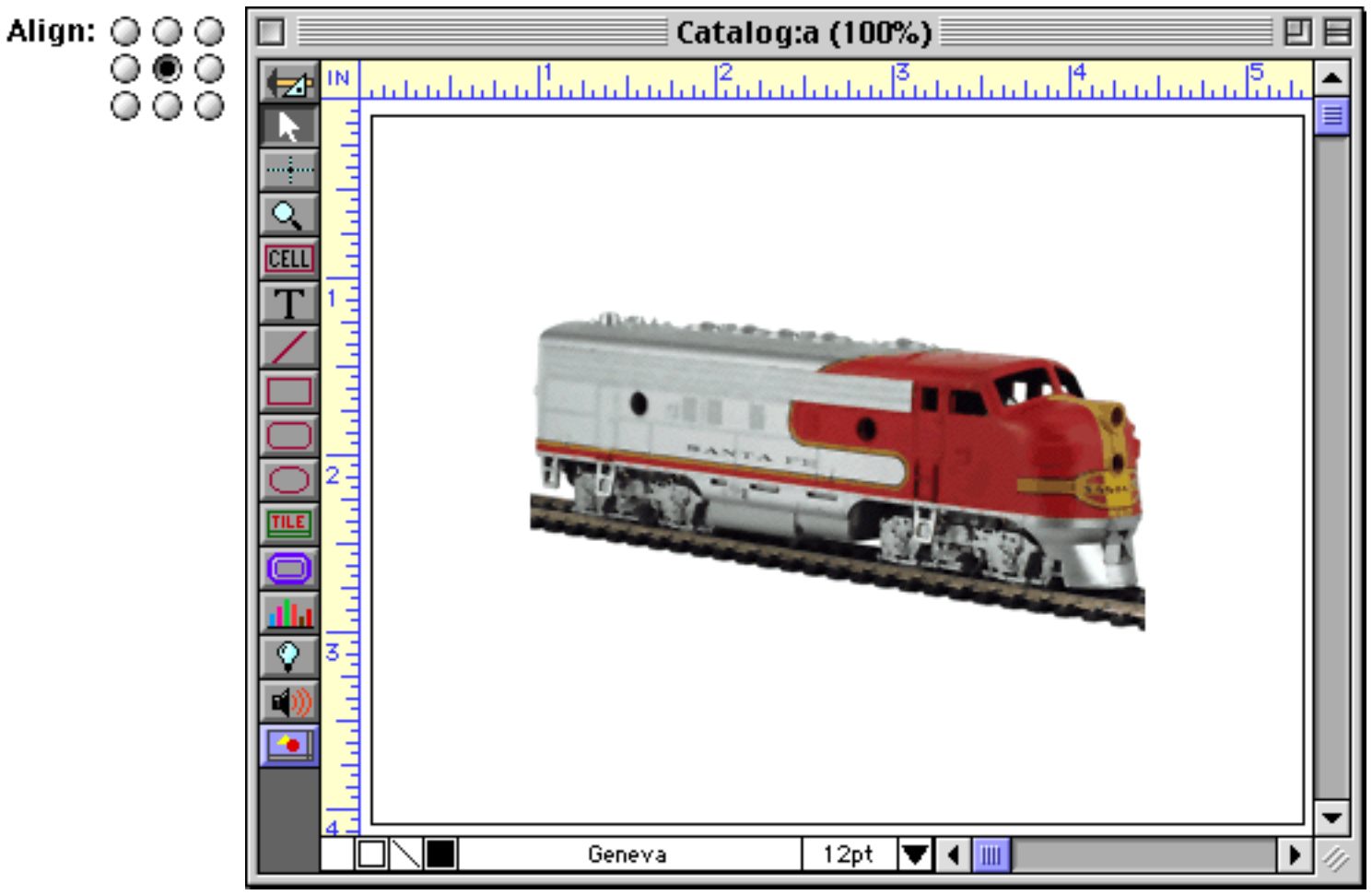
The dimensions of the picture being displayed often do not exactly match the size of the Super Flash Art object. This section of the dialog specifies how the picture should be adjusted if it is too large or too small for the Super Flash Art object.



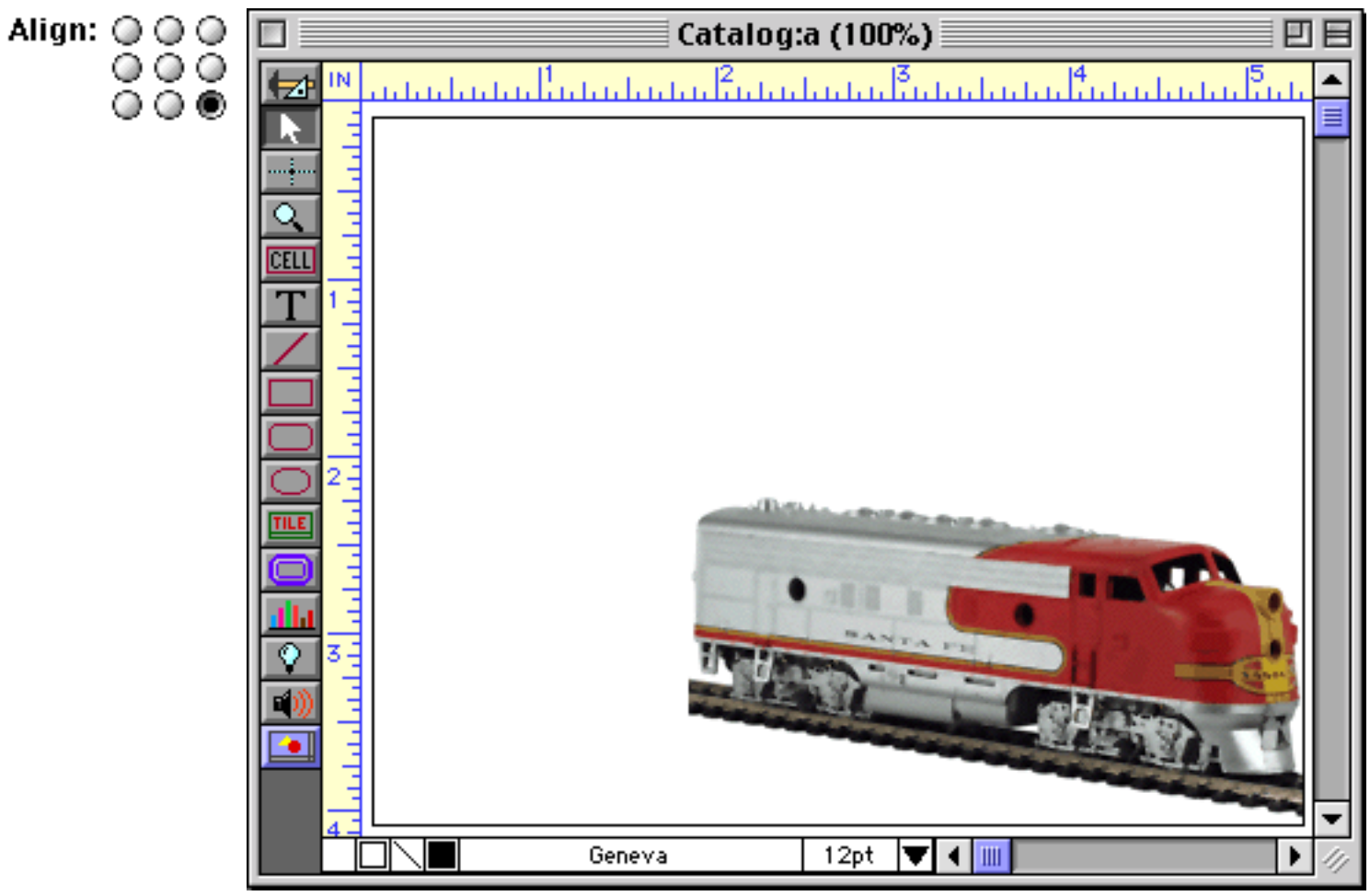
The left side of this section contains nine radio buttons in a tic-tac-toe arrangement. These buttons allow the picture to be aligned within the Super Flash Art object. The picture can be aligned with any corner, centered on any side, or centered in the middle of the object. Wherever the object is aligned, it will be displayed actual size; i.e. it will not be enlarged or reduced. For example, here is an image displayed in the top left corner.



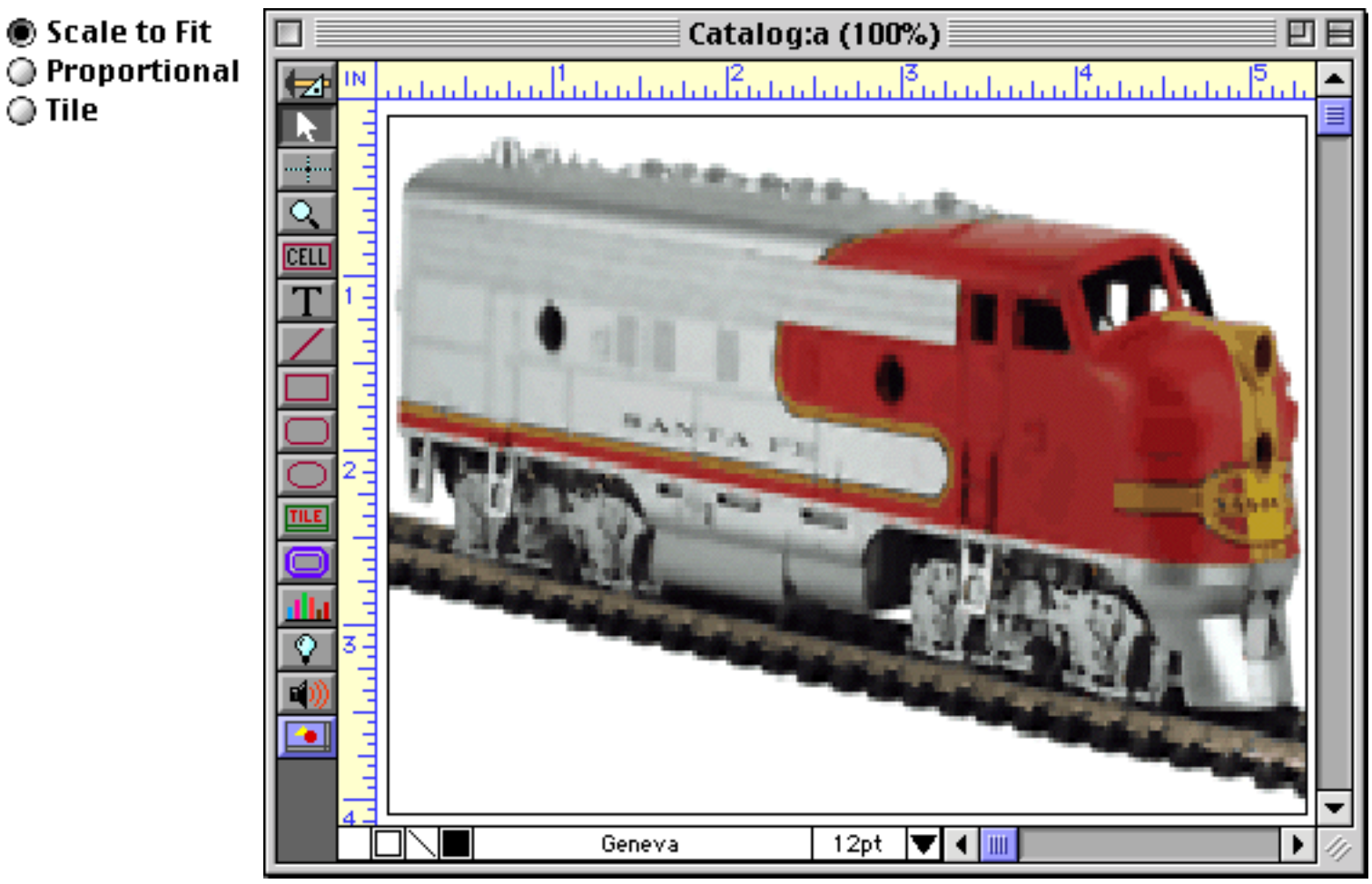
Here is the same image displayed in the middle center.



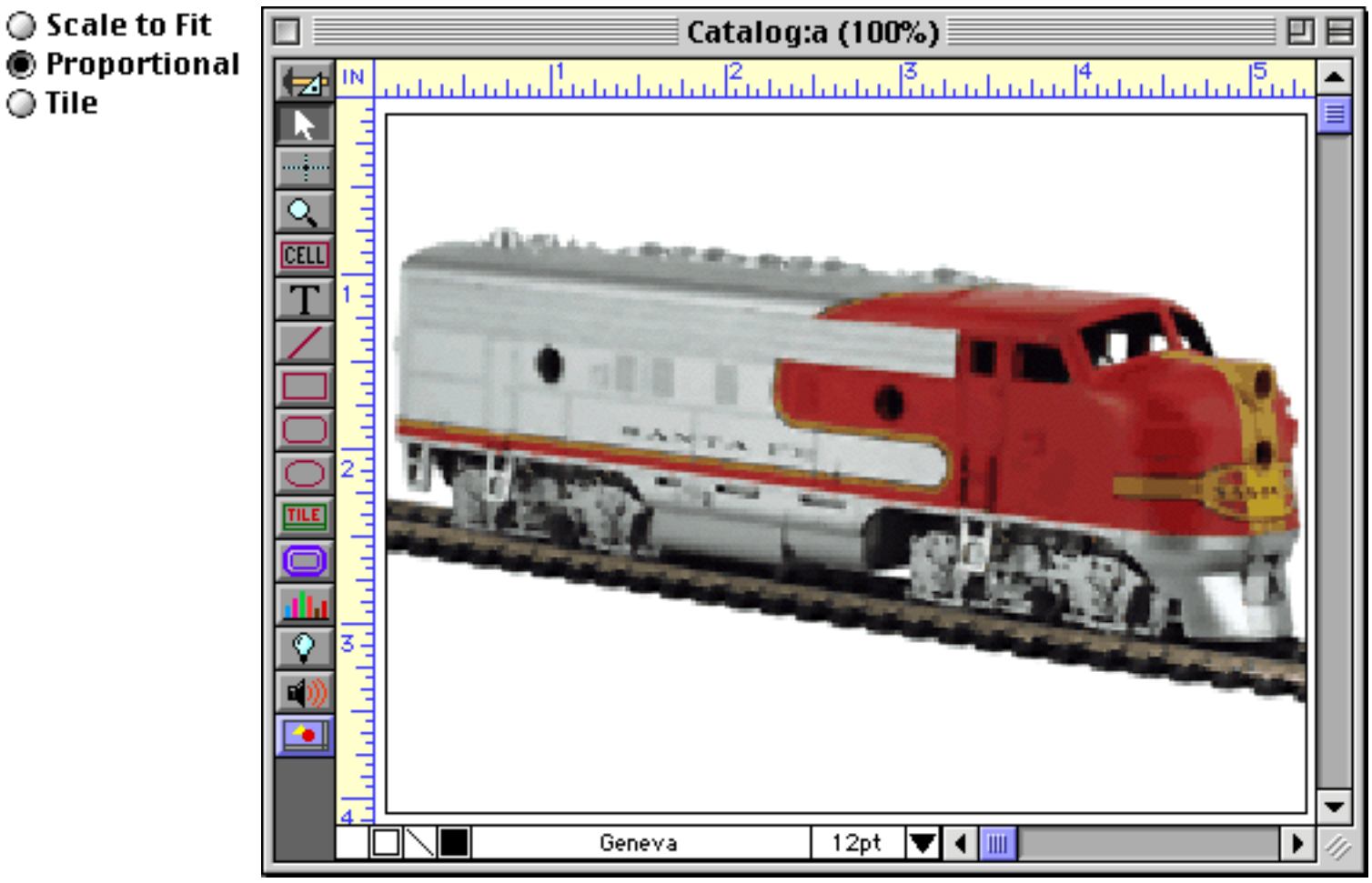
And again in the bottom right.



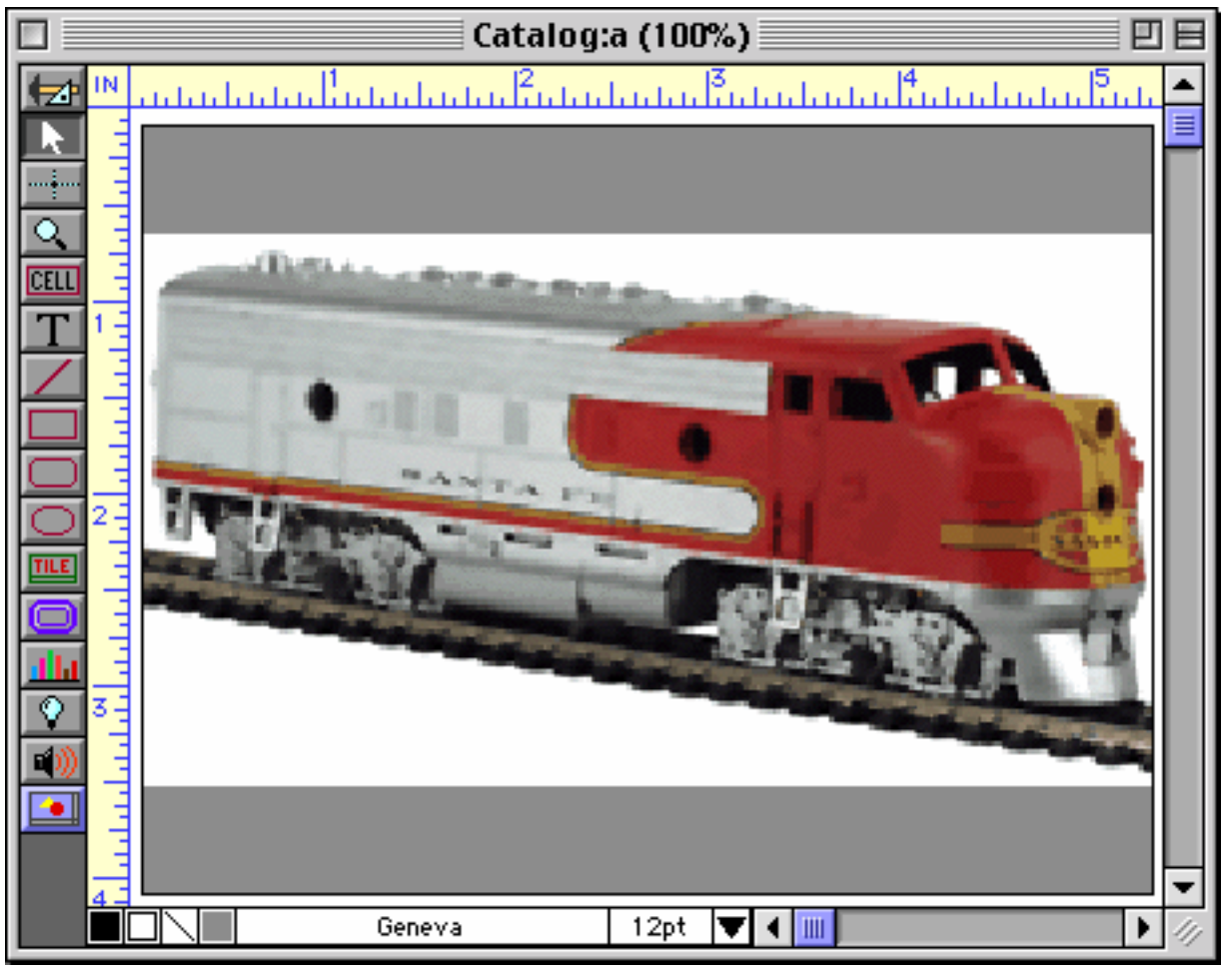
The Scale to Fit option will enlarge, reduce, and / or stretch the picture so that it exactly fits in the Super Flash Art object. The picture may be distorted to make it fit, as you can see in this illustration.



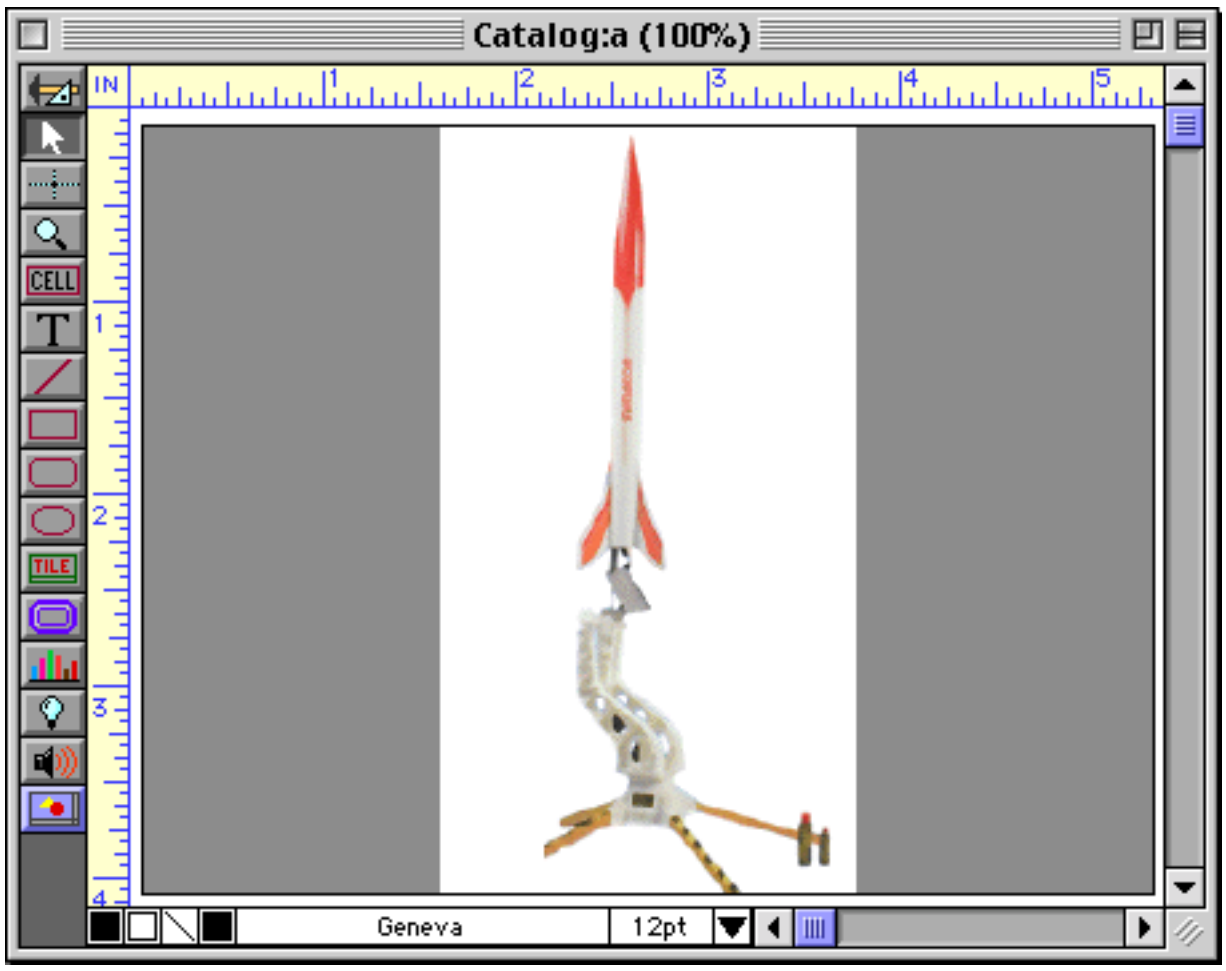
The Proportional option will enlarge or reduce the object as much as possible so that it will fit into the object, but it will not distort the picture. In other words, it will not change the proportions of the picture.



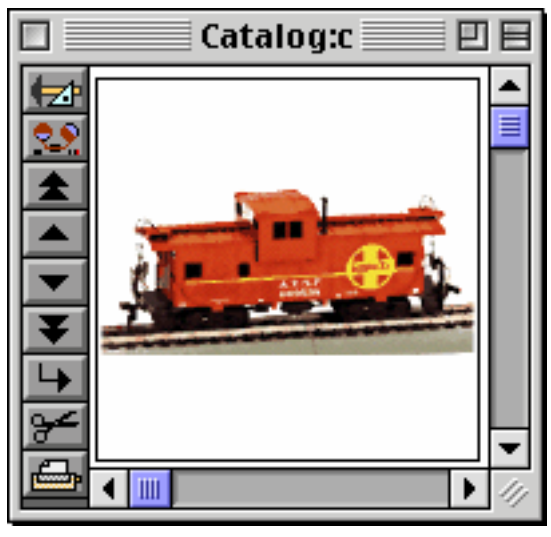
If the proportions of the original picture do not match the proportions of the Super Flash Art object, Panorama will leave a border along the top and bottom or left and right. We've added a gray background to this example so you can clearly see the borders. (The gray background is simply a solid rectangle object placed behind the Flash Art SuperObject.)



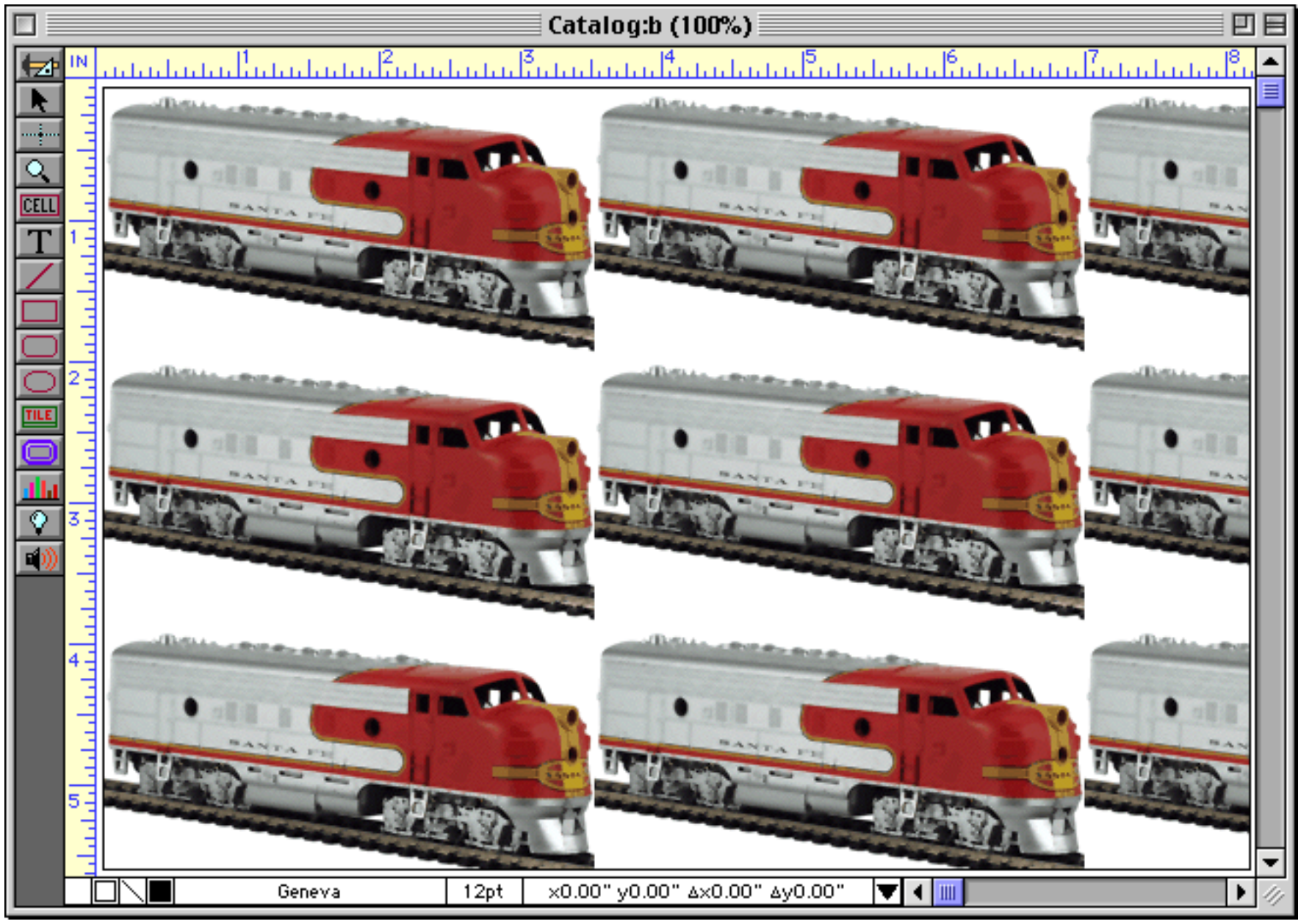
Depending on the aspect ration of the image (width vs. height) the borders may also be on the sides instead of the top and bottom, like this.



Keep in mind that the Scale to Fit and Proportional options may be used to reduce an image as well as to enlarge it. Here's a typical reduced image that has been scaled down with the Proportional option.



The Tile option displays the picture over and over again in a tile arrangement, starting from the upper left. This option allows you to cover a large area with a small picture.



The only disadvantage of this technique is that if the original picture is really small there will be a perceptible delay as the tile pattern is drawn.

Displaying Images from Resource Files

Super Flash Art normally displays images from the Flash Art Gallery or directly from disk files. However, images can also be displayed from resource files. An image can be placed in a resource file using a program like ResEdit or Resourcerer. Before an image can be displayed from a resource file the file must be opened with the `openresource` statement. See “Working with Resources” on page 1560 to learn how to do this.

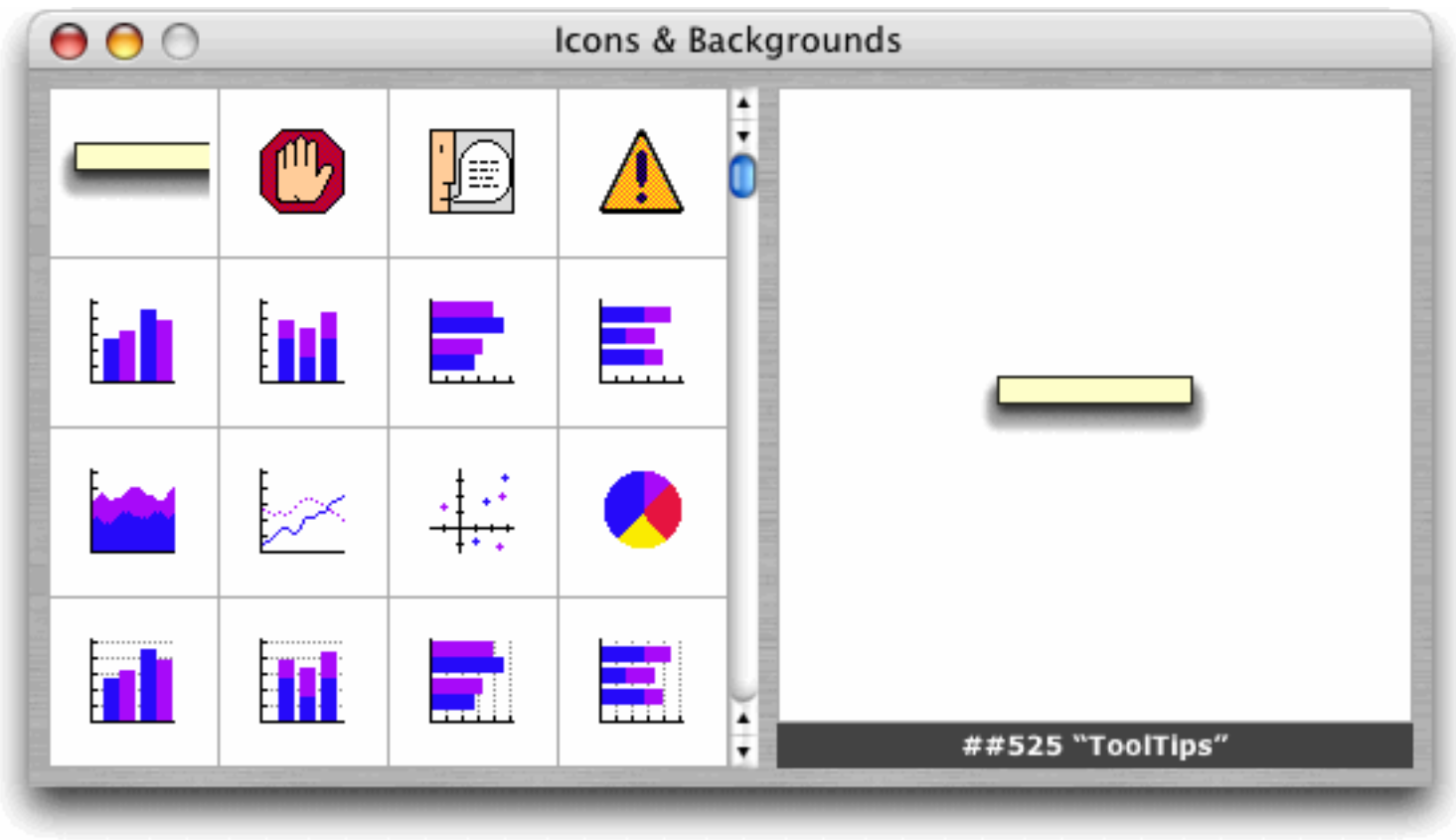
To display an image from a resource file you must use a special prefix as part of the Super Flash Art formula. If the formula begins with `//`, the rest of the formula is treated as the name of the resource to be displayed. For example, this formula would display the resource picture named Blue Sky.

```
"//Blue Sky"
```

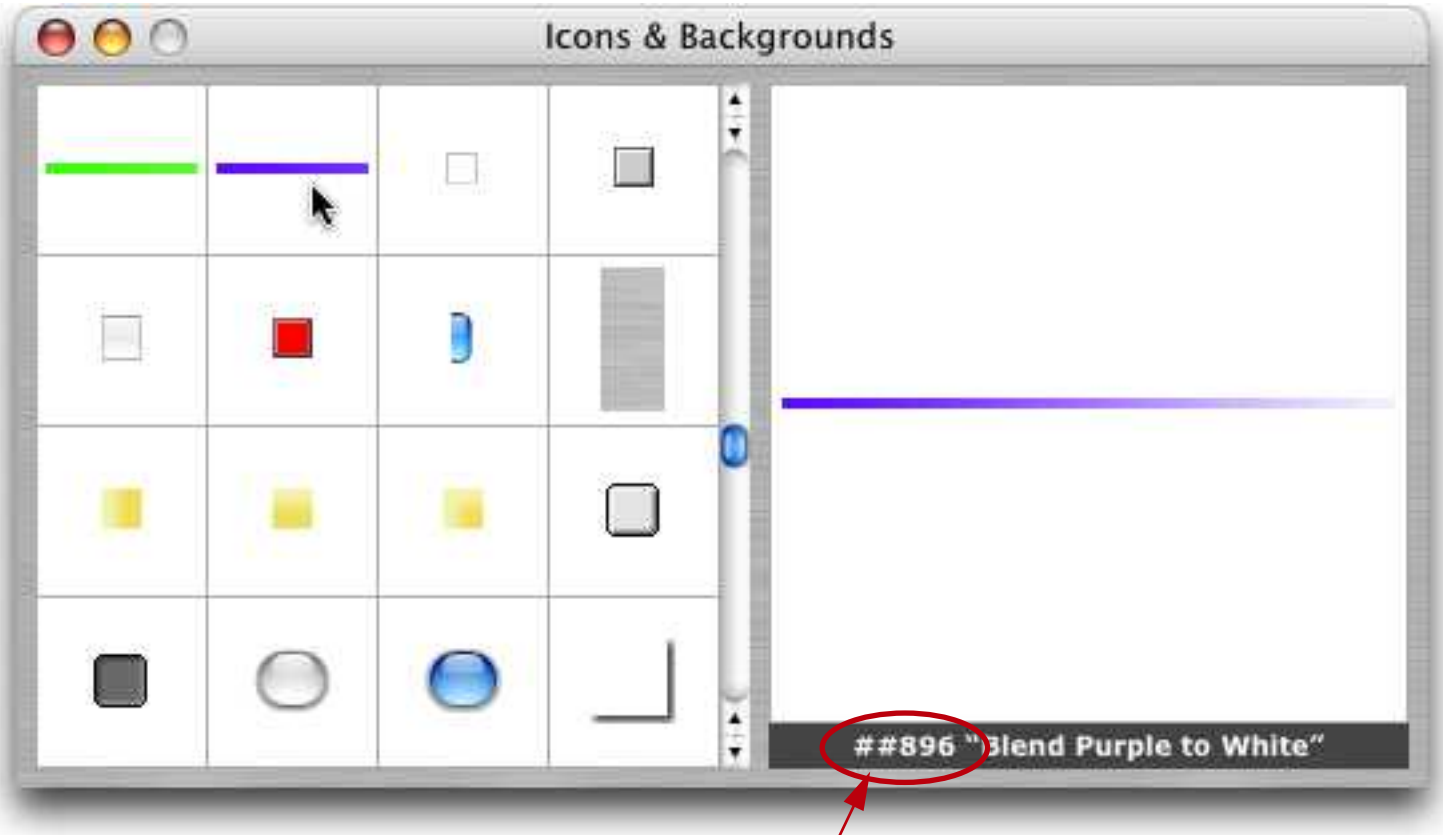
If the formula begins with ##, the rest of the formula is treated as the number of the resource to be displayed. For example, this formula would display the resource picture number 4387.

```
"##4387"
```

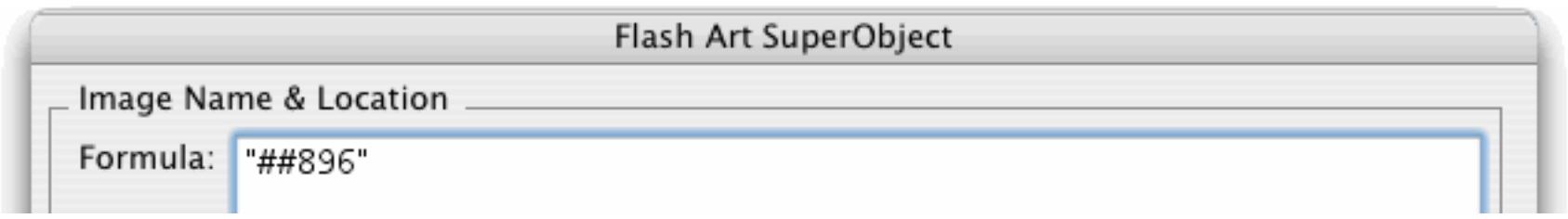
Panorama itself includes a number of resource based images. Most of these are used by Panorama itself in various windows and dialogs, but they are available for use in your databases also. To see a list of these images open the **Icons & Backgrounds** wizard in the Form Tools submenu of the Wizard menu.



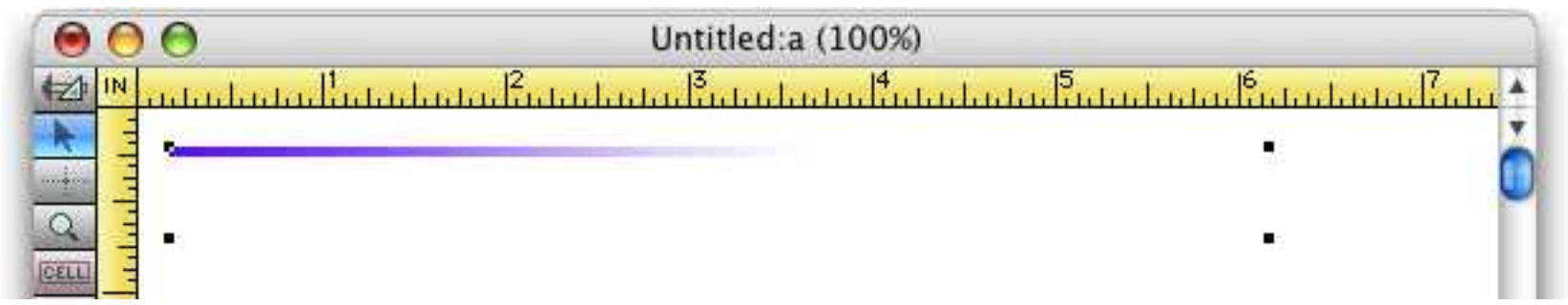
On the left is a thumbnail matrix of images. Click on any image to see an enlarged view on the right, along with the resource number.



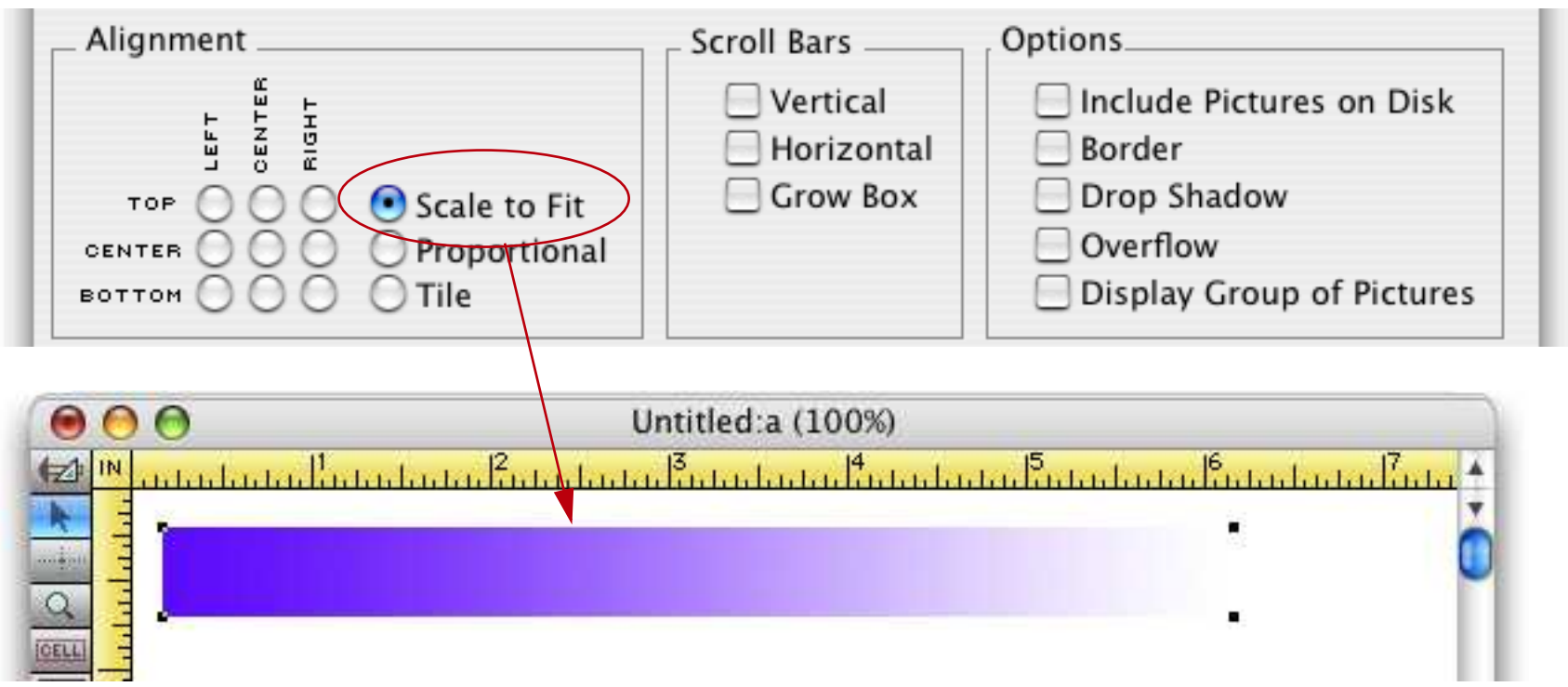
Here is the formula for displaying this purple to white blend:



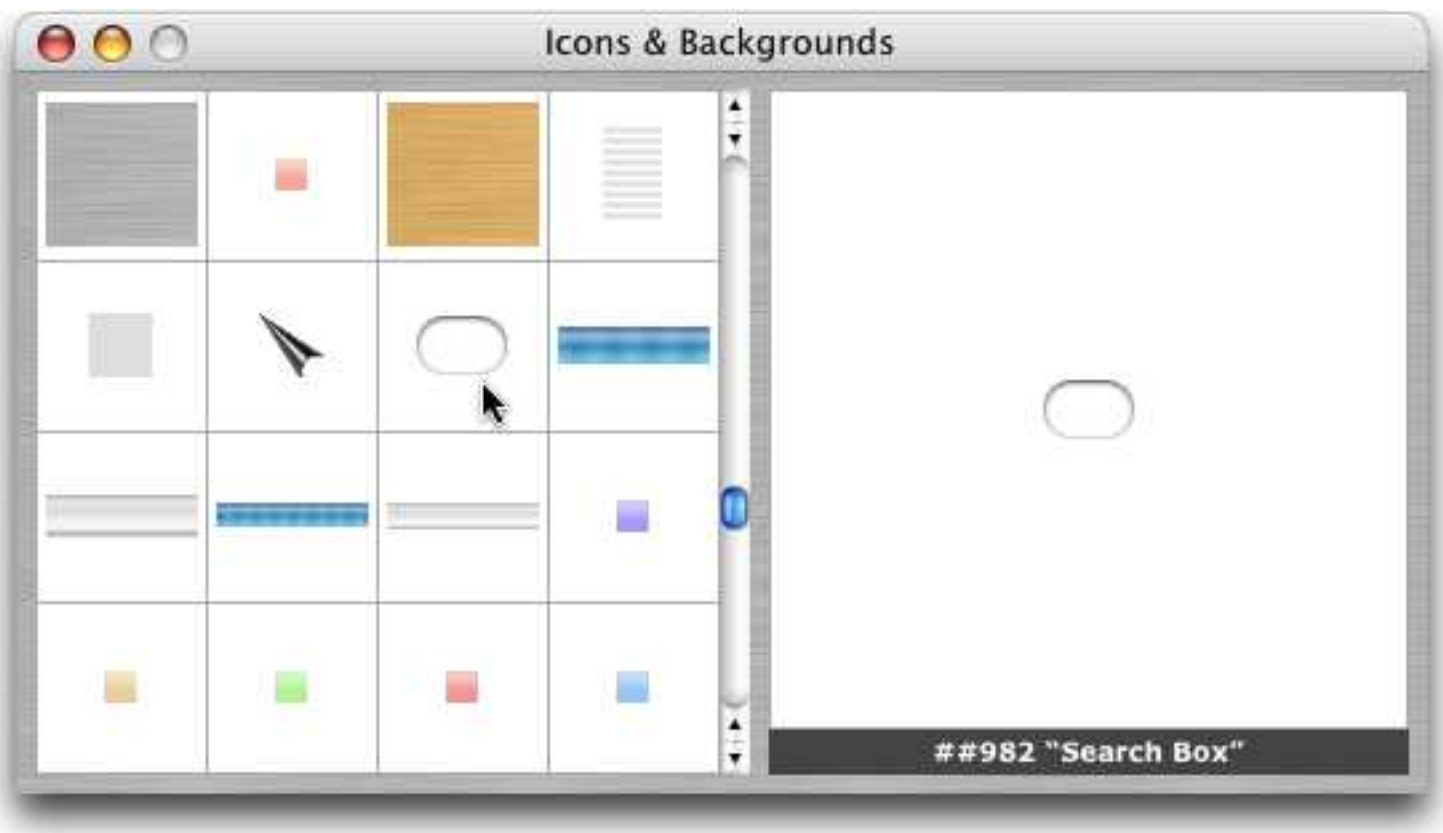
Here's what this image looks like in a form.



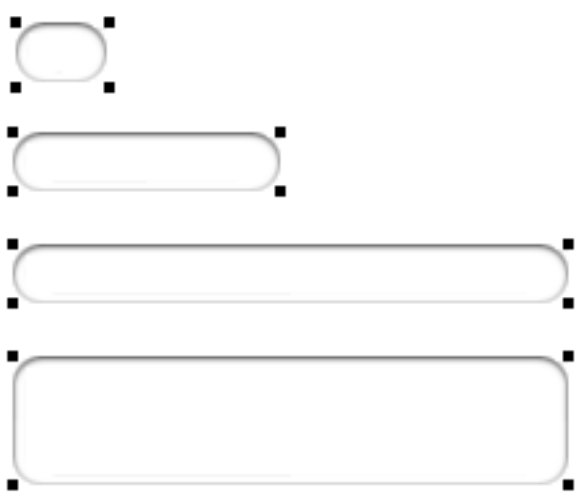
Many of these standard images can be stretched to fit different sizes. The object above is displayed with the alignment set to top left. Changing the alignment to Scale to Fit gives a expands the blend to fit a larger area.



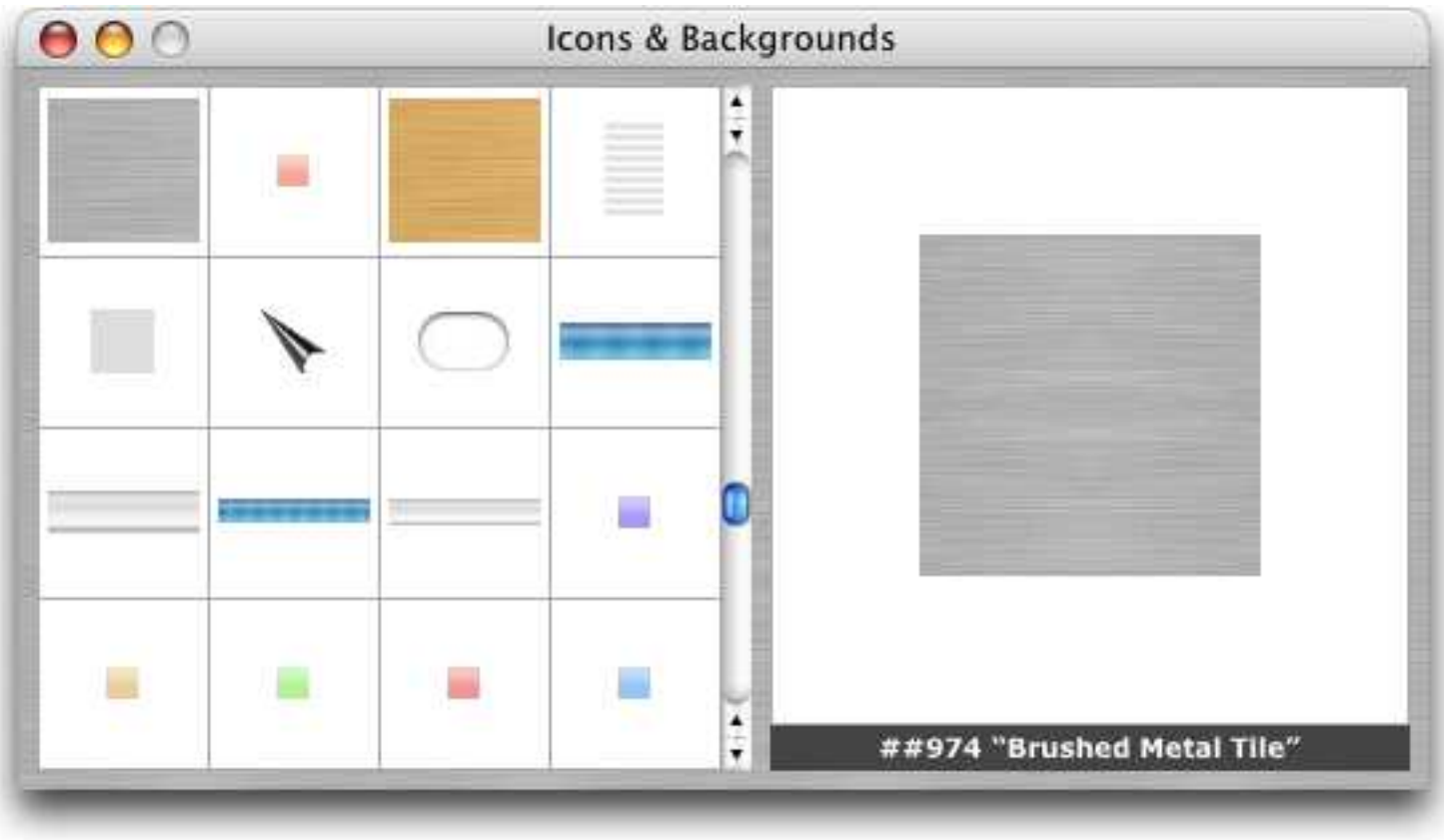
Some of the standard images are elastic (see “Elastic Pictures” on page 800). This means that they can be stretched without distortion. For example, consider this search box image (##982).



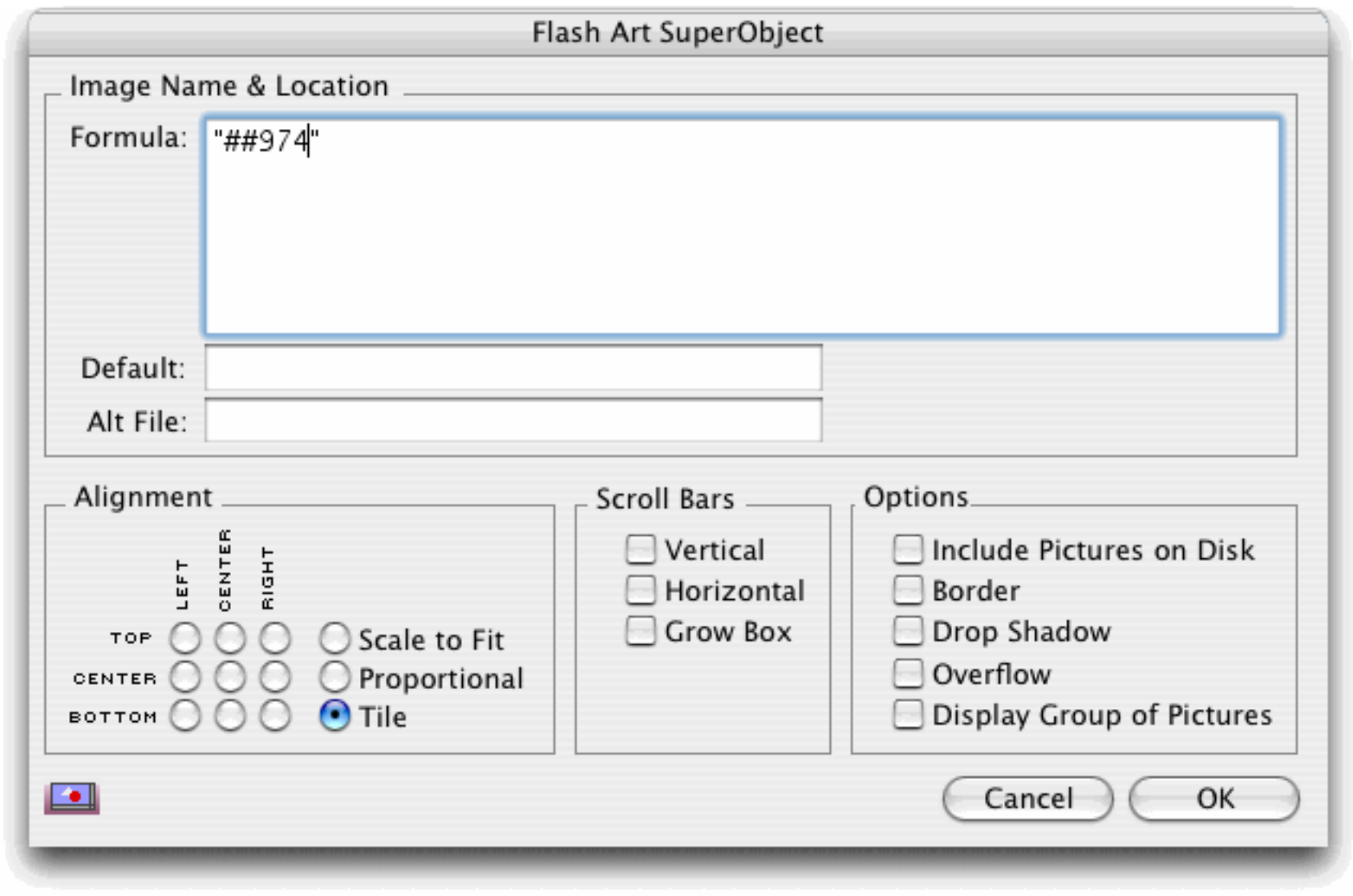
Because this image is elastic you can stretch it without distorting the corners.



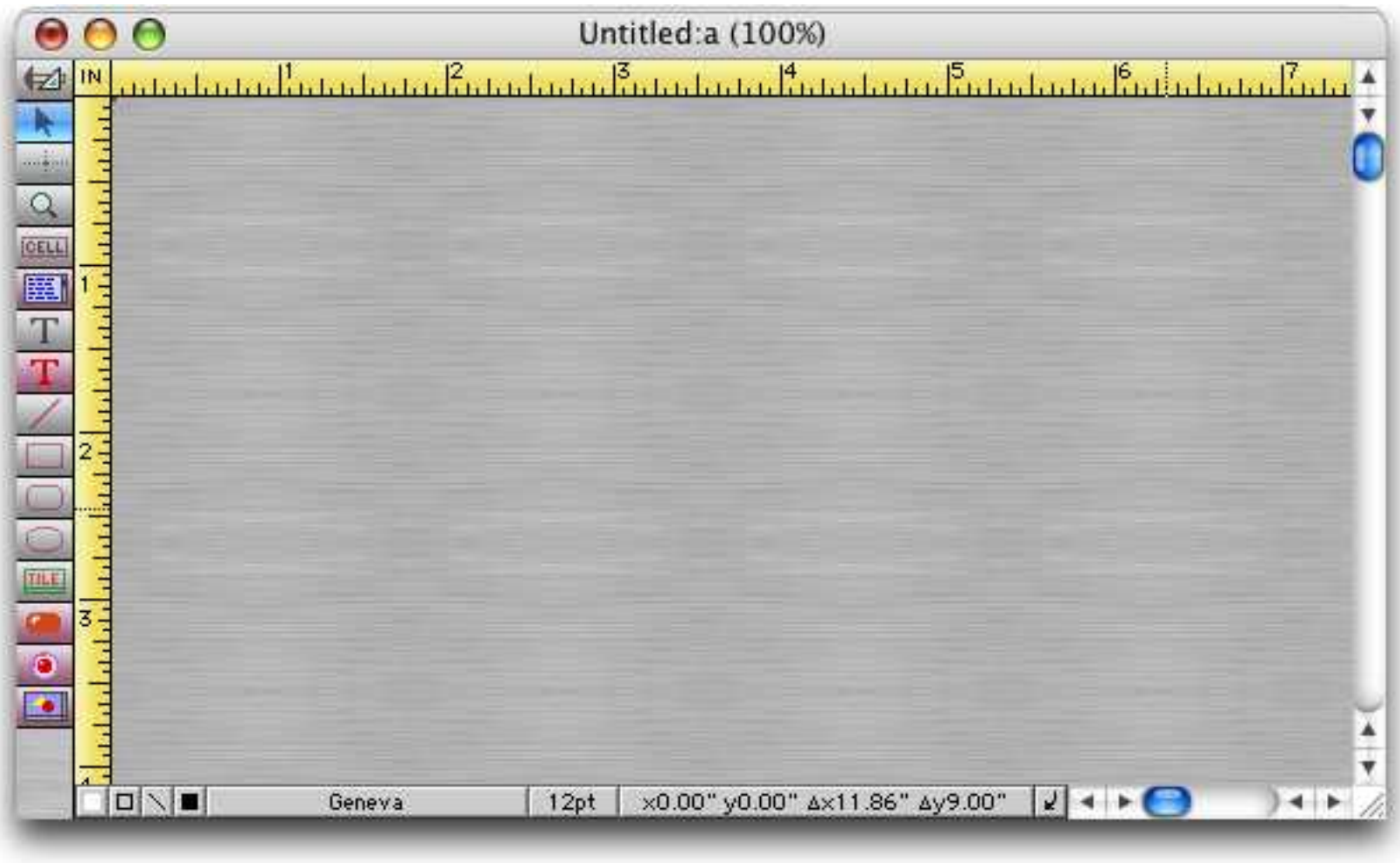
Panorama also includes several background images that are designed to be tiled, like this Brushed Metal Tile.



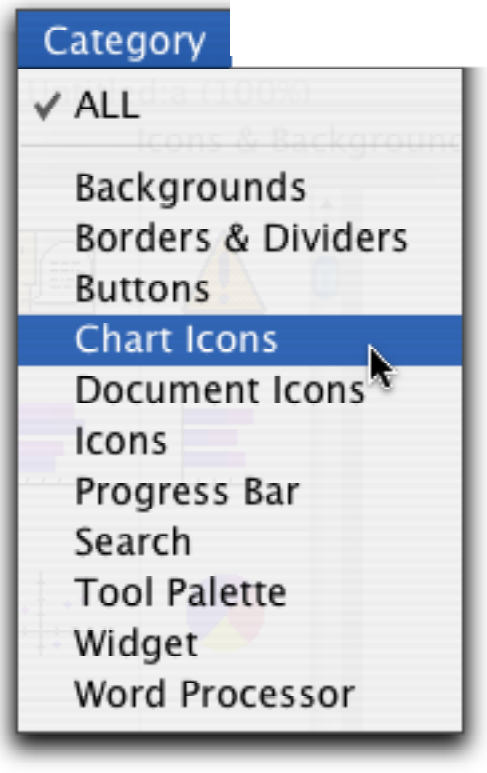
When displaying these background images we recommend that you set the Alignment to Tile.



This option will display a seamless brushed metal surface no matter how large your object is.



The **Icons & Backgrounds** wizard normally displays all of the available images. If you want to see just a particular subset you can choose from the **Category** menu.



The wizard will respond by displaying the selected images.

Displaying Icons from Resource Files

To display an icon (ICON or CICN resource) use the prefix `#*`. Icon resources can be created with ResEdit or Resourcerer. Unlike a picture, an icon definition includes a mask that allows the image to have an irregularly shaped edge (not just rectangular). This allows the icon to display correctly on a colored background. Here is a formula that will display ICON 3981.

```
"#*3981"
```

Displaying Form Preview Pictures

Panorama allows you to attach a special preview picture to any form (see “Form Comments” on page 1845). This picture can be displayed with a Super Flash Art object. To display the preview picture for any form in the current database use a formula with the format `;;<form name>`. For example, this formula will display the preview picture for the form named Avery 4932 (if any).

```
";;Avery 4932"
```

To display the preview picture for a form in another database use a formula with the format `;;<database>:<form name>`. For example, this formula will display the preview picture for the form named Avery 4932 in the Mailing Labels database (if any).

```
";;Mailing Labels:Avery 4932"
```

The database containing the form (in this case Mailing Labels) must be open to display the picture.

Elastic Pictures

Many forms require borders, buttons and widgets need to be used over and over again but with different sizes. Any image can be stretched with the Scale to Fit option, but the result is often a distorted image. For example, consider this bevel button, which looks good at 32 by 32 pixels.



Using the Scale to Fit option to stretch this image to 64 by 128 pixels doesn't look good because the corners and bevels are stretched as well.



What we would really like is for the object to stretch intelligently, with the corners and bevels remaining tight while the interior of the object stretches, as shown below.

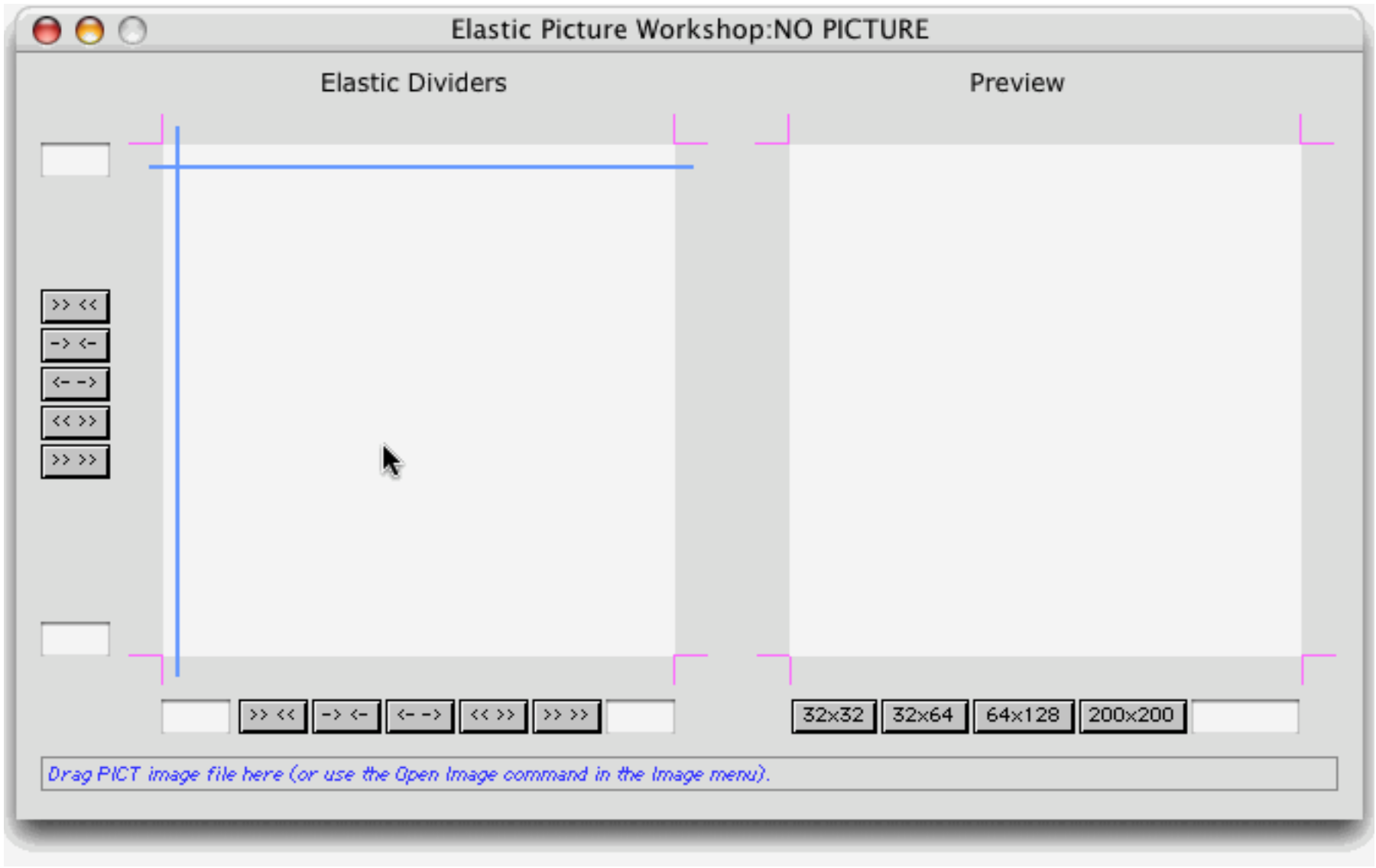


Panorama's **Elastic Picture** feature allows you to take any PICT image and enhance it so that it can be stretched at will without disturbing the corners and bevels. The image above was not customized in Photo-Shop – it's the same image displayed in Panorama using the Scale to Fit option, but after it has been enhanced with the **Elastic Picture Workshop**.

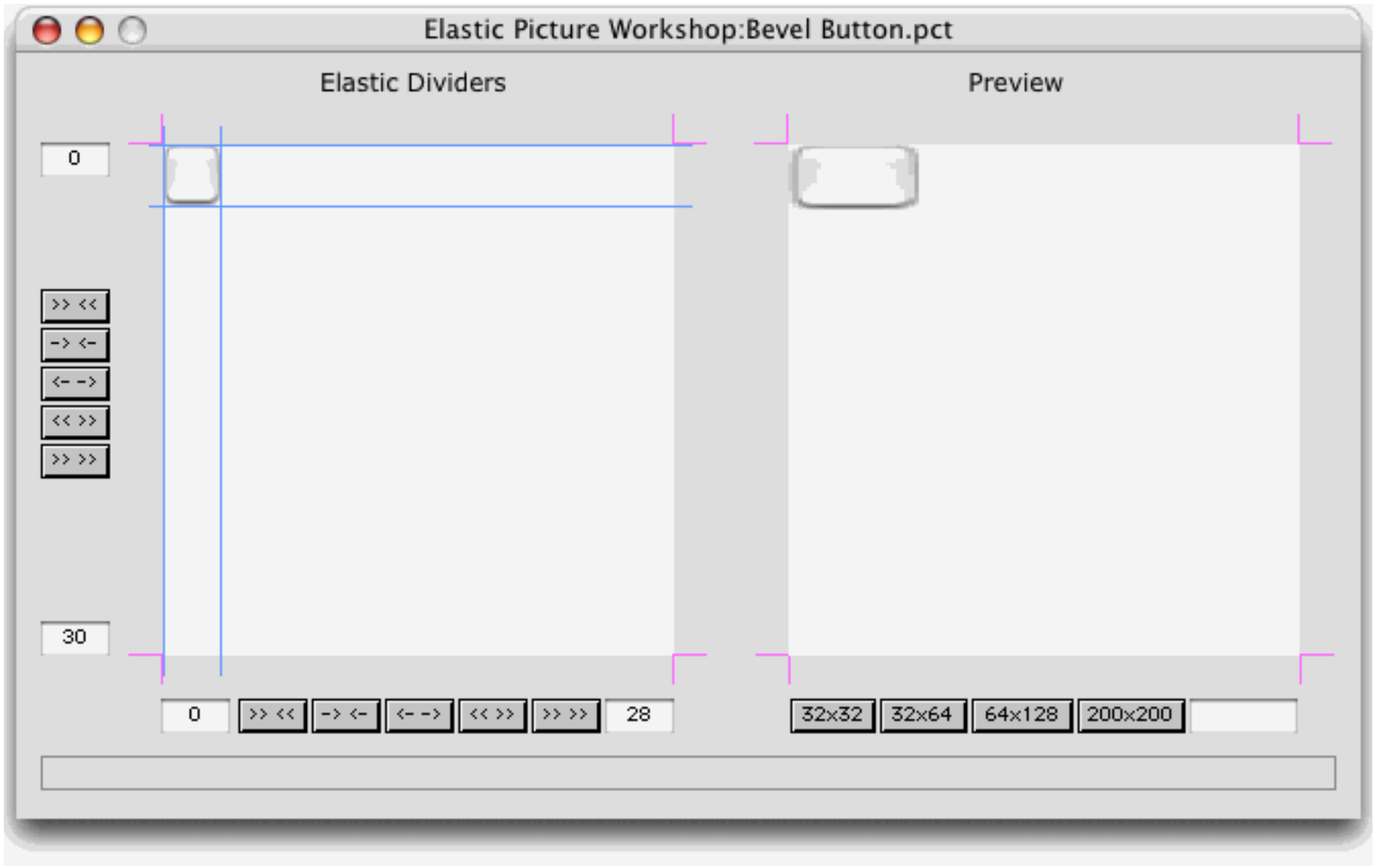
(Note: Only images in Apple's PICT format can be made elastic. Panorama takes advantage of the extensibility of the PICT format to add additional stretching information into the picture. This information cannot be added to other formats like JPEG, TIFF or BMP, so they cannot be made elastic. Fortunately many graphics programs can convert from these formats into the PICT format (even on the PC) so you will be able to make any image elastic after converting it to this format.)

Using the Elastic Picture Workshop

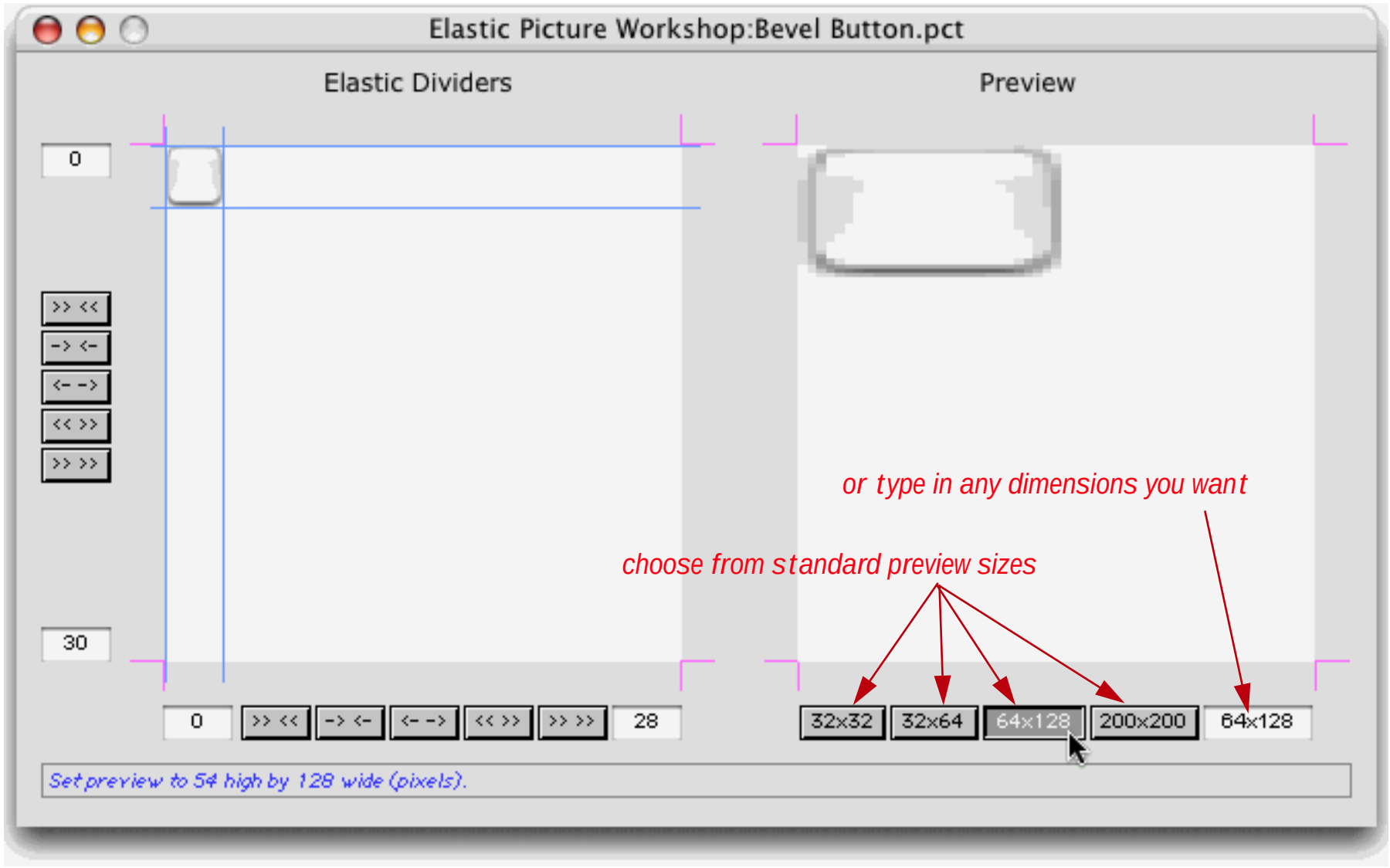
You'll use the **Elastic Picture Workshop** wizard to add the stretching information to an image. You'll find this wizard in the Developer Tools submenu of the Wizard Menu.



The first step is to open the wizard. You can either drag the image file onto the Elastic Dividers area or use the **Open Image** command in the Image menu. In the example below an image named Bevel Button.pct has been dragged onto the wizard.

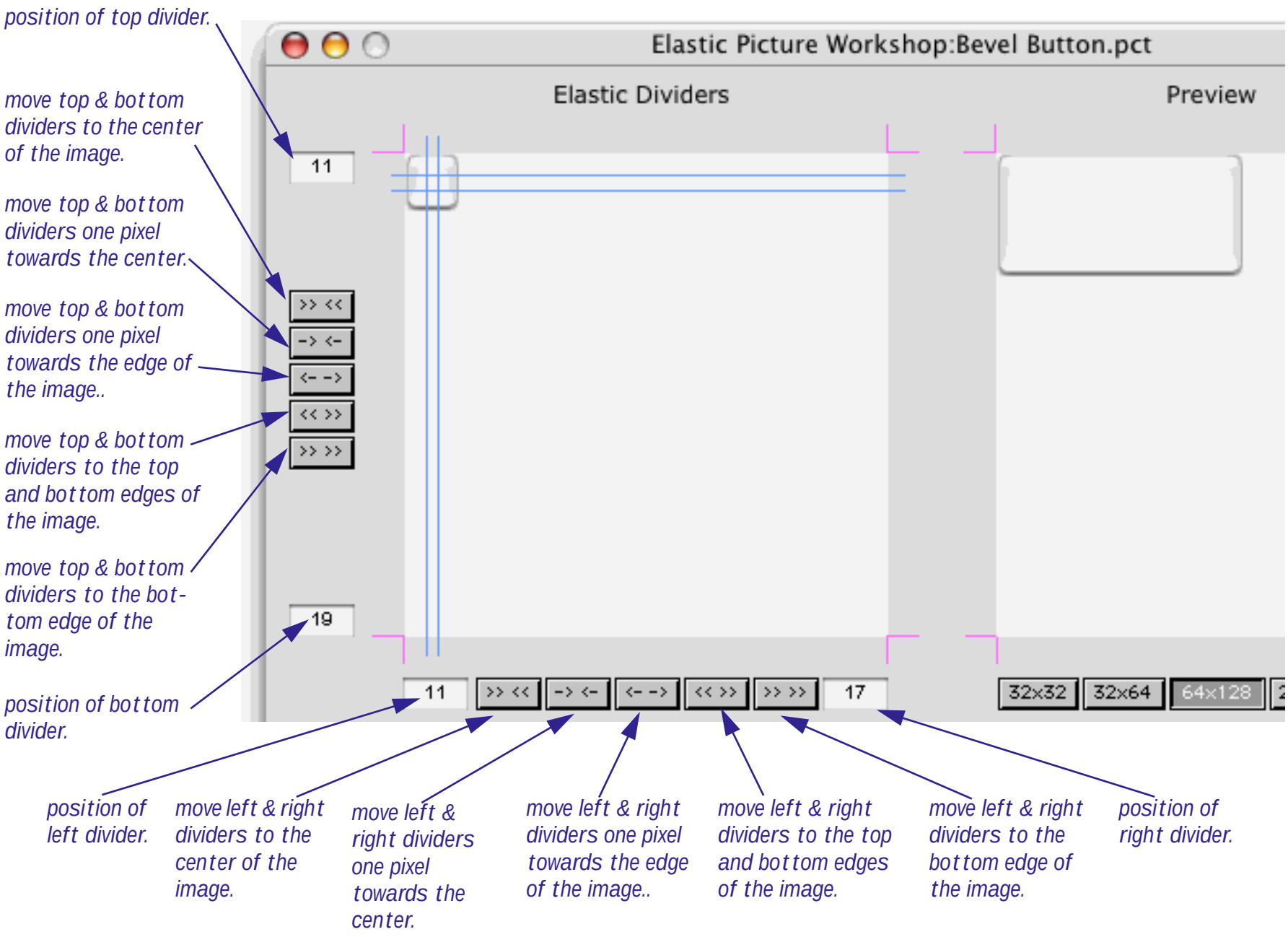


The wizard is divided into two sections. The left section, **Elastic Dividers**, displays the original size object. The right section, **Preview**, displays a stretched version of this object. You can use the buttons along the bottom of this section to control how much the image is stretched, you can also type in any stretch size you want in the dimension box on the right.



Since the image isn’t elastic yet, the preview shows the stretched corners and bevels that we want to get rid of. Let’s do that now.

The left side of the wizard displays four light blue elastic dividers around the edges of the original image. You can manipulate these dividers with the buttons along the left and bottom edges. The buttons along the left hand side control the position of the top and bottom dividers, while the buttons on the bottom control the position of the left and right dividers. The goal is to position the dividers so that the edges and bevels are divided from the main body of the image. As you adjust the dividers into place you'll see the edges and bevels in the preview magically clean up.



Tip: When you press the -> <- or <- -> buttons the dividers will normally move one pixel. However, if you hold down the **Shift** key the dividers will move five pixels in the specified direction.

Saving the Elastic Picture

When the dividers are in place use the **Save Image** or **Save Image As** commands in the Image menu to save the elastic image. The wizard will save an adjusted version of the picture. (Technical Note: The information necessary to allow the image to be displayed elastically is stored in the picture itself in a PICT comment. The image can still be edited with PhotoShop or another image editing program, but doing so will remove the PICT comment and the image will no longer be elastic.)

The picture can also be saved directly into a resource file with the **Save in Resource** command. This command will prompt you to locate an existing resource file (you must create this file in another application) and then prompt you for the resource number.

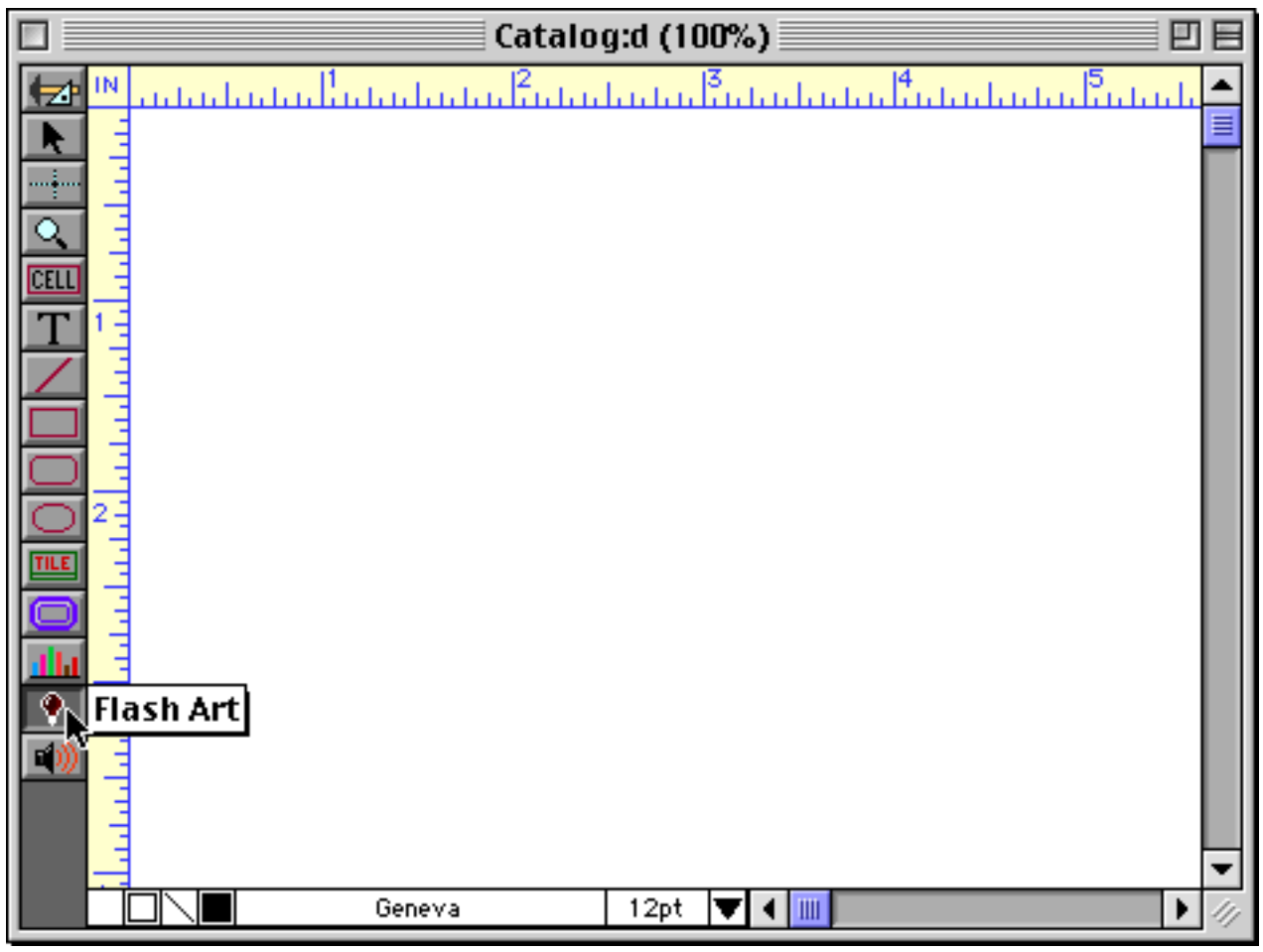
Using the Elastic Picture

To display an Elastic Picture on a form you'll need to use the Flash Art SuperObject. Set up the options just as you would for any other PICT file (make sure the Include Files on Disk option is checked). You'll also want to make sure that the Scale to Fit option is checked. That's all there is to it. You can also use elastic pictures with any SuperObject that displays PICT files – Flash Art Push Buttons, Flash Art Data Buttons, etc.

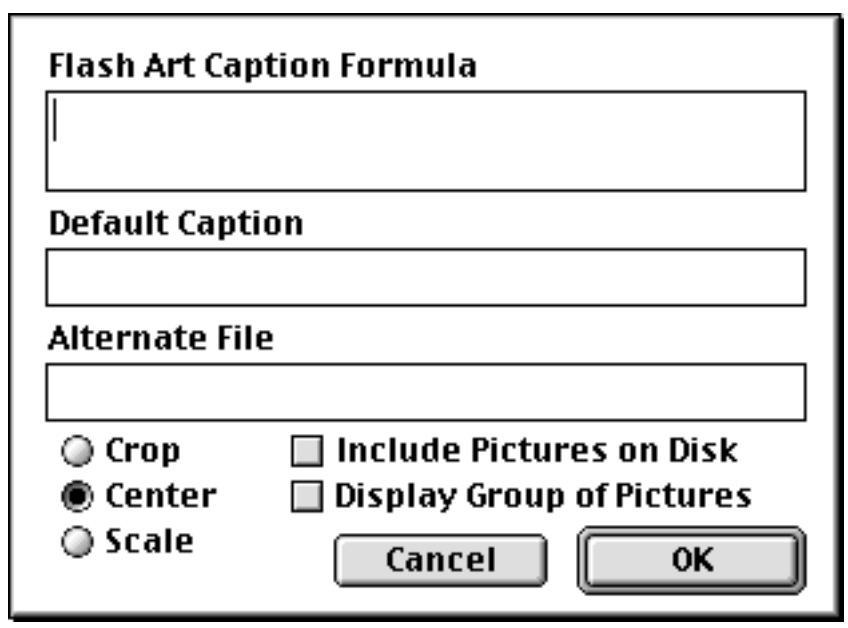
“Classic” Flash Art Objects

In addition to the Super Flash Art object described throughout most of this chapter Panorama also has a “classic” Flash Art object. When Super Flash Art objects were added as part of Panorama 3.0, “classic” Flash Art objects were retained for compatibility with older databases. We recommend that you use Super Flash Art for new applications.

Using “classic” Flash Art objects is very similar to working with Super Flash Art objects. The **Flash Art** tool looks like a lightbulb and is part of the standard Panorama tool palette.



To create a “classic” Flash Art object select this tool and drag the mouse across the form (see “Creating Super Flash Art Objects” on page 749). The configuration dialog for a “classic” Flash Art object looks like this.

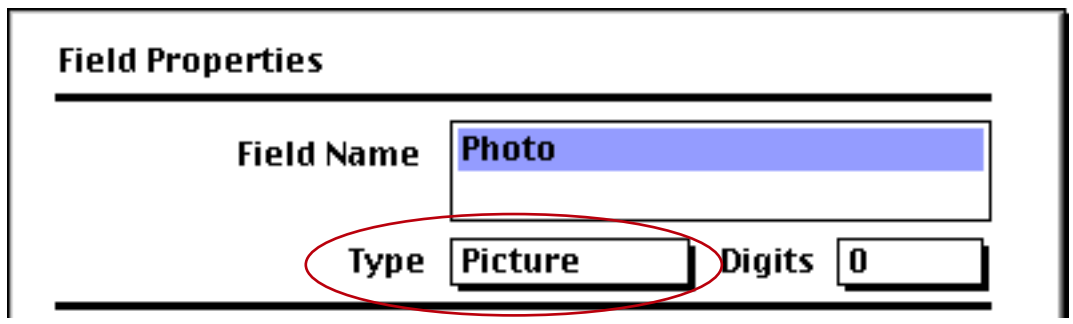


The options in this dialog are a subset of the options in the Super Flash Art object. See “Super Flash Art™ Options” on page 777 for a description of each option. Note: The “classic” Flash Art formula is limited to 30 characters, not 255 like the Super Flash Art version.

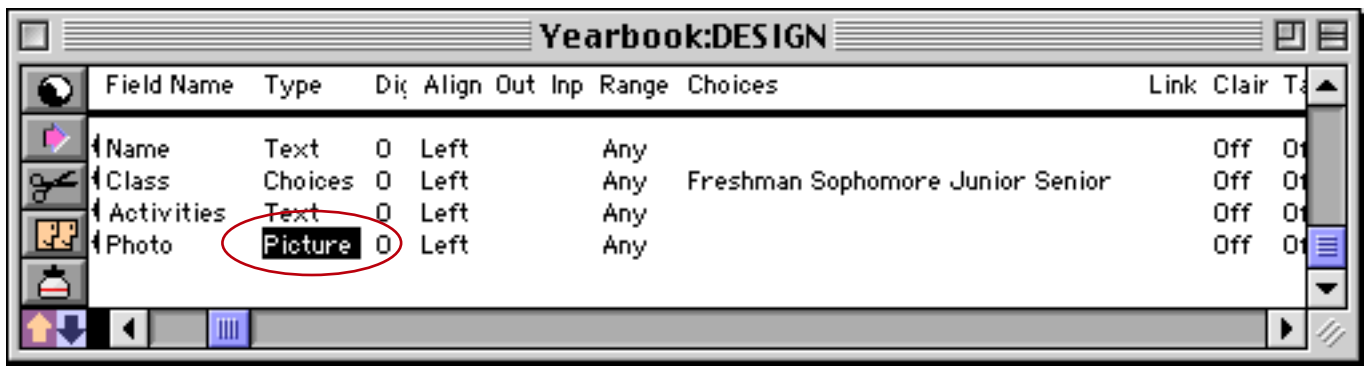
Storing Images in a Field

Flash Art is the recommended method for displaying images in a Panorama database. Panorama does, however, have another method for storing and displaying images. Instead of using Flash Art you can actually set up a Picture field and store the images directly in the database. This has many disadvantages—it uses up gobs of memory and you have no control over the alignment or scaling of the image. Nevertheless, this has been an option since the first version of Panorama and is retained for compatibility with databases that may use this feature.

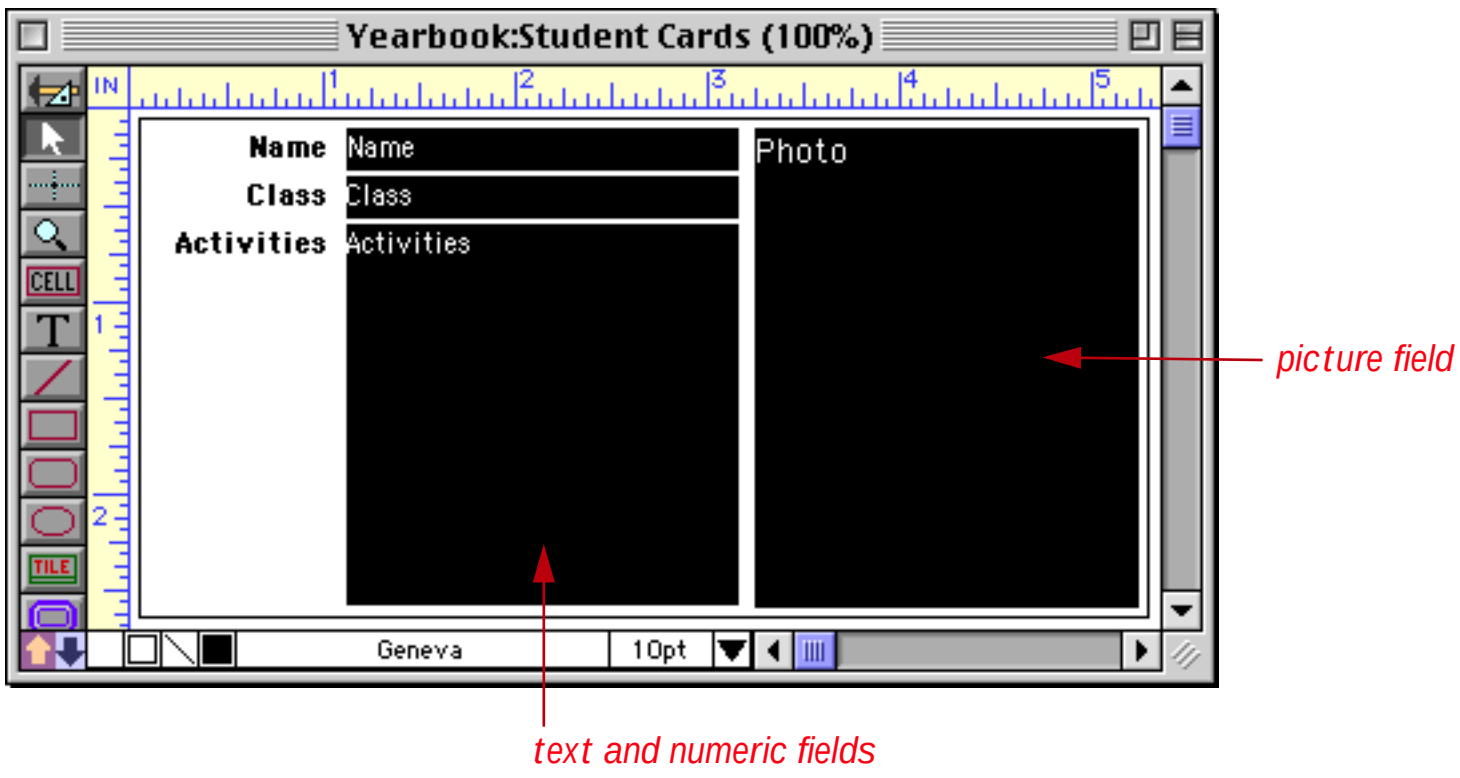
To store images directly in the database, the database must contain a field that is set up using the picture data type. You can set up this data type using the **Field Properties** dialog (see "Setting Up a Field's Data Type" on page 286).



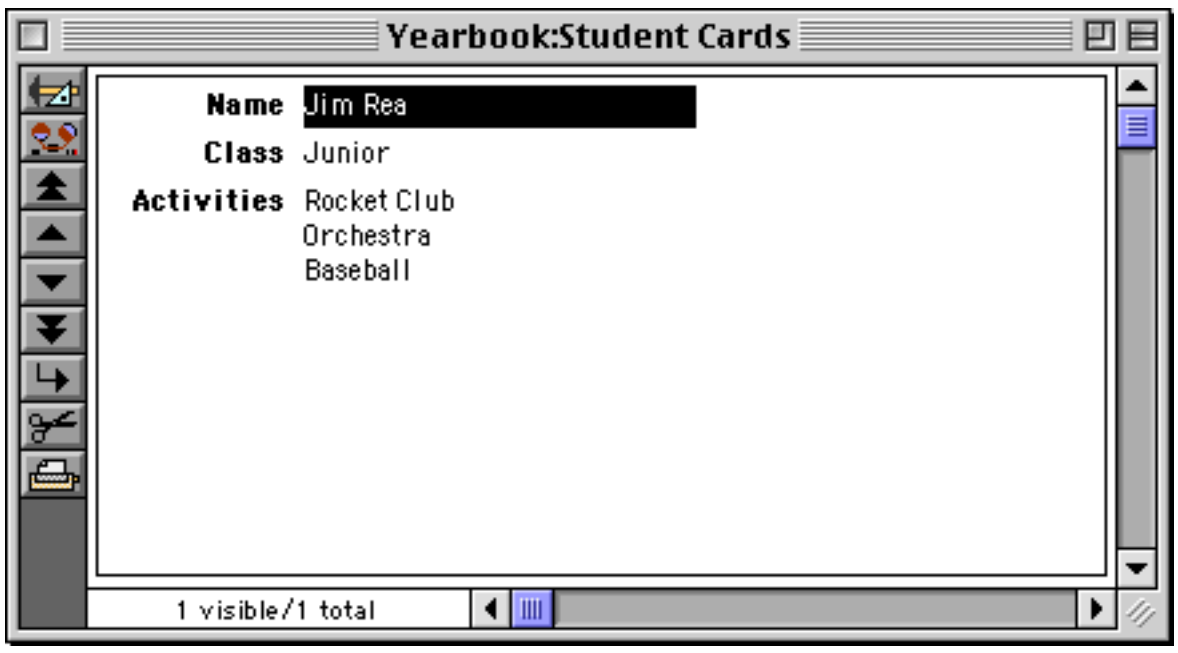
You can also set up a Picture type field with the design sheet (see "The Design Sheet" on page 252).



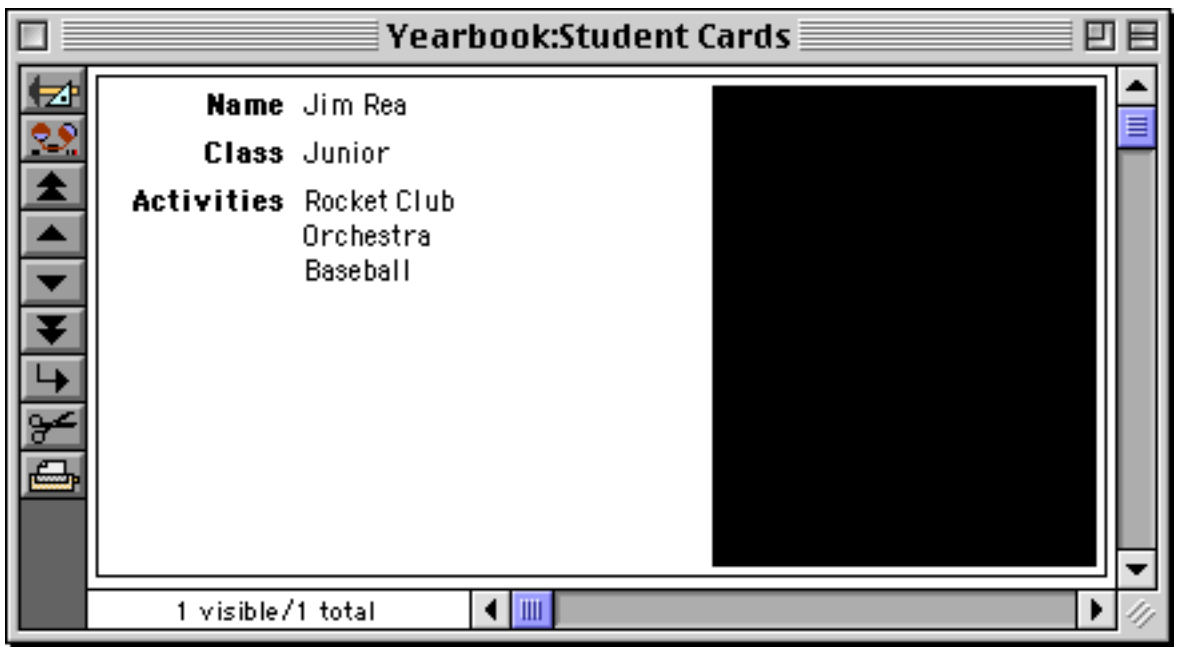
The Picture data type cannot be used in the data sheet—you must set up the form. Create a **Data Cell** object for the picture field (see "Working with Data Cell Objects" on page 635). The Text Display Object should be large enough to display the largest picture you want to use. For other fields you may use either Data Cells, Text Editor SuperObjects, Text Display SuperObjects or Auto-Wrap text objects.



Once the form is set up, switch to Data Access Mode. The text and numeric fields can be filled in using the normal techniques (i.e. the keyboard).



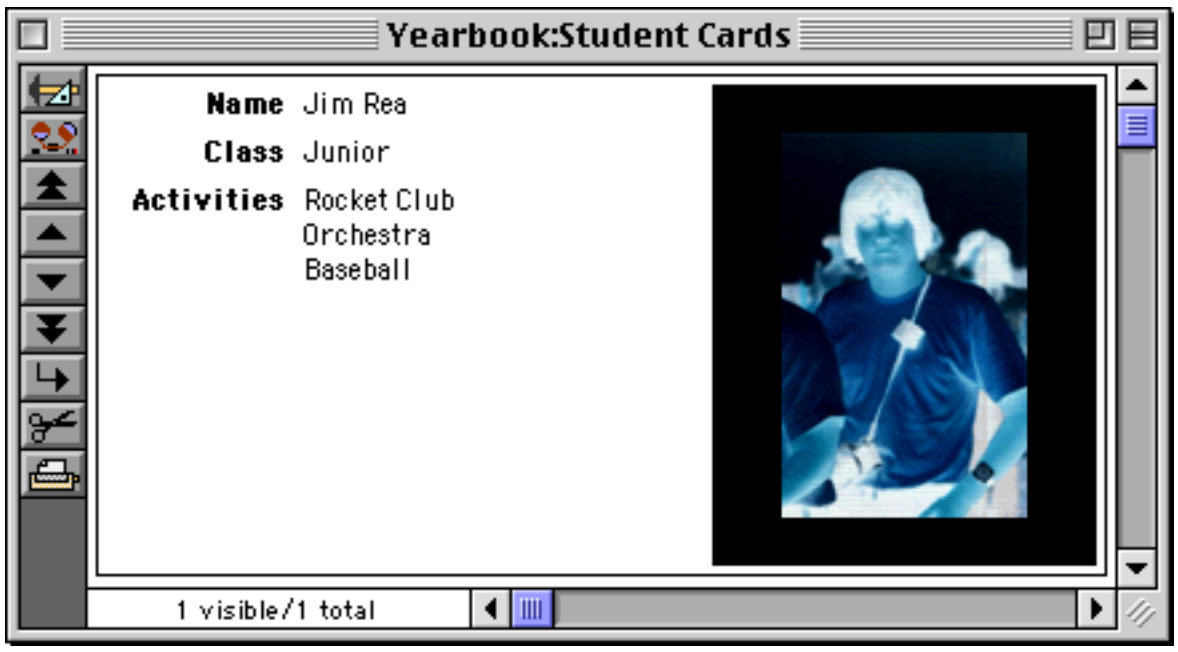
Now for the picture. Start by clicking on the picture cell to select it.



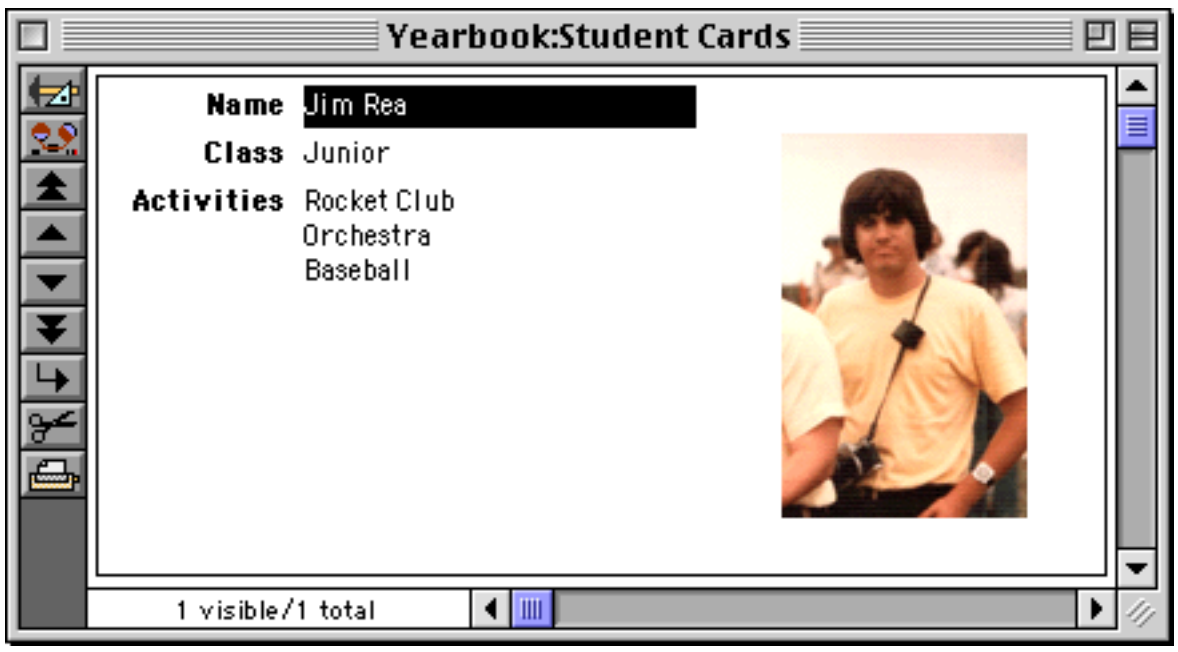
The next step is to go into your graphics application (Photoshop, etc.), locate the image, and copy it onto the clipboard.



Now go back to Panorama and use the **Paste** command to paste the image into the database.



When a normal data cell is selected, the text in that cell switches to a negative image, with white text in a black background. As you can see, the same thing happens with a picture data cell. When the data is selected, the picture appears in reverse, like a photographic negative. When another data cell is selected, the normal picture image will appear.

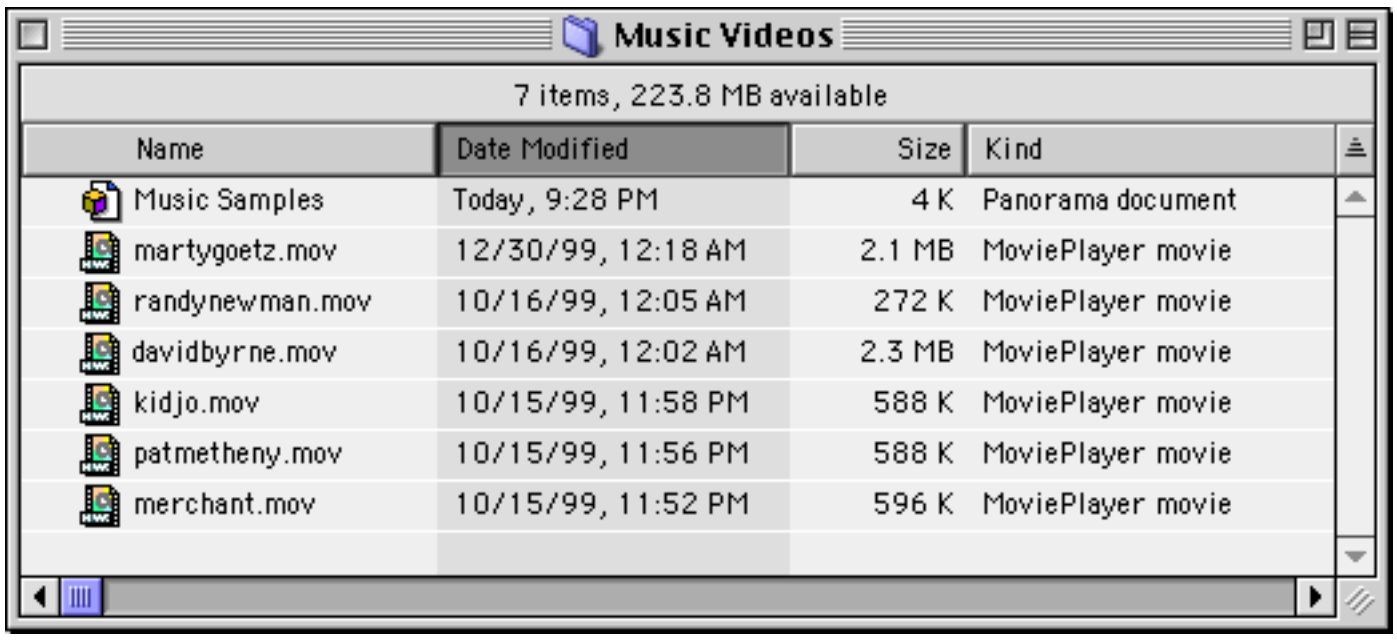


You cannot edit a picture by double clicking on the data cell, nor can you sort or select using a picture field or use a picture field as part of a formula. In the data sheet the picture field appears to be blank, and Panorama will not allow you to double click on the field.

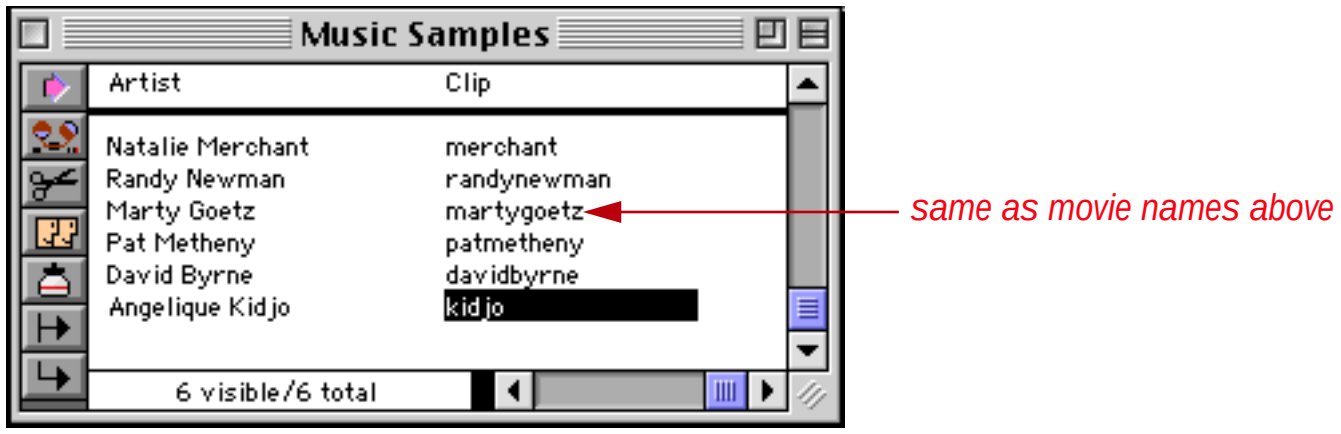
Displaying Movies in a Form

Panorama's Super Flash Art object is capable of displaying movies as well as still images. To do this you must have Apple's QuickTime software installed on your computer. If you don't have this software you can download it from **www.apple.com** for either Macintosh or Windows PC systems.

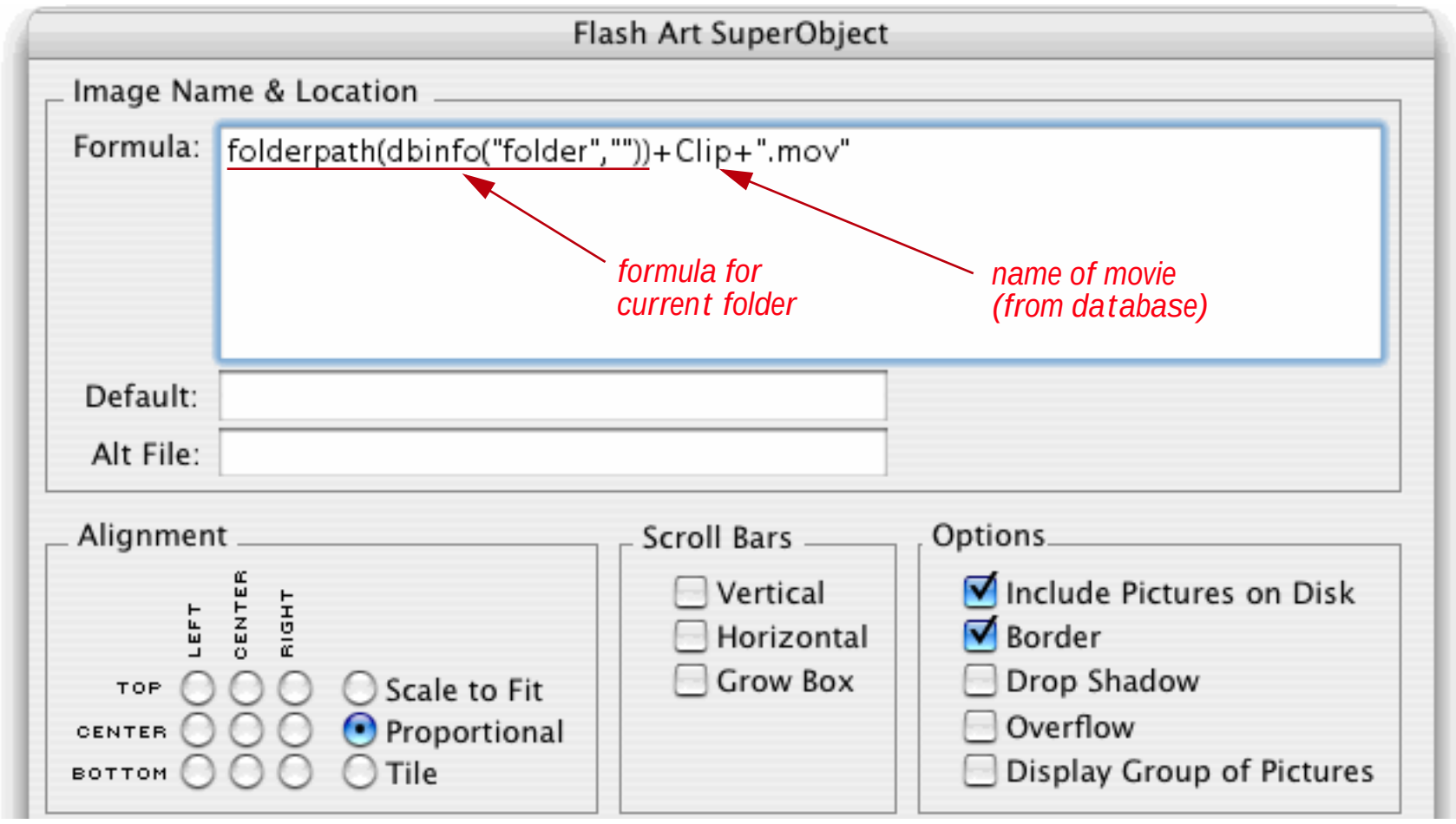
The first step in displaying movies is to create some movie files.



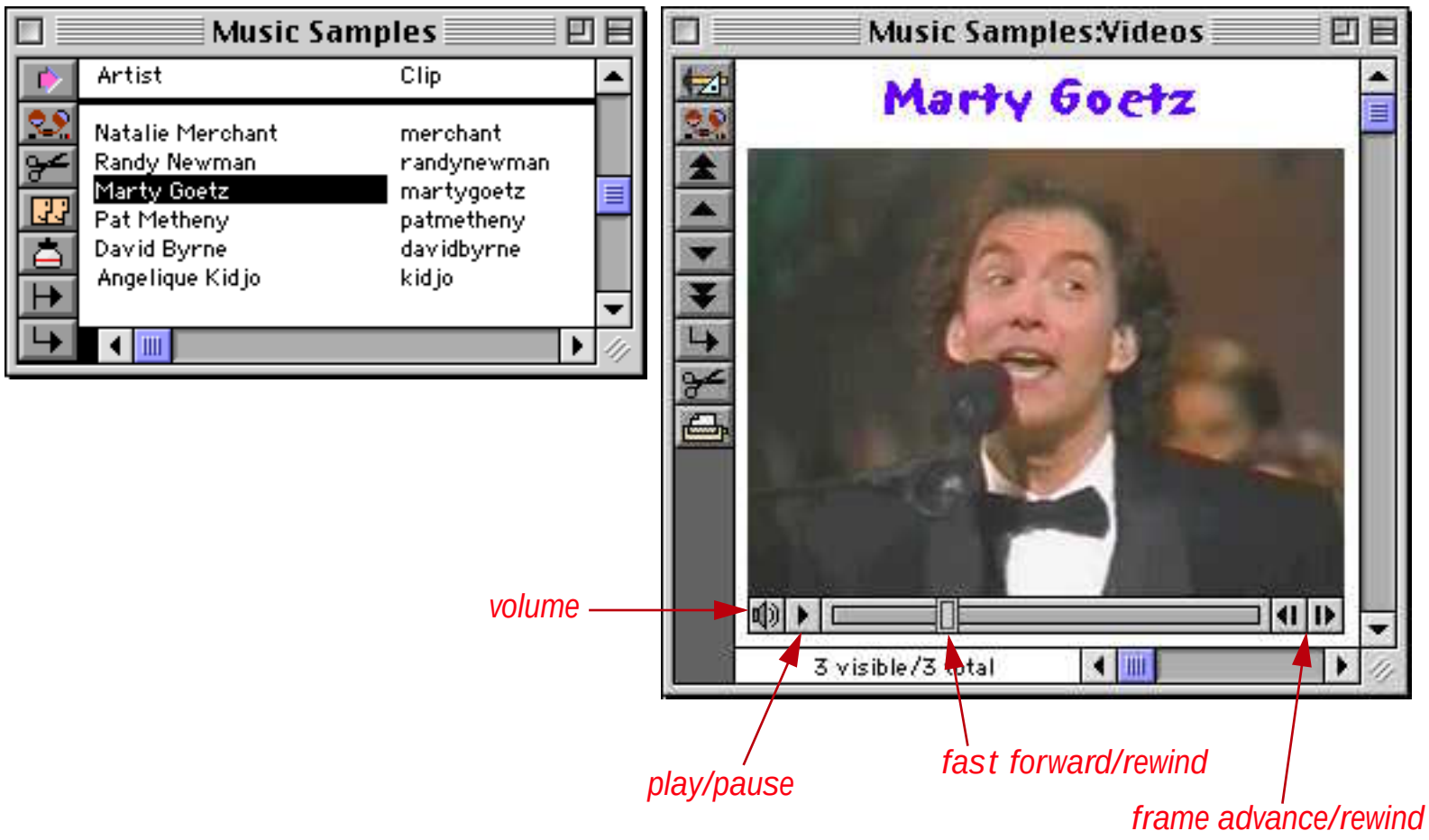
Next, you'll need to create a database. In this case we created a very simple database that has the artist's name along with the name of the movie (we'll add the .mov extension in a moment).



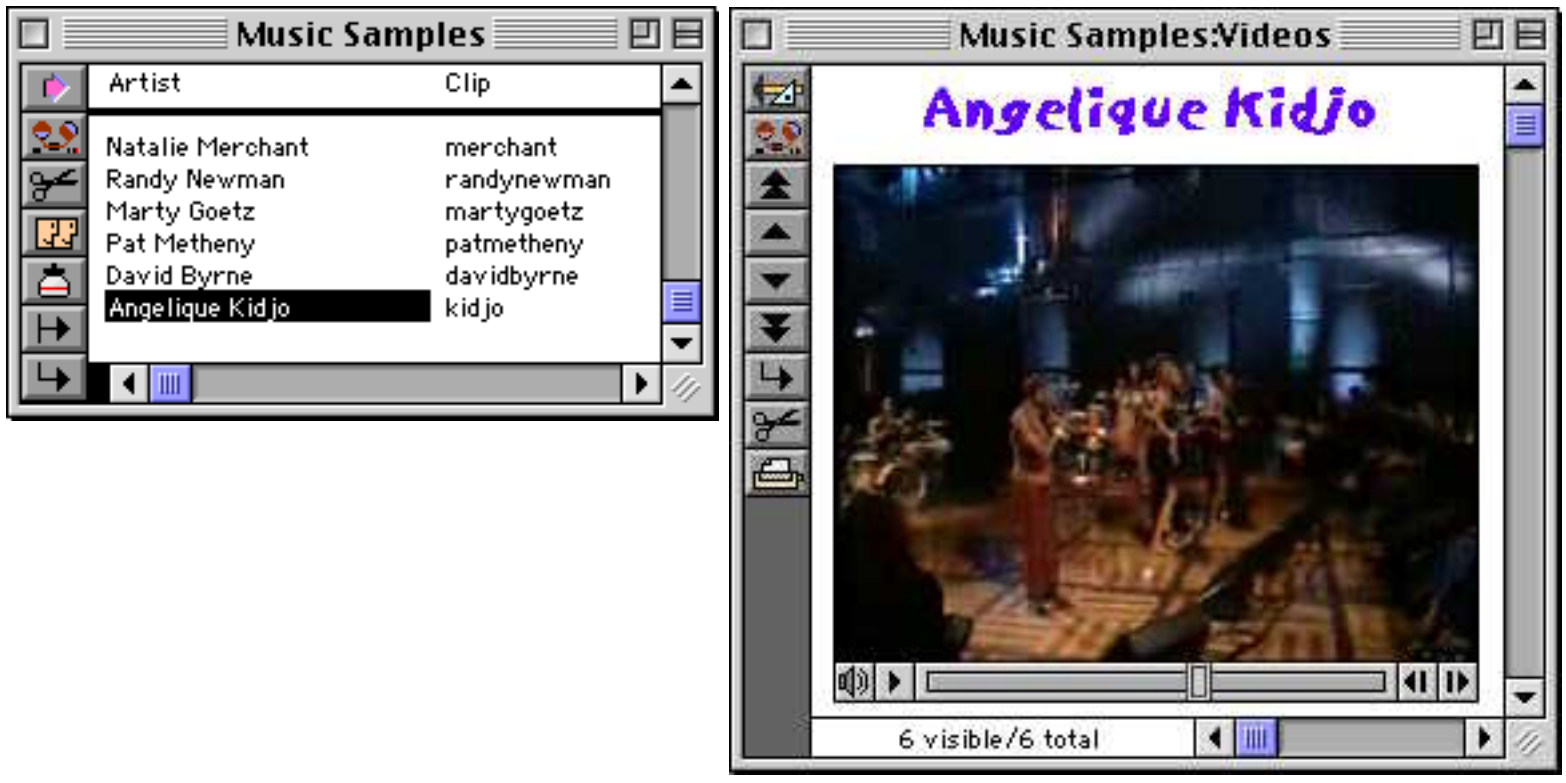
To display the movies we'll need to create a form, and then to create a Super Flash Art object within the form (see "Creating Super Flash Art Objects" on page 749). Unlike still images, movies always require you to specify the complete location of the file. The formula below shows how to do that (assuming that the movies are in the same folder as the database) and also adds the .mov extension. Make sure that the Include Pictures on Disk option is enabled, because a movie cannot be pasted into the Flash Art Scrapbook.



Switch to Data Access Mode to try out the movie. The normal QuickTime controls appear at the bottom of the Super Flash Art object.



To see a different movie simply move to the record for the movie you want to view.



In Graphics Mode editing a Super Flash Art object that is set up to display a movie can be a bit tricky. Even though you are in Graphics Mode, clicking on the movie tends to activate the movie controls instead of selecting the object. If you have difficulty, try dragging a marquee around the object instead of clicking on it (see “Selecting Multiple Objects at Once” on page 502). Instead of double clicking on the object to open the configuration dialog, try using the **Object Properties** command in the Edit menu.

Once a movie has been set up it can be controlled with a procedure as well as with the movie control strip. The procedure can start and stop the movie, move to a specific spot within the movie, change the playback rate and the volume. To learn how to program a movie see “Super Flash Art Commands (Including Movie Control)” on page 1810.

For those of you that can’t stand not knowing who Marty Goetz and Angelique Kidjo are, check them out at **http://www.martygoetz.com** (Marty Goetz) and at **http://wwwusers.imaginet.fr/~kidjo/home.html** (Angelique Kidjo).

Chapter 17: Buttons & Widgets



The Industrial Revolution introduced machinery with all kinds of knobs, levers, and doo-dads. In our electronic age these have been replaced by virtual push buttons, checkboxes, “radio” buttons, pop-up menus, scrolling lists, and other widgets. Panorama has a number of tools for incorporating these types of controls into your forms.

Although most of the buttons and widgets discussed in this chapter can be used without programming, most of them are often combined with Panorama procedures. Because of this, we will often reference procedures and programming techniques throughout this chapter. Before reading this chapter you may want to skip ahead and learn how to create and work with procedures (see “Procedures” on page 1340) and variables (see “Variables” on page 1199).

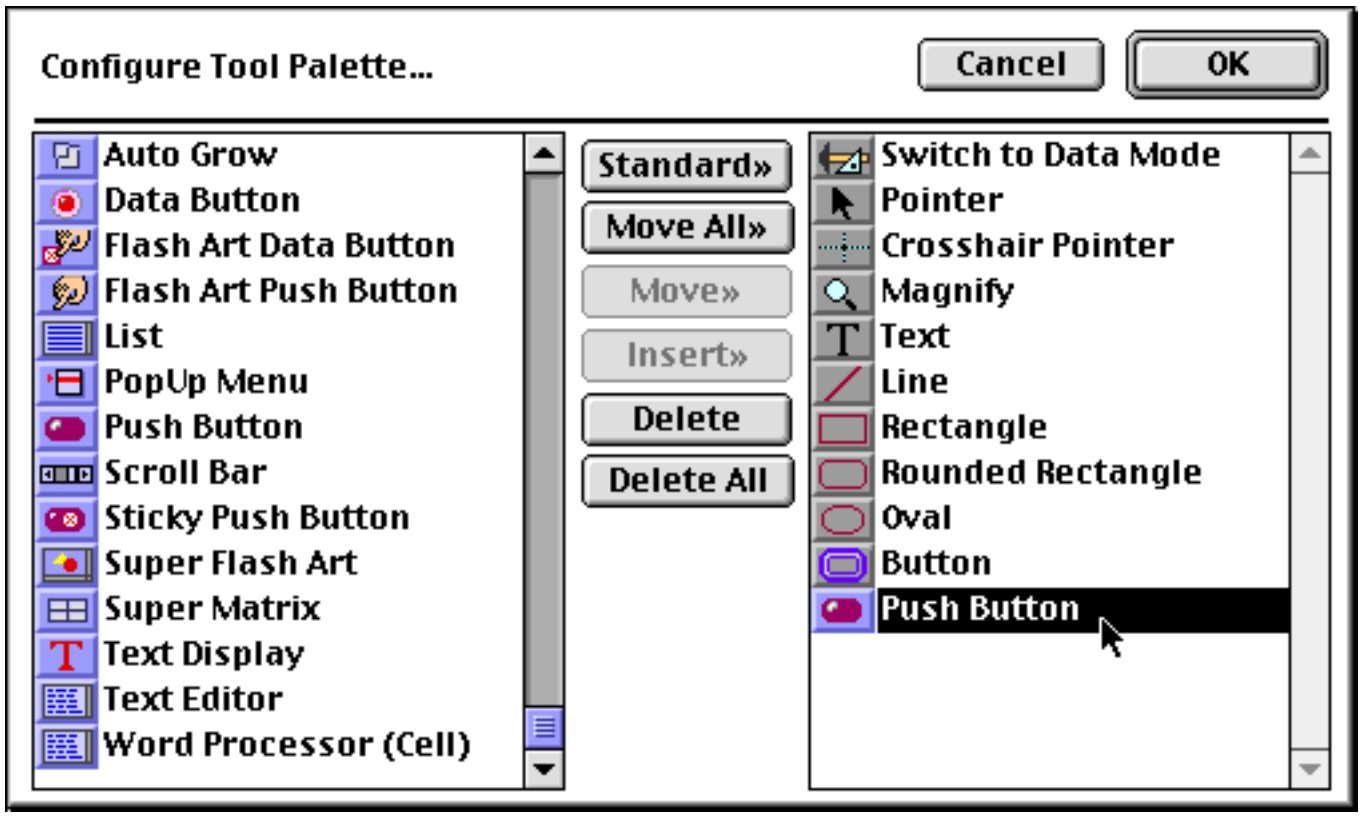
Push Buttons

Push buttons have one mission in life—to start something. In Panorama, clicking on a push button triggers a procedure (see “50 Ways to Trigger a Procedure” on page 1458). You push the button and the program starts, simple as that. Panorama has three different types of button objects—Super Object Push Buttons (next section), “Classic” Button Objects (see “‘Classic” Push Buttons” on page 820) and Flash Art Push Buttons (see “Flash Art™ Push Button SuperObjects™” on page 823).

Super Object Push Button

The SuperObject Push Button makes it easy to create attractive buttons that trigger procedures. Unlike Panorama’s standard button tool, the SuperObject Push Button can create circular or oval buttons, and buttons with various 3-D effects.

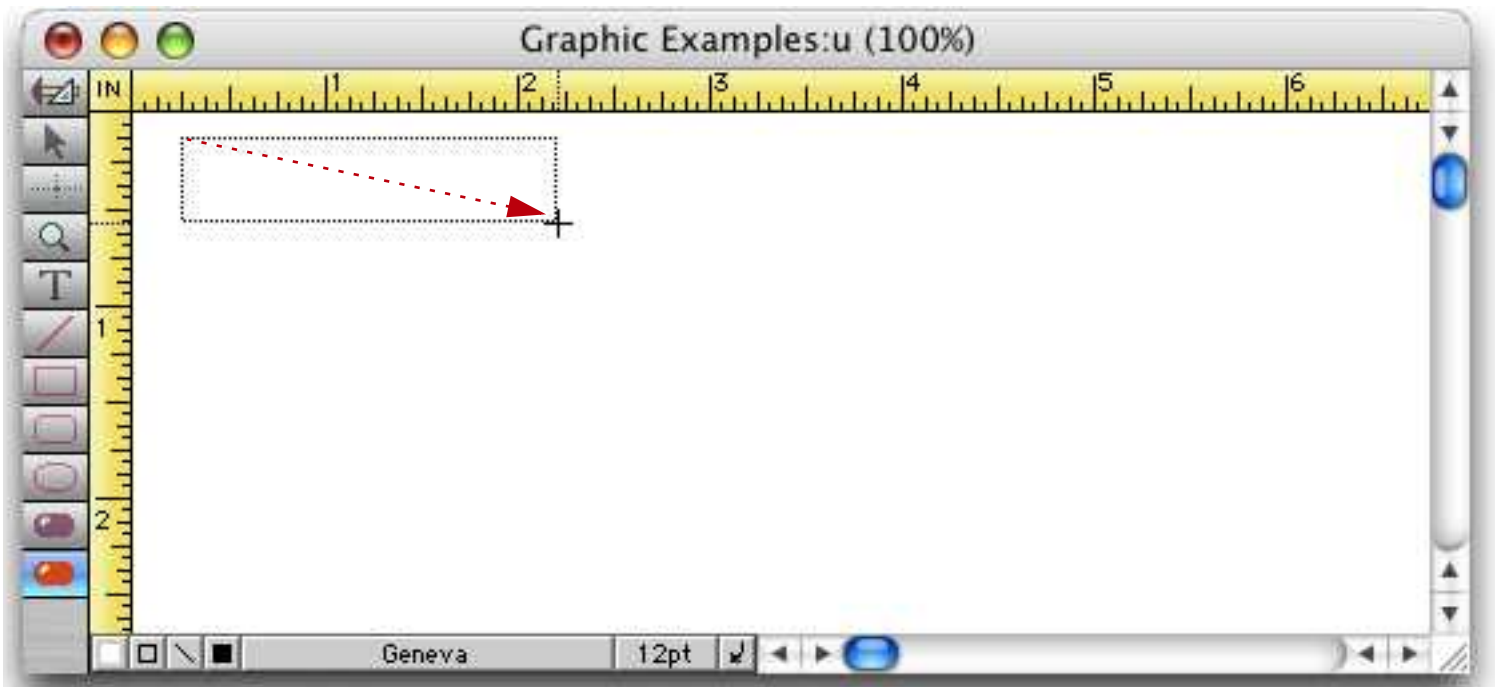
The SuperObject Push Button tool is not in the default tool palette, so you'll need to use the **Tool Palette** dialog to add this tool to the palette if it is not already there (see "Customizing the Tool Palette" on page 497).



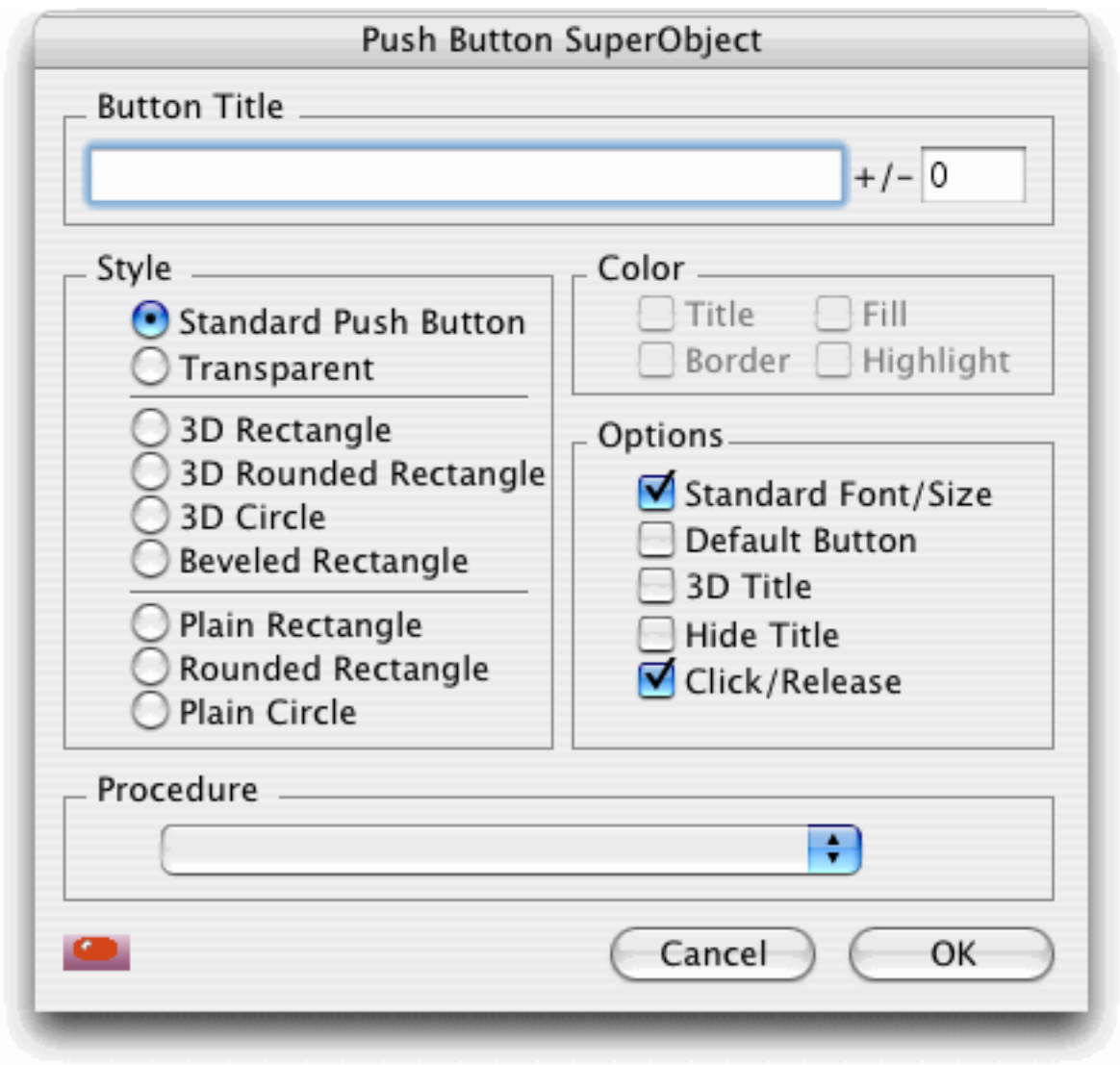
Now that the tool is added to the palette you can select it.



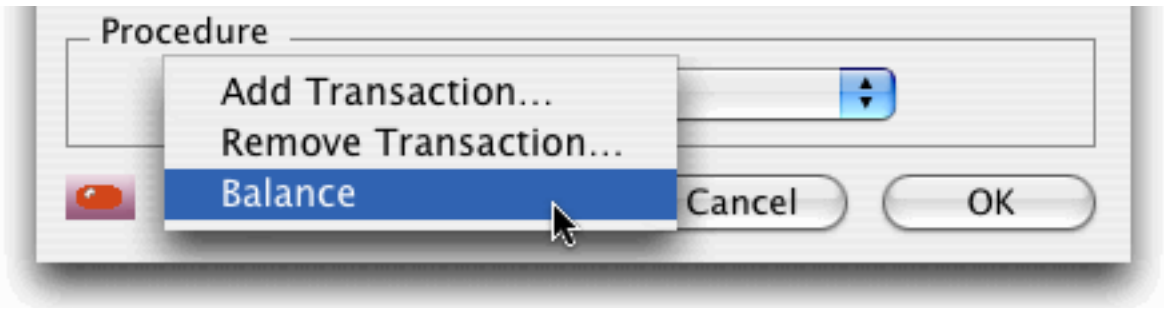
Once the tool is selected, drag the mouse across the form in the location where you want to create the button.



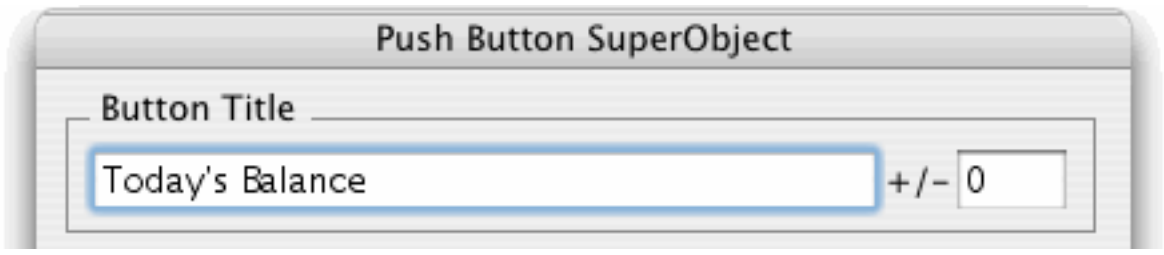
When you release the mouse, the SuperObject Push Button configuration dialog will appear.



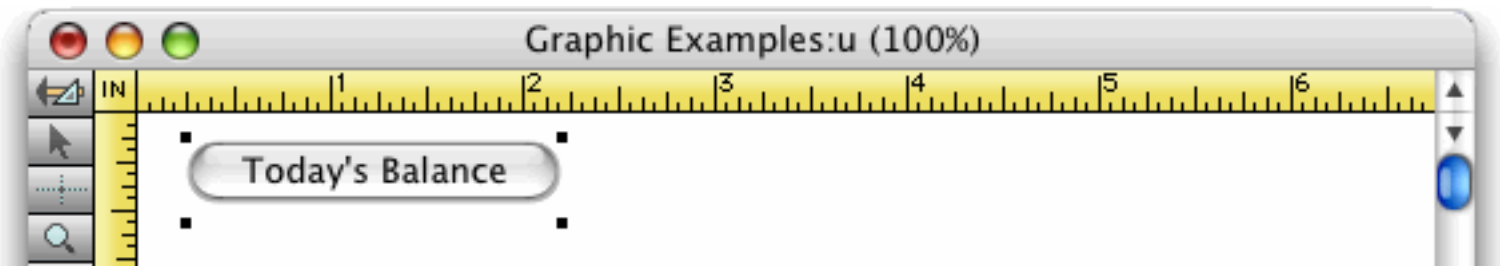
Use the pop-up menu to select the procedure that will be triggered by this button. The button can trigger any procedure in the current database. (If you haven't created the procedure yet you can still create the button, then go back later and choose the procedure.)



Usually you'll want to set up a title for the button, and a style. The title can be the same as the procedure name or it can be different.



When you press the **OK** button the new button will appear in your form.



You can modify the appearance of the title with the **Font** and **Size** menus (see "Font" on page 529 and "Text Size" on page 531). To re-open the configuration dialog you can either double click on the button or select the button and open the dialog with the **Object Properties** command in the Edit menu.

Push Button Styles

There are over a dozen controls for changing the push button appearance. The primary control is the button style.

Standard Push Button: This option displays a standard button for the current platform. The appearance of the button will vary depending on the operating system you are using: Windows, OS X, or OS 9. The example below is for OS X.



The appearance of a Standard Push Button can be modified using the Default Button option. When this option is used the button appearance changes to indicate that this is the default button (the button in a dialog that is activated automatically when the Enter key is pressed).

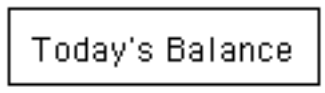


Remember, the Default Button option only changes the appearance of the button. You still have to program the button to make it the default (see "Custom Dialogs" on page 1616).

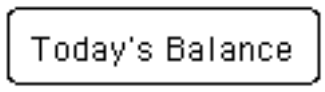
Transparent: This option displays a transparent button. A transparent button will be completely invisible except for the title (and you can make that invisible too with the Hide Title option).

Today's Balance

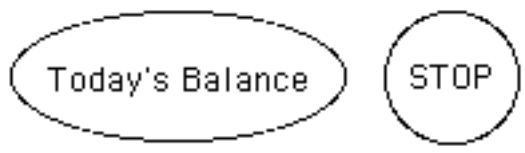
Rectangle: This button style has square corners.



Rounded Rectangle: This is a standard two dimensional button. Panorama will display the name of the button inside the button. If you want this to look like a standard Macintosh button, you should use Chicago 12 point type.



Circle / Oval: This style has an oval border. If the button's dimensions are square the button will be a circle.



3D Rectangle: This style looks like a 3 dimensional rectangle with beveled edges.



3D Rounded: This style looks like a 3 dimensional rounded rectangle with beveled edges on all four sides.



3D Circle / Oval: This style looks like a 3 dimensional oval with beveled edges.



Beveled Rectangle: This style looks like a 3 dimensional rectangle that is beveled on the top and bottom. This style looks a lot like the buttons on many VCR's and stereos.



Note: On black and white (b / w) monitors, Panorama automatically converts the 3D buttons to the corresponding 2D button.

Button Title

The button title is the title that appears on the button. This title will be centered in the middle of the button. The procedure that is triggered by the button can find out what button was pressed with the `info("trigger")` function. This function will return Button. followed by the title of the button. Using this function you can have several buttons that trigger a single procedure. The procedure can then use the `info("trigger")` function to find out which button was actually pressed. The procedure will usually use `if` or `case` statements to decide what button was pressed, like this.

```
local actionDate
case info("trigger") = "Button.Mon" actionDate=date("Monday")
case info("trigger") = "Button.Tue" actionDate=date("Tuesday")
case info("trigger") = "Button.Wed" actionDate=date("Wednesday")
case info("trigger") = "Button.Thu" actionDate=date("Thursday")
case info("trigger") = "Button.Fri" actionDate=date("Friday")
endcase
select Date=actionDate
```

Advanced Note: In this particular case, there is an easier way to write this procedure. Instead of using `case` statements for each button, this procedure simply uses a text funnel to strip out only the name of the day.

```
select Date=date(info("trigger")[8,-1])
```

You can often use a trick like this if you choose your button names carefully. See “Taking Strings Apart (Text Funnels)” on page 1214 for more details about text funnels.

Title Positioning

Panorama attempts to center the title both vertically and horizontally within the SuperObject push button. However, some fonts have non-standard vertical dimensions and need an adjustment to center properly. If necessary, you can adjust the title’s vertical position with the +/- option. For example, to move the title up by 2 pixels, enter 2 (or +2) into this option. To move the title down by one pixel enter -1.



Standard Font/Size

If this option is selected the button title will always be displayed in the operating system’s standard font and size. For example, on OS X systems the standard font is 14 point Lucinda Grande.

3D Title

If the 3D Title option is turned on, Panorama will display the title with a white shadow, giving it an “etched” 3D look. Note: This option is ignored if you have use the Standard Push Button option. Also if you display the button on a black and white (b/w) monitor, Panorama ignores this option and displays the title normally.

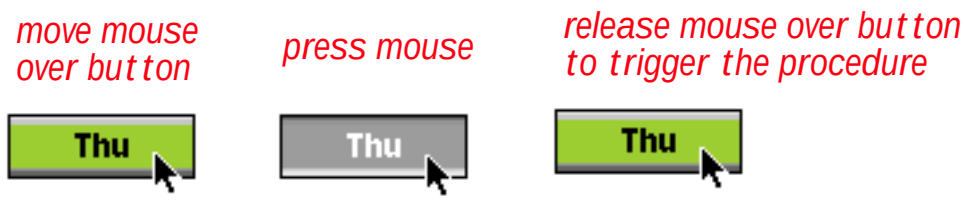


Hide Title

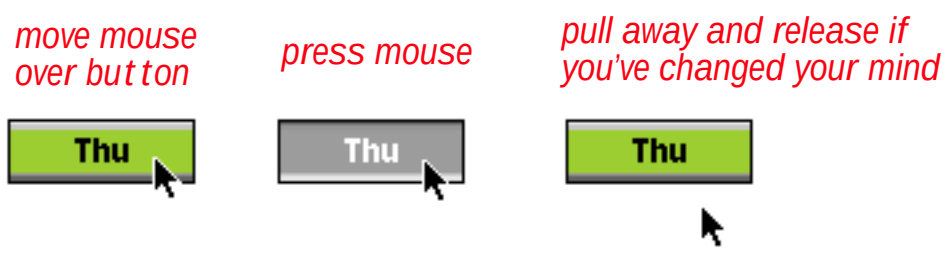
This option hides the title. In other words, the button will be blank, even though it has a title. Use this option if you want to place some graphics on top of the button, or to make a transparent button completely invisible. The title will be invisible so that it will not interfere with the graphics, but the procedure can still find out the title with the `info("trigger")` function.

Click/Release

Unlike most options, Click / Release is enabled by default—you have to turn it off if you don't want it. When this option is enabled, the button acts like a normal button—it highlights when you press on it, then activates (triggers a procedure) when you release the mouse.



If you decide you don't want to activate the button you can pull the mouse away before you release.



When the Click / Release option is disabled, there is no highlighting when you press the mouse. Instead, the procedure is triggered immediately as soon as you press the mouse. There's no chance to back out by pulling away from the button.

Color Options

Like other graphic objects, you can assign any color to a Push Button (see "Color" on page 526). Unlike most other types of objects, however, you can control what portions of the button are displayed in color. (However, if you are using the Standard Push Button style then these color options are not available.) The four options are:

Title: If this option is checked, the title will be displayed in color, otherwise the title will be displayed in black.



Border: If this option is checked, the border will be displayed in color, otherwise the border will be displayed in black. The option can be used with 2D or 3D buttons, like this.

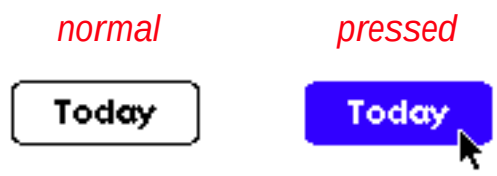


Fill: This option only applies to 3D buttons. If the Fill option is checked, the body of the button will be displayed in color instead of gray.

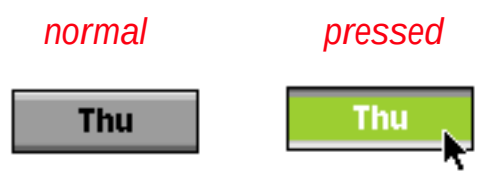


This option only looks good with a few very light colors (use your judgement).

Highlight: If this option is checked, the button will highlight in color when you press on it. For 2D buttons, Panorama simply displays the highlight color when the button is pressed (without this option the button will turn black when pressed).



For 3D buttons, Panorama will use the selected color instead of the dark gray it normally uses.

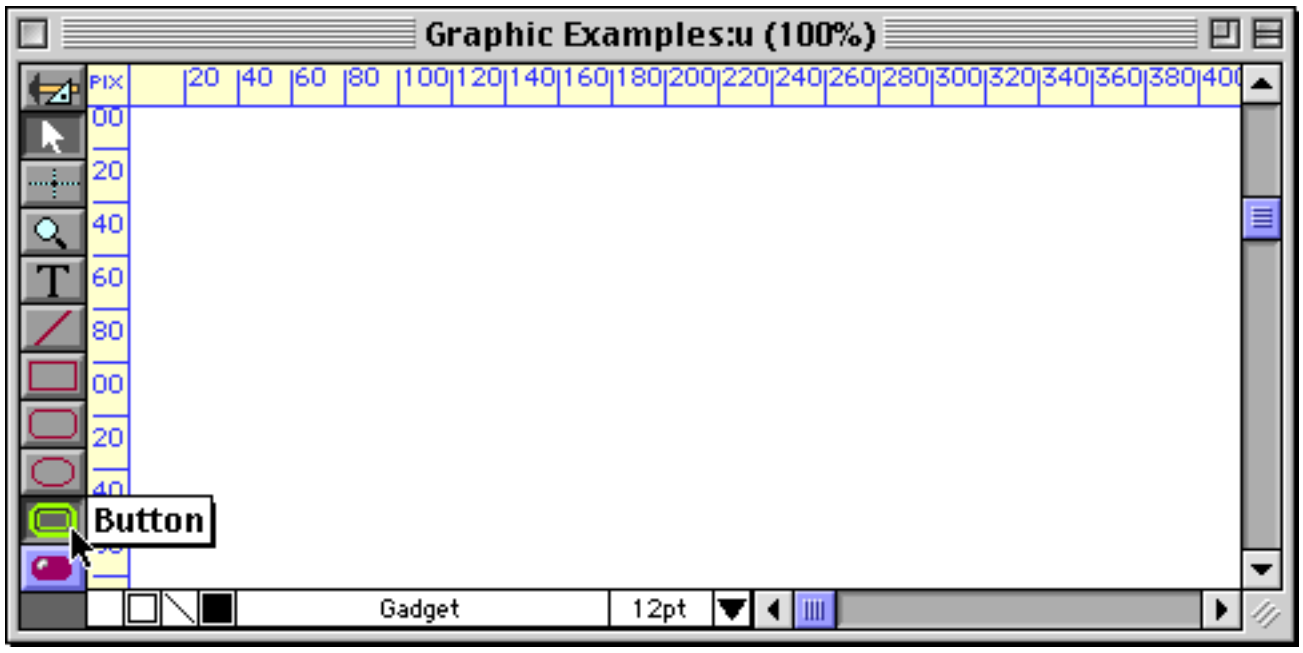


Note: Both 2D and 3D buttons will ignore this option if the Click / Release option is not checked.

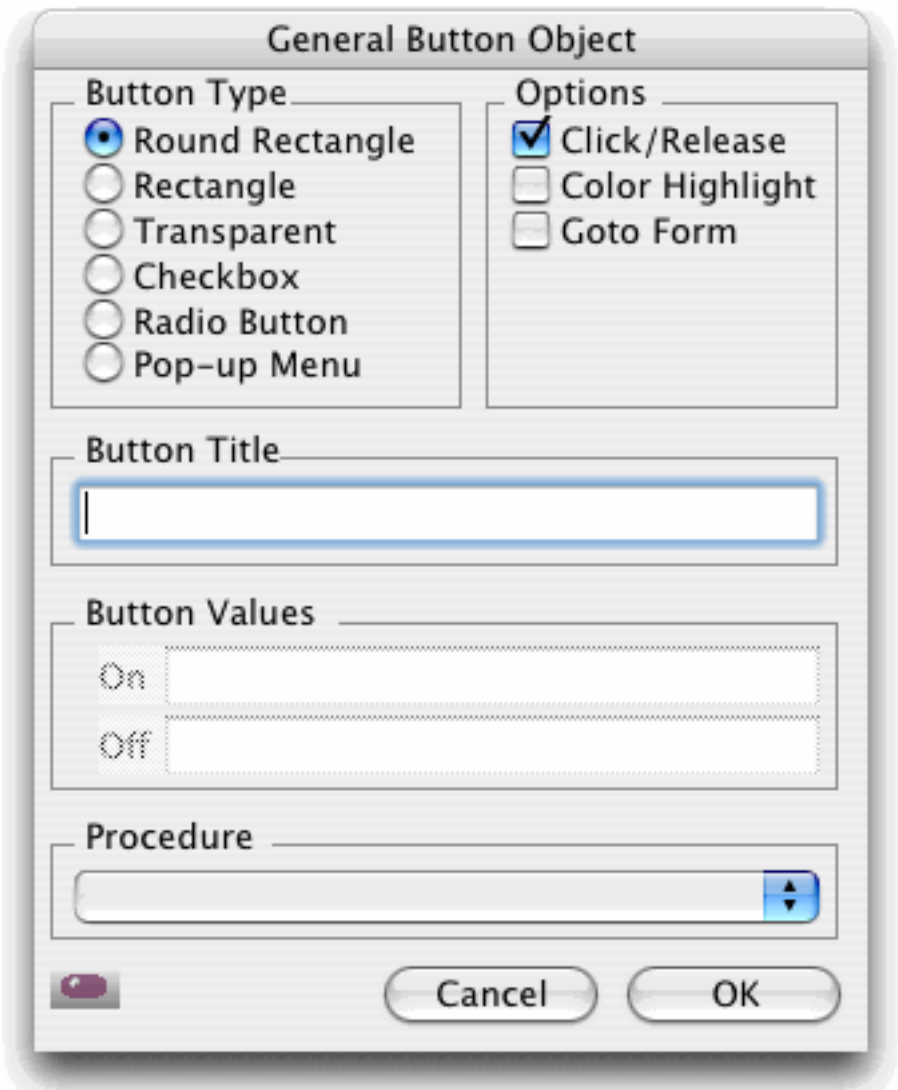
"Classic" Push Buttons

In addition to the **Super Push Button** object described in the previous section Panorama also has a "classic" Button object. When Super Push Button objects were added as part of Panorama 3.0, "classic" Push Button objects were retained for compatibility with older databases. For most applications we recommend that you use Super Push Button for new applications.

Using "classic" Button objects is very similar to working with Super Push Button objects. The **Button** tool looks like a button and is part of the standard Panorama tool palette.



To create a “classic” Button object select this tool and drag the mouse across the form (see “Super Object Push Button” on page 813). The configuration dialog for a “classic” Button object looks like this.



The “classic” button object can actually create three types of buttons: push buttons (round rect, rectangle, or transparent), data buttons (checkbox or radio button) and pop-up menus. In this section we’ll just concentrate on push buttons. See ““Classic” Checkbox and Radio Buttons” on page 846 and ““Classic” Pop-Up Buttons” on page 861 to learn about the other capabilities of the “classic” button object.

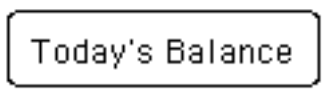
At the top of the dialog is a pop-up menu that allows you to select the procedure triggered by this button. The button can trigger any procedure in the current database. (If you haven’t created the procedure yet you can still create the button, then go back later and choose the procedure.)

The next line is the button title. This is the title that appears centered on the button. The triggered procedure can find out the button title with the `info("trigger")` function. See “Button Title” on page 818 to learn more about this function and button titles.

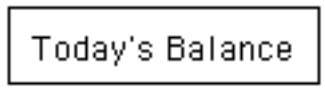
The On and Off options are only used for checkboxes and radio buttons. These options will remain dim as long as one of the push button options (round rect, rectangle, or transparent) are selected.

There are three classic push button styles: round rect, rectangle and transparent.

Rounded Rectangle: This is a standard two-dimensional button. Panorama will display the name of the button inside the button. If you want this to look like a standard Macintosh button, you should use Chicago 12 point type.



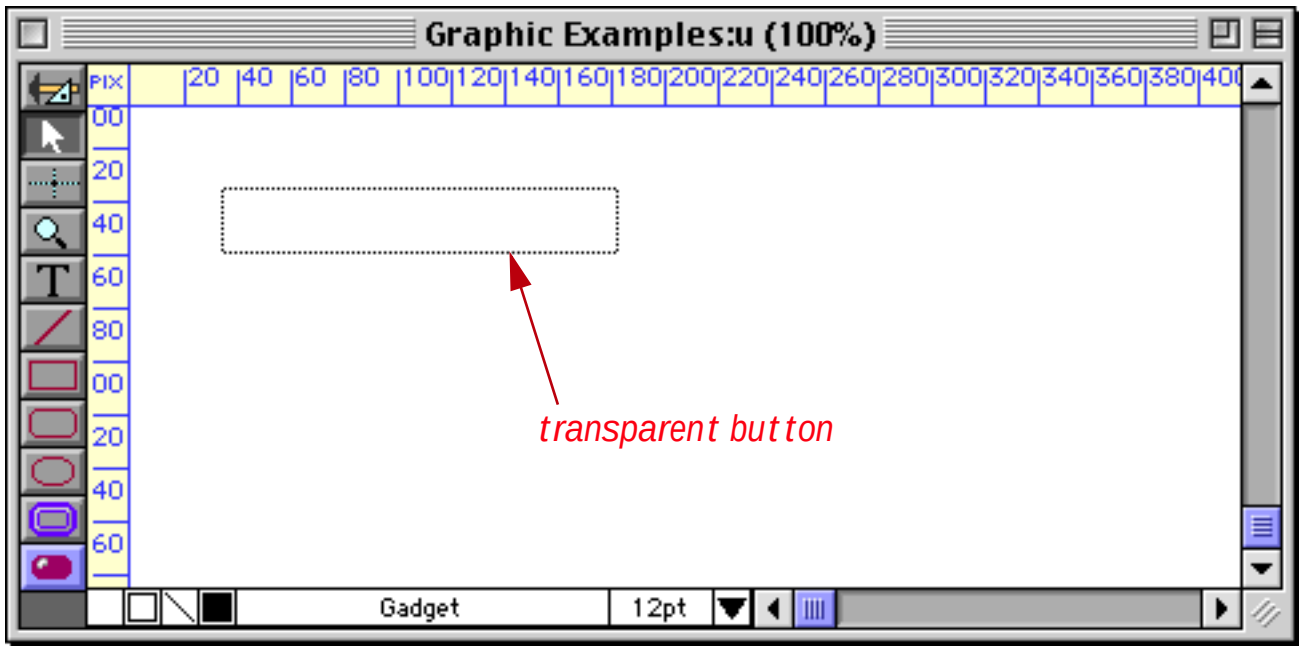
Rectangle: This button style has square corners.



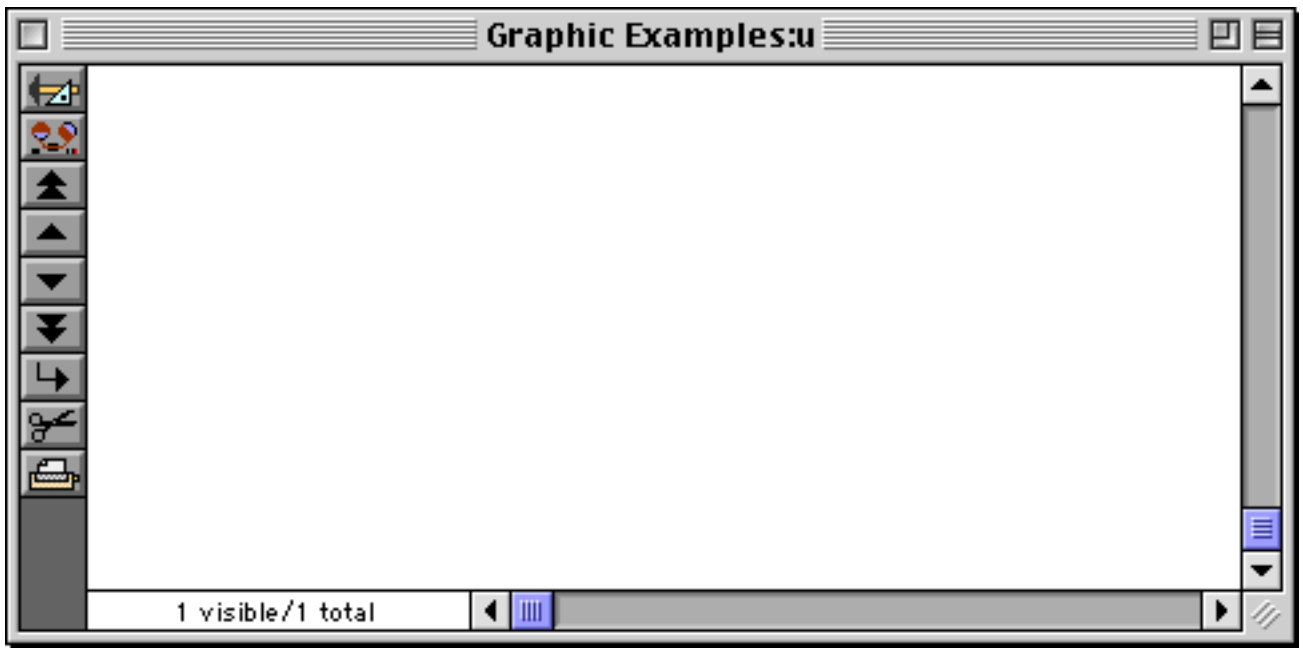
Transparent buttons are described in the next section.

Transparent Push Buttons

A transparent button is just that—transparent. It's invisible (including the title). However, in Graphics Mode Panorama does display a dotted line around the button to help you locate it for editing.

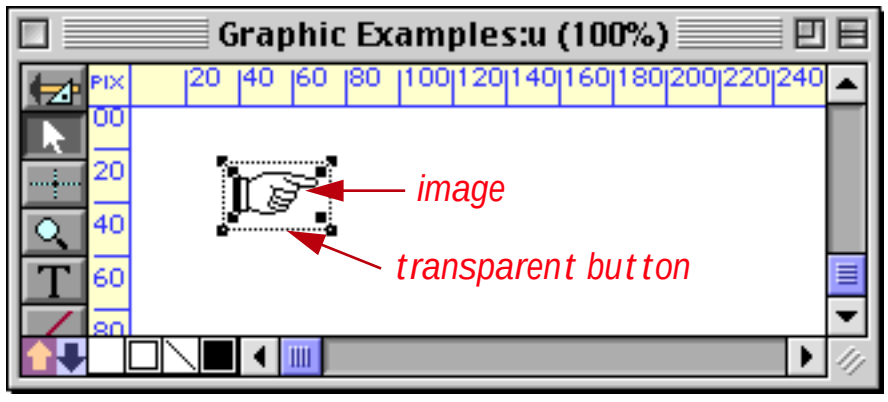


When you switch to Data Access Mode the button completely disappears.

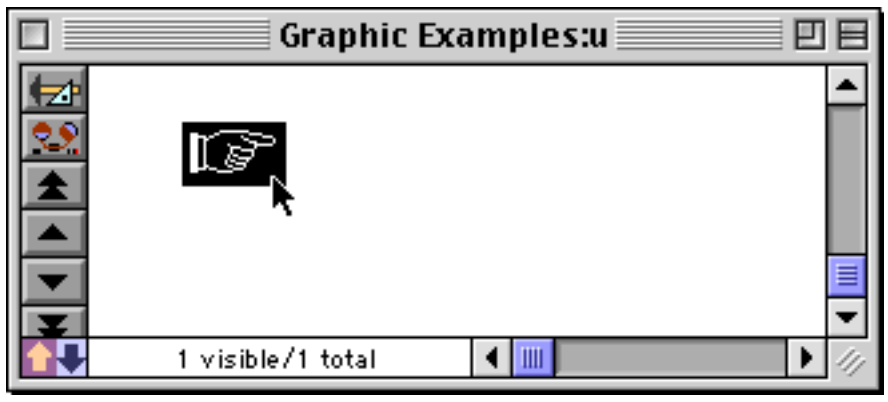


Even though the button is invisible, you can still click on it and trigger the procedure if you know where it is.

Sometimes you may really want a button to be completely invisible—for example a secret button that only you know about. However, it's more common to overlay a transparent button over a graphic image, turning the image into a button.



In Data Access Mode you can now click on the "hand button" to trigger a procedure.

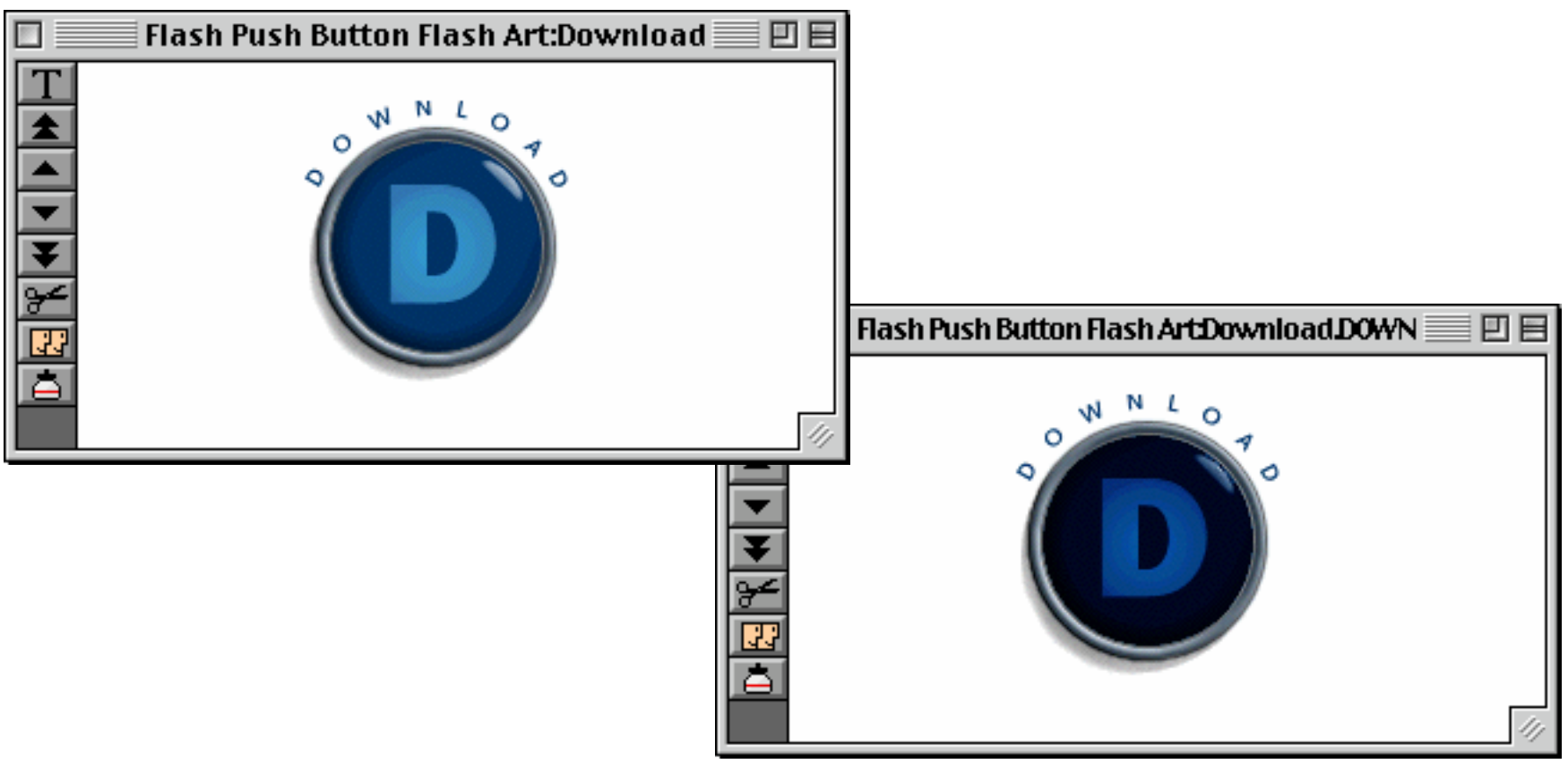


The illustration above shows what happens when you click on a transparent button with the Click/Release option turned on. If this option is turned off, the procedure will trigger immediately when the mouse is clicked, without highlighting (reverse image) the button. See "Click/Release" on page 819.

Flash Art™ Push Button SuperObjects™

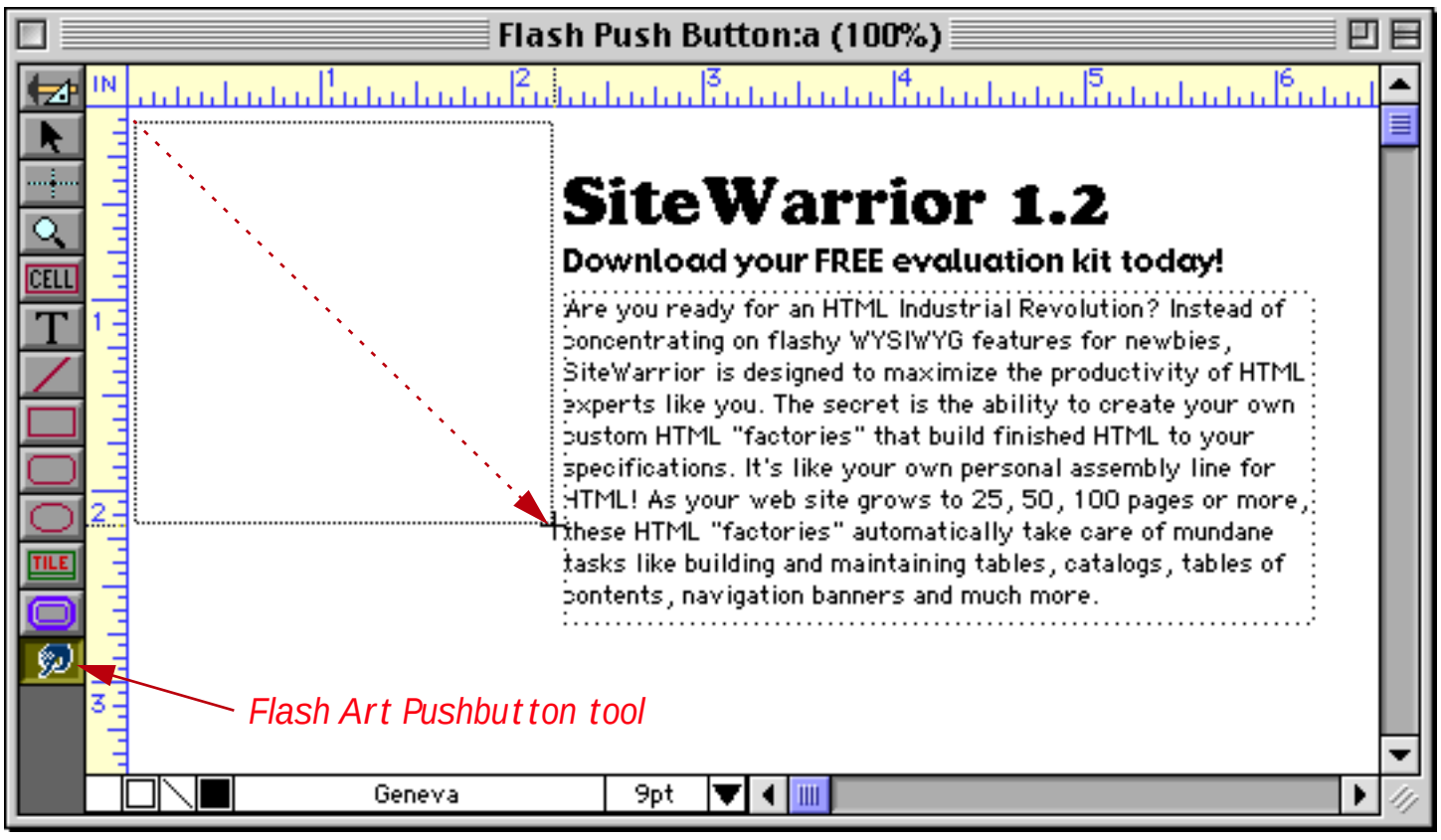
The Flash Art Push Button SuperObject™ lets you use images in the Flash Art gallery (see "The Flash Art Scrapbook (Gallery)" on page 762) to create push buttons that trigger procedures. You can use any application that can create PICT images to draw your button: Photoshop, Canvas, Freehand, etc. It takes more work to create a Flash Art™ Push Button, but you have the flexibility to create any kind of button you want!

The first step in creating a Flash Art Push Button is to create two Flash Art images: the first showing the button in it's "normal" (unpressed) state and the second in its "activated" (pressed) state. When the button is clicked, Panorama will automatically switch between these two pictures as the button is pressed with the mouse. (The two pictures should have exactly the same dimensions or you will notice a shift as the mouse is pressed on the button.) The second picture must have the same name as the first picture, but with .DOWN added to the end. For example, if the "normal" picture is called Download, the "activated" picture must be called Download.DOWN.

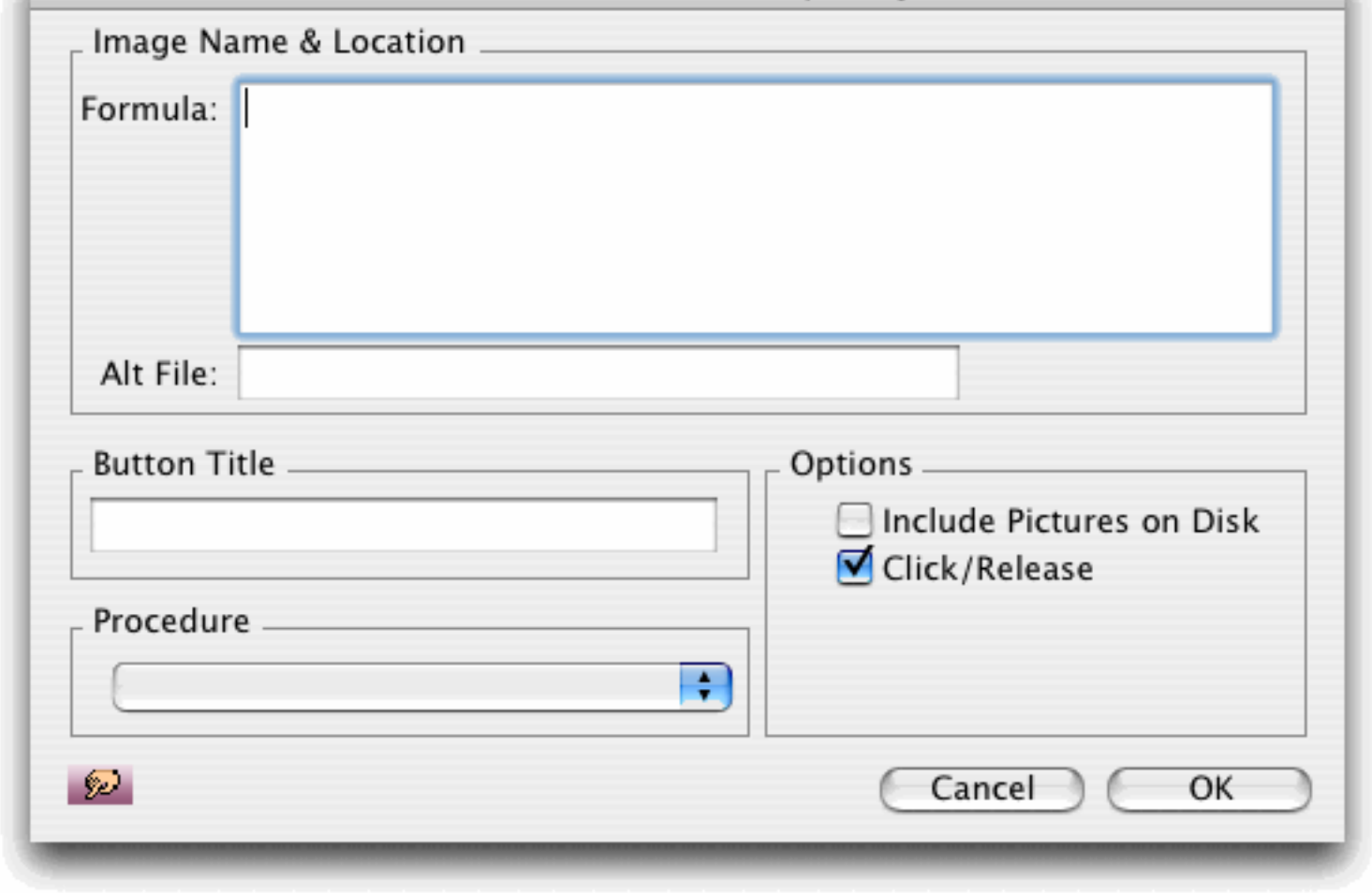


(Note: The Flash Art Push Button will display your pictures actual size, without any scaling or resizing. If the button you create is smaller than the picture, the picture will be cropped. If the button is larger than the picture, the picture will appear in the upper left hand corner of the button.)

Once the pictures have been created, you are ready to create the button itself. Select the **Flash Art Push Button** tool and drag across the form to create the button. (If the tool is not currently installed, use the **Tool Palette** dialog to install it — see "Customizing the Tool Palette" on page 497.)



When you release the mouse the configuration dialog will appear.



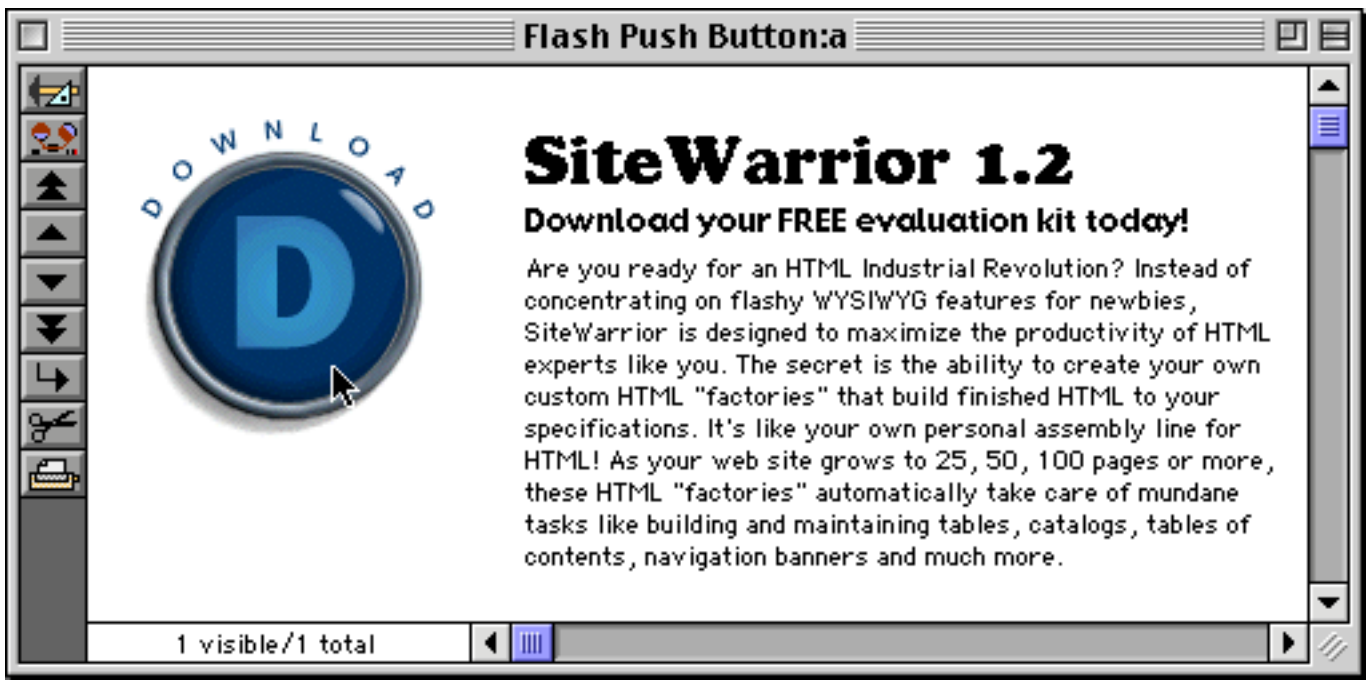
Select the procedure to be triggered and enter the title for the button (the title is not displayed but is returned by the `info("trigger")` function).

The Formula box is used to specify the pair of pictures displayed by this button. If you always want to use the same two pictures, simply enter the name of the "normal" picture surrounded by quotes, for example "Download". The formula can also be set up so that the picture used for this button changes depending on circumstances—for example, you might want to use a different button for black and white vs. color monitors (using the `info("windowdepth")` function) or a different button in the morning vs. the afternoon.

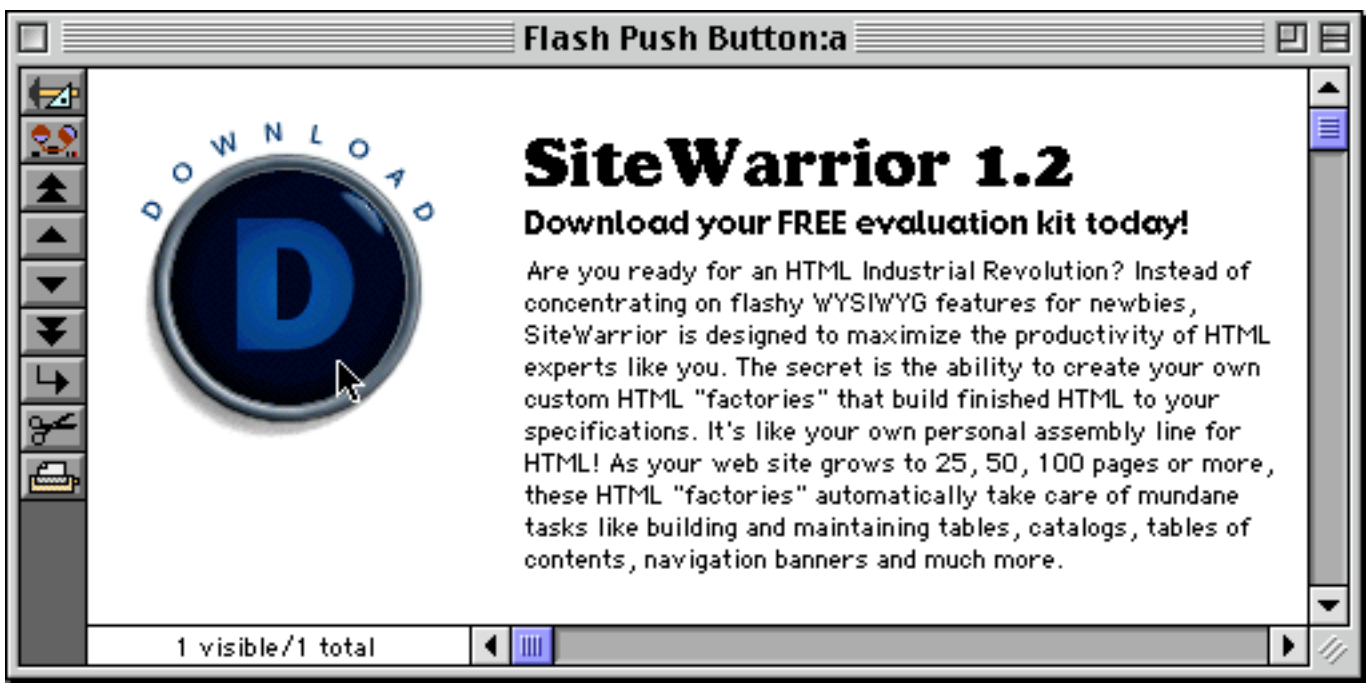
The Alt File option allows the button to display pictures stored in the Flash Art gallery of another database (the other database must be open). Enter the name of the database here. See "Alt File" on page 779 for more information on this option.

The Include Pictures on Disk option allows the button to display pictures that have been stored as PICT files on the disk. Normally these files should be in the same folder as the database itself, however, you may supply a file path as part of the Formula (see "Displaying Images Directly From Disk Files" on page 767).

The Flash Art Push Button tool allows you to make virtually any custom button you want. Here's what our finished button looks like in use.



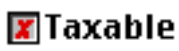
When you click on the button, the second image appears. This gives you complete control over the appearance and highlighting of the button.



Your imagination is the only limit to the buttons you can create.

Data Buttons

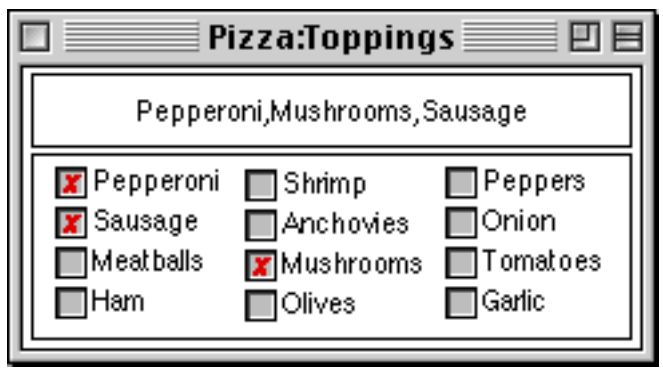
Data buttons are buttons that have a value associated with them. The value may be stored in a database field or in a variable (see “Variables” on page 1199 and “Variables” on page 1370). An on / off data button is usually called a **checkbox**.



Data buttons may be grouped together as **radio buttons**. Only one button in the group may be selected at a time.

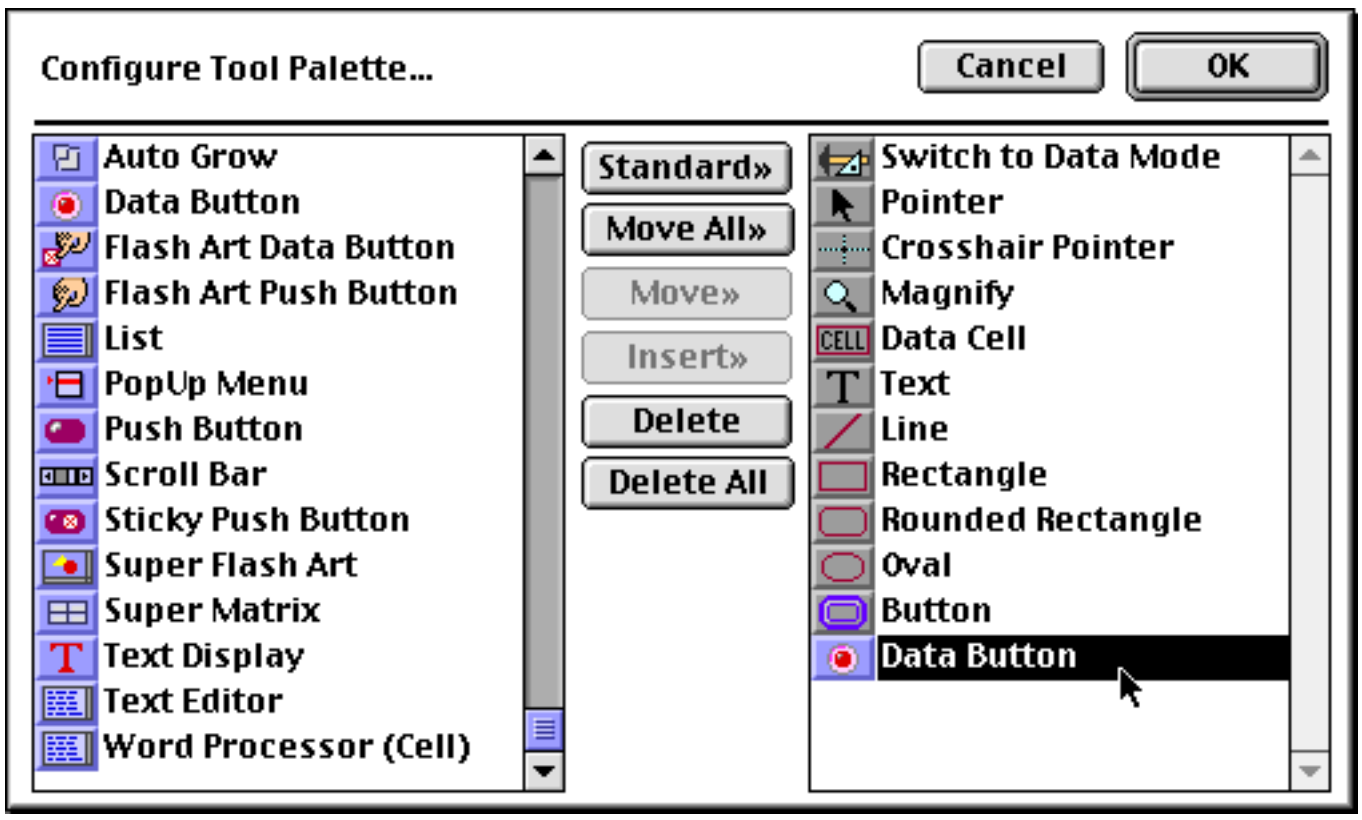


Panorama also allows you to build a group of buttons where more than one member of the group can be selected at a time. See “Multiple Value Button Groups” on page 835 to learn more about this option.

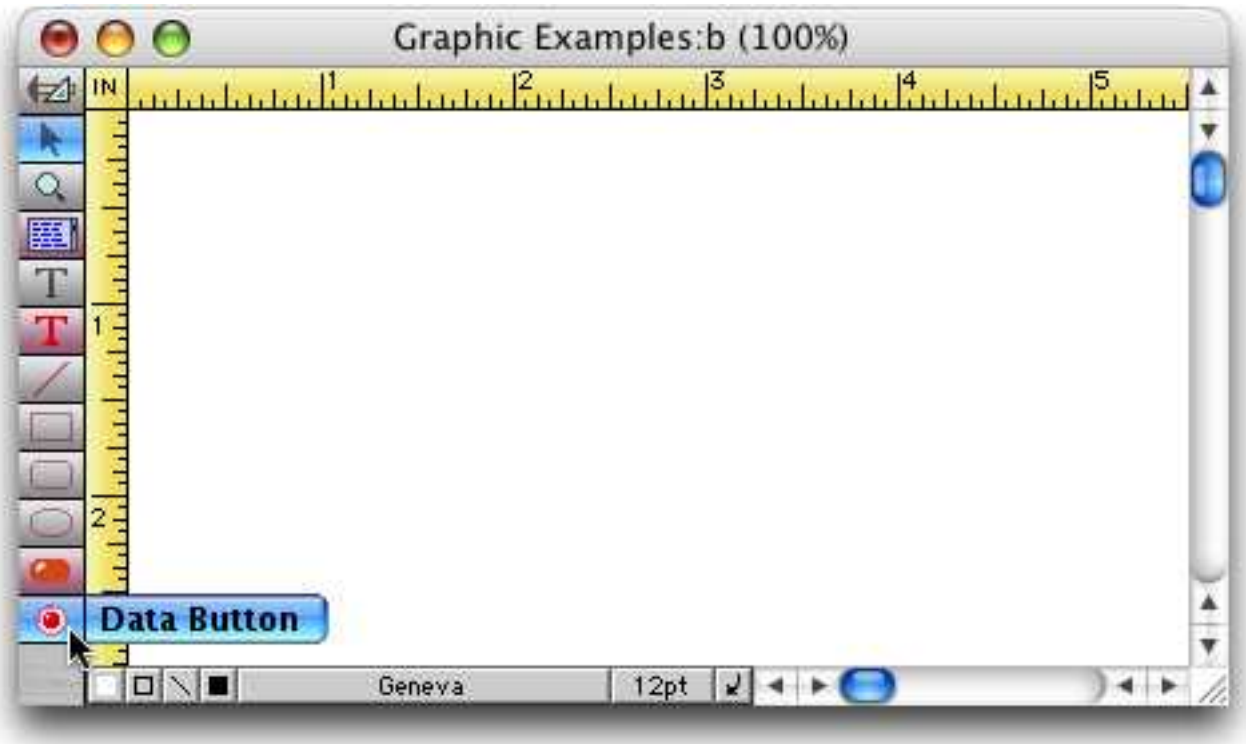


Data Button SuperObjects™

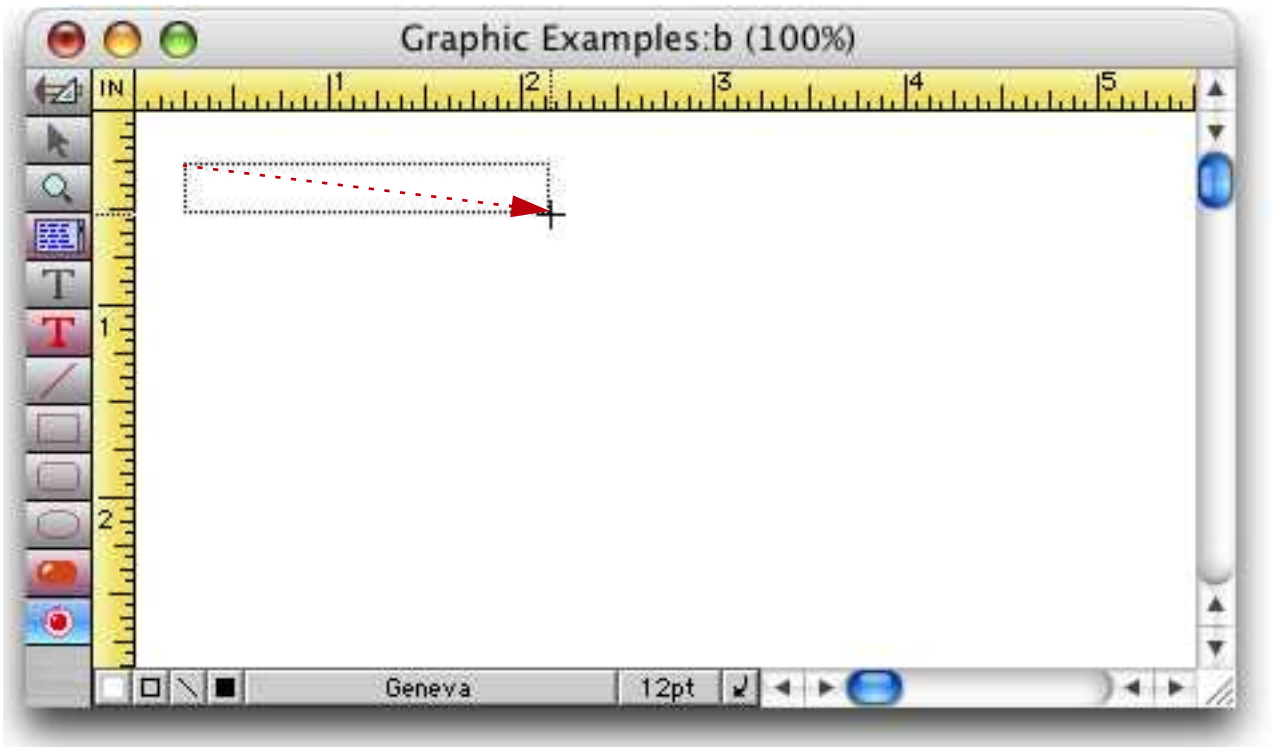
The Data Button SuperObject™ tool can be used to create checkboxes and radio buttons in a variety of styles. Unlike the "classic" button tool, this tool can work with global variables as well as fields. The SuperObject Data Button tool is not in the default tool palette, so you'll need to move the use the Tool Palette dialog to add this tool to the palette if it is not already there (see "Customizing the Tool Palette" on page 497).



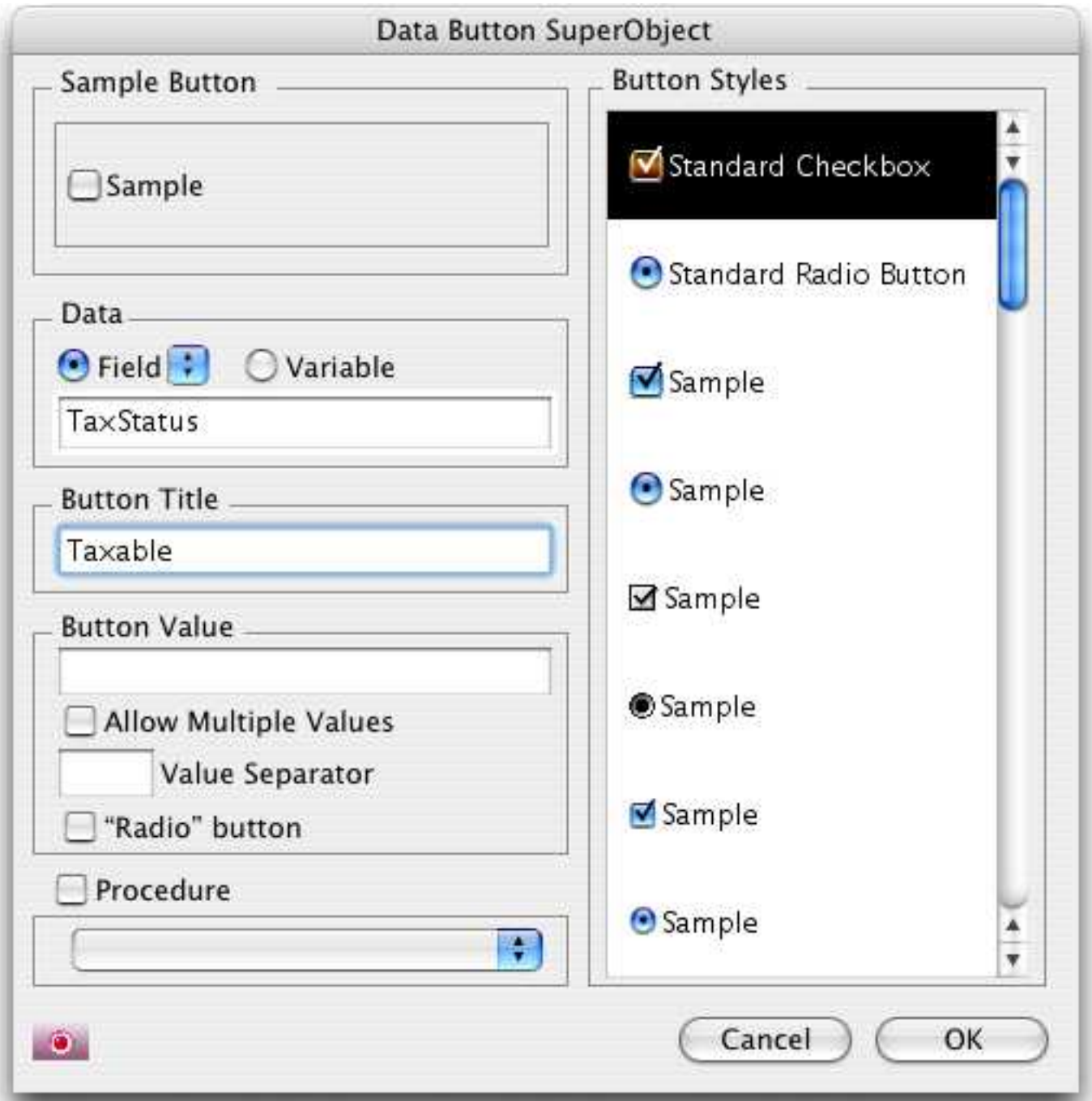
Now that the tool is added to the palette you can select it.



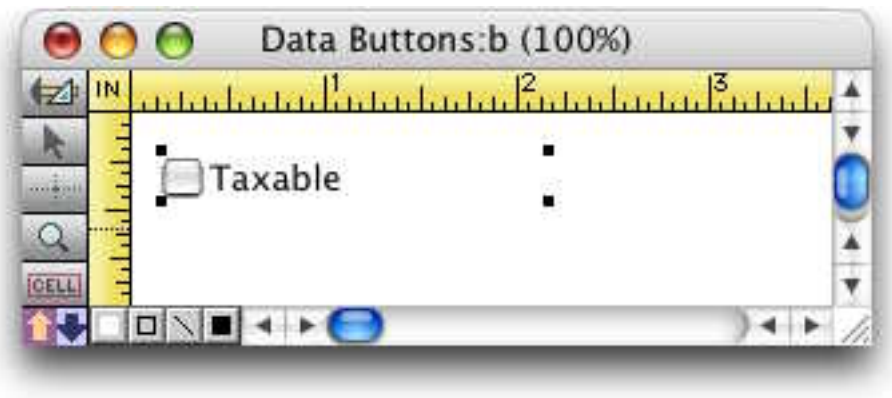
Once the tool is selected, drag the mouse across the form in the location where you want to create the button.



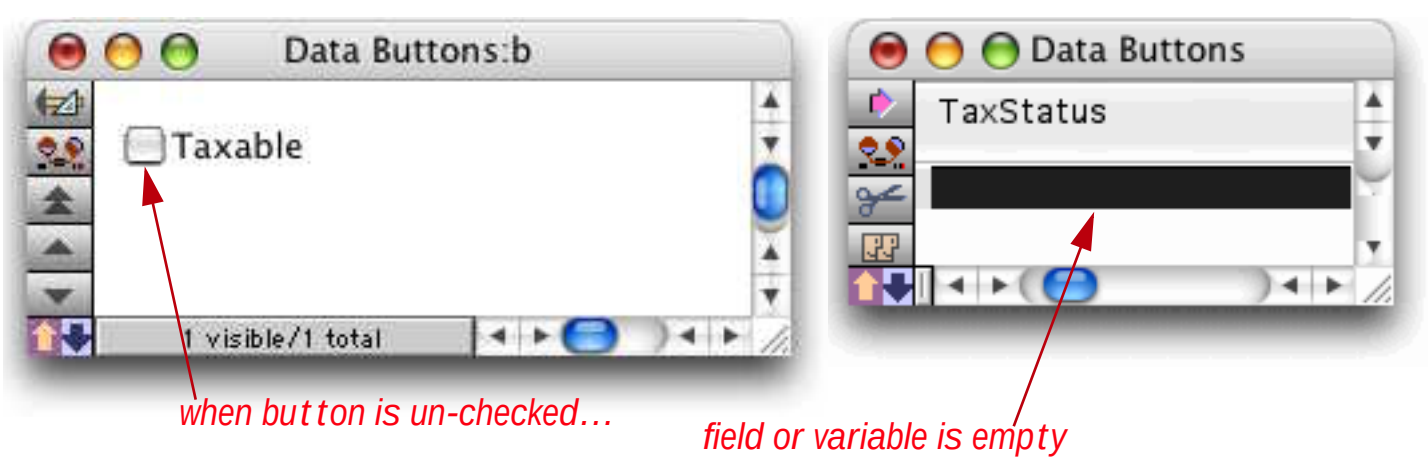
When you release the mouse, the SuperObject Data Button configuration dialog will appear.



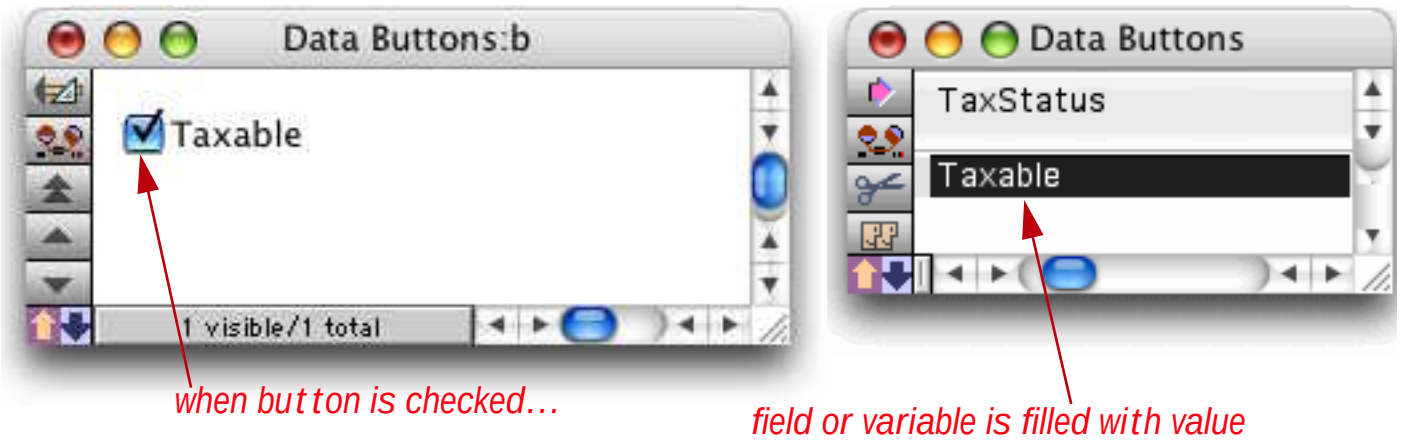
At a minimum you must select a field or variable to store the value, and enter a Title. When you press **OK** the new button will appear on the form.



To try out the button, switch to Data Access Mode. Since this button's value is stored in a field we can use the data sheet to watch the value as the button is checked and unchecked.



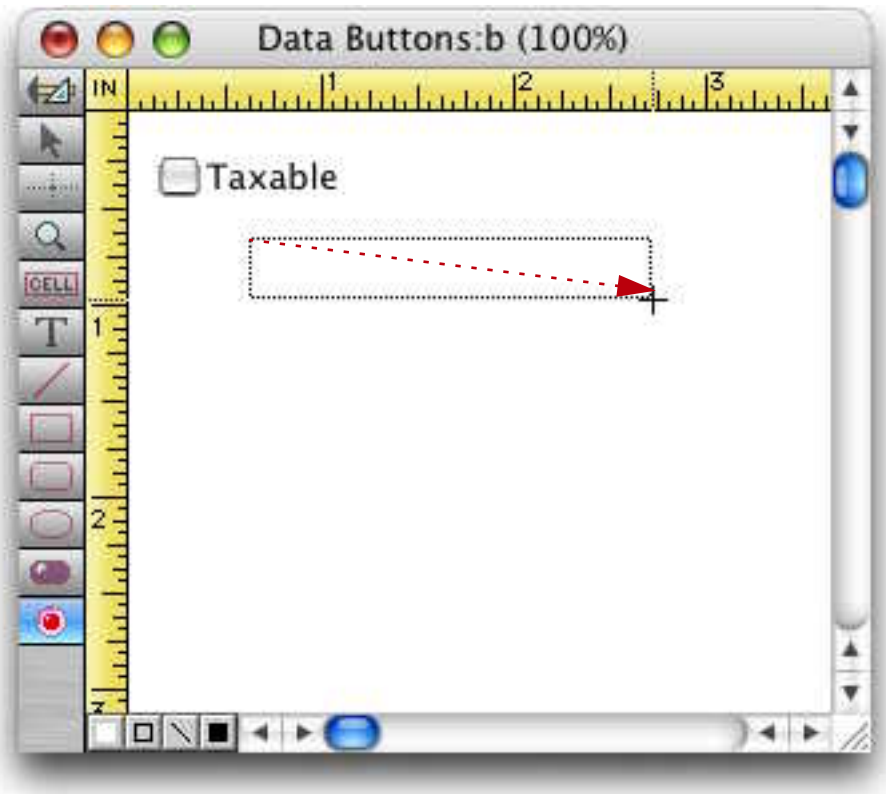
Clicking on the checkbox causes Panorama to fill the field or variable with the value.



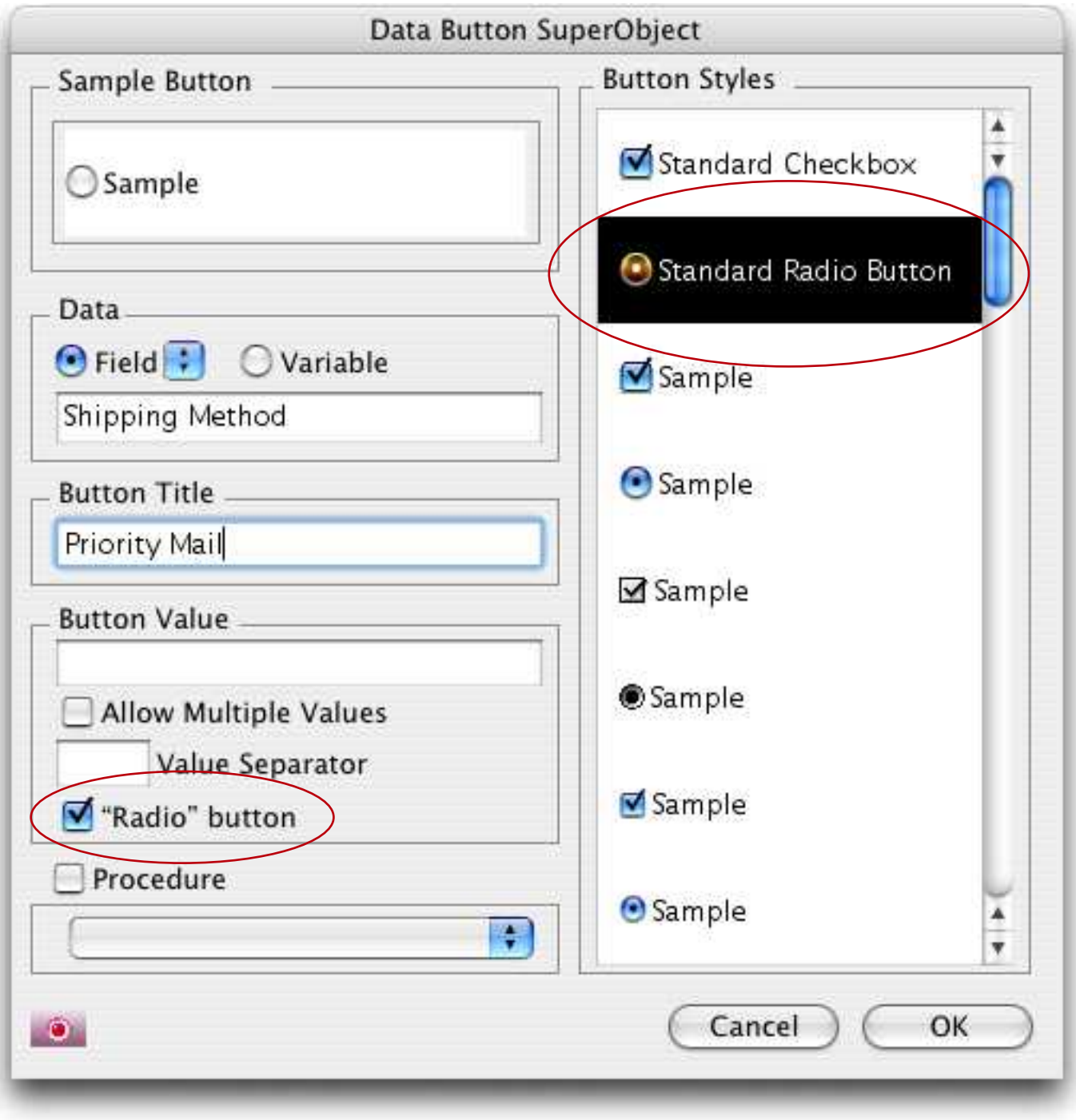
If you click again to un-check the button the field or variable will become empty again. (You may wonder if typing the value into the Data Sheet will cause the button to become checked. The answer is yes!)

Creating a Group of Radio Buttons

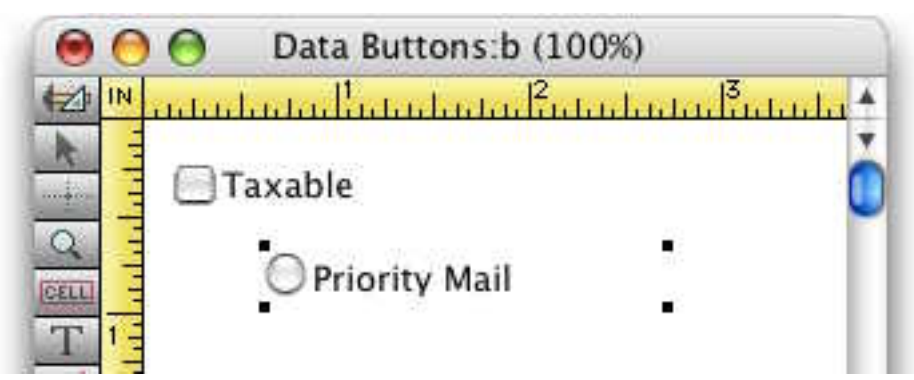
The easiest way to create radio buttons is to create a single button and then make copies. For example, suppose you want to make radio buttons for shipping options — Priority Mail, UPS Next Day Air, Federal Express, Airborne and DHL. Start by selecting the Data Button tool (see previous section) and dragging to create the first button.



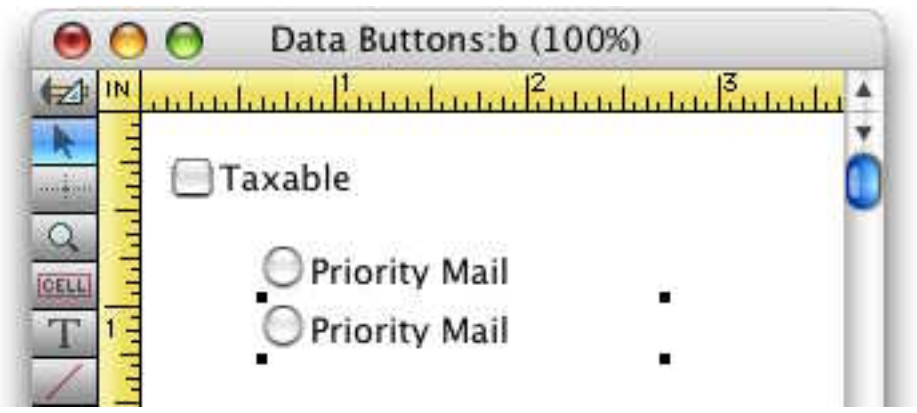
The field or variable and title are set up exactly the same as for a checkbox. You should also enable the "Radio" button option, and select a radio button style from the list of styles on the right hand side of the dialog. (Note: The visual style does not affect the operation of the button, so Panorama will happily let you create a radio button that looks like a checkbox, or vice versa.)



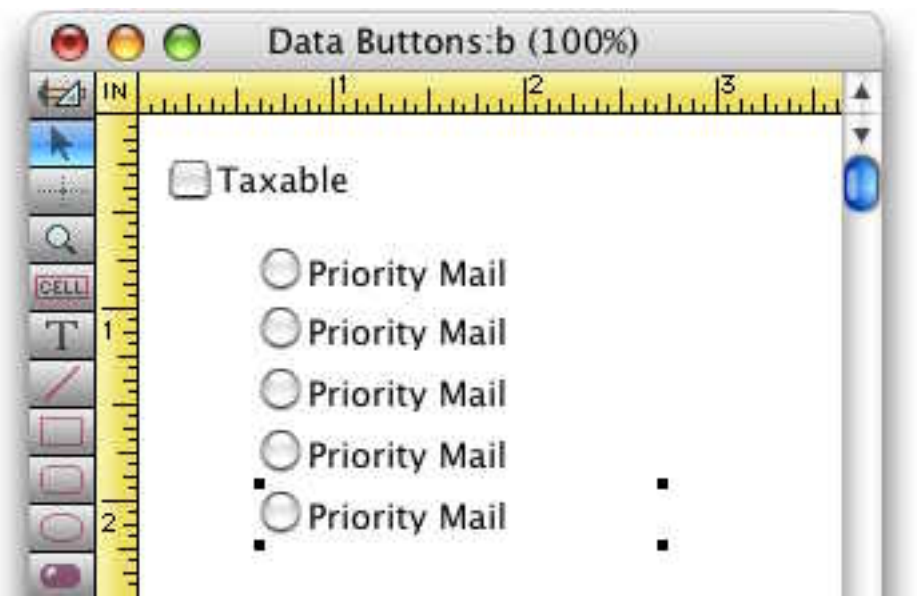
Press **OK** to create the first radio button.



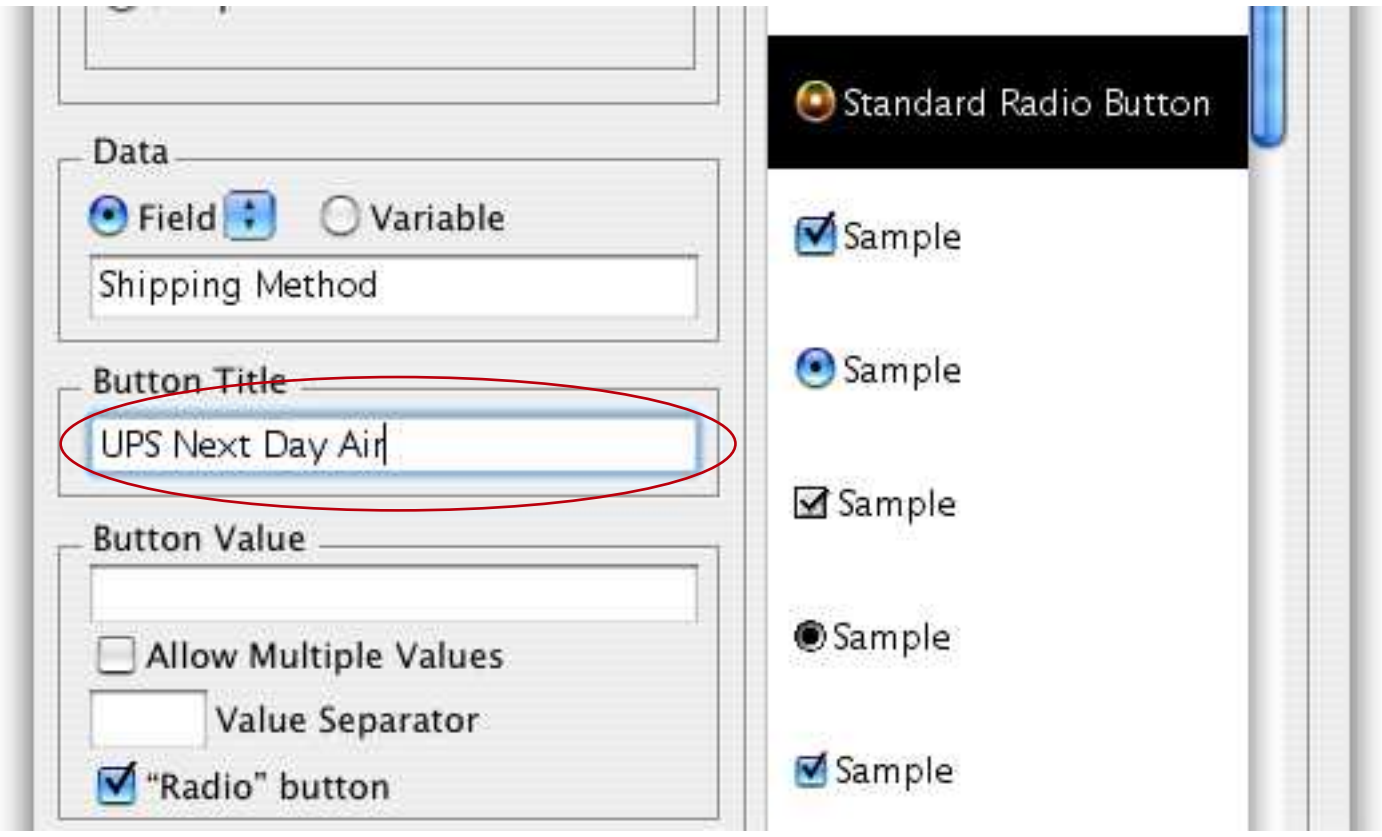
To duplicate this button, make sure that the **Pointer** tool is selected and hold down the **Shift** key and the **Option** (Mac) / **Alt** (PC) key while you drag the button. When you release the mouse Panorama creates a copy of the selected object (see "Drag Duplicating" on page 561).



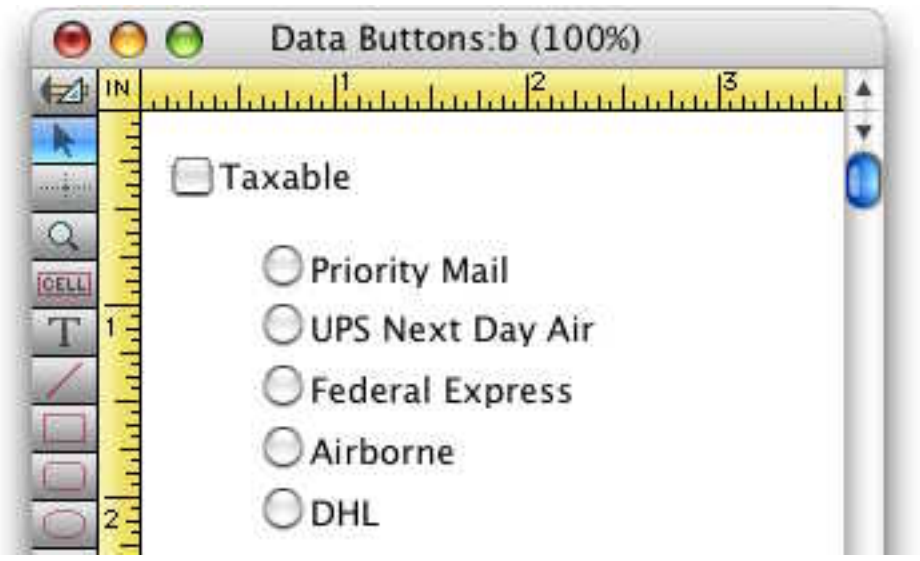
To make additional copies simply use the **Duplicate** command (see "Step and Repeat" on page 562).



Now you need to go back and change the title of each button (except the first). To change the title, simply double click on the button, then fill in the new title.

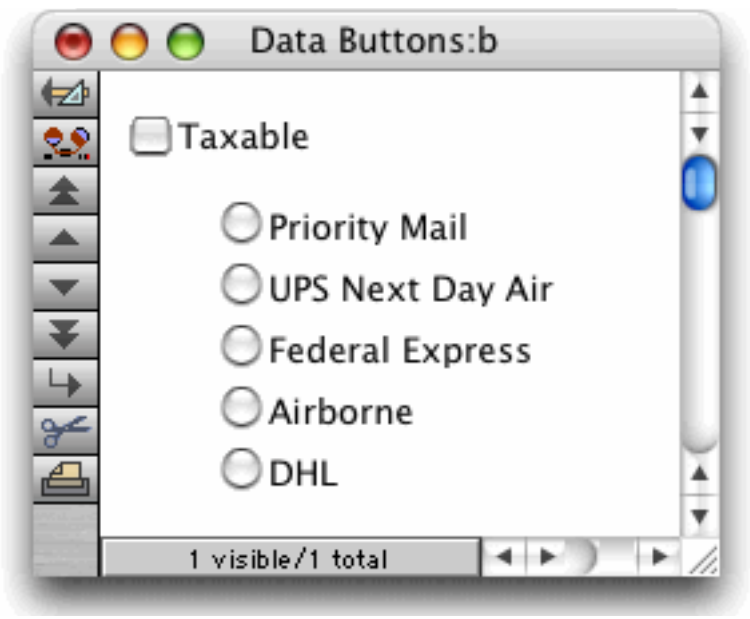


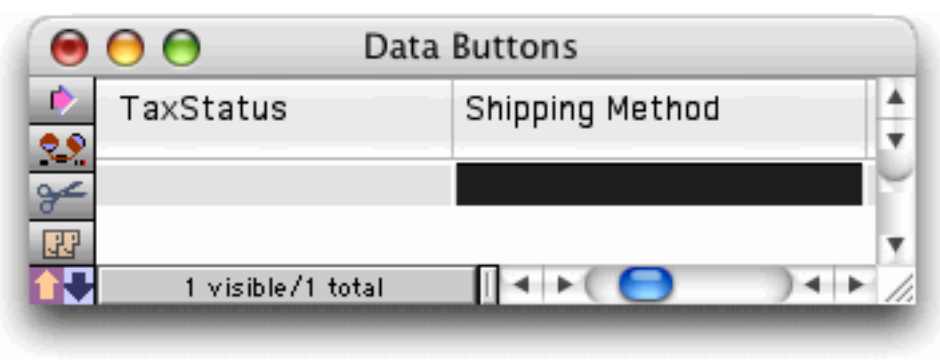
All the other options remain the same — only the title needs to be changed. Repeat this process for each button in the group.



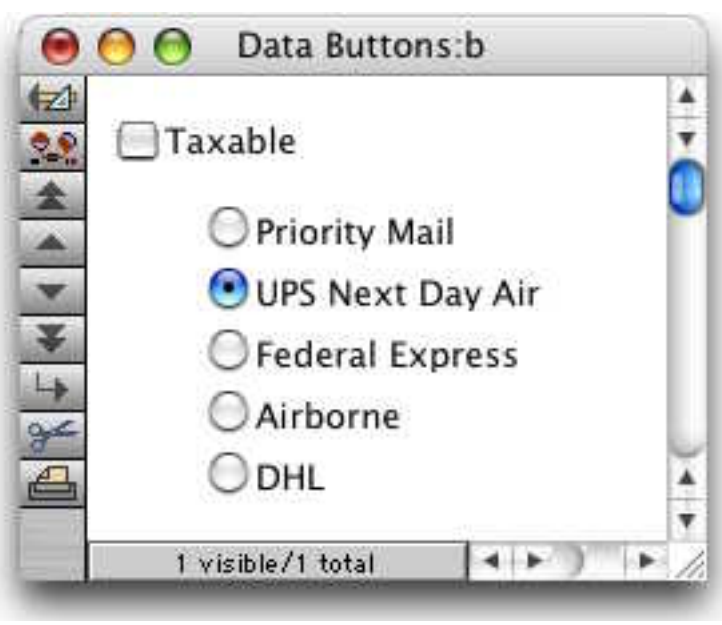
If necessary, you can adjust the spacing of the buttons with the **Spacing** command (see "Setting Exact Dimensions of Multiple Objects" on page 550 and "Adjusting Spacing Between Multiple Objects" on page 556). You can also adjust the width of the buttons using Cluster Resize (see "Cluster Resize" on page 541).

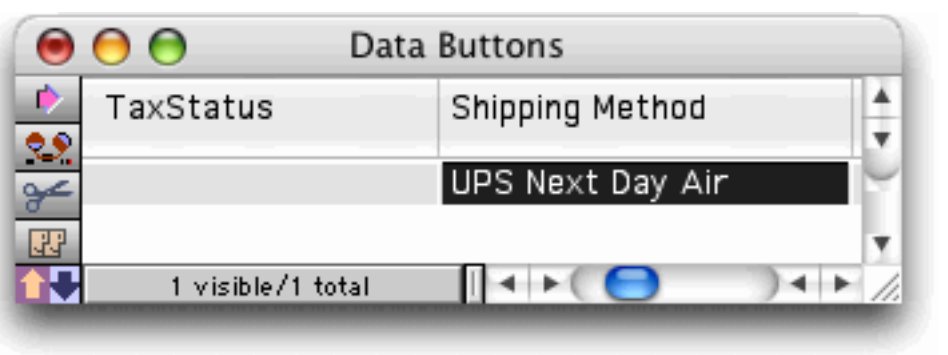
To try out the buttons, switch to Data Access Mode. Since the button's value is stored in a field we can use the data sheet to watch the value as the different buttons in the group are checked. To start out, none of the buttons are checked and the field is empty.





When a button is checked, the corresponding value appears in the field.

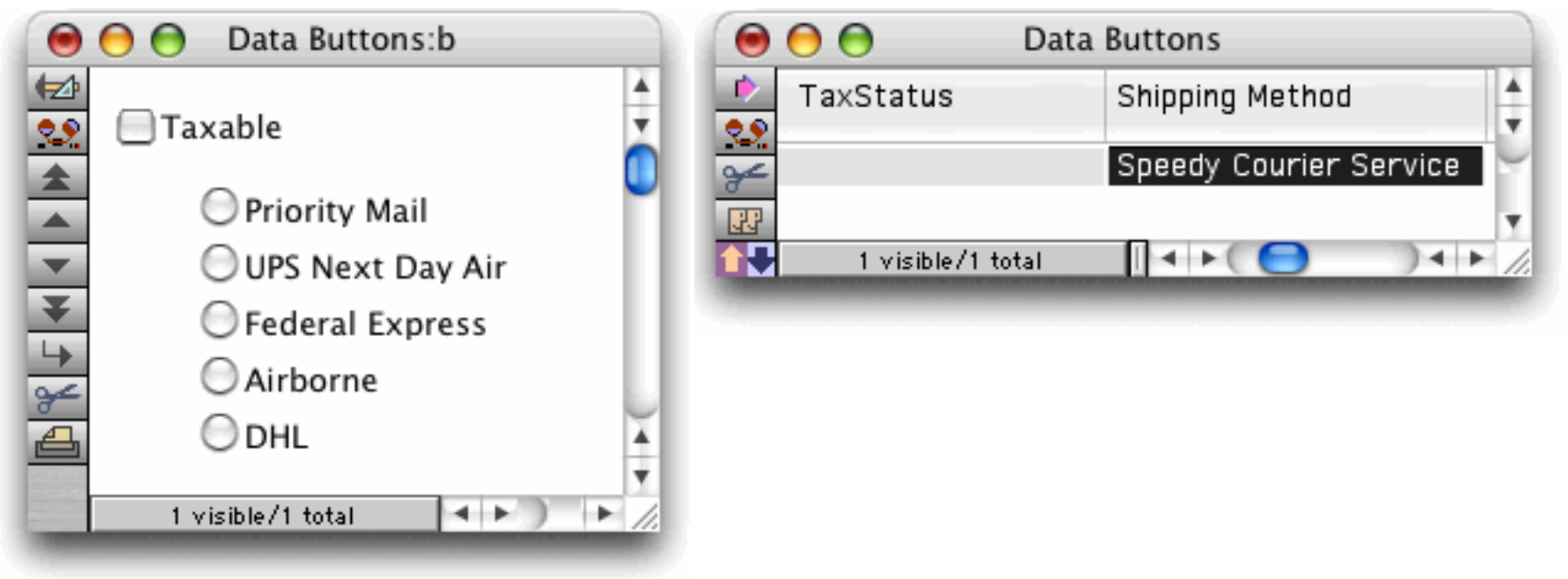




Clicking on any button causes the corresponding value to appear in the field.



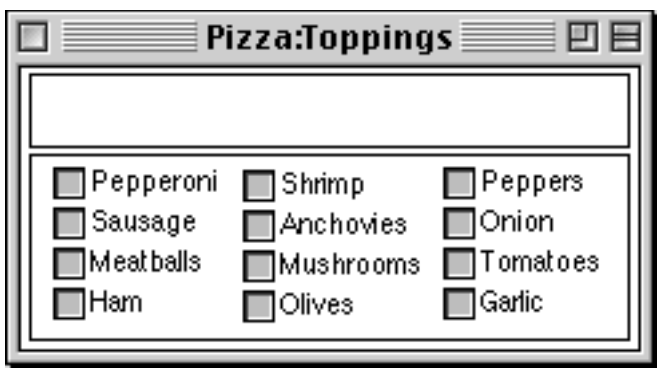
The synchronization between buttons and the field also works in reverse. If you type a value into the data sheet, the corresponding button will be activated. If none of the buttons match the value, all of the buttons will be turned off.



The buttons can also be completely turned off by clearing the contents of the field (making it empty).

Multiple Value Button Groups

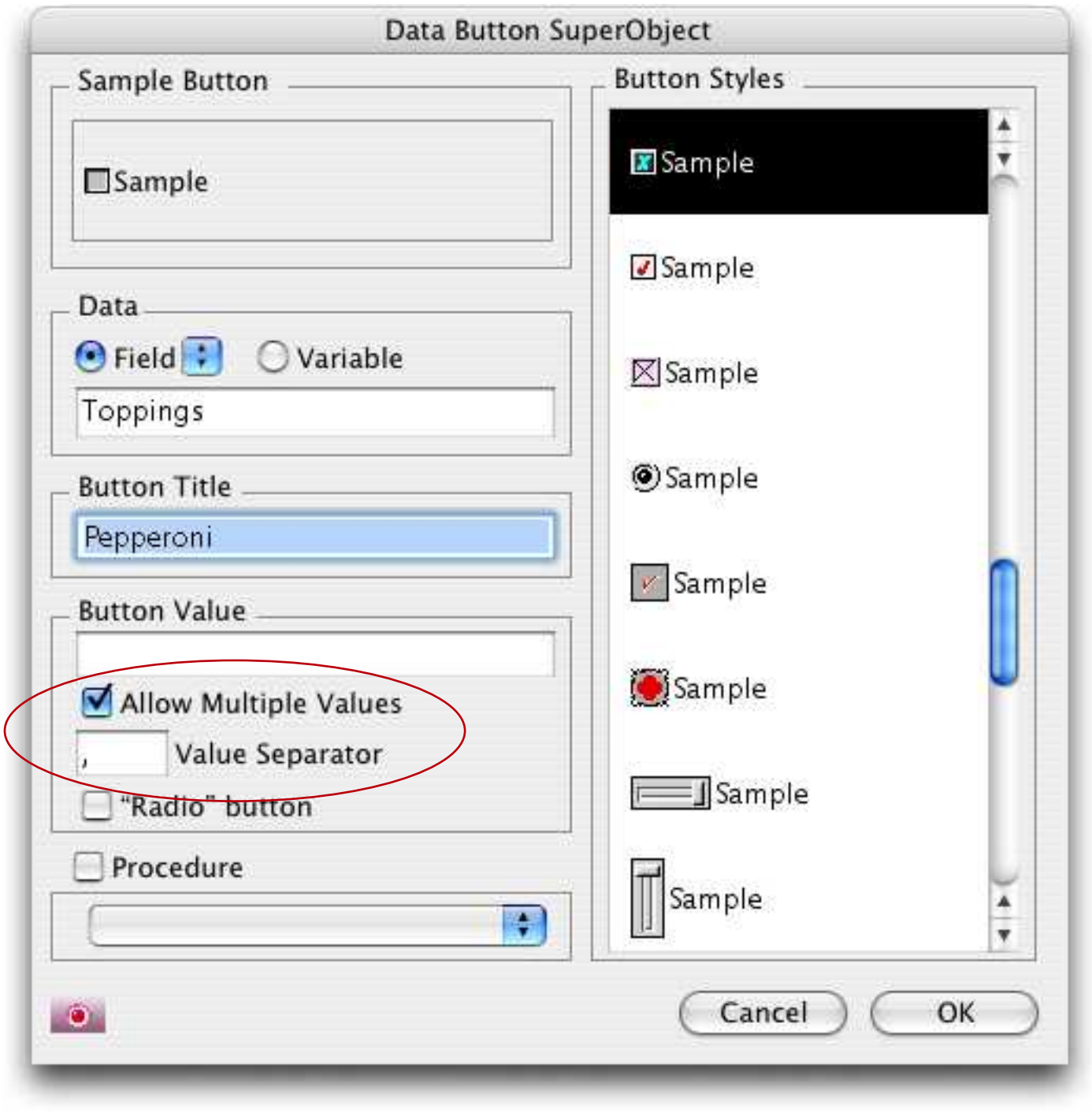
A group of radio buttons works find as long as only one value at a time is valid. You can't ship a package by both Priority Mail and Federal Express, only one or the other! Some applications, however, require that multiple options be selected. For example, consider a group of buttons for selecting pizza toppings.



In this application there may be zero, one, two, or even 12 values at one time! There are two ways to create a group of buttons like this.

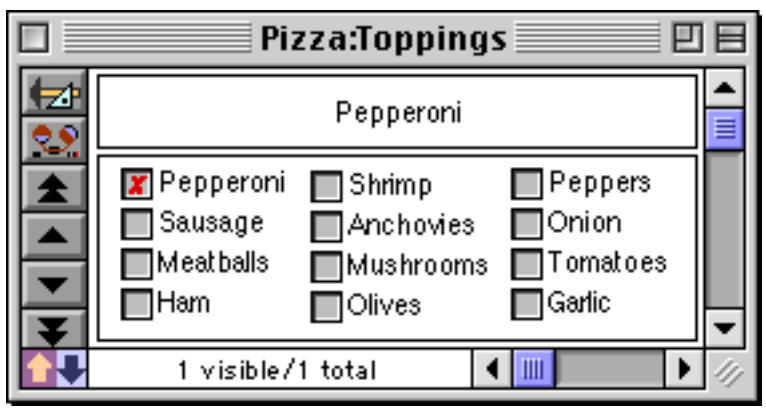
The first method is simply to create a field or variable for each option, and then create a standard checkbox for each field. See “Data Button SuperObjects™” on page 828 to learn how to create a checkbox.

The second method allows you to combine all of the values into a single field. Just as with a group of radio buttons, you’ll start by creating a single button and making copies (see previous section). However, instead of enabling the “Radio” button option you’ll enable the Allow Multiple Values option. You also need to specify what character(s) you want placed between each value. In this example, we’ve chosen a comma.



Set up the first button using the options shown above. Then make copies of the button and adjust the titles, just as described for radio buttons in the previous section.

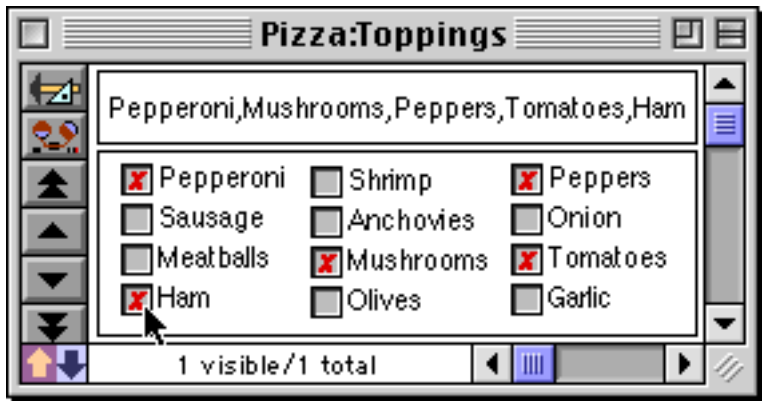
Switch to Data Access Mode to try out the group of buttons. When a single option is clicked the value appears in the field, just as for a standard checkbox. (A Text Display SuperObject has been added to the form to display the value of the Toppings field, see "Text Display SuperObjects™" on page 608 to learn how to create such an object.)



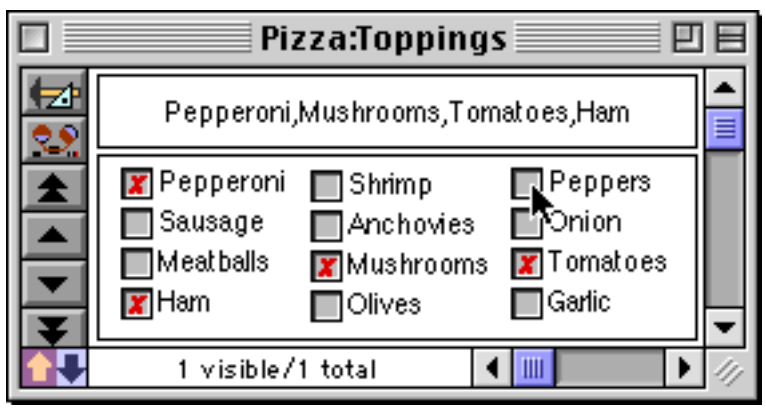
When a second button is clicked, that value is added to the field.



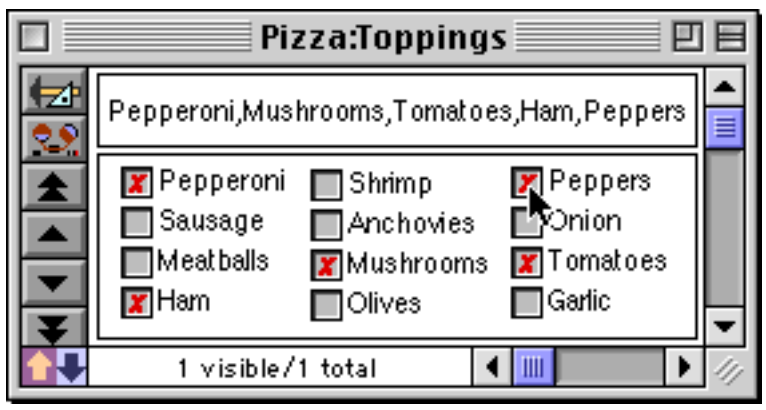
You can keep adding as many values as you like.



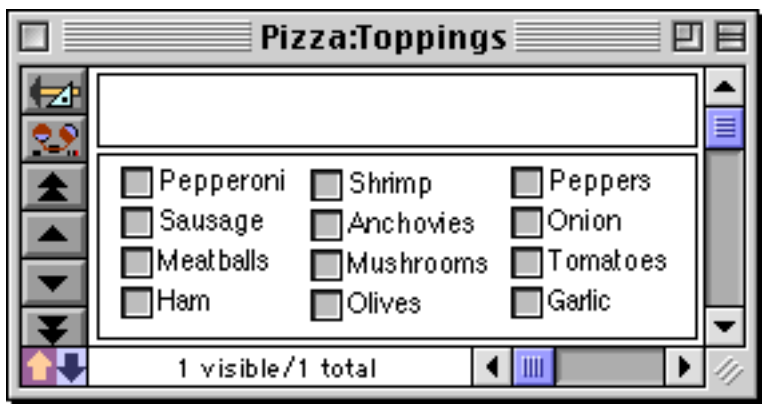
If you un-check a button, that value will be removed from the field (or variable). In this case we have removed Peppers.



If Peppers is re-enabled, it is added to the end of the value. You cannot control the order of the options within the field or variable.



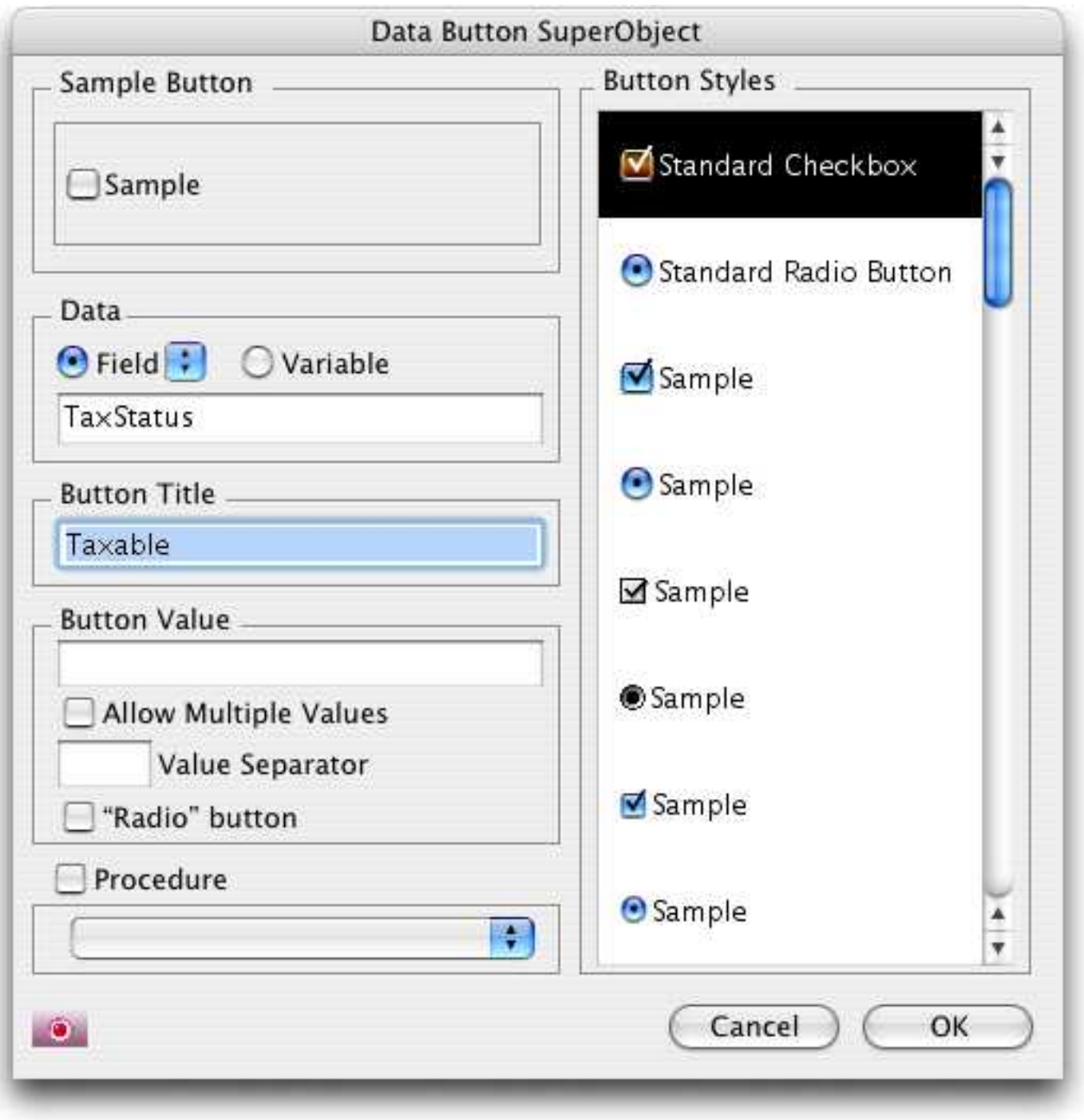
If you un-check all of the buttons the field will become completely empty.



Like other types of buttons, the synchronization between the buttons and the field or variable works both ways. You can type in a value or combination of values into the field or variable and the appropriate buttons will automatically “light-up.”

Super Data Button Options

The left hand side of the Data Button dialog controls the operation of the button, the right hand side controls the appearance of the button.



Data

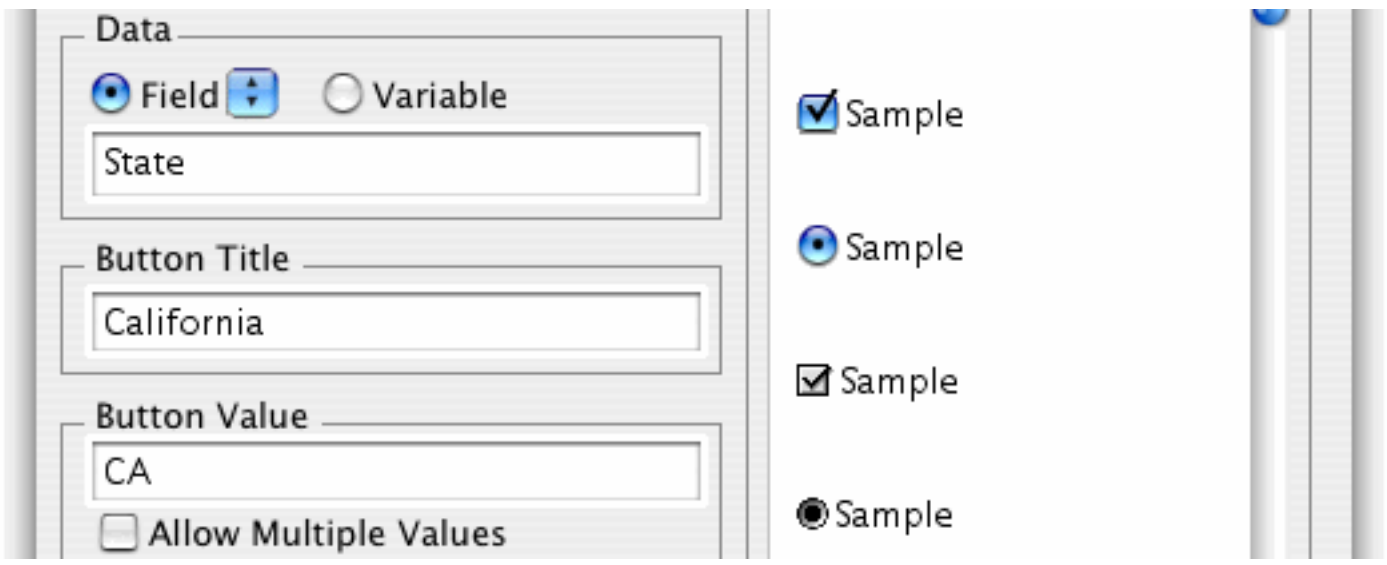
This section of the dialog specifies the field or variable associated with this button. Type the name of the field or variable into the box (or select the field name from the pop-up menu next to the Field radio button). If the button is associated with a variable that has not been created with a procedure, Panorama will automatically create a global variable with this name whenever the button appears. This global variable can be used in formulas and procedures just like any other global variable (see "Variables" on page 1199 and "Variables" on page 1370).

Title

This section of the dialog specifies the title of the button. This is the title that is actually displayed on the screen. If the value is empty (see next section) Panorama will use the title as the value.

Value

This section of the dialog specifies the data value corresponding to this button. Each button can have a single value, which you should type into the box next to the word Value. If you leave this box empty, Panorama will use the Title for the data value (see previous section). This example shows how you can use a value that is different from the title.



Using this technique you can make a group of radio buttons for western states.



The State field will not contain the fully spelled out state names, but only the 2 letter abbreviations (AZ, CA, NV, etc.).

Allow Multiple Values

It's possible to group together multiple buttons associated with the same field or variable. Normally clicking any button in the group erases the current value and replaces it with the new value (normal radio button behavior). However, if you have the Allow Multiple Values option turned on, clicking on the button will add the new value to the existing data in the field or variable, with the Value Separator in between.

As an example of the Allow Multiple Values option, consider pizza toppings. A pizza may have one, two, three or more toppings, or even none at all. Using the Allow Multiple Values option, you can create a series of checkboxes that will generate a list of toppings in a single field or variable. As the user clicks on each topping, it will be added to the end of the list. To remove a topping from the list, click on it again. See See "Multiple Value Button Groups" on page 835 for an example of this technique.

Value Separator

This is the text that will appear between each value in a multiple value list. Common separators include commas, spaces, slashes and hyphens. The separator may be up to 6 characters long, so you can use a multi character separator like comma-space. (However, if you want to process the value with Panorama's array functions you should use only a single character separator. See "Text Arrays" on page 1236 for more information on these functions.

"Radio" button

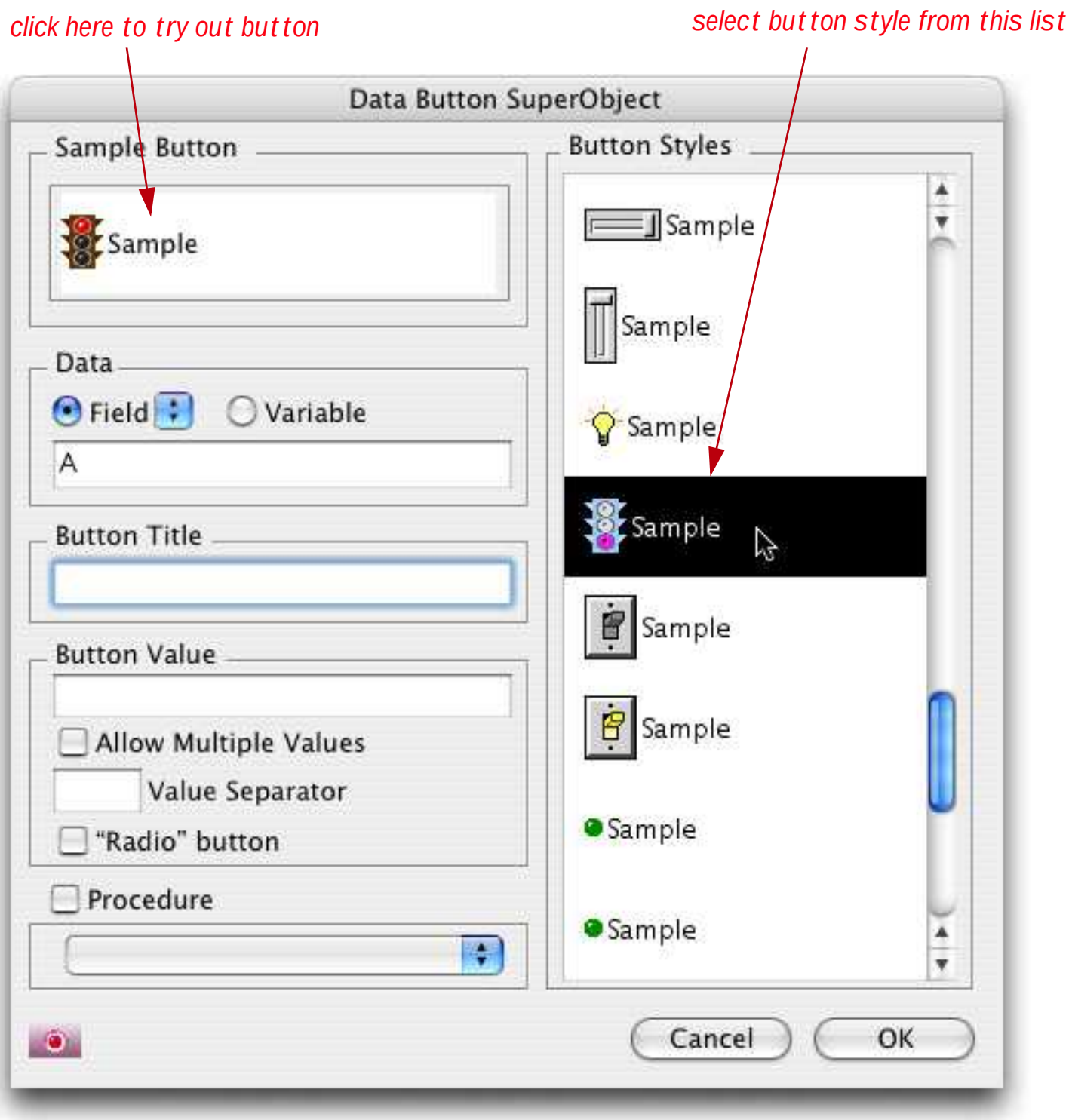
The data button normally toggles the value on and off each time you click on it. Turning on the "Radio" button option prevents the button from toggling off. In other words, you can turn the button on, but you can't turn it off again. Usually this option is only used for radio button combinations, where the value will be automatically turned off by clicking on another button in the group. This option prevents the user from turning off all the buttons in the group, so there is always at least one radio button checked.

Procedure

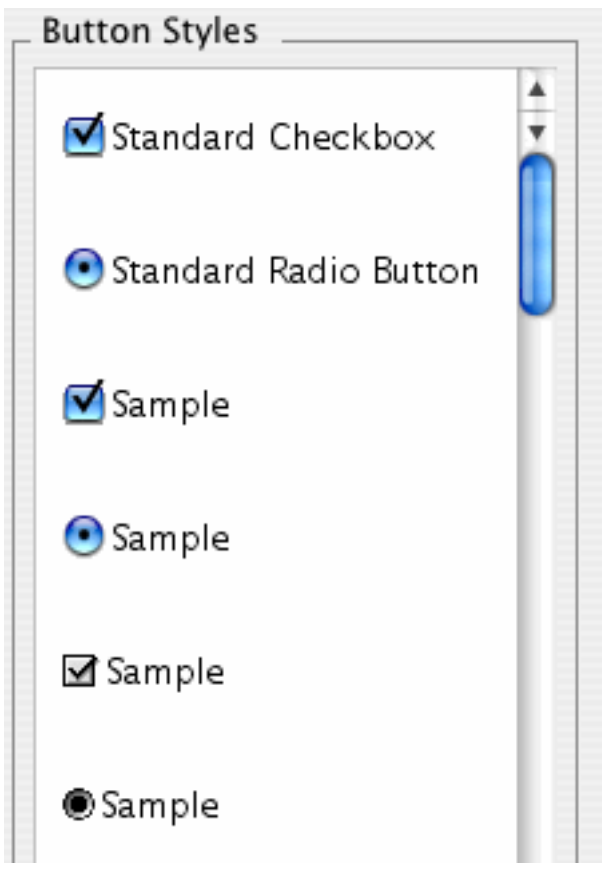
This section of the dialog specifies the procedure that will be triggered when this button is clicked (if any). The procedure can get information about what button was clicked by using the `info("trigger")` function. Even if you don't specify a procedure here, clicking on the button will trigger any automatic formulas and procedures associated with the field for the button (see "Automatically Triggering a Procedure" on page 350).

Sample

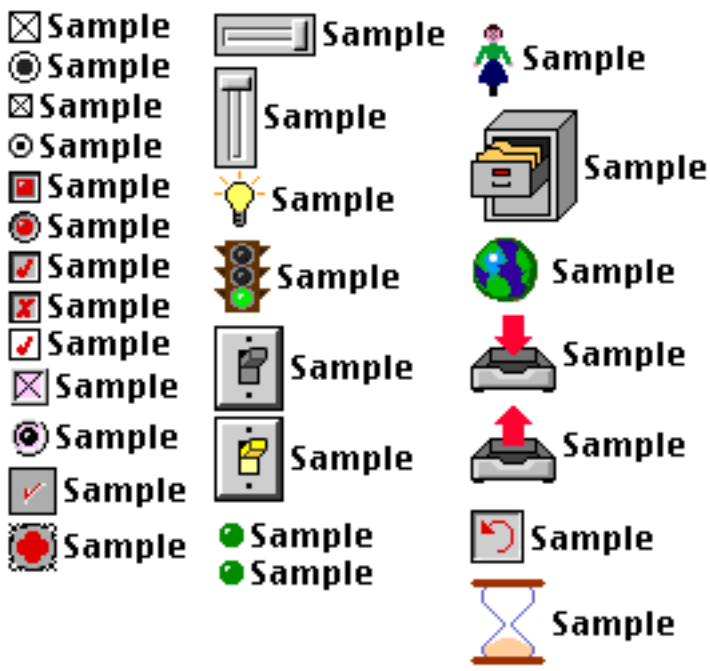
This section of the dialog shows a sample of the button. You can select the button style from the scrolling list on the right. Click on the sample button on the left to see what the button style will look like in both the on and off positions.



The first two styles, Standard Checkbox and Standard Radio Button, are special. Data buttons created with these styles will adjust automatically depending on what operating system is being used. In other words, the same button will change appearance depending on whether the database is being used on Mac OS X, Mac OS 9, or Windows.



With all of the other options what you see is what you get. Here are some of the other built-in choices available. (If you need to create your own custom button, use the Flash Art Data Button, see “Flash Art Data Button SuperObjects™” on page 842).



Note: Button operation is not affected by the button’s appearance. You can create a button that “looks” like a radio button but acts like a checkbox, or visa versa.

Flash Art Data Button SuperObjects™

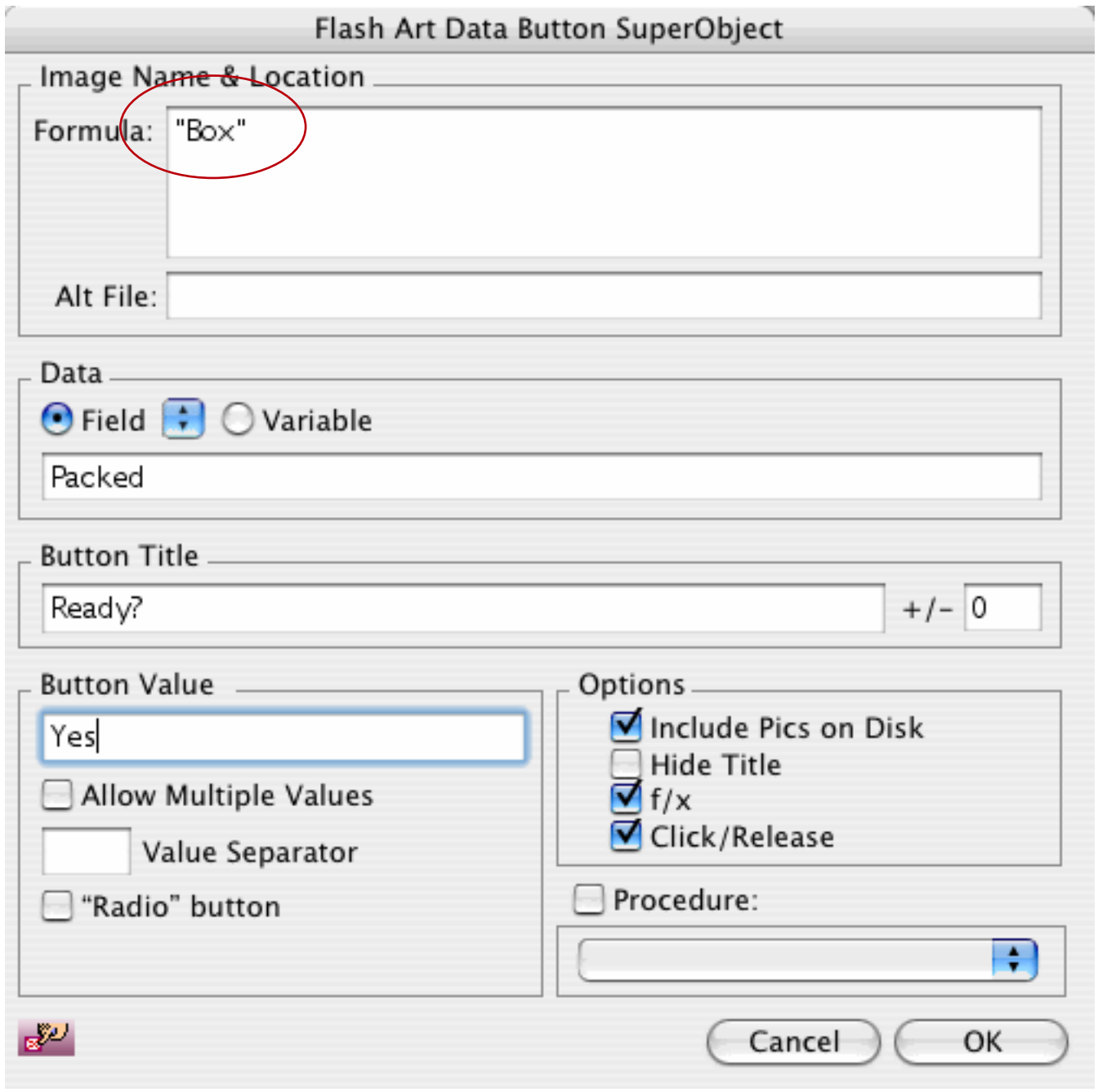
Flash Art Data Buttons are identical in operation to regular SuperObject Data Buttons. However, instead of picking the button’s appearance from a list of predefined styles, you create the artwork for the button yourself using Flash Art™. Before you attempt to use a Flash Art Data Button, you should be familiar with Flash Art creation and usage (see “Flash Art™” on page 748).

The first step in creating a Flash Art Data Button is to create two Flash Art pictures: the first showing the button in its “off” state and the second in its “on” state. When the button is clicked, Panorama will automatically switch between these two pictures as the button is toggled on and off. (The two pictures should have exactly the same dimensions or you will notice a shift as the mouse is pressed on the button.) The second picture must have the same name as the first picture, but with .DOWN added to the end. For example, if the “off” picture is called Box, the “on” picture must be called Box.DOWN.

If you wish, you may optionally create two additional pictures. These pictures will be displayed temporarily when the mouse is actually pressed on the button, allowing the button to “highlight” as it is being clicked. Two pictures are needed: one to highlight the “off” state and one to highlight the “on” state. These pictures must have the same name as the original two pictures but with • added after the name (press **Option-8** to create the • symbol on the Macintosh, **Alt-0149** on the PC). In our example these pictures would be named Box• and Box•.DOWN. Here are four images that have been prepared for a Flash Art Data Button. They show a box in both open and shut positions. (The images are shown enlarged to 4X in Adobe Photoshop).



Once the pictures have been created, you are ready to create the button itself. Select the **Flash Art Data Button** tool and drag across the form to create the button. (If the tool is not currently installed, use the Tool Palette dialog to install it, see “Customizing the Tool Palette” on page 497.) Type in the name of the base picture for this button (enclosed by quotes) into the Formula box. In our example, you would type "Box" into the Formula box.



The rest of the options in the Flash Art Data Button dialog are the same as the options in the Flash Art Push Button and Data Button dialogs. See “Super Data Button Options” on page 839 for descriptions of these options.

The finished button looks like an open box.



When you click on the button it highlights.



Releasing the mouse causes the button to switch to a closed box.



Click again to highlight the closed box.



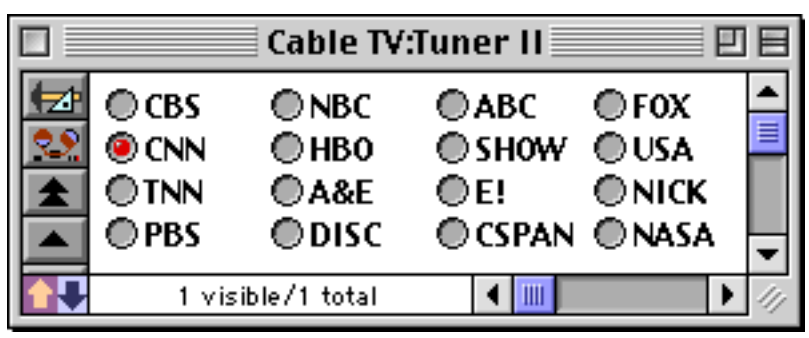
And release to switch back to an open box.



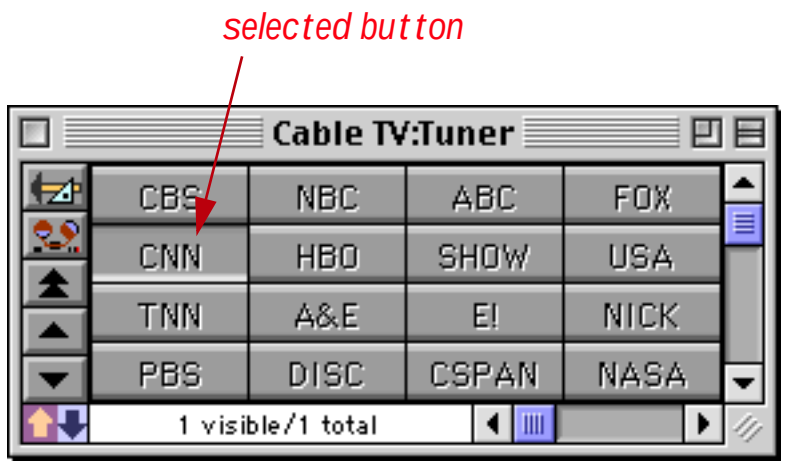
This is just one example. You can use this tool to make any size and shape of data button you like.

Sticky Push Button SuperObjects™

Sticky Push Buttons look like Push Buttons (see “Super Object Push Button” on page 813), but they operate like Data Buttons (see “Data Button SuperObjects™” on page 828). Instead of popping back up when you release the mouse, a Sticky Push Button stays pushed in like a checkbox or radio button. Like Data Buttons, Sticky Push Buttons are associated with a field or variable, and can be used separately or in groups (radio sticky push buttons, anyone?) To illustrate, here is a collection of options created as regular data buttons.

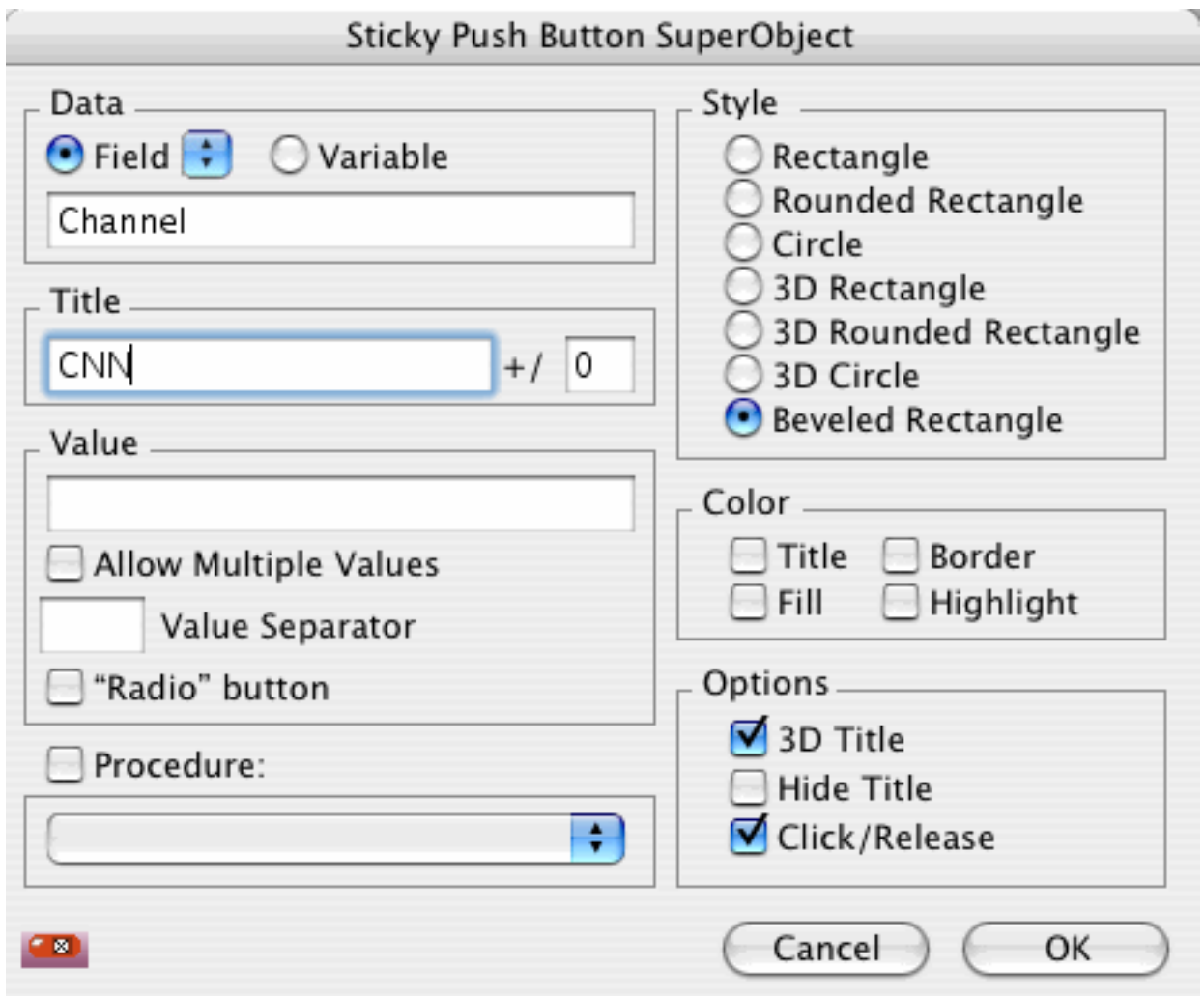


Here is the exact same example, but created with Sticky Push Buttons. Notice that CNN is also selected in this group of buttons.



These two examples operate exactly the same — the only difference is their appearance.

The Sticky Push Button configuration dialog is a combination of the options for Push Buttons and Data Buttons.

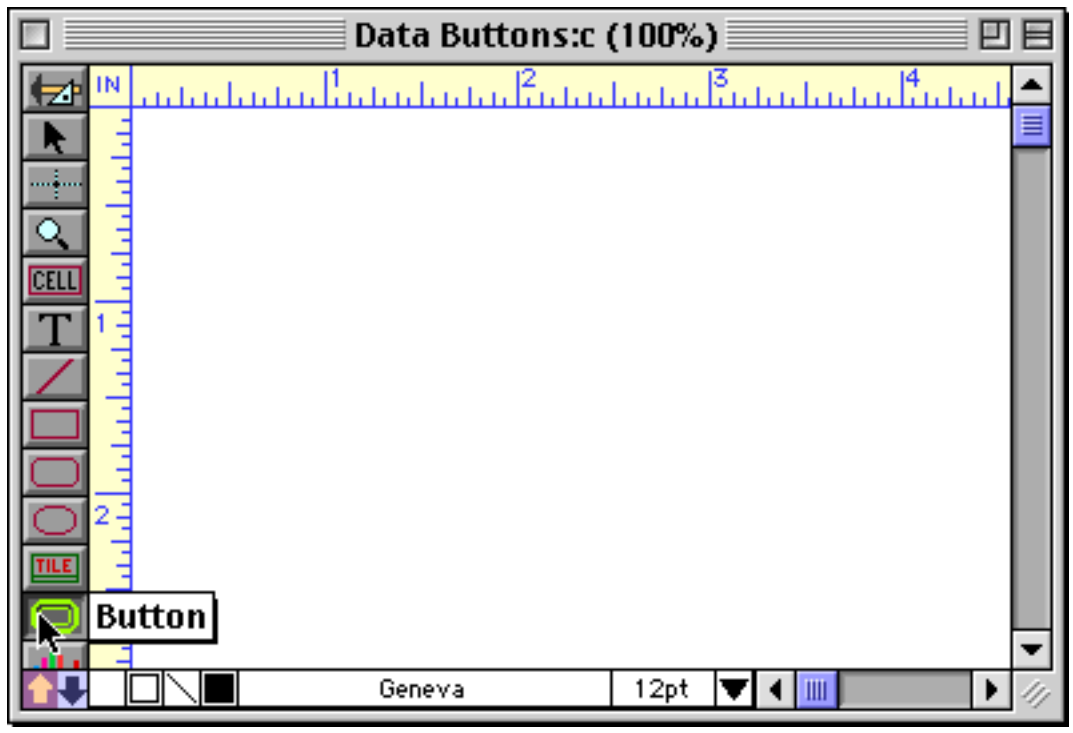


See "Super Object Push Button" on page 813 and "Data Button SuperObjects™" on page 828 for the details of these options.

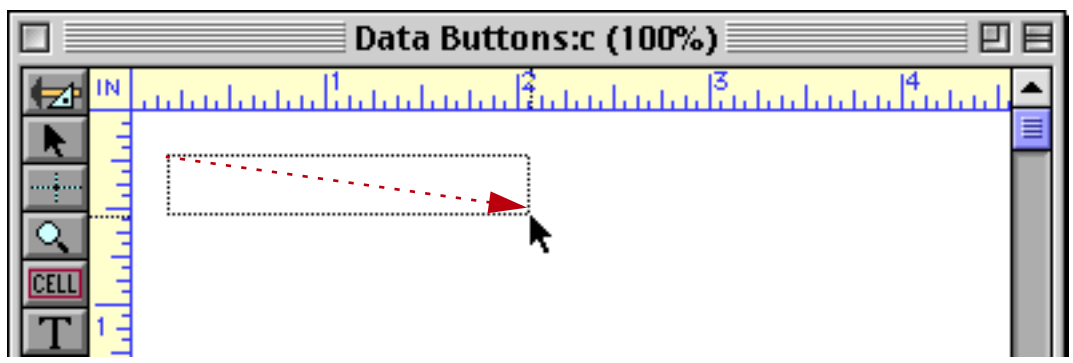
"Classic" Checkbox and Radio Buttons

In addition to the SuperObject data buttons described previously Panorama also has a "classic" Button object. When SuperObject buttons were added as part of Panorama 3.0, "classic" button objects were retained for compatibility with older databases. We recommend that you use SuperObject buttons for new applications. SuperObject buttons have many more style options, and can also work with variables as well as fields.

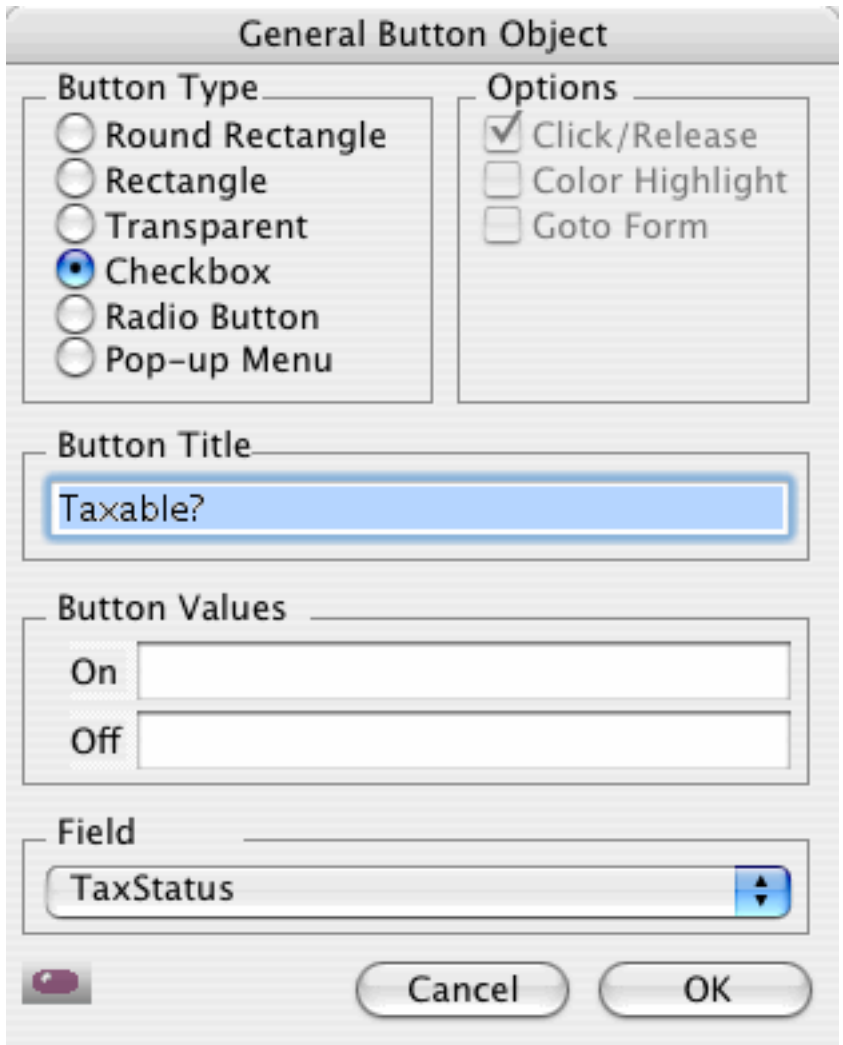
To create a checkbox or radio button, use the **Button** tool, which is part of the standard tool palette.



Drag the mouse across the form to specify the size and location of the button.



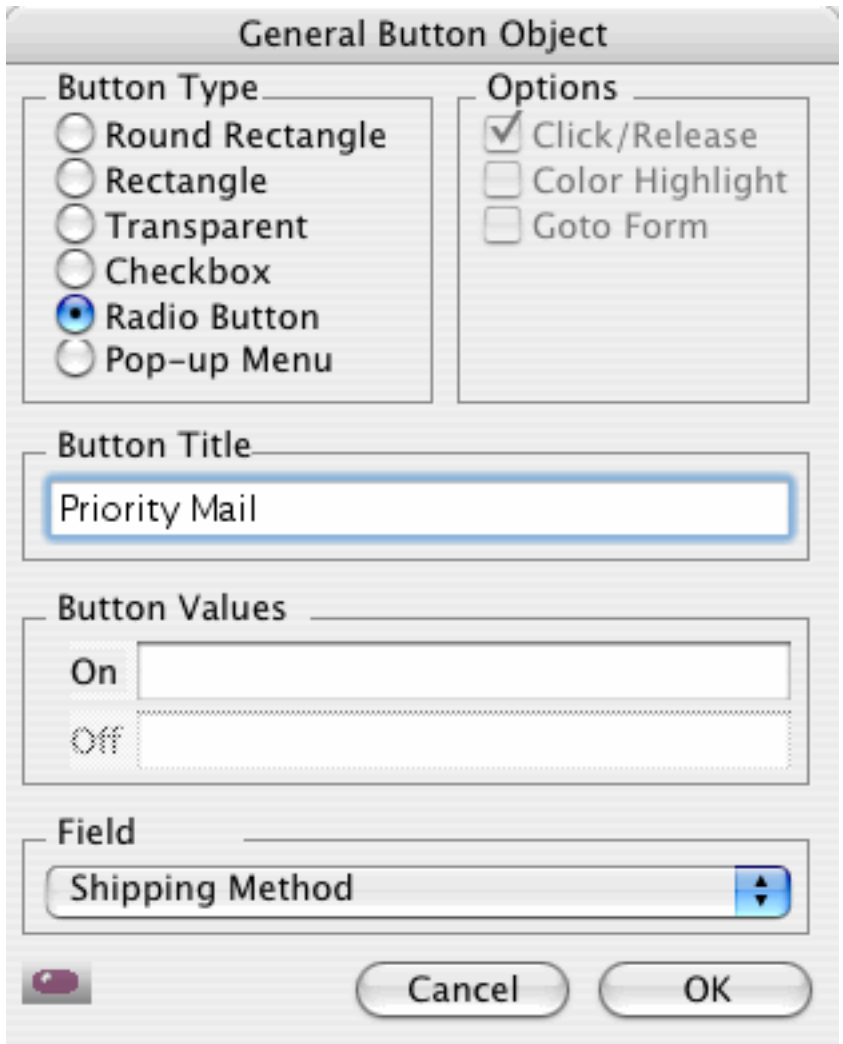
When you release the mouse, the Button Dialog appears. This dialog allows you to select the type of button you want to create.



Click on the type of button you want to create (checkbox or radio button) and select the field from the Field pop-up menu. Next type in the button title. This is the text that will be displayed as part of the button.

Use a checkbox button when a field has only two possible values (especially if one of these values is an empty cell). You can specify both on and off values. If you leave the on value blank Panorama will use the button title as the on value.

Use radio buttons when a field has three or more possible states. Each radio button has an on value. If you leave the on value blank, the button title will be used as the on value.



If you need to change a checkbox or radio button, click on it with the **Button** tool to make the Button configuration dialog re-appear. You can also double click on the button with the **Pointer** tool selected.

An easy way to create a group of radio buttons is to create the first button and then make copies of the button. See “Creating a Group of Radio Buttons” on page 831 for an example of this technique.

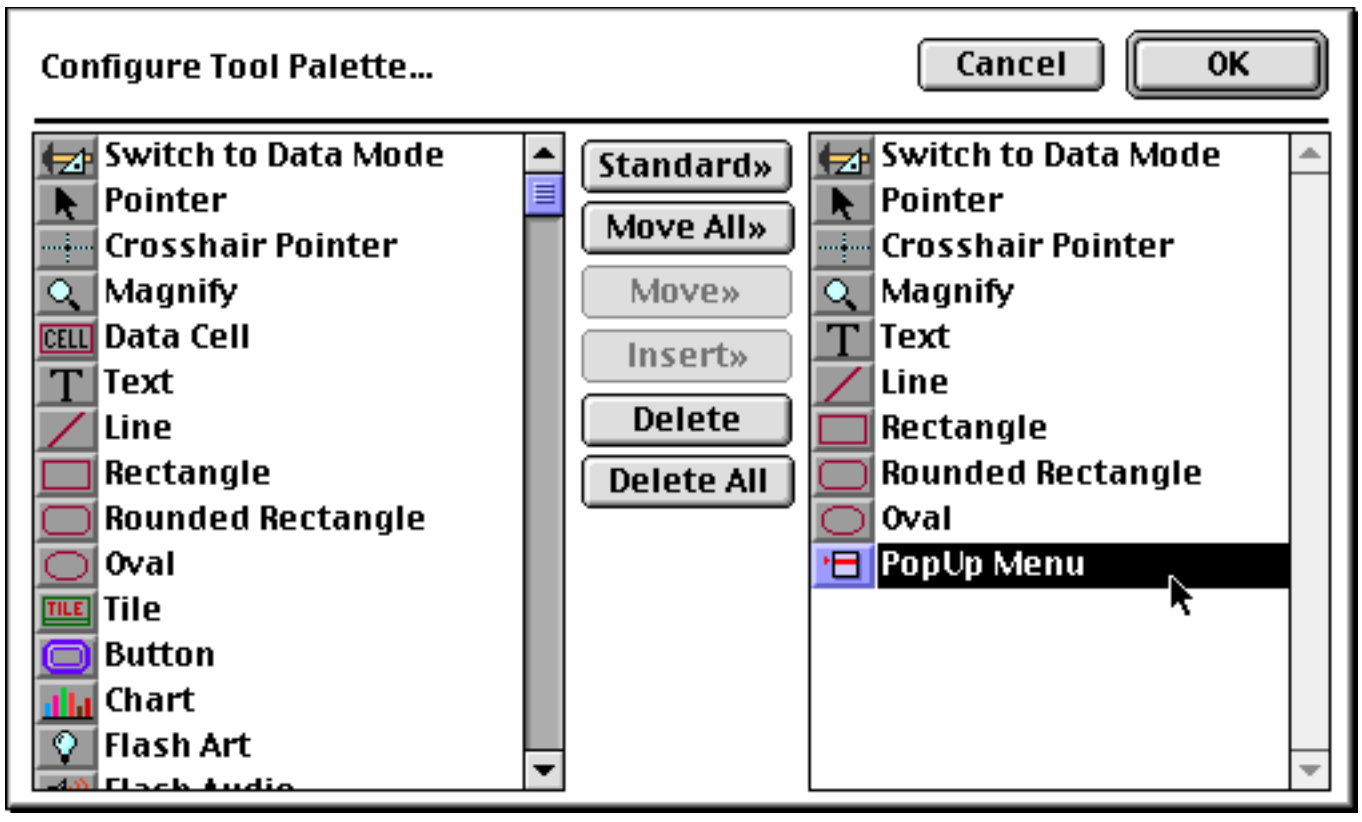
Pop-Up Menus

Radio buttons work well when there are only a few options. When you get past a dozen or so options, you'll probably want to use a pop-up menu or scrolling list (see "List SuperObjects" on page 869) instead. Panorama has three methods for creating pop-up menus: 1) Pop-Up Menu SuperObjects, 2) "Classic" buttons, and 3) Procedures (programming).

Pop-Up Menu SuperObjects™

The easiest and most flexible way to create a pop-up menu is with the Pop-Up Menu SuperObject™. A Pop-Up Menu SuperObject™ may be associated with any field or global variable. When the user makes a selection from the pop-up menu, the corresponding field or variable is automatically updated with the new value. Because the list of menu choices is calculated with a formula, the pop-up menu can change on the fly if necessary. You can also choose the menu font, color, and style (multi-column or scrolling).

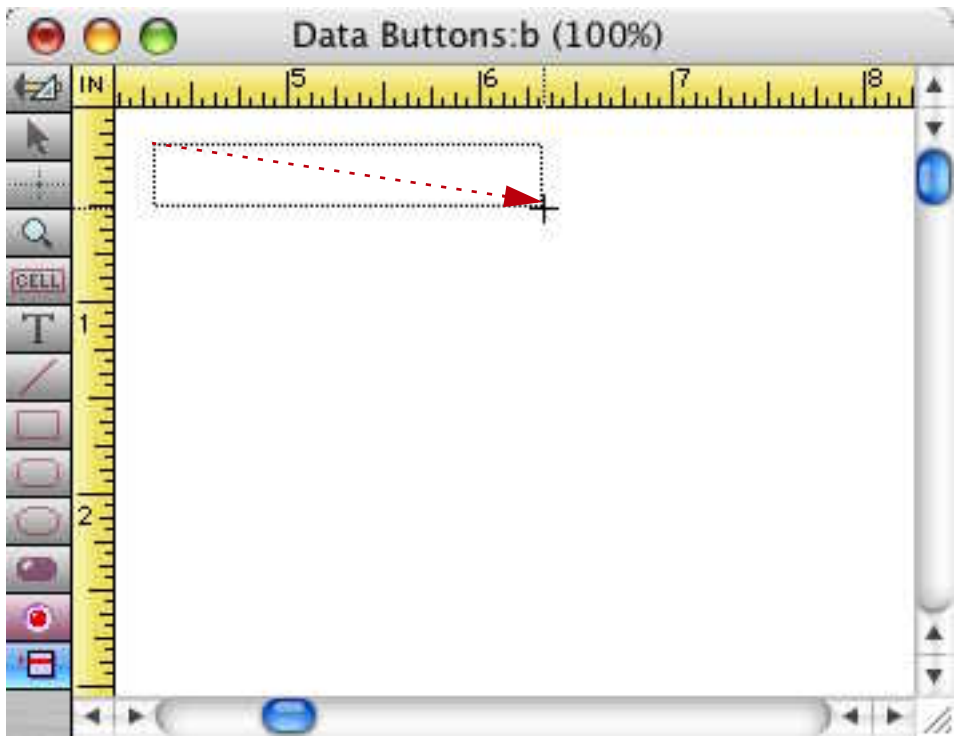
The Pop-Up Menu SuperObject tool is not in the default tool palette, so you'll need to move the use the Tool Palette dialog to add this tool to the palette if it is not already there (see "Customizing the Tool Palette" on page 497).



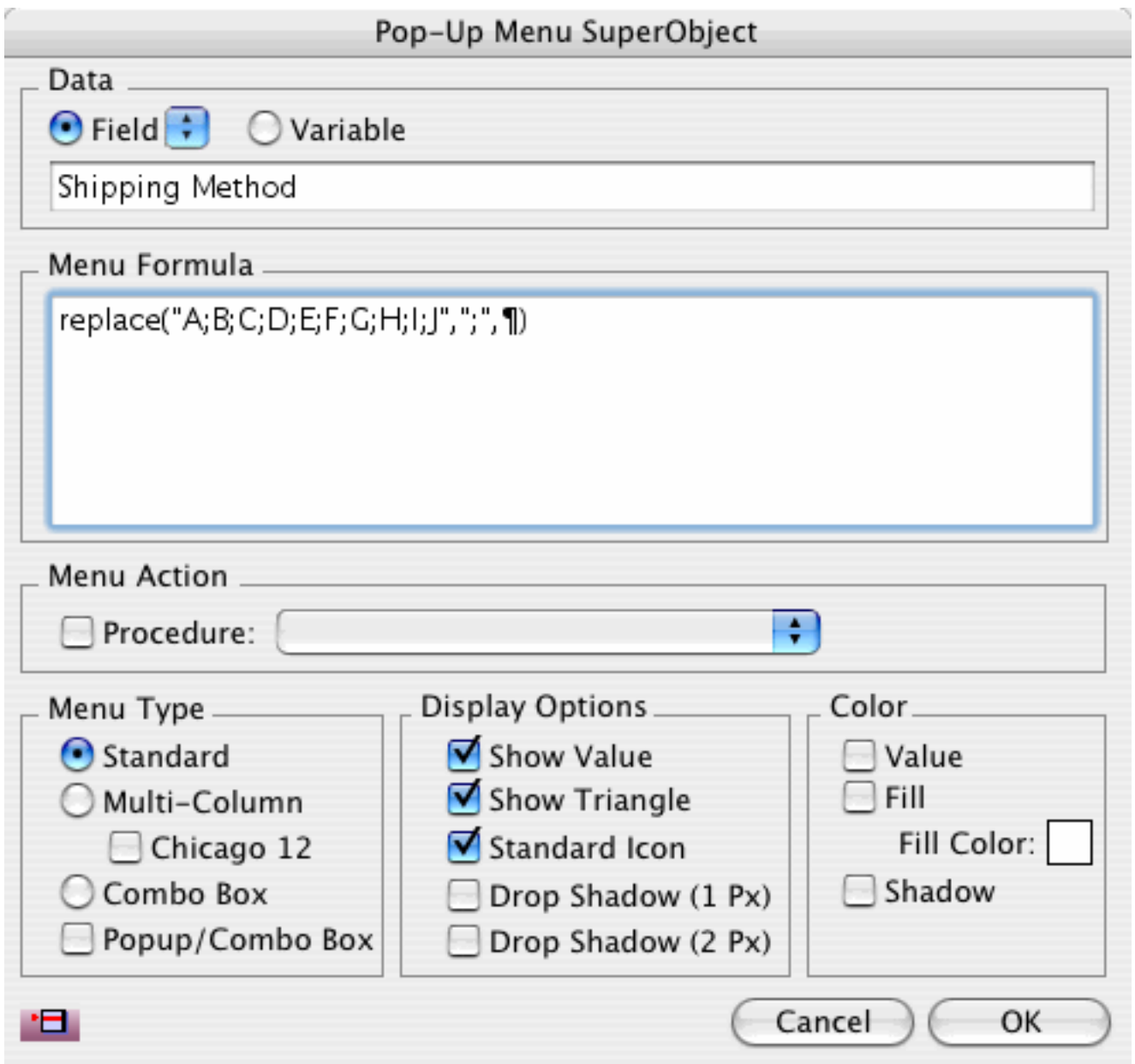
Now that the tool is added to the palette you can select it.



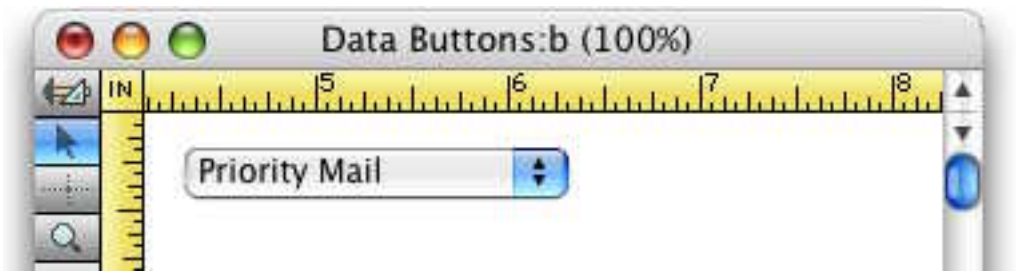
Once the tool is selected, drag the mouse across the form in the location where you want to create the pop-up menu object.



When you release the mouse, the Super Pop-Up Menu configuration dialog will appear.



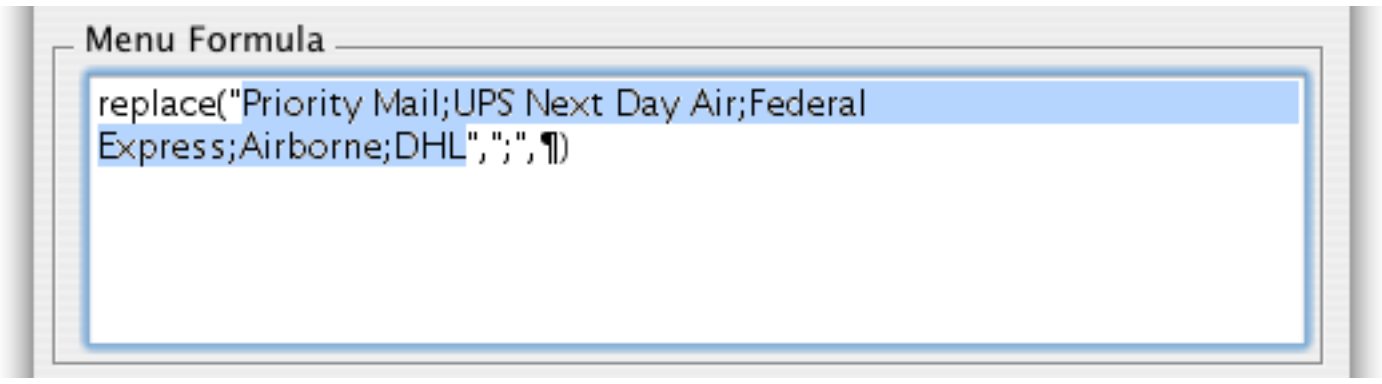
At a minimum you must select a field or variable for the pop-up menu. This field or variable will hold the result of the pop-up selection. Press **OK** to create the pop-up menu object.



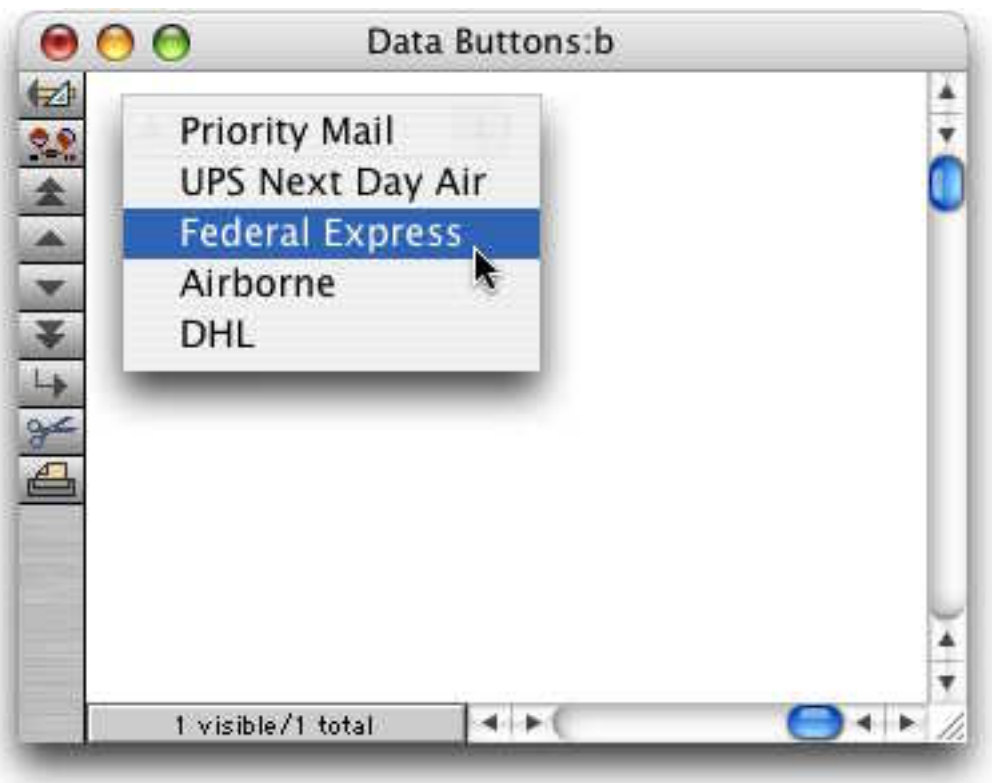
Switch to Data Access Mode to try out the pop-up menu.



The default pop-up menu contains ten items, A thru J. To change the items in the menu, switch back to Graphics Mode, select the **Pointer** tool and double click on the pop-up object. In the formula, replace A;B;C;D;E;F;G;H;I;J with the actual items you want to appear in the menu, with each item separated by a semicolon.



Press **OK** and switch back to Data Access Mode to try out the revised pop-up menu.



The Pop-Up Menu Formula

The menu formula calculates the list of menu choices. Each menu choice is on a separate line, separated by a carriage return.

When the user presses on the pop-up menu button, Panorama takes the formula, calculates it, splits the result into individual menu items (one per line), then displays the pop-up menu and allows the user to make a selection. All this happens in the blink of an eye as the user clicks on the button.

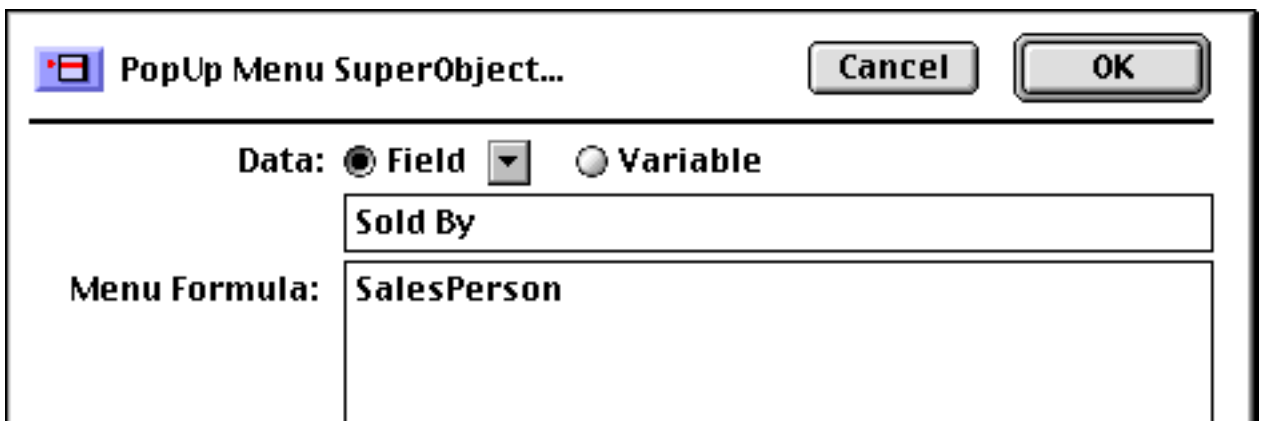
So much for theory, now let's take a look at some real world menu formulas. Suppose you want to create a pop-up menu with three choices: Gold, Silver and Bronze. Keeping in mind that the ¶ symbol represents a carriage return, the most basic formula that can be used to create this menu would be:

```
"Gold"+¶+"Silver"+¶+"Bronze"
```

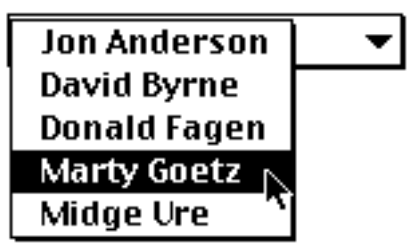
To type the ¶ symbol, press **Option-7** on the Macintosh and **Alt-0182** on the PC. If your menu has a lot of items it can be kind of a pain to type in this symbol over and over again. We can use Panorama's `replace(` function to make this formula easier to type (see "REPLACE(" on page 5665). In the formula below, the `replace(` function will change the semicolons into carriage returns. This allows you to type the menu choices with just a semicolon in between them. This option is so useful that Panorama preloads this function into the menu formula.

```
replace("Gold;Silver;Bronze",";",¶)
```

The real power of the menu formula is unleashed when you use a variable or field in the formula. Since the variable or field may be changed at any time with a procedure (or even with standard data entry), the pop-up menu can change at any time. For example, suppose you want to create a pop-up menu of salespeople in your company. Simply create a **permanent variable** (see "Long Life Variables" on page 1372) named SalesPeople and fill it with the name of each salesperson in your company on a separate line. You can create a preferences form that allows you to edit the list using a Text Editor SuperObject™ (see "Text Editor SuperObject" on page 639). When you create the pop-up menu, the menu formula will simply be `SalesPeople`.



Each time you click on the pop-up menu it will display the current list of salespeople.



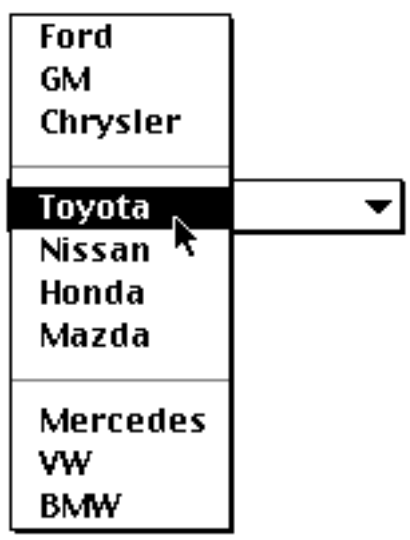
This list can be edited at any time simply by editing the permanent variable.

Dividing Lines in the Menu

To put a dividing line in a pop-up menu, simply create a line with the entry (-. The formula below produces a menu of car manufacturers in three sections: US, Japanese, and German.

```
replace("Ford;GM;Chrysler;(-;Toyota;Nissan;Honda;Mazda;(-;Mercedes;VW;BMW",";",¶)
```

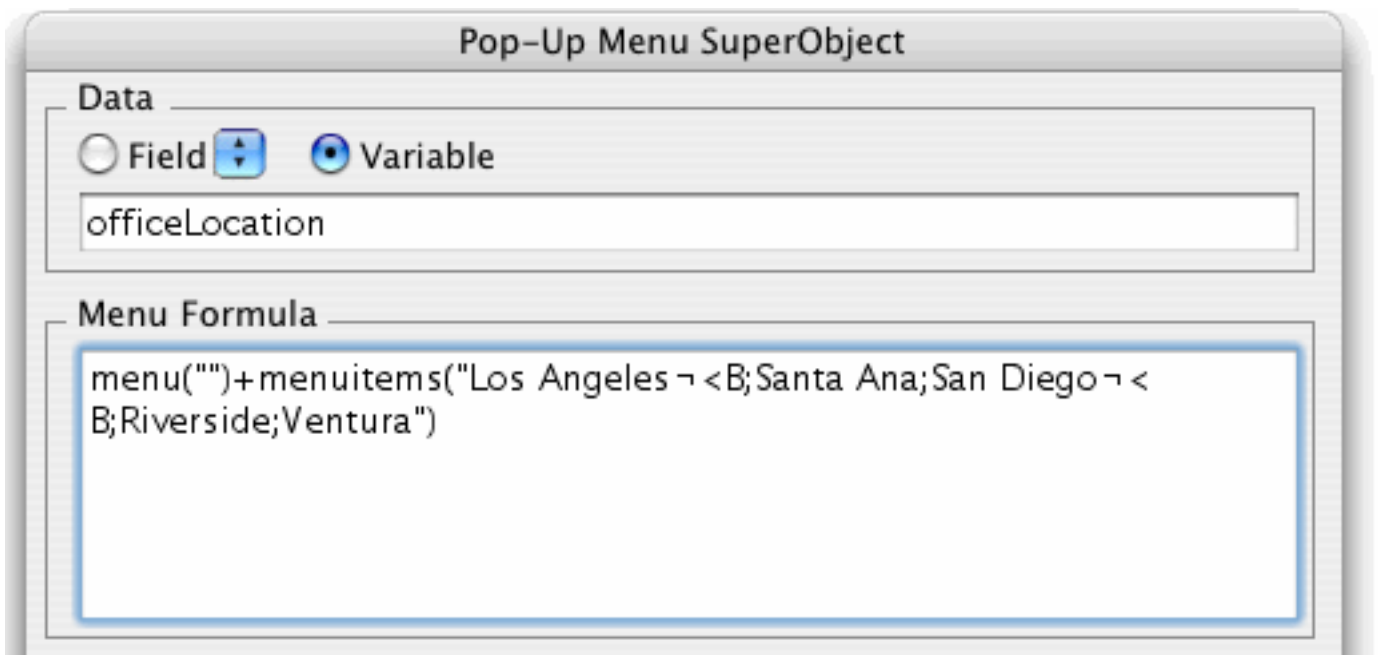
The dividing lines are not enabled in the menu, so the user cannot accidently choose a dividing line.



“Live” Pop-Up Menu Formulas

If the first character of the text generated by the Pop-Up Menu formula is (, Panorama will build the pop-up menu using the same rules used for “Live Menus” in the menu bar (see “Live Menus” on page 1465). This means that pop-up menus can include checked items, dimmed items, bold, italic, and even submenus!

If you are defining a normal pop-up menu with no submenus, the formula should start with the `menu(` function (see “The FileMenuBar Statement” on page 1465). Since this is a pop-up menu you can leave the name blank if you want to. A pop-up menu with the formula shown below could be used to select Southern California field offices, with the main offices in bold.



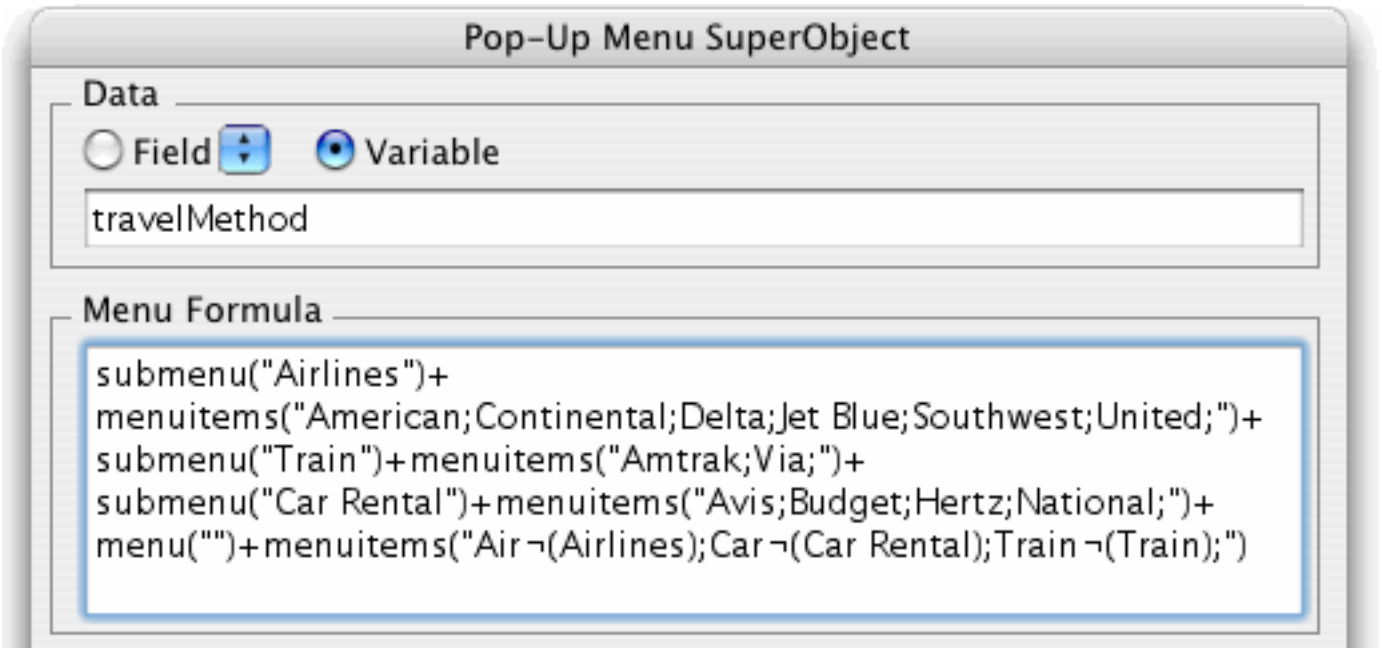
Here's what this pop-up menu will look like in use. The main offices, Los Angeles and San Diego, appear in bold.



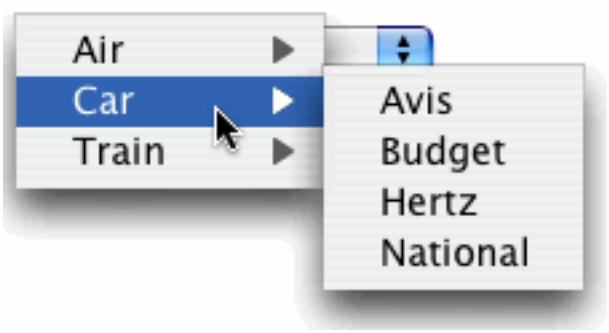
For more details on using the `menuitems(` function, see “Live Menus” on page 1465.

Pop Up Submenus

To create pop-up submenus the submenus must be defined first. The last menu defined is the actual main pop-up menu. This main menu doesn't need a name, it can simply be blank, but the submenus must have names. Here is an example of a formula for creating a pop-up menu with three submenus:



When you click on this pop-up menu button the main menu will spawn three submenus, like this:



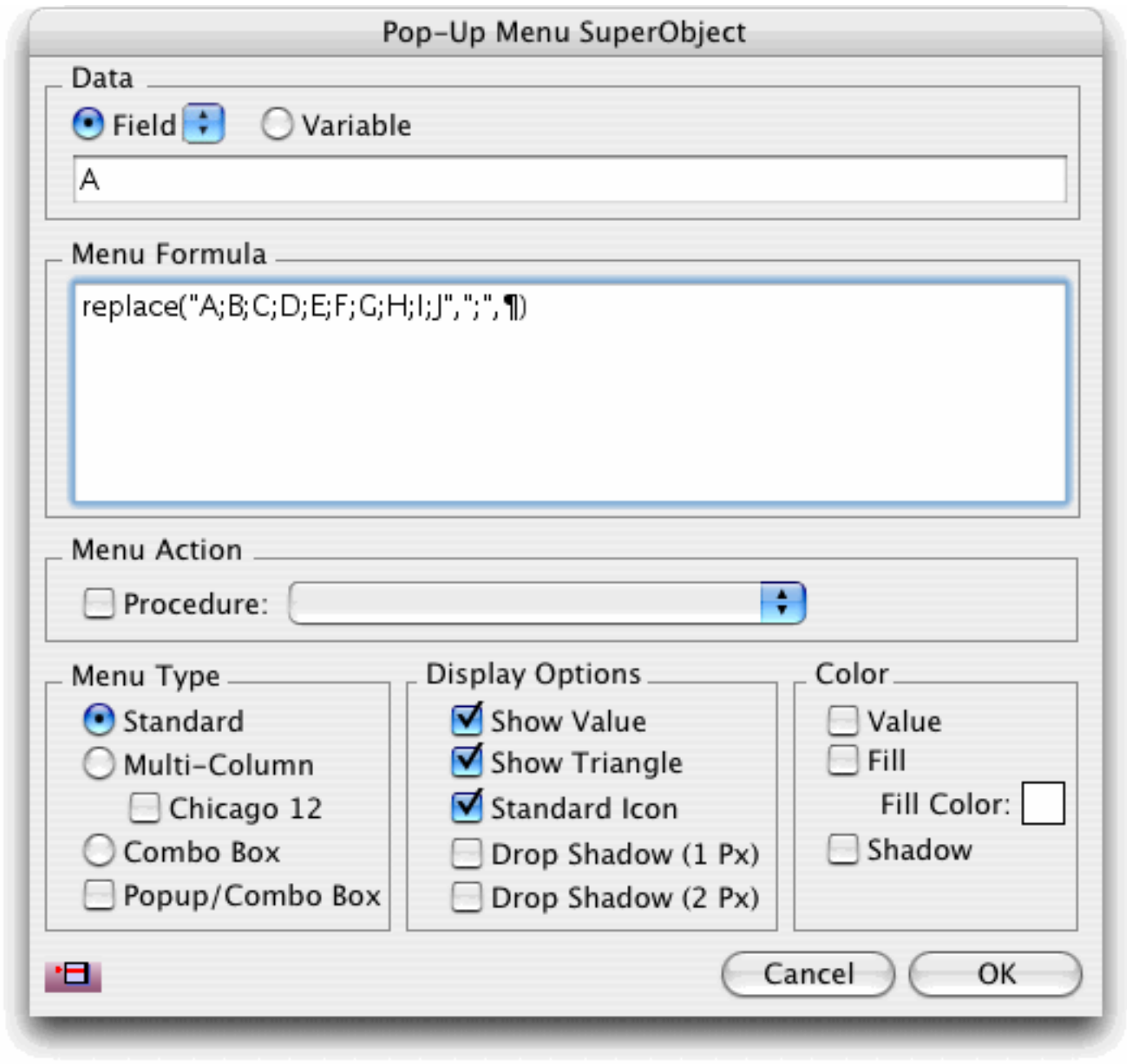
For more information on setting up live submenus see “Submenus (Heirarchical Menus)” on page 1473.

When an item in a submenu is selected, the pop-up SuperObject will simply be filled in with the menu item. For example, if **Jet Blue** is selected, the value in the field or variable will be "Jet Blue". There is no way to determine that this was selected from the **Airlines** menu.

Live menus can also be used with the PopUp statement (see “Creating a Pop-Up Menu with a Procedure” on page 864). When used with this statement the resulting trigger will include both the menu title and the menu name, for example "Airlines.Jet Blue". The `PopUpByNumber` statement is similar, the trigger value will be: "Airlines.4". See the **Programming Reference** wizard for detailed information on these statements.

Pop-Up Menu Options

The SuperObject™ Pop-Up menu dialog is divided into several sections.



Data

This section of the dialog specifies the field or variable associated with the pop-up menu. Type the name of the field or variable into the box (or select the field name from the pop-up menu next to the Field radio button). If the pop-up menu is associated with a variable that has not been created with a procedure, Panorama will automatically create a global variable with this name whenever the form containing this object appears. This global variable can be used in formulas and procedure just like any other global variable.

Menu Formula

This section specifies the choices listed in the actual pop-up menu. Instead of simply typing in the choices, you enter a formula that calculates the choices. (Since the formula can use a field or variable, this allows the menu to be changed easily on the fly.) The formula must calculate the list of choices, with each choice separated by a carriage return. (Remember, a carriage return can be represented by the ¶ symbol (Mac: **Option-7** / PC: **Alt-0182**) in a formula.) See “The Pop-Up Menu Formula” on page 853 for a detailed discussion of the menu formula.

Menu Type

The Pop-Up Menu SuperObject™ supports three menu styles: Standard, Multi-Column and Combo Box. Standard menus look and operate just like all of the other menus on your system. Multi-Column menus will automatically split the menu into an array of two or more columns. This allows dozens or even hundreds of options to appear on the screen at one time.

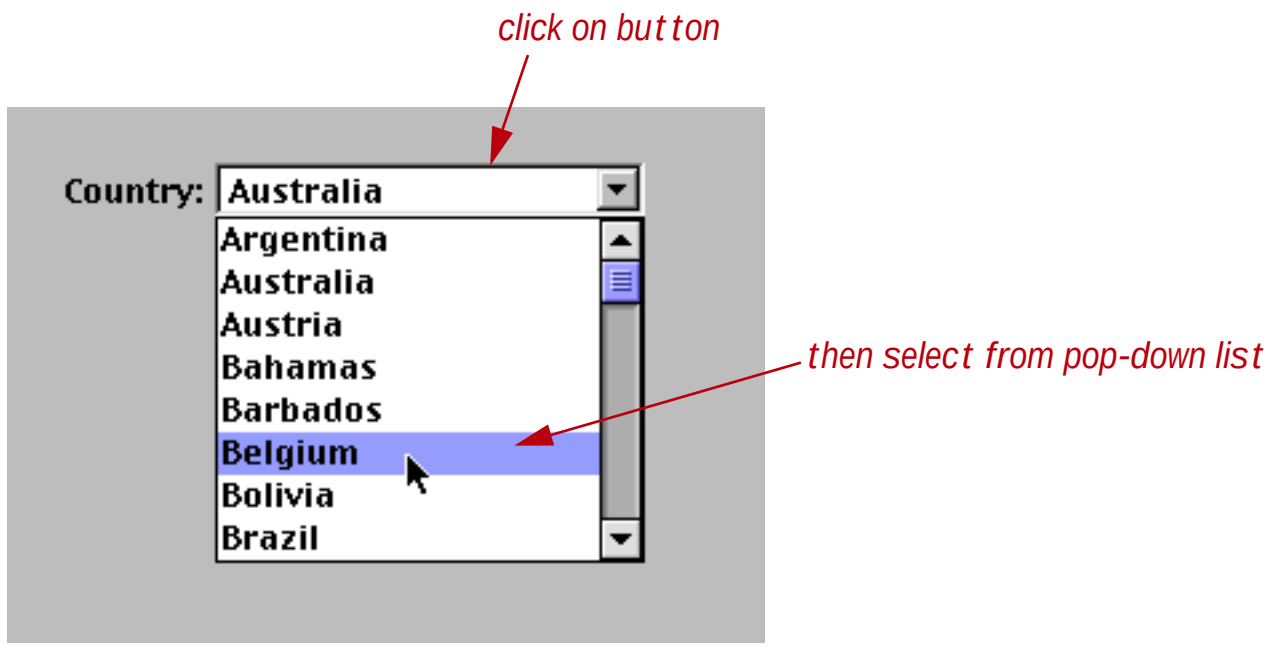


The Chicago 12 option forces the actual pop-up menu to always appear in the standard system font, no matter what font and size are selected for the pop-up object. (On early Macintosh systems this font was Chicago 12, hence the name of the option.) If you leave this option off, the menu can be displayed in any font and size. Just select the Pop-Up Menu SuperObject™ and choose the font and size the normal way. (Note: If you select the Scrolling option, this option is ignored and the menu will always be displayed in Chicago 12 point type. Only Multi-Column menus can be displayed in non-standard sizes.)

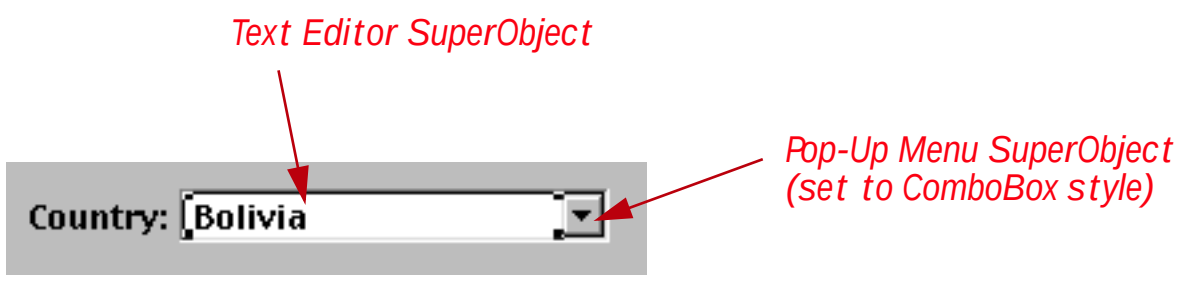
The ComboBox option makes the pop-up button look and operate like a Microsoft Windows style Combo Box. This type of button looks best on a gray or colored background.



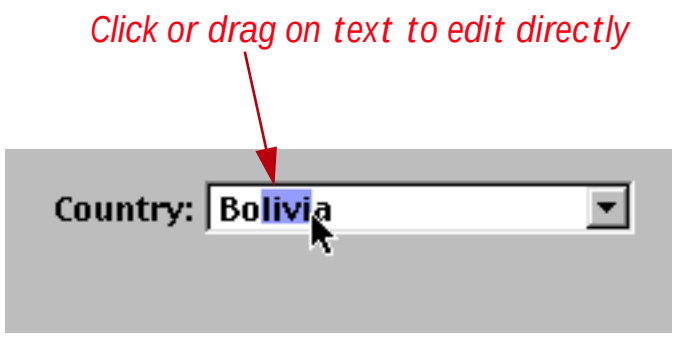
To use a Combo Box, simply click anywhere on the box to make a pop-down list appear, then make your choice from the list.

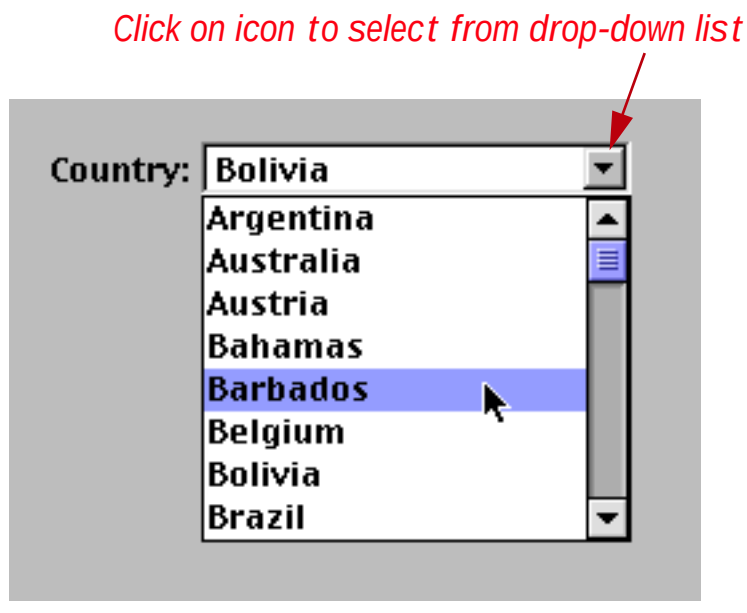


If you would like to be able to directly edit the value (the Country in this case) you can superimpose a Text Editor SuperObject (see "Text Editor SuperObject" on page 639) on top of the Combo Box.



Now you have a choice of editing methods. You can click or drag on the text to edit it directly, or click on the Combo Box icon to pop-down the list of choices.



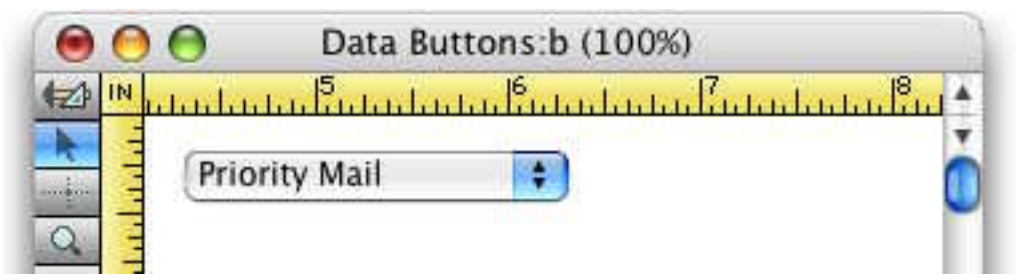


The final menu type is Mac Pop-Up/Windows Combo Box. The appearance and operation of this type depends on the type of computer being used. On a Macintosh computer this button will look and operate like a standard pop-up menu. On a Windows computer this button will look and operate like a Combo Box.

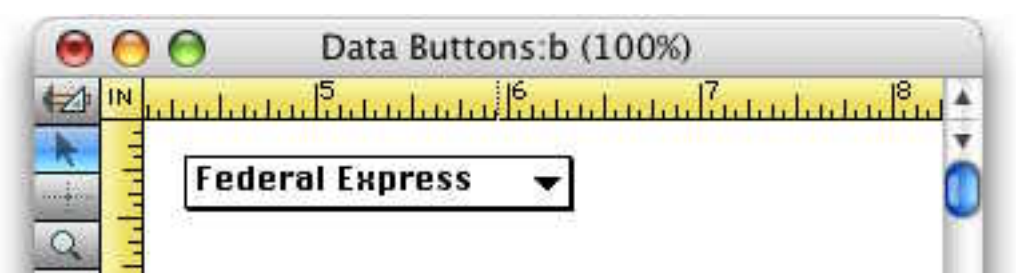
Display Options

This section controls how the pop-up menu object is displayed on the form. If the Show Value checkbox is turned on, the field or variable associated with the pop-up menu will be displayed in the object. If the Show Triangle option is on, a small downward pointing triangle will appear on the right side of the pop-up menu.

You can only pick one of the last three options (or none). The Standard Icon option is the default choice, and displays the standard pop-up menu graphics for the operating system being used. Here is the standard icon for Mac OS X.



If the Drop Shadow checkbox is on, Panorama will automatically draw a drop shadow around the edges of the pop-up menu object. You can select whether the drop shadow is 1 or 2 pixels deep.



If you turn off all three of these options the pop-up menu will be invisible. This can be handy if you want the pop-up menu to appear with a custom image.

Color

This section controls the color of various parts of the pop-up menu object on the form. If an option is checked, that part will appear in the color specified for the object, otherwise that part will appear in black. (To specify a color for an object, use the Graphics menu or the Graphic Control Strip (see “Color” on page 526). Note: The color options only apply to multi-column menus - they are ignored for Standard menus.)

Value: If this option is checked, the field or variable value will appear in color, otherwise it will appear in black. (This option is ignored if the Show Value option is not turned on.)

Fill: If this option is checked, the pop-up menu object will be filled with the object color. You should not check both this option and the Value option (this makes the value invisible).

Shadow: If this option is checked, the border and drop shadow will appear in color, otherwise it will appear in black. (This option is ignored if the Drop Shadow option is not turned on.)

Fill Color: This pop-up menu controls the background color of the pop-up menu itself. Usually, the background color is white, but you can switch it to any color you want. (Note: This option only works with Multi-Column menus. Scrolling menus always have a white background.)

Procedure

The pop-up menu can optionally trigger a procedure whenever the user makes a selection. This procedure can perform additional tasks that need to be done when a selection is made.

Since the pop-up menu updates a field or variable, the procedure can simply read the menu choice from that field or variable. In addition, the procedure can find out the name of the menu object using the `info("trigger")` function. The `info("trigger")` function will return the name of the pop-up menu object itself (if any). If you haven't assigned a name to this object, the default is custom (see "Object Type / Object Name" on page 533 to learn how to assign a name to an object). If you want to have several pop-up menus that trigger the same procedure, you should give each a unique name so that the procedure can tell which pop-up menu was used.

Pop-Up Menu Font, Size and Dimensions

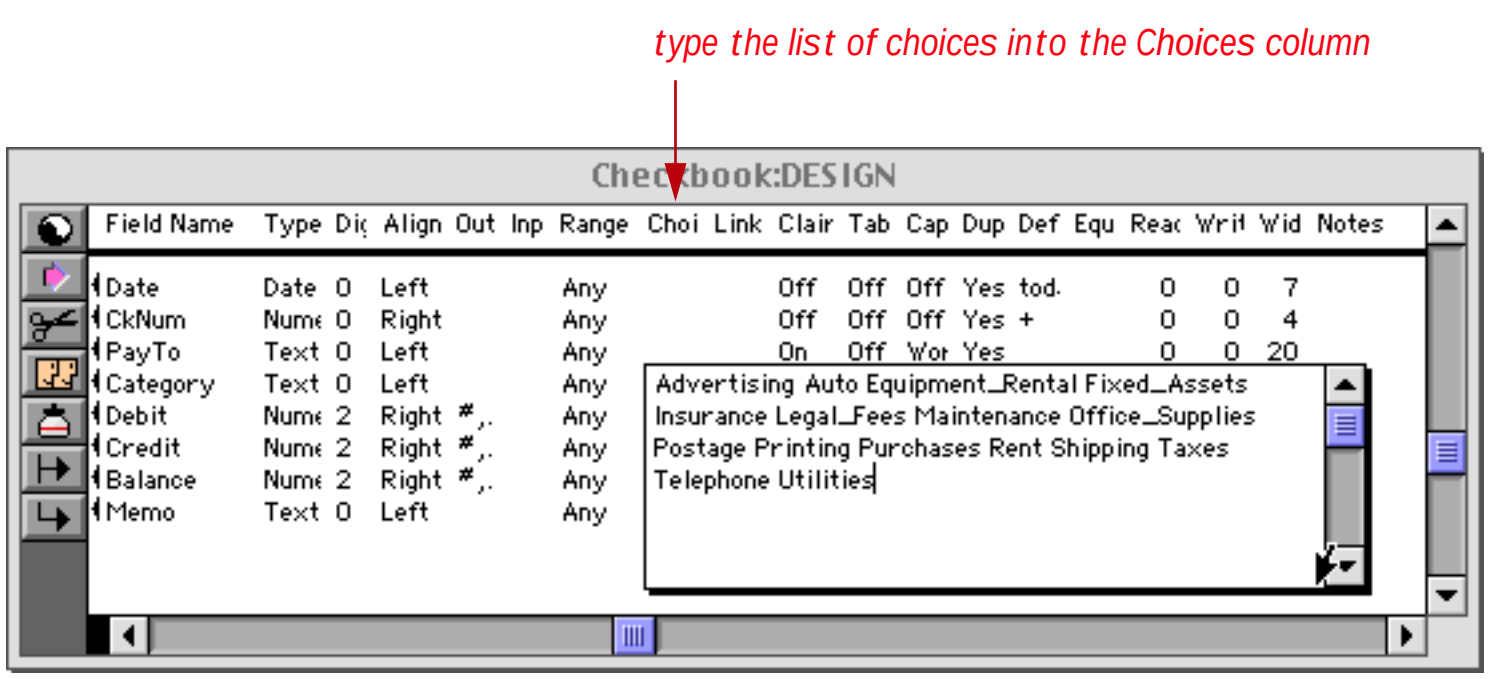
If you are using the Standard menu style Panorama will use the standard menu font and size for the operating system in use (for example 13 point Lucinda Grande on Mac OS X).

If you are using a Multi-Column menu you can use other fonts and point sizes. To get the most options in the least space, use Geneva 9 point (Alpine 9 point on a PC). For the best performance, you should stick to simple fonts and sizes that you actually have available in your system file (these sizes appear outlined in the Size menu.)

"Classic" Pop-Up Buttons

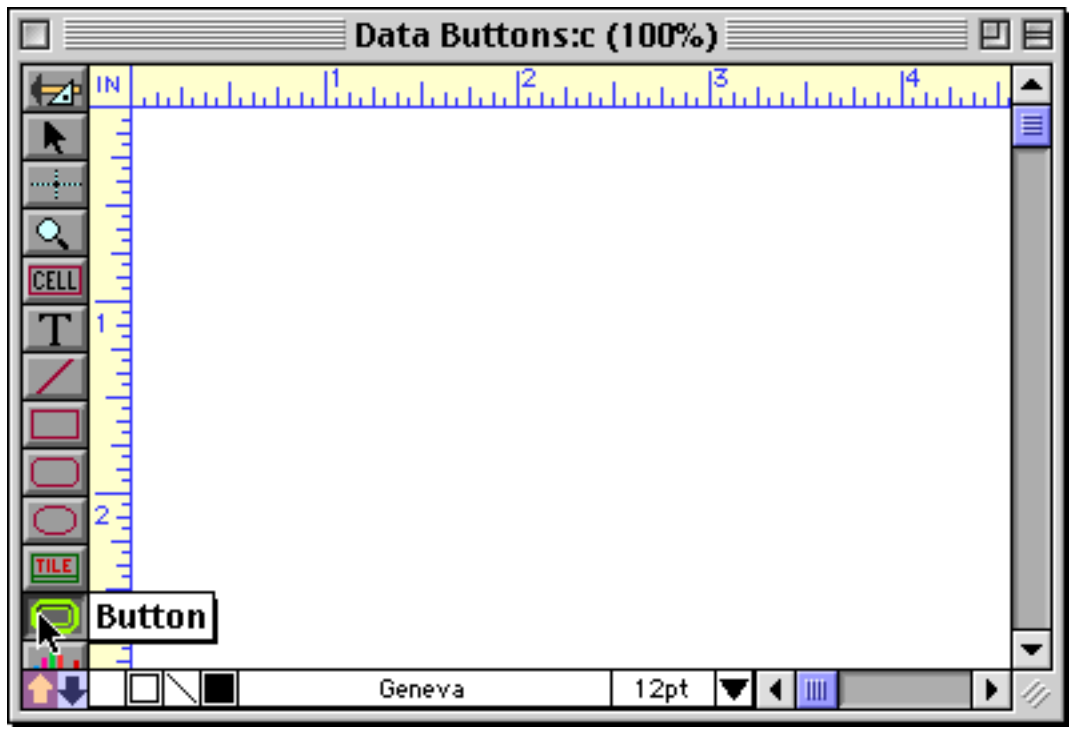
In addition to the SuperObject pop-up menu buttons described previously Panorama also has a "classic" Button object that can also create pop-up menus, although with very limited options. When SuperObject buttons were added as part of Panorama 3.0, "classic" button objects were retained for compatibility with older databases. We recommend that you use SuperObject buttons for new applications. SuperObject buttons have many more style options, and can also work with variables as well as fields.

The "classic" pop-up menu button uses the Choices field of the design sheet as the template for the menu. The items in the menu will be the same as the choices in the list.

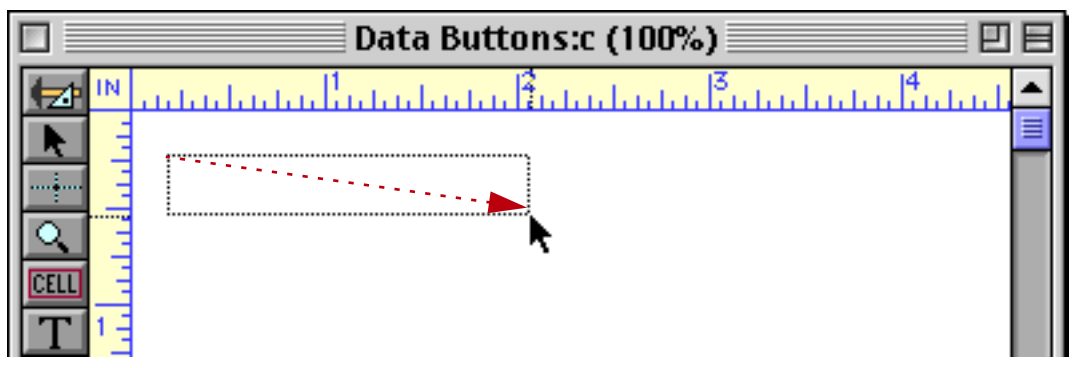


See "Choices" on page 298 to learn more about setting up a list of choices.

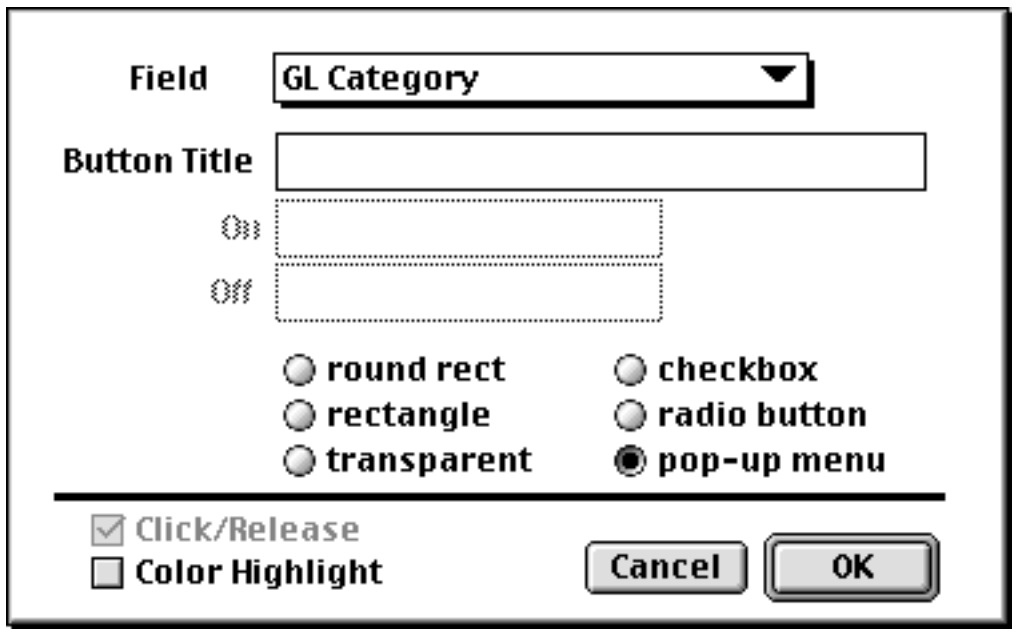
To create a classic pop-up menu button, use the **Button** tool, which is part of the standard tool palette.



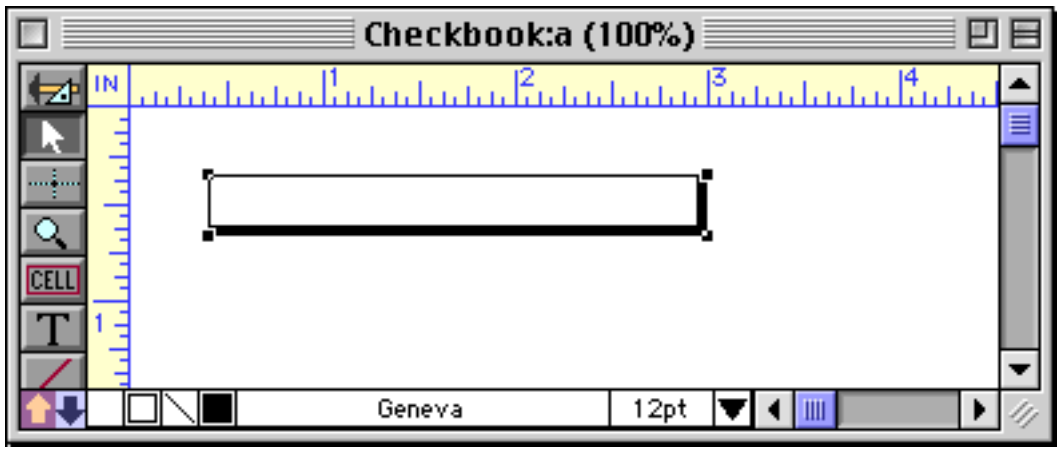
Drag the mouse across the form to specify the size and location of the button.



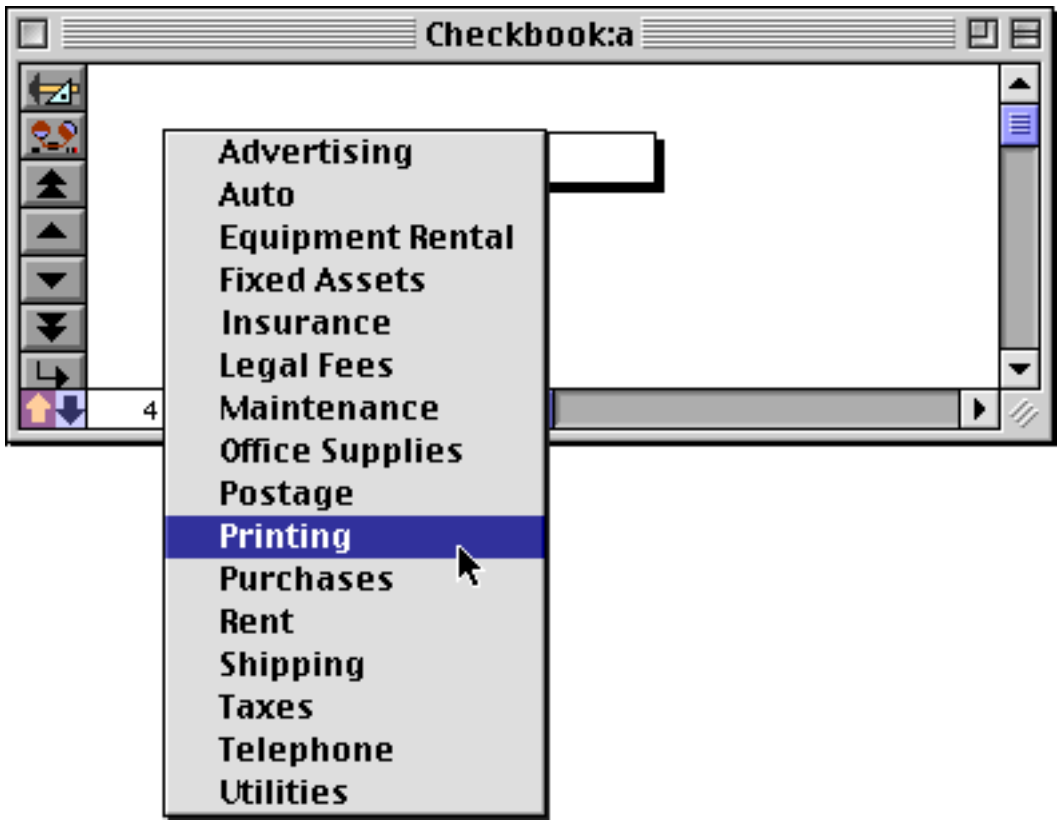
When you release the mouse, the Button Dialog appears. This dialog allows you to select the type of button you want to create.



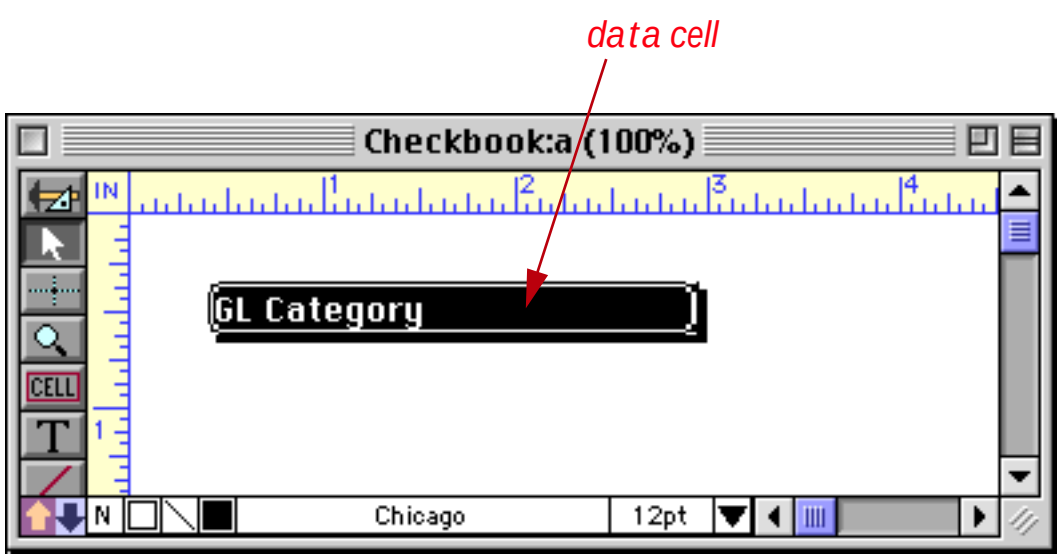
Click on the type of button you want to create (pop-up menu) and select the field from the Field pop-up menu. You can type in a button title, but it will be ignored. Press **OK** to create the button.



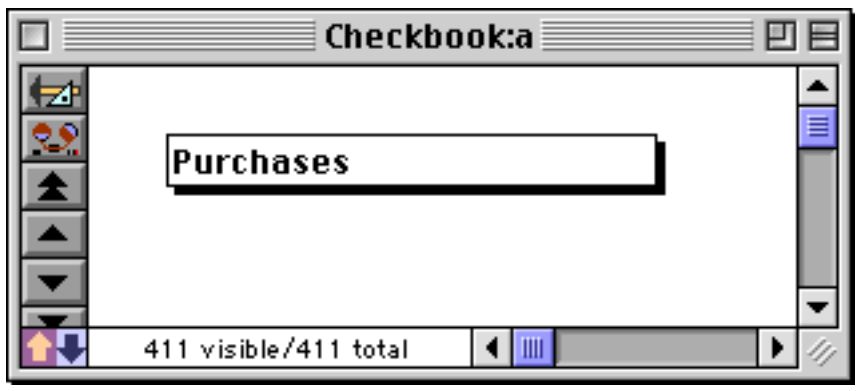
Switch to Data Access Mode to try out the pop-up menu.



Unlike SuperObject Pop-Up Menu buttons, the “classic” button does not display the contents of the field. If you want the field contents to appear you must add a Data Cell (see “Working with Data Cell Objects” on page 635), Auto-Wrap Text (see “Displaying Data in Auto-Wrap Text” on page 595), or Text Display SuperObject (see “Text Display SuperObjects™” on page 608) to the form. In this example we have added a Data Cell.



If the Data Cell remains on top of the button it will interfere with clicking on the button. Use the **Send to Back** command to move the Data Cell behind the button (see “Changing the Stacking Order” on page 569). Switching to Data Access Mode you can now see the field value displayed “inside” the pop-up menu.



If this all seems like a lot of extra tedious work, it is. To avoid all this extra work use the SuperObject Pop-Up Menu instead.

Creating a Pop-Up Menu with a Procedure

In very rare cases you may need even more flexibility than the Pop-Up Menu SuperObject™ provides. For even more control over your pop-up menu, you can create a pop-up menu with a standard button and with a procedure. In this procedure, you can control what happens both before and after the pop-up menu appears.

Here’s how it works. First, the user clicks on a standard transparent button (see “Transparent Push Buttons” on page 822). This button must have the Click / Release option turned off, so that the procedure is triggered immediately when the mouse is clicked. Now the procedure determines the location of the mouse and makes the pop-up menu appear at that location. The procedure pauses while the user makes a selection from the menu. Once the selection is made, the procedure takes the choice and stores it or processes it. These are the same steps taken by the Pop-Up Menu SuperObject™, but in this case each step must be programmed by you as part of the procedure.

Where Will the Pop-Up Menu Appear?

The first step the procedure must take is to figure out where the pop-up menu will appear on the screen. You must determine two numbers: 1) The distance (in pixels) from the top of the current window and, 2) The distance from the left edge of the current window. (These two numbers are known as the local co-ordinates, because they are relative to the current (local) window. Global co-ordinates are relative to the entire screen.)

The usual choice is to make the pop-up menu appear in the same spot as the button that was pressed. You can find out the local co-ordinates of this button using the `GetLocalButton` statement. You must create four local variables to hold the four dimensions of the button (top, left, height, width).

```
local PopTop,PopLeft,ButtonHeight,ButtonWidth
GetLocalButton PopTop,PopLeft,ButtonHeight,ButtonWidth
```

For the purposes of a pop-up menu, you’re only interested in the first two values, PopTop and PopLeft.

Another option is to make the pop-up menu appear right over the current mouse location. This is especially useful if you have created a large button where the current mouse location may be quite far from the upper left corner of the button. To find out the current mouse location in local co-ordinates, use the GetLocalClick statement.

```
local PopTop,PopLeft
GetLocalClick PopTop,PopLeft
```

(Note: The names of the local variables are not important. You can use any names you want as long as you are consistent within the procedure.)

The PopUp Statement

Once the procedure has calculated where the pop-up menu should appear, it should use the PopUp statement to actually make the menu appear and allow the user to make a choice from the menu. This statement has five parameters:

```
PopUp MenuFormula, V, H, CurrentValue, NewValue
```

The first parameter, MenuFormula, is a formula that specifies the menu items that should appear in the pop-up menu. This formula uses exactly the same rules that the Pop-Up Menu SuperObject menu formula uses, so you should refer to that section (see “Menu Formula” on page 857) for details on setting up the menu formula.

Experts only: The MenuFormula can also be used to specify that a custom menu you have created with ResEdit should be used for the pop-up menu. In this case the menu formula should result in a single number which corresponds to the resource number for the custom menu. Using this technique, you can create a pop-up menu with icons, or a pop-up menu that uses a non-standard menu proc (perhaps a non-text menu). (You can also use a Live Menu Formula, see below.)

The next two parameters, V and H, specify the location where the pop-up menu should appear. Use the values you calculated with the GetLocalButton or GetLocalClick command.

The final two parameters, CurrentValue and NewValue, tie the pop-up menu to a field or variable. CurrentValue specifies what item in the menu should be highlighted when the menu first appears. After the user makes a choice, the result will be placed in NewValue. NewValue must be a variable (see “Variables” on page 1370). If you need this value in a field, the procedure must copy it from the variable into the field after the PopUp command. (However, if the user pulls the mouse off the menu and releases the mouse without making a choice, NewValue will be empty. In this case you probably want to leave the original field alone.)

The example below shows a simple procedure for a pop-up menu that allows a medal to be selected. This example assumes that the database contains a field named Medal.

```
local PopTop,PopLeft,ButtonHeight,ButtonWidth
local TempMedal
GetLocalButton PopTop,PopLeft,ButtonHeight,ButtonWidth
PopUp replace("Gold;Silver;Bronze",";",¶),PopTop,PopLeft,Medal,TempMedal
if TempMedal≠""
   Medal=TempMedal
endif
```

The only problem with this example is that it is a lot of work, and the same effect could be achieved in a few seconds with the Pop-Up Menu SuperObject™. Only use this technique when you need to accomplish something that absolutely cannot be done any other way (for example a non-text menu or a menu that pops up anywhere in a large area).

The PopUpButton Statement

This statement makes it simpler to produce a pop-up menu in response to clicking on a button. Unlike the popup statement, you don’t have to specify the pop-up menu location. The menu will automatically pop-up over the button that was pressed. This statement should not be called unless the original procedure was triggered by a button. This button must have the click / release option turned off.

The popupbutton statement has three parameters:

```
PopUpButton MenuFormula, CurrentValue, NewValue
```

These three parameters are exactly the same as the corresponding parameters described above for the popup statement. Here is an example of how this statement can be used.

```
popupbutton "Red"+¶+"Green"+¶+"Blue","Red",ColorChoice
```

The PopUpClick Statement

This statement makes it simpler to produce a pop-up menu in response to clicking somewhere on a form. Unlike the popup statement, you don't have to specify the pop-up menu location. The menu will automatically pop-up at the current mouse position. This statement is usually triggered by a transparent button that covers a large area. This button must have the click / release option turned off.

The popupclick statement has three parameters:

```
PopUpClick MenuFormula, CurrentValue, NewValue
```

These three parameters are exactly the same as the corresponding parameters described above for the popup statement. Here is an example of how this statement can be used.

```
popupclick "Red"+¶+"Green"+¶+"Blue","Red",ColorChoice
```

The PopUpFieldChoices Statement

This statement can be used to automatically produce a pop-up menu containing a list of data values in a field. The statement must be triggered in response to clicking on a button. When the button is clicked, the menu will pop-up over the button. The menu will pop-up over the button that was pressed. The button must have the click / release option turned off.

The popupfieldchoices statement has three parameters:

```
PopUpClick Choice, Field, InitialChoice
```

The *Choice* parameter is the name of a field or variable. When the statement is complete this field or variable will contain the value that was chosen from the pop-up menu. If no choice was made this field or variable will contain "".

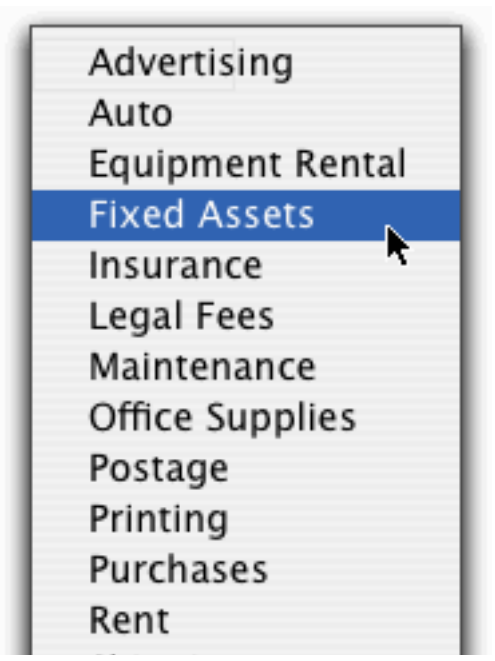
The *Field* parameter is the name of the field that contains the data to be included in the pop-up menu. Panorama will build a list of all of the current values in this field, and include those values in the menu.

The *InitialChoice* parameter is the initial value of the pop-up menu. If possible, the this value will be initially selected when the menu appears.

This procedure is in a checkbook database:

```
local newCategory
popupfieldchoices newCategory,Category,""
```

When the button is clicked, a list of all the categories in the checkbook appears:



As new categories are added to the database they will automatically be added to the menu.

The PopUpDoubleFieldChoices Statement

This statement can be used to automatically produce a pop-up menu containing a list of data values in a field, along with submenus for data values in a second field. The statement must be triggered in response to clicking on a button. When the button is clicked, the menu will pop-up over the button. The menu will pop-up over the button that was pressed. The button must have the click / release option turned off.

The `popupfieldchoices` statement has four parameters:

```
PopUpClick Choice, Field, SubField, InitialChoice
```

The *Choice* parameter is the name of a field or variable. When the statement is complete this field or variable will contain the value that was chosen from the pop-up menu. If no choice was made this field or variable will contain "".

The *Field* parameter is the name of the field that contains the data to be included in the primary pop-up menu. Panorama will build a list of all of the current values in this field, and include those values in the menu.

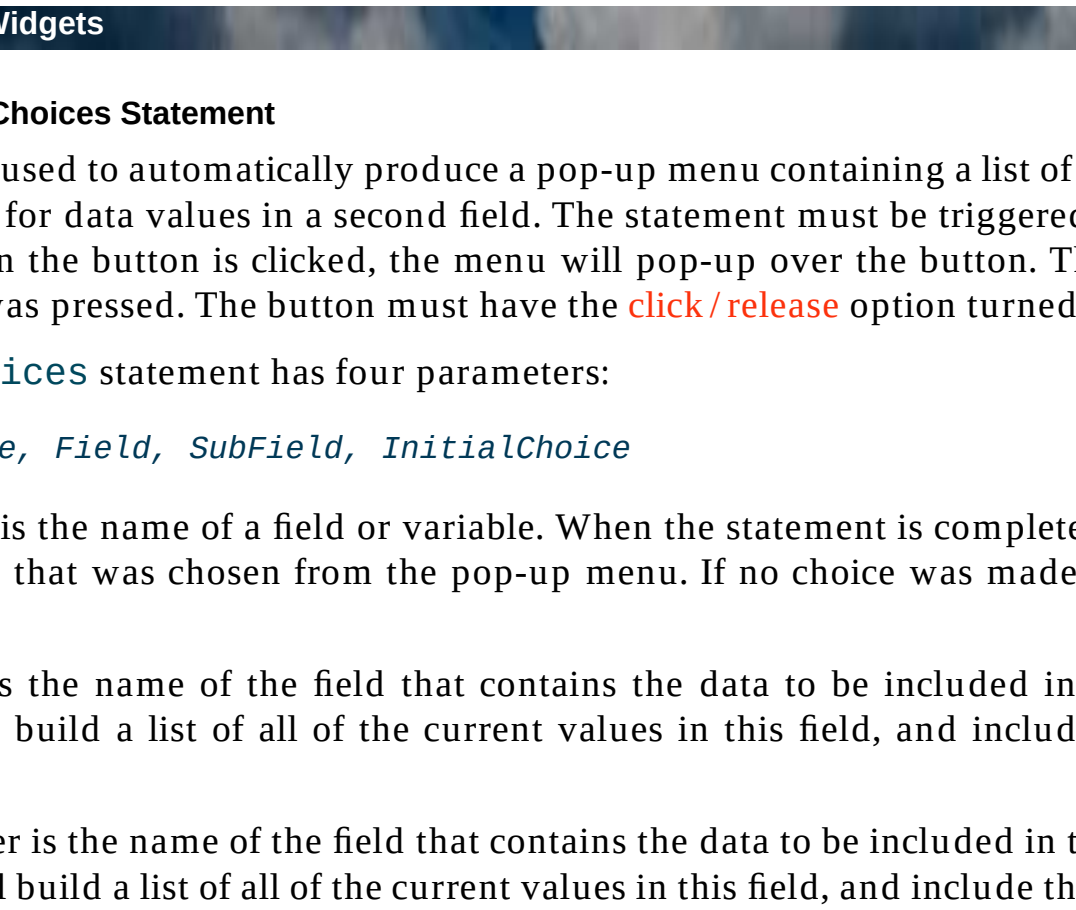
The *SubField* parameter is the name of the field that contains the data to be included in the secondary pop-up menus. Panorama will build a list of all of the current values in this field, and include those values in the submenus.

The *InitialChoice* parameter is the initial value of the pop-up menu. If possible, the this value will be initially selected when the menu appears.

This procedure is in a checkbook database:

```
local newCategory
popupdoublefieldchoices newCategory,Category,PayTo,""
```

When the button is clicked, a list of all the categories in the checkbook appears, along with a submenu for each category listing the actual payees for each category:



If there are many items in the menus and submenus there may be a delay while the menus are being calculated.

The PopUpByNumber Statement

The PopUpByNumber statement is exactly the same as PopUp except for the last two parameters. The PopUpByNumber statement treats CurrentValue and NewValue as menu item numbers, instead of as text. For example, if the user selects the third menu item, NewValue will be 3. This command is useful for non-text menus, for example, a menu of color swatches.

The PopUpStyle Statement

The PopUpStyle statement allows you to control the font, size and color of pop-up menus created with the PopUp and PopUpByNumber statements. The PopUpStyle statement must be placed directly above the PopUp or PopUpByNumber statement in your procedure. This statement has four parameters:

```
PopUpStyle Font, Size, Color, BackgroundColor
```

The first parameter, Font, is the font you want to use for the menu, for example "Chicago" or "Geneva".

The second parameter, Size, is the size of the text you want to use for the menu, for example 9 or 12.

The final two parameters specify the foreground and background colors for the menu, which may be created with the rgb(or hsb(functions (see “Colors” on page 1294). For a standard black on white menu you can simply use empty strings for the colors: "","".

Here are a couple of examples of the PopUpStyle statement. Both set the menu to Geneva 9pt, but the second example creates a menu with red text on a dark gray background.

```
PopUpStyle "Geneva",9,"",""
PopUp replace("Gold;Silver;Bronze",";",¶),PopTop,PopLeft,Medal,TempMedal

/* red text on dark gray background */
PopUpStyle "Geneva",9,rgb(65535,0,0),rgb(50000,50000,50000)
PopUp replace("Gold;Silver;Bronze",";",¶),PopTop,PopLeft,Medal,TempMedal
```

“Live” Pop-Up Menu Formulas

If the first character of the text generated by the menu formula is (, Panorama will build the pop-up menu using the same rules used for “Live Menus” in the menu bar (see “Live Menus” on page 1465). This means that pop-up menus can include checked items, dimmed items, bold, italic, and even submenus!

If you are defining a normal pop-up menu with no submenus, the formula should start with the menu(function (see “The FileMenuBar Statement” on page 1465). Since this is a pop-up menu you can leave the name blank if you want to. A pop-up menu with the formula shown below could be used to select Southern California field offices, with the main offices in bold.

```
local PopTop,PopLeft,ButtonHeight,ButtonWidth
local officeChoice
GetLocalButton PopTop,PopLeft,ButtonHeight,ButtonWidth
popup menu("")+menuitems("Los Angeles¬<B;Santa Ana;San Diego¬<B;Riverside;Ventura"),
   PopTop,PopLeft,"Los Angeles",officeChoice
```

Here's what this pop-up menu will look like. The main offices appear in bold (Los Angeles and San Diego).



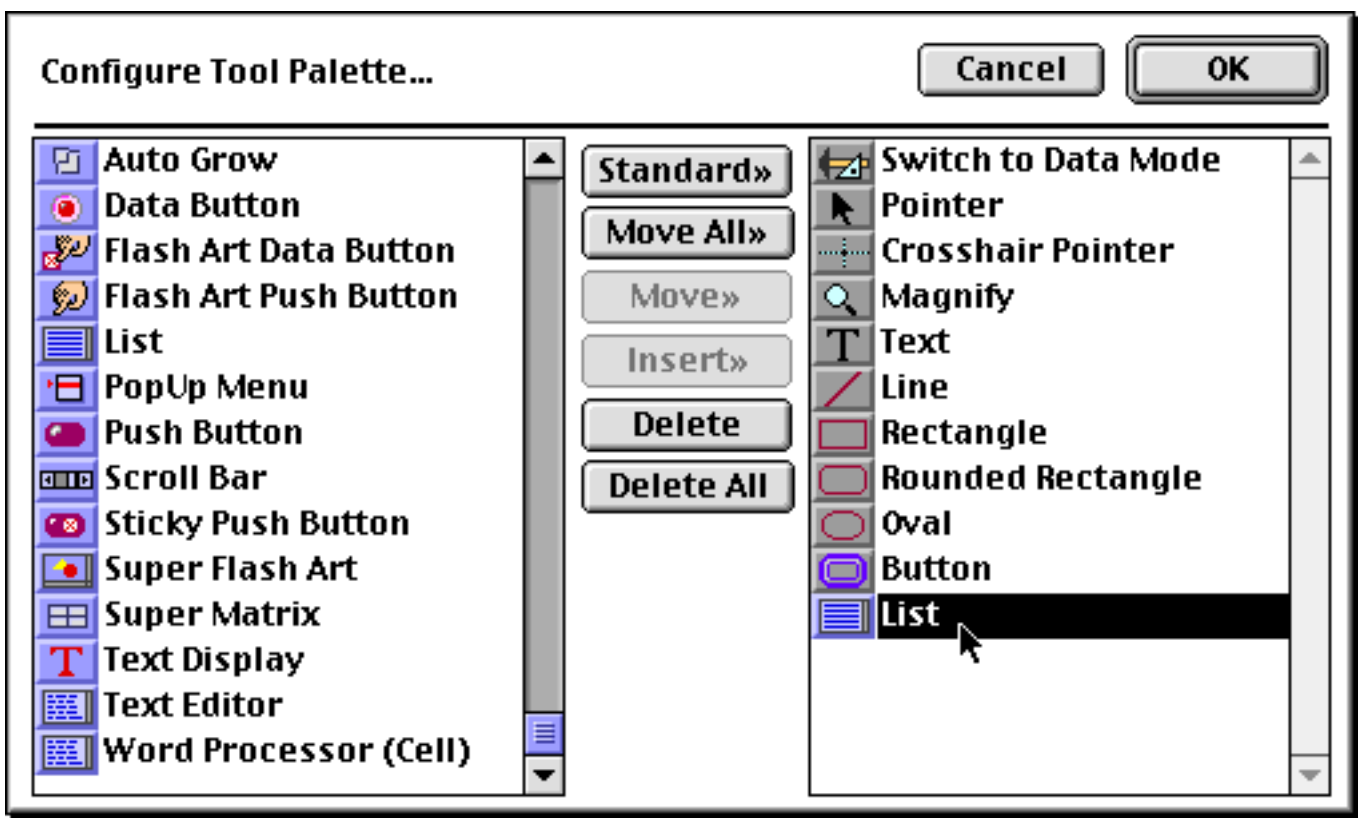
For more details on using the menuitems(function, see “Live Menus” on page 1465.

List SuperObjects

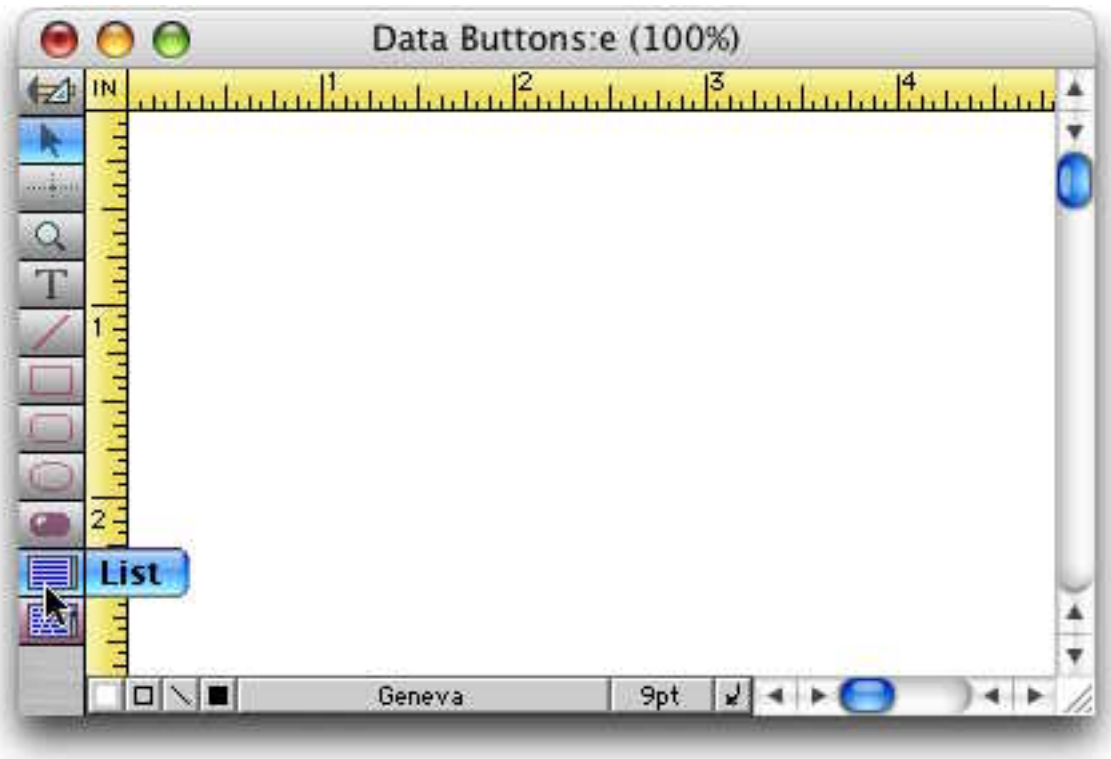
Using a List SuperObject™ it's easy to add a scrolling list to a Panorama form or dialog. This is the same type of scrolling list used in the standard Open and Save dialogs. The scrolling list allows a large amount of data to be displayed in a small area. Using the scroll bars, the user can quickly locate the items they are interested in. Each Panorama scrolling list can be filled with information from a field or variable or from an entire database.

Creating List SuperObjects™

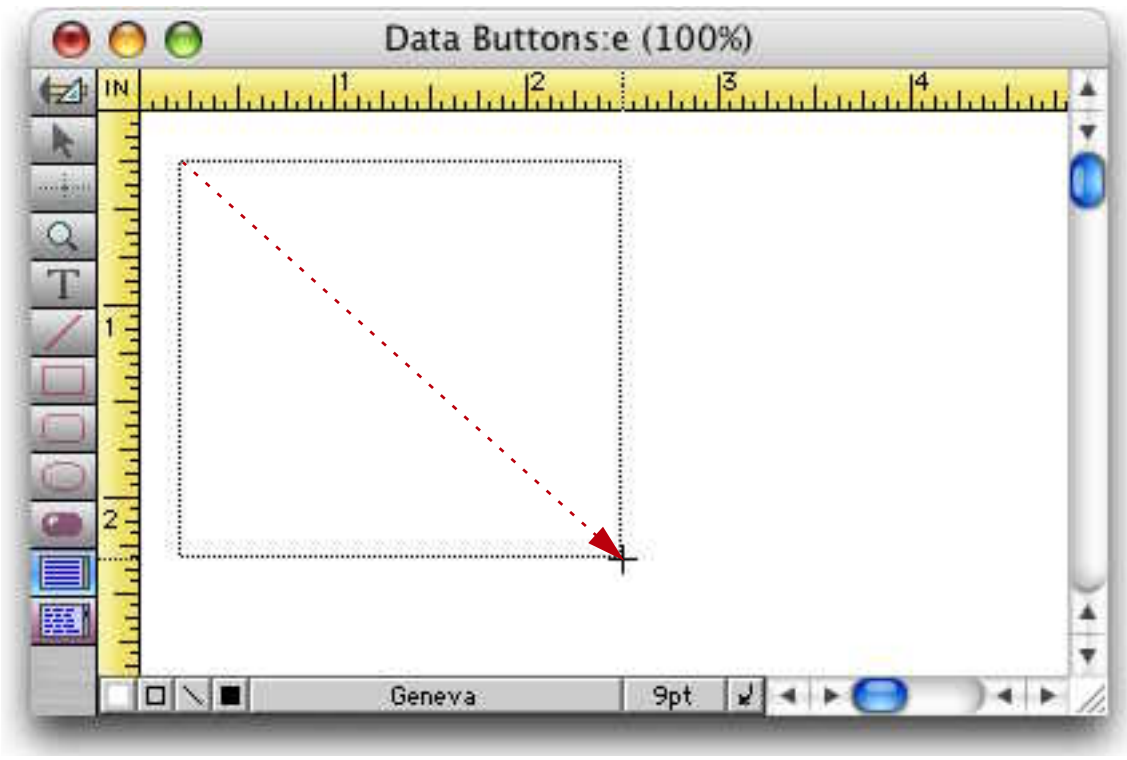
List objects are created just like any other SuperObject™. First make sure that the **List** tool is installed in the tool palette (see "Customizing the Tool Palette" on page 497 if it isn't.)



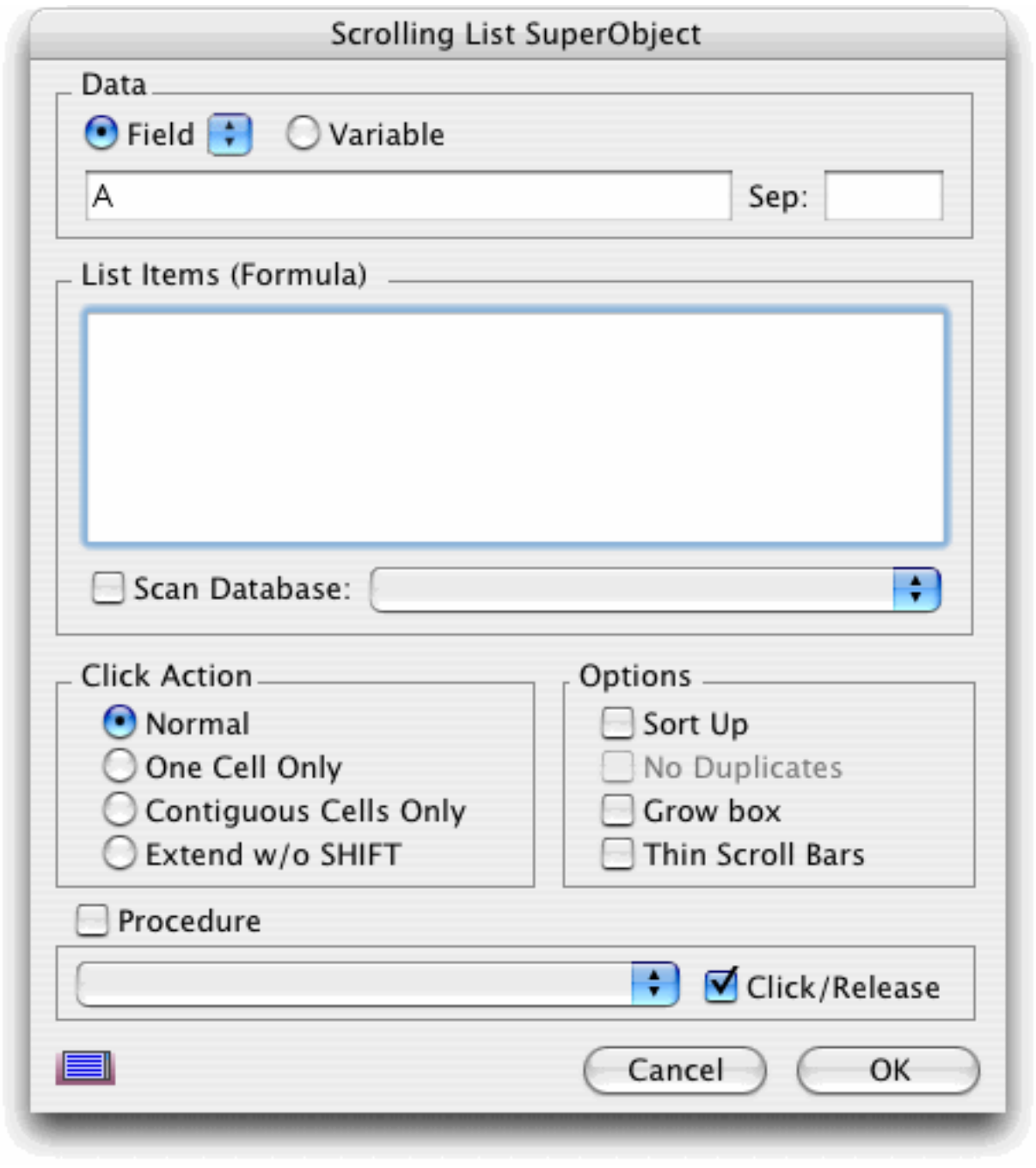
Select the **List** tool…



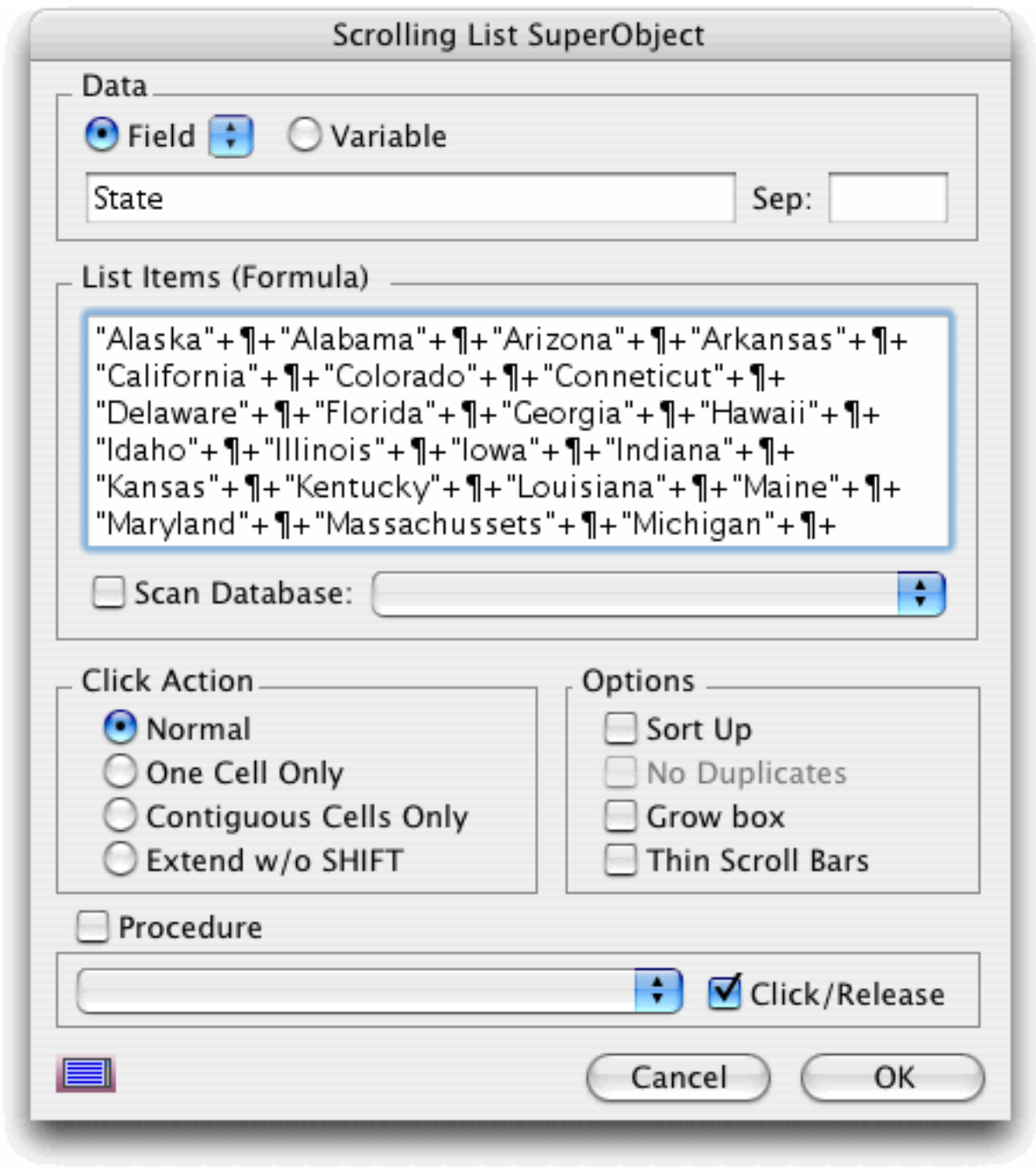
then drag the mouse across the form in the spot where you want the text to appear.



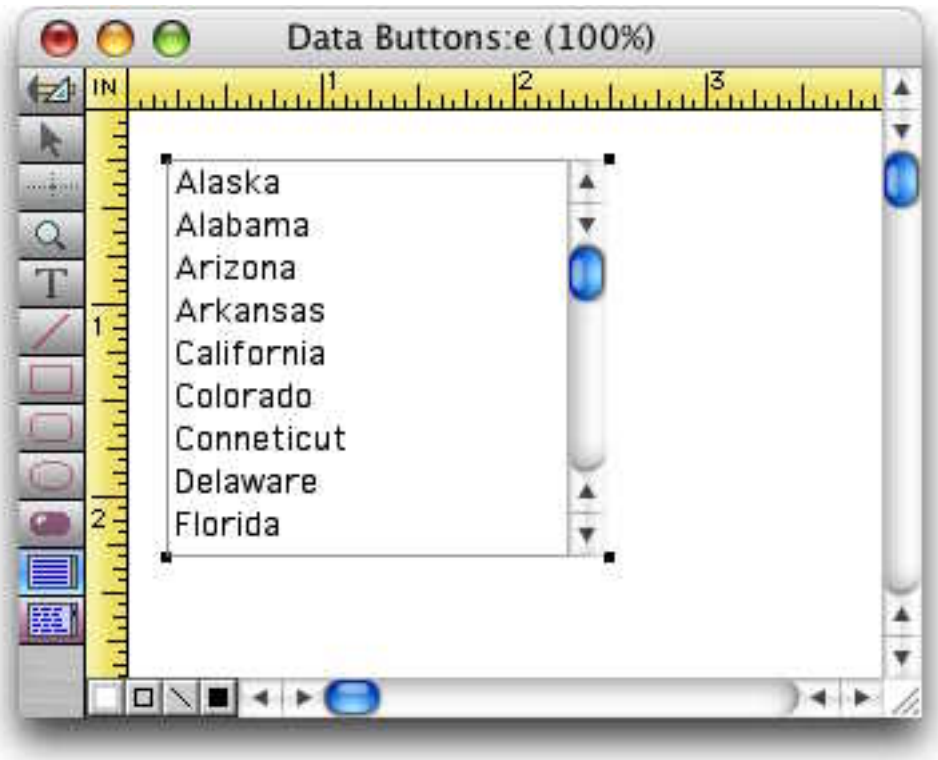
When you release the mouse, the List configuration dialog appears.



In some ways configuring a list is similar to configuring a pop-up menu. You need to specify a field or variable to hold the result, and you need to supply a formula to generate a list of items. Each item is separated by a carriage return. Here's a typical configuration to display a list of state names. (To type the ¶ symbol, press **Option-7** on the Macintosh and **Alt-0182** on the PC.)



Press OK to create the List object.

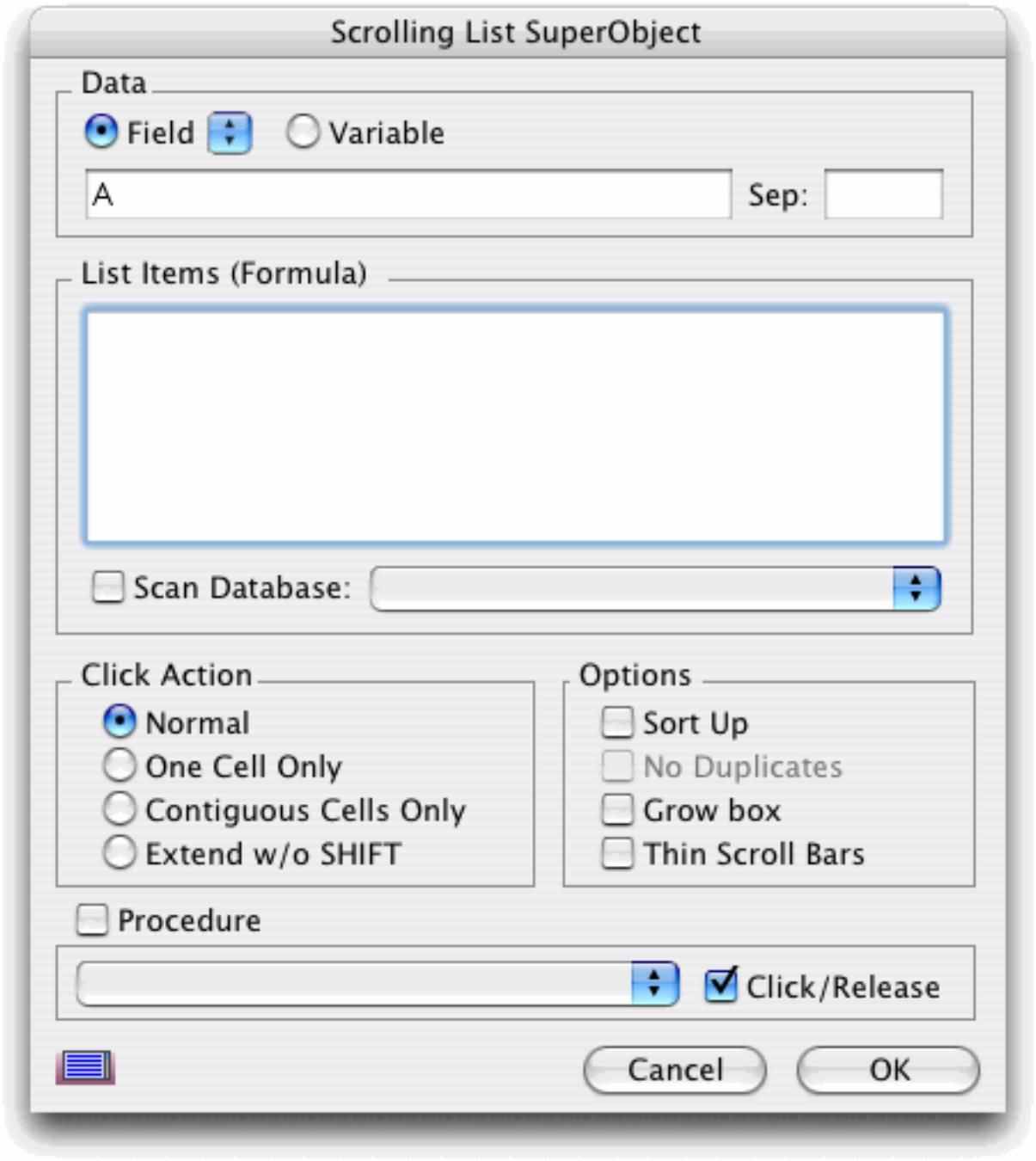


To actually use the list, switch to Data Access Mode. In this mode you can use the scroll bar to slide the contents of the list up and down, and click on list items to select them.



List Options

The SuperObject™ List dialog is divided into several sections. This dialog allows you to configure the way the text is calculated and formatted. The options in this dialog are described in the following sections.

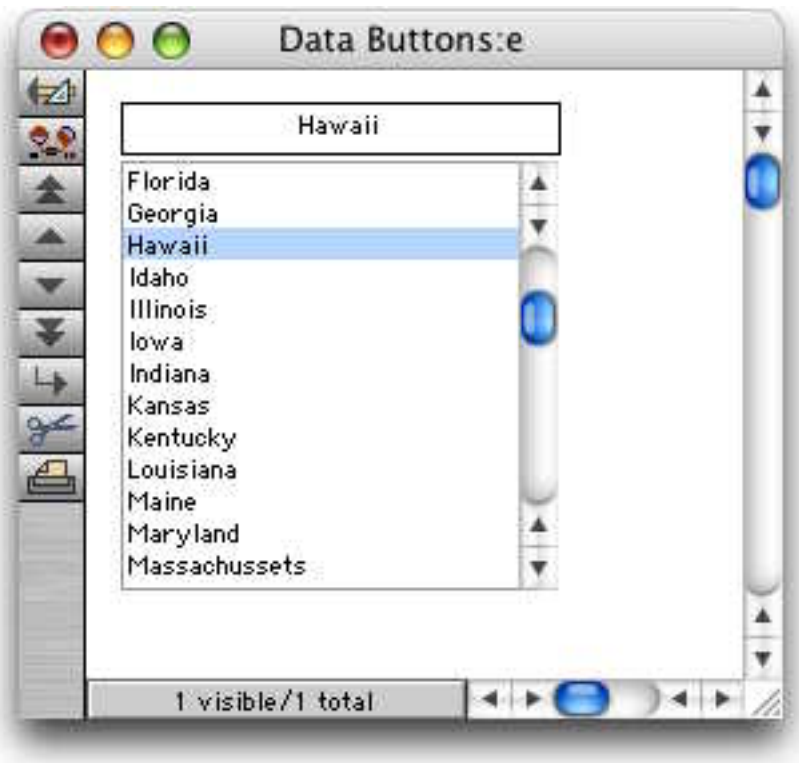


Once the options are set, press the **OK** button and the List object is ready to use. If you need to change the options later, double click on the List object to re-open the dialog.

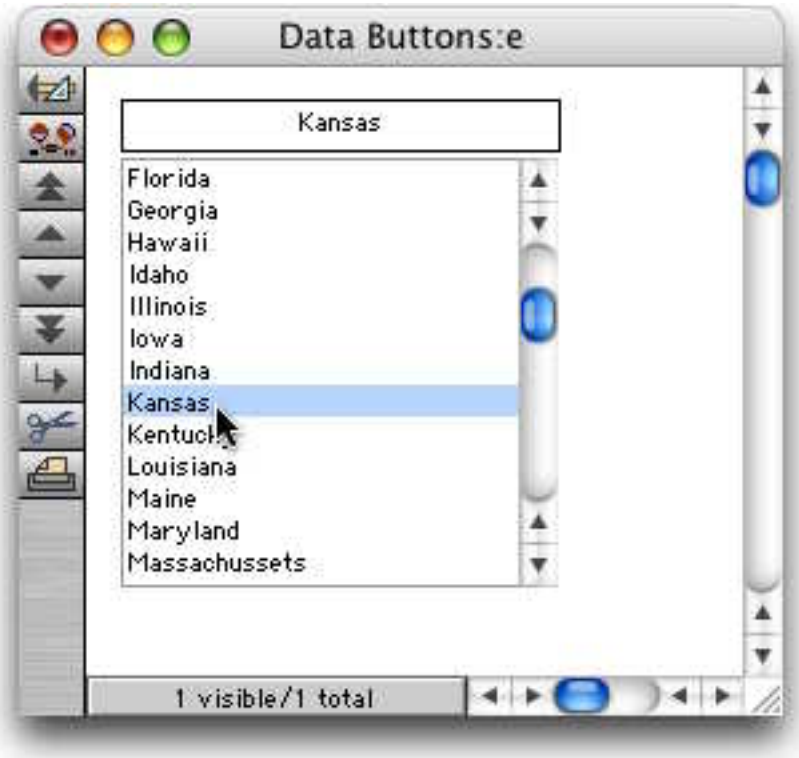
Data

This section of the dialog specifies the field or variable associated with this list, if any. This field or variable doesn't contain the list itself, but only the selected value(s) within the list. Type the name of the field or variable into the box (or select the field name from the pop-up menu next to the Field radio button). If the list is associated with a variable that has not been created with a procedure, Panorama will automatically create a global variable with this name whenever the list appears. This global variable can be used in formulas and procedures just like any other global variable.

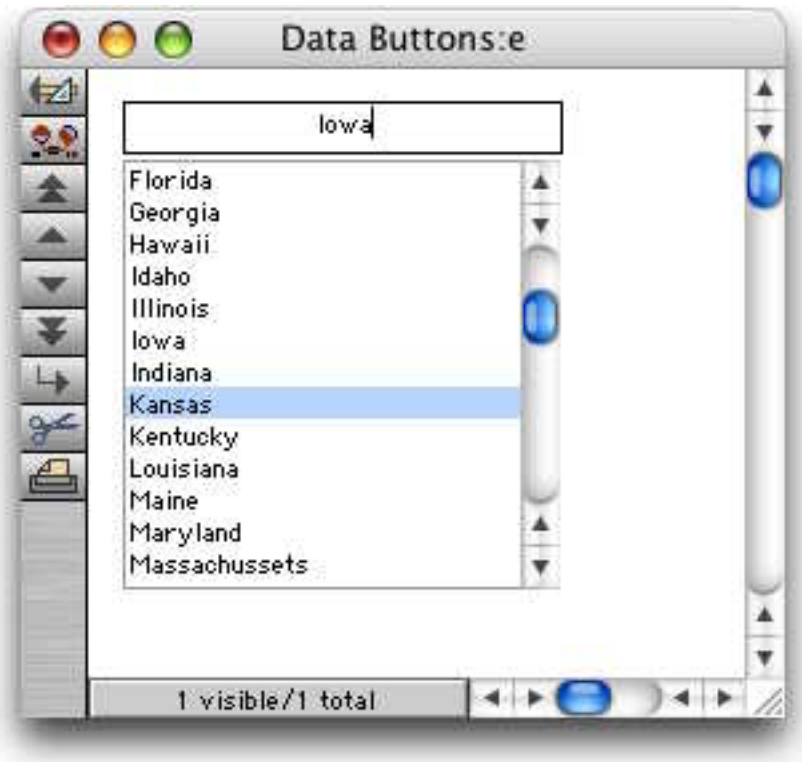
When first learning how to use the List SuperObject it's handy to include a Text Editor SuperObject on the form to allow you to see and modify the data value (see "Text Editor SuperObject" on page 639). In this example both the Text Editor and the List objects are displaying the same value — the field State.



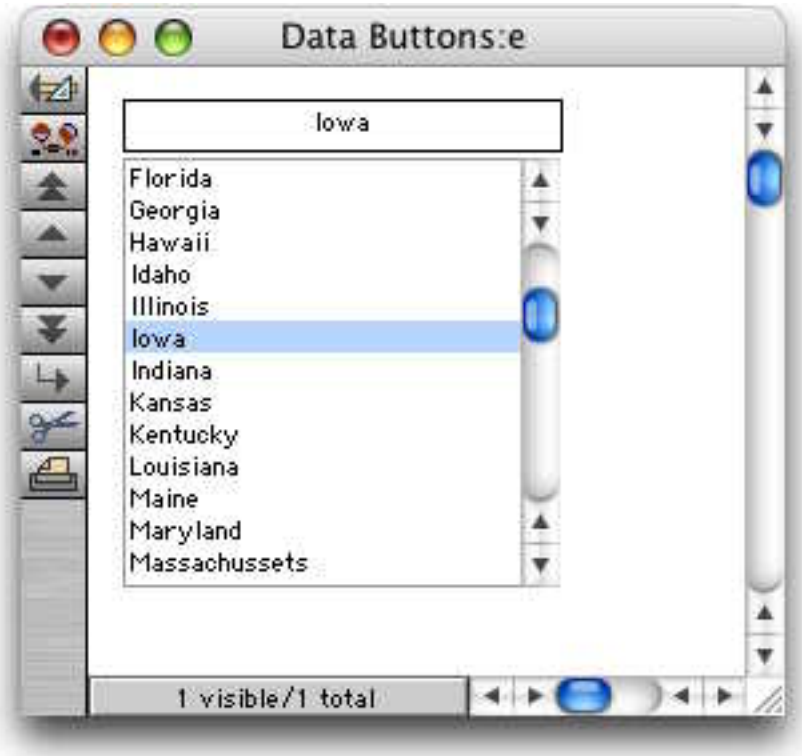
Clicking on an item fills the field with that value.



It's also possible to work this process in reverse. If you type a value into the Text Editor SuperObject the List will automatically select the corresponding item in the list, if any. For example, you could type Iowa into the field:



When you press **Enter** Iowa will be selected in the list.

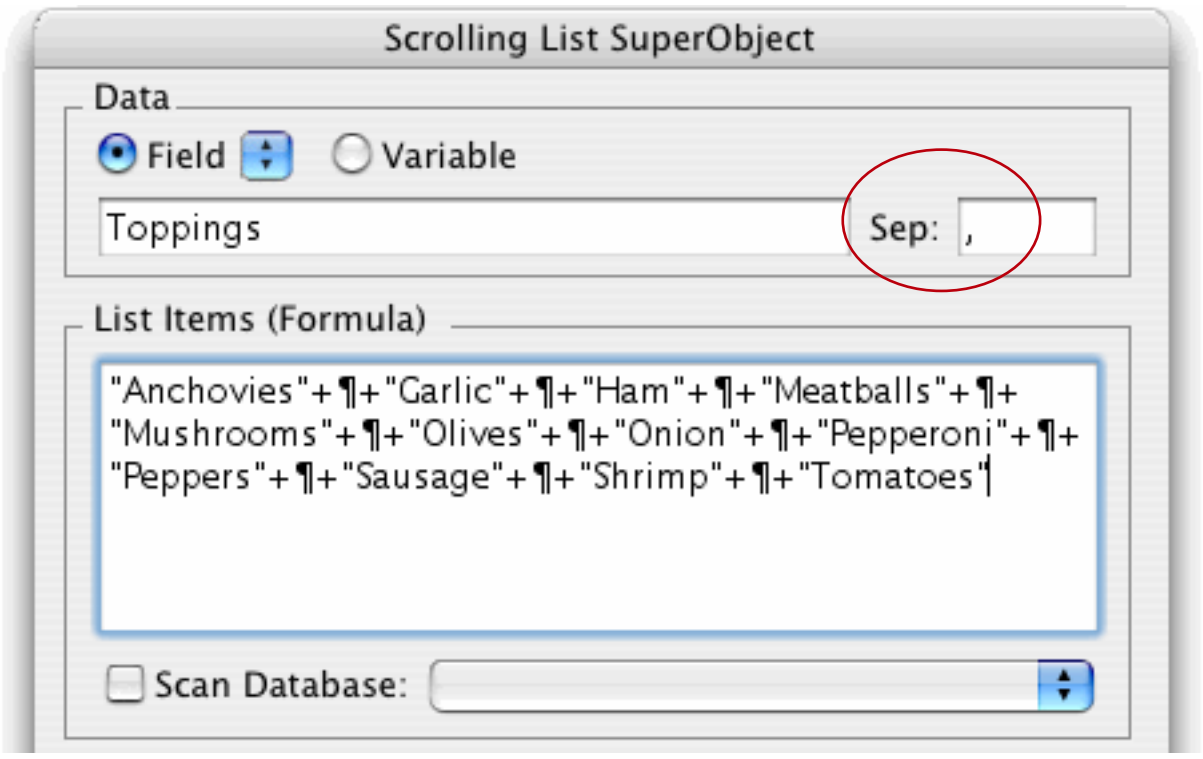


The item must be spelled exactly the same as it appears in the list, including capitalization. Only Iowa will work, not iowa or IOWA.

Sep

This is the separator text that will appear between each value if multiple items are selected in the list. Common separators include commas, spaces, slashes and hyphens. The separator may be up to 6 characters long. (Note: If the separator is left blank, Panorama will use a carriage return as the separator.)

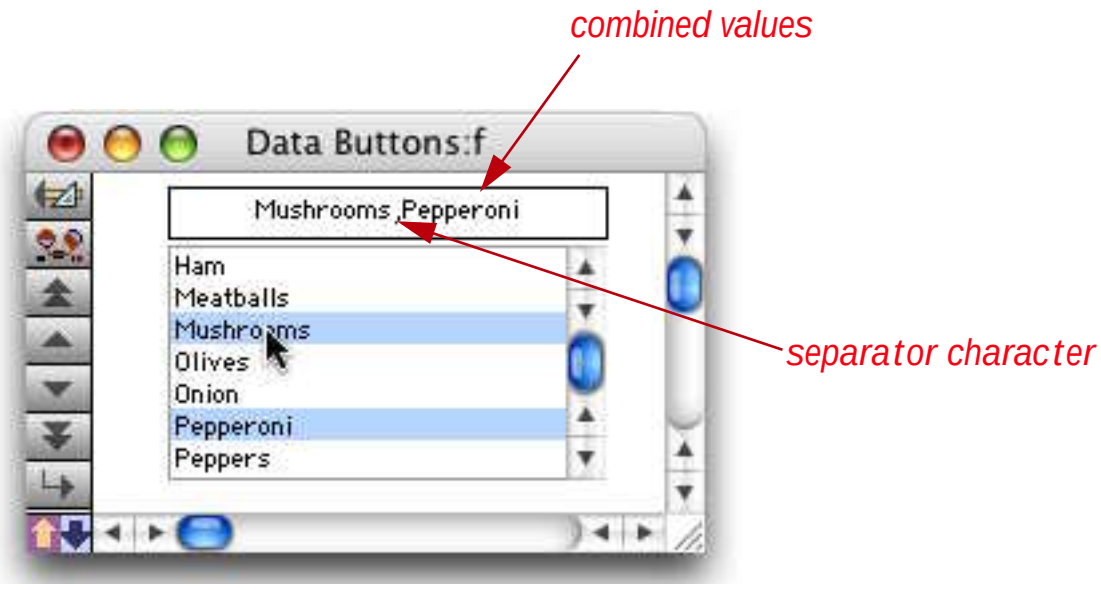
As an example of multiple items, consider pizza toppings. A pizza may have one, two, three or more toppings, or even none at all. You can create a list that shows all the different pizza toppings. In this example the separator character is a comma.



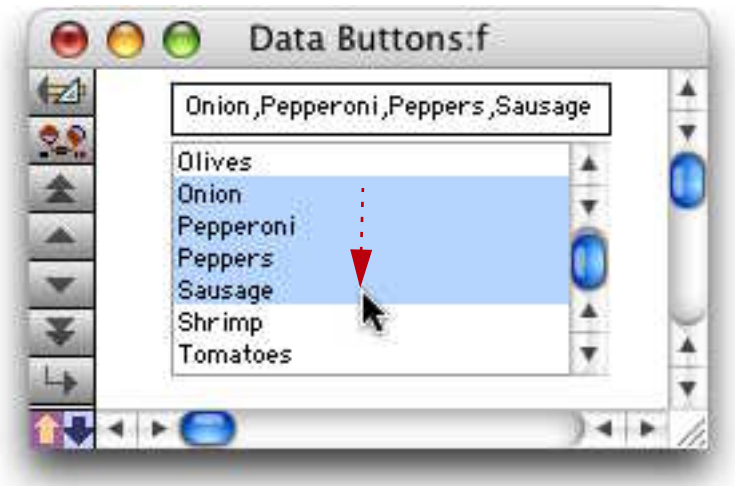
When you click on a single topping it appears in the field, just as in the previous example.



By holding down the **Command** key (Mac) or the **Control** key (PC) you can click on and select additional items, as shown below. The data field (or variable) will contain all of the selected items, with each item separated by a comma (or whatever separator character you have specified.) The values will always be in the same order as they appear in the list, no matter what order you click on them (unlike a group of radio buttons — see "Multiple Value Button Groups" on page 835).



You can continue to select additional items if you wish. You can also de-select an item by holding down the **Command** (Mac) / **Control** (PC) key and clicking on the selected item. Another option is to hold down the **Shift** key and drag across several items to select them all.



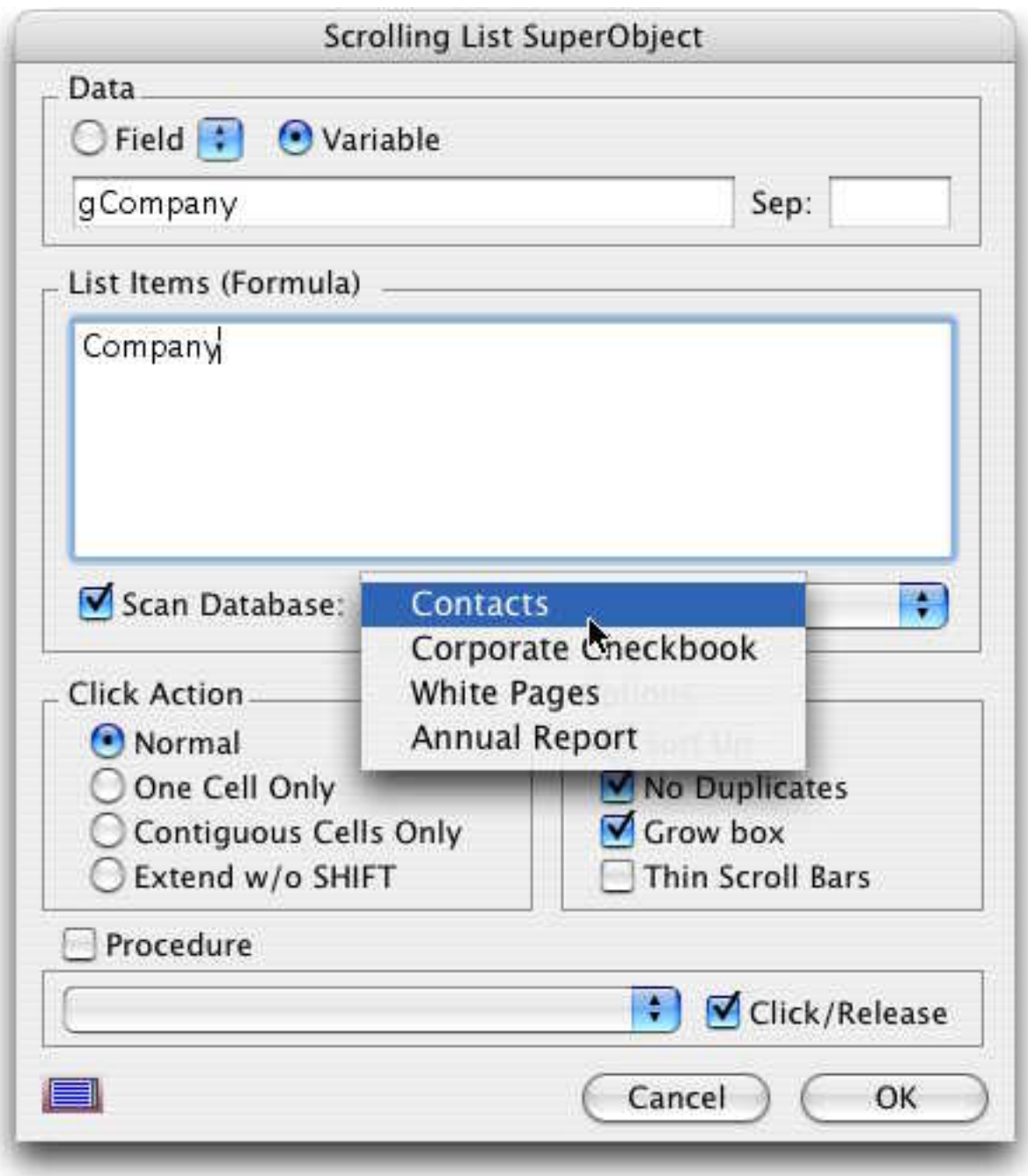
You can use any separator character you want. If you leave the Sep option empty, Panorama will use a carriage return as the separator. In other words, each item will be on its own separate line.

Database

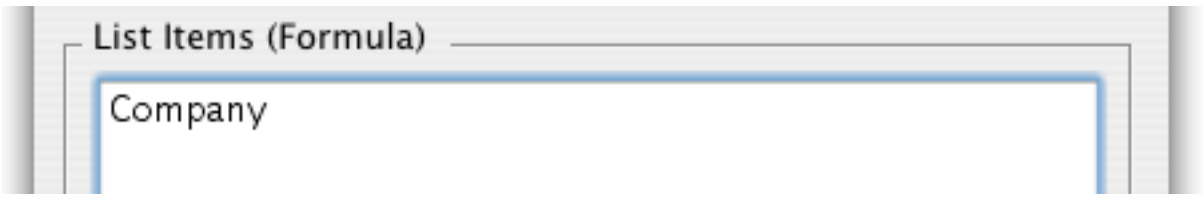
There are two ways that Panorama can build the list of items: it can use a formula to build the list or it can scan a database to build the list. So far all of the examples have shown building the list with a formula. If you want the list built by scanning an entire database, select the name of the database from the pop-up menu in this section (the database must be open). To illustrate this feature we'll use this contact database.



Create a List SuperObject the usual way (see "Creating List SuperObjects™" on page 869). Select the database to be scanned from the pop-up menu of open databases. In this case you'll select the name of the current database.



In the formula field you must enter a formula that will be used to process each field within the database. Usually this is simply a field name, which can either be typed in or selected from the **Field** menu. For this first example we'll build a list of company names.



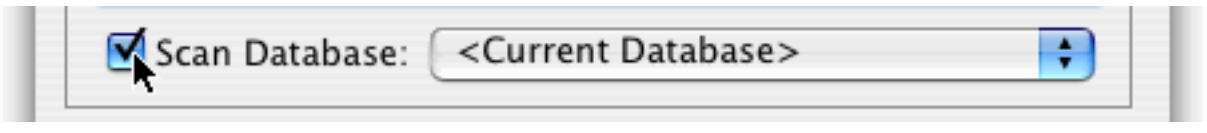
When you press **OK** the list appears, already filled in with the items scanned from the database.



If the database name ever changes, this list will stop working. Here's how this problem can be fixed. Open the configuration dialog again (by double clicking on the object). Click on the Database checkbox to disable the database scanning procedure.



Now click the checkbox again to turn database scanning back on. As you can see, this enables scanning of the current database, no matter what the name is.



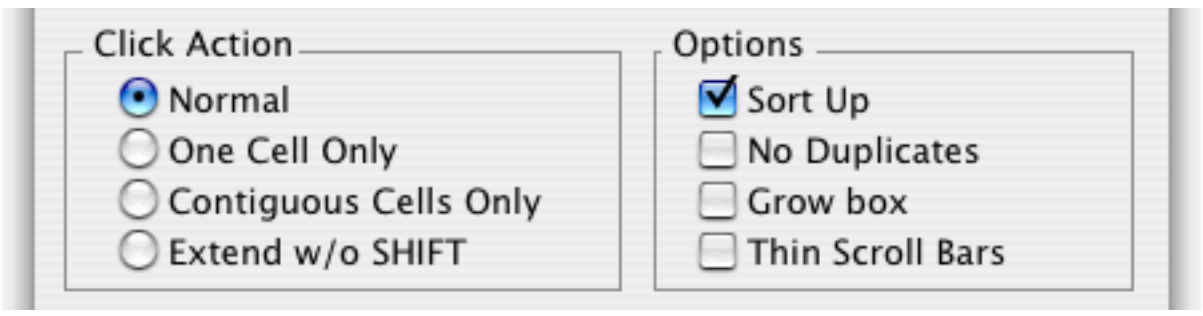
This trick only works with the current database. If the list is built from another database you'll have to make sure that the database name doesn't change (or if it does, you'll have to open the configuration dialog and adjust the name to make the list work again).

Sort Up

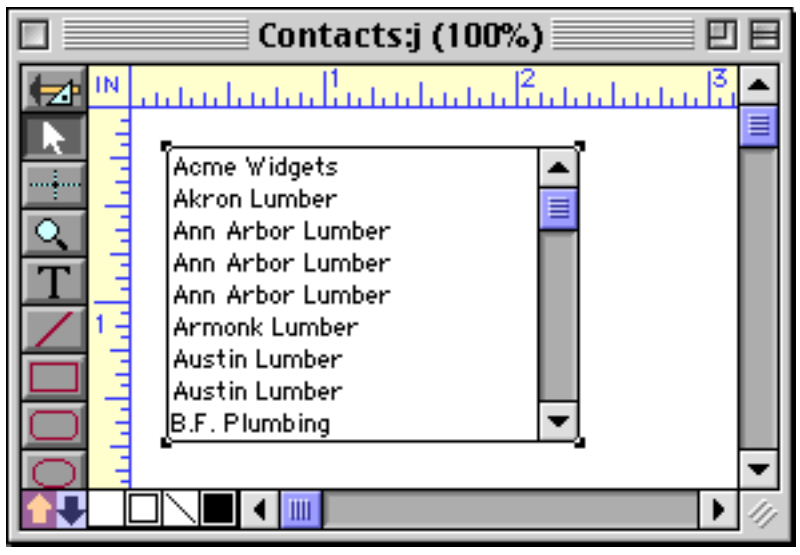
If this checkbox is turned on, the list of items will be sorted in ascending alphabetical order, otherwise the list will be displayed in the order the data was scanned. For example, the list of companies built in the previous example displays the company names in the order they appear in the database, which is definitely not alphabetical.



To display the list in order, enable the Sort Up option.



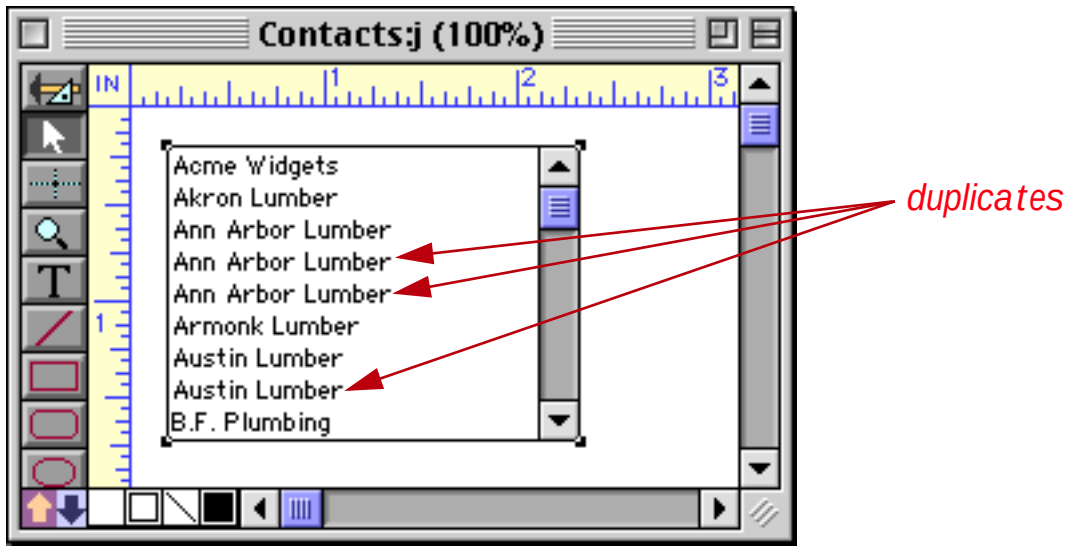
Here's the alphabetized list.



The Sort Up option works both with database scanning and when the list is generated by a single formula.

No Duplicates

If this checkbox is turned on, duplicate data items will be eliminated from the list of items. (This option is not available if the Sort Up option is not also turned on.) For example, the list generated in the previous example contains many duplicate company entries.



To automatically remove the duplicate items enable the No Duplicates option.



Here is the revised list, without the duplicates.



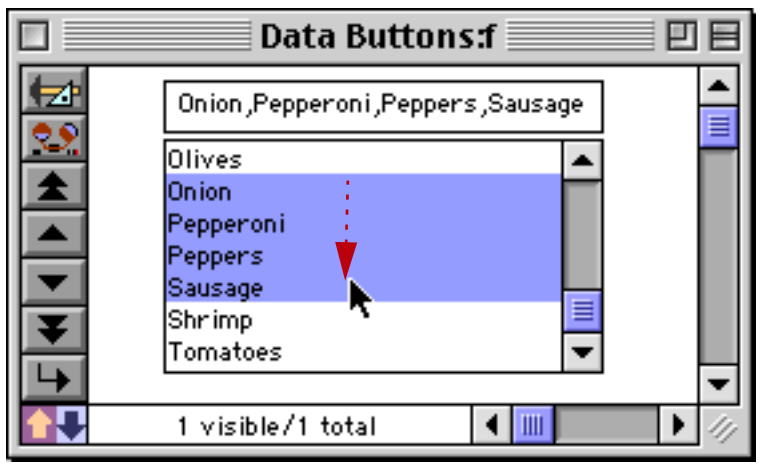
Formula

This section of the dialog contains the formula used to extract the data items from the database, field or variable. See "Building the List" on page 884 for details on how this formula is used.

Click Action

This section of the dialog controls how multiple items in the list are selected.

Normal: This option normally allows only one item to be selected at a time. When the user clicks on an item, it is selected and all other items are un-selected. However, if you hold down the **Shift** key you can drag to select a contiguous range of items.



By holding down the **Command** key (Mac) or the **Control** key (PC) you can click on and select additional items, as shown below.



You can continue to select additional items if you wish. You can also de-select an item by holding down the **Command** (Mac) / **Control** (PC) key and clicking on the selected item.

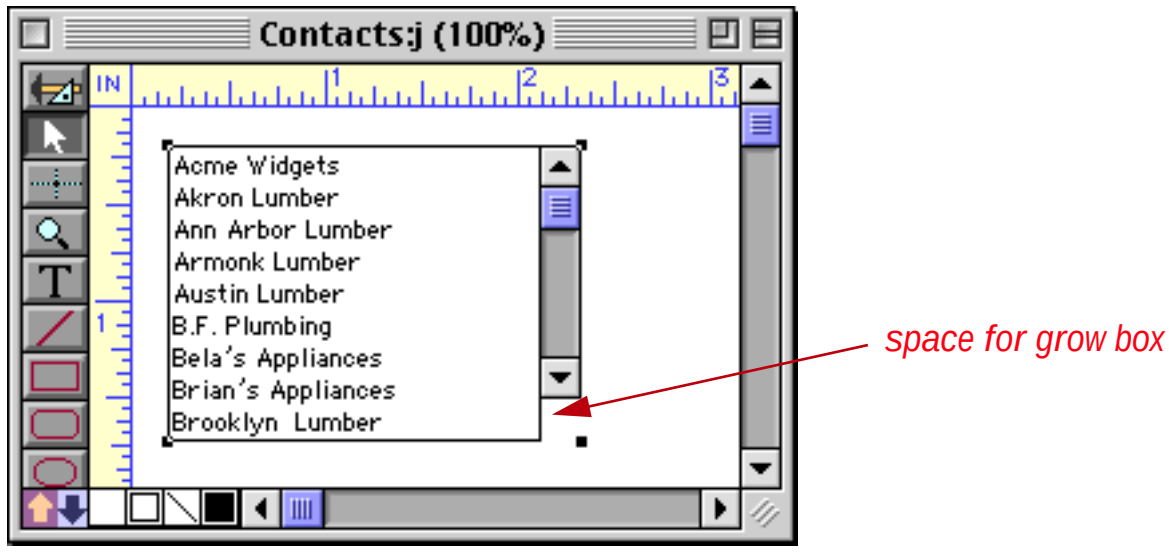
One Cell Only: This option only allows one item to be selected at a time, no matter what modifier keys are pressed.

Contiguous Cells Only: This option is similar to the Normal option, except that holding down the **Command** (Mac) / **Control** (PC) key does not allow you to select/un-select individual items. You can, however, select multiple items by holding down the **Shift** key and dragging across the items.

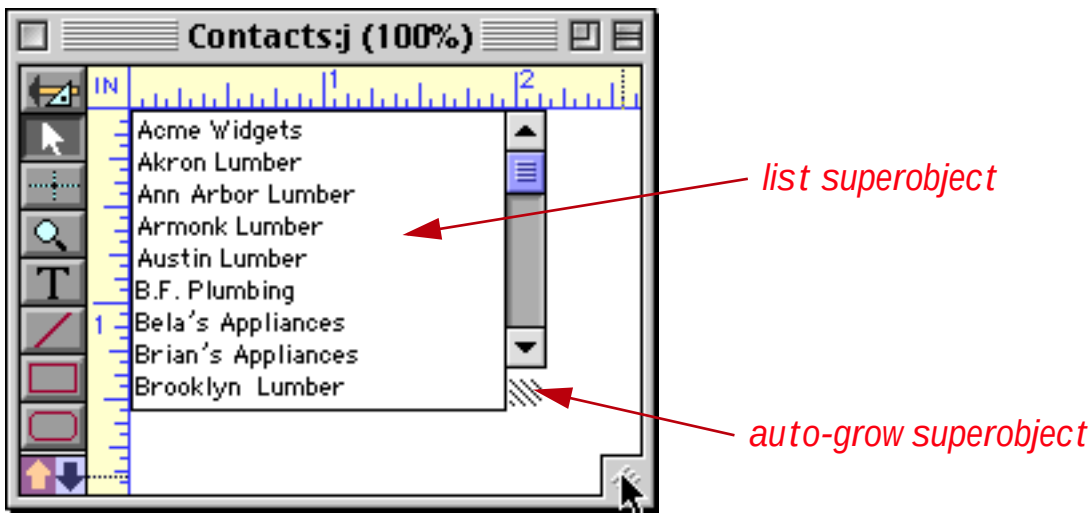
Extend w/o Shift: This option allows you to select a group of items simply by dragging over the items (you don't have to hold down the **Shift** key if this option is enabled).

Grow Box

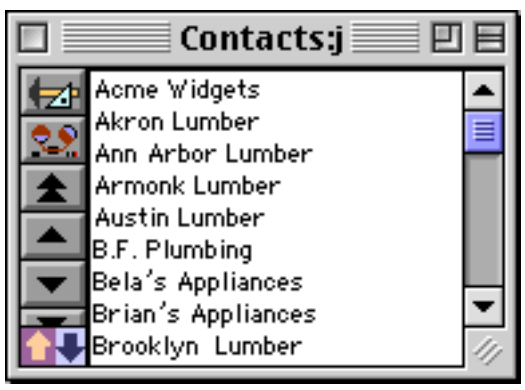
This option makes the scroll bar shorter to reserve space for a grow box in the lower right hand corner of the object.



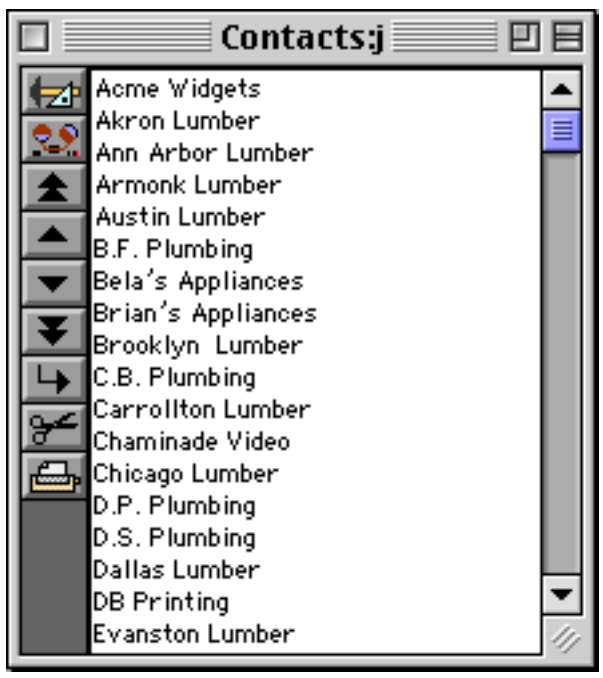
Usually you would only use this if the list was in the lower right hand corner of the window, or even filling the entire window. Then you could use this option with an **Auto-Grow SuperObject** to create an elastic form (see "Elastic Forms" on page 912). Here is an example of such a form in Graphics Mode…



and in Data Access Mode.



This form will automatically adjust to different window sizes.



See "Elastic Forms" on page 912 to learn more about elastic forms.

Thin Scroll Bars

If this option is checked the scroll bar will be narrower (11 pixels wide instead of 16 pixels).

Procedure

This pop-up menu allows you to specify a procedure that will be triggered every time the user clicks on an item in the list.

Click/Release

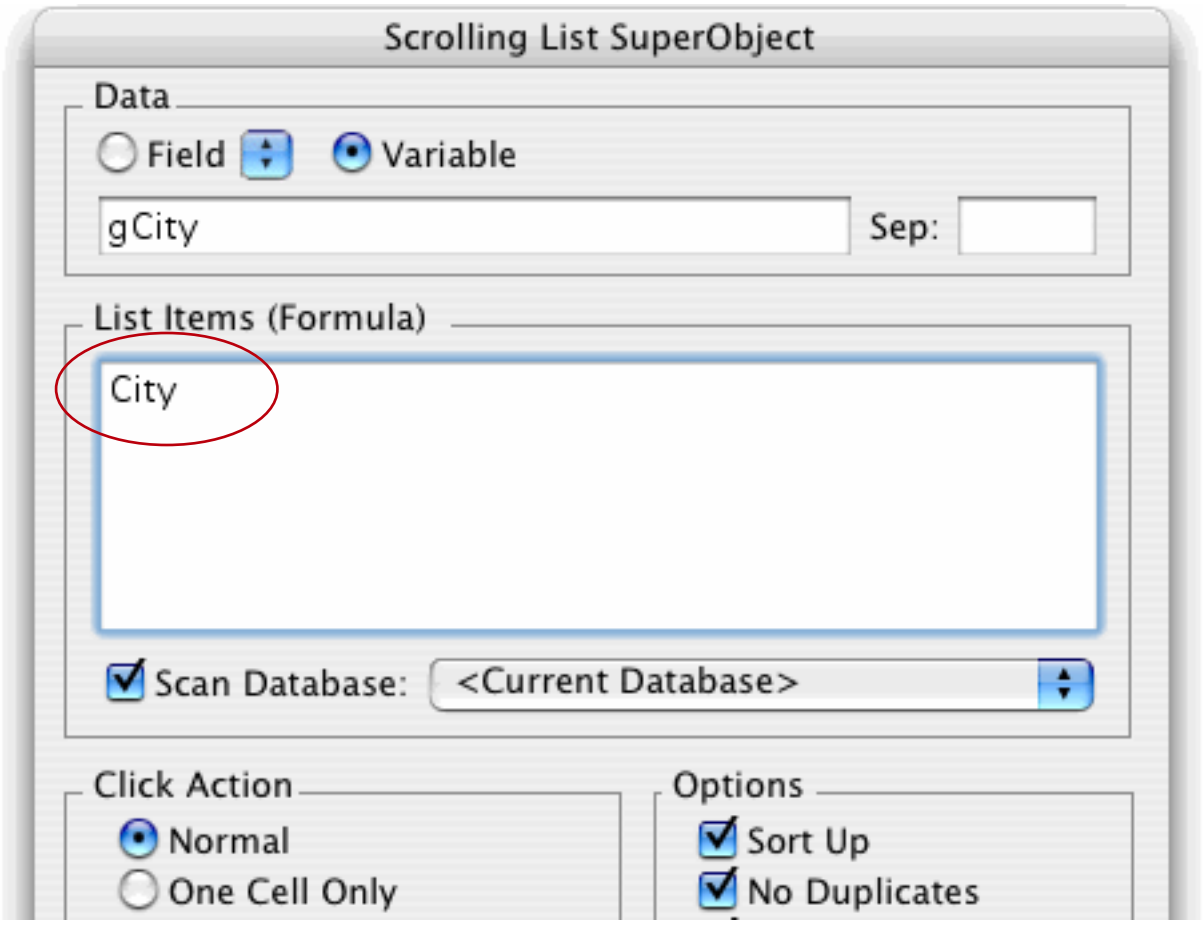
This option controls when the procedure (if any) is triggered. Normally, the Click/Release option is enabled and the procedure is triggered when you release the mouse. If the Click/Release option is disabled the procedure will be triggered immediately, as soon as you click on the list. This means that you cannot hold down the **Shift** key and drag to extend the selection (see "Click Action" on page 882). Turn off the Click/Release option if you want to trigger a procedure to drag list items (see "Using Drag and Drop to Change the Order of Items in a List" on page 1831).

Building the List

There are two ways that Panorama can build the list of items: it can scan a single field or variable, or it can scan an entire database. Panorama normally builds the list once when the form containing the list is first opened. If you need to update the list or change it later, you'll need to send the list a command with a procedure (see "List SuperObject™ Commands" on page 1827).

Scanning a single formula: If this method is selected, Panorama calculates the result of the formula, then scans the result to separate it into individual items for the list. Each line (separated by carriage return) in the result is treated as a separate item. This is the same method used by the Pop-Up Menu SuperObject to build a list; see "The Pop-Up Menu Formula" on page 853 for tips and tricks for setting up this formula.

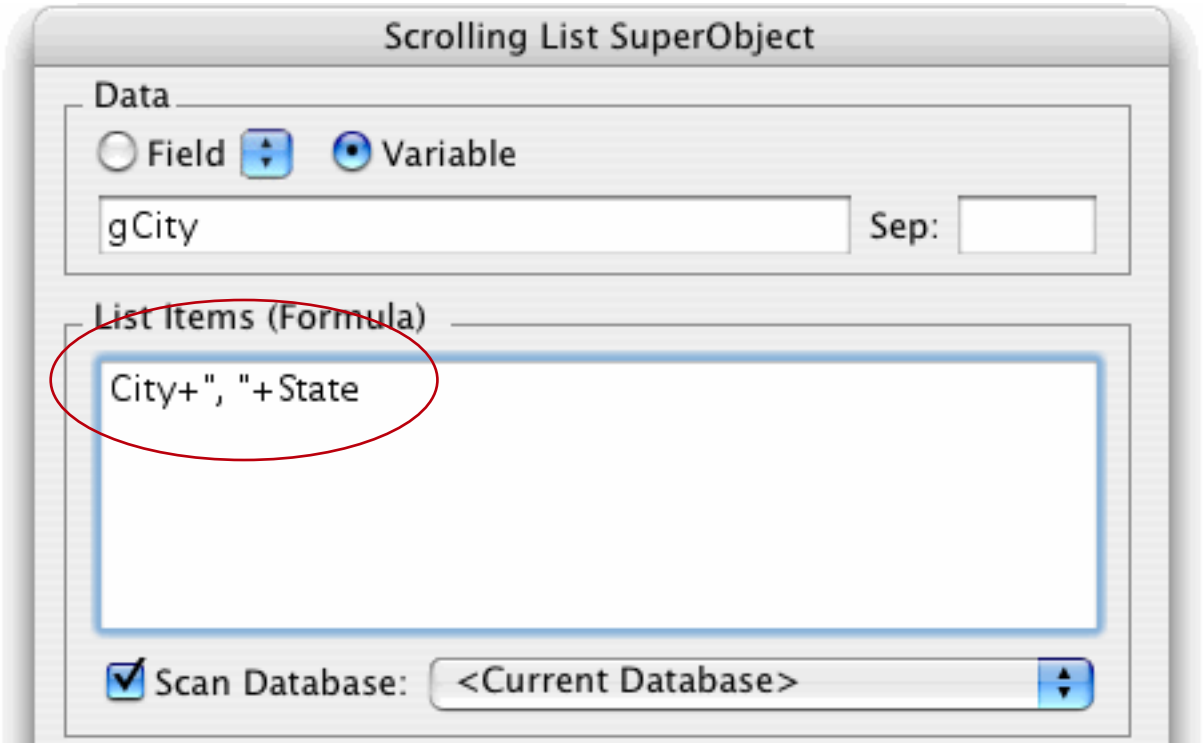
Scanning an entire database: If this method is selected, Panorama will scan the entire specified database, building up the list one record at a time as it scans. The formula determines what information is extracted from each record in the database and places it in the list. For example, suppose you want to build a list of cities extracted from an address database. For this application, the formula would be simply:



For this application you would probably want to turn on the Sort Up and No Duplicates options, so that each city will be listed only once.



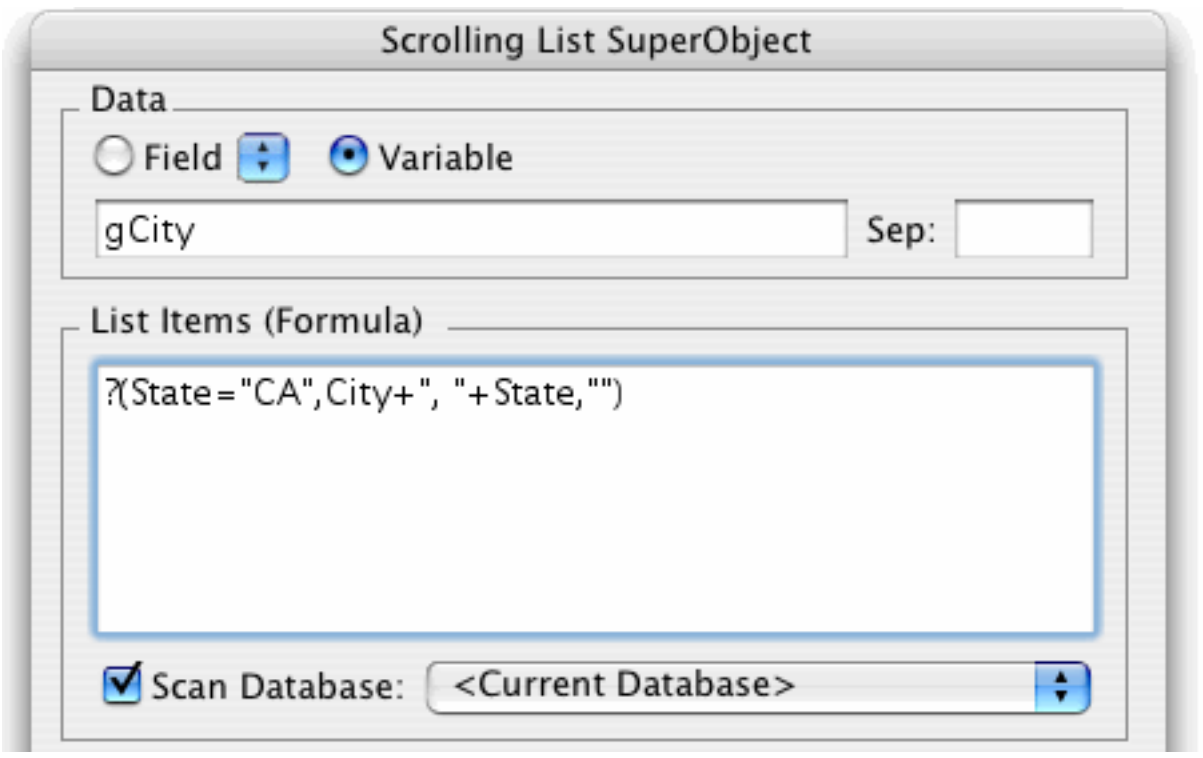
If you wanted both the city and state to appear in the list, a more complex formula could be used:



Here's the revised list.



If you want to build a list of only part of the database, use the ?(function to select only the information you want (see "The ? Function" on page 1271). If a record should not be included in the list, the result of the formula should be an empty string (""). The formula below will fill the list with cities in California.



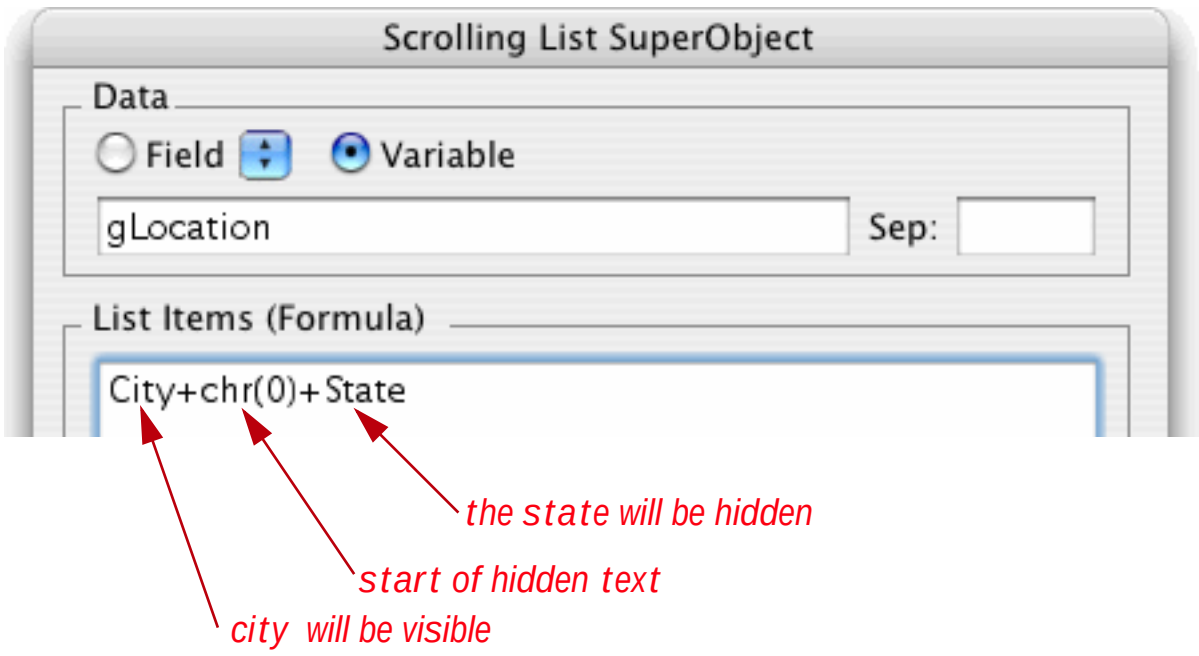
All other states will be left off the list.



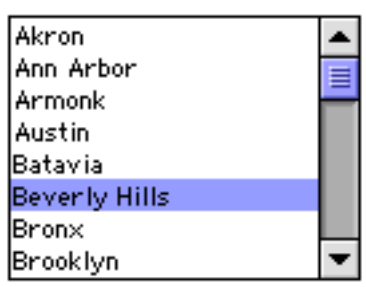
"Hiding" Part of a List Item

The List SuperObject normally supplies all of the text in each line you supply to it. However it is possible to add hidden text to each line. This hidden text will be included in the value when a list item is clicked on, but will not be displayed. Any text after a null byte (`chr(0)`) will be hidden.

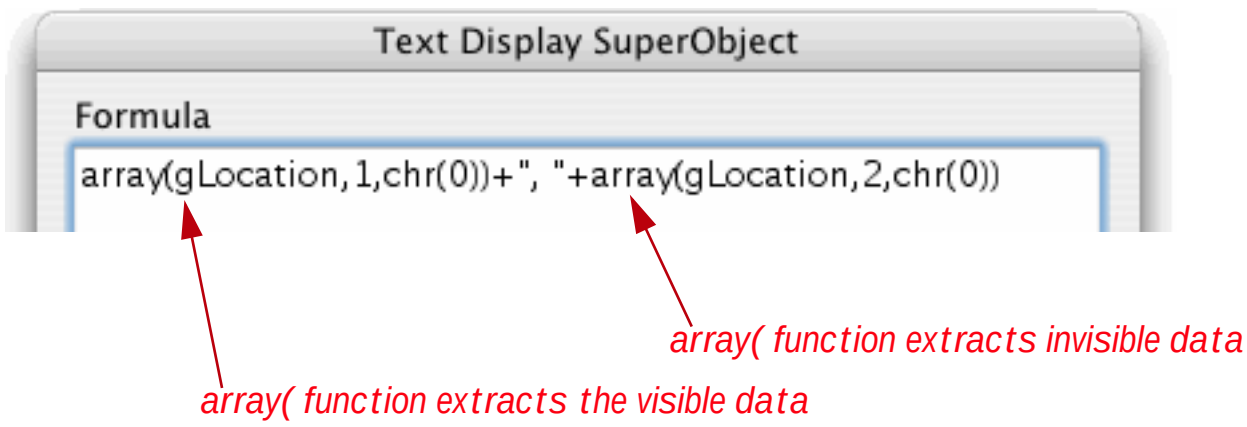
To illustrate this we'll create a list that displays only the city but actually contains both the city and state. Here's the configuration dialog for this list.



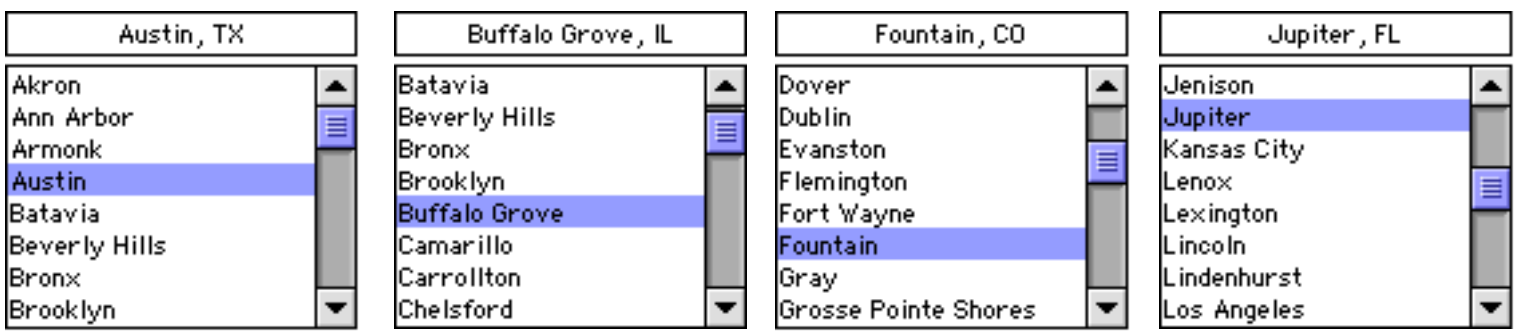
When the list is displayed only the city names are displayed.



Although the states are not displayed they are still part of the list. We can build a Text Display SuperObject to display the selected city and state. Here is the formula to be used to display the information.



Clicking on different items in the list displays both the city and the state.



An excellent use for the hidden text information is to keep secret identifying information in the list. For example a list of invoices might display only company names while keeping the invoice number hidden. When you click on an invoice the procedure can extract the hidden invoice number to display the corresponding invoice.

Maximum List Size

The List SuperObject is capable of handling lists up to a few thousand items. For large lists, you may need to increase your scratch memory allocation. However, keep in mind that the List SuperObject is not intended to replace Panorama's normal database operation. The performance of the list (speed) will degrade as the size of the list increases. Very large lists are also difficult for the user to handle. It's usually best to keep the list size to several hundred items or less. If necessary, you should split the list into multiple lists: for example by state, by price, or even by starting letter (A-Z).

Chapter 18: Form Goodies

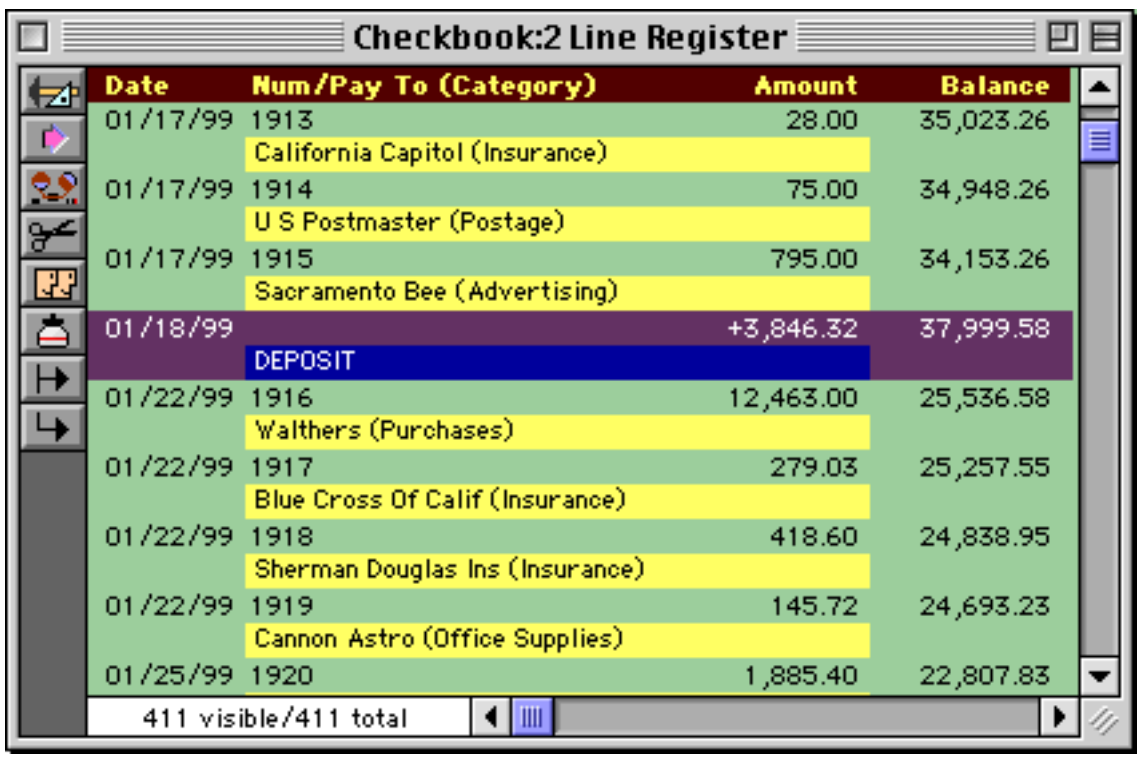


This chapter covers some cool things you can do with forms that don't fit into any particular category. **View-as-List** forms allow you to display multiple records at a time, instead of just a single record at a time like a normal form. **Elastic Forms** allow you to create forms that automatically stretch to fit any window size (see "Elastic Forms" on page 912). **Super Matrix** objects allow you to quickly build repeating arrays like monthly calendars and photo arrays (see "Super Matrix Objects" on page 929). **Scroll Bar SuperObjects** can be used to create standalone scroll bars within a form (see "Scroll Bars" on page 968). **Balloon Help Objects** (Macintosh only) allow you to add detailed help to your forms (see "Balloon Help" on page 974).

View-As-List Forms

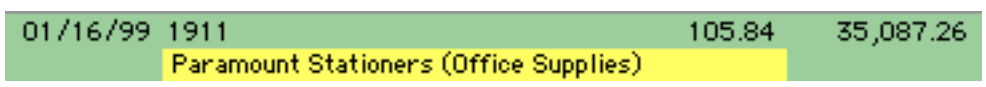
Panorama allows you to set up blank forms as individual pages or as a continuous sheet (view-as-list). When forms are set up as individual pages you see one record at a time. You can flip through the records just as you would shuffle through a stack of paper forms.

A view-as-list form displays data as a continuous sheet, as shown below. Instead of flipping from record to record, you scroll up and down through the data in a manner similar to the data sheet. However, unlike the data sheet, a view-as-list form allows you to arrange the data any way you like, and even include graphics in the display. On the other hand, view-as-list forms are slower than the data sheet (because of the overhead in displaying the graphics) and they are much more work to set up.

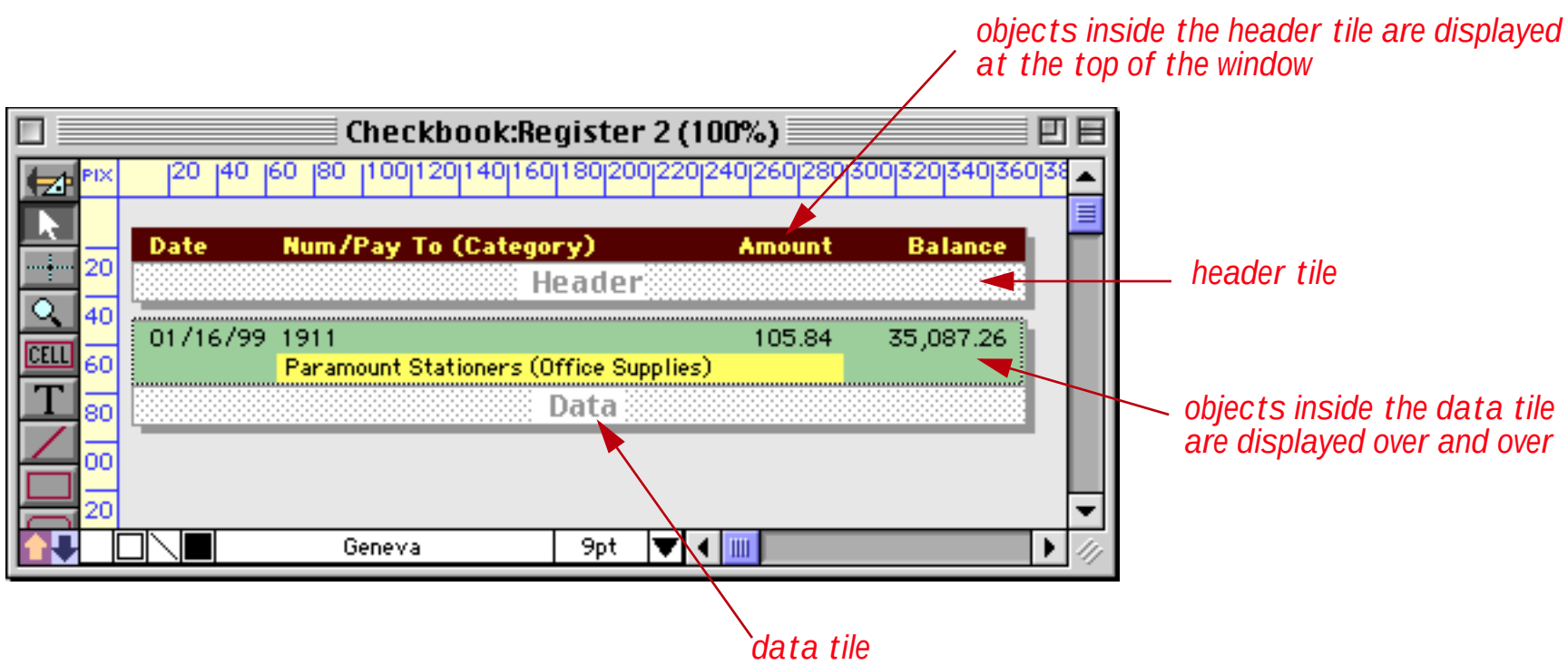


How View-As-List Forms Work

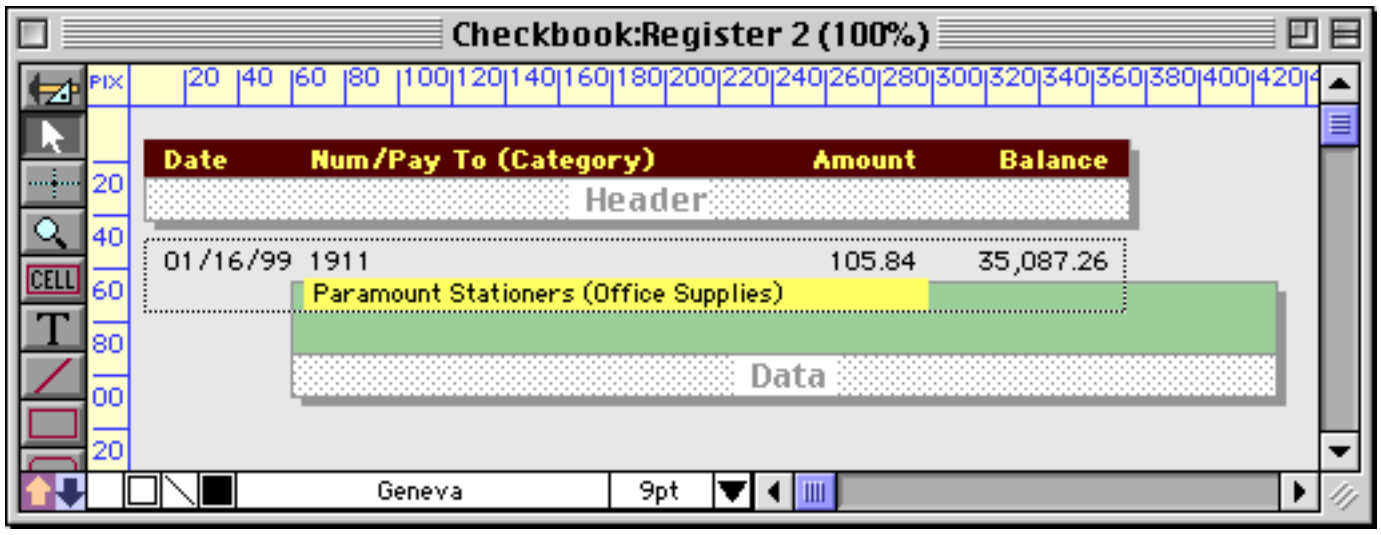
Panorama displays a View-As-List form by taking a collection of graphic objects and displaying them over and over again, once for each record. For the form in the previous section the collection of graphic objects looks like this.



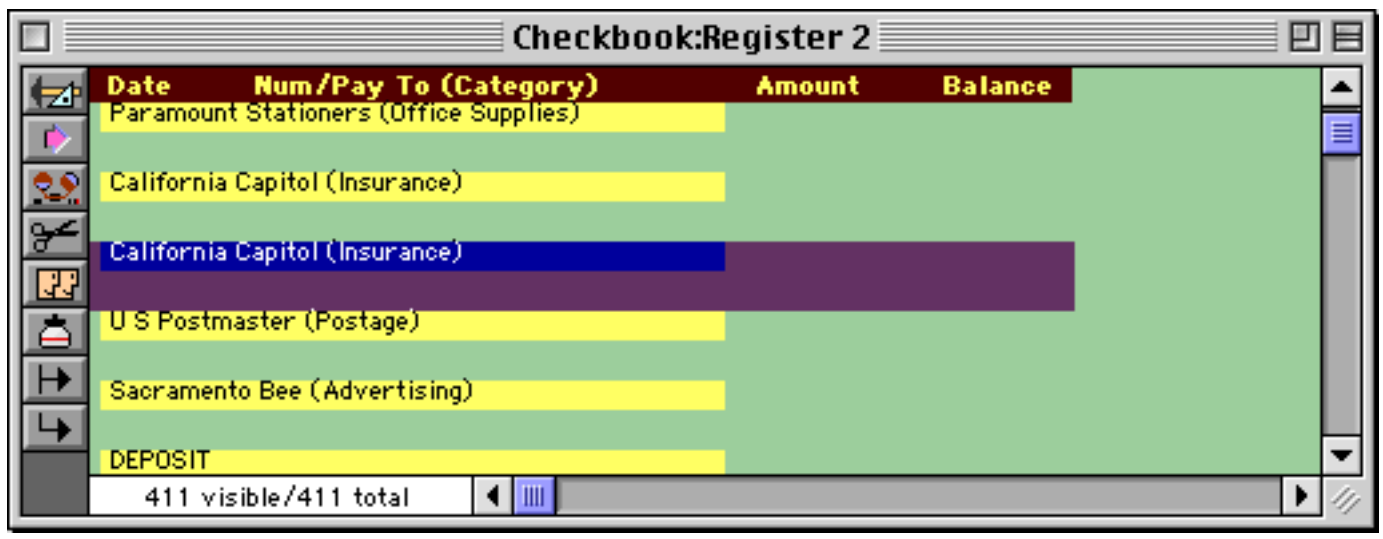
As you can see, this collection of objects is just a small portion of a form. How does Panorama know which collection of objects to include? You tell it by enclosing the objects in a special object called a **tile**. Panorama has over 40 different kinds of tiles, but for now we are interested in just two types—**data tiles** and **header tiles**.



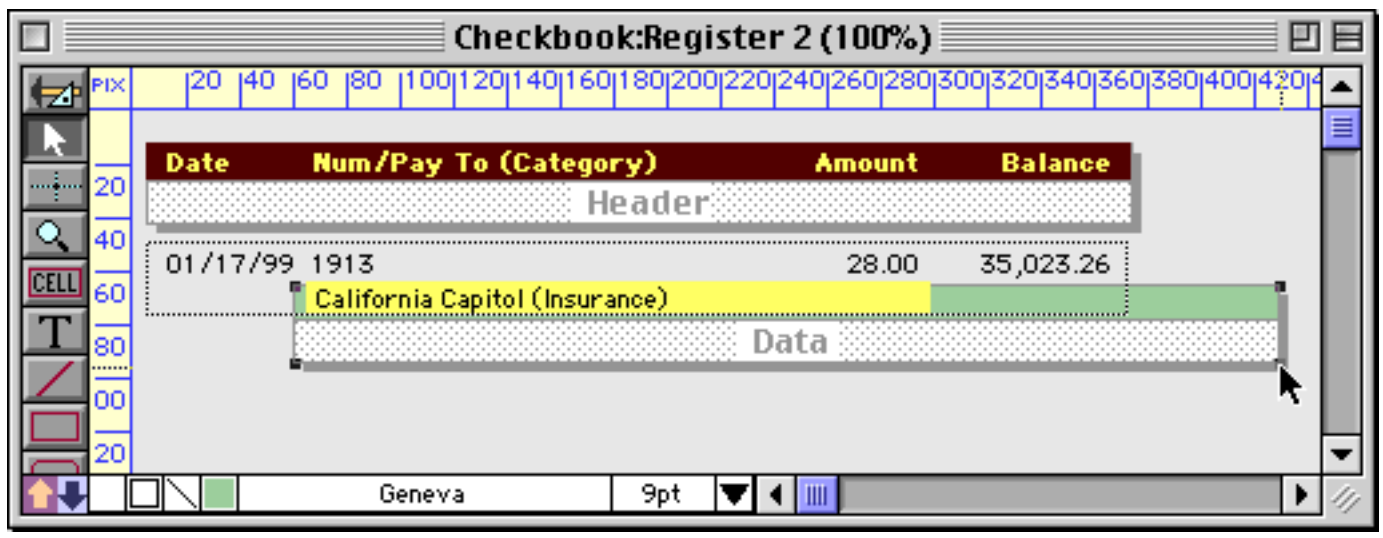
Panorama doesn't care what the exact location of the tile is. It simply looks to see what is contained within the tile. For example, you could move the tile to a new position like this (see "Working with Tiles" on page 1050).



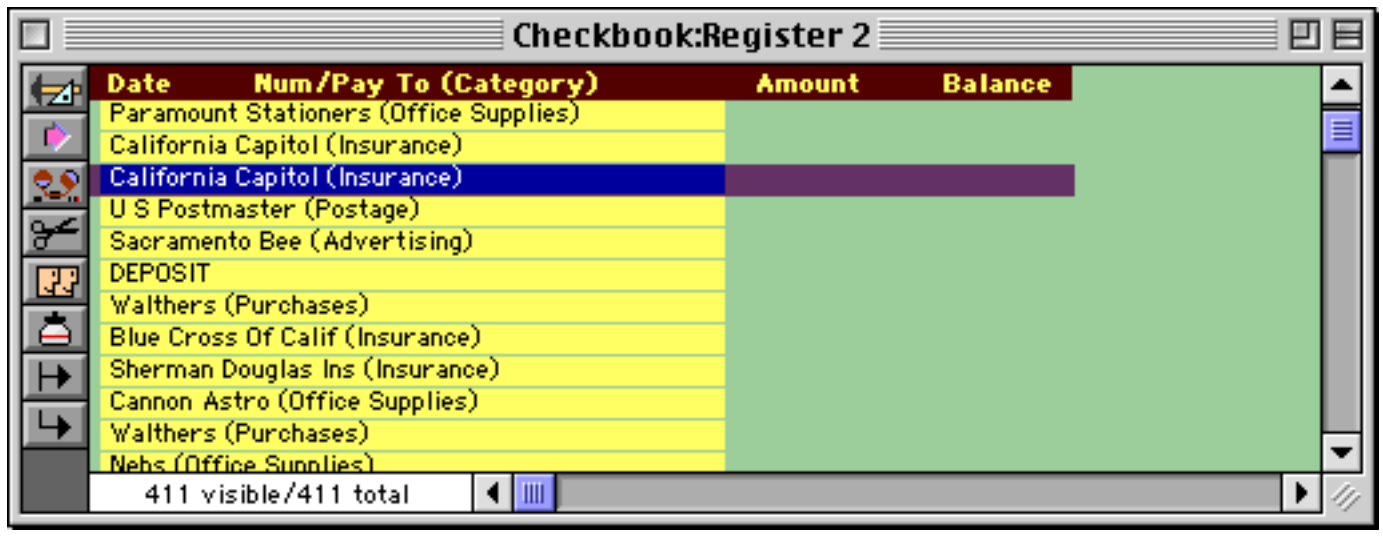
In Data Access Mode this new configuration will look like this.



The height of the Data Tile controls the spacing of the records. For example, we can reduce the spacing by reducing the height of the tile (see “Working with Tiles” on page 1050).

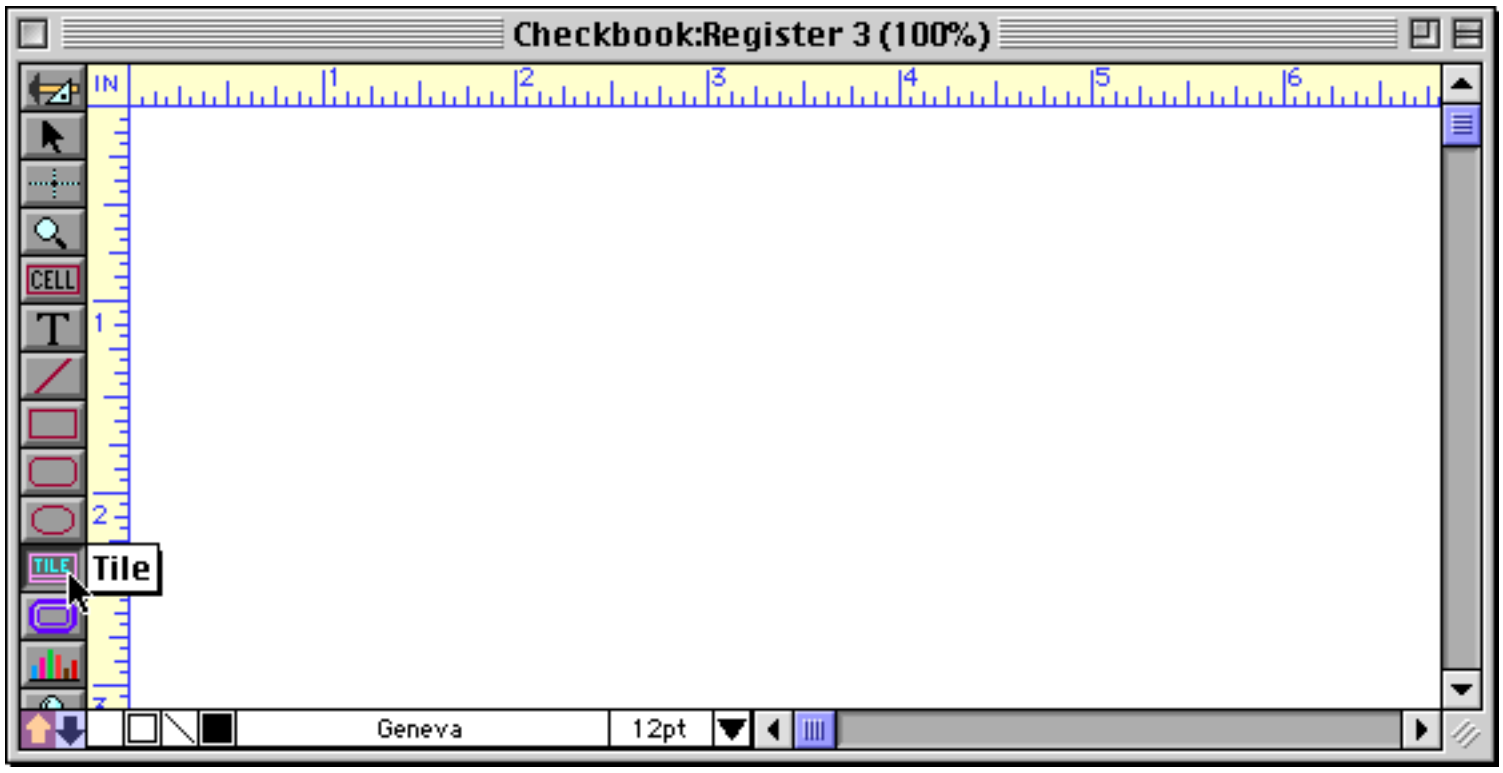


Here’s what this configuration looks like.

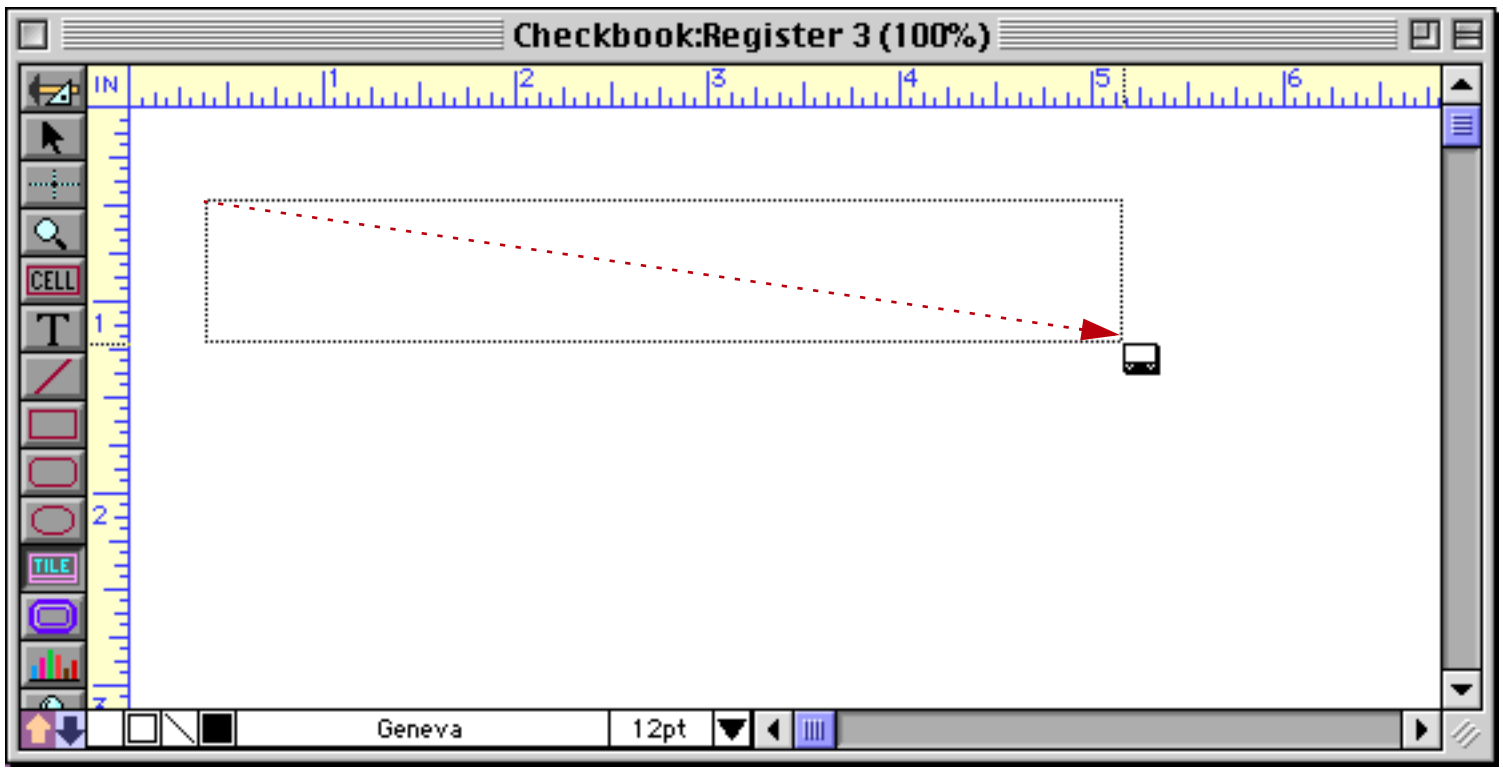


Creating a View-As-List Form

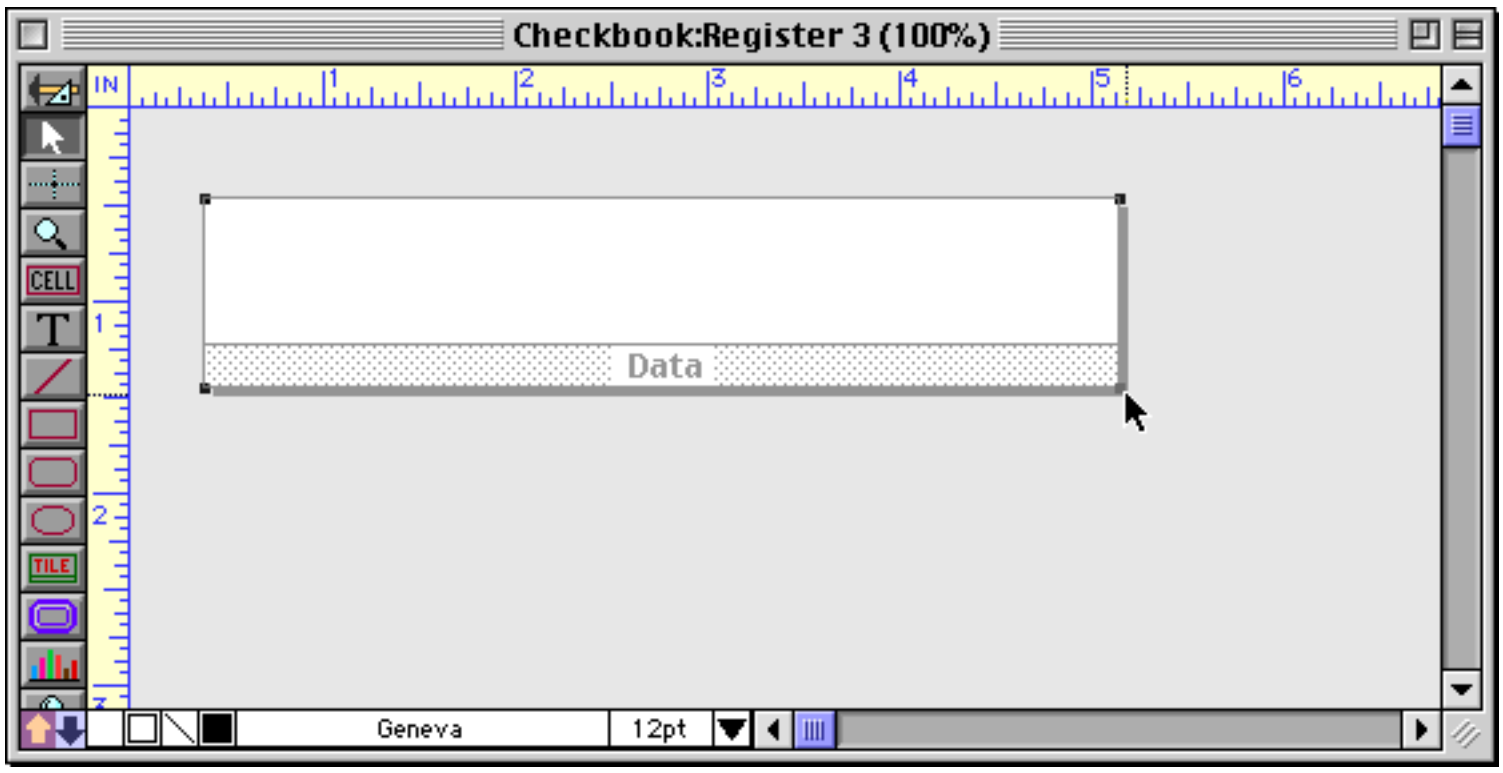
The first step in creating a view-as-list form is to create a Data Tile. To do this, select the **Tile** tool.



Now drag the mouse across the form to define the location and size of the tile.

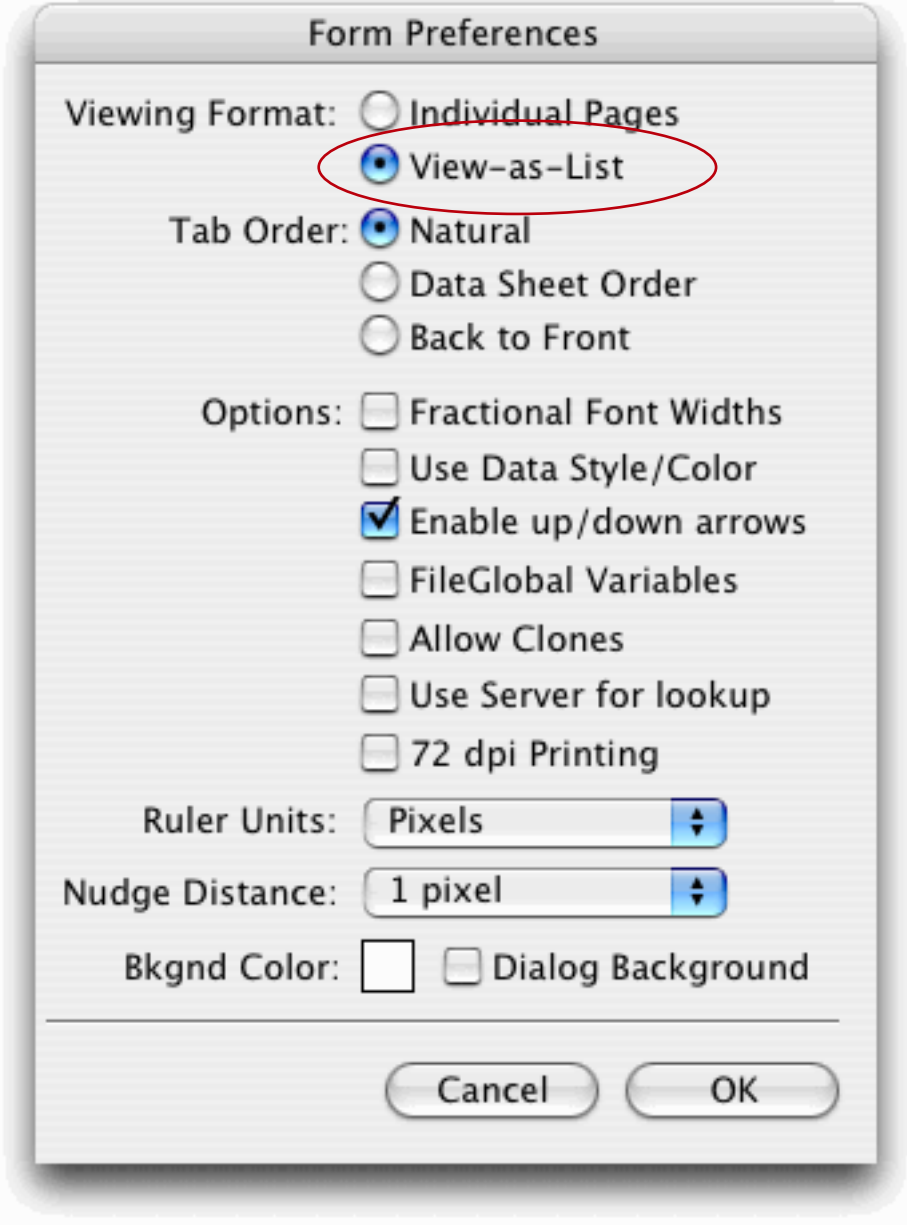


When you release the mouse Panorama will create the tile.

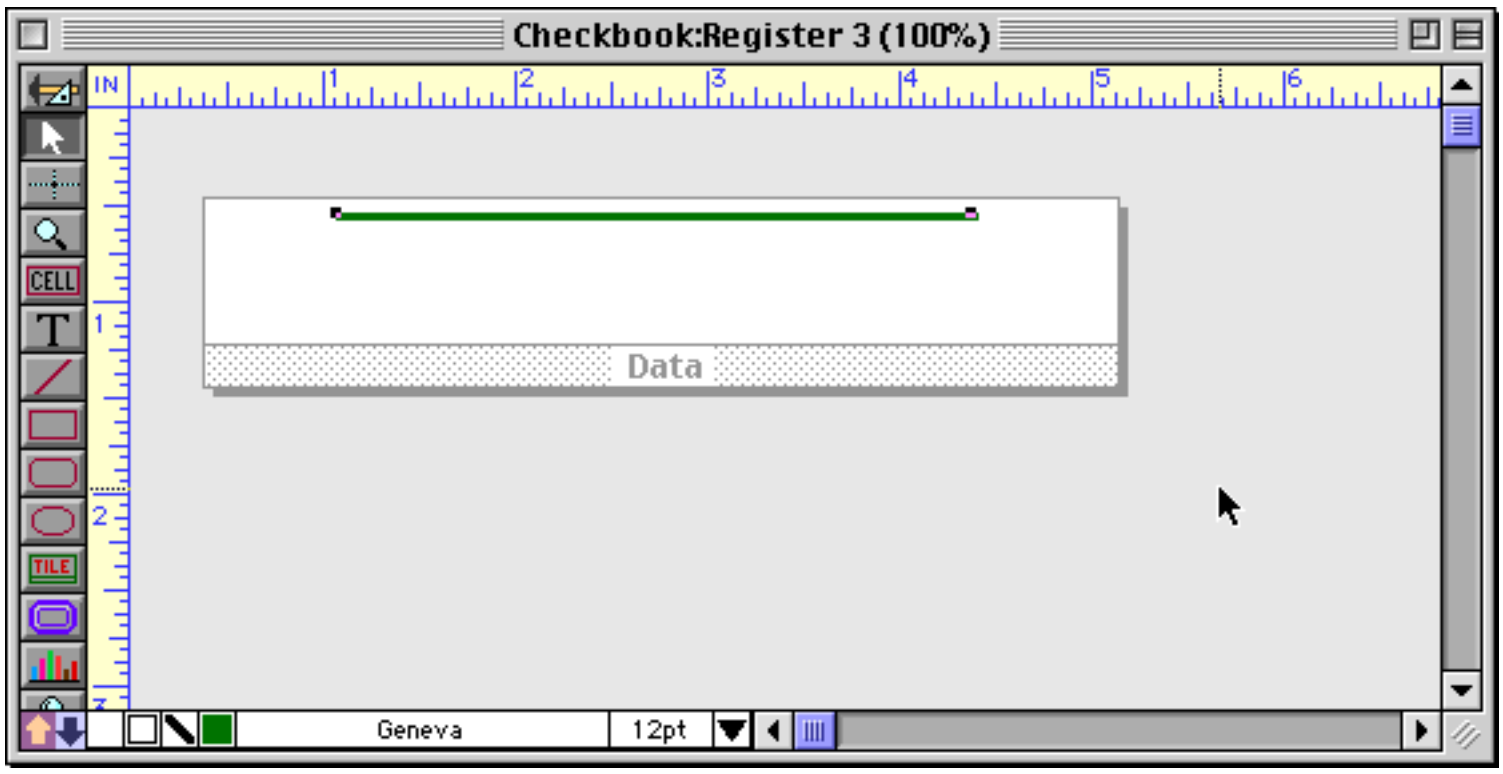


As you can see, the area outside of the form turns gray. This indicates that any object placed in this area will not be included as part of the form when in Data Access Mode.

The next step is to enable the View-As-List option in the **Form Preferences** dialog. You'll find this dialog in the Setup menu.

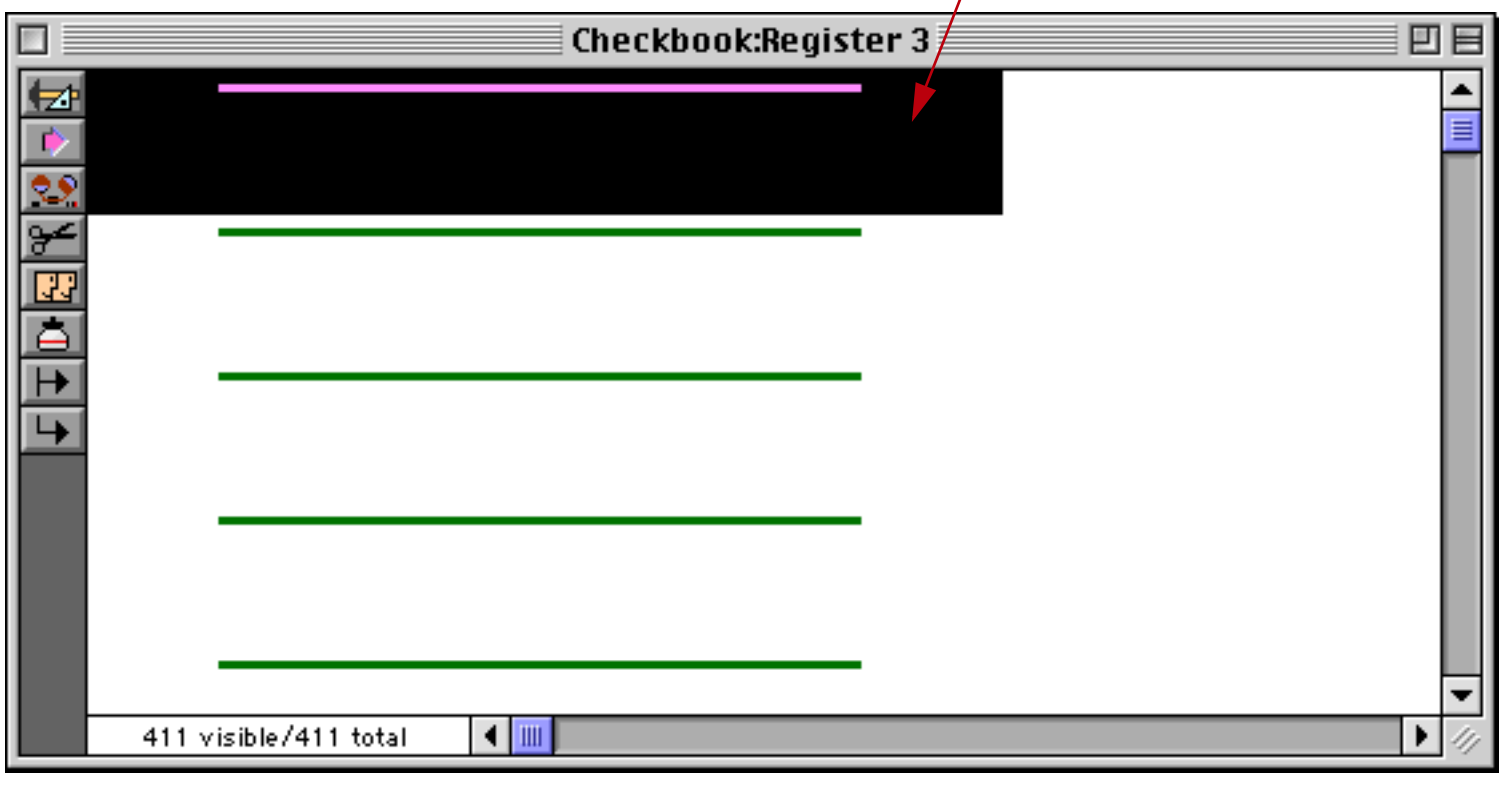


Press the OK button to confirm the new preferences. Now we're ready to start adding graphics to the form. We'll start by using the **Line** tool to add a horizontal line across the top of each record (see "Creating a Graphic Object" on page 494).



Switch to Data Access Mode to see what this configuration looks like. The line is repeated once for each record. (You'll also notice that the current record is highlighted in reverse - see "Buttons on a View-As-List Form" on page 911 for more information about this.)

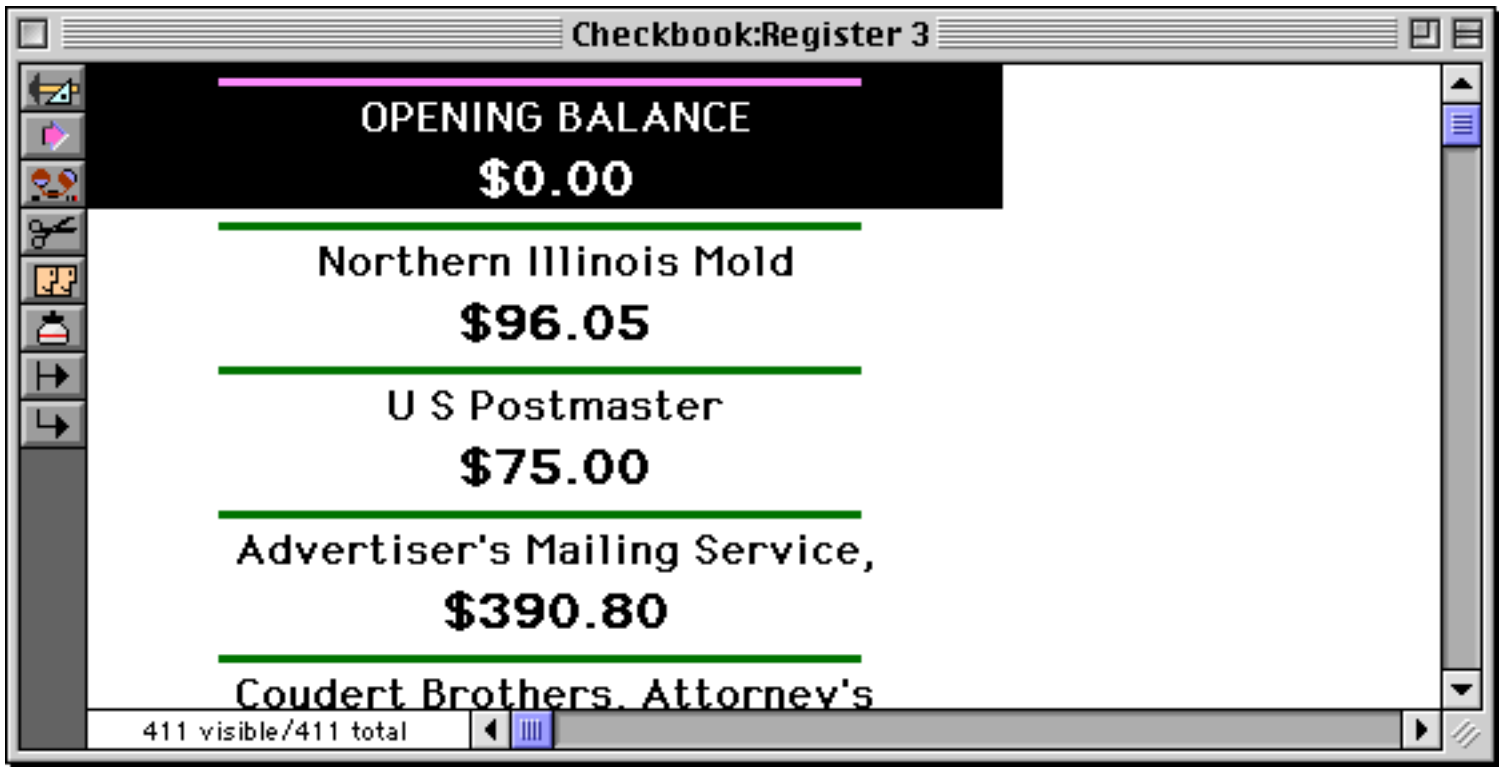
current record is highlighted



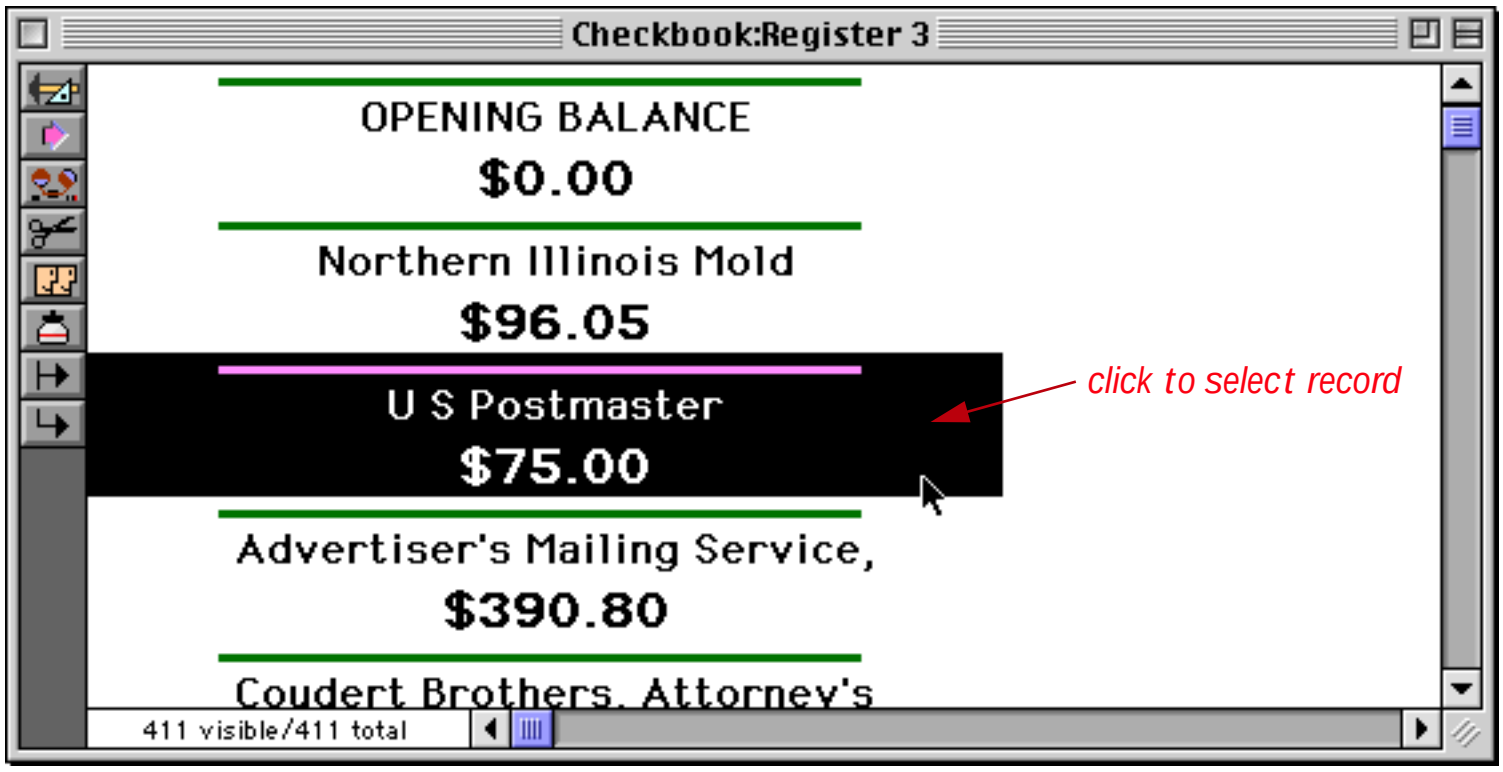
Got the idea? Now we can go back to Graphics Mode and add more objects. In this example we used auto-wrap text objects to display two fields from the database (see “Displaying Data in Auto-Wrap Text” on page 595). You can also use Data Cells (see “Working with Data Cell Objects” on page 635), Text Editor SuperObjects (see “Text Editor SuperObject” on page 639), Text Display SuperObjects (see “Text Display SuperObjects™” on page 608) or even Flash Art (see “Flash Art™” on page 748).



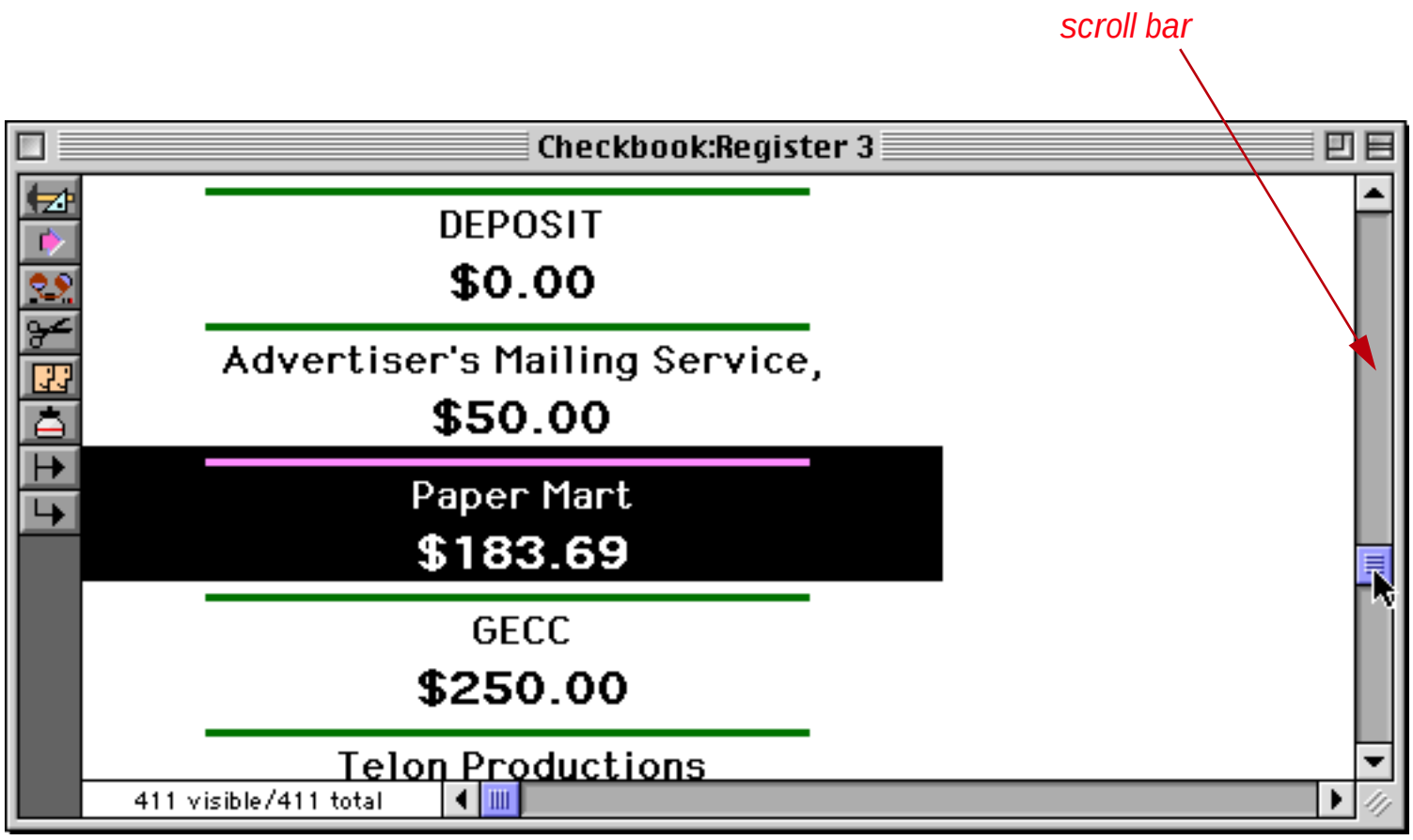
You can switch to Data Access Mode at any time to check out your work.



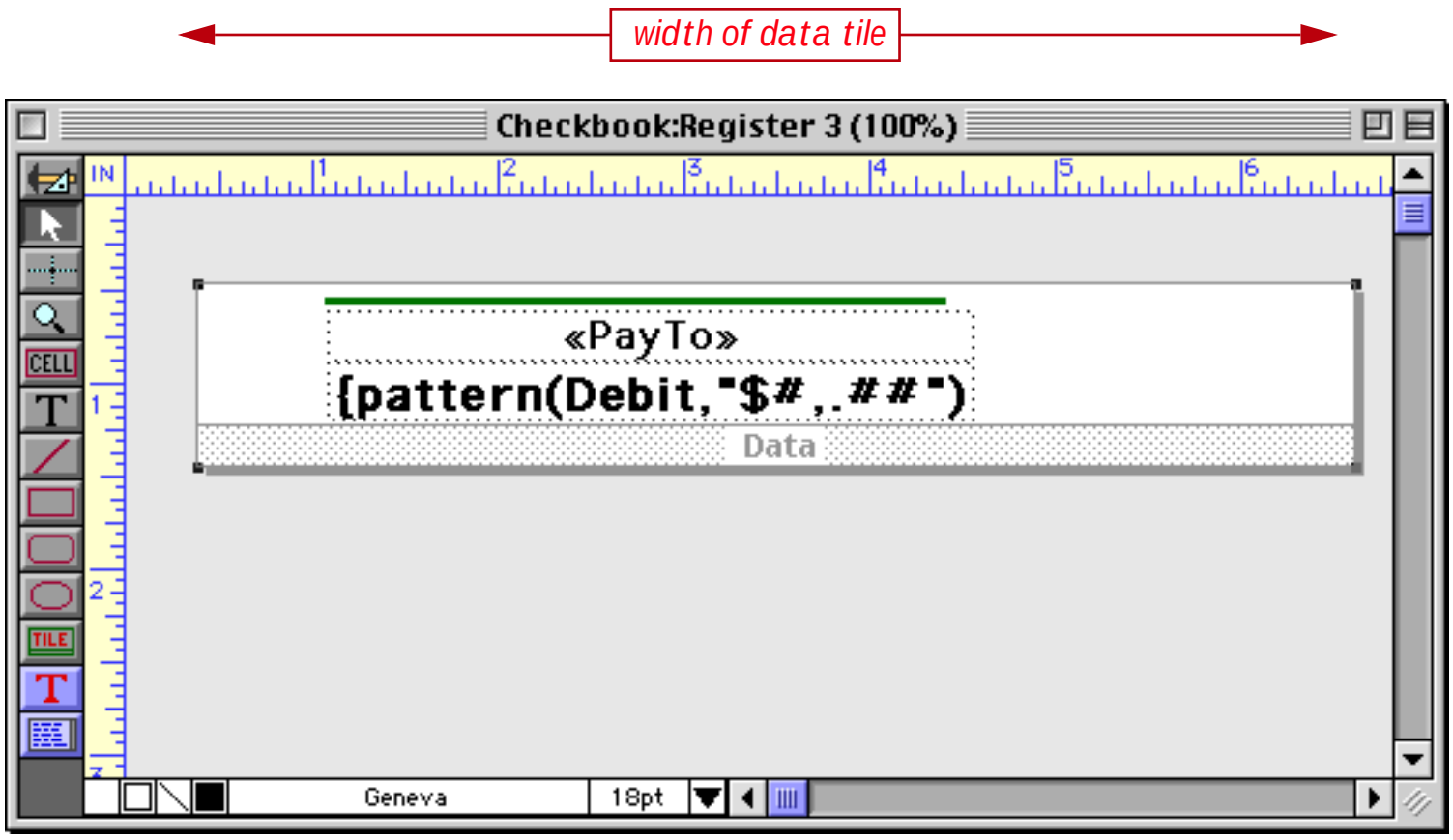
The view-as-list form kind of works like the data sheet. You can select a different record by clicking on it.



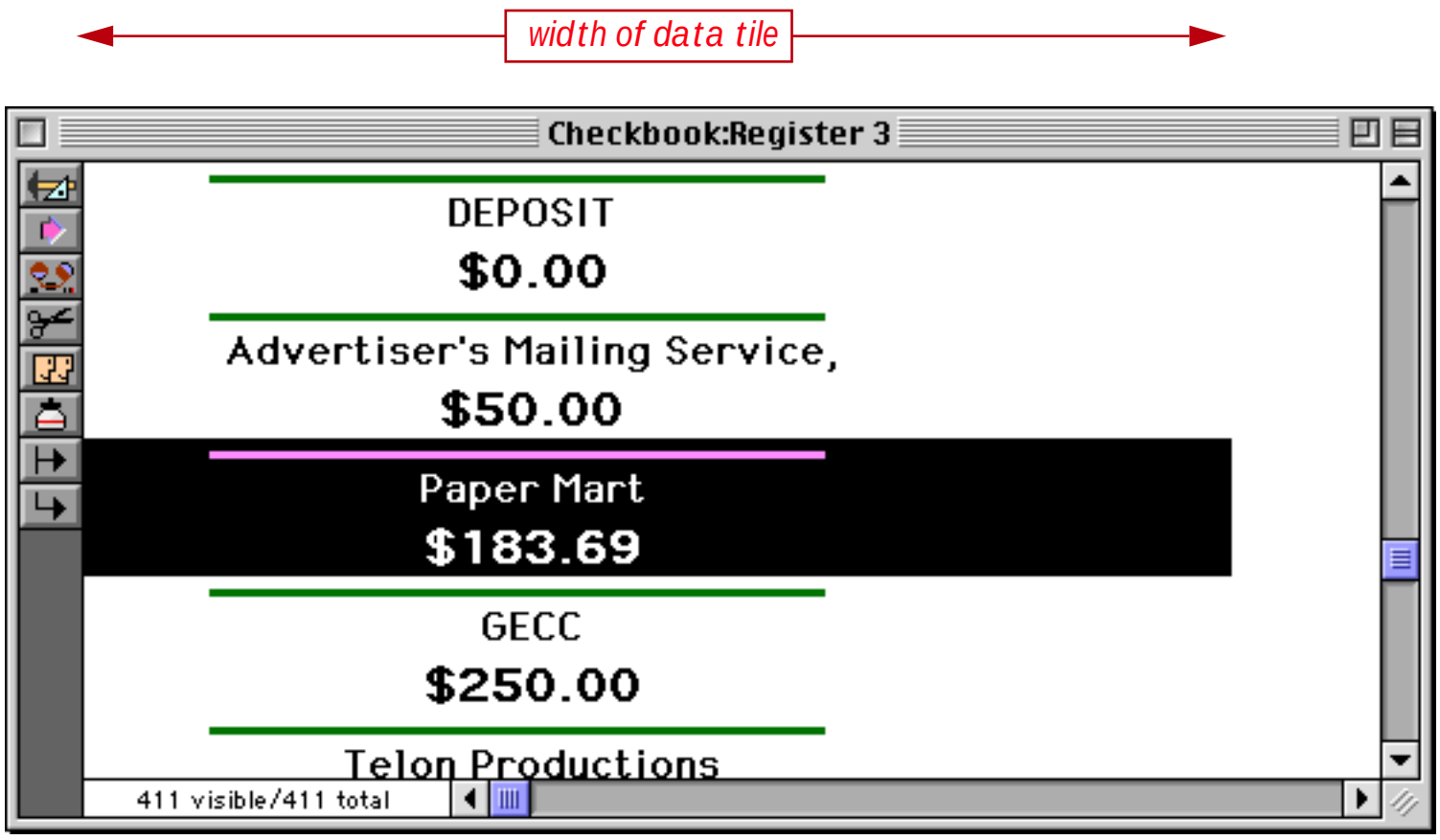
Or you can use the vertical scroll bar to navigate to any location within the database.



The width of the current record's highlighting corresponds to the width of the data cell. By changing the width of the data cell you can change the width of the highlight.



In Data Access Mode you can see the new, revised highlight width.



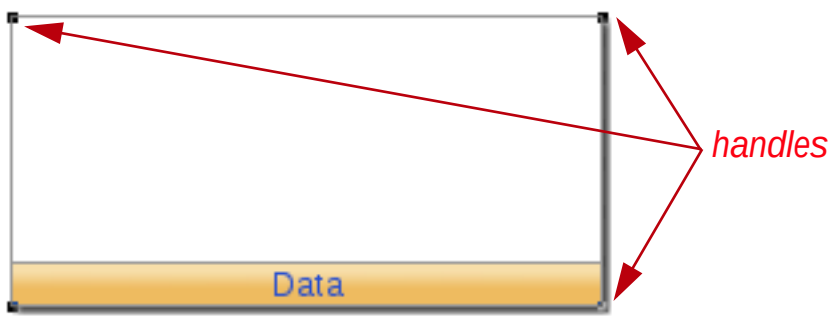
Since this form doesn't contain any Text Editor SuperObjects or Data Cells you cannot edit the data using this form. See "Editable View-As-List Forms" on page 903 to learn how to create an editable view-as-list form.

Working with Tiles

Like all other graphic objects, tiles can be moved and resized with the **Pointer** tool. However, tiles are slightly different than other objects. On the screen, a tile looks similar to an upside down window. Tiles are divided into two parts: the surface and the drag bar.



Unlike other graphic objects that can be manipulated by clicking anywhere in the object, a tile is only sensitive to clicks on its drag bar. To move a tile, press the mouse on the drag bar and drag the tile to the new position. To select a tile, click on the drag bar. When the tile is selected, four handles appear around the corners of the tile.



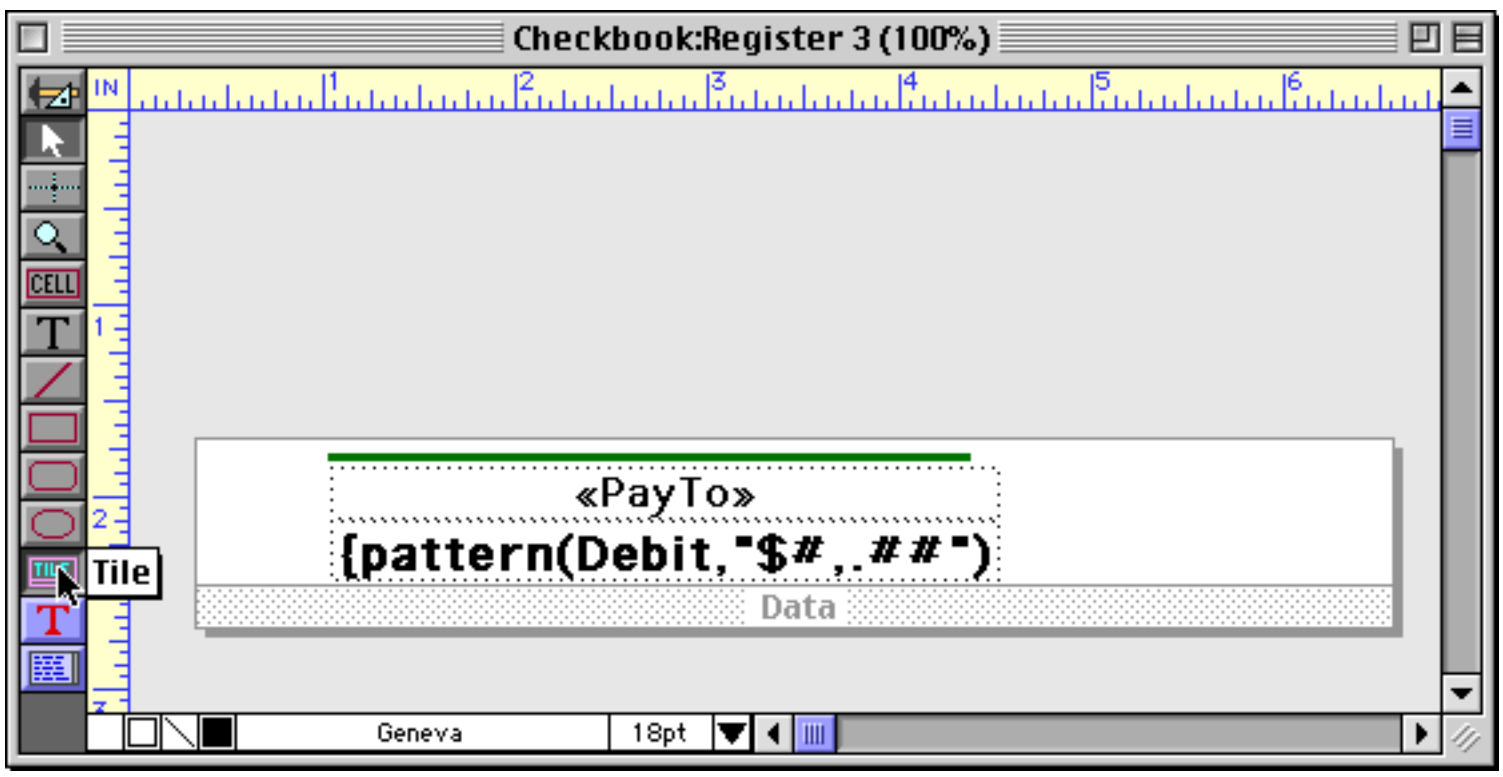
You can use these handles as grips to change the size of the tile (see “Changing the Size of a Single Object” on page 513). Tiles can also be moved or resized with the **Dimensions** command (see “Viewing and Setting Exact Object Dimensions” on page 512).

The surface of the tile is not sensitive to the mouse. In other words, clicking on the surface area does not select the tile, and you cannot move the tile by dragging on the surface.

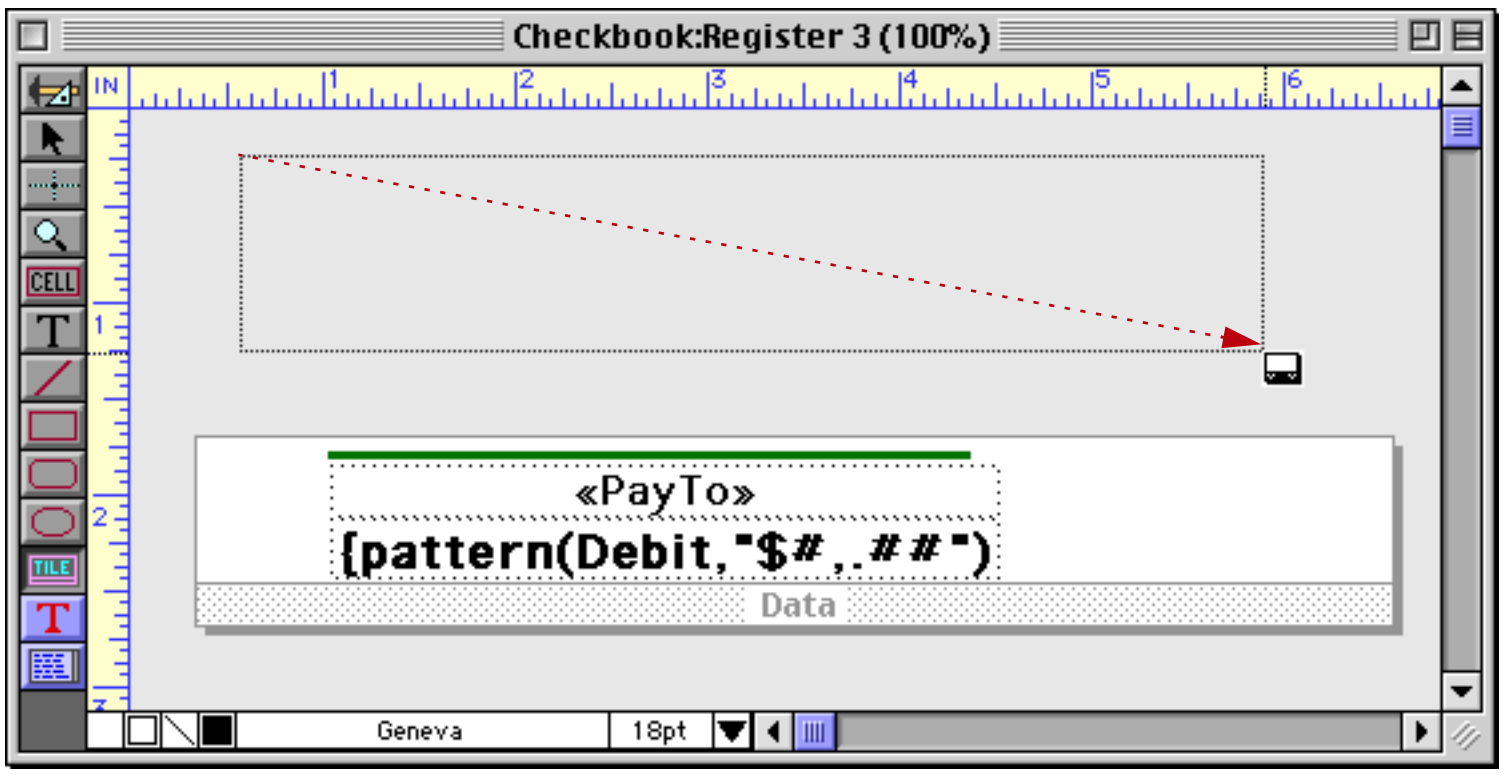
Adding a View-As-List Header

To add a header to your view-as-list form you'll need to add a second tile. Surprisingly enough this is called a **header tile**. There are two methods for creating this tile—you can create it from scratch or you can create it from a copy of the data tile.

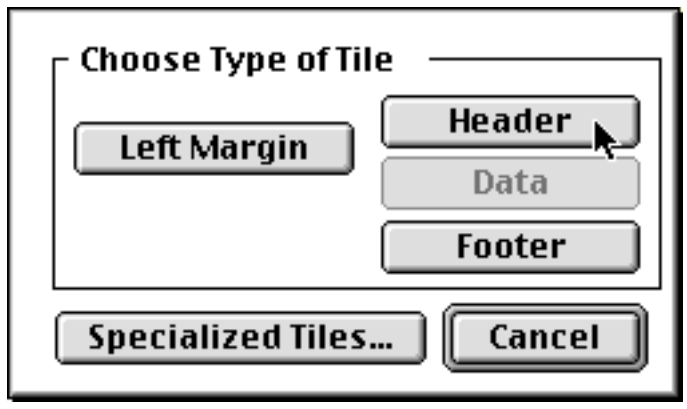
To create a header tile from scratch start by selecting the **Tile** tool.



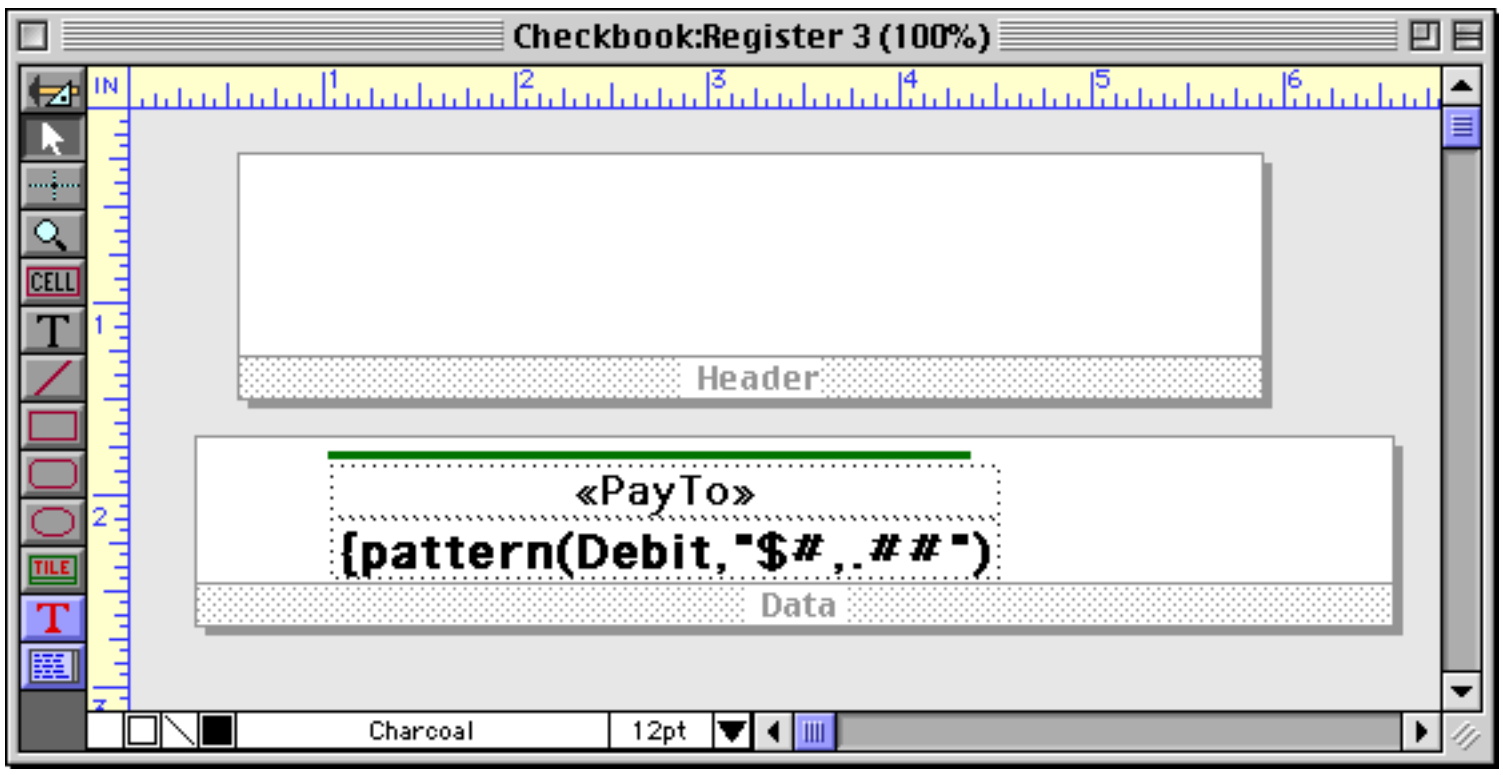
Now drag the mouse across the form where you want the tile to appear. The header tile does not have to line up with the data tile, although that can make it easier to visualize the final result.



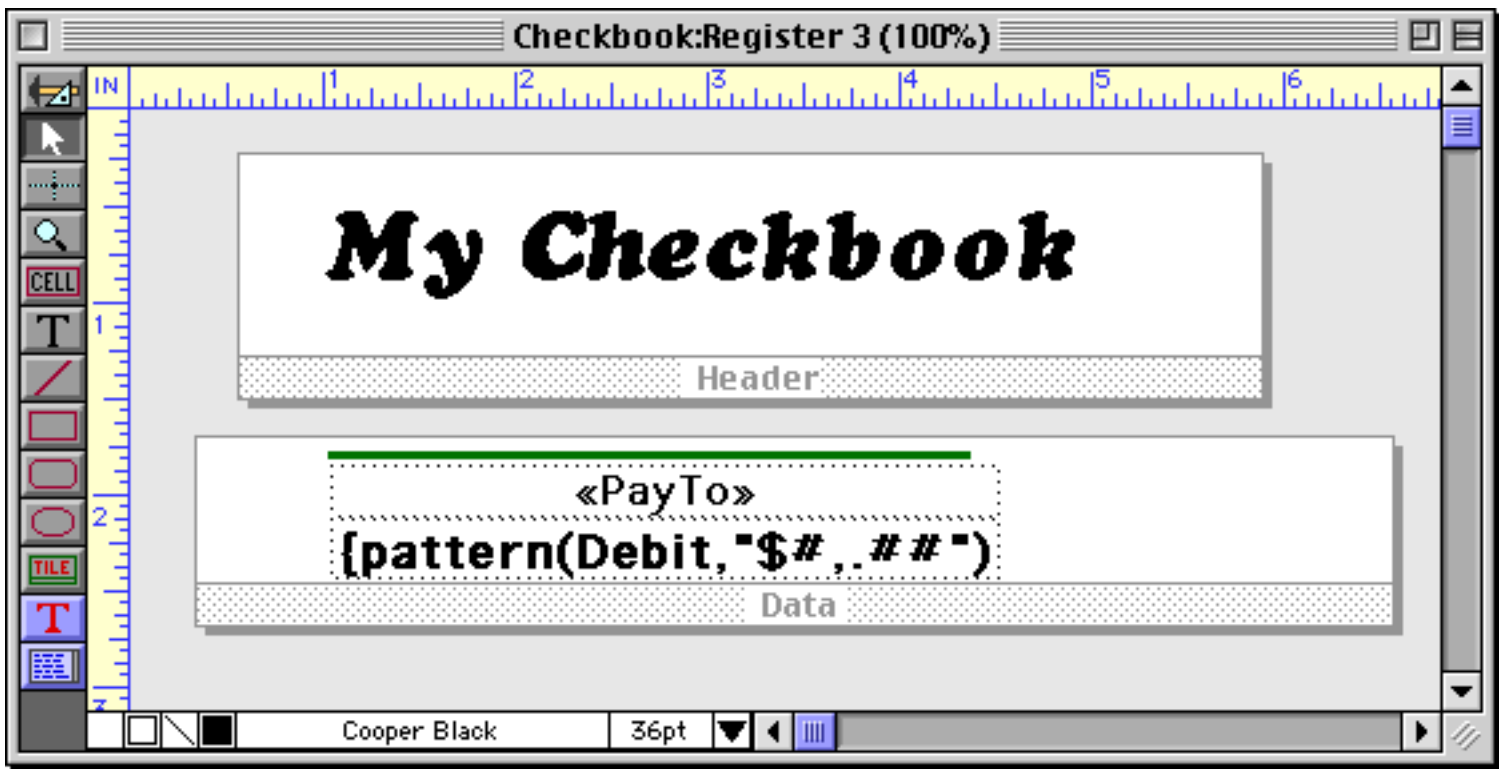
When you release the mouse a tile configuration dialog appears. This dialog allows you to select the type of tile you are creating.



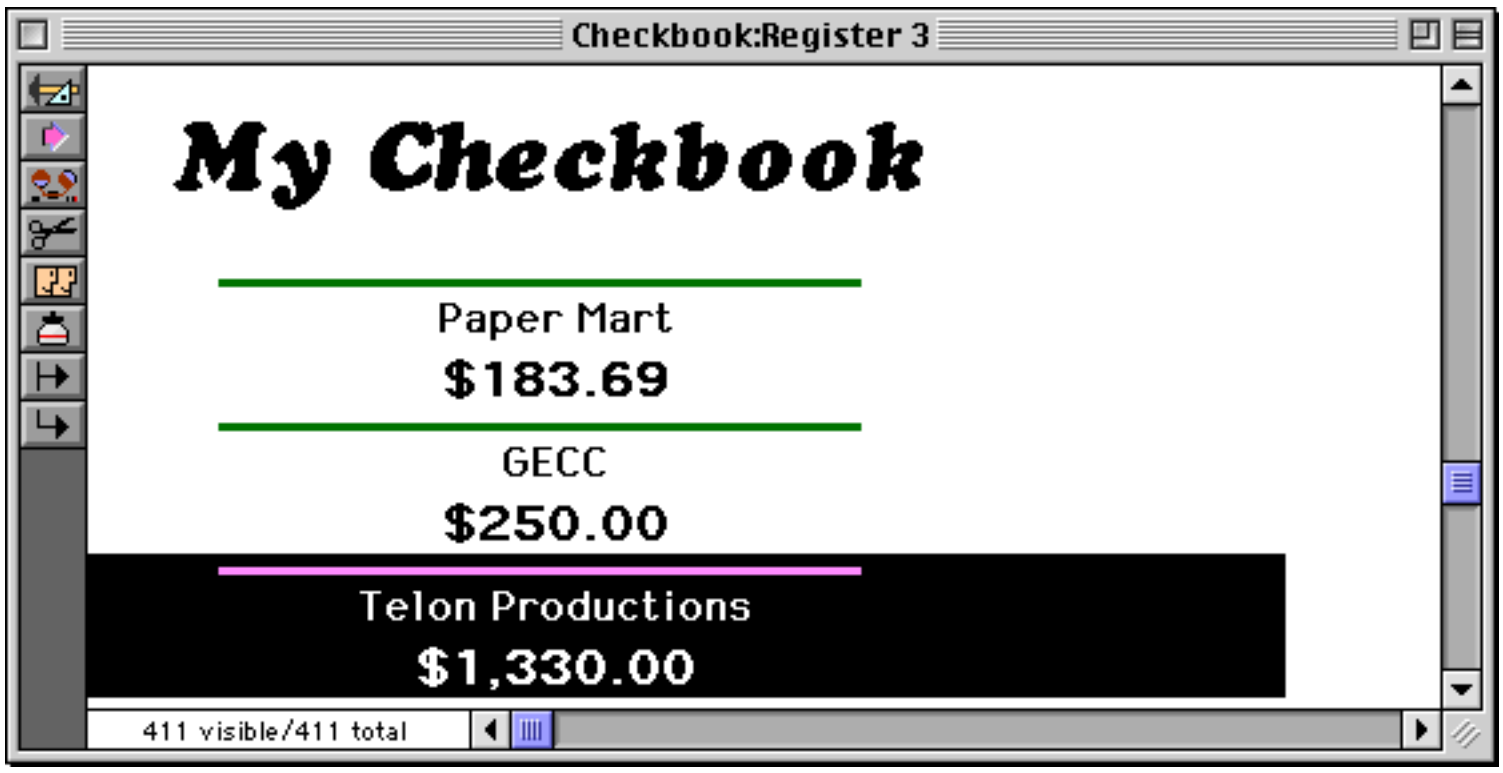
Press the **Header** button. When you press the button, Panorama creates the new tile.



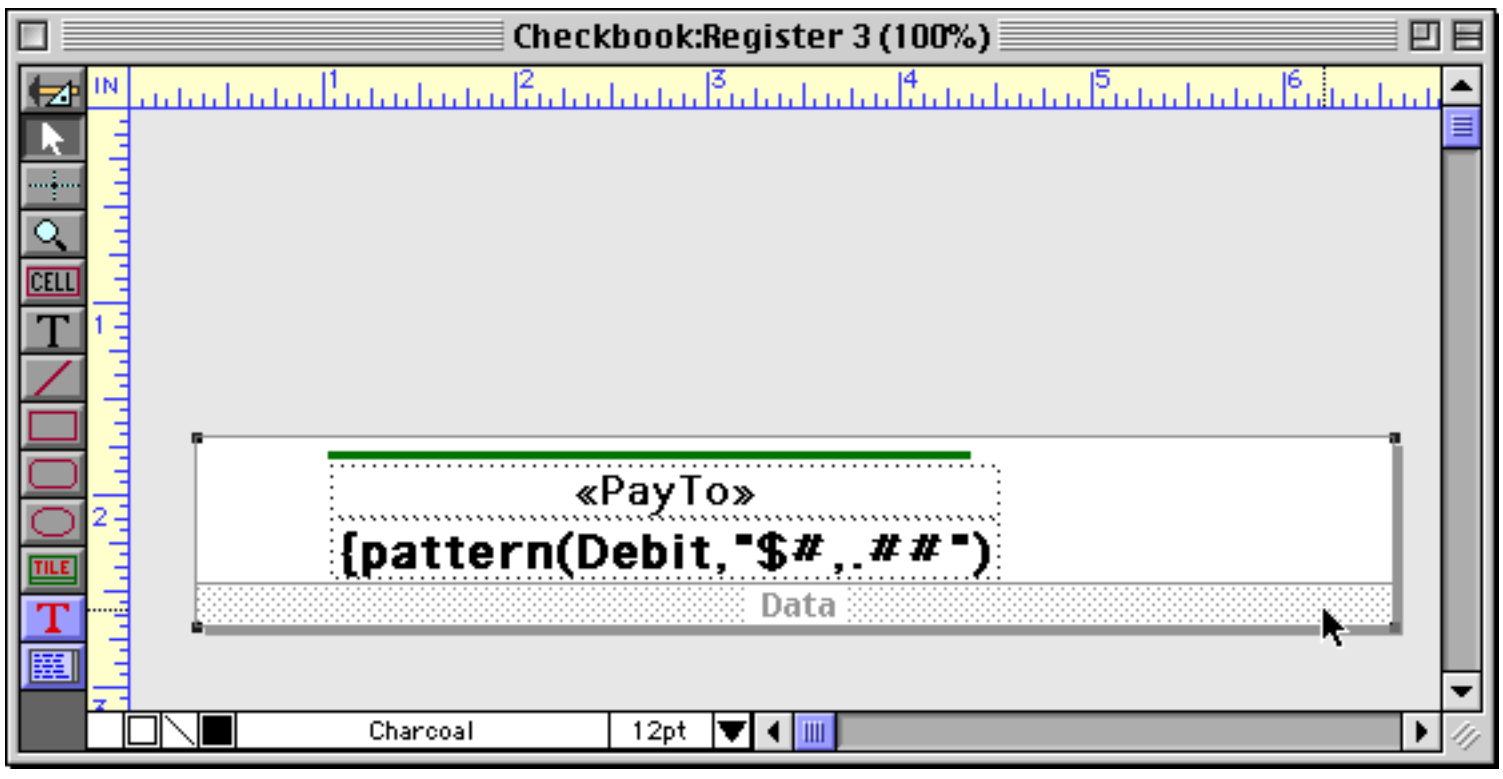
Now you should add any graphics and text that you want to appear on the header.



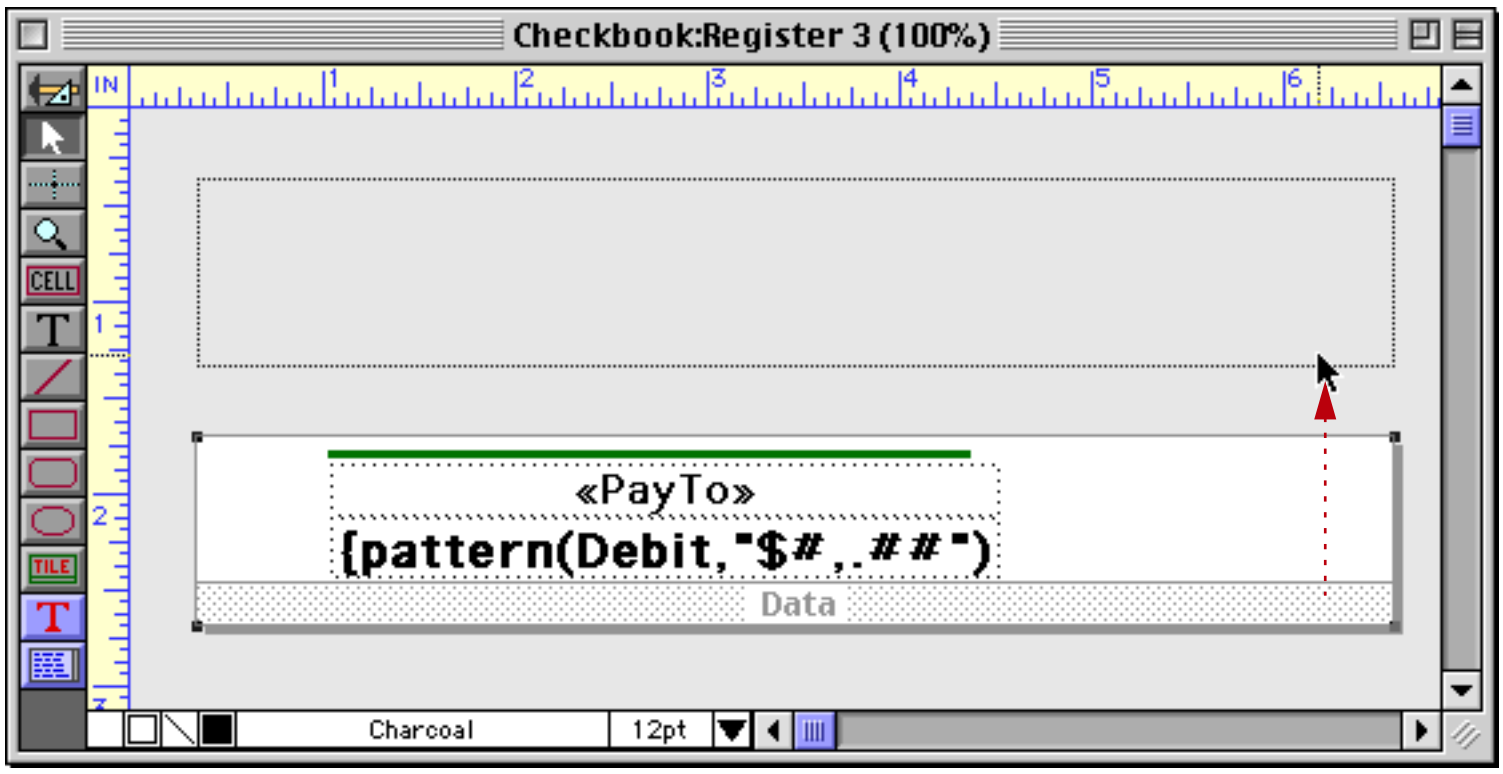
Switch to Data Access Mode to see the finished result.



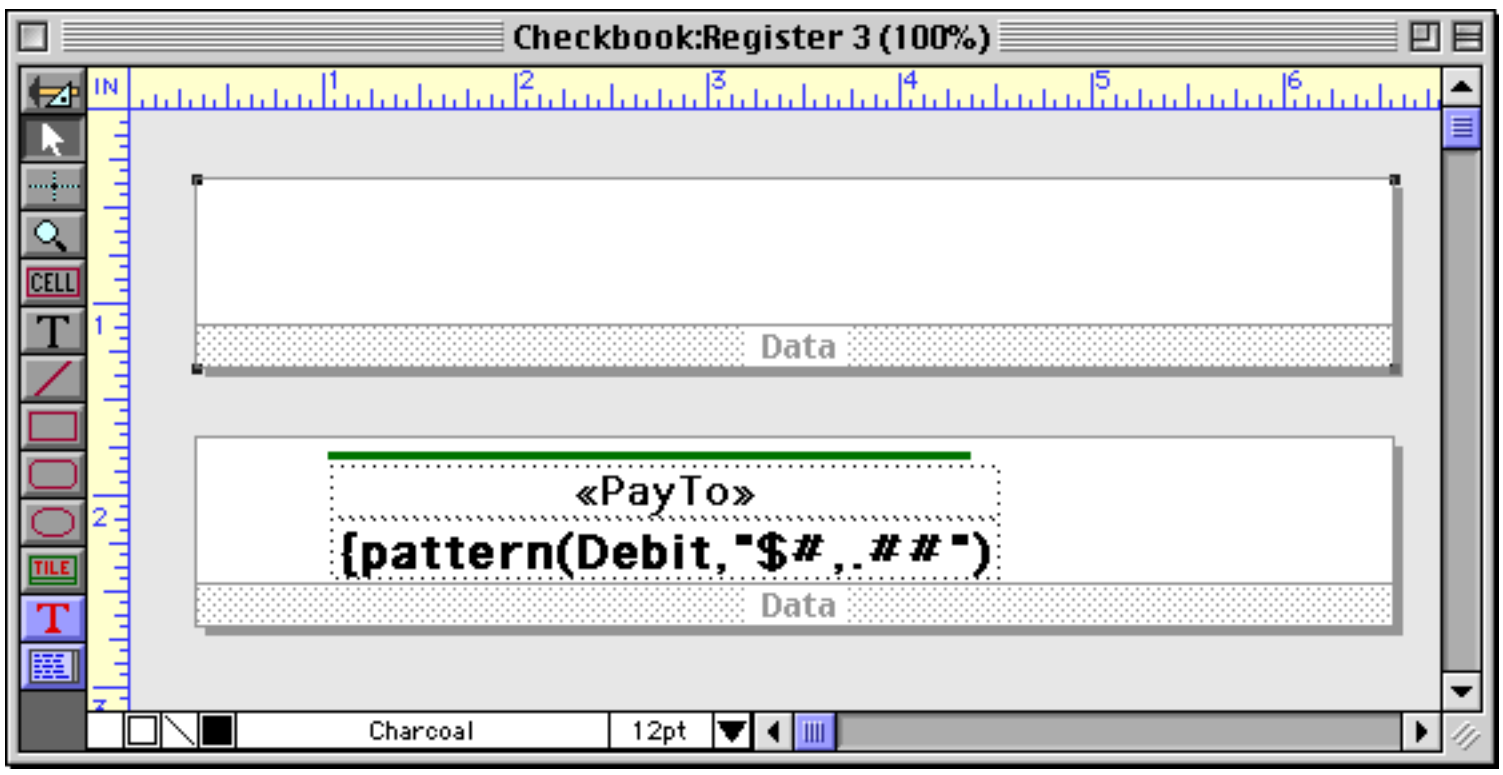
Now let's create a header tile by duplicating the data tile. Start with just the data tile.



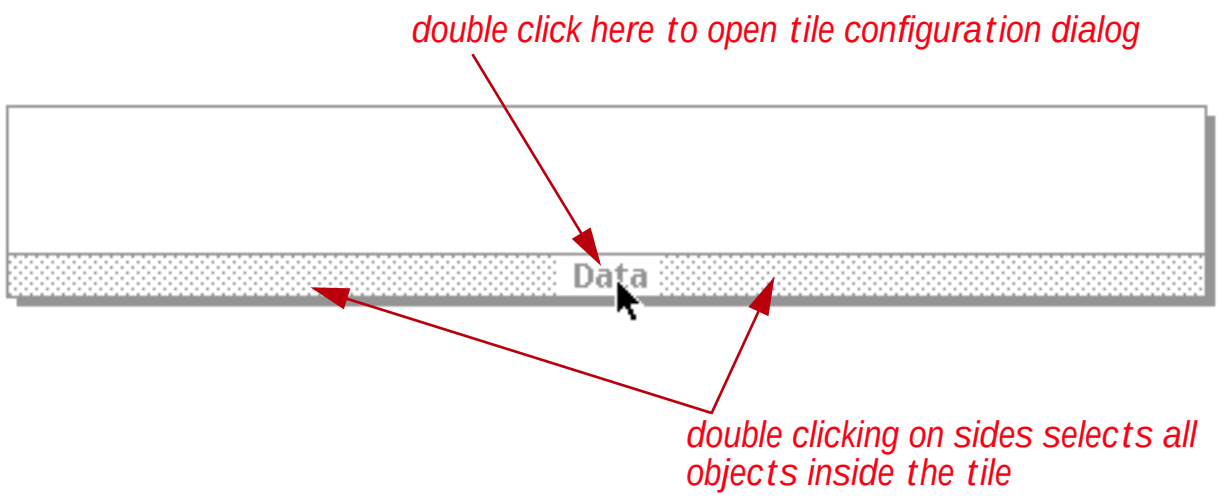
To duplicate the tile, hold down the **Option** key (Mac) or **Alt** key (Windows) and drag the tile (see "Drag Duplicating" on page 561). You may also want to hold down the **Shift** key at the same time to make sure that the two tiles stay in perfect alignment.



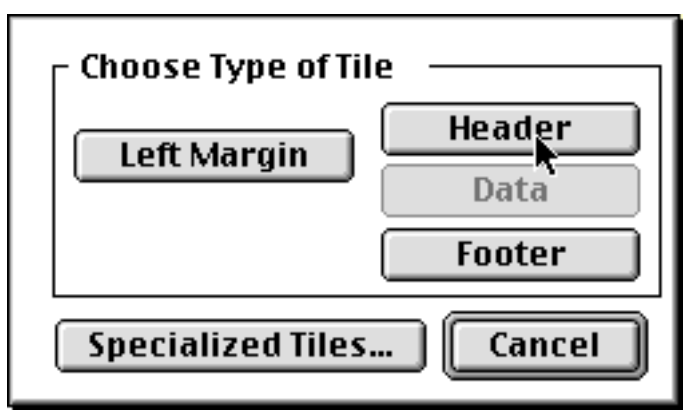
When you release the mouse your form will contain two data tiles.



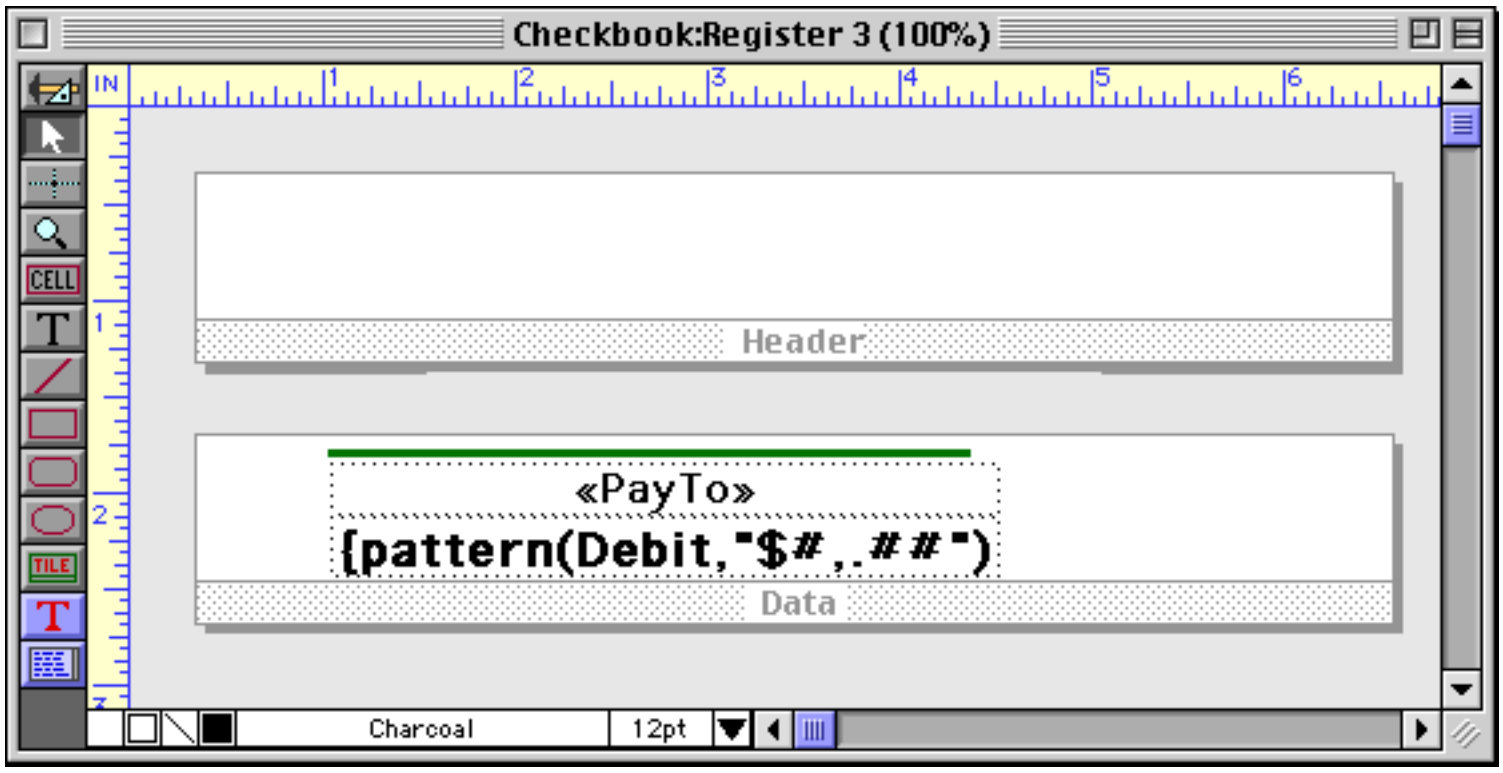
To transform the new data tile into a header tile, double click on the word **Data**.



This opens the tile's configuration dialog.



Press **Header** to convert the data tile into a header tile.

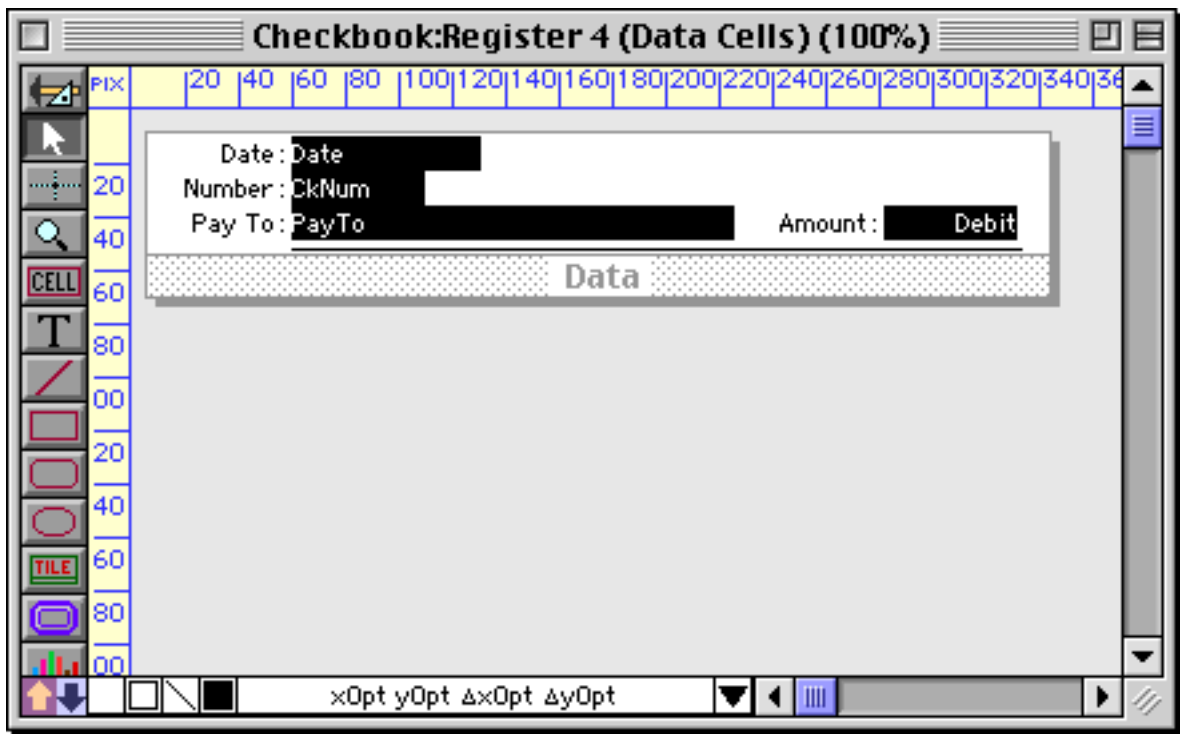


Voila! Now you can add any graphics and text you want to add to the header, and your form will be complete.

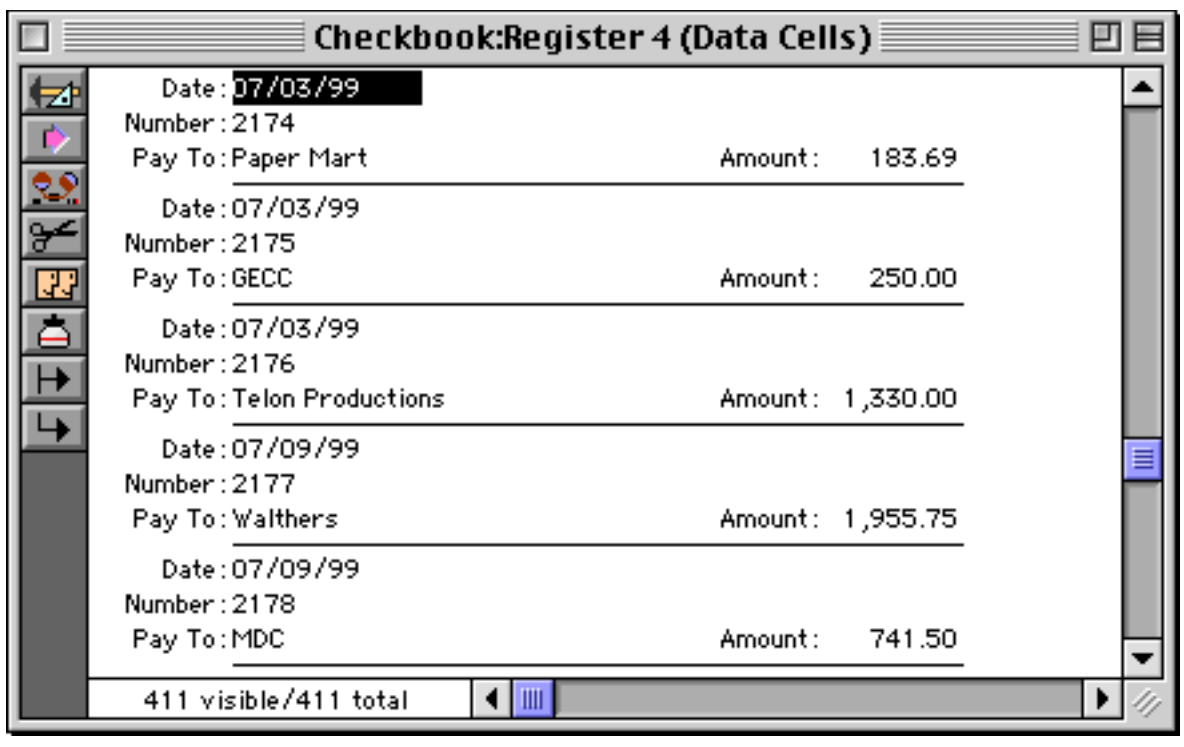
Editable View-As-List Forms

Adding data cells or Text Editor SuperObjects to the data tile makes it possible to edit the database using the view-as-list forms. The operation of each is a bit different.

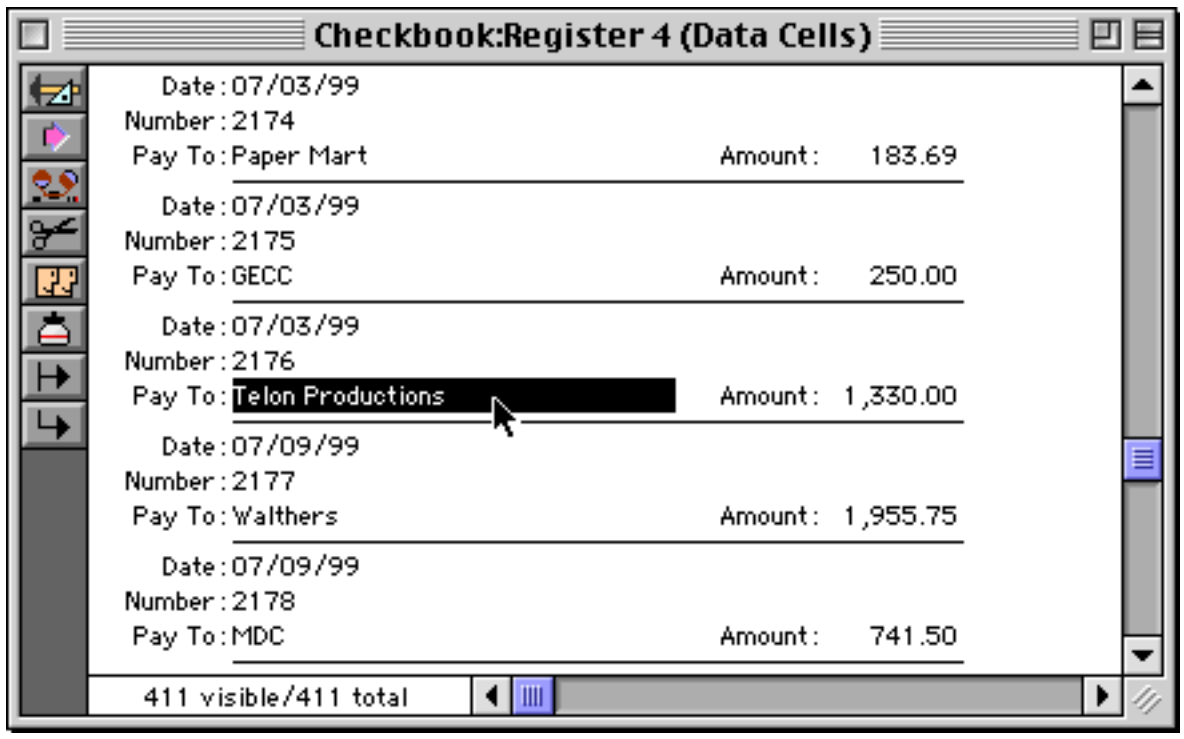
Here is a view-as-list form designed with Data Cells for editing. To learn how to add data cells to your form see “Working with Data Cell Objects” on page 635.



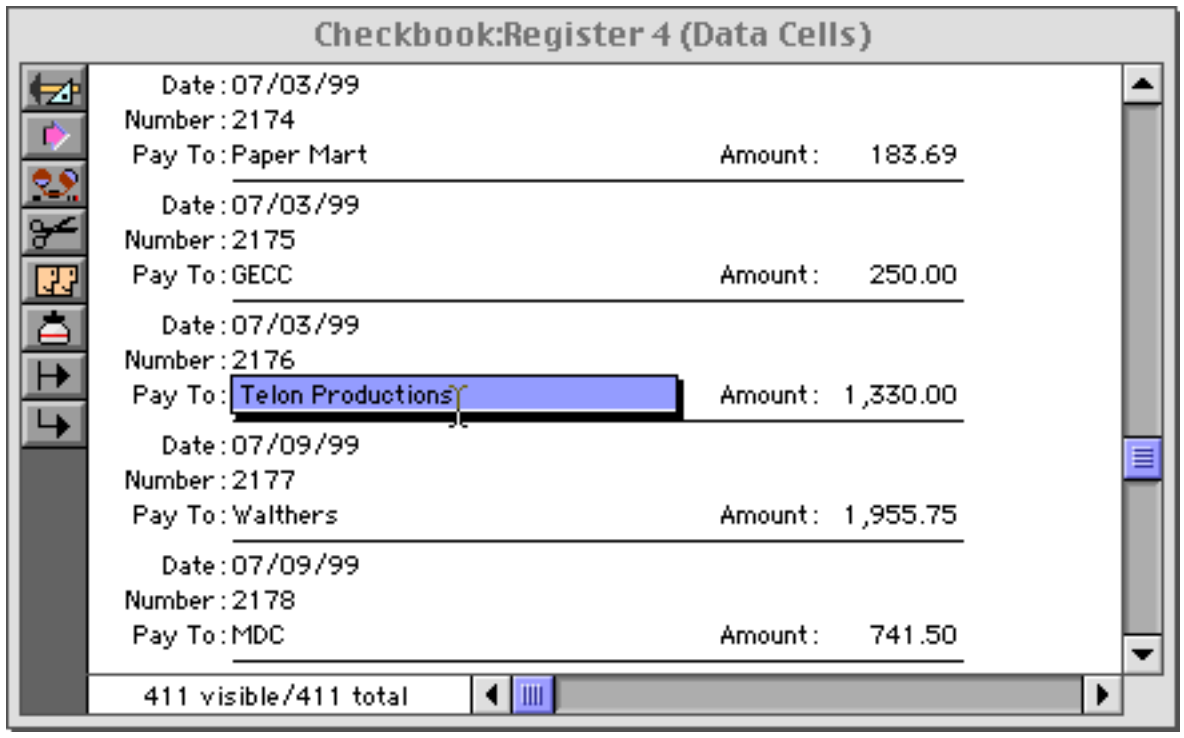
Switch to Data Access Mode to see the list view.



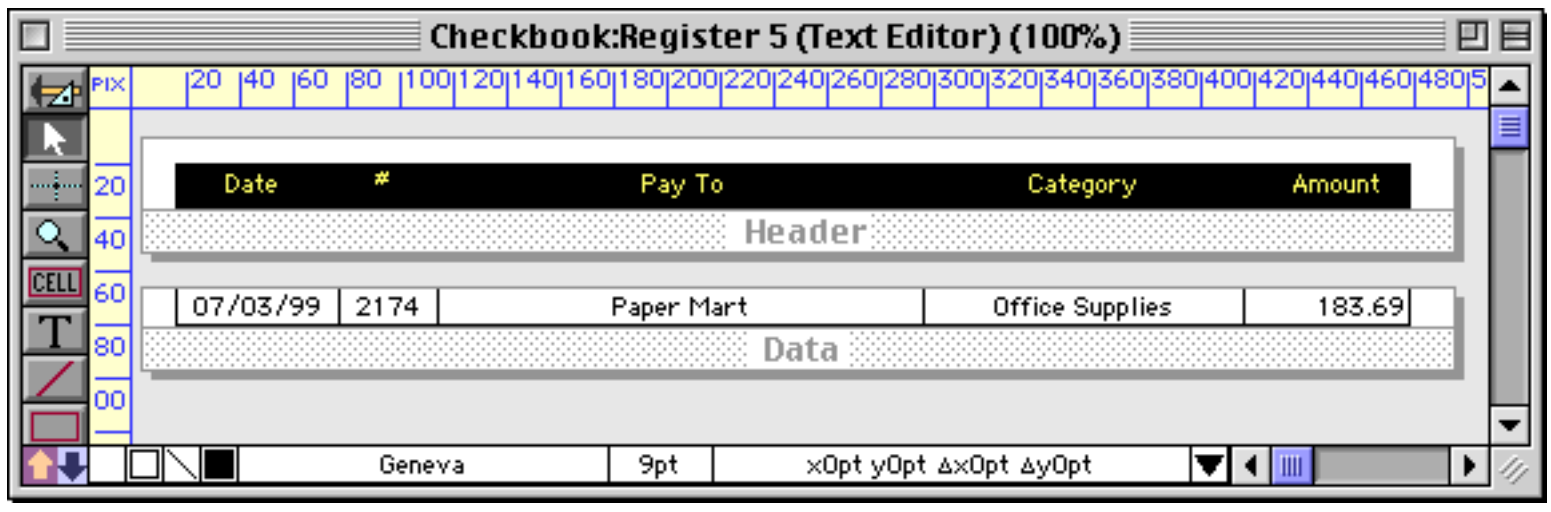
As you can see, when Data Cells are added to the form Panorama no longer highlights the entire record. Only the current cell itself is highlighted. You can click on any visible cell to highlight it.



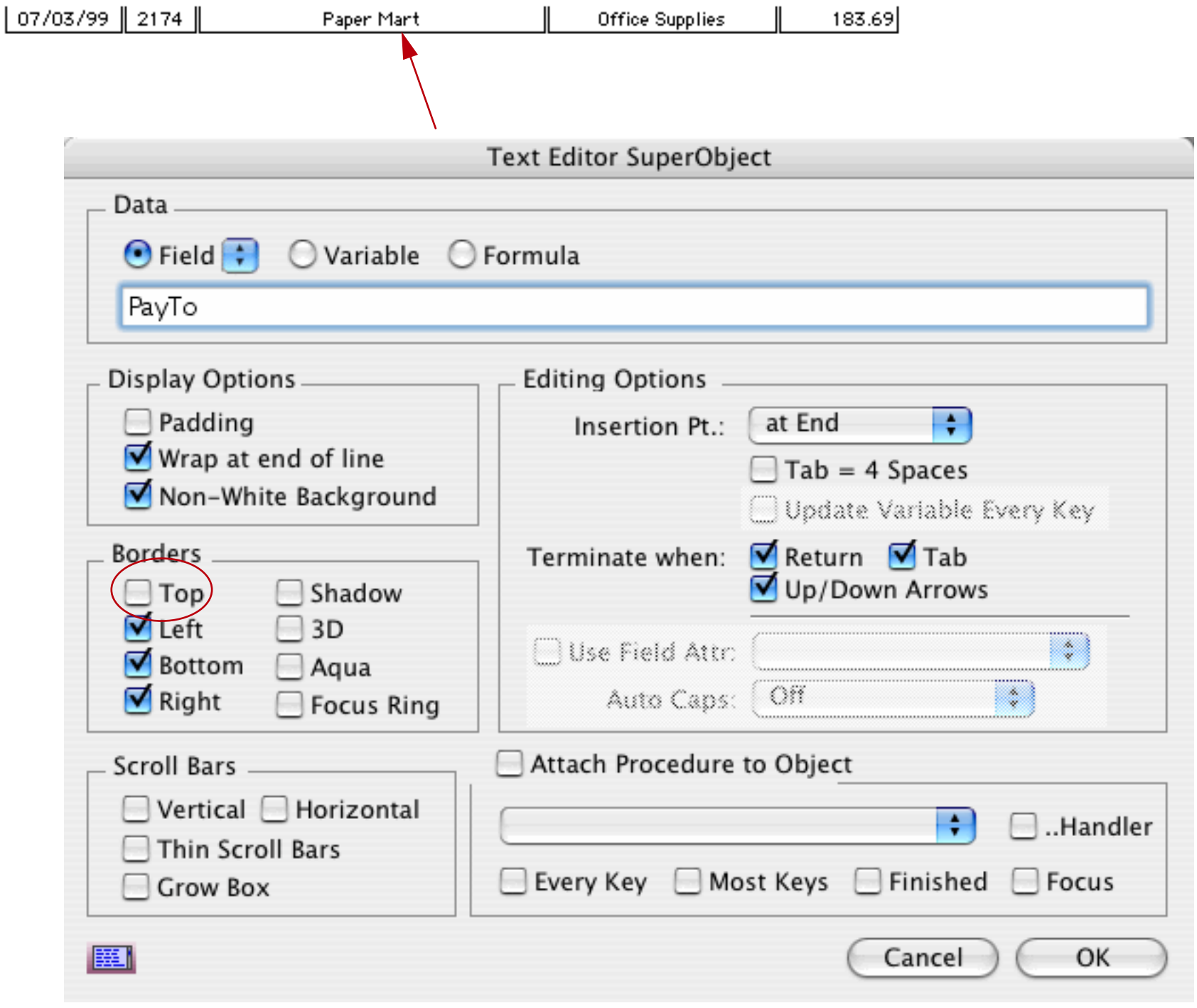
To edit a cell, double click on it. To learn more about editing within a cell see “Editing Data Within a Cell” on page 310.



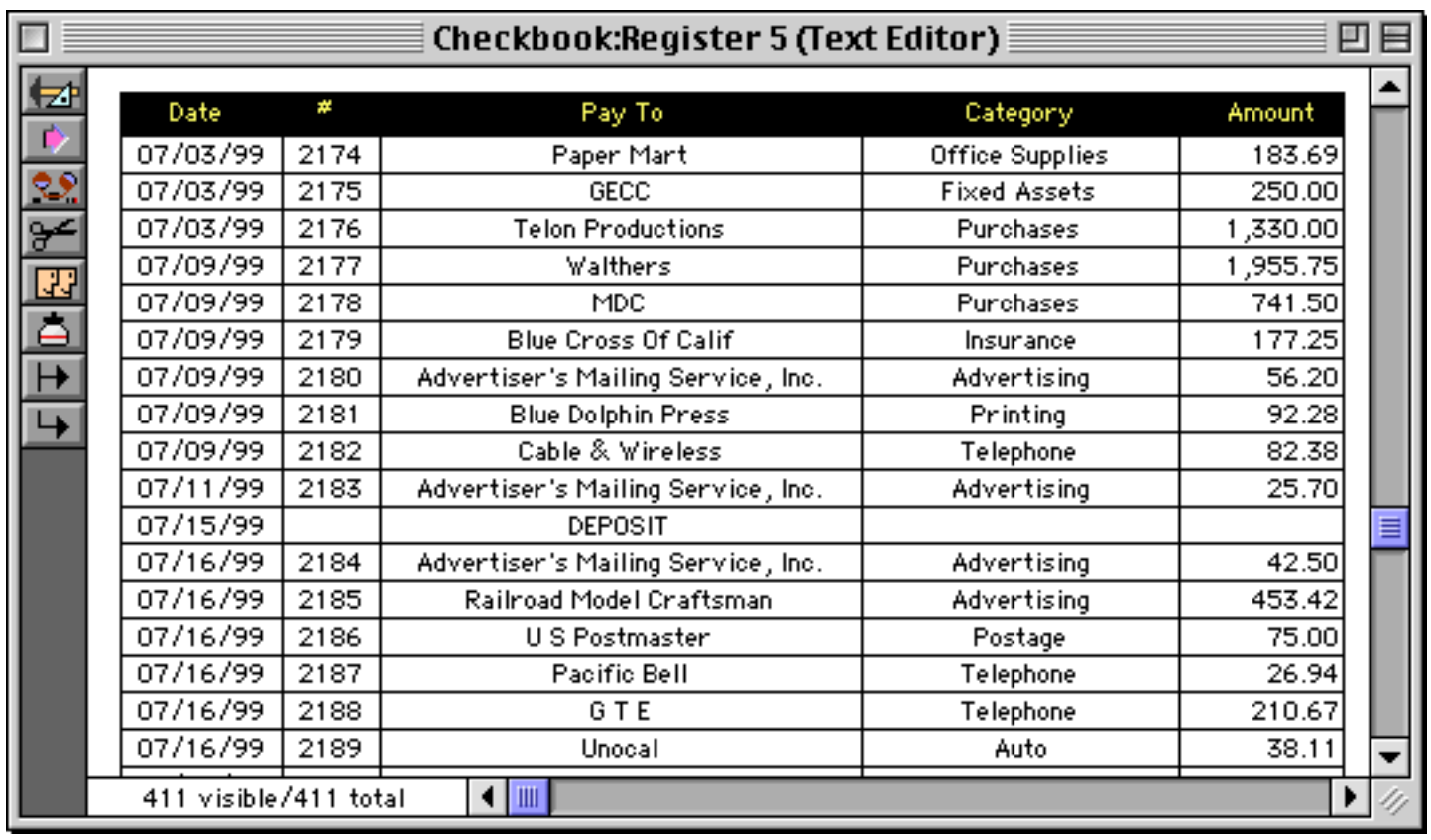
Here is a form created using Text Editor SuperObjects (see “Text Editor SuperObject” on page 639).



In this case, each object was created with borders on the left, bottom and right but not on the top. Here is an “exploded view” of these objects, along with the configuration dialog for one of the objects.



Switch to Data Access Mode to see the finished result.

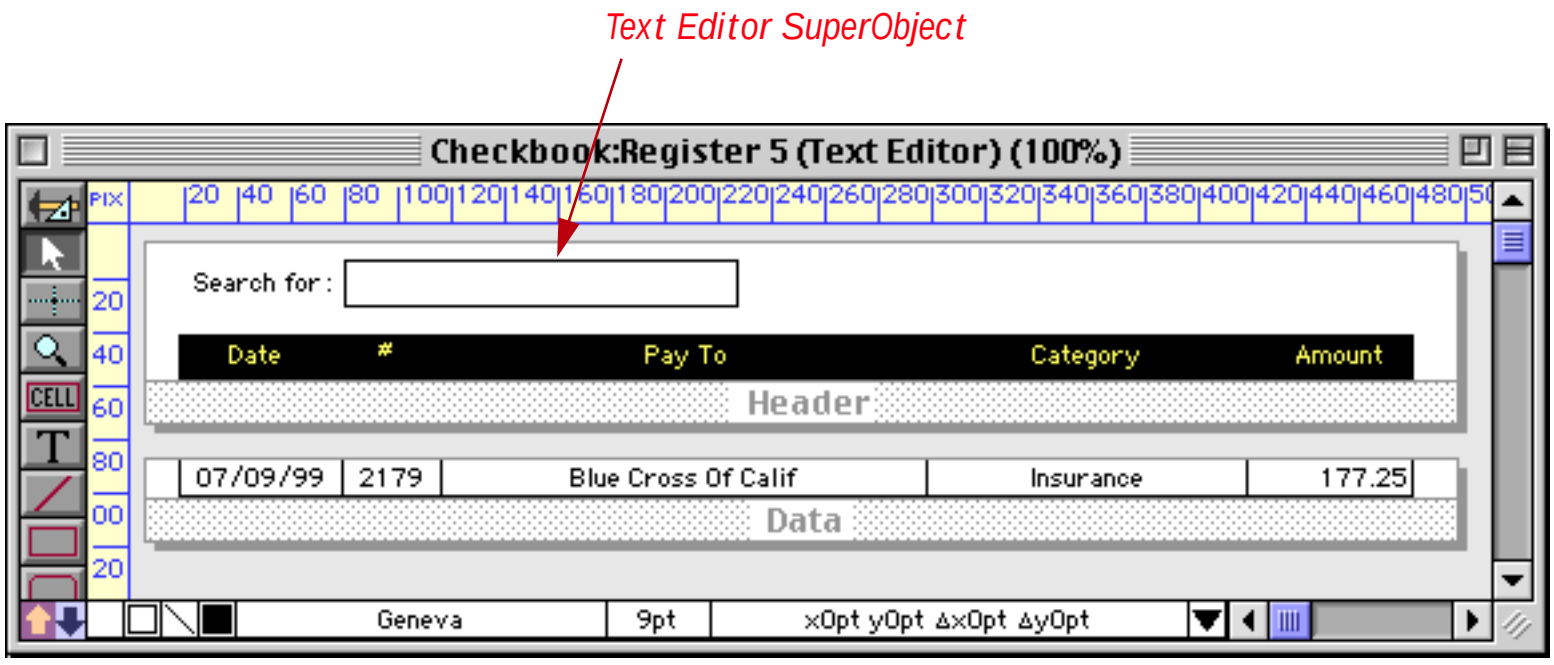


Date	#	Pay To	Category	Amount
07/03/99	2174	Paper Mart	Office Supplies	183.69
07/03/99	2175	GECC	Fixed Assets	250.00
07/03/99	2176	Telon Productions	Purchases	1,330.00
07/09/99	2177	Walthers	Purchases	1,955.75
07/09/99	2178	MDC	Purchases	741.50
07/09/99	2179	Blue Cross Of Calif	Insurance	177.25
07/09/99	2180	Advertiser's Mailing Service, Inc.	Advertising	56.20
07/09/99	2181	Blue Dolphin Press	Printing	92.28
07/09/99	2182	Cable & Wireless	Telephone	82.38
07/11/99	2183	Advertiser's Mailing Service, Inc.	Advertising	25.70
07/15/99		DEPOSIT		
07/16/99	2184	Advertiser's Mailing Service, Inc.	Advertising	42.50
07/16/99	2185	Railroad Model Craftsman	Advertising	453.42
07/16/99	2186	U S Postmaster	Postage	75.00
07/16/99	2187	Pacific Bell	Telephone	26.94
07/16/99	2188	G T E	Telephone	210.67
07/16/99	2189	Unocal	Auto	38.11

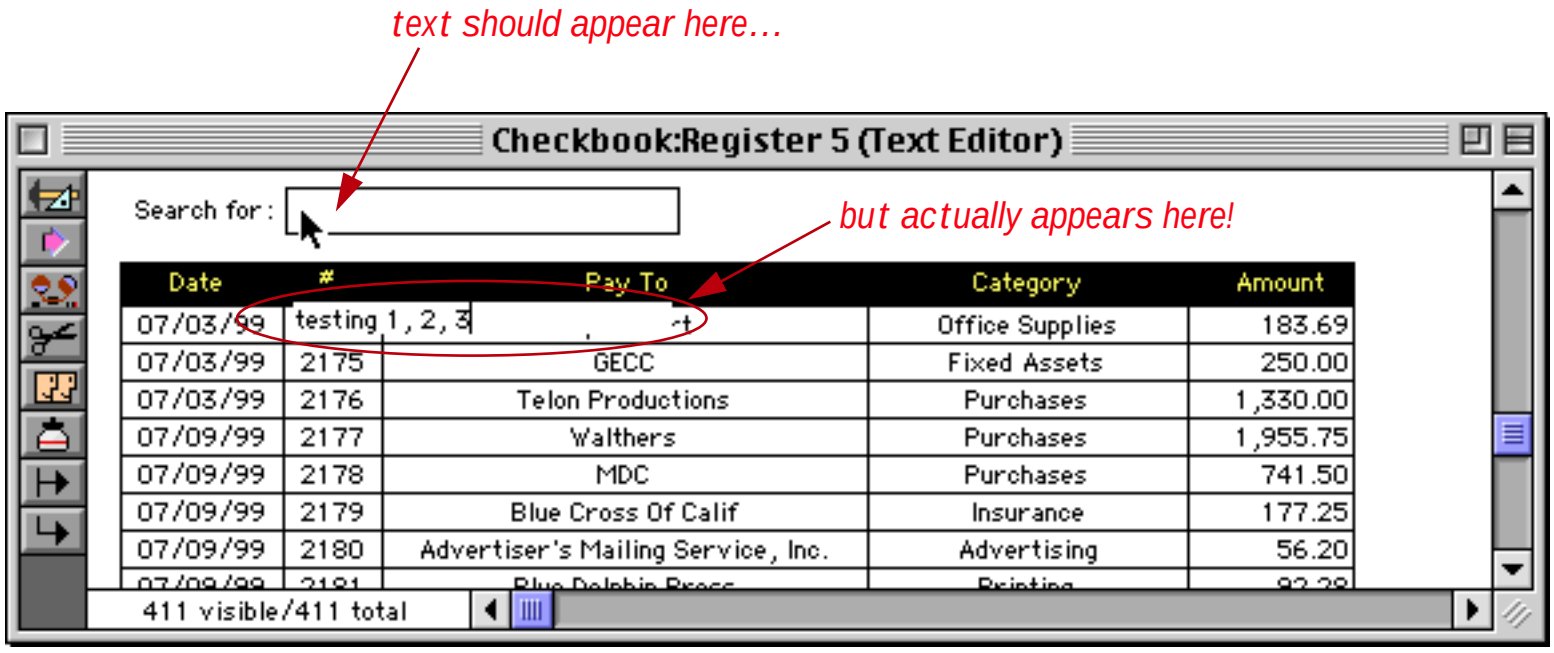
Date	#	Pay To	Category	Amount
07/03/99	2174	Paper Mart	Office Supplies	183.69
07/03/99	2175	GECC	Fixed Assets	250.00
07/03/99	2176	Telon Productions	Purchases	1,330.00
07/09/99	2177	Walthers	Purchases	1,955.75
07/09/99	2178	MDC	Purchases	741.50
07/09/99	2179	Blue Cross Of Calif	Insurance	177.25
07/09/99	2180	Advertiser's Mailing Service, Inc.	Advertising	56.20
07/09/99	2181	Blue Dolphin Press	Printing	92.28
07/09/99	2182	Cable & Wireless	Telephone	82.38
07/11/99	2183	Advertiser's Mailing Service, Inc.	Advertising	25.70
07/15/99		DEPOSIT		
07/16/99	2184	Advertiser's Mailing Service, Inc.	Advertising	42.50
07/16/99	2185	Railroad Model Craftsman	Advertising	453.42
07/16/99	2186	U S Postmaster	Postage	75.00
07/16/99	2187	Pacific Bell	Telephone	26.94
07/16/99	2188	G T E	Telephone	210.67
07/16/99	2189	Unocal	Auto	38.11

As you can see, one potential disadvantage of this technique is that nothing is highlighted, so it is very difficult to see which record is current. (If you need to be able to tell which record is highlighted, use Data Cells.) To edit a particular item just click on it.

You might be tempted to try placing a Text Editor SuperObject or Data Cell onto the header tile, like this.



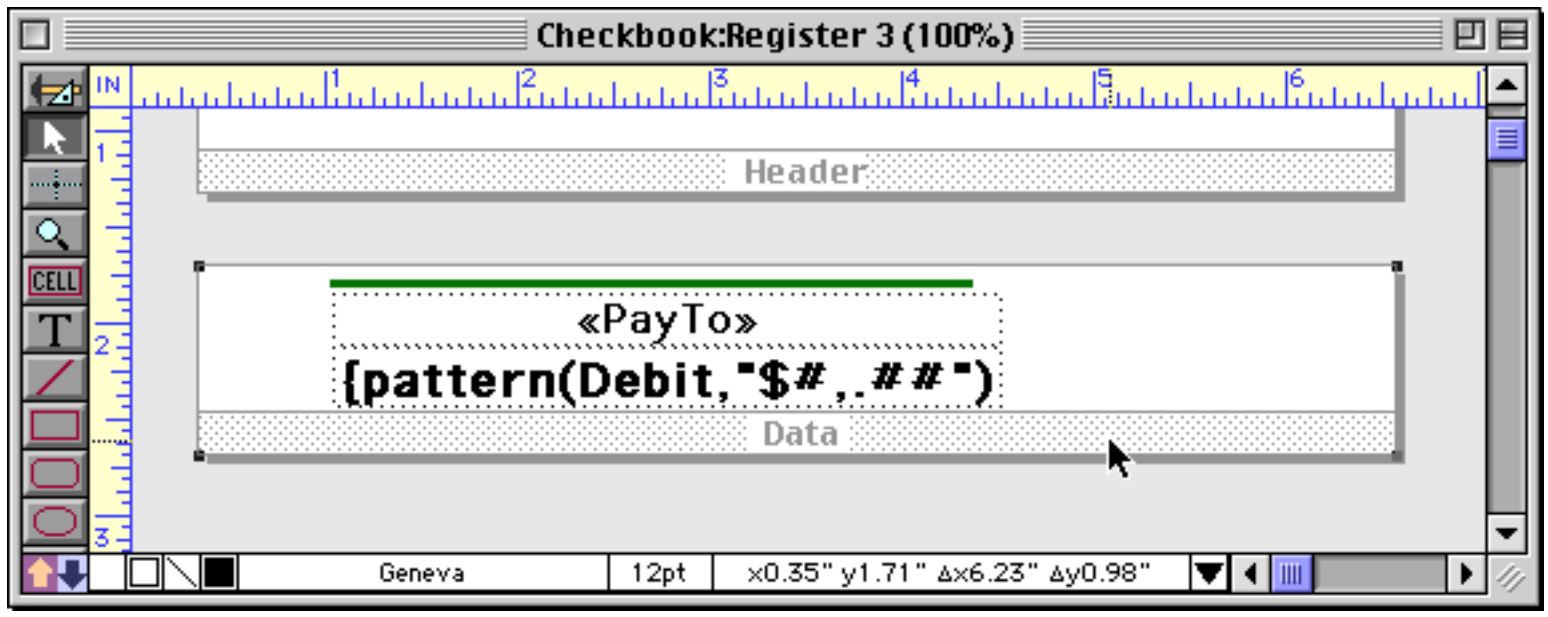
Unfortunately, this does not work correctly. When you attempt to edit the text, the edited text appears in the wrong position, something like this.



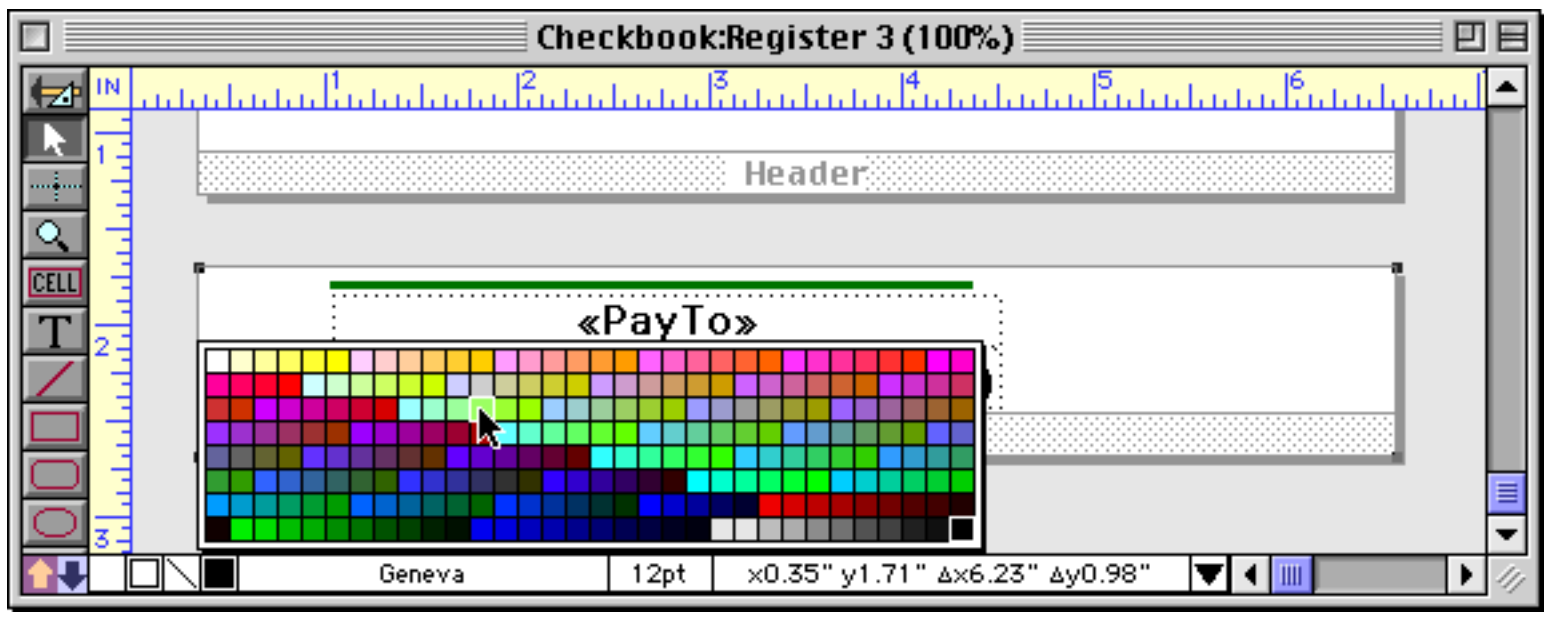
This problem may be corrected in a future version of Panorama, but for now you can only edit text within the data tile.

View-As-List Background Colors

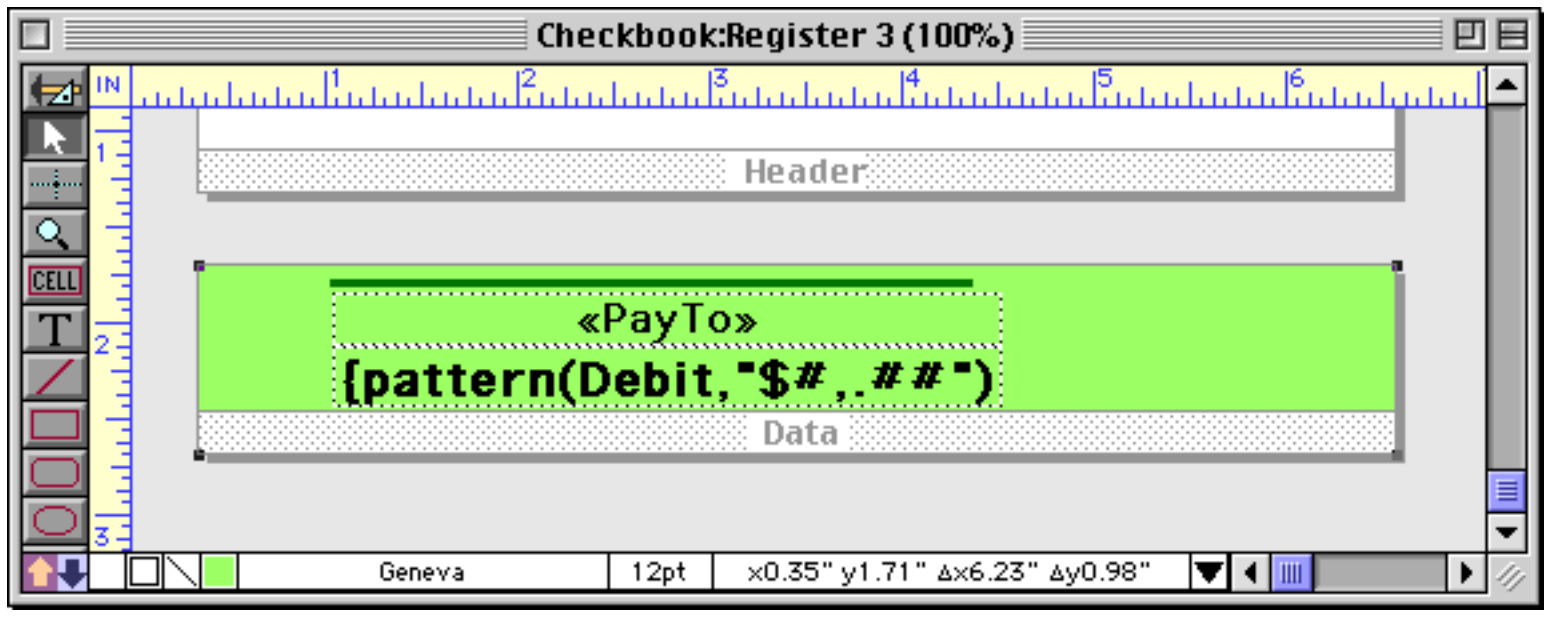
A tile's surface is normally white. However, it is possible to change the color of the tile surface to any color in Panorama's color palette. To change the color, start by clicking on the tile's drag bar to select it (see "Working with Tiles" on page 898).



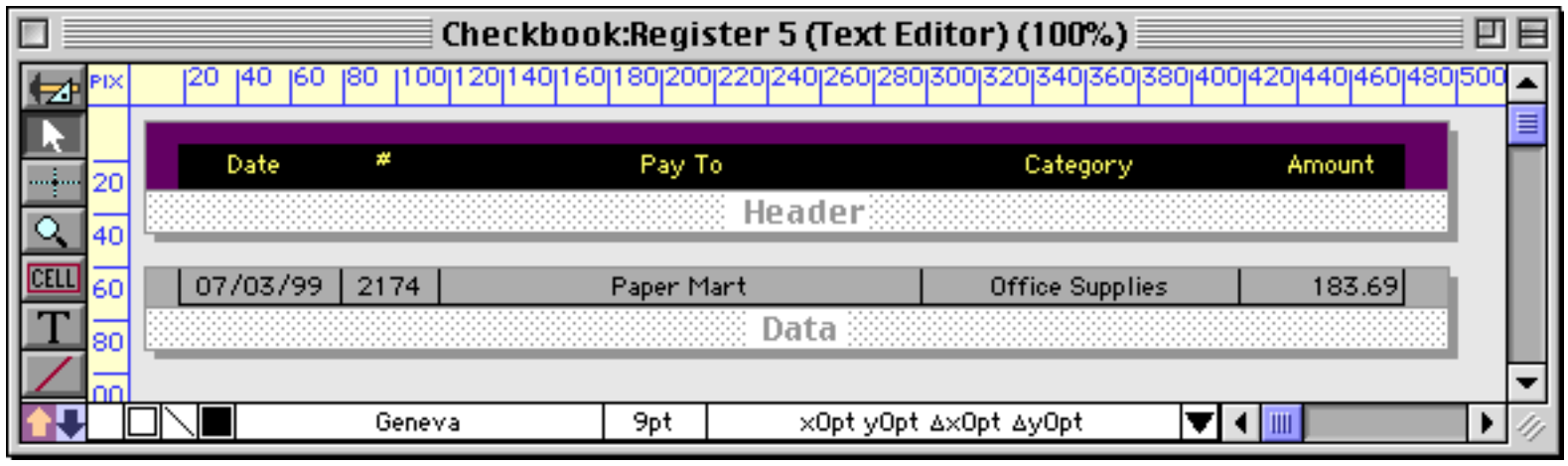
Once the tile is selected you can select a new color from the Graphic Control Strip (see "Color" on page 526). You can choose any color you want, but usually the lighter grays and pastels work best.



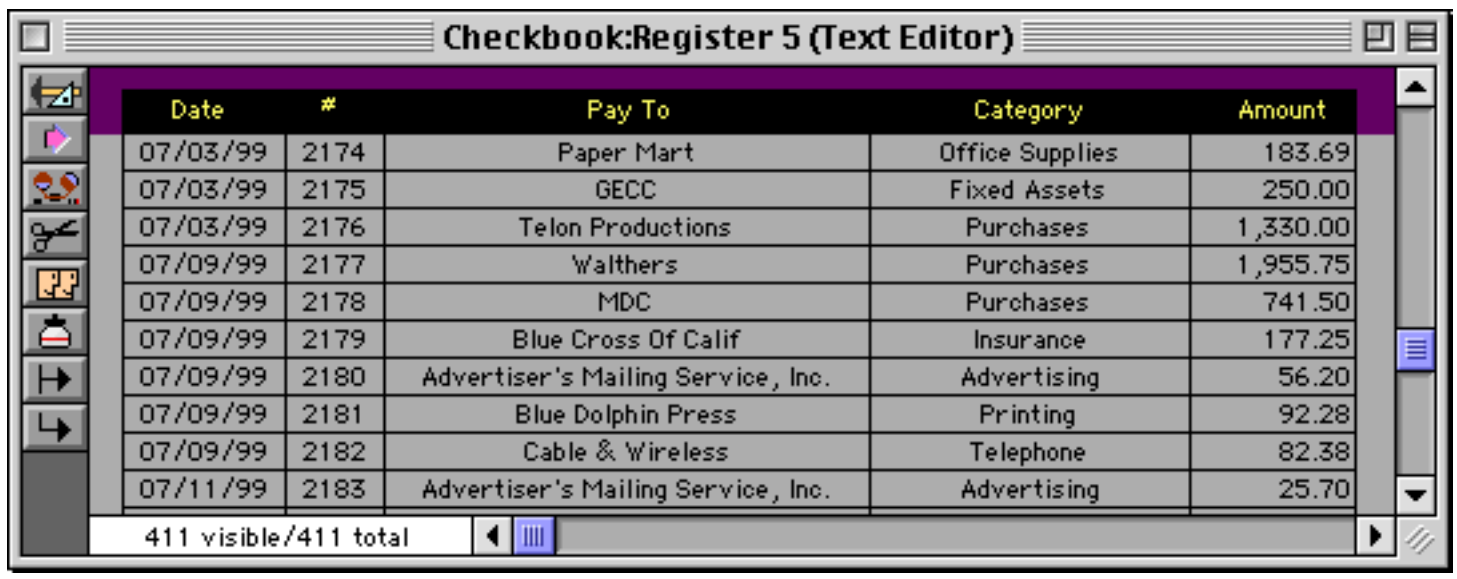
When you release the mouse the tile's surface changes to the new color. By the way, if you want to change the tile's surface back to white, select black (the color in the far bottom right).



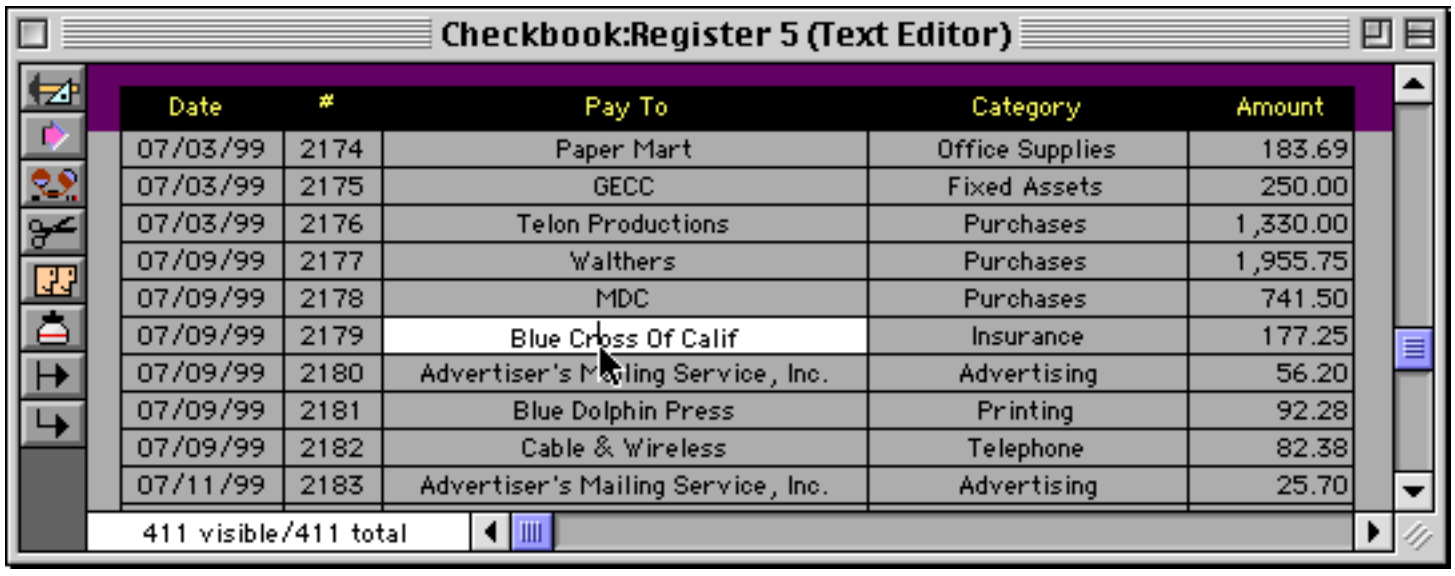
Colored tiles work well with Text Editor SuperObjects. Here's an example.



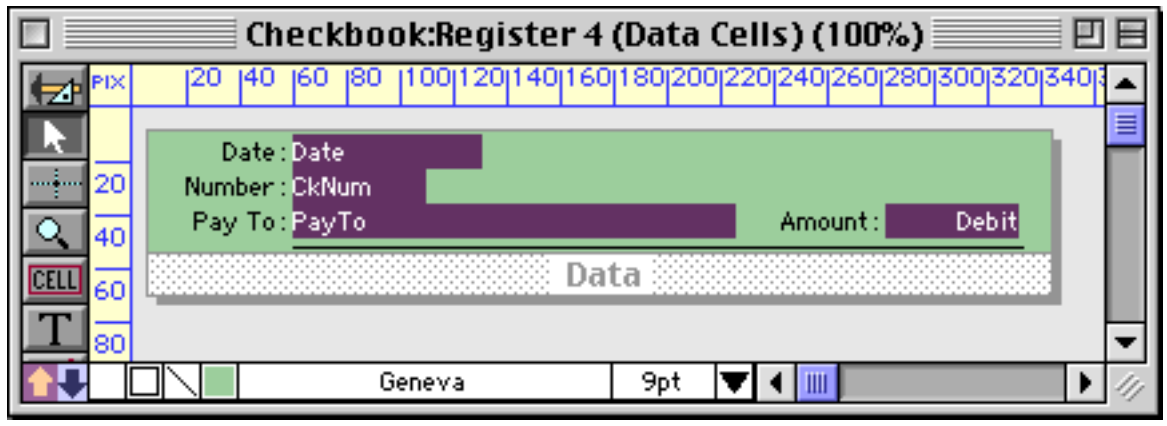
Here's the same form in Data Access Mode.



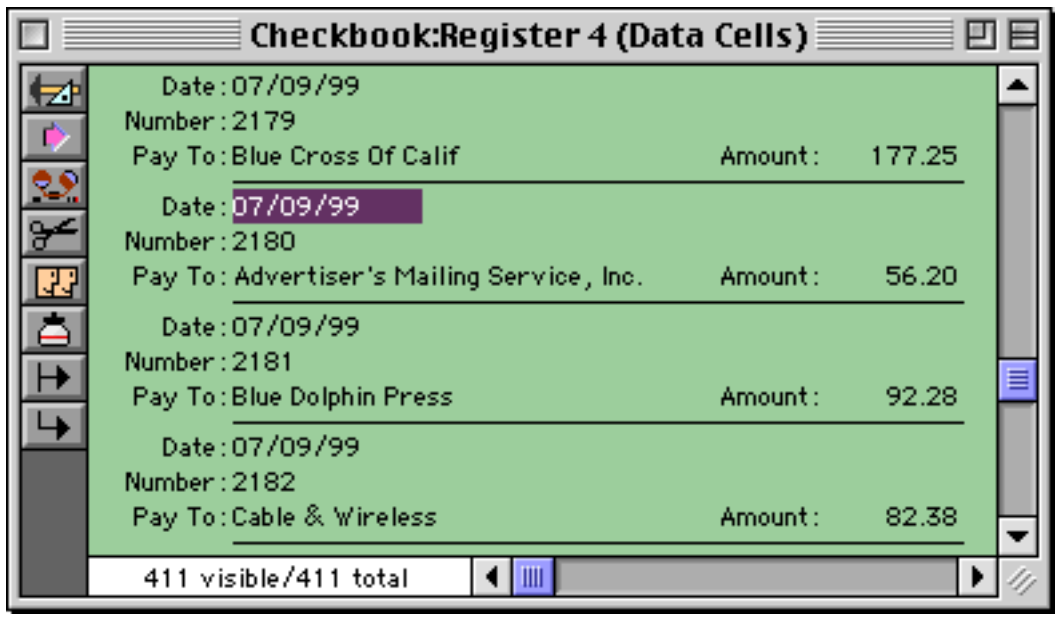
When you click on an item to edit it, the entire area turns white (be sure you have selected the Non-White Background option, see “Text Editor Options” on page 643).



Colored tiles can also work with Data Cells.



Here’s what this form looks like in Data Access Mode.

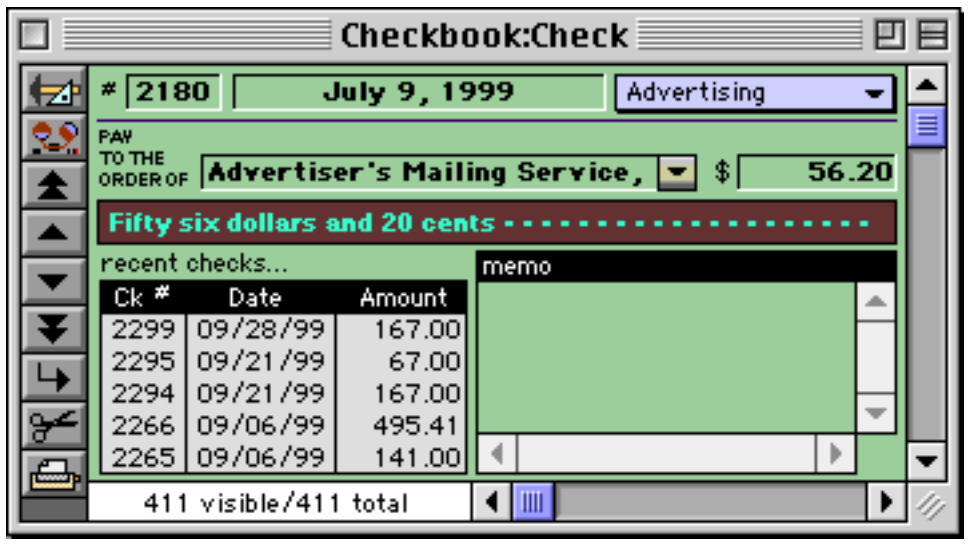


Buttons on a View-As-List Form

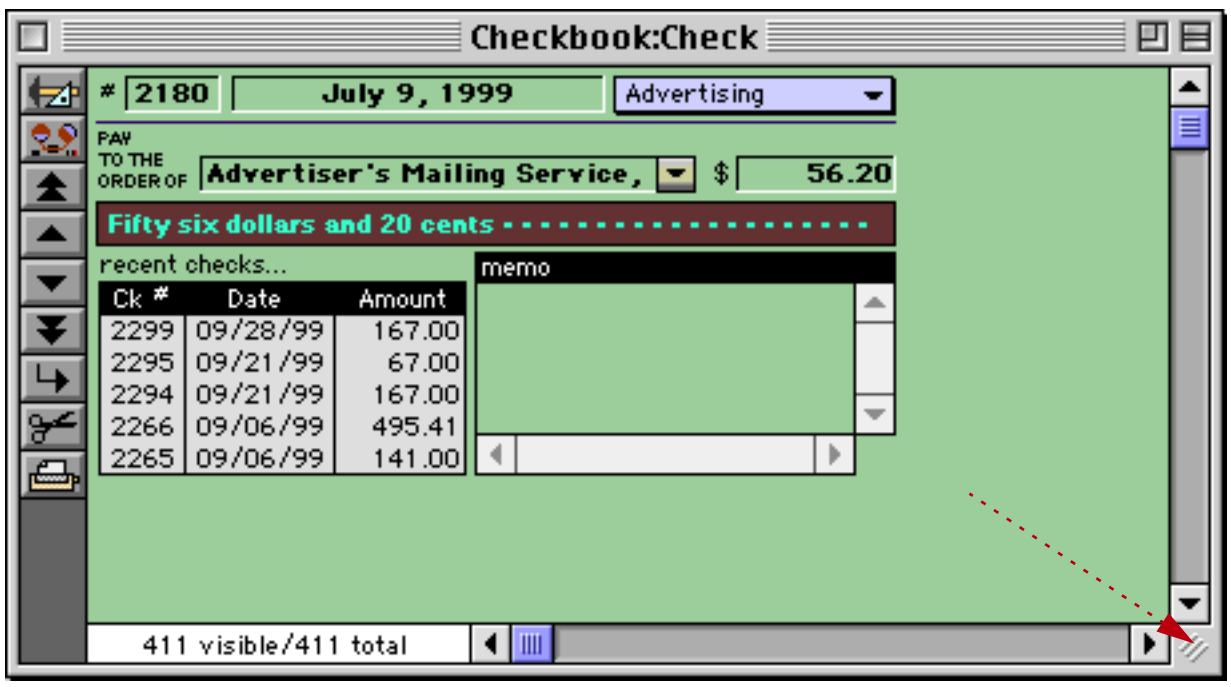
You can place any kind of button on either the data tile or header tile of a view-as-list form — push buttons, data buttons, pop-up menus etc. Data buttons that are placed on the data tile should be linked to fields. Data buttons that are placed on the header tile should be linked to variables (see “Variables” on page 1199 and “Variables” on page 1370).

Elastic Forms

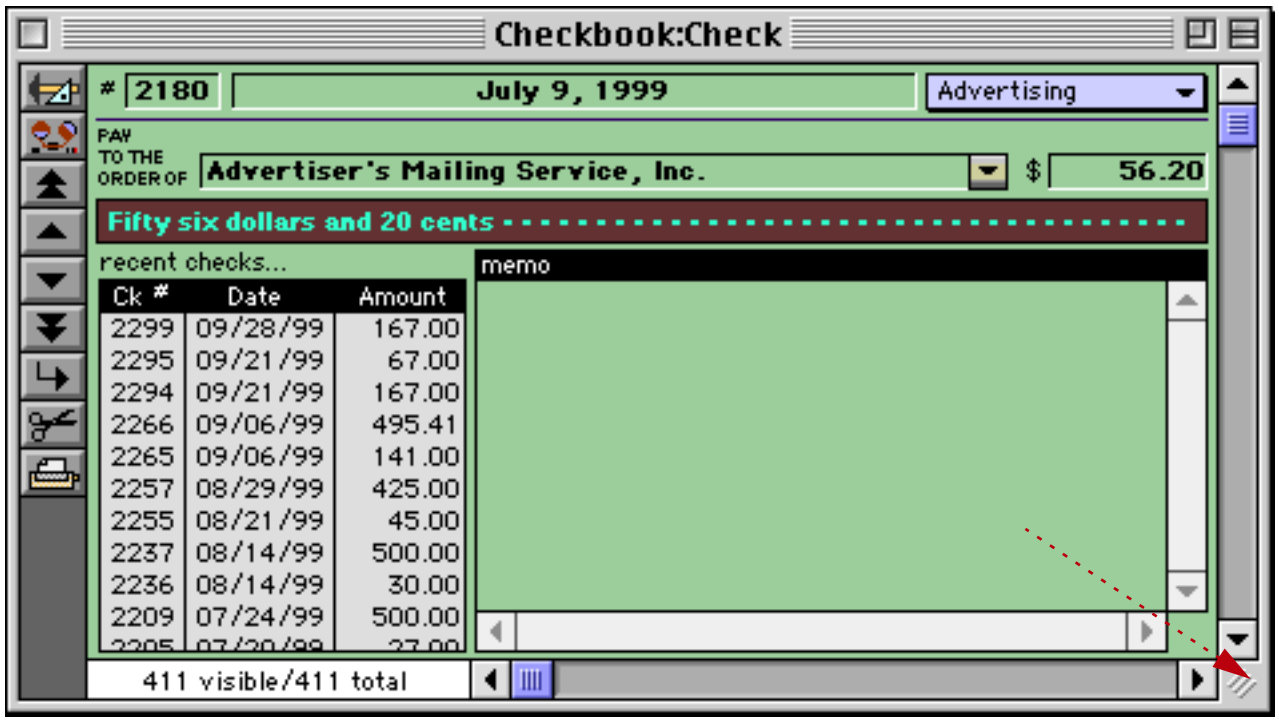
Elastic Forms is a feature that allows a form to adjust intelligently when the window containing the form is resized or zoomed. When the form is designed, you decide how the individual elements will expand or shift as the form changes size. Here's an example of a typical form.



If the window containing an ordinary form is resized, the form remains the same, in this case leaving a large blank area.



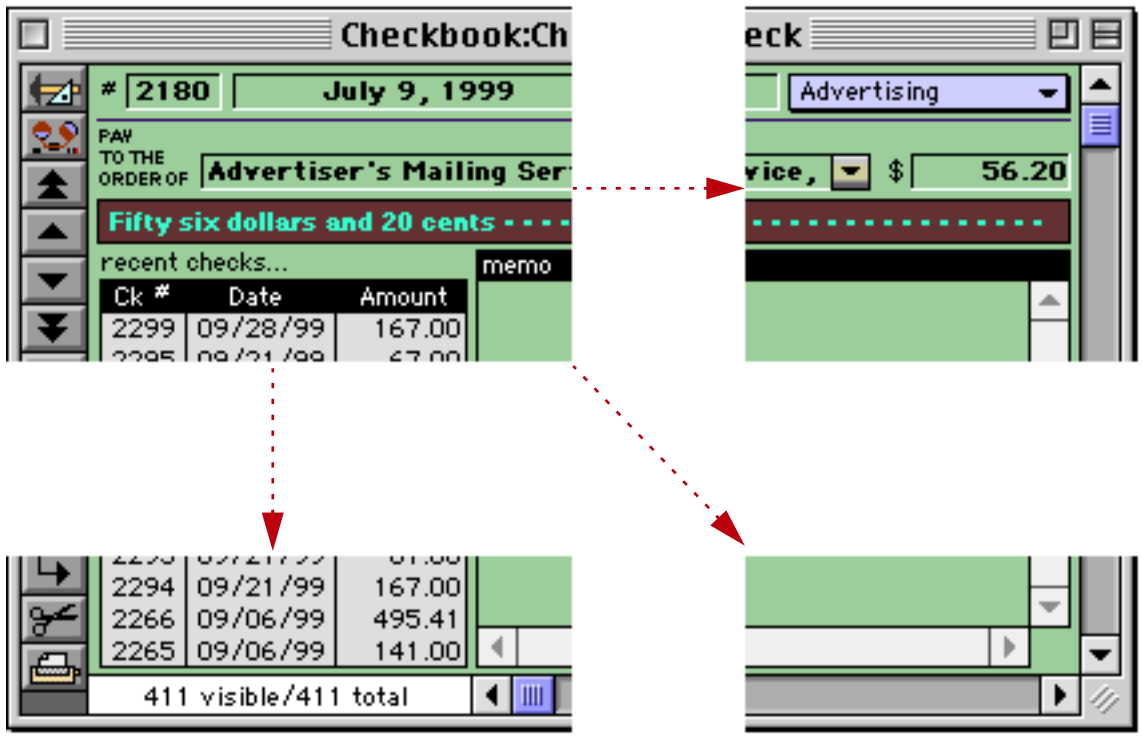
However, if the form is elastic the objects within the form will adjust themselves to the new size.



The best part is that once you learn how, you can turn almost any form into an elastic form in just a couple of minutes!

Theory of Elastic Forms

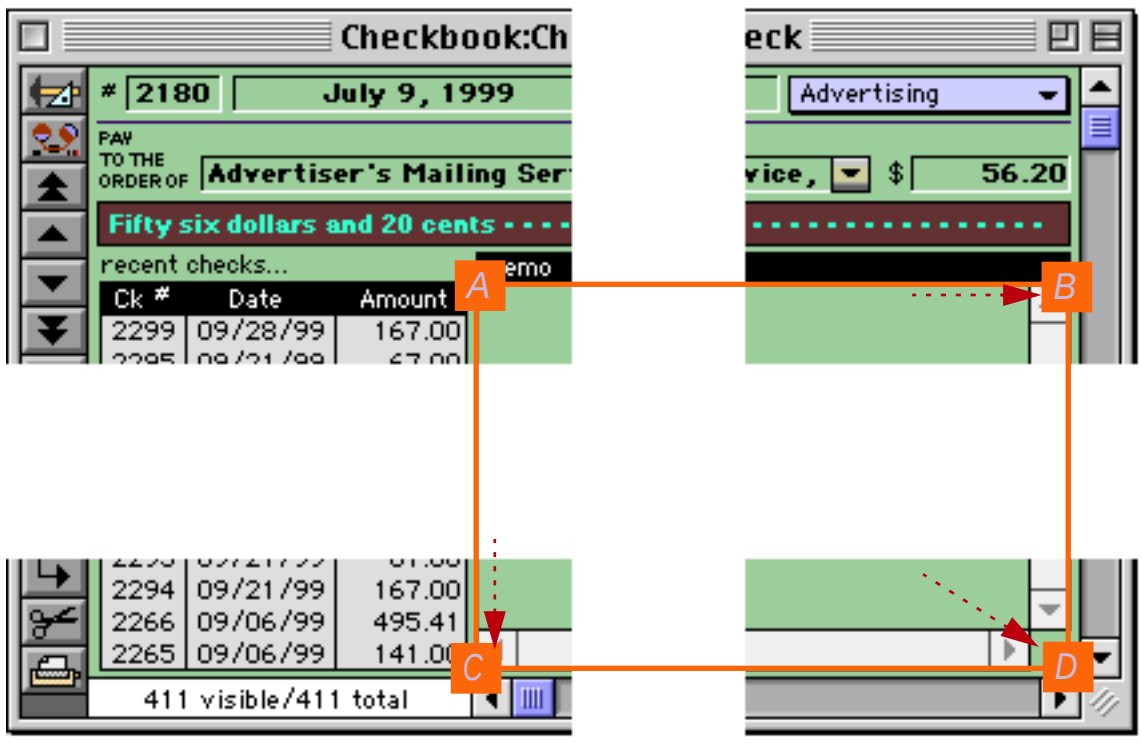
The basic theory of elastic forms is simple. The form is divided into four quadrants, like this.



The upper left hand quadrant always remains fixed, no matter how large or small the window gets. The upper right hand quadrant slides back and forth horizontally as the window size changes, while the lower left hand quadrant slides vertically. The lower right hand quadrant slides diagonally, always sticking to the bottom right hand corner of the window.

As Panorama adjusts the form, it doesn't adjust entire objects. Instead it adjusts individual points (i.e. object corners). If an object is split across multiple quadrants, that object will adjust in size as the form expands and shrinks. If an object is not split across multiple quadrants, it will remain the same size but may slide to a new position depending on which quadrant it is in.

As an example consider the Text Editor SuperObject shown below. The four corners of this object are split across the four quadrants. Point A, in the upper left quadrant, stays put. Point B, in the upper right quadrant, slides to the right. Point C slides down, while Point D slides diagonally. The end result is that the object expands to fill up the newly available space.



On the other hand the check number, amount and the category pop-up menu are all contained within a single quadrant, so these objects do not change in size. The Date and Pay To fields are split across two quadrants, so they expand in width but not in height.

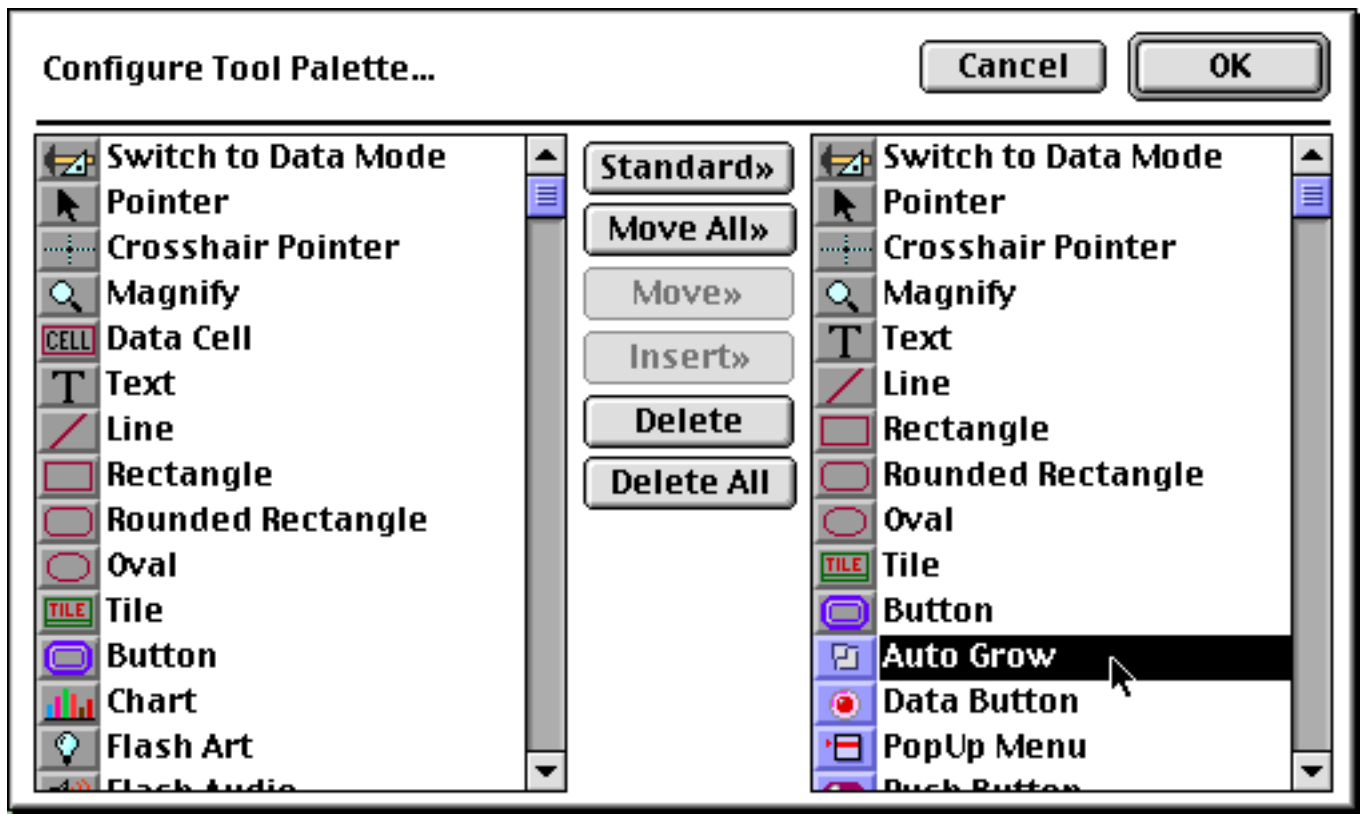
As you can see, the point where these four quadrants come together is very important (see “Defining the Quadrants” on page 915). You’ll need to pick this point carefully to create a form that expands and shrinks the way you want it to. Usually you’ll have one primary object that will expand and shrink as the window expands and shrinks. The quadrant meeting point should be inside this object. You may also have secondary objects above or below this object that need to expand and shrink in width only. The quadrant meeting point should line up with the middle of these objects. (If it is impossible to line up the quadrant meeting point with all of the secondary objects, you can create one or more slave Auto Grow objects. Slave Auto Grow objects allow you to create extensions that stick out of one or more quadrants (see “Non-Rectangular Quadrants” on page 924).)

Building an Elastic Form

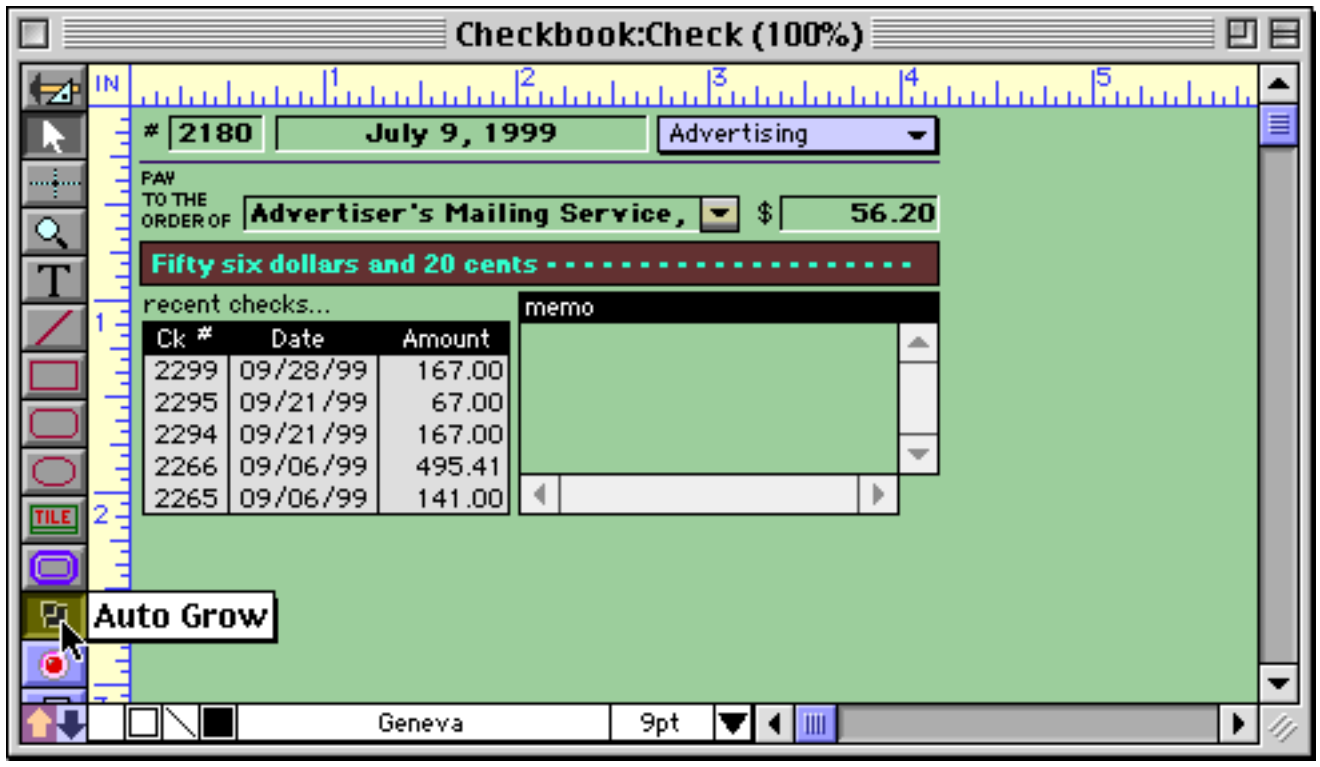
The first step in building an elastic form is to create a regular non-elastic form. Just use Panorama's regular shape and user interface elements. It's usually best to create the form in its smallest possible configuration. In other words, any form elements that may expand and shrink should be created in their "maximum shrink" size and shape. Once the form is created, be sure to save the database before going further.

Defining the Quadrants

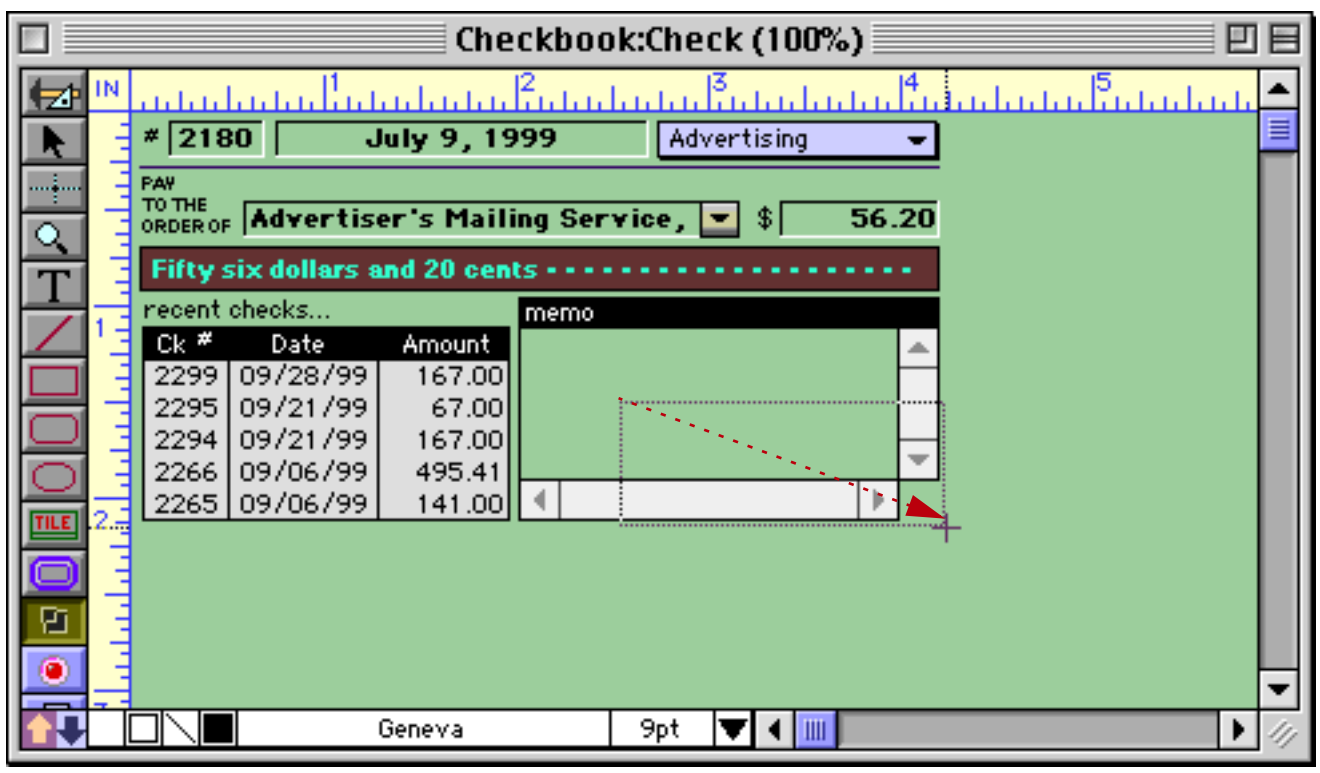
Once the form is created, the next step is to divide the form into four quadrants. This is done with the **Auto Grow SuperObject™**. The Auto-Grow tool is not in the default tool palette, so you'll need to use the **Tool Palette** dialog to add this tool to the palette if it is not already there (see "Customizing the Tool Palette" on page 497).



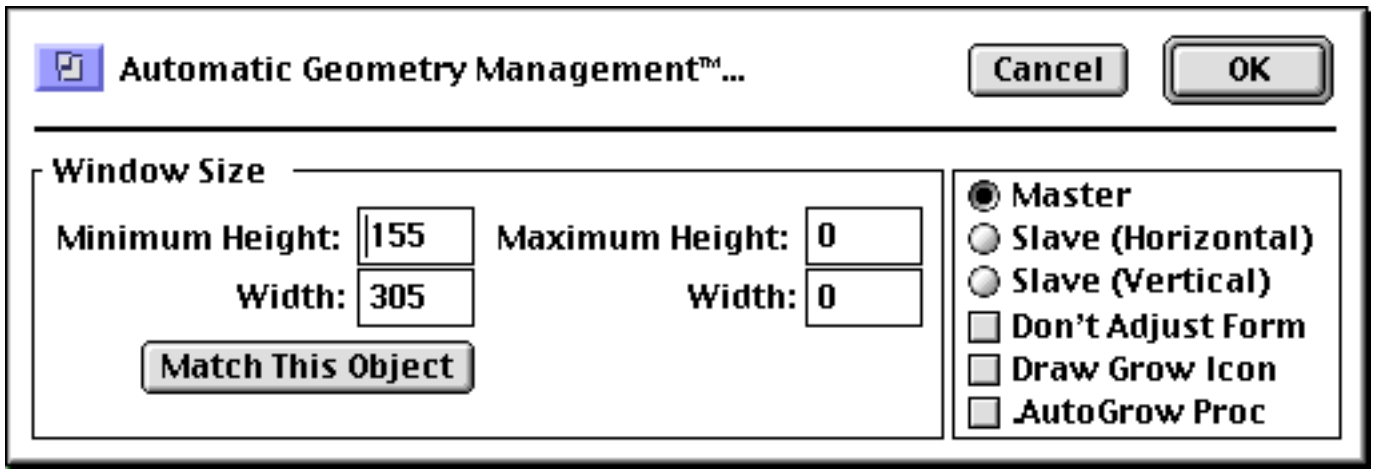
Now that the tool is added to the palette you can select it.



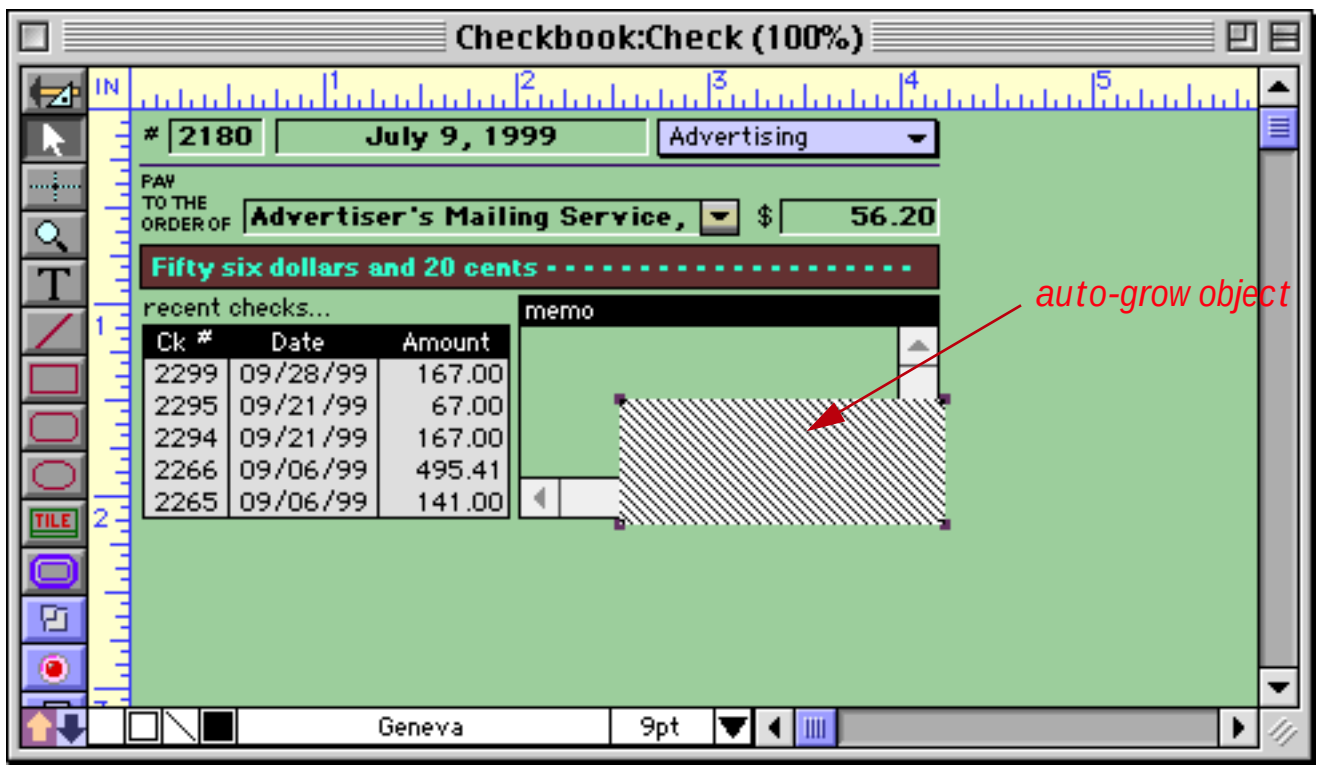
Once the tool is selected, drag the mouse across the form in the location where you want to create the auto-grow object. The Auto Grow SuperObject should cover the lower right hand quadrant of the form (the quadrant that will shift down and to the right diagonally when the window expands). The upper left corner of this Auto Grow SuperObject will determine the point where the four quadrants meet.



When you release the mouse, the Auto-Grow configuration dialog will appear.

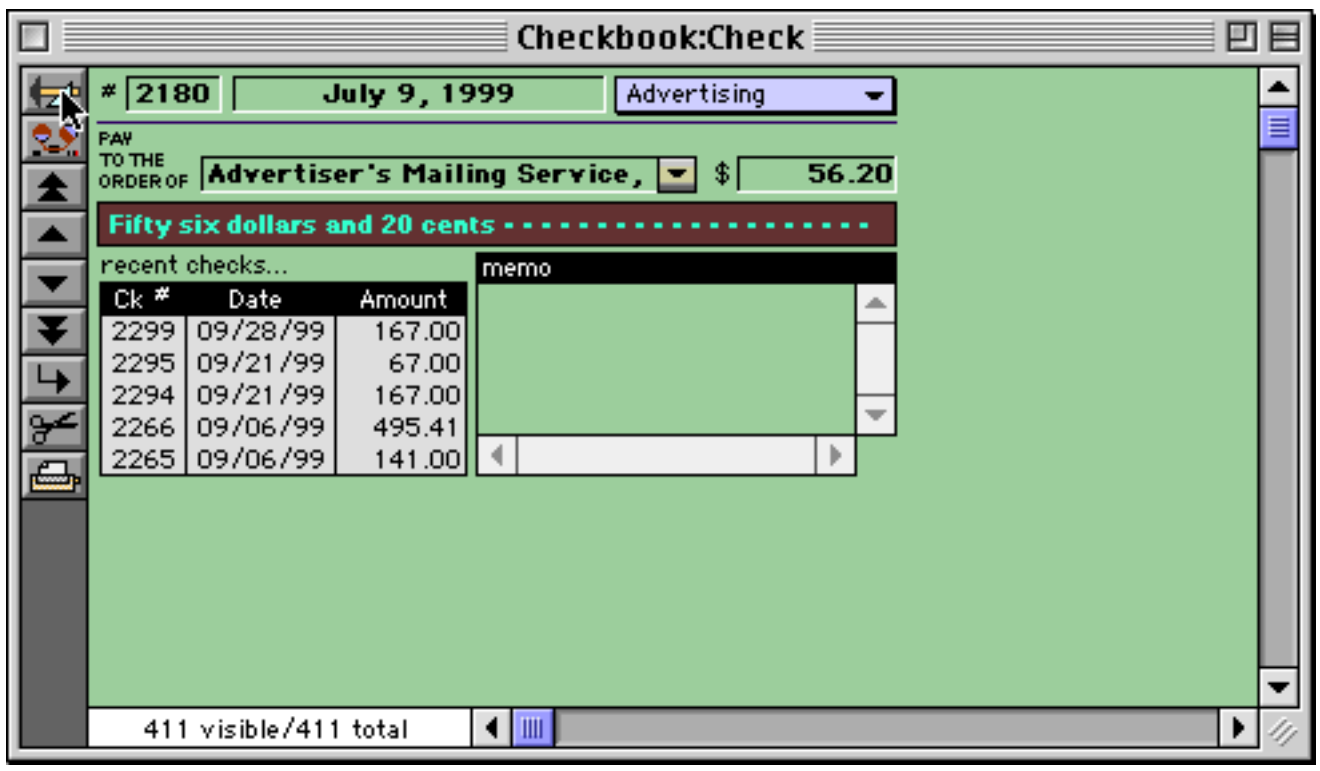


For a basic elastic form just press the **OK** button.

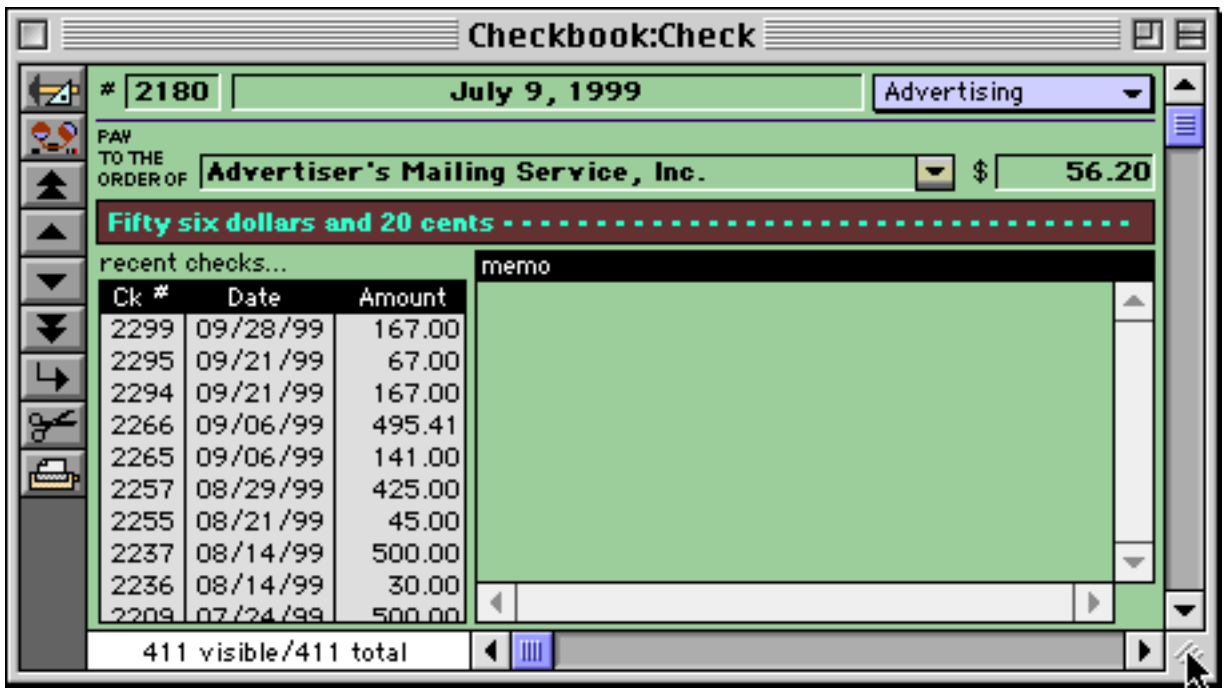


Using the mouse and the arrow keys, make sure the Auto Grow object is positioned exactly where you want the lower right quadrant of the form to be (see "Nudging an Object (or Objects)" on page 509 and "Nudging the Size of an Object" on page 513). Be sure to allow some space for a slight margin on the right and bottom side of the form.

Once the Auto Grow object is set up, switch the form to Data Access Mode. At first, all you'll notice is that the Auto Grow object disappears.



Now change the size of the window. As soon as you change the size of the window, the form will adjust to the new window size.

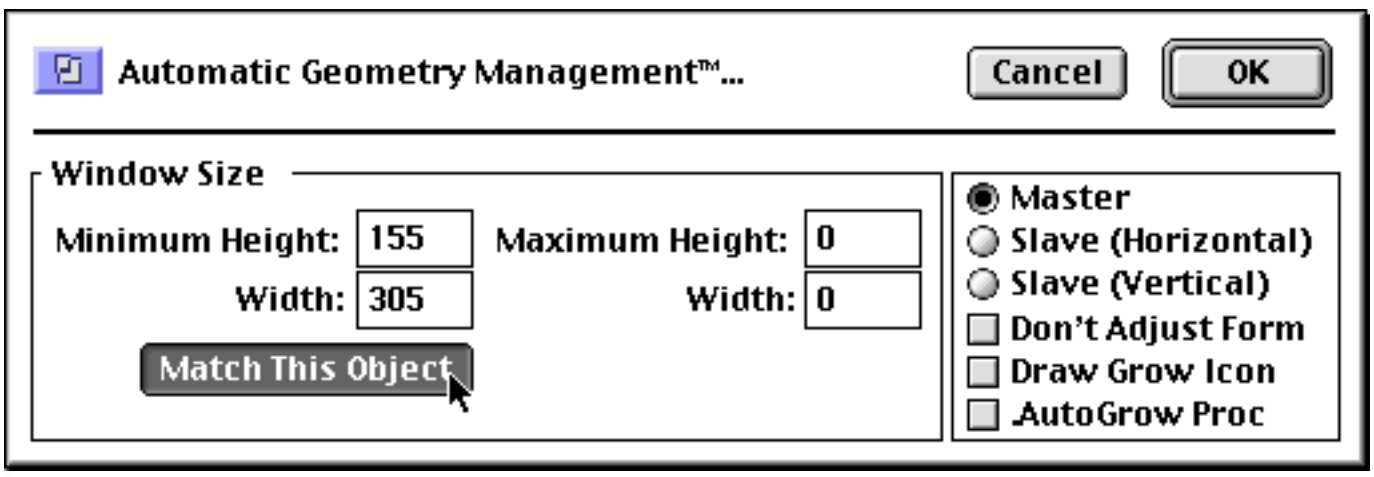


Try making the window larger or smaller, or zooming the window. The form will automatically adjust to the new size.

If the form doesn't quite adjust the way you wanted, set it back to the minimum size and then go into Graphics Mode and adjust the auto-grow object. In extreme cases you may need to **Revert To Saved** to get the original form configuration back (you *did* save the database before you tried your elastic form, right?).

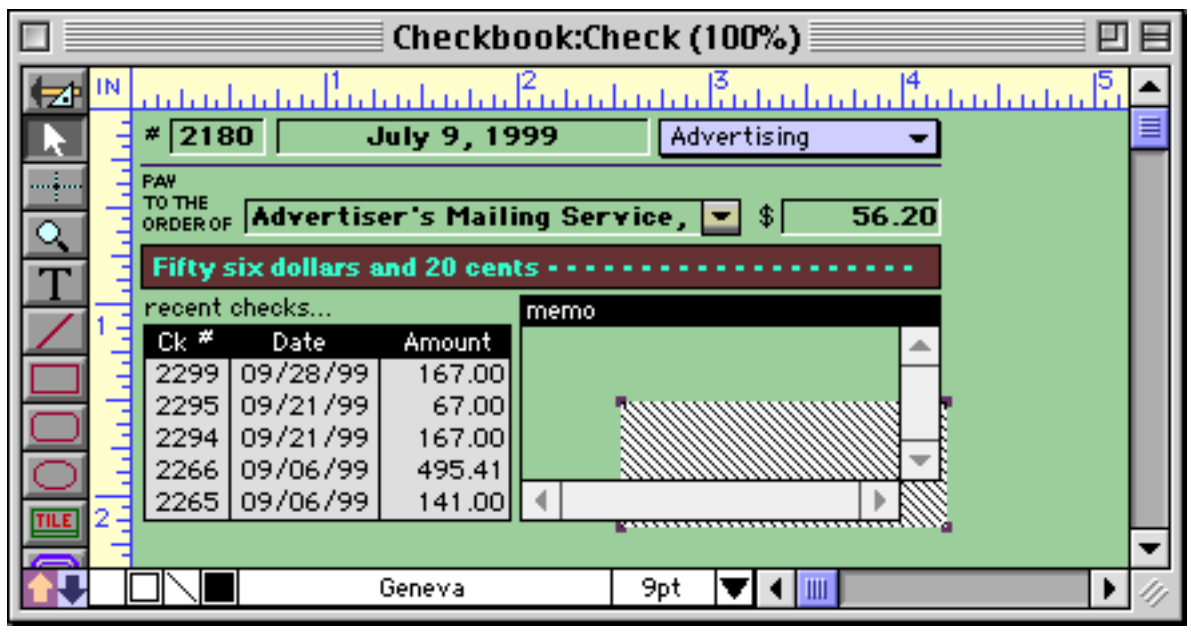
Once the Auto Grow object is exactly positioned over the lower right quadrant of the form you can set the minimum window size. The following discussion assumes that the form is currently in its minimum window size configuration, with all expandable objects set to their smallest size. In other words, the bottom right hand corner of the Auto Grow object defines the smallest possible window size.

To set this minimum size, double click on the Auto Grow object, then click on the **Match This Object** button.



The dialog will show the dimensions of the minimum window size. Click on the **OK** button to permanently set this size.

After the minimum window size is set, use the **Send to Back** command (Arrange menu) to move the Auto Grow object underneath all of the other objects in the form (see "Changing the Stacking Order" on page 569).

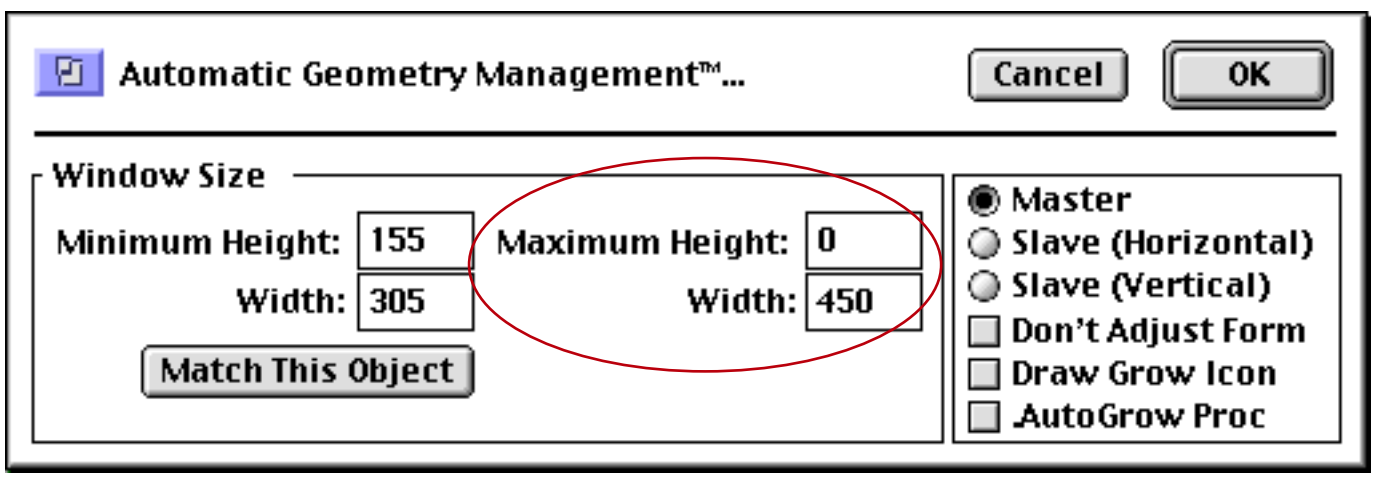


This prevents the Auto Grow object from interfering with the operation of other objects (for example buttons). Whether it is on top or on the bottom, the Auto Grow object will become invisible when the form is switched from Graphics Mode to Data Access Mode.

The Auto Grow dialog also allows you to manually set the minimum window size by typing in the dimensions. This is necessary if the form elements are not currently at the minimum window size. We recommend that you avoid this if at all possible. Only edit the form when it is in its minimum size configuration.

Maximum Window Size

Panorama windows can normally be expanded to the full height and width of the computer screen (or even across multiple monitors if your computer has them). In some cases, however, you may wish to restrict the maximum height or width of the window. Double click on the auto-grow object to set the maximum dimensions.



In this case the Maximum Height is set to 0. When the maximum is set to zero, Panorama treats it as if there is no maximum. In other words, the vertical expansion of this window is unlimited, but the horizontal expansion is limited to 450 pixels. You can set a limit on the vertical expansion (height), horizontal expansion (width) or both.

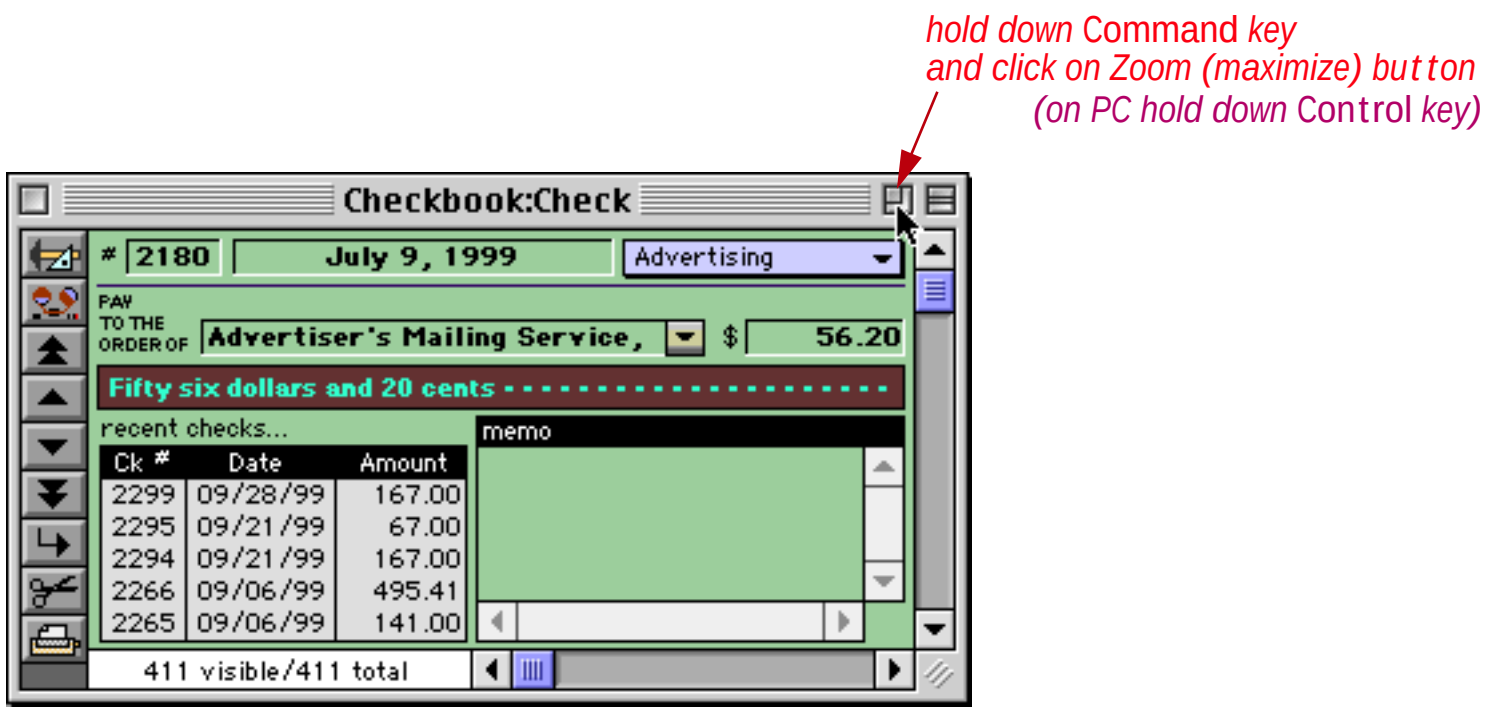
You can set a maximum window size even if the form is not elastic. To do this, create an auto-grow object anywhere on the form. In the configuration dialog, set the maximum dimension and also enable the Don't Adjust Form option. When this option is enabled Panorama does not adjust the objects in the form when the window is resized.

If you want the window to be a fixed size (not expandable) set the minimum and maximum dimensions to the same value.

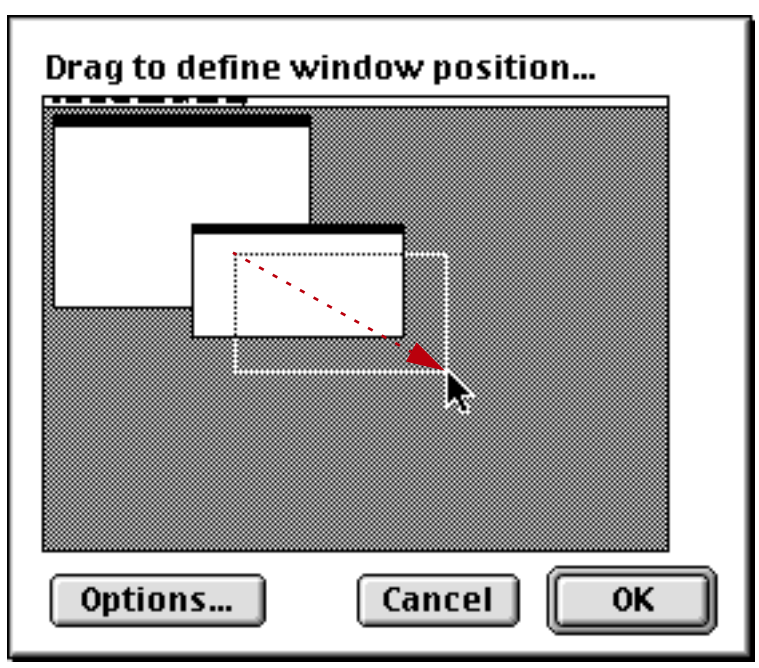
Removing the Window's Scroll Bars

Form windows normally have scroll bars on the right and bottom edges of the window. When using an elastic form these scroll bars really aren't necessary because the form objects are always adjusted to fit inside the visible area. Panorama allows you to remove the unnecessary scroll bars either manually or with a procedure.

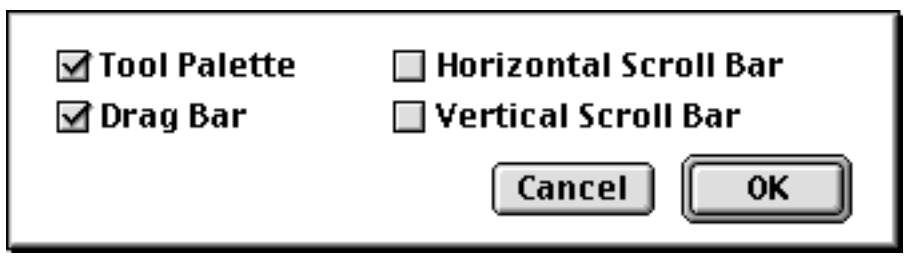
To manually remove the scroll bars, hold down the **Command** key (Macintosh) or **Control** key (PC) while you click on the Zoom box (in the upper right hand corner of the window).



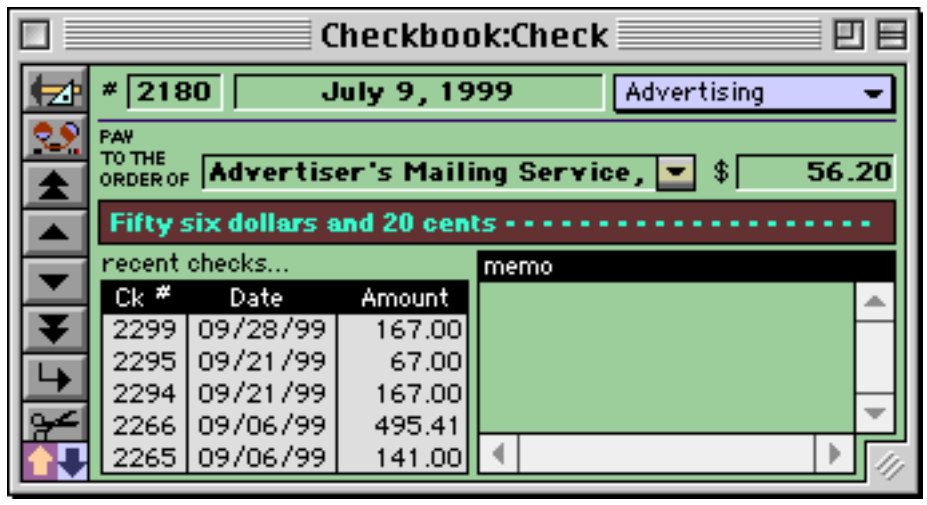
A dialog with a miniature diagram of the monitor appears. Drag the mouse across this diagram to define a new window position.



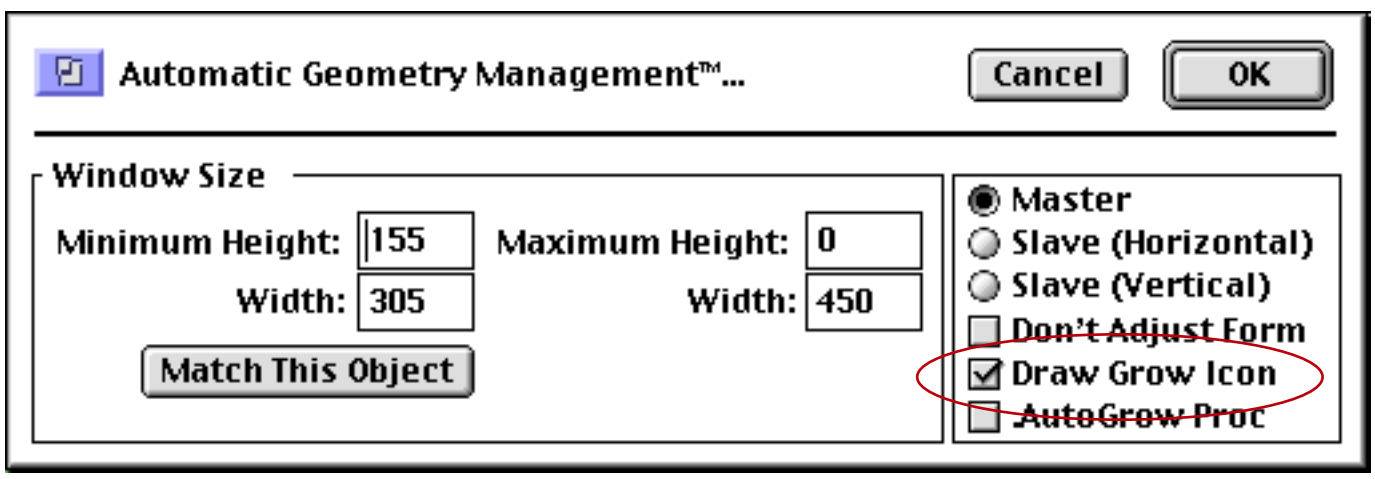
Now press the **Options** button. This opens a second dialog. Using this dialog you can enable and disable four different window options. Normally all four of these options are enabled. In the example below the scroll bars have been disabled.



Press **OK** twice to close both dialogs. The window will be re-displayed without the scroll bars.



Depending on the operating system you are using, the grow box in the lower right hand corner may not appear. To make sure that it appears, open the auto-grow configuration dialog and check the Draw Grow Icon option.



If you save the database with this window open, Panorama will remember to leave the scroll bars off when you re-open the database. However, if you close just this window you will have to repeat this process to turn off the scroll bars when you re-open the window.

The Window Tweak Wizard

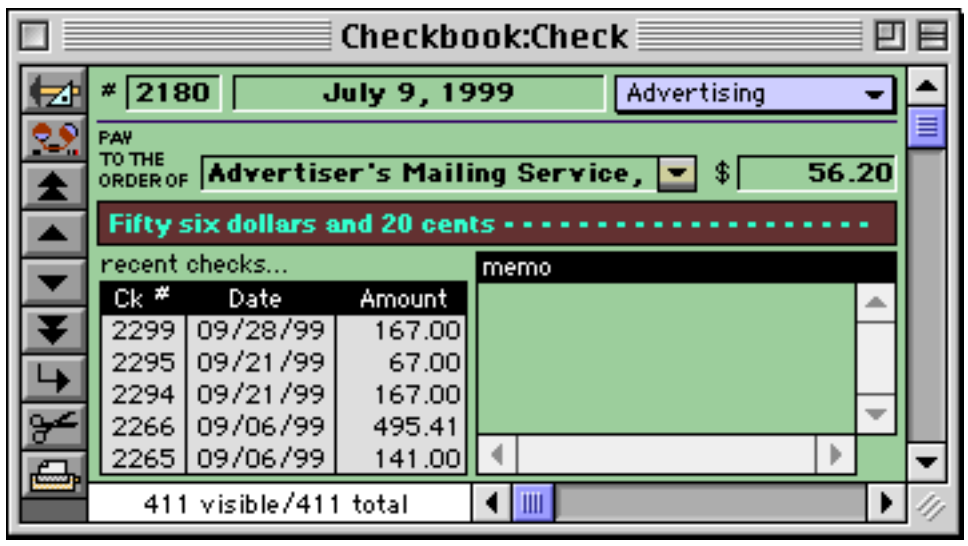
The process described in the previous section gets rather tedious if you wind up repeatedly adding and removing the scroll bars. A simpler method is to use the **Window Tweak** wizard, which you'll find in the Form Tools submenu of the Wizard menu. When you select this wizard it will automatically toggle the scroll bars and tool palette from the current window (if they are currently visible they will be removed, if they are currently disabled they will be made visible.

When you open the **Window Tweak** wizard a small window appears in the lower right hand corner. The buttons on this window allow you to perform additional "tweaks" without using the menu.

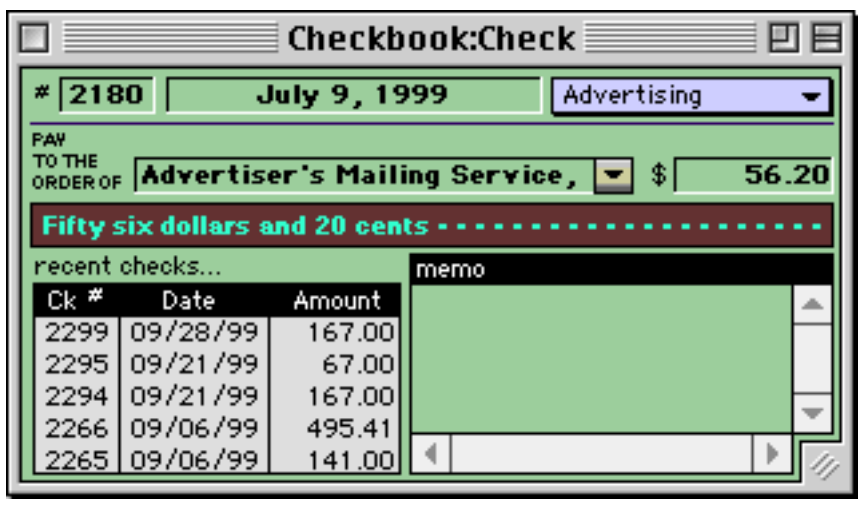


Press the **Tweak** button to toggle the palette and scroll bars of the window just below the Window Tweak window. Using the checkboxes you can modify the "tweak" action to exclude the tool palette or either scroll bar.

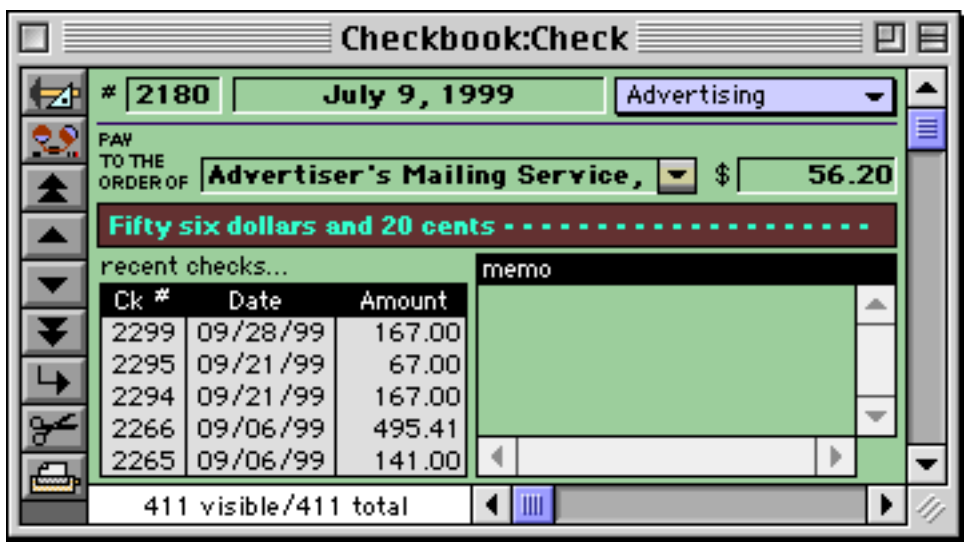
To illustrate the “tweak” action we’ll use this Checkbook database. Here is the original window.



Once the form is set up and ready to go, run the **Window Tweak** wizard. Voila! The scroll bars and tool palette magically disappear, and the window size shrinks slightly to compensate.



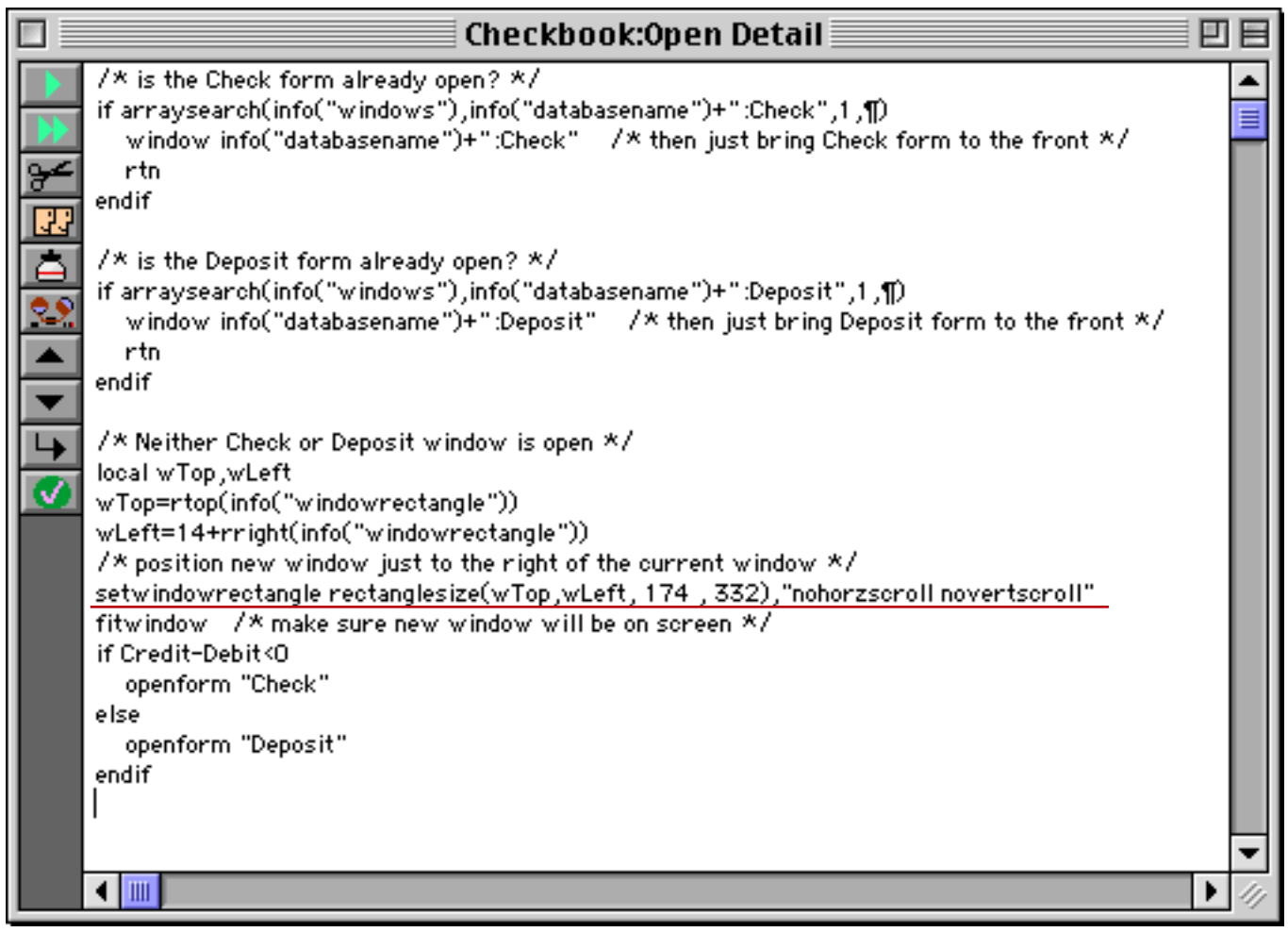
When you need to get the palette and scroll bars back again, run the **Window Tweak** wizard again (or click on the **Tweak** button). Poof! The missing items re-appear (and the window expands slightly).



Each time you run the **Window Tweak** wizard the scroll bars and palette will toggle on or off.

Opening Windows with a Procedure

It's possible to write a procedure that opens a form window with the scroll bars already disabled. Here's such a procedure from the Checkbook database that comes with Panorama.



This procedure automatically opens either the Check or Deposit form, depending on the type of transaction contained in the current record. The key statement in this procedure is `setwindowrectangle`. This statement tells Panorama the dimensions of the new window, and also tells it not to include horizontal or vertical scroll bars in the new window. For more information on this statement see "Specifying the New Window Location" on page 1573.

Modifying an Elastic Form

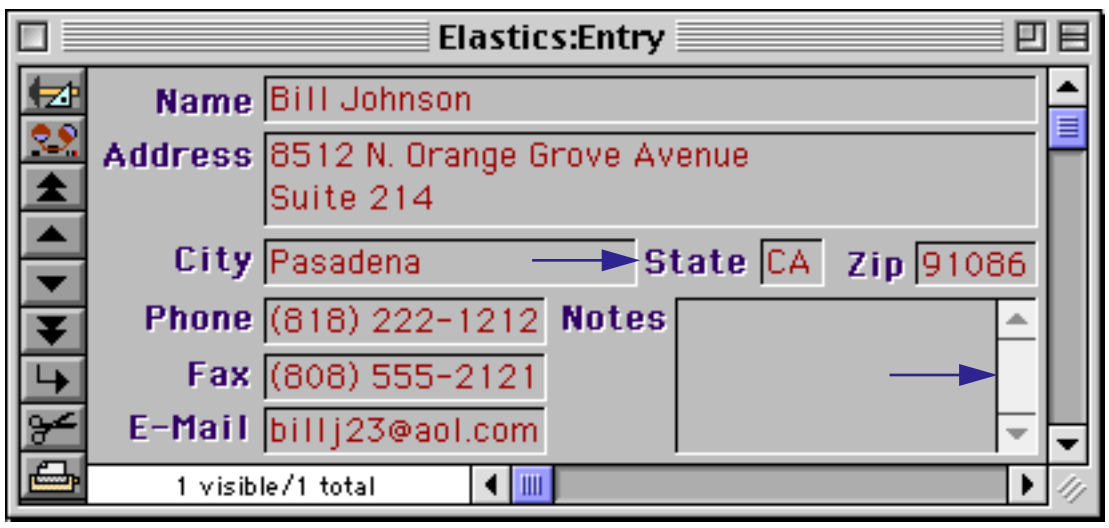
If you need to change an elastic form, we recommend that you start by resizing the window to its minimum size. Once the window is at the minimum size, you can switch to Graphics Mode and then expand the window (if necessary). In Graphics Mode the window can be expanded without automatically adjusting the form.)

If you change the position or size of the Auto Grow object, you must re-set the minimum window size. Simply double click on the object, then click on the **Match This Object** button, just as you did before. The dialog will show the new minimum window size. Click on the **OK** button to permanently set this size.

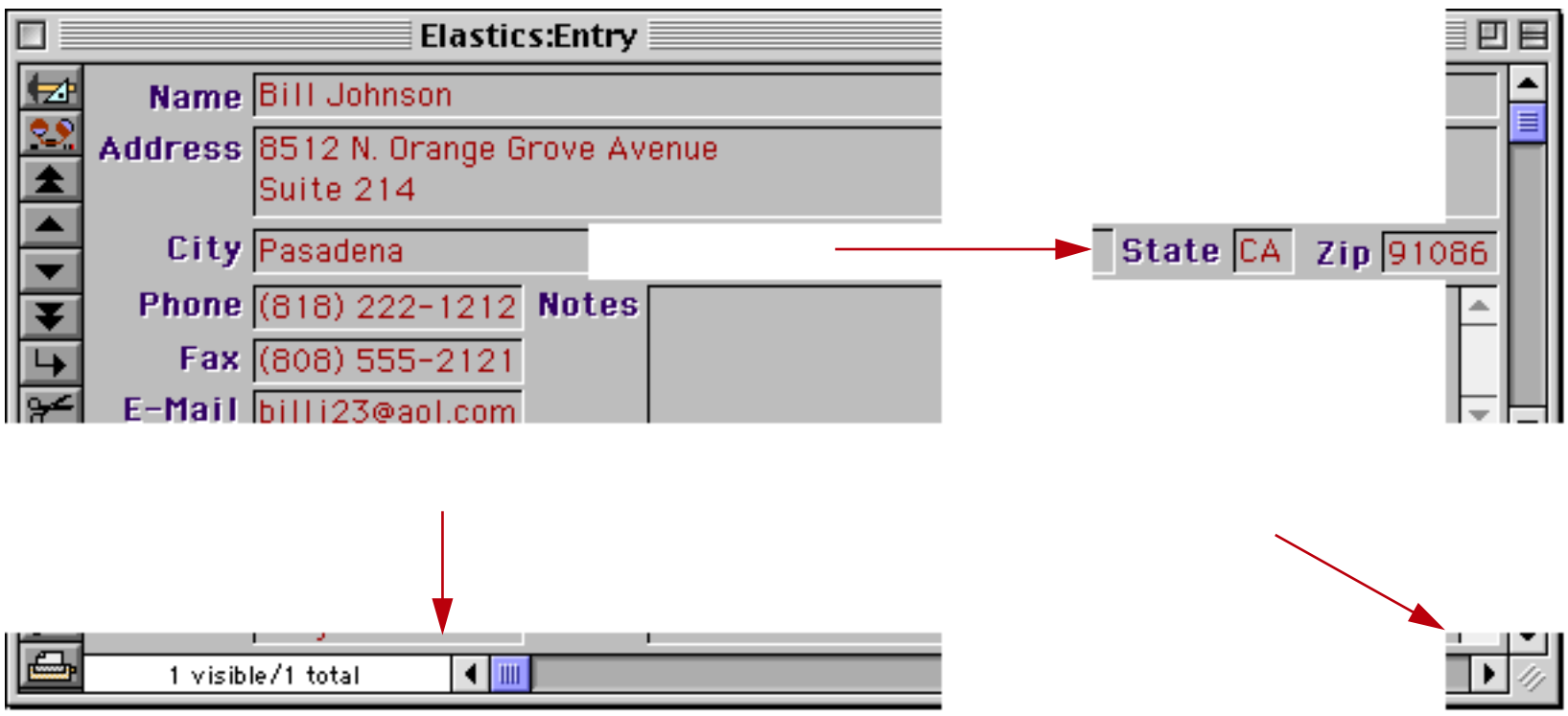
Non-Rectangular Quadrants

Sometimes it is not possible to position the intersection point of the four quadrants so that all the objects in your form expand and shrink the way you want them too. In that case you can add “extensions” to the upper right and / or lower left quadrants that stick out into the fixed quadrant. The area covered by these extensions will adjust when the window is resized. These extensions are created by “slave” Auto Grow objects. These objects are called slaves because they always follow exactly what the “master” Auto Grow object does. An extension to the upper right hand quadrant is called a horizontal extension because it sticks out horizontally into the fixed quadrant. This is the most common type of extension. It allows different lines of a form to expand differently. A vertical extension sticks out from the lower left quadrant into the fixed quadrant.

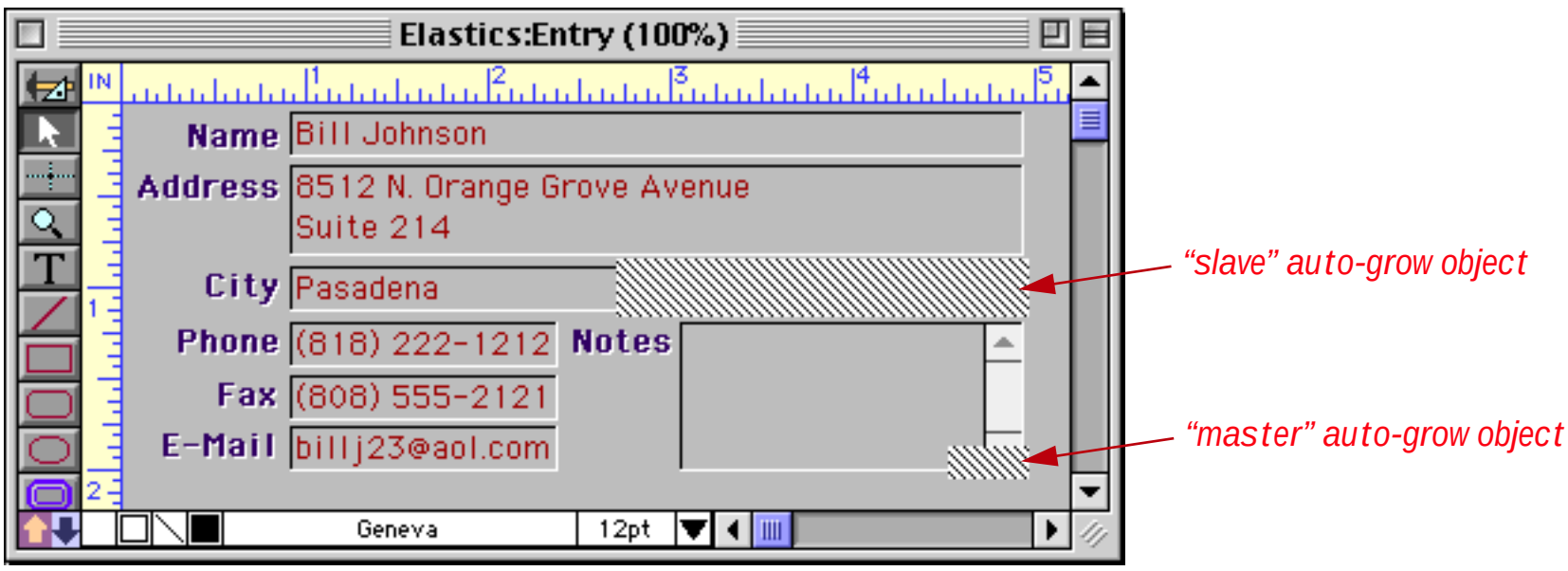
Here is an example of a form that requires an extension. When this form expands we would like both the Notes and the City fields to expand.



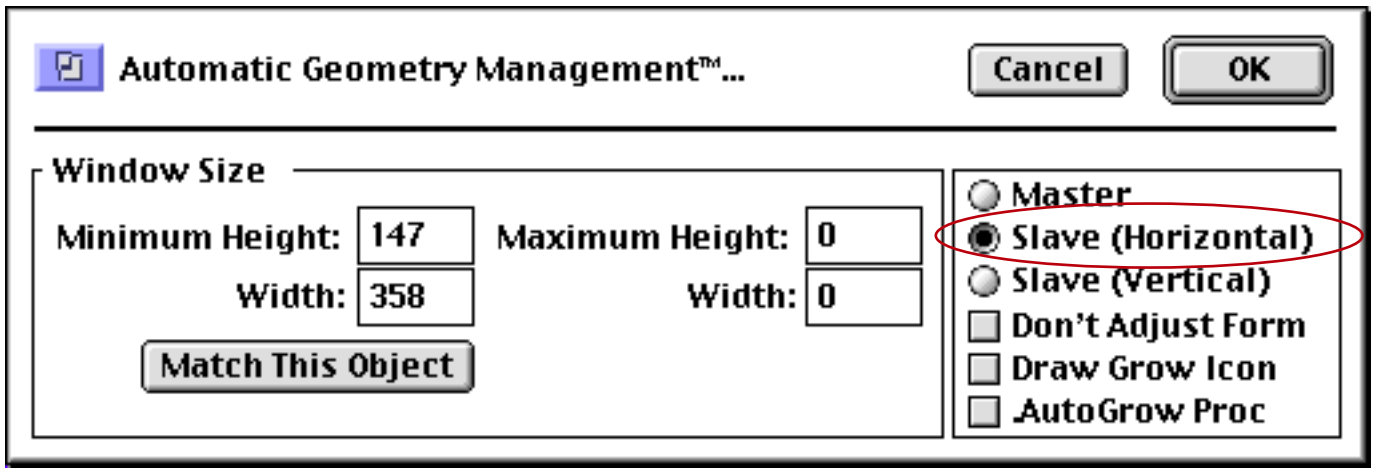
There’s no way to divide the form into quadrants and have both of these objects expand. Instead, an extension is needed, like this.



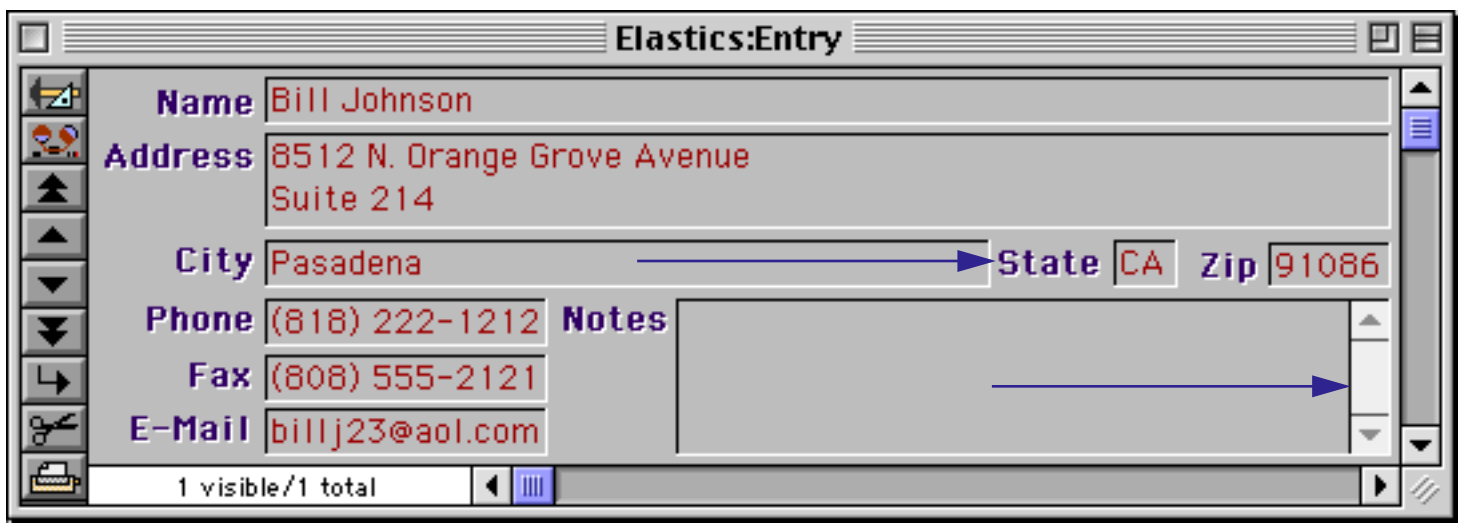
To create an elastic form with an extension like this, start by creating a normal auto-grow object in the lower right hand corner. This is the **master**. Then add a second auto-grow object over the area of the extension. This second auto-grow object is a **slave**. Make sure that the slave object extends past the edge of the fixed quadrant into one or more of the movable quadrants.



Here is the configuration dialog for the slave auto-grow object shown above. A slave may be horizontal or vertical. In this case the extension area will slide left and right so the extension is horizontal.

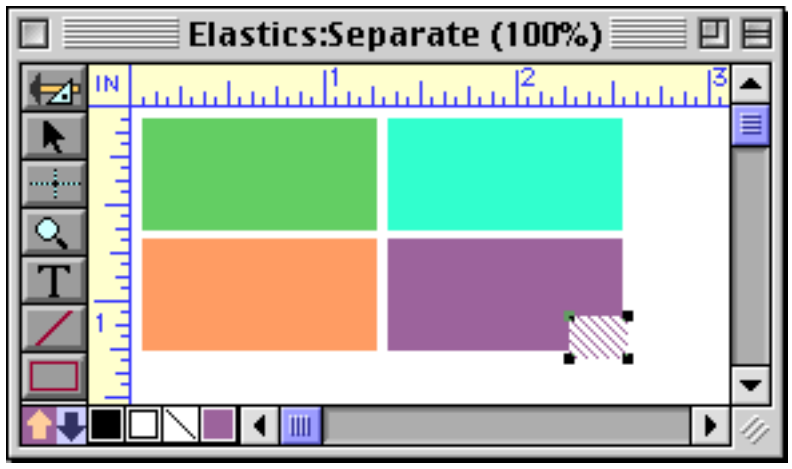


You can add as many slaves as necessary to your form. Once they have been added the form will adjust automatically with the extensions, like this.

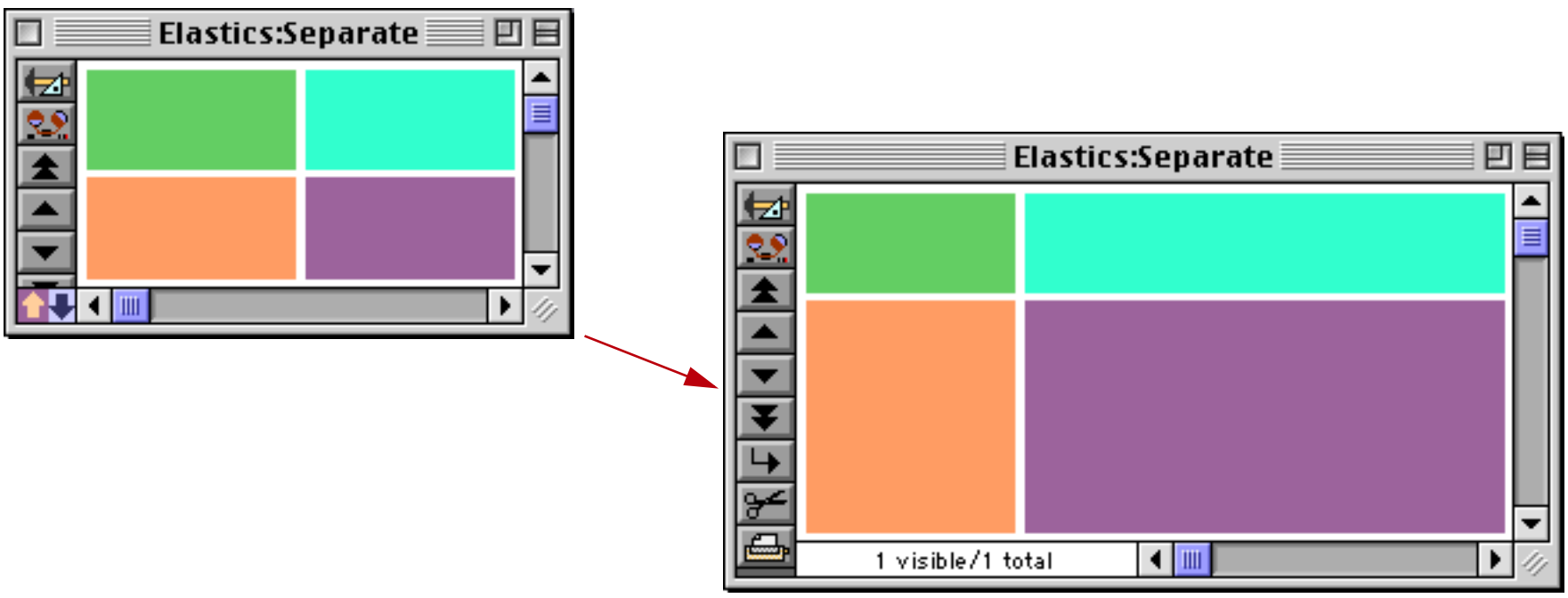


Expanding Multiple Objects Proportionally

Panorama normally adjusts each object separately, either by expanding them or sliding each object. For example, consider the form shown below.

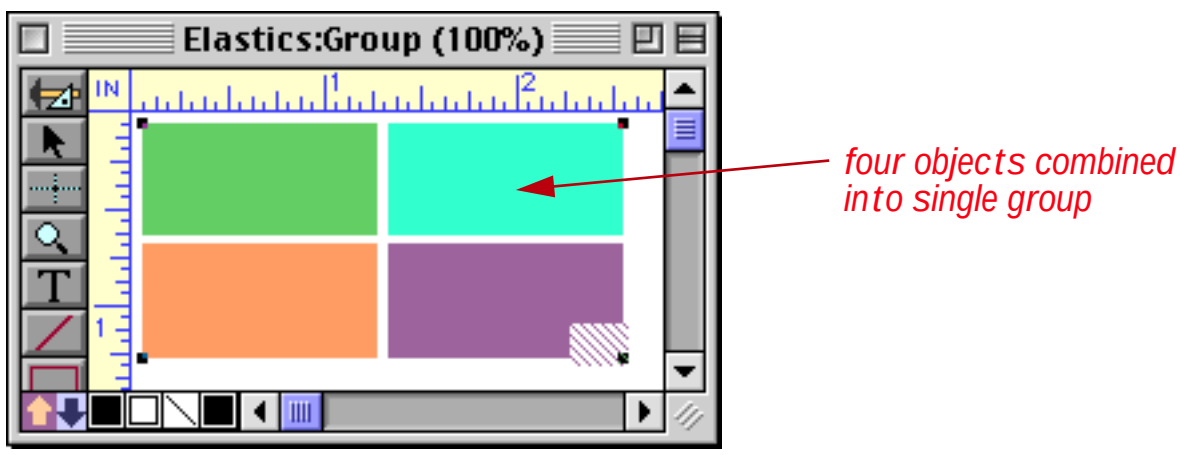


This form expands like this.

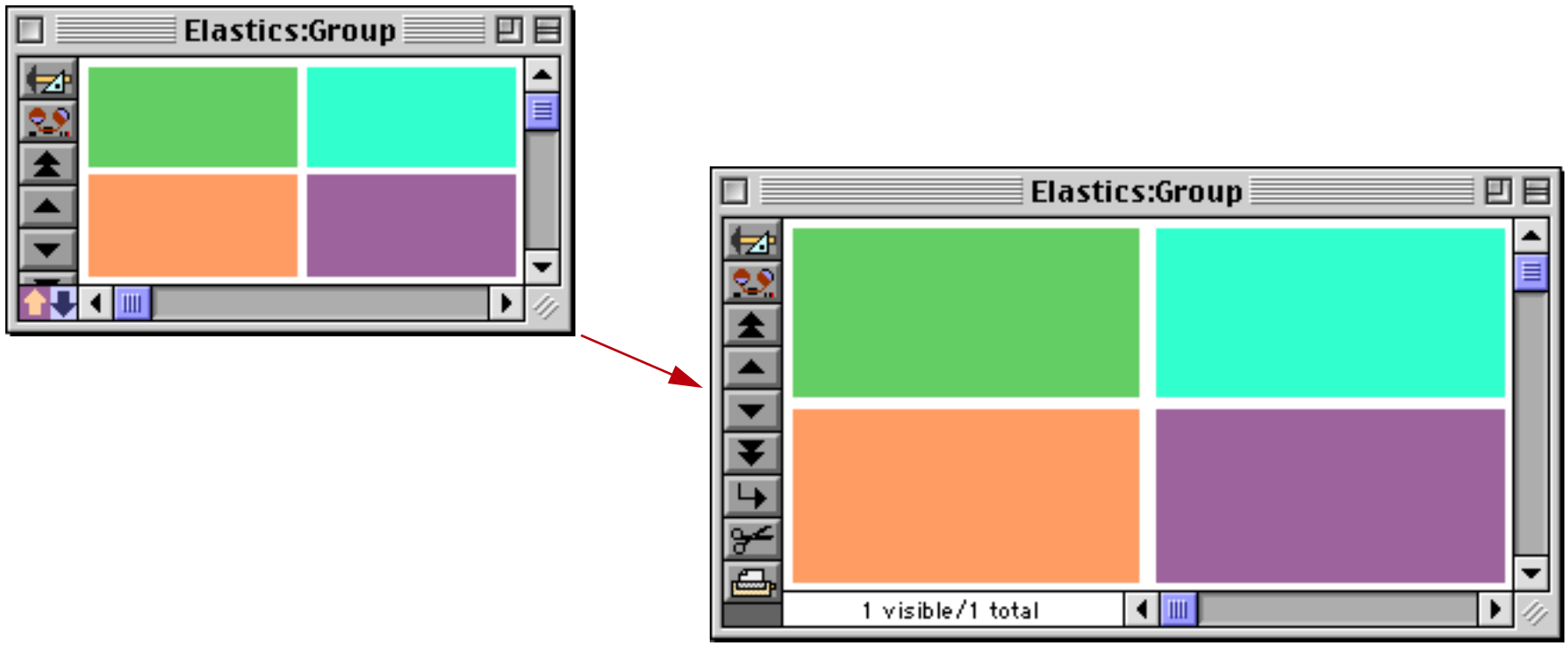


Sometimes, however, you may want several objects to expand or shrink proportionally as a group. For example, if the window grows by an inch, you want four objects to grow equally by 1/4 inch.

One way to do this is by grouping the objects (see “Grouping Objects Together” on page 536).



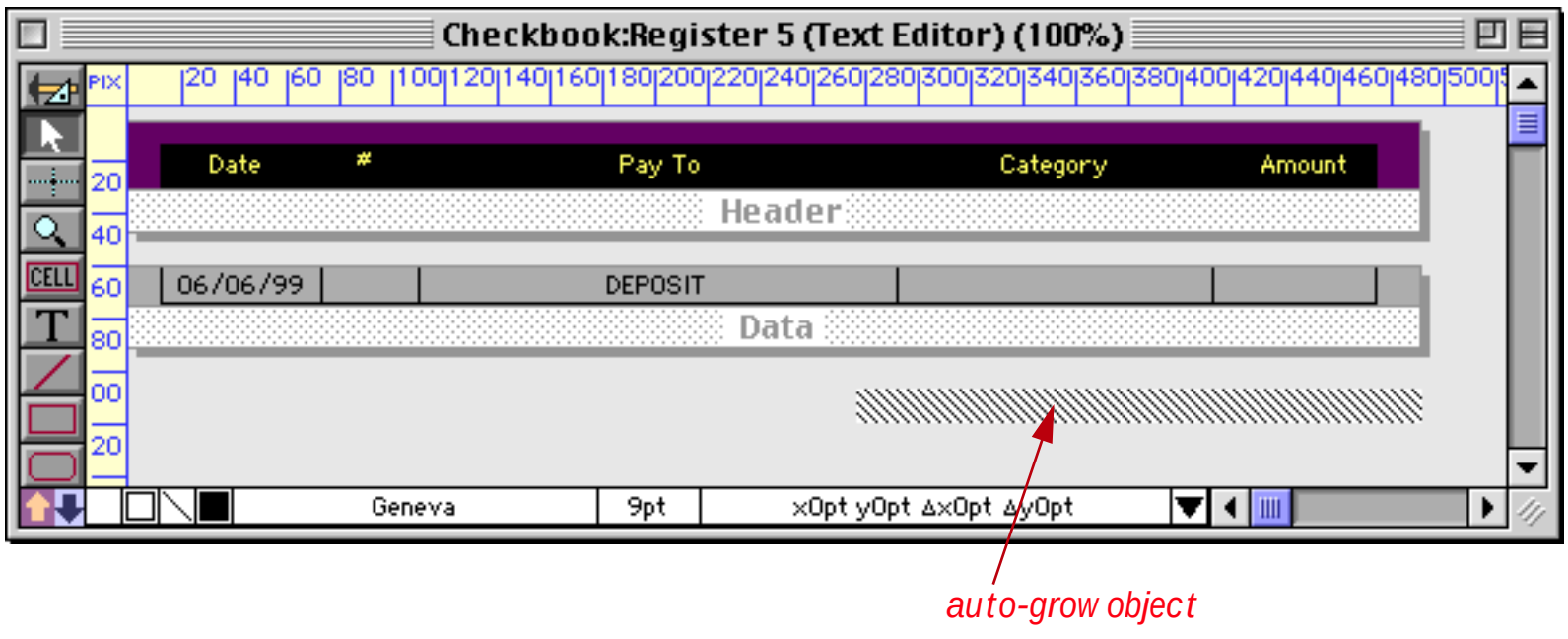
The form adjusting mechanism treats the group as a single object. Within the group, each object will expand or shrink proportionally as the entire group expands or shrinks. Notice that the gaps between the objects also expand proportionally.



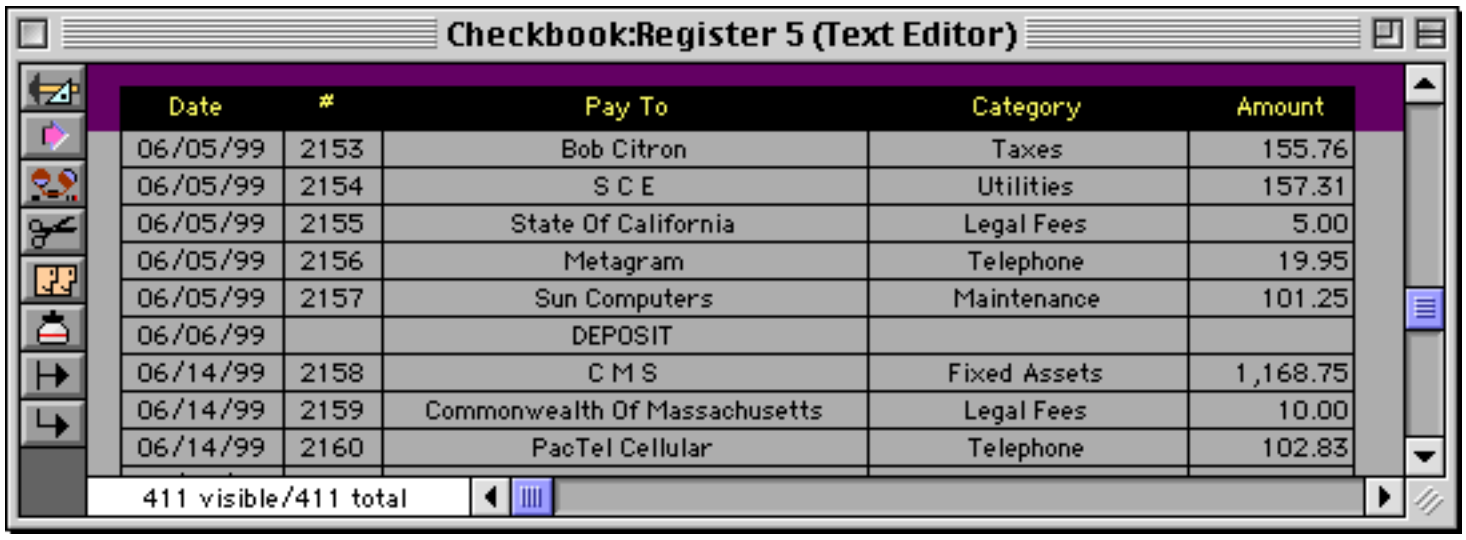
Another method is to use a Matrix SuperObject (see “Super Matrix Objects” on page 929). Again, as the entire matrix expands or shrinks, the individual cells in the matrix expand or shrink proportionally. This method is ideal for calendars (see “Building a Calendar” on page 960). If you use the Text Display SuperObject to display text, the text can automatically increase or decrease in size as the form expands and shrinks (see “Text Display Options” on page 611).

Elastic View-As-List Forms

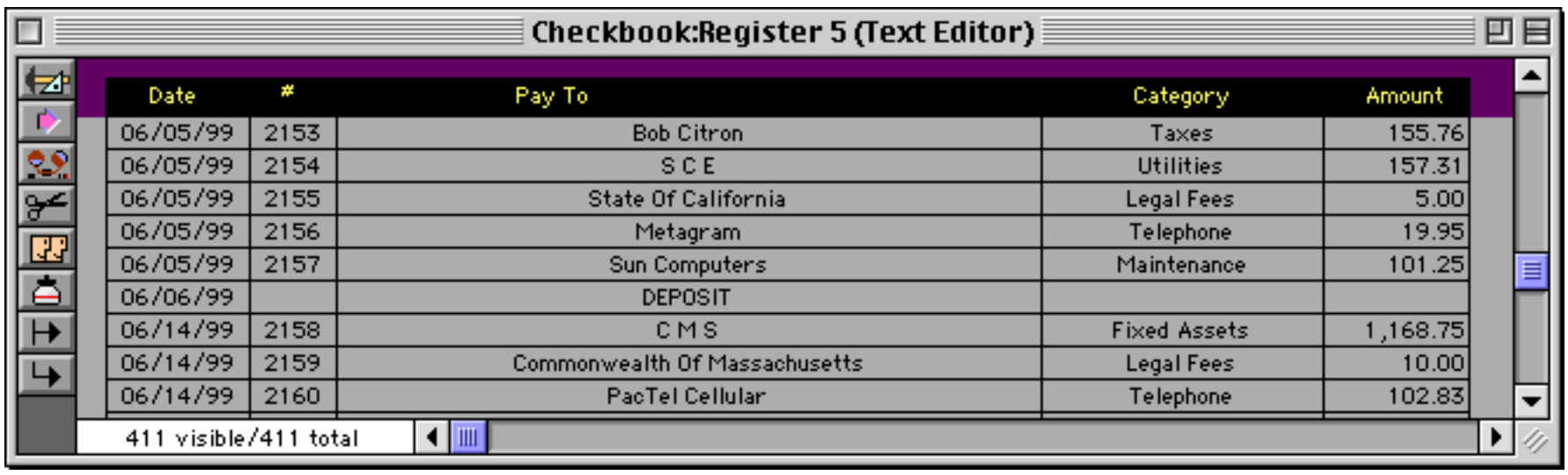
Starting with Panorama 4.0 it is possible to create an elastic view-as-list form. In this case the vertical position of the auto-grow object is unimportant. The horizontal position of the object should correspond to the right edge of the visible area. It’s easier to set this up if the data tile is aligned with the left edge of the form. In the example below the width of the Deposit field will increase when the form is expanded.



Here's what this form looks like in a minimum width configuration.



When the window is expanded, the form adjusts automatically.



You can also use the auto-grow object to set the minimum and maximum dimensions of the view-as-list form, just as with a regular form.

Super Matrix Objects

Some applications require a rectangular array (or matrix) of data, pictures, and / or pushbuttons (for example a monthly calendar or a thumbnail artwork preview).

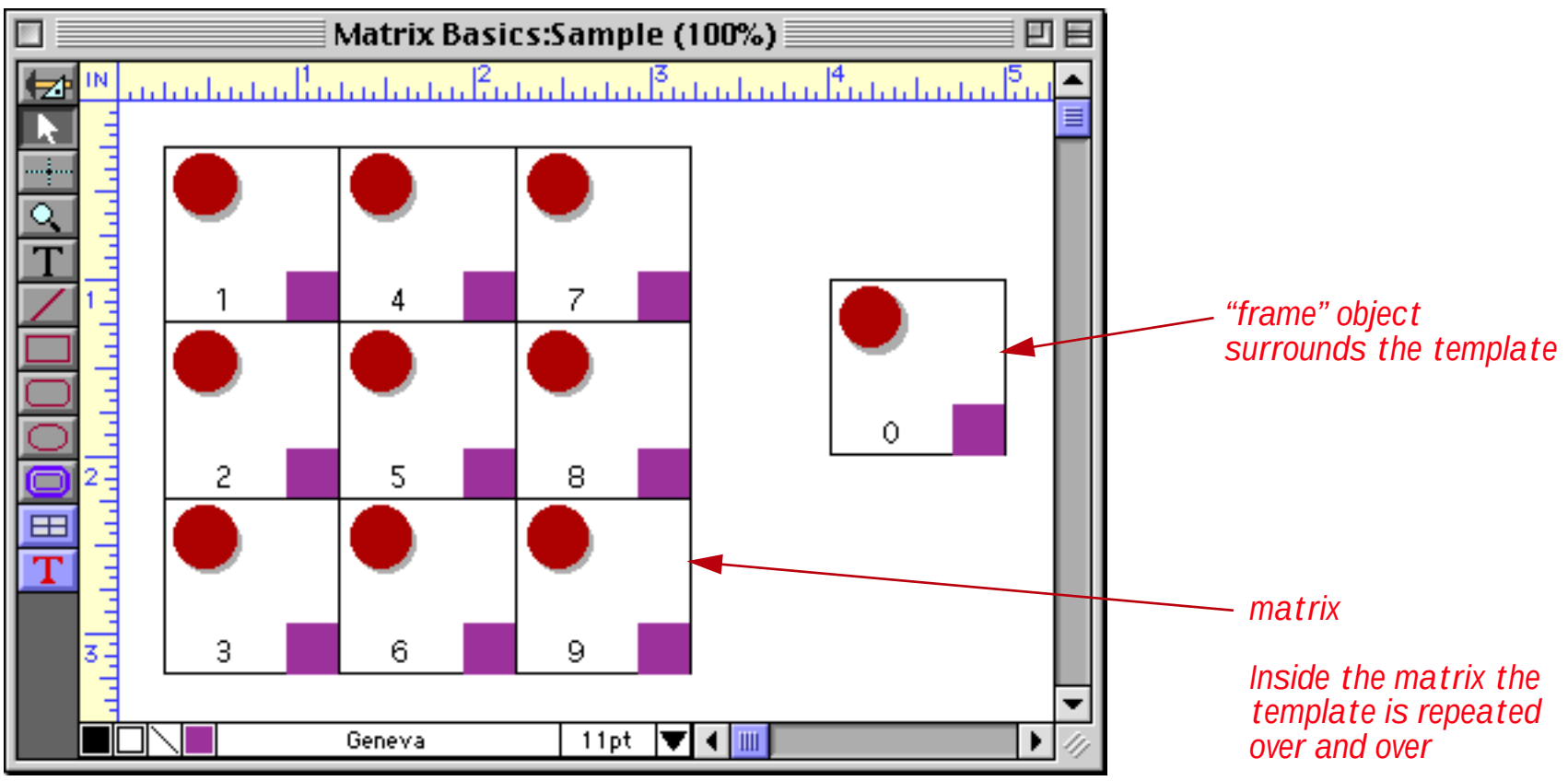


Such a matrix consists of a series of cells assembled into rows and columns. Of course, you can build up such a matrix from individual objects, but the Super Matrix SuperObject™ allows the entire matrix to be built by repeating a single template. The template may contain artwork and text that will be repeated over and over again for each cell in the matrix. The template can use formulas to display the appropriate information in each column and row. This system has several advantages: 1) The array can be built quickly, 2) If changes are necessary later, they only have to be made once in the template and are automatically repeated throughout the entire matrix, 3) It is very easy to change the number of columns or rows in the matrix, 4) The matrix and template can be constructed to adjust automatically as the window changes size and shape.

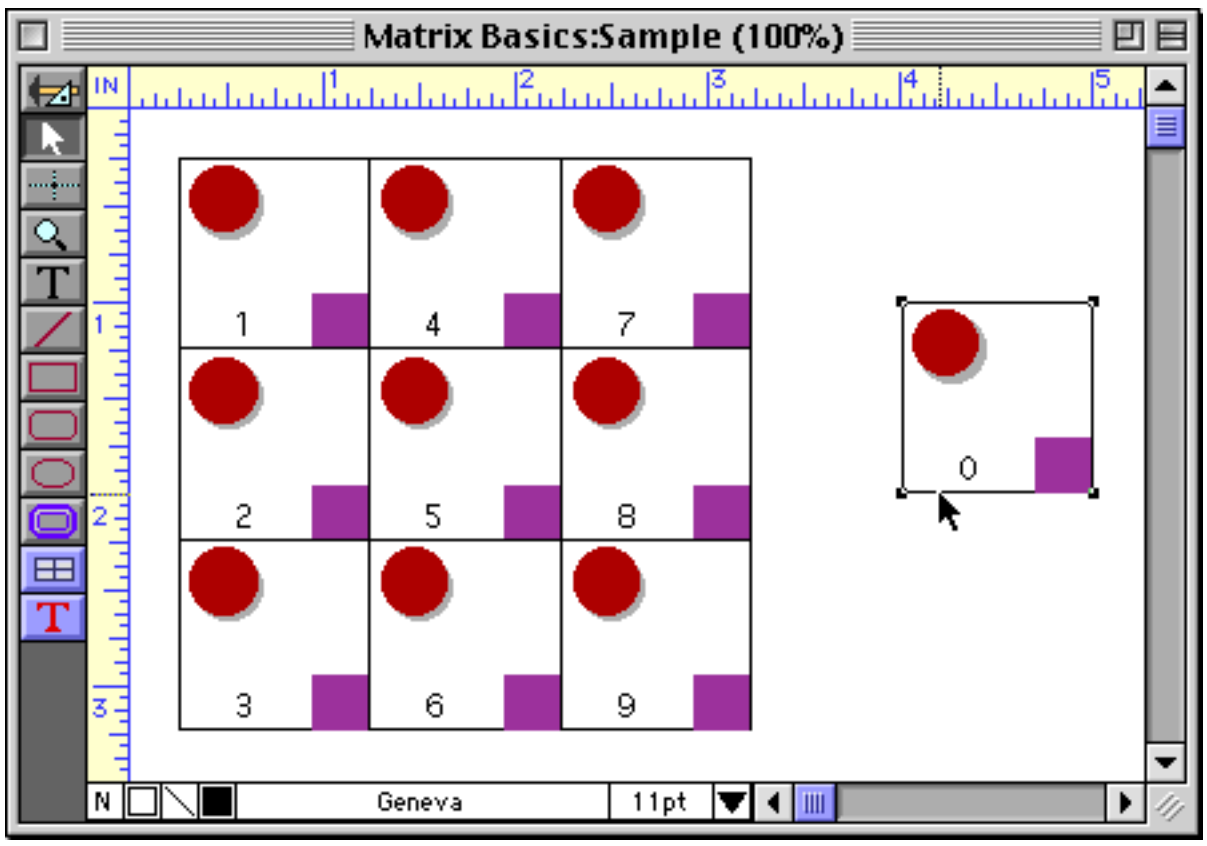
(Note for Panorama 3 and 4 users: The Matrix Data, Scroll Bar, and Overflow options in the Super Matrix configuration dialog only work for new super matrix objects that are created with Panorama V or later. If you have an old Super Matrix object that was created with an older version of Panorama you'll need to delete it and create a new object to use these features.)

The Matrix Template (and Frame Object)

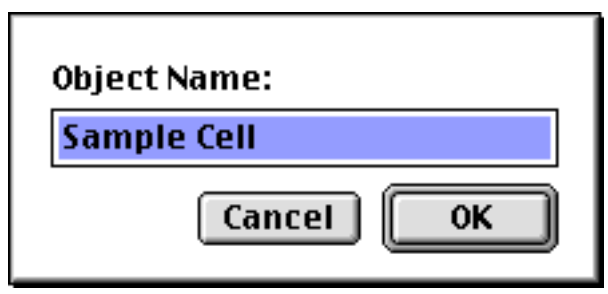
Unlike most other objects, the Super Matrix is not completely self contained. For non-trivial applications, it requires a template that tells Panorama what to draw within each matrix cell. This template consists of Panorama objects enclosed within an object you designate as the frame object. We call the object a frame object because it surrounds the objects inside, just like a picture frame. Any object that is inside the frame object will automatically appear inside the cells of the matrix. The frame is usually placed off to the side or below, somewhere where it will not normally be visible.



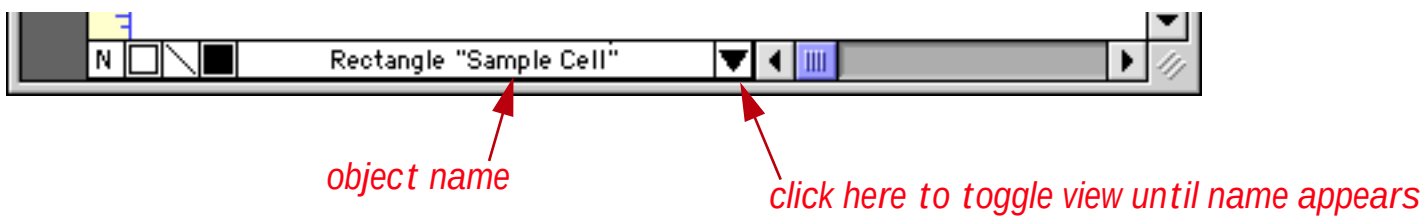
To designate an object as a frame object, you must give it a unique name. To give an object a name, first select the object (frame objects are usually rectangles, but any kind of object will work).



Next, use the **Object Name** command in the Edit menu or click on the object name in the Graphic Control Strip (along the bottom of the window). See "Object Type / Object Name" on page 533 for more details about setting and viewing an object's name. The name you pick is unimportant, but keep it handy, because you'll need it when you create the matrix itself.



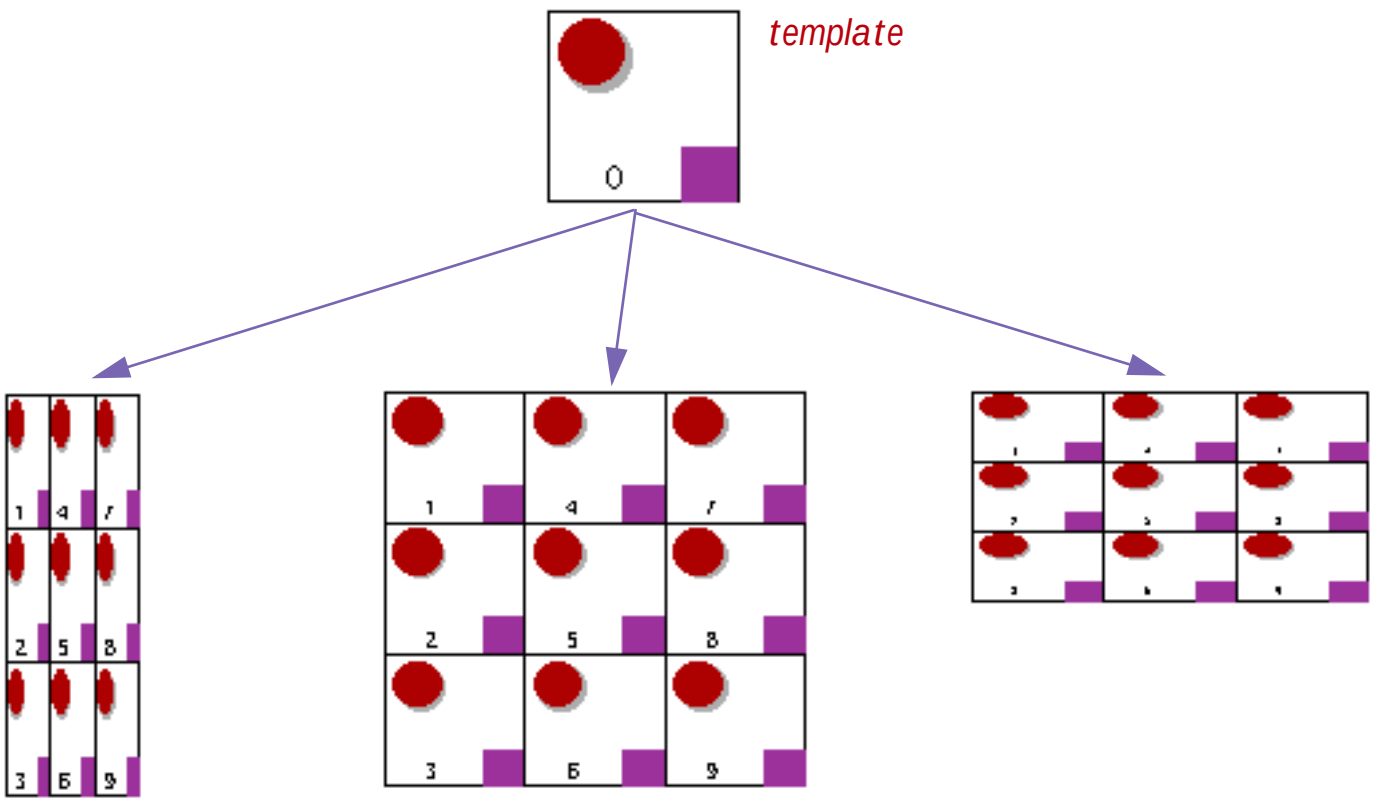
If you later forget the name of the object you can use the Graphic Control Strip to remind you.



The frame object may be located anywhere on the form, and may be any size you want. Usually it is most convenient to make it about the same size as the cells in the matrix will be. You can move the frame object around at any time, but remember that only objects inside the frame object will appear inside the cells of the matrix.

In some ways, the matrix frame object is similar to a report tile object. Both are used to designate a template built up with other objects. However, there are some important differences. First of all, the tile is a special kind of object, while any object may be designated as a matrix frame.

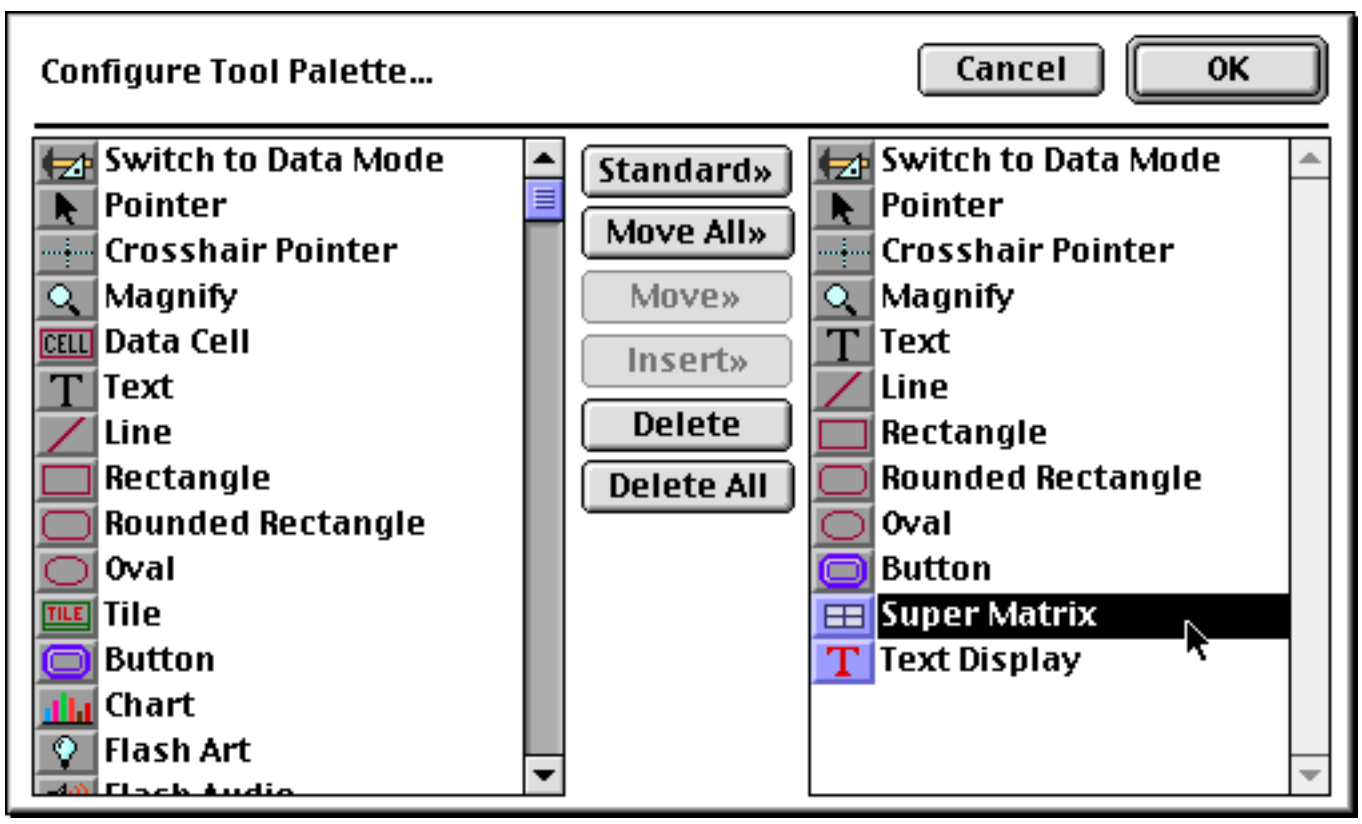
A more important difference is that the tile object always prints at a fixed size and proportion, no matter what the size of your paper is. The matrix frame, however, will be squeezed and adjusted as necessary to fit into the actual matrix cells. If the matrix cells are tall and skinny then the matrix frame and template will be sqeezed to fit—even if in graphics mode the matrix frame appears to be short and wide.



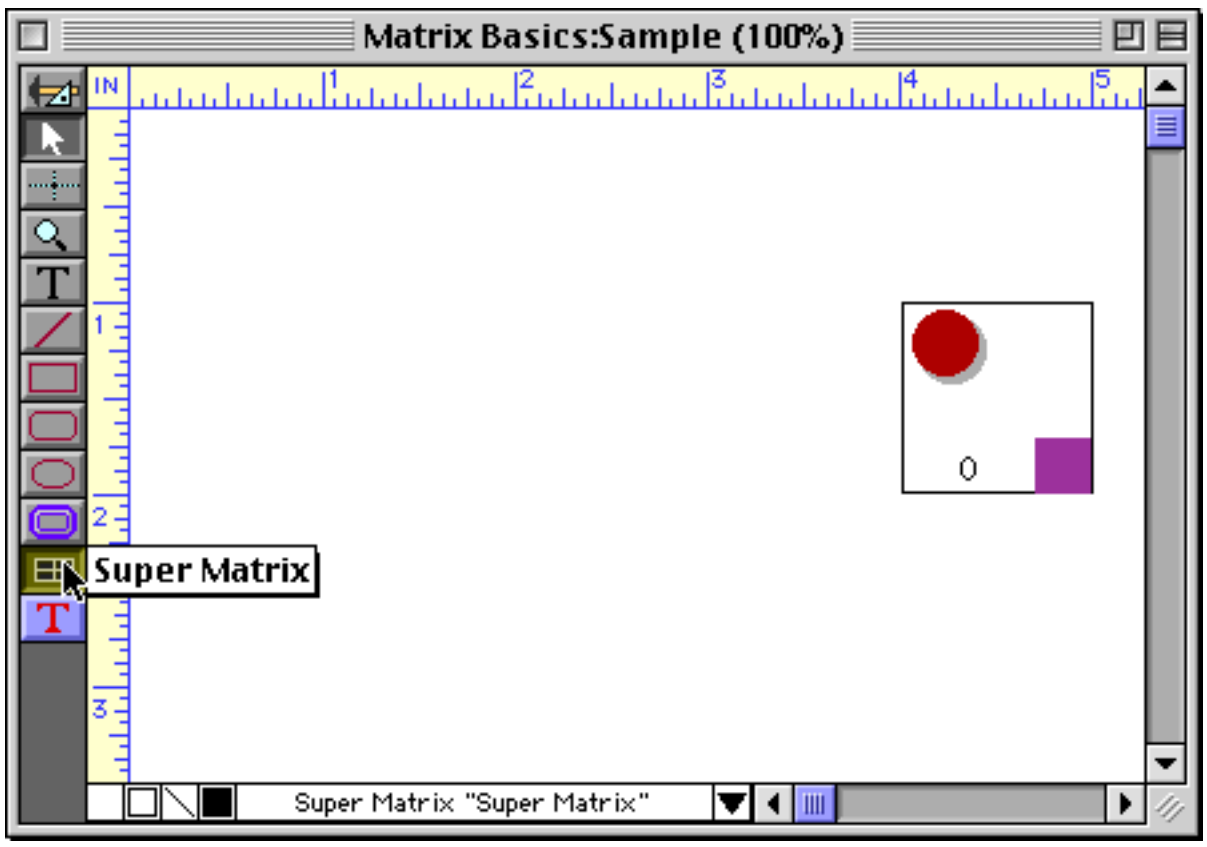
If you know the exact size your matrix cells will be, make the matrix frame the same size and you won't have to worry. But if the size of the matrix cells may change, you'll have to design your matrix template to adjust as the matrix changes size and shape. Techniques for making adjustable templates are discussed later in this section.

Creating Super Matrix Objects

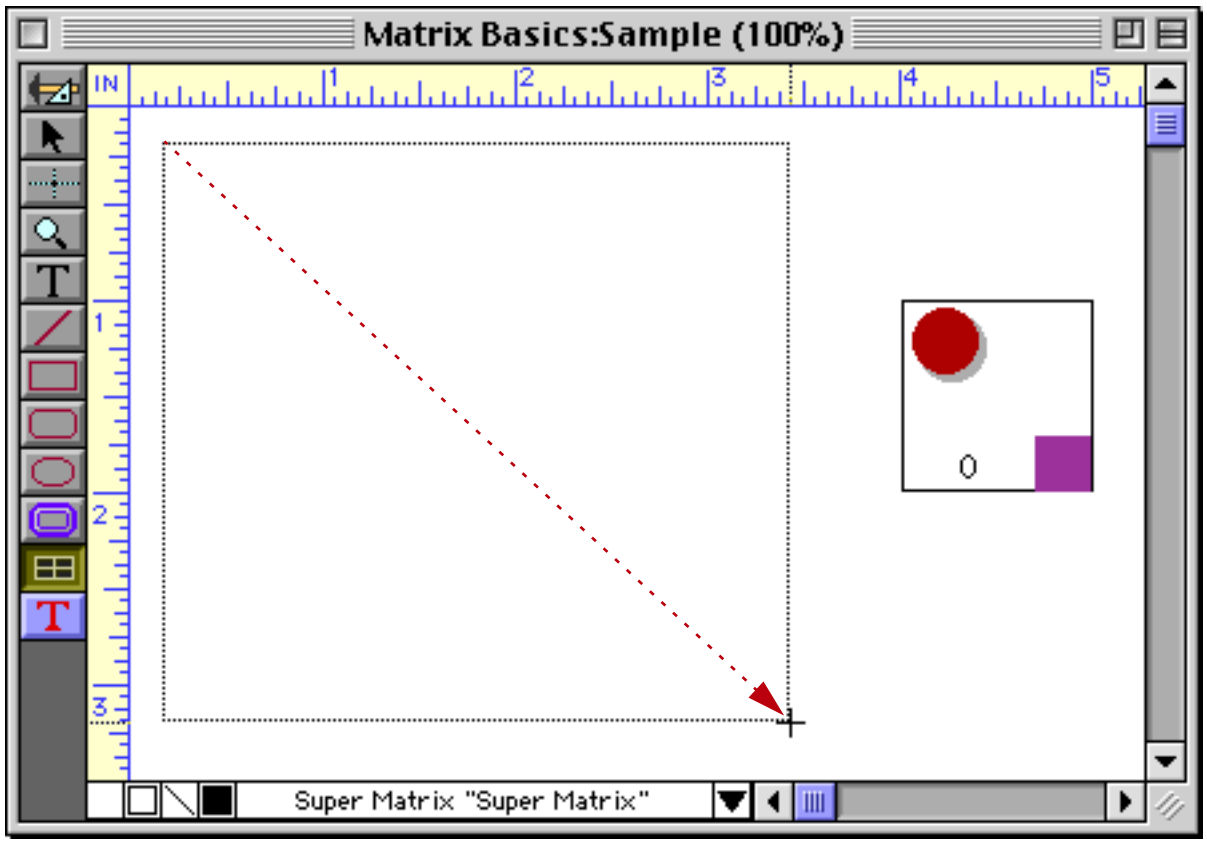
Super Matrix objects are created just like any other SuperObject™. The Super Matrix tool is not in the default tool palette, so you'll need to use the **Tool Palette** dialog to add this tool to the palette if it is not already there (see "Customizing the Tool Palette" on page 497).



Now that the tool is added to the palette you can select it.



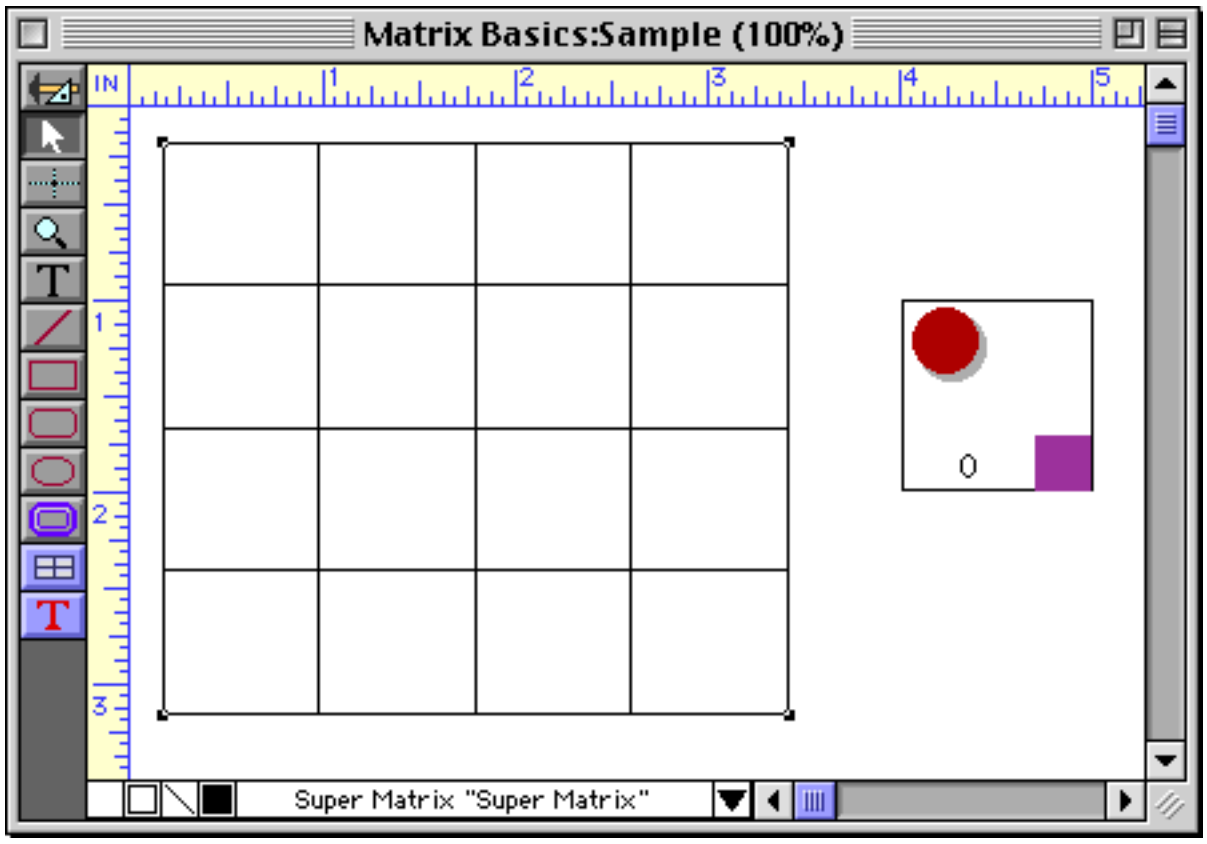
Once the tool is selected, drag the mouse across the form in the location where you want to create the matrix to appear.



When you release the mouse, the Super Matrix configuration dialog will appear.



For a blank matrix just fill in the number of columns and rows and press the **OK** button.

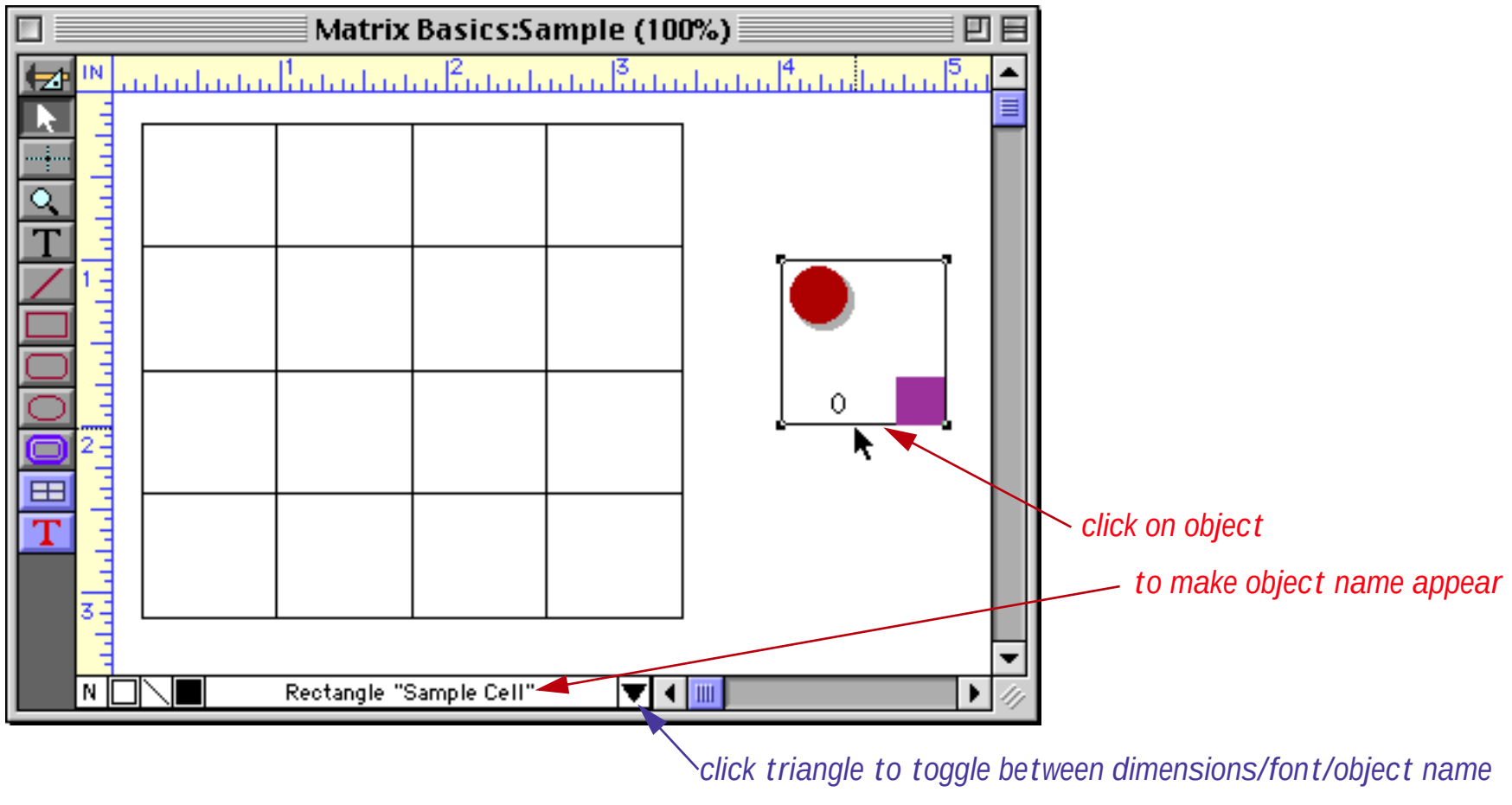


If you need to change the matrix options later, double click on the matrix object with the **Pointer** tool to re-open the configuration dialog.

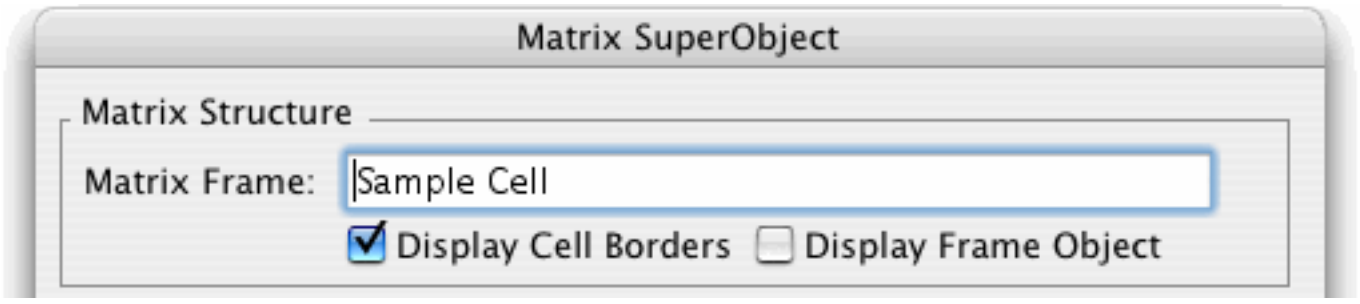
Linking with the Matrix Frame

The first option in the Super Matrix dialog is called Matrix Frame. This is the name you gave to the frame object described earlier (see “The Matrix Template (and Frame Object)” on page 930). If you haven’t created this object yet, don’t worry. Just type in the name you intend to use here, then later go back and create and name the frame object and template.

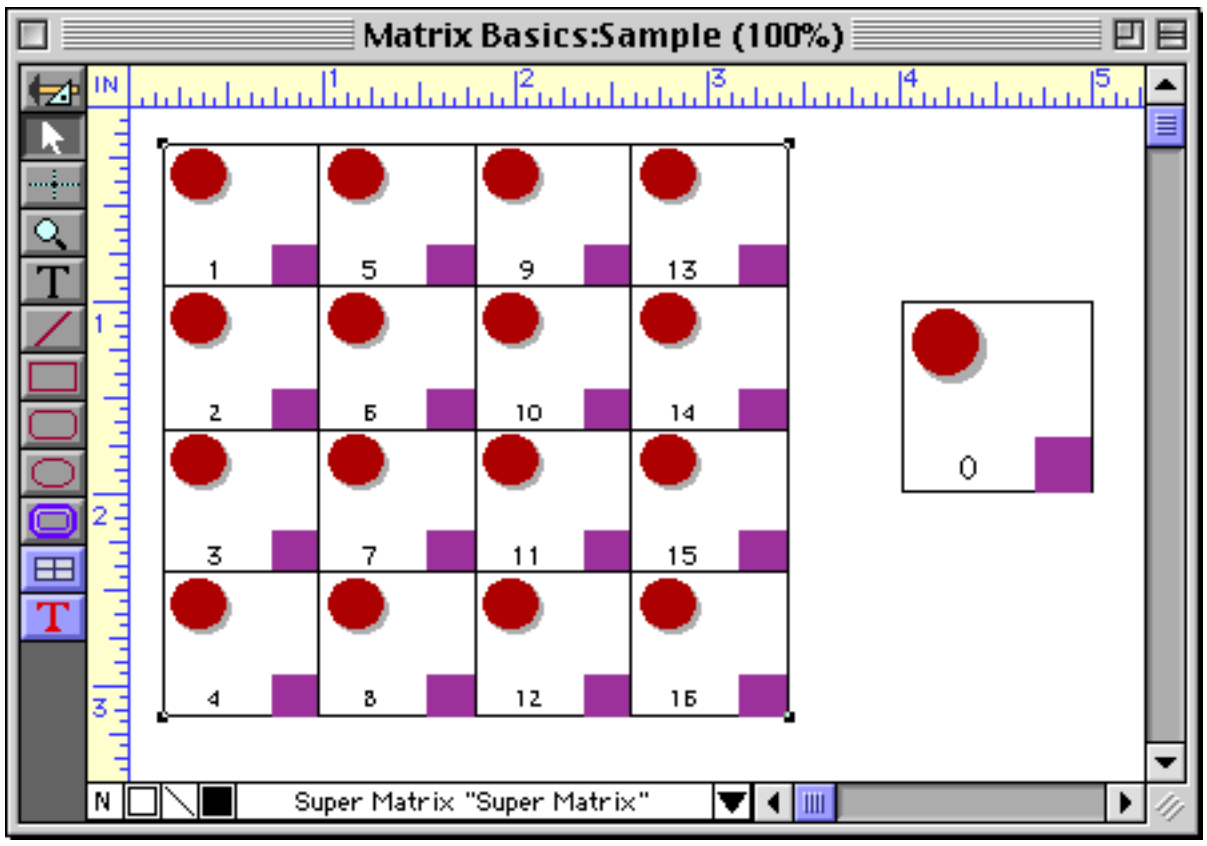
As you may recall, in our example we named the frame object Sample Cell. You can double check the name by clicking on the frame object (see “Object Type / Object Name” on page 533).



Now that we've double checked on the name of the frame object, double click on the Super Matrix object to open the configuration dialog. Type in the name of the frame object, Sample Cell.



When you press the OK button the frame is linked to the matrix. The objects in the template will be repeated inside each matrix cell.



It's possible to have more than one Super Matrix object in a single form. In this case, usually each one will have its own matrix frame with a unique name. The unique name is important so that Panorama can tell which frame belongs to which matrix.

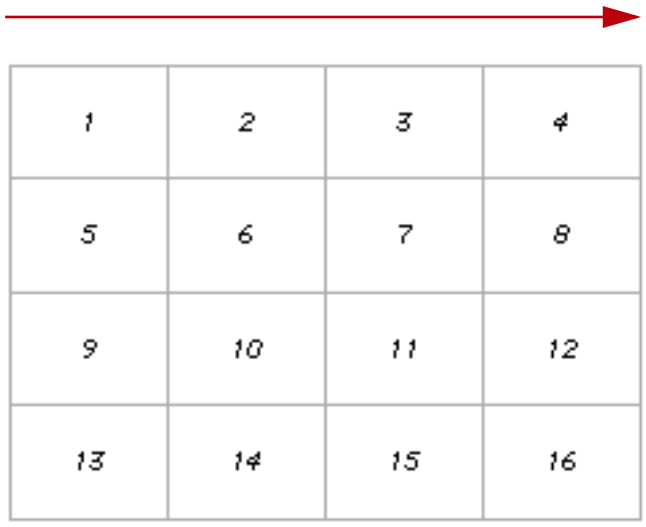
Matrix Cell Borders & Background

If you turn on the Cell Borders option in the Super Matrix dialog, Panorama will display a border around each cell in the object. You can control the color, line width, and pen pattern of the border using the Graphic Control Strip or the Graphics menu. (If you would like a dotted border, use a diagonal stripe pen pattern.) Panorama will adjust the borders of adjacent cells so that they overlap (instead of creating a double width border). The amount of overlap is adjusted according to the width of the line. (Note: This overlap feature does not work with hairlines.)

The Frame Object option controls whether or not the frame object itself is displayed as part of each matrix cell. Usually you'll leave this option off. However, you might want to use this option to display a background for each cell, perhaps using a Flash Art object as the frame object. You could also display a border around each cell using the frame object, but in this case Panorama will not make adjacent cells overlap.

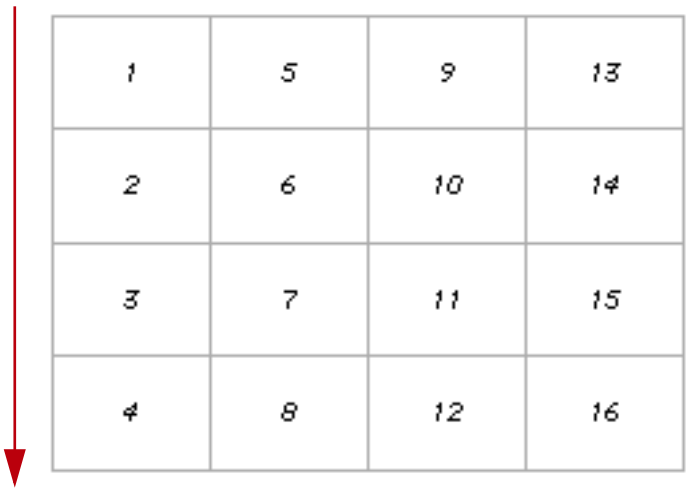
Matrix Order

Each matrix cell is numbered, starting from 1. The matrix can be drawn one of two ways: vertically or horizontally. In both cases the upper left hand cell is number 1. If the matrix order is horizontal, then the cell numbers will be consecutively numbered from left to right in each row.



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

If the matrix order is vertical, then the cell numbers will be consecutively numbered from top to bottom in each column.

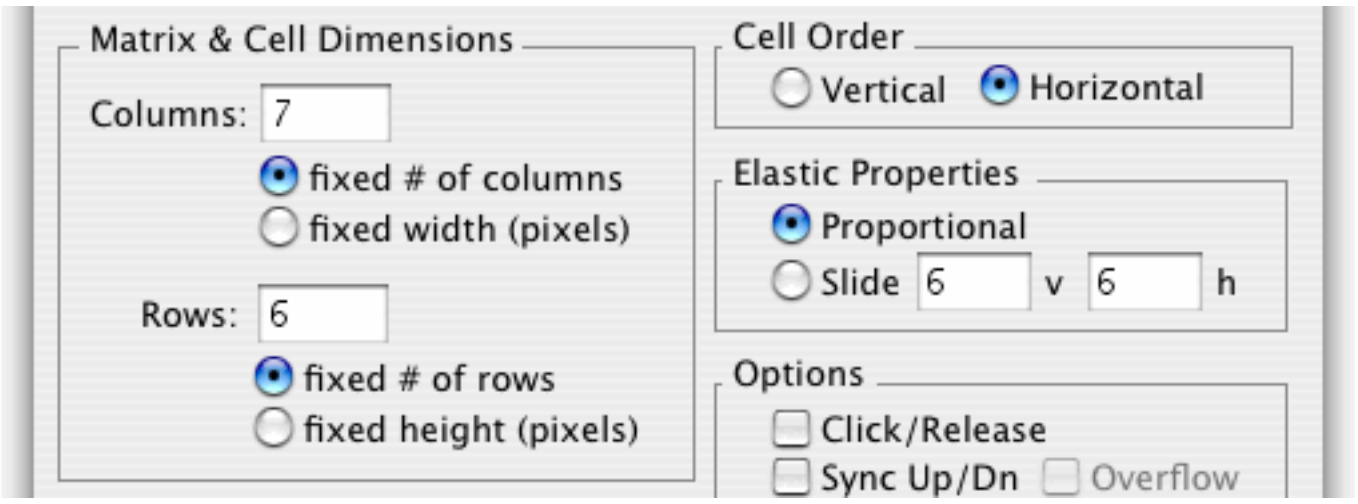


1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Matrix Rows and Columns

The Super Matrix dialog allows you to control how many rows and columns your matrix has. You can choose either variable sized (fixed #) or fixed size rows and columns.

For example, suppose you wanted to build a monthly calendar. In that case you would always want 7 columns and 6 rows, so you should choose the fixed # of columns and fixed # of rows options.

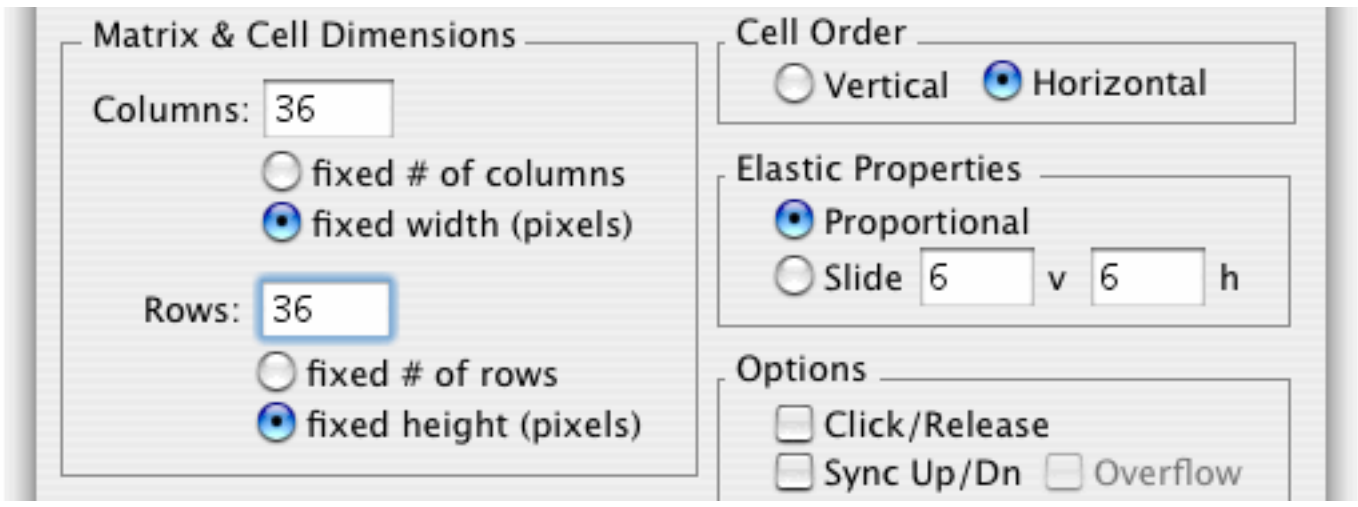


This matrix will always have 7 columns and 6 rows, but the size of the rows and columns will change if the matrix changes size.

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42

In another application you may want to display icons in a matrix with cells that are always 36 pixels wide and 36 pixels high (1/2 inch). To get this effect you must choose the fixed width and fixed height options.



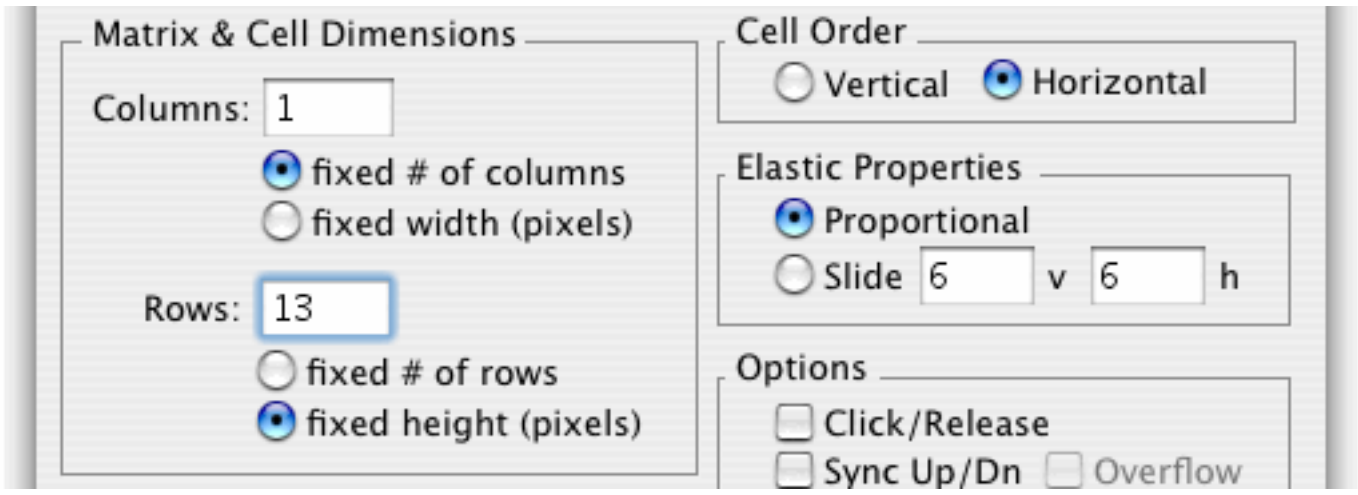
The rows and columns will always be 36 pixels (1/2 inch) high and wide. If the overall matrix changes size, the number of rows and columns will increase or decrease as necessary.

1	2	3	4
5	6	7	8
9	10	11	12

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40

If the size of the matrix is not an exact multiple of the size of the individual cells there will be extra space left-over on the right and/or bottom of the matrix. Panorama will leave this space blank. The only solution to this problem is to make sure that the matrix height and width are an exact multiple of the height and width of each individual cell.

It's possible to mix the fixed width/height and fixed # of rows/columns options in a single matrix. For example, this matrix always has one column, but has a variable number of rows that are always 13 pixels high.



As the height of the matrix increases, more and more rows appear. Each row is always 13 characters high.

Burritos
Super Deluxe $4.99
Jr. Super Deluxe $3.99
Super $3.99
Deluxe $4.99
Beef $4.99
Beef & Bean $4.99

Burritos
Super Deluxe $4.99
Jr. Super Deluxe $3.99
Super $3.99
Deluxe $4.99
Beef $4.99
Beef & Bean $4.99
Green Chile $4.99
Carnitas $5.50
Carne Azada $5.50
Bean & Cheese $2.99
Chorizo $3.99

Burritos
Super Deluxe $4.99
Jr. Super Deluxe $3.99
Super $3.99
Deluxe $4.99
Beef $4.99
Beef & Bean $4.99
Green Chile $4.99
Carnitas $5.50
Carne Azada $5.50
Bean & Cheese $2.99
Chorizo $3.99
Chicken $4.99
Machaca $3.99
Relleno $4.99
Bean & Egg $2.99
A La Mexicana $3.99

Designing a Matrix Template

Matrix templates were introduced earlier in this chapter. The next few sections explain how to create the actual "guts" of a matrix template.

In creating matrix templates, it may help to understand how Panorama draws a matrix. First, it calculates the exact size and location of each cell in the matrix. Then, starting from the upper left, it draws each cell. To draw each cell, it first locates the matrix frame, and the objects inside the matrix frame. It temporarily moves and adjusts these objects so that they fit inside the matrix cell. It then draws the cell. The process repeats for each cell. Your job is to create graphic objects that: 1) display the appropriate information in each cell, and, 2) will look satisfactory when adjusted to fit inside any reasonable size and shape matrix cell (as a bonus, you get to decide what constitutes a reasonable size and shape matrix cell!).

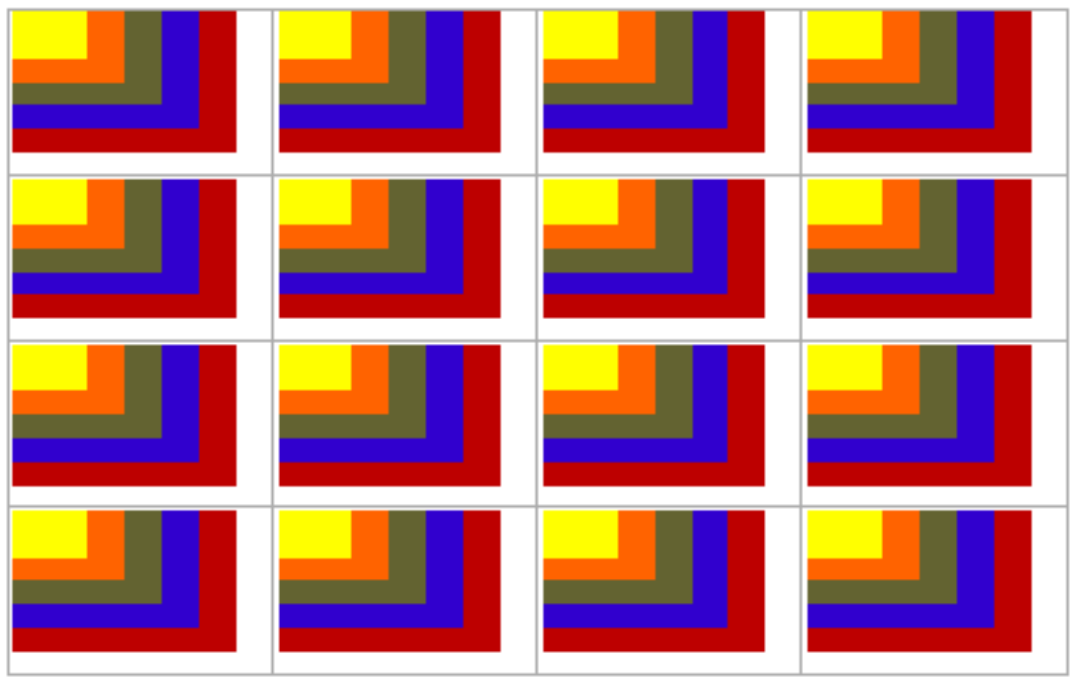
Adjustable Size Templates

The cells in a Super Matrix object can change size and shape several ways: 1) You can change the number of rows or columns, 2) You can change the size of the entire matrix in graphics mode, or 3) If you've created an elastic form, the size of the entire matrix can automatically change when the size of the window changes. Whenever any of these things happen, the graphics inside each matrix cell need to adjust automatically.

The Super Matrix object has two methods it can use for adjusting the graphics you create so that they automatically fit into each matrix cell. We'll use this matrix template to illustrate both of these methods.

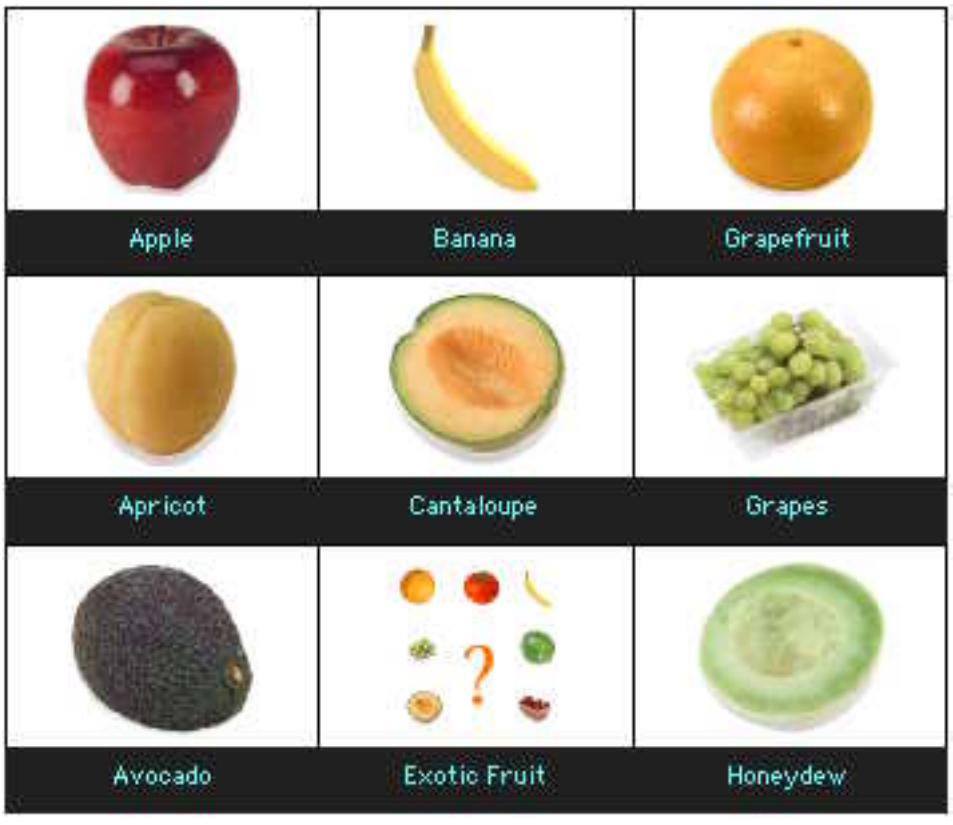


The standard method is Proportional. In this method, each graphic object will occupy the same relative position in the matrix cell that it occupies inside the matrix frame object. So if, for example, a flash art object is located in the bottom right third of the matrix frame it will also appear in the bottom right third of each cell. It's almost as if you drew the matrix template on a rubber sheet, and Panorama stretches the rubber sheet as necessary to fit each cell.

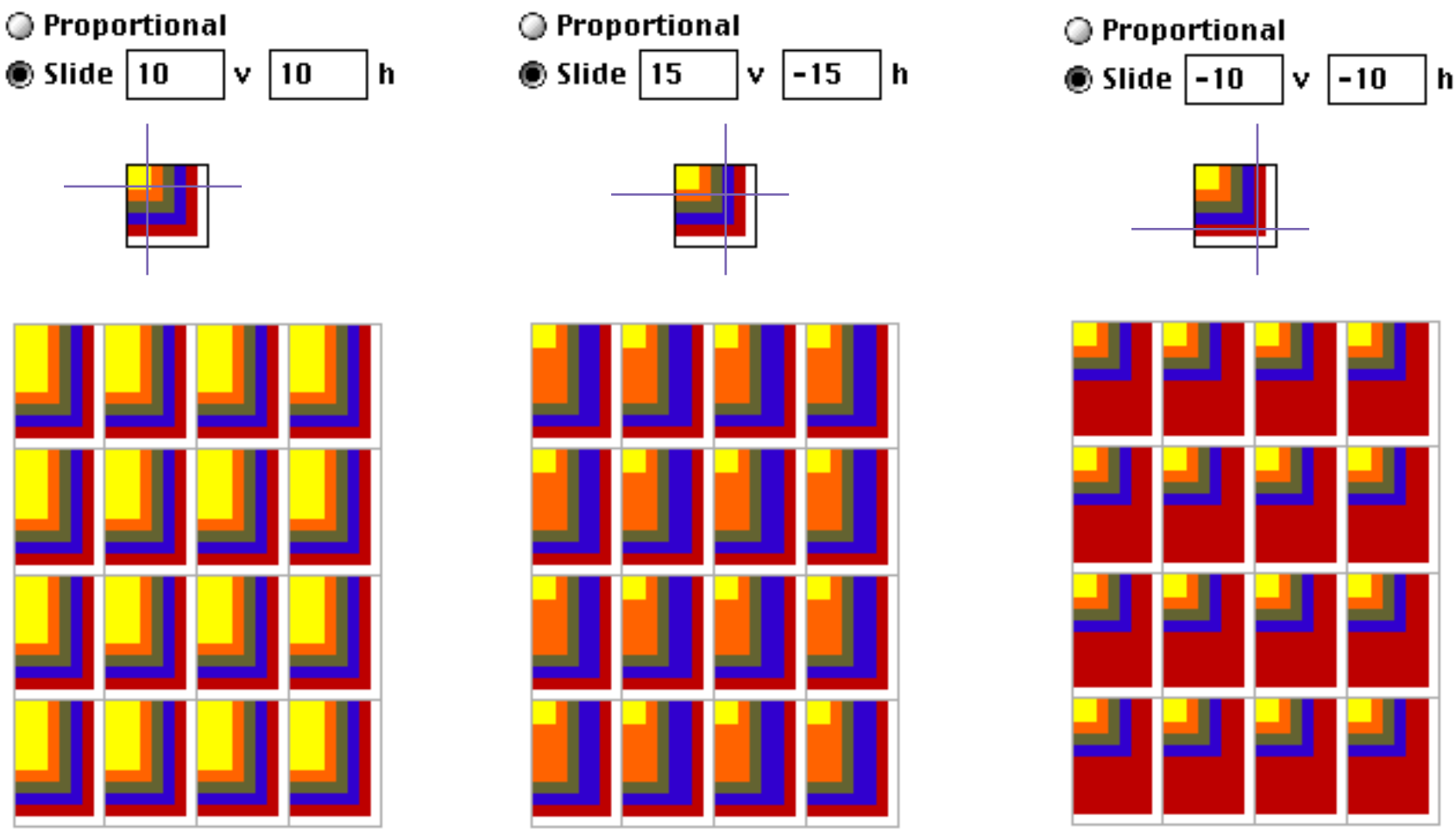


A potential disadvantage of this method is that as the cell expands, everything in the cell expands with it. For example, if each cell contained a photograph with a 9 point caption on the bottom, the caption would also expand if the cell expanded. This may or may not be what you want to happen.



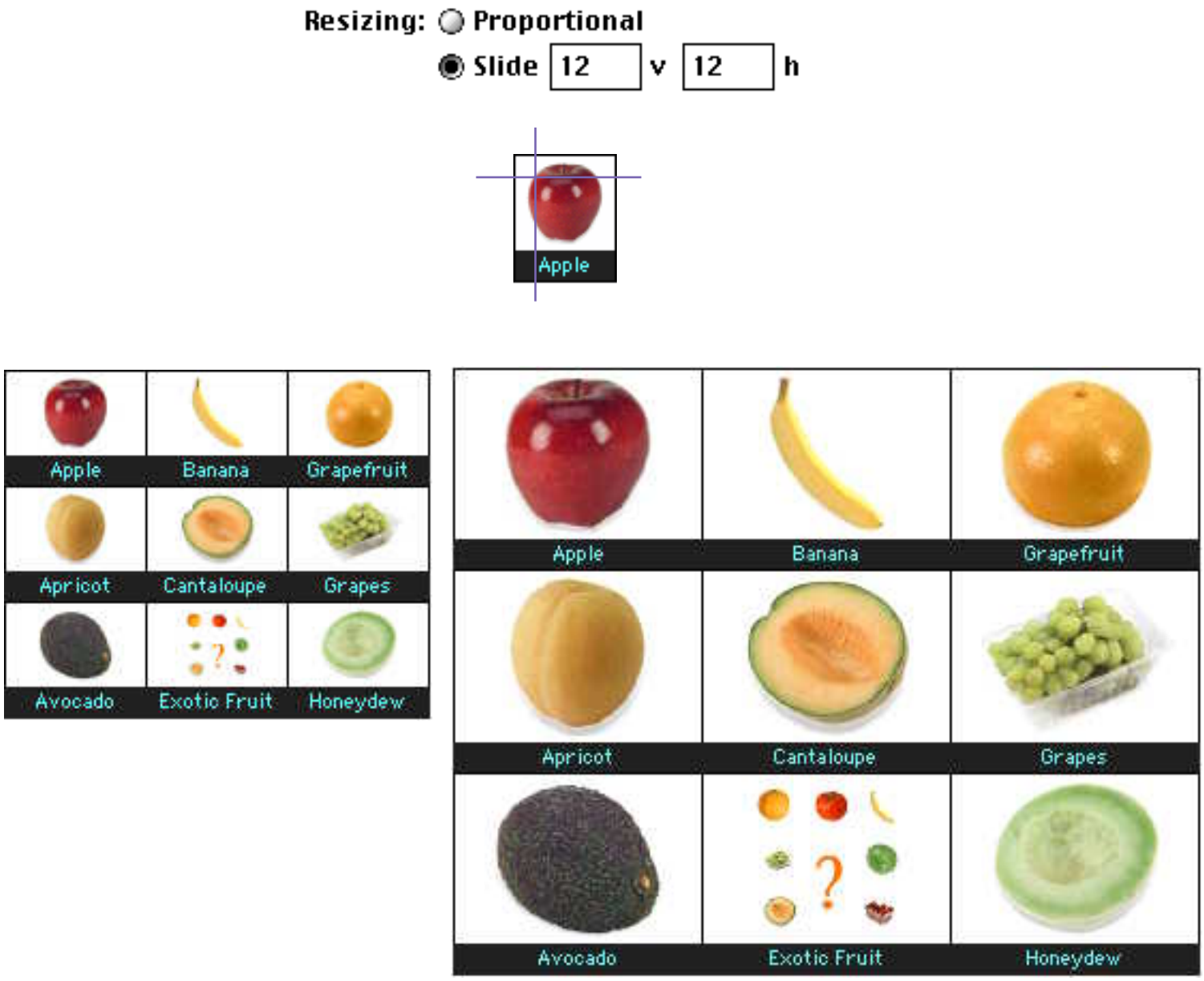


The second method for adjusting the matrix template is called Sliding. As the matrix cell gets bigger, points in the upper left hand corner stay put, while points toward the right and the bottom slide over and down. Essentially, the template "explodes" as the cell gets bigger. This method is somewhat similar to Panorama's cluster resize feature, and is also similar to an elastic form. To control the point at which the template "explodes," enter the size (in pixels) of the upper left hand corner that you want to stay together in the boxes marked v and h. V is the height of this box (stands for vertical) and h is the width (for horizontal). In some cases, it may be more convenient to enter negative values. In this case the dimension will be measured from the bottom right hand corner instead of the upper top.



If you use the sliding method, be careful that your matrix cells don't get too small. If they do, your template may "implode." The result isn't as horrible as it sounds, but it may look very strange and become unreadable.

Using the sliding method you can create a variable size photograph with a fixed height caption, like this. No matter how small or large the matrix is, the caption area will always remain the same height.



Tips for Adjustable Size Templates

The Text Display SuperObject™ is very handy for displaying text in a matrix cell (see See “Text Display SuperObjects™” on page 608). It has flexible alignment options that allow the text to be aligned both horizontally and vertically within the cell.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Even more handy is the Scale Text Size option, which allows the text to grow and shrink as the matrix grows and shrinks (see See “Text Display Options” on page 611). You may need to play with the lines option to get the effect you want (as this value is increased, the text size decreases).

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Both standard Flash Art™ and SuperObject™ Flash Art are also handy for displaying pictures in a matrix cell (see See “Creating Super Flash Art Objects” on page 749). Use the Center or Scale options to automatically display the picture attractively within each cell. For displaying photographs, the Proportional option (Super Flash Art only) is ideal. The photograph will always be displayed in the largest possible size without distortion.

Matrix Co-Ordinates and Data

Whether you are displaying text or pictures (or both), chances are you want to display something in each cell in the matrix. To do this you’ll use one or more formulas. These formulas can either be displayed directly (using an Auto Wrap Text object or Text Display SuperObject™) or may be incorporated into a Flash Art™ object to display a picture. Within each formula, you can choose from several different functions to identify the cell being drawn and the appropriate data to draw in that cell.

Matrix Co-Ordinates (What cell is this?)

The `info("matrixcell")` function returns the cell number within the matrix, starting with 1 in the upper left hand corner. If the matrix order is horizontal, then the cell numbers will be consecutively numbered from left to right in each row. If the matrix order is vertical, then the cell numbers will be consecutively numbered from top to bottom in each column. This illustration shows a matrix with an auto-wrap text object being used to display the cell number along the bottom.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

The `info("matrixcolumn")` function returns the column number, starting with 1 for the left hand column and increasing by one for each column to the right.

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

The `info("matrixrow")` function returns the row number, starting with 1 for the top row and increasing by one for each row in the matrix.

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

Usually the result of these functions is not used alone, but is fed into another function. For calendars, you would feed the cell number into `calendardate(` and `calendarday(` functions. To display items in an array or list, you would feed the cell number into the `arrayelement(` or `extract(` functions. These functions can help convert the raw cell number into the actual data that should be displayed in the cell.

Co-Ordinates and Scroll Bars

If the matrix has scroll bars, the coordinates will adjust as the scroll bars are used. If the matrix has both vertical and horizontal scroll bars then the `info("matrixcell")` function will not return a reliable number. Instead you should use the `info("matrixrow")` and `info("matrixcolumn")` functions to identify the cell.

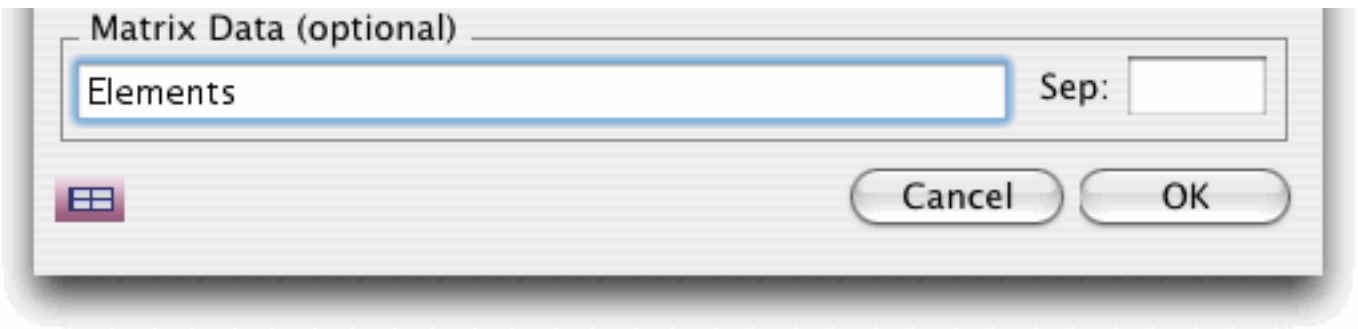
Using a Matrix to Display an Array

Starting with Panorama V there are two techniques available for displaying an array with a matrix. The first technique is to use just the `info("matrixcell")` function (this technique will also work in earlier versions of Panorama). For example, suppose that you want to display a carriage return separated array named Elements in a matrix. To do this simply use a Text Display SuperObject with this formula:

```
array(Elements,info(“matrixcell”),¶)
```

This will work, but it has some drawbacks. In this case the array is stored in a field or variable, but if it is a more complex formula (perhaps a `lookupall(` function, for example) that formula must be calculated over and over again. The matrix object has no way of detecting if the array data changes, so the matrix can’t automatically redisplay the data if it changes. The matrix also can’t automatically the scroll bar(s), if any, to account for changes in the data.

The second technique remedies all of these drawbacks. Instead of calculating the array elements in a formula, they are calculated by the matrix object itself. To do so you must enter the formula for the array in the matrix configuration dialog, as well as the separator. (Note: This technique only works for new super matrix objects that are created with Panorama V or later. If you have an old Super Matrix object that was created with an older version of Panorama you'll need to delete it and create a new object to use this feature.)



If the separator is left blank (as in the example above) Panorama will assume the array is carriage return separated. Otherwise you can type any single character into this box.

Once the matrix data and separator are set you can use three functions to access the data within a matrix cell. The the `info("matrixcelldata")` function returns the array element corresponding to the current cell. In our example, to display the array elements with a Text Display Superobject the formula would be simply:

```
info(“matrixcelldata”)
```

A formula can find out the entire array value with the `info("matrixdata")` function and the current separator value with the `info("matrixseparator")` function.

Using the Matrix as a Button

The Super Matrix object doesn't just display a matrix, it also allows you to click on matrix cells and trigger a procedure. You can select the procedure to be triggered using the Procedure pop-up menu in the Super Matrix dialog. If you decide later that you don't want a procedure to be triggered, simply un-check the Procedure check box.

If the Click/Release option is checked, Panorama will highlight the matrix cell the user clicked on by inverting it (black becomes white, white becomes black). The procedure will be triggered if the mouse is released over the same matrix cell it was originally clicked on. If the Click/Release option is not checked, Panorama will trigger the procedure immediately without highlighting the matrix cell.

What Cell Was Clicked?

The procedure can determine what cell in the matrix was clicked by using the `info("matrixcell")`, `info("matrixcolumn")` and `info("matrixrow")` functions described earlier in this section. It can also use the `info("matrixcelldata")` function if the Matrix Data and Separator have been set up. In addition, the `info("trigger")` function will return the name of the matrix object itself (if any — see “Object Type/Object Name” on page 533). If you haven't assigned a name to this object, the default is custom.

Buttons Within Matrixes

If you place buttons (3D buttons, checkboxes, radio buttons, etc.) within a matrix template, these buttons will be displayed in each cell of the matrix. However, these buttons will not be active and will not do anything when you click on them.

Updating the Matrix Display

So far this discussion has assumed that the contents of the matrix never change. But appointments are rescheduled, new photos are added to portfolios, and all these changes must be displayed as they happen. To cause the entire matrix to update whenever the current record moves up or down check the Sync Up/Dn option.

If you have set up the Matrix Data formula (see "Using a Matrix to Display an Array" on page 944) the matrix will update whenever any field or variable in that formula changes. This also usually means that the Sync Up / Dn option is not necessary.

If you want to update only a portion of the matrix you must use a procedure. To review, a procedure can send a message to any SuperObject that has a unique object name using the `SuperObject` statement (see "Program Control of SuperObjects™" on page 1786). The format for the statement that tells a Super Matrix to redraw some or all of the cells is:

```
SuperObject "matrix name","redraw","area",start,end
```

The first parameter, "matrix name", is the name assigned to the matrix object. (Note: To give an object a name, first select the object, then use the **Object Name** command in the Edit menu or click on the object name in the Graphic Control Strip. See "Object Type / Object Name" on page 533 for more details on object names.)

The second parameter, "redraw", tells the Super Matrix object that you want to redraw part or all of the matrix.

The third parameter, "area", defines the area that will be redrawn. Legal options for this parameter are: "all", "column", "row", and "cell".

The fourth and fifth parameters define the start and end of the area to be redrawn. For example, if the third parameter was "column" and the last two parameters were 3 and 5, then columns 3 thru 5 would be redrawn. (Note: The start and end values are ignored if the "all" area is chosen.)

The following one-line examples illustrate different ways a matrix might be updated.

```
; This command redisplays the entire month
SuperObject "Month","redraw","all",0,0

; This command redisplays only weekdays
SuperObject "Month","redraw","column",2,6

; This command redisplays photo 7 only. Use a similar command
; if you update a single item in a matrix.
SuperObject "Thumbnails","redraw","cell",7,7

; This command redisplays all photos after photo 12. Use a similar
; command if you insert or delete an item in the middle of a matrix.
SuperObject "Thumbnails","redraw","cell",12,9999
```

A Trick for Updating the Matrix Display Automatically

(Note: The following trick is unnecessary if you are have set up the Matrix Data formula (see "Using a Matrix to Display an Array" on page 944), but may still be used in older databases that do not use this option.) If you simply want the entire matrix to redraw whenever a specific field changes, you can do so without writing a procedure. Simply overlay the matrix with an auto-wrap text object (see "Displaying Data in Auto-Wrap Text" on page 595) or text display SuperObject™ (see "Text Display SuperObjects™" on page 608) with a formula that includes the field or fields you want to update. (If you use an auto-wrap text object, don't forget that formulas must be enclosed in { } characters.) Use text funnels to make the result of the formula invisible (see "Taking Strings Apart (Text Funnels)" on page 1214). For example, if you want to update the matrix when the Photos field is updated, overlay the matrix with an object that contains the formula:

```
Photos[2,1]
```

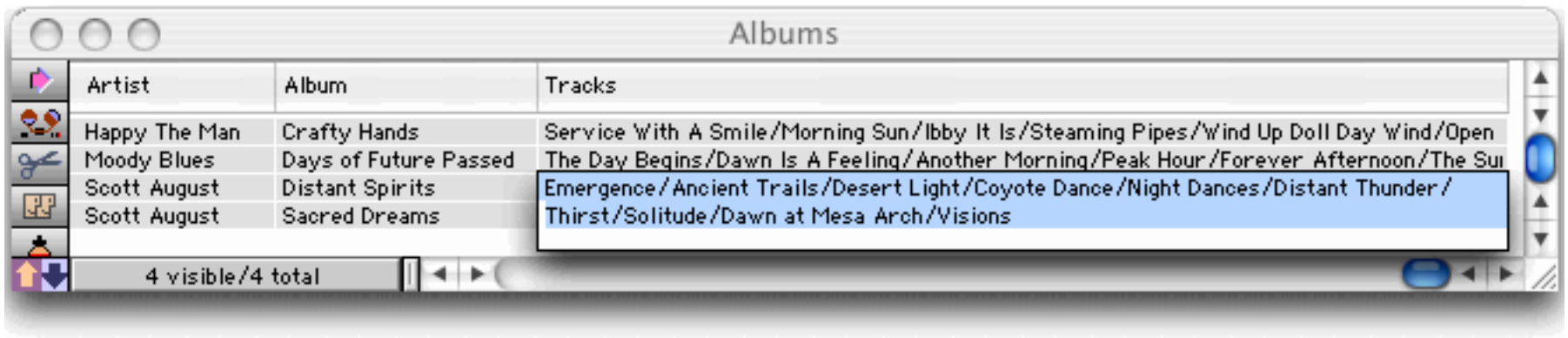
The result of this text funnel will always be an empty string, so nothing is displayed. Nevertheless, Panorama will always try to redisplay this text whenever Photos is changed, and your matrix will go along for the ride and get updated also.

Super Matrix Case Studies

The following case studies show several examples of how super matrix objects can be used in your databases. In addition to these examples you'll find a number of databases that are included with Panorama that use matrix objects: the Favorite Databases wizard, Text Import and Text Export wizard, White Pages, Fedex Tracker, Icons & Backgrounds, Disk Permissions and more. You can examine these databases for additional ideas on how to use matrixes.

Displaying Music Tracks

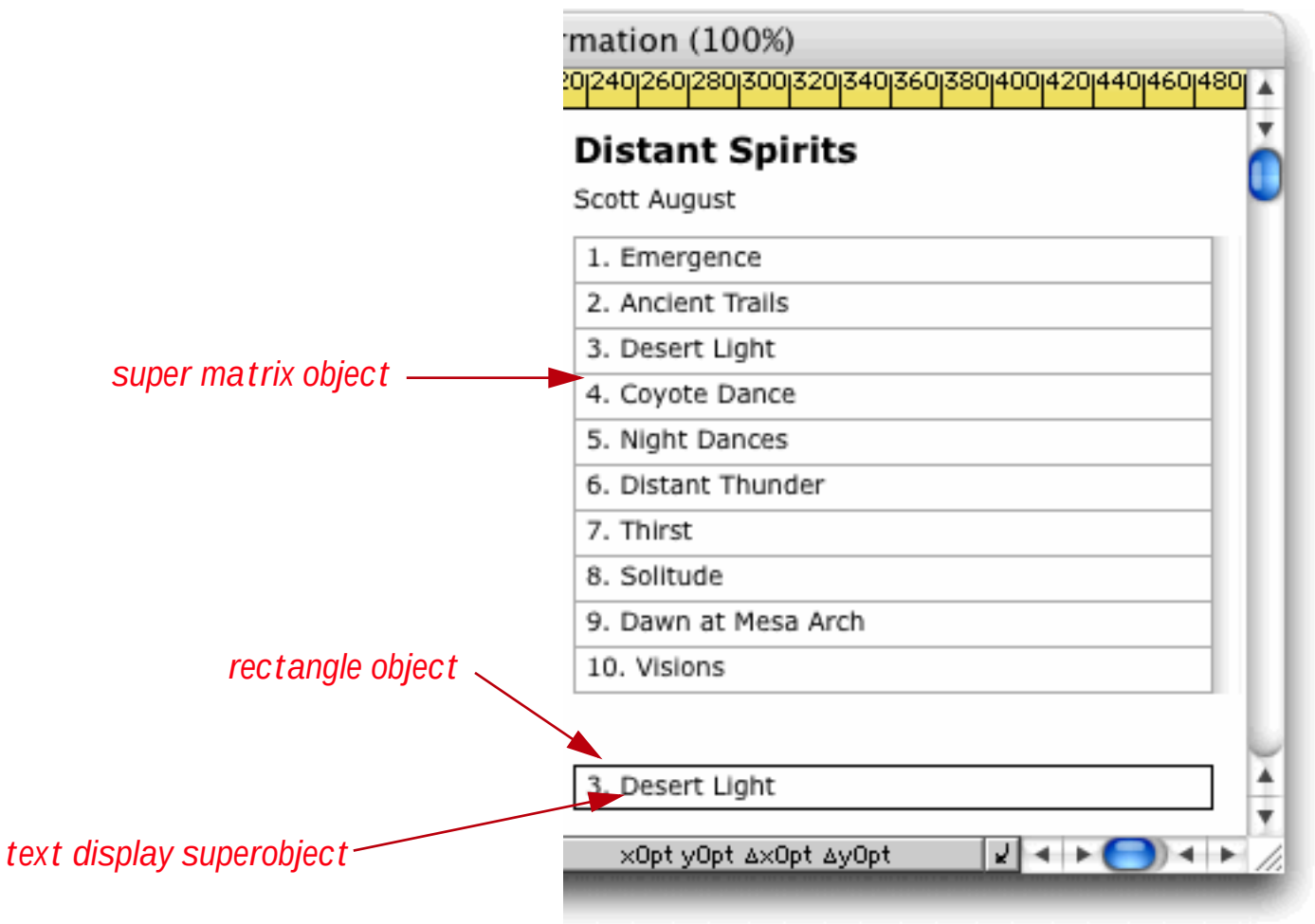
Our first simple example displays album track names. These track names are stored in the Tracks field, with a / character separating each element of the array.



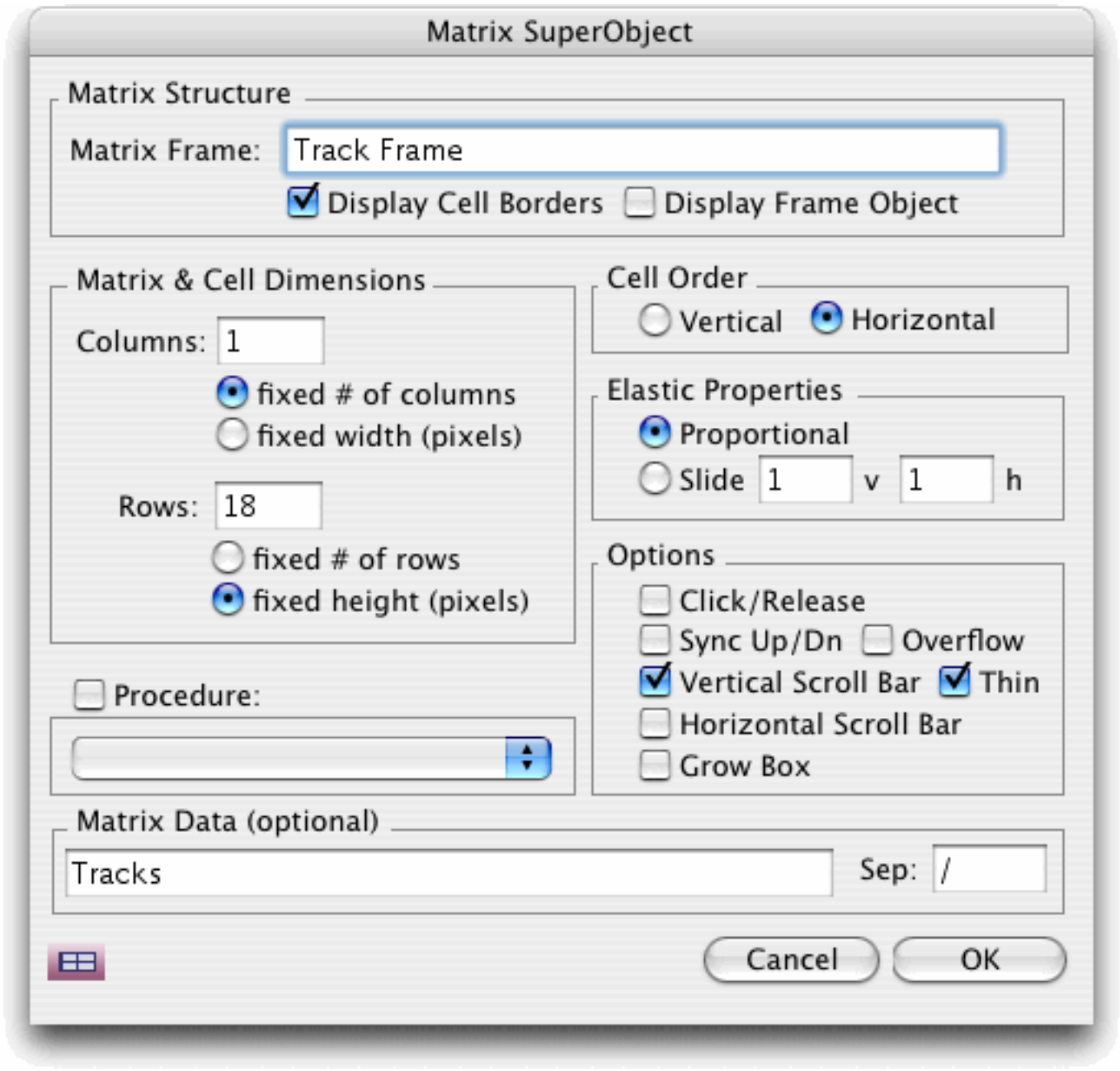
Instead of displaying the data in the data sheet, we'd really like to display it in a form like this. This form displays the exact same data as the data sheet above, but formatted with the use of a Super Matrix object.



Switching to Graphics Mode reveals that the album tracks are displayed with three objects: the primary *super matrix* object, a *rectangle* object (used as the frame object) and a *Text Display SuperObject*.



Let's look at the settings for each of these three objects. First is the *Super Matrix object* (this is usually the object we create first).



The Matrix Frame option is set to Track Frame. You can set this option to any name you like, but it must match the name of the frame object (more on this in a minute).

The Matrix & Cell Dimensions are set to display a single column with each cell being 18 pixels high. This height is set according to the height of the font you choose to use in the Text Display SuperObject. In the Options section we've turned on a thin vertical scroll bar.

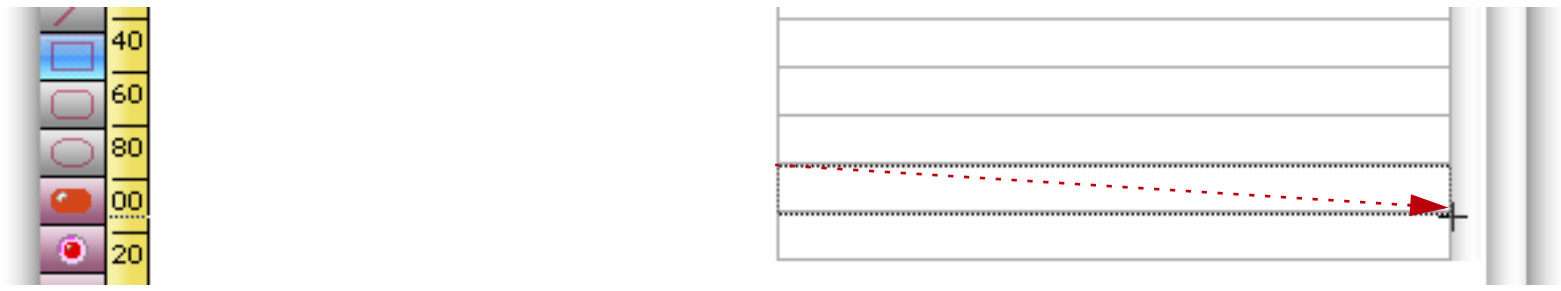
At the bottom of the dialog the Matrix Data has been configured so that this matrix will display data from the Tracks field. The data for each field is separated by the / character (see the original data sheet above).

After the matrix object was created we made one further change - the color of the object was changed to light gray. If you don't do that the divider lines between each cell will be black, which we thought was too dark.

The second object is the *frame rectangle*. This is simply an ordinary rectangle with its name set to Track Frame (see "Object Type / Object Name" on page 533).

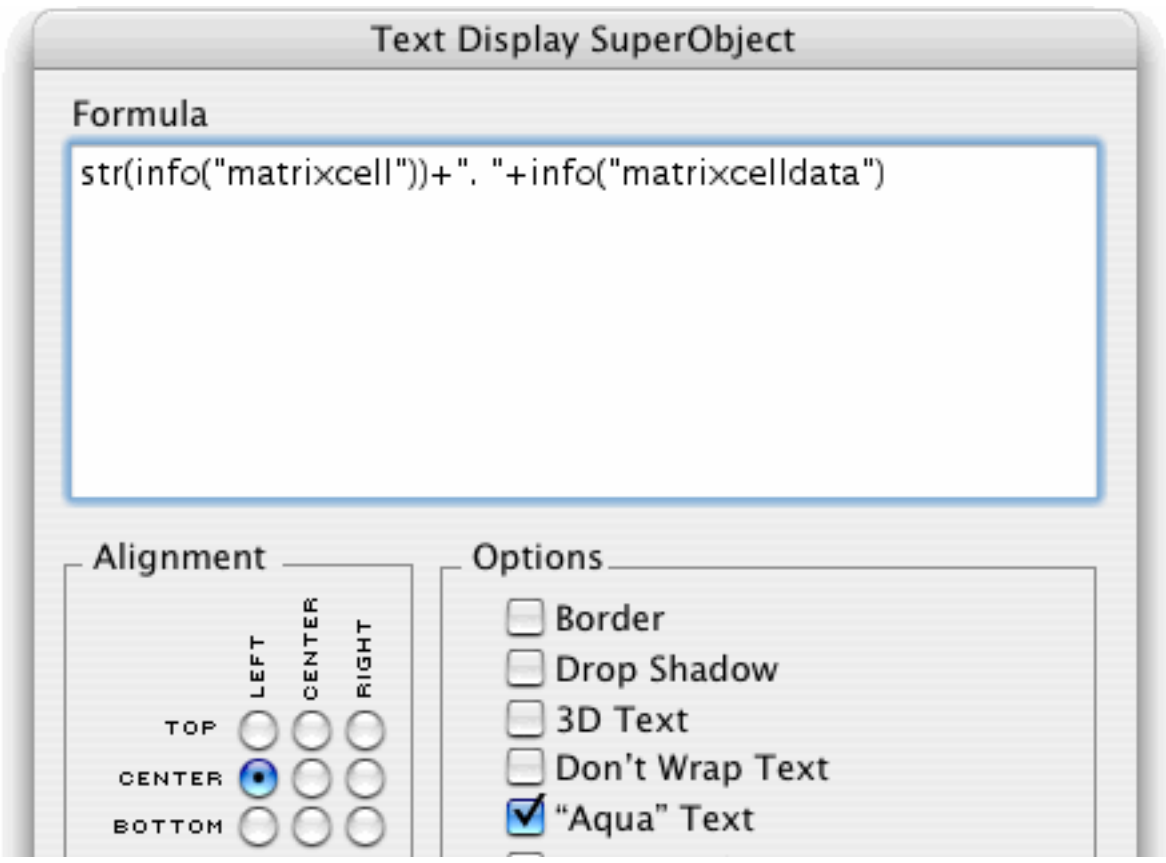


It's not necessary for the frame rectangle to be the same size as the cells in the super matrix object, but it is convenient. We've found that an easy way to do that is to initially create the rectangle on top of the matrix object.



Using the matrix as a guide it's easy to create the rectangle the exact size needed. Once it is created you can move the rectangle to another location, then set the object name.

The final object is the *Text Display SuperObject*.



The formula controls what is displayed in each cell of the matrix. The first part of the formula

```
str(info("matrixcell"))+". "
```

displays the track number: 1., 2., 3., etc. The second part of the formula

```
info("matrixcelldata")
```

displays the actual track names that have been extracted from the Tracks statement.

The end result is that the form displays a numbered list of the track names.



The form will update automatically when you edit the list of track names or move to a different record. If there are more than 10 tracks on an album a thin scroll bar will automatically appear, allowing you to scroll to see the additional tracks.



(A personal note from the ProVUE staff - the two albums above are really excellent. If you'd like to know more about this artist see **http://www.cedarmesa.com** or **amazon.com**.)

Displaying Olympic Results

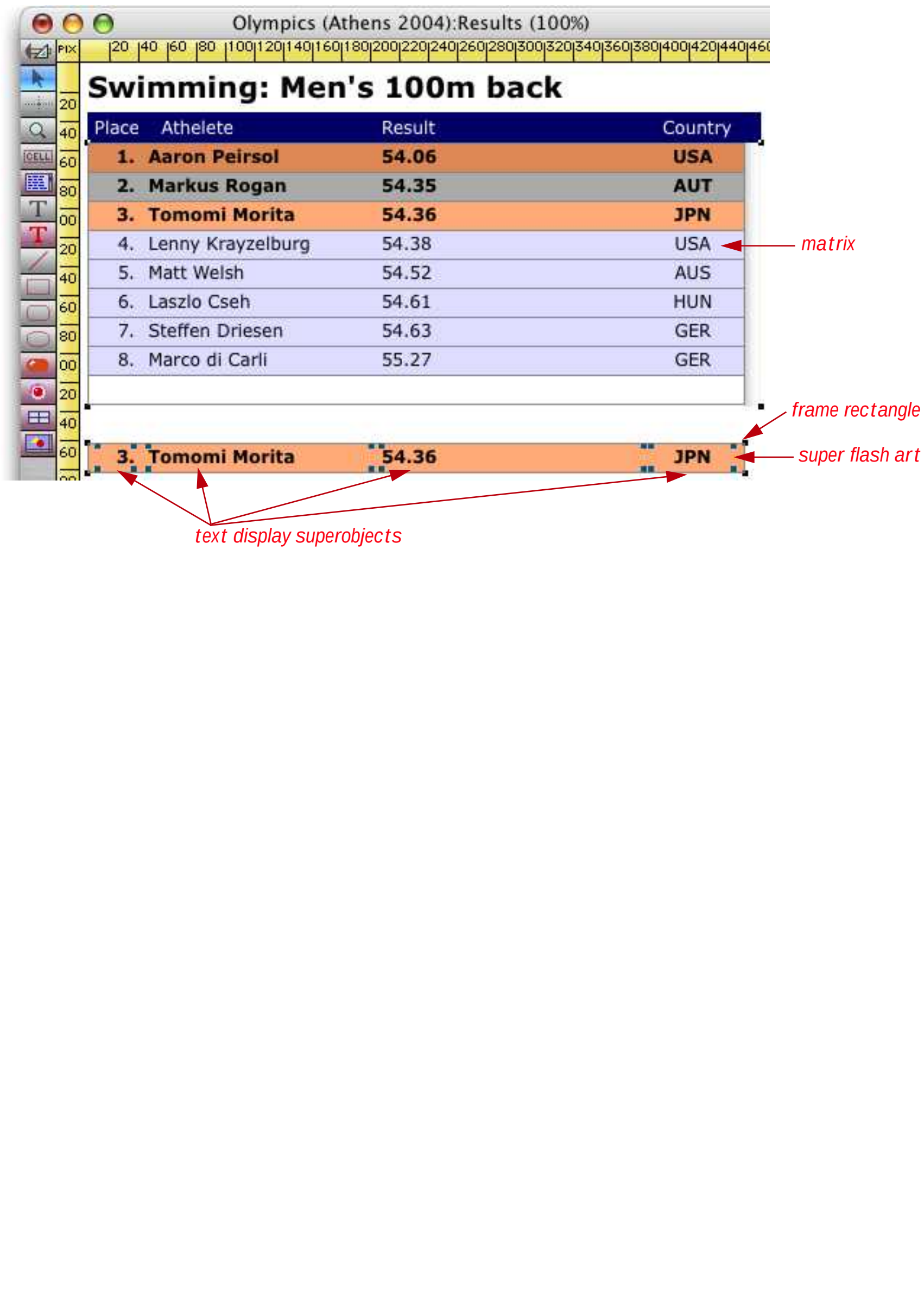
The next example displays Olympic competition results. These track names are stored in the Results field, with a carriage return character separating each element of the array.



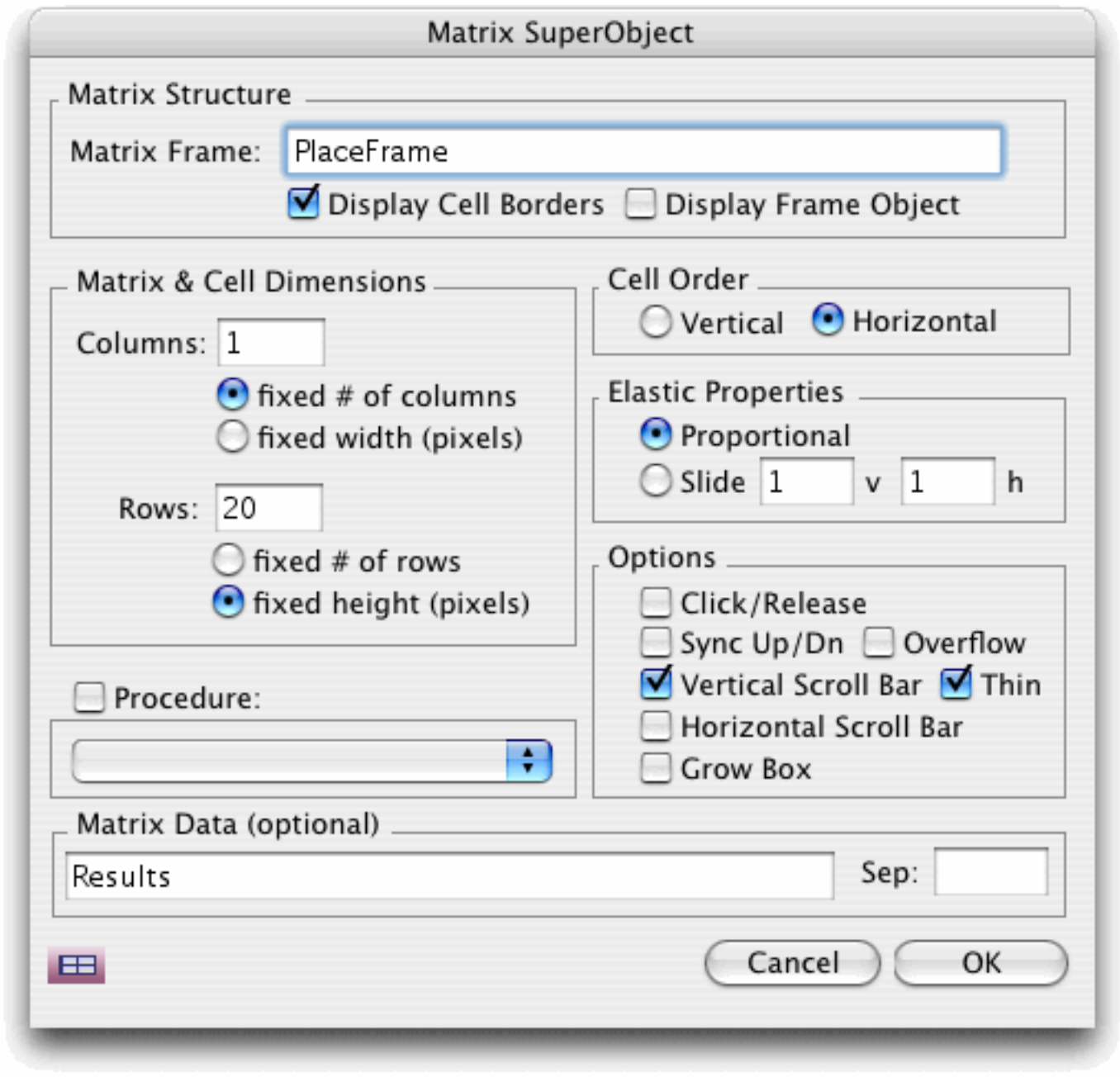
This example is a bit more complicated because each element contains four subelements - the place, athlete name, country, and time (or distance, weight etc.). We'll use a matrix to display each of these subelements in a separate column.



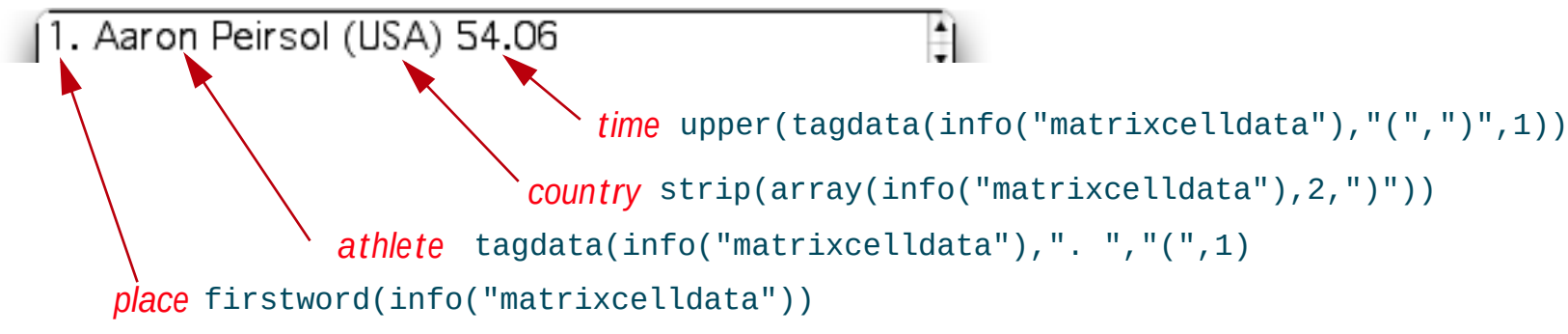
Switching to Graphics Mode reveals a design similar to the Album tracks example, including the primary *super matrix* object, and a *rectangle* object (used as the frame object). However, instead of just one *Text Display SuperObject* there are now four (one for each subelement), and there is also a *Super Flash Art* object that draws the colored background.



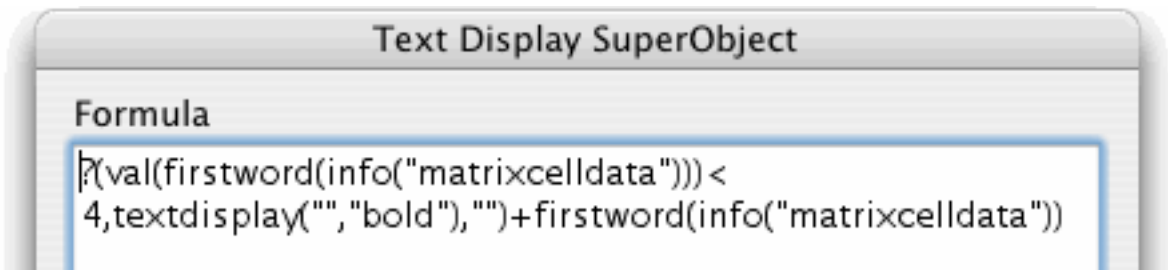
Here is the Matrix SuperObject configuration for this example. It's almost identical to the previous example, except for using an empty separator (which means that a carriage return is the separator - in other words, each cell will display data from an individual line in the Results field).



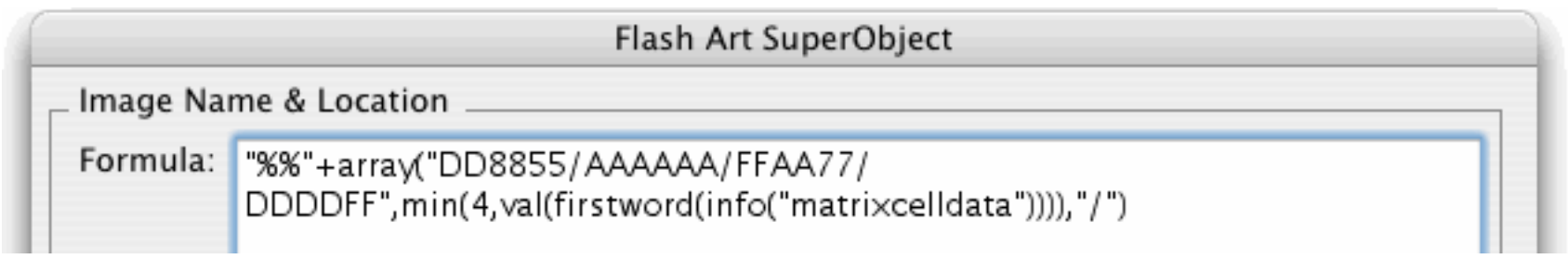
Before looking at the Text Display SuperObjects, let's review the format of each line in the Results field (this data was extracted from a news web site in this format). The illustration below shows the four subelements of each line, along with the formulas needed to extract each element from the line.



Here is the configuration dialog for the leftmost Text Display SuperObject. Notice that in addition to the formula for extracting the subelement, the formula also has a prefix that uses the `textdisplay(` function (see "Controlling Text Display Color and Style on the Fly" on page 619) to display the first three results (the medal winners) in bold.



The form also has a Super Flash Art object that displays the background color for each element (see "Using Flash Art to Display a Color" on page 759). The formula uses the `array(` function to pick a different color for the first (gold - DD8855), second (silver - AAAAAA), third (bronze - FFAA77) and subsequent lines (light blue - DDDDFF).



In data mode the results are automatically formatted into four columns.

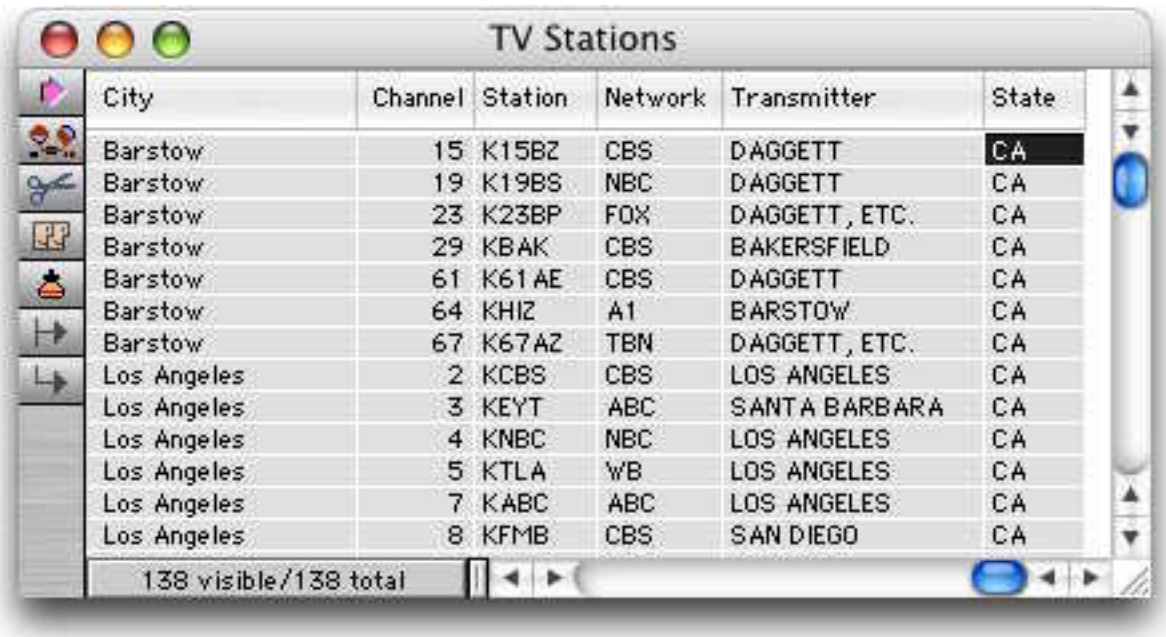


If the results don't fit within a matrix the scroll bar is automatically activated.



Television Station Guide

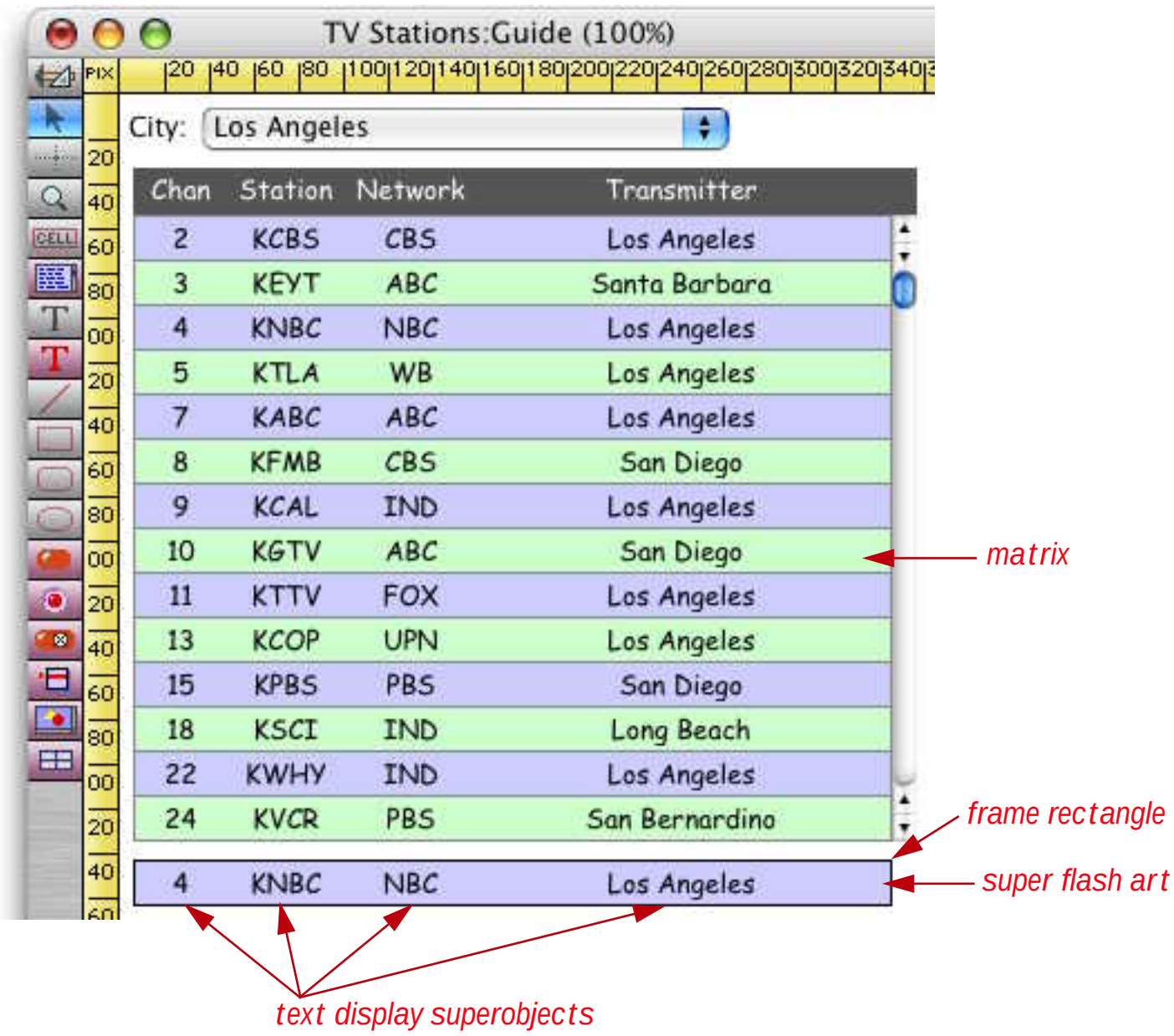
In the previous examples the matrix array data was contained in a single field. In this example we'll show how a more complex formula can be used to assemble the array "on-the-fly." Our example will display the television stations available in a particular city. Here's the data sheet:



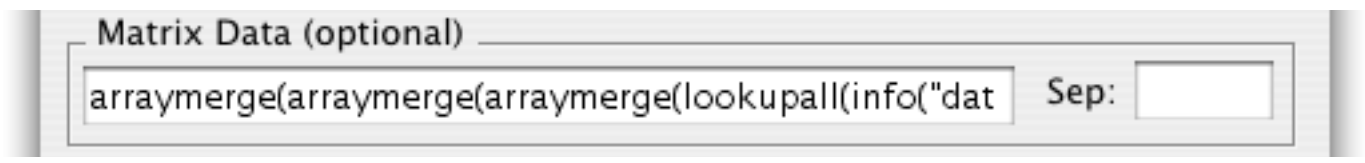
Our goal is to display the data using a matrix like this:



In graphics mode this form looks very similar to the previous example:



The big difference in this case study is the Matrix Data. Instead of simply a field or variable this contains a long formula.



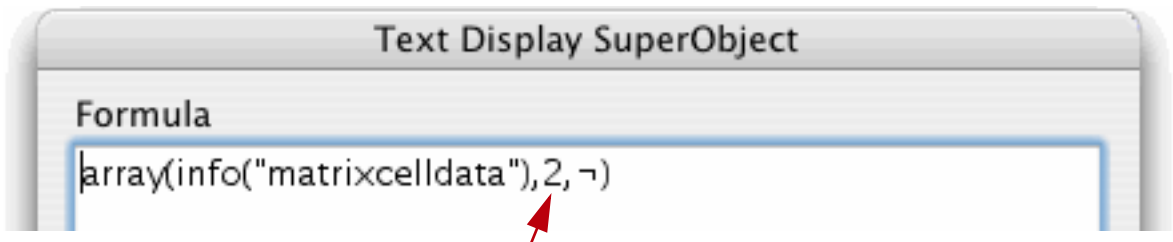
The formula is so long that most of it is invisible. Here is the complete formula:

```
arraymerge(
arraymerge(
arraymerge(
   lookupall(info("databasename"),"City",tvLocation,"Channel",¶),
   lookupall(info("databasename"),"City",tvLocation,"Station",¶),¶,¬),
   lookupall(info("databasename"),"City",tvLocation,"Network",¶),¶,¬),
   lookupall(info("databasename"),"City",tvLocation,"Transmitter",¶),¶,¬)
```

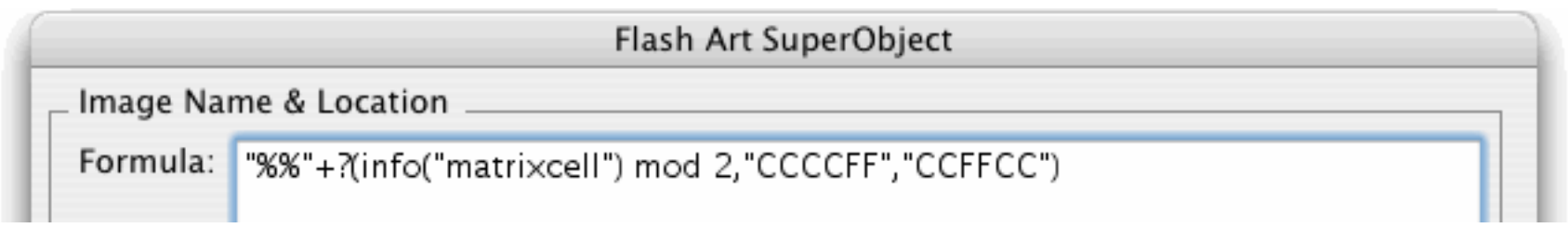
The variable tvLocation is set by the pop-up menu at the top of the form, and will contain the name of a city (Los Angeles, San Diego, etc.). Each of the `lookupall(` functions returns a carriage return separated array with data from one field (Channel, Station, Network and Transmitter). These four arrays are combined into a single array with the `arraymerge(` functions. (For detailed information on these functions see the **Programming Reference** wizard.) The final combined array will look like this (though of course you can never see it directly):

```
69      K69FB    TBN      SACRAMENTO
3       KEYT     ABC      SANTA BARBARA
6       KSBY     NBC      SAN LUIS OBISPO
8       KAGP-LP  SPA      ARROYO GRANDE
12      KCOY     CBS      SANTA MARIA
15      K15BD    PBS      SAN LUIS OBISPO
20      KCCE-LP  IND      SAN LUIS OBISPO
22      K22EE    TBN      MORRO BAY
```

The four text display objects simply extract a sub-element from the matrix generated by the formula above.



The remaining object is the Super Flash Art object, which displays the alternating blue/green background. Here is the formula that displays the colors. The result of the `mod` operator will be either 0 or 1 depending on whether or not `info( matrixcell )` is even or odd.



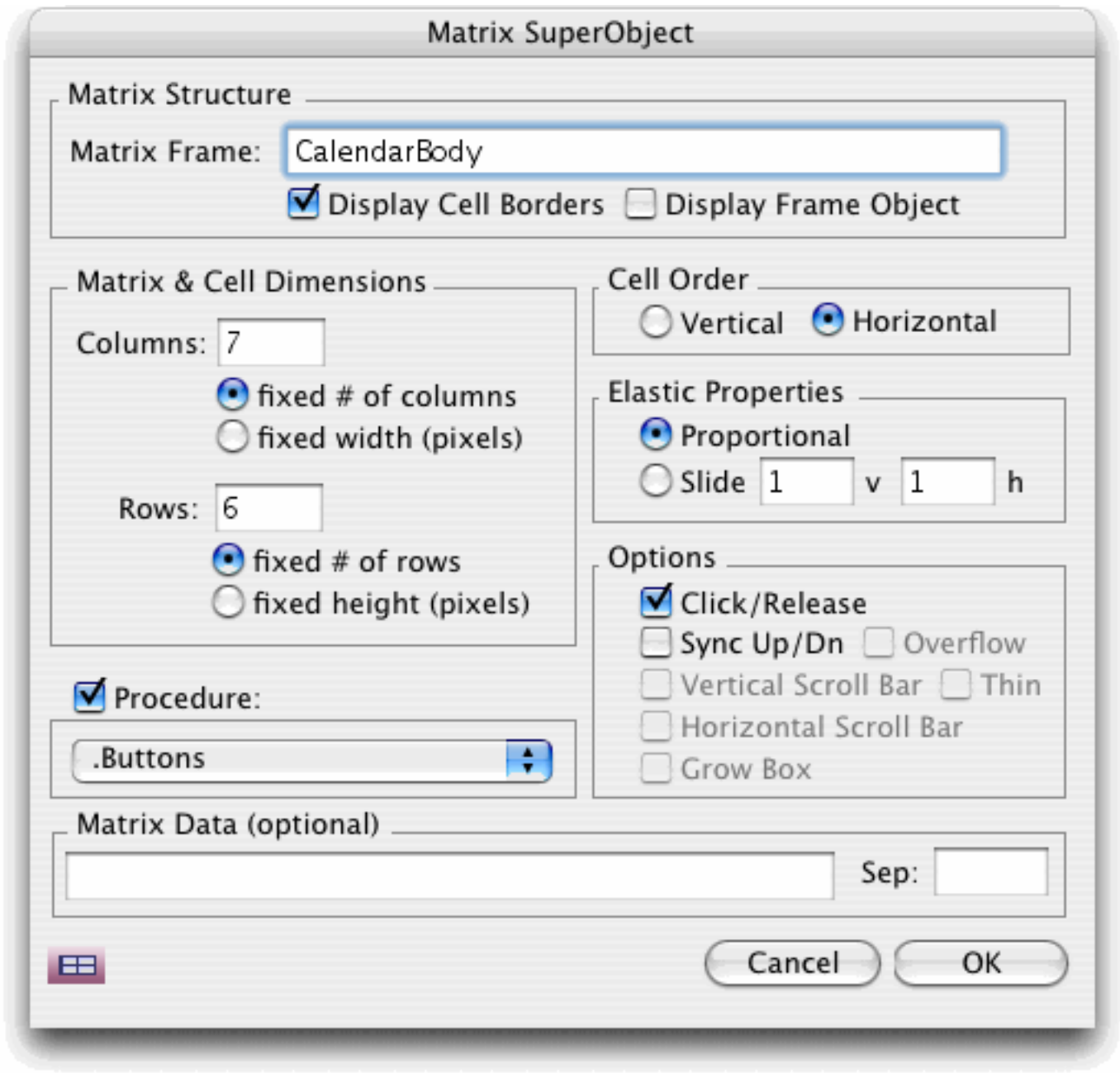
The final result is shown below. Whenever you select a different city with the pop-up menu the matrix re-calculates the data formula and redisplays the matrix.



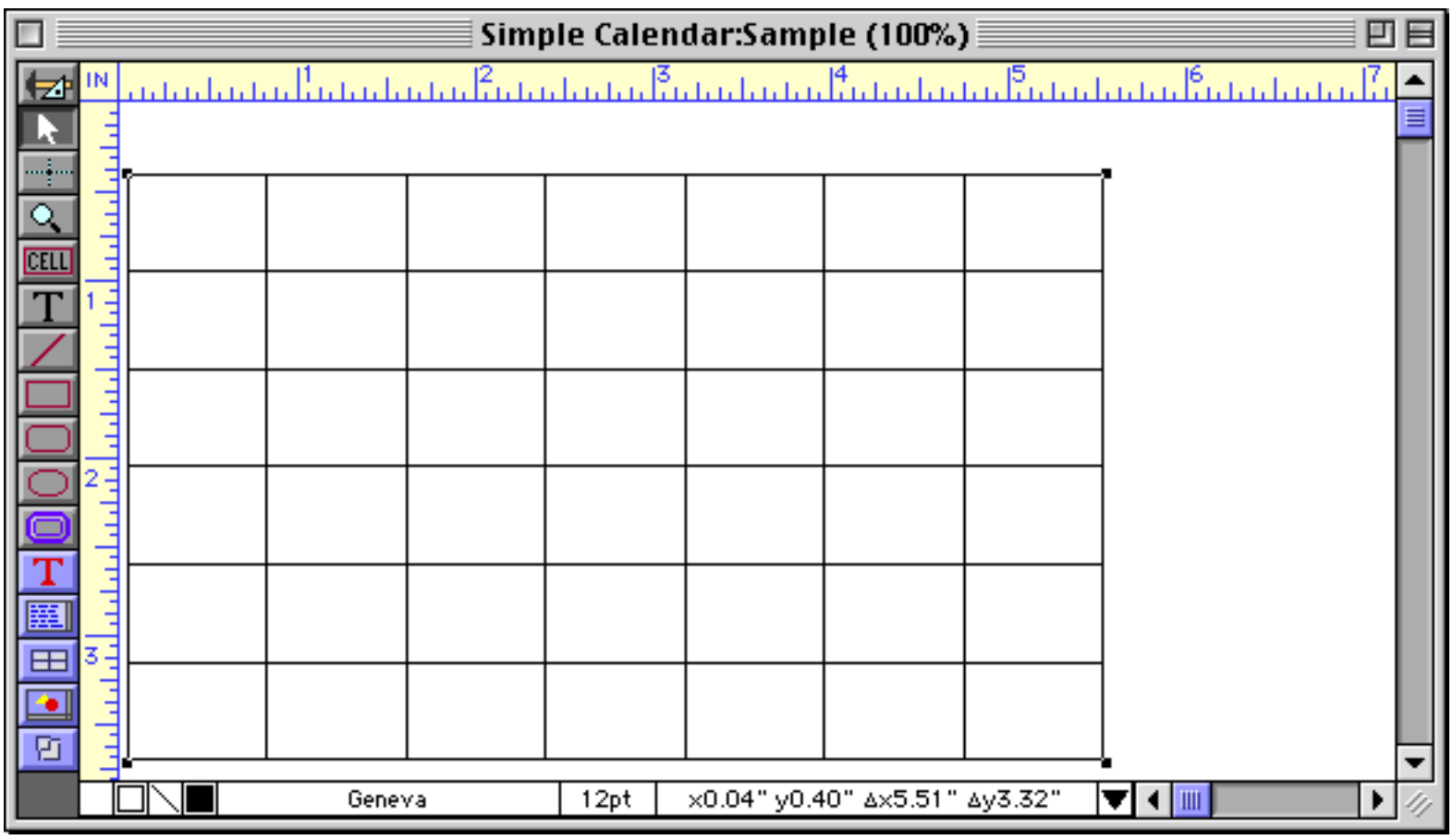
In this example the `lookupall(` functions extracted data from the current database, but they could just as easily extract data from another database. For example, a vendor database could display a list of purchase orders for this vendor that is extracted from a purchase order database, or a contact database could display a list of phone calls from a phone log database. This makes the Super Matrix object an ideal way to display one-to-many relationships between different databases.

Building a Calendar

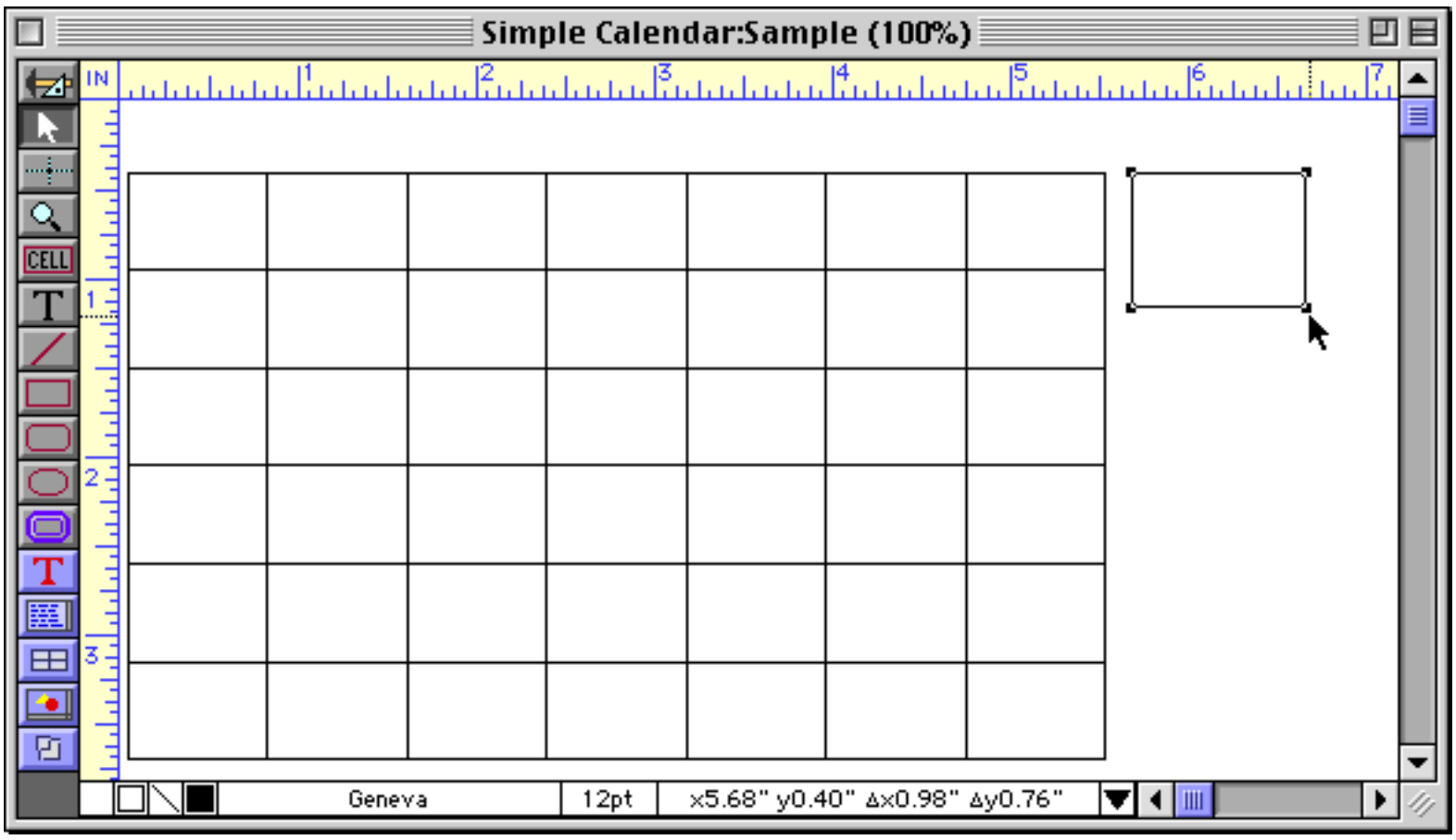
Our final case study uses a SuperMatrix object to build a monthly calendar. Unlike the other case studies in this example the Matrix Data is not used. A simple calendar can be constructed in a few minutes. Start by creating a Super Matrix object.



The object should be seven columns wide (one for each day of the week) by six rows high. Make sure that the matrix order is horizontal and assign a name for the Matrix Frame of One Day. We've also set the resizing method to Slide (see "Adjustable Size Templates" on page 939).



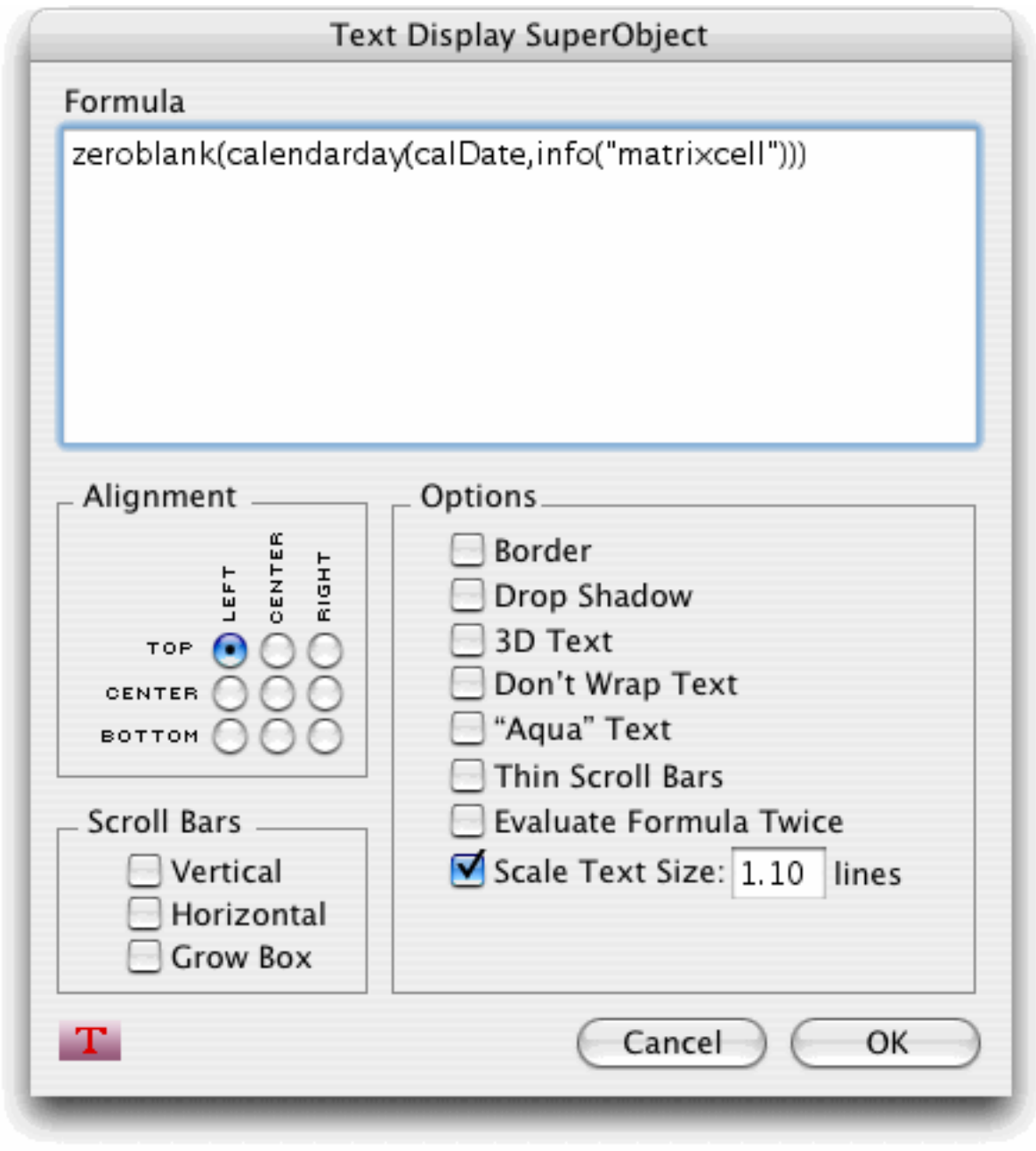
Next create a rectangle to serve as the matrix frame. Notice that the rectangle frame does not have to be the same size as the cells in the matrix, though it can't hurt either.



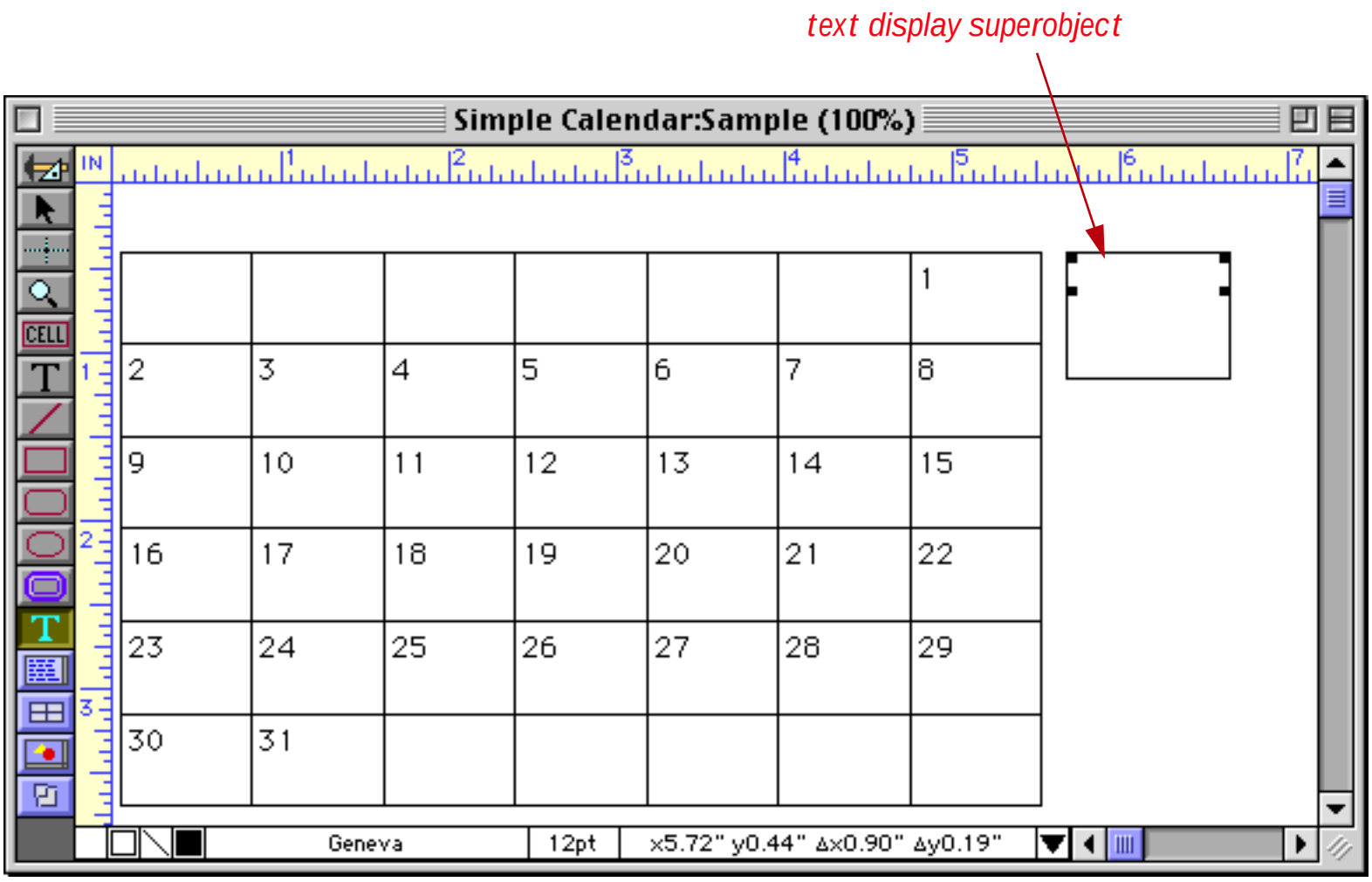
Use the **Object Name** command to assign the name One Day to this object (see "Object Type / Object Name" on page 533).



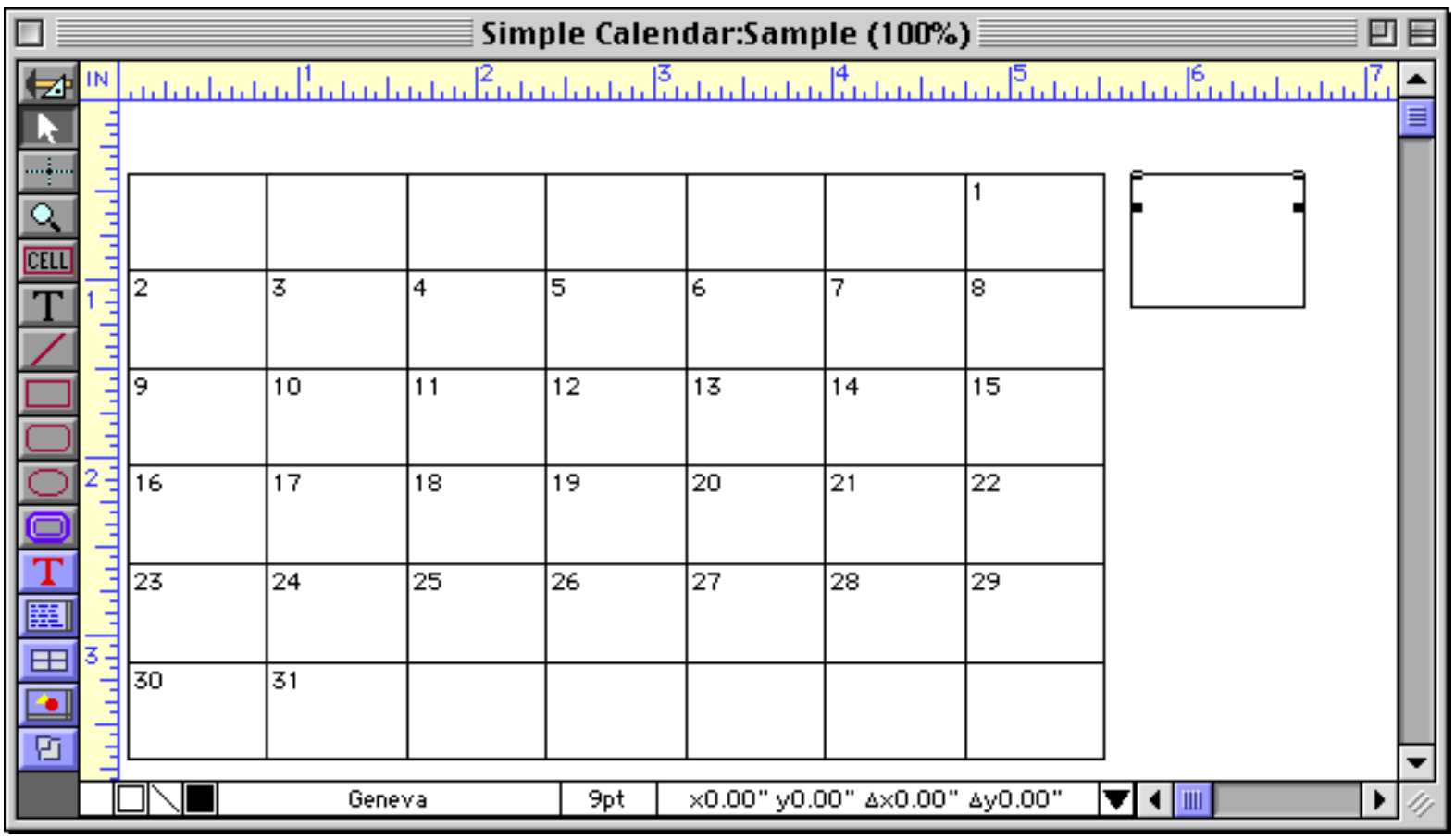
The next step is to add a Text Display SuperObject to display the day of the month (1, 2, 3, … 30, 31). See "Creating and Modifying Text Display SuperObjects" on page 608 to learn how to create such an object. Use the formula shown in the illustration below.



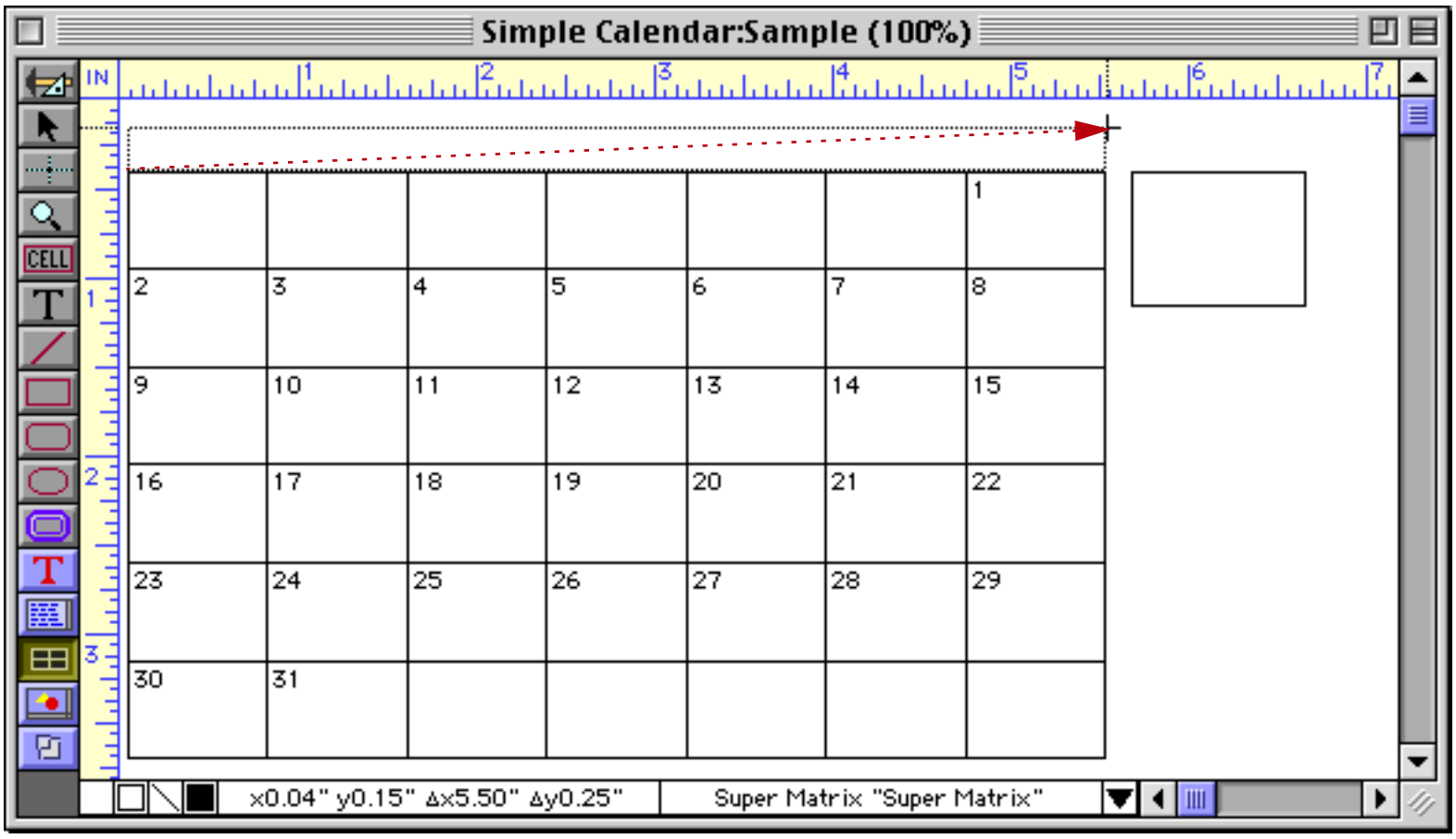
When you press **OK** the beginnings of a calendar will appear.



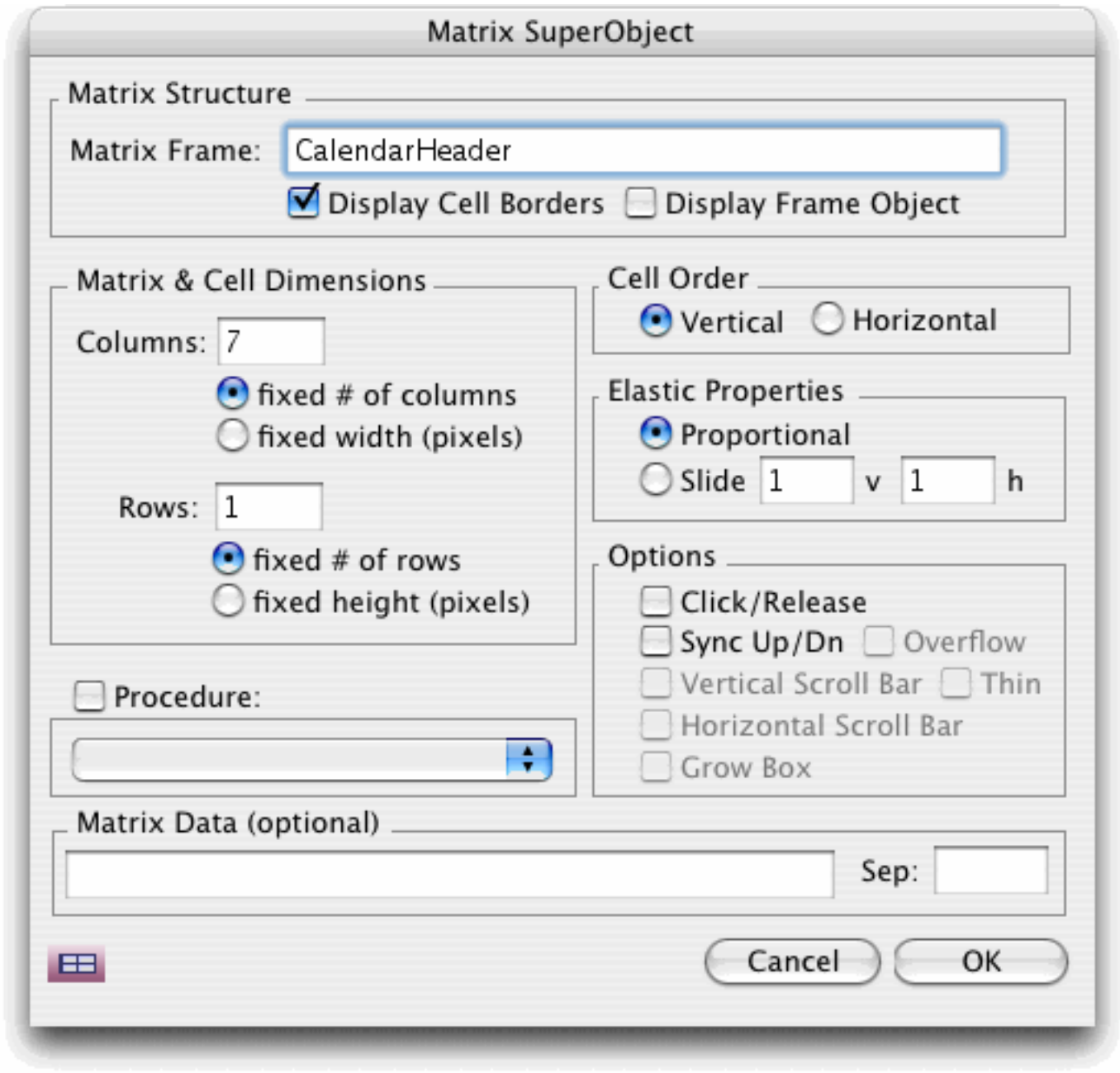
The 12 pt text looks a bit big. Change the text to 9 pt. You'll notice that nothing happens when you make this change. To actually see the change you'll need to force the window to redisplay. The easiest way to do that is to scroll down a page, then scroll back. On Macintosh systems you can also click on the window minimize icon twice. Either way, once the screen updates you'll see the new look.



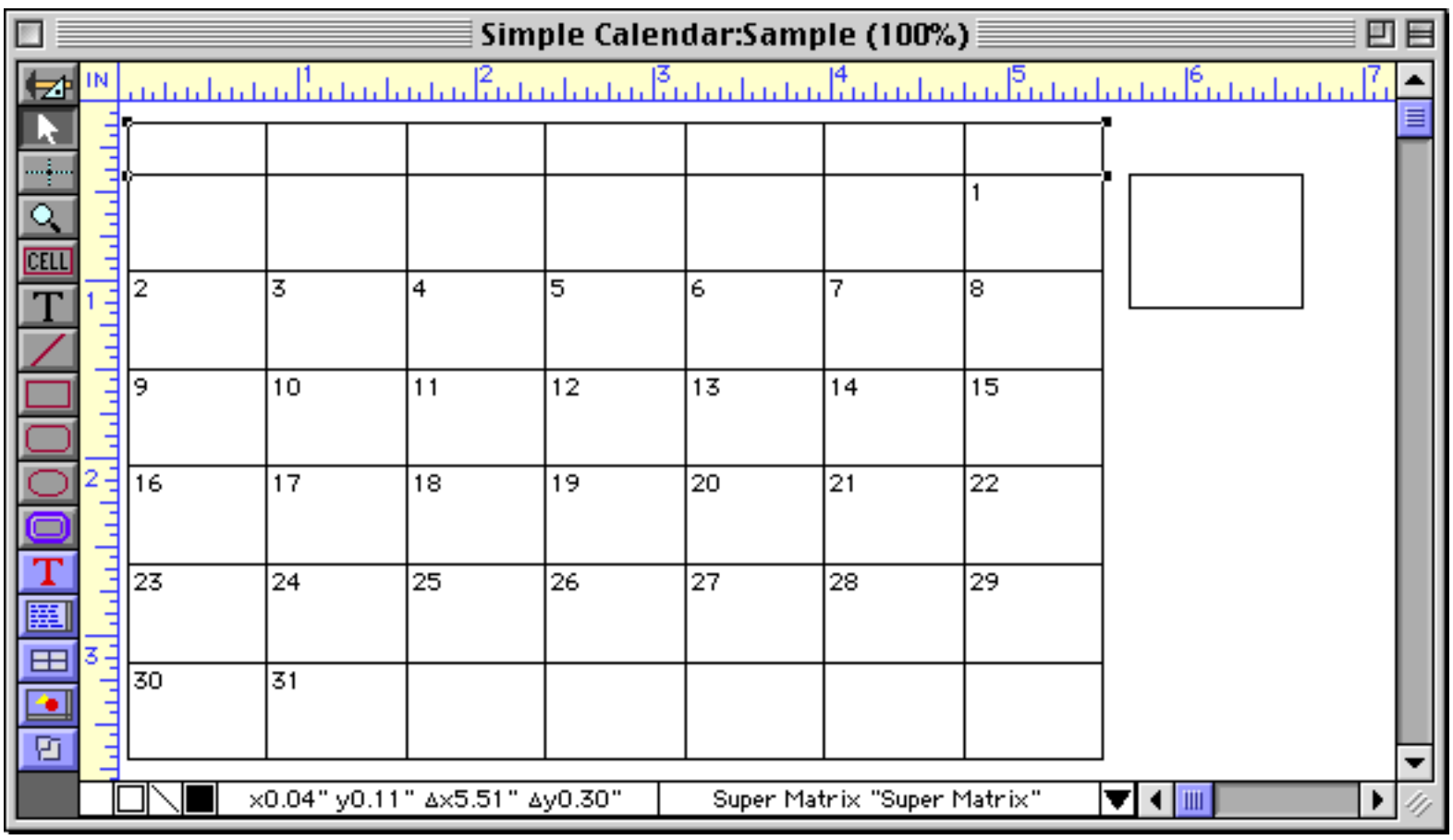
Next our calendar needs a header for the days of the week. You could create the header with separate text objects, but we'll create it with another matrix. Select the **Matrix** tool and drag to create the new matrix.


Simple Calendar:Sample (100%)
IN
CELL
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
x0.04" y0.15" Δx5.50" Δy0.25"
Super Matrix "Super Matrix"

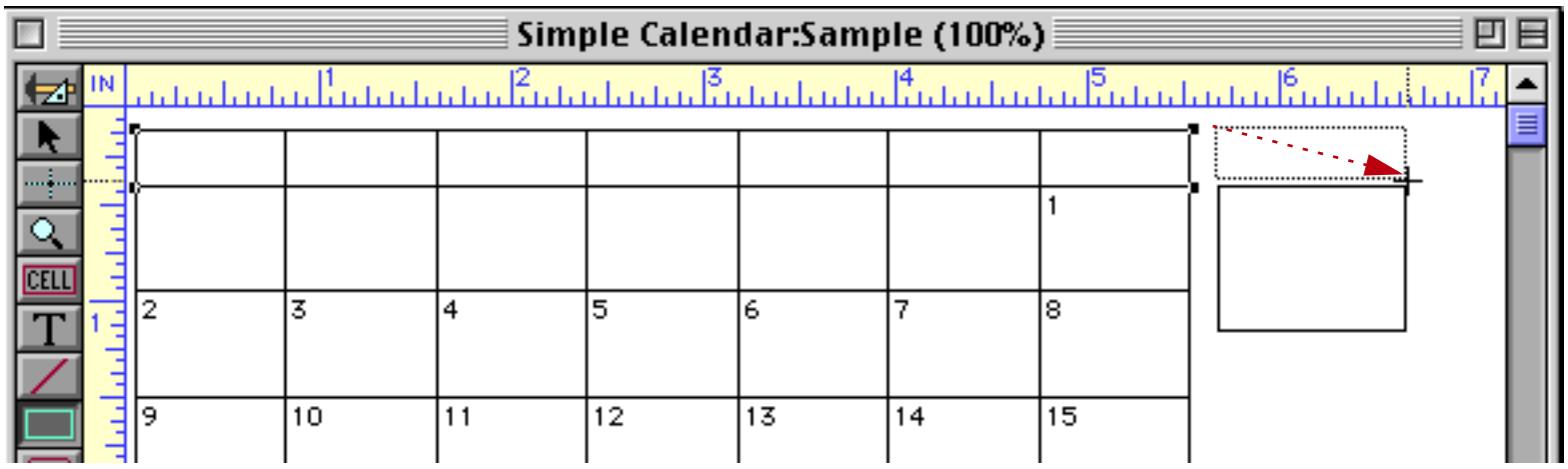
This new matrix will have only one row with seven columns (one for each day of the week). Type in Day Of Week for the matrix frame, which we will create in a moment.



Once the new matrix is created you may need to nudge it a bit to get it perfectly lined up with the main body of the calendar (see "Nudging an Object (or Objects)" on page 509 and "Nudging the Size of an Object" on page 513).



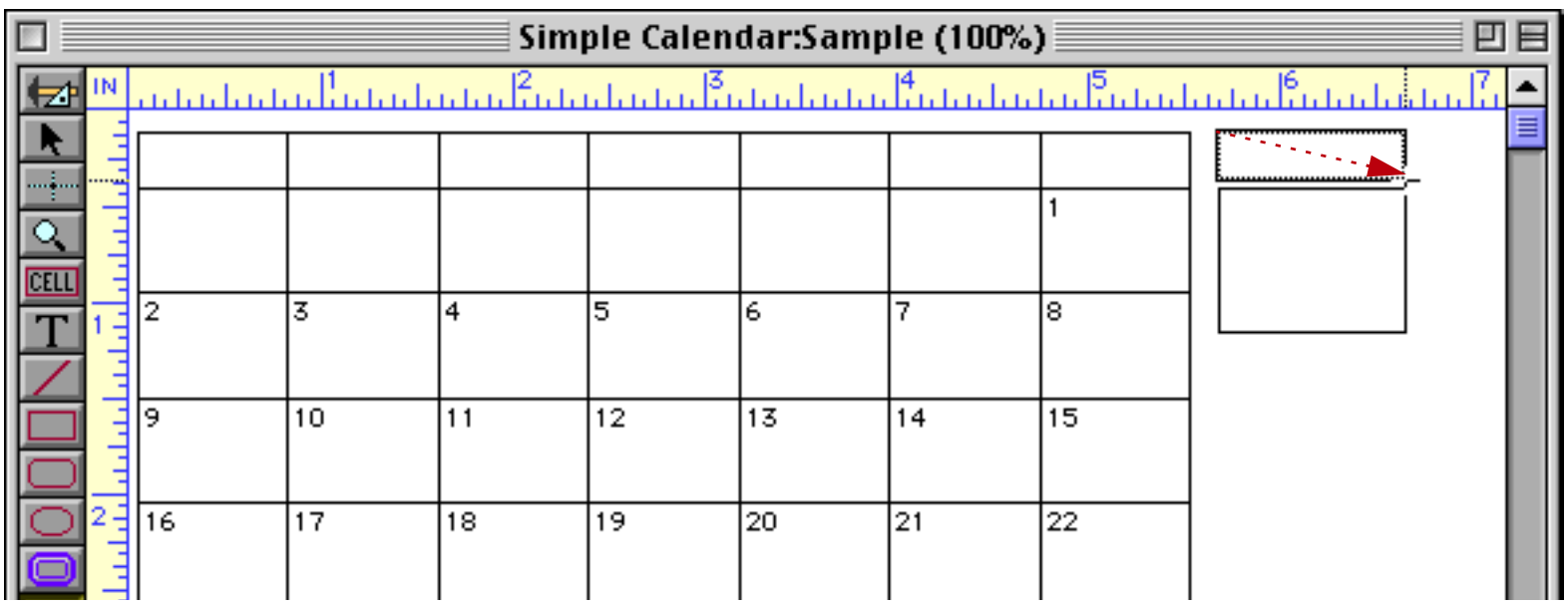
The next step is to create the second matrix frame. Start by creating a rectangle.



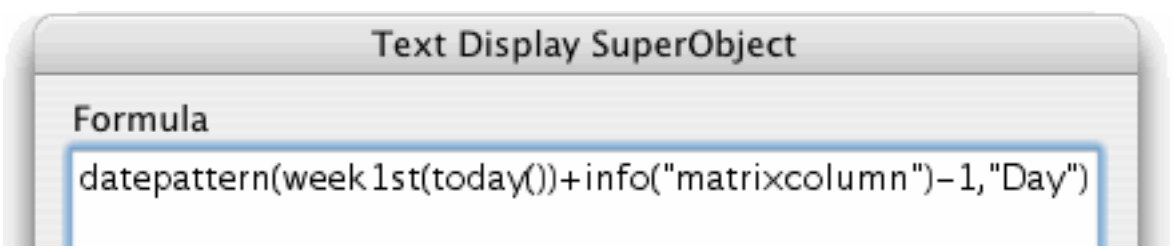
Set the name of the rectangle to Day Of Week (see "Object Type / Object Name" on page 533).



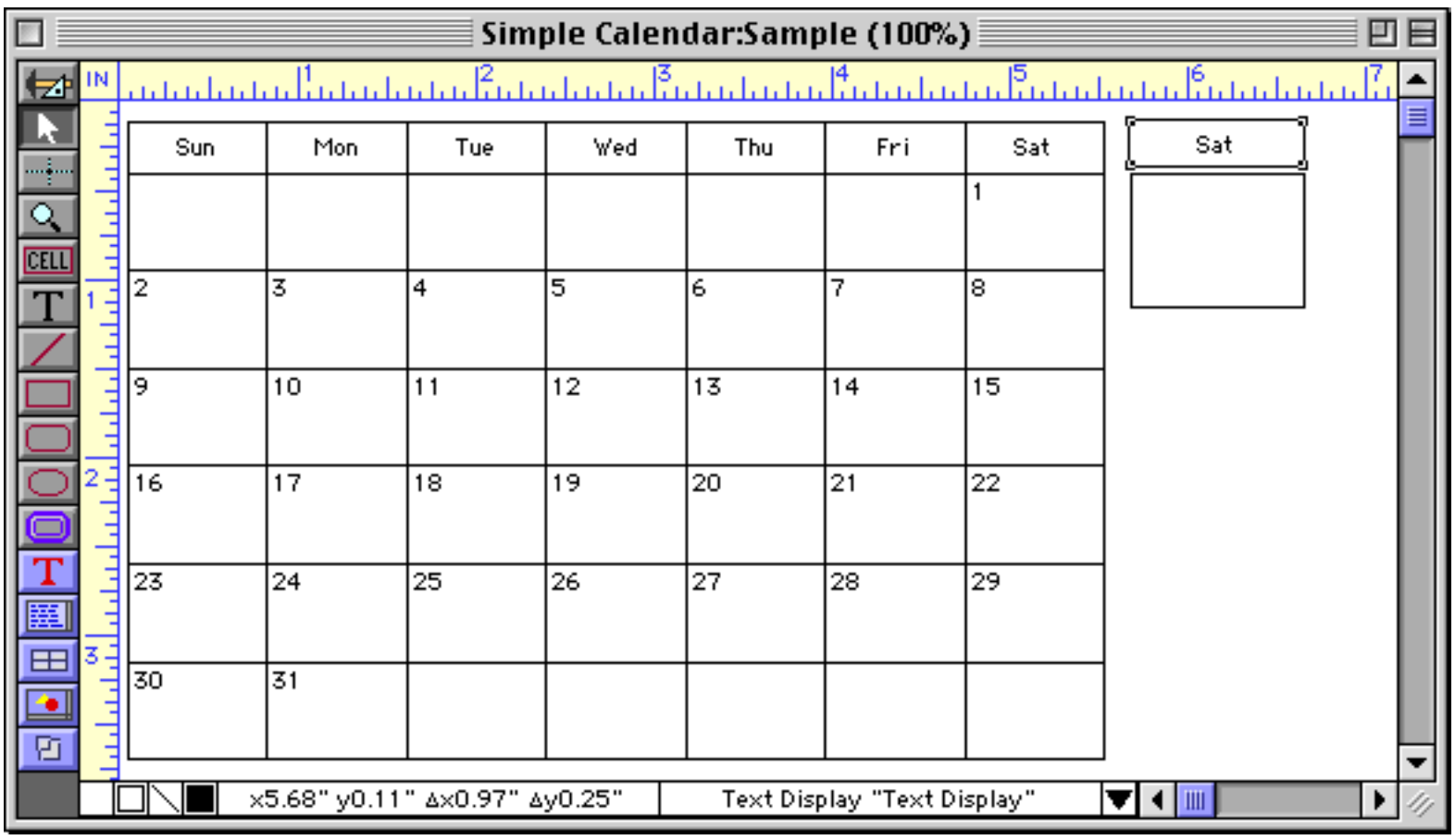
The next step is to add a Text Display SuperObject to display the day of the week (Sun, Mon, Tue, …). See “Creating and Modifying Text Display SuperObjects” on page 608 to learn how to create such an object. The Text Display SuperObject should be created to almost fill the frame rectangle.



Use the formula shown in the illustration below to display the day of the week. You’ll also want to set the alignment to centered, as shown below.



Depending on where your window is on the screen, you may need to scroll the form down a page and then back to see the finished result.



Believe it or not, this entire calendar has been created with only six graphic objects — two matrixes, two rectangles, and two text objects.

Scroll Bars

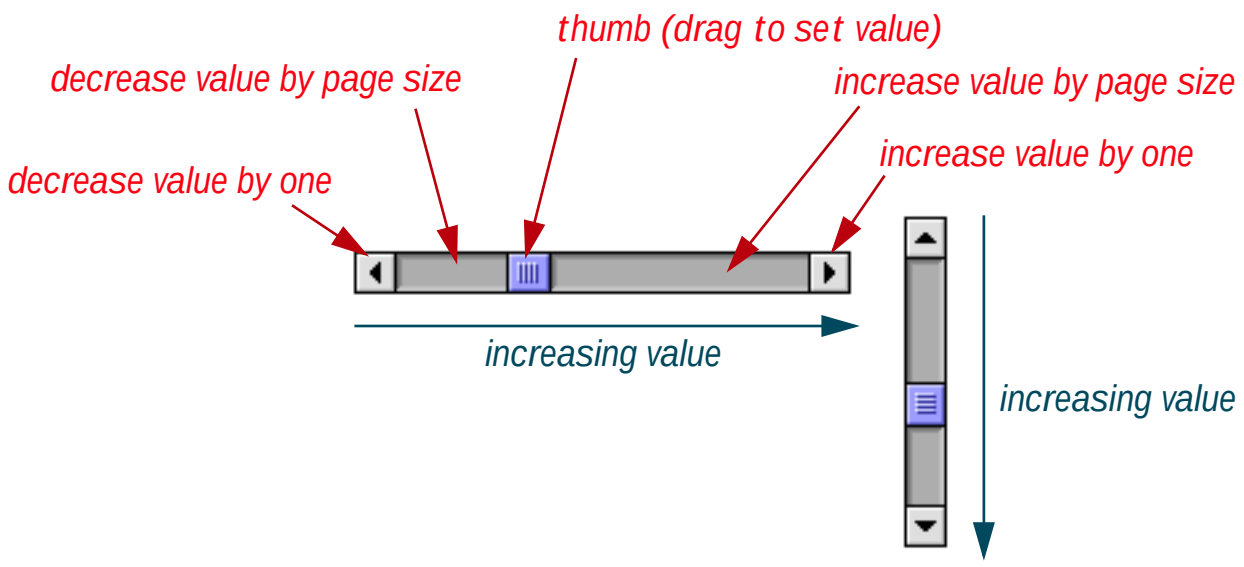
Scroll bars are another common user interface element, and in fact may be included as part of several other user interface elements (text editor, flash art, lists). The **Scroll Bar SuperObject™** allows you to create a scroll bar by itself as an independent user interface element.

Scroll bars can be used two ways. One is to act as a graphical method of displaying and modifying a numeric value. For example, you could set up a scroll bar that allowed you to display and control the Temperature field in a database. Instead of typing in the temperature, you would simply dial it in.

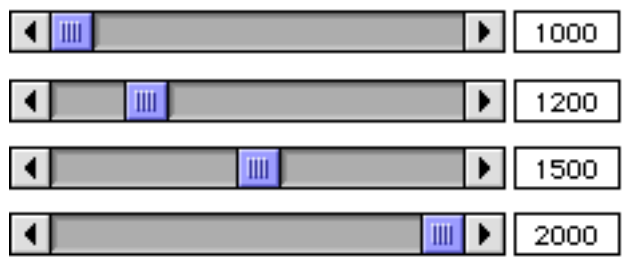
The second way to use a scroll bar is to scroll something. A Scroll Bar object can be combined with other objects (a matrix, for example) to produce a scrolling section within a form.

Scroll Bar "Theory"

A scroll bar has two arrows at the end, and a sliding "thumb" that can be positioned anywhere between the two ends. The position of the thumb moves corresponds to the current "value" of the scroll bar. As the value increases, the thumb moves down or to the right (depending on the direction of the scroll bar). Conversely, moving the thumb (by dragging it) changes the current value associated with the scroll bar. This value is kept in a field or variable.



The scroll bar value is not unlimited, but ranges between preset minimum and maximum values. The minimum value corresponds to a thumb position all the way to the top (or left), while the maximum value corresponds to a thumb position all the way to the bottom (or right). Values in between the minimum and maximum represent intermediate thumb positions. Only integer values are allowed; fractional values (like 2.49) are not allowed. The minimum and maximum values may be preset to any value between 1 and 65535. For example, if the minimum is set to 1000 and the maximum set to 2000 then the scroll bar will have 1001 possible positions. A thumb position halfway in the middle corresponds to 1500.

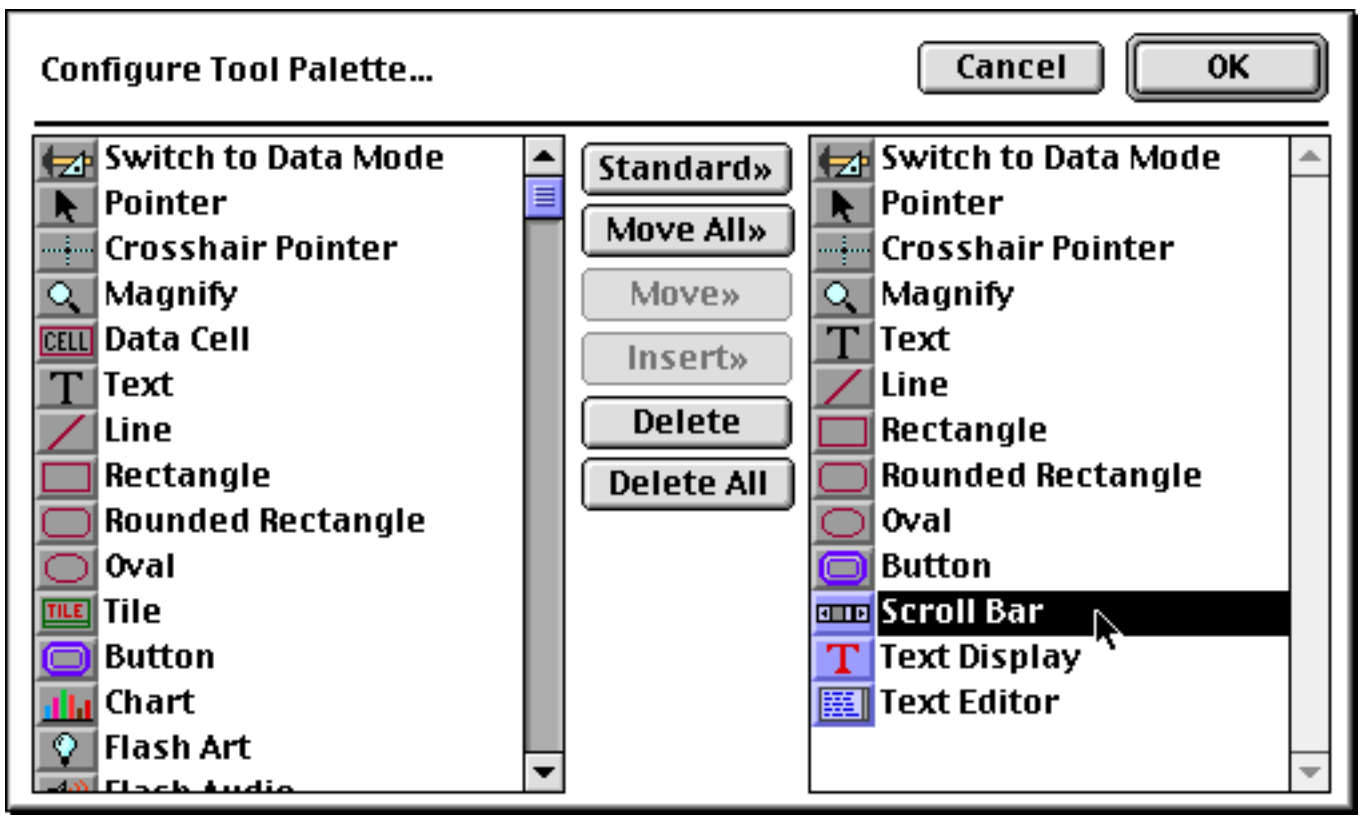


Each time the user presses on one of the scroll bar arrows the value will increase or decrease by one. If the minimum and maximum are set close together (for example 1 and 10) you'll be able to see a definite jump each time the arrow is clicked. If the minimum and maximum are far apart (for example 1 and 1000) you may have to click several times to see even a tiny change in the thumb position.

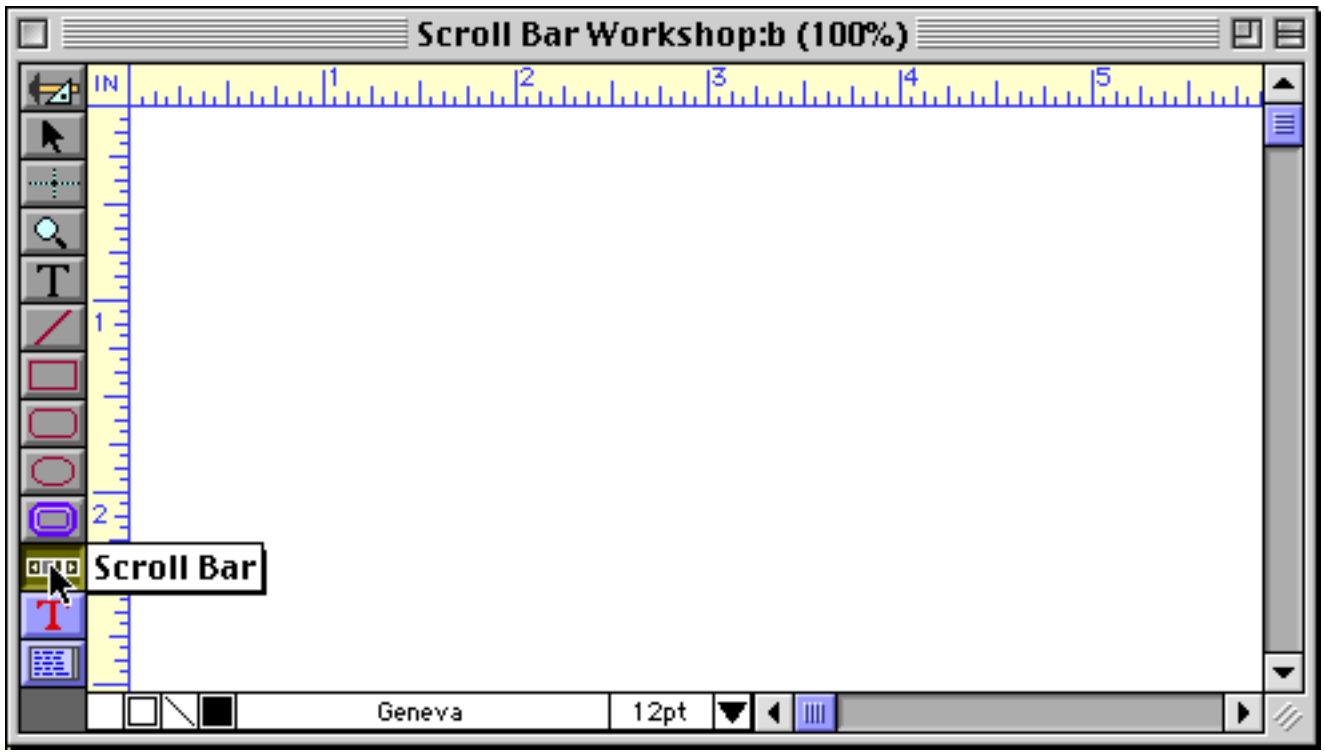
If the user presses on the gray area above or below the thumb the scroll bar value changes by a larger, predetermined amount. This usually corresponds to scrolling a page at a time, instead of a line at a time. The exact value change is set in the scroll bar's configuration dialog.

Creating Scroll Bar SuperObjects™

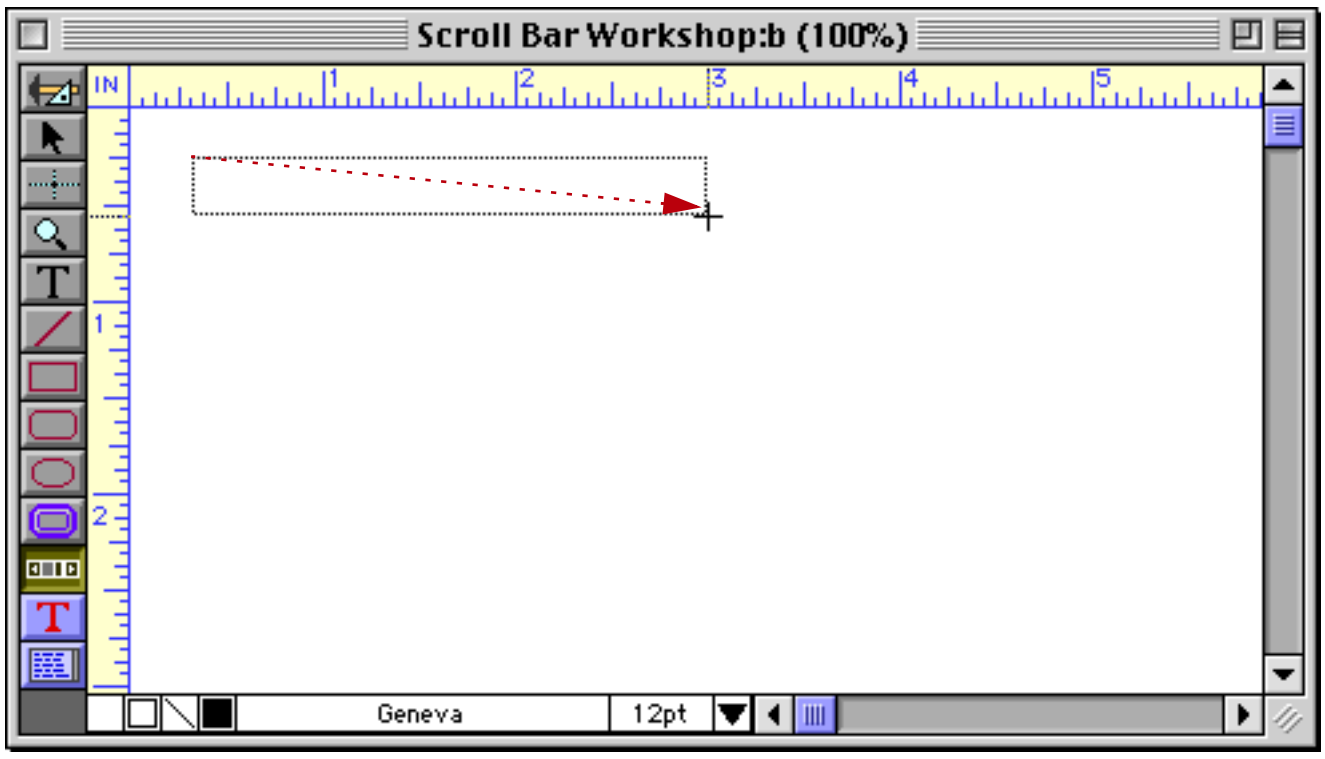
Scroll Bar objects are created just like any other SuperObject™. First make sure that the **Scroll Bar** tool is installed in the tool palette (see "Customizing the Tool Palette" on page 497).



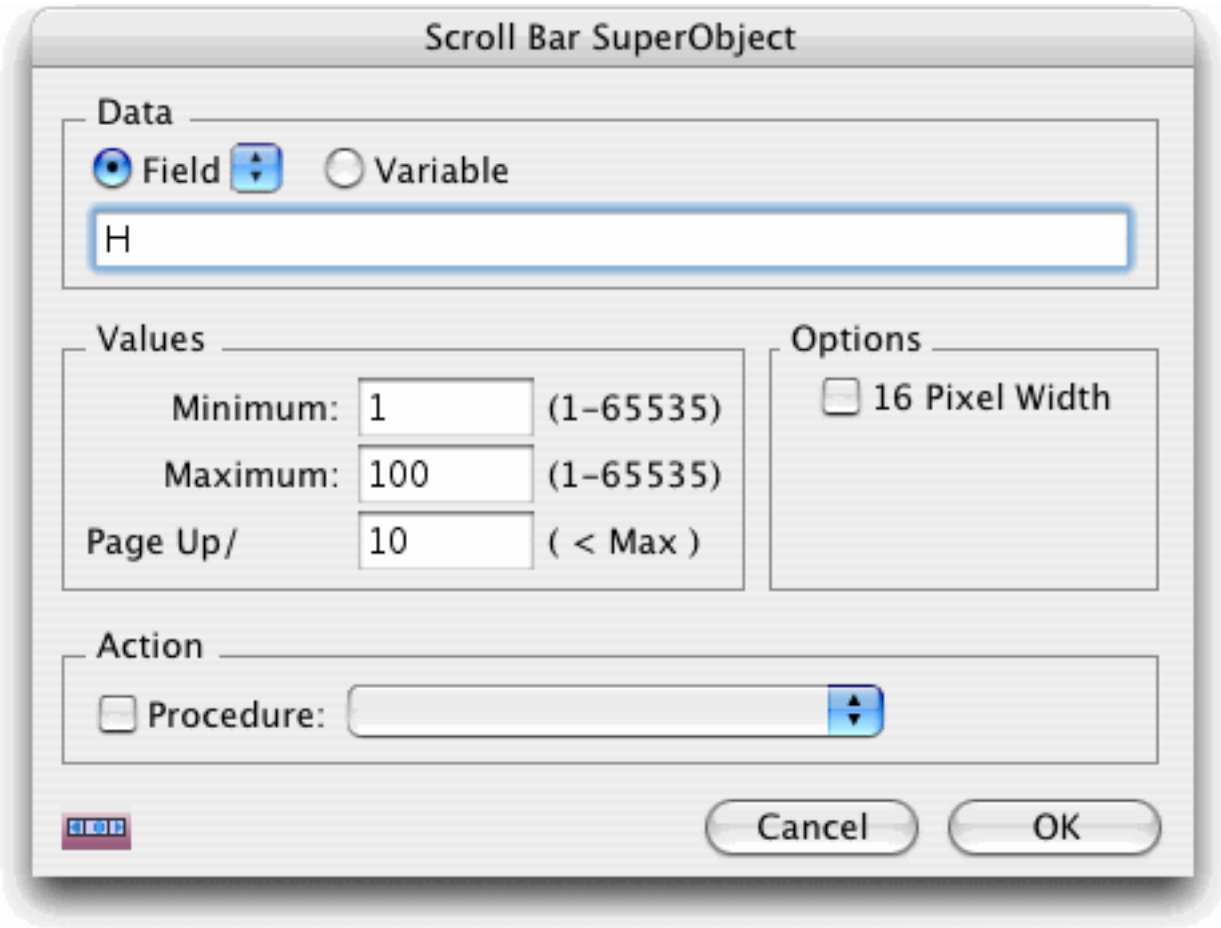
Now that the tool is added to the palette you can select it.



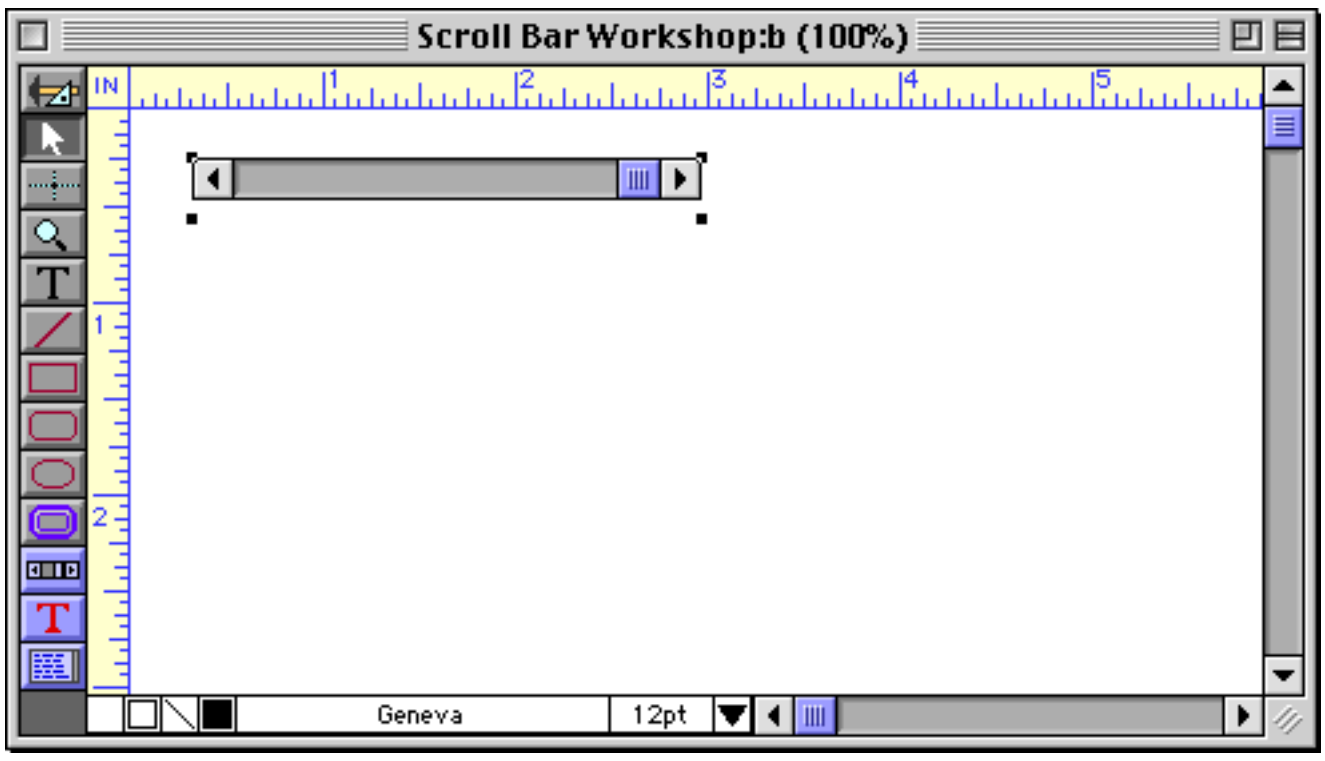
Next drag the mouse across the form in the spot where you want the scroll bar to appear. If you want a vertical scroll bar, drag a tall skinny box; if you want a horizontal scroll bar drag a squat wide box.



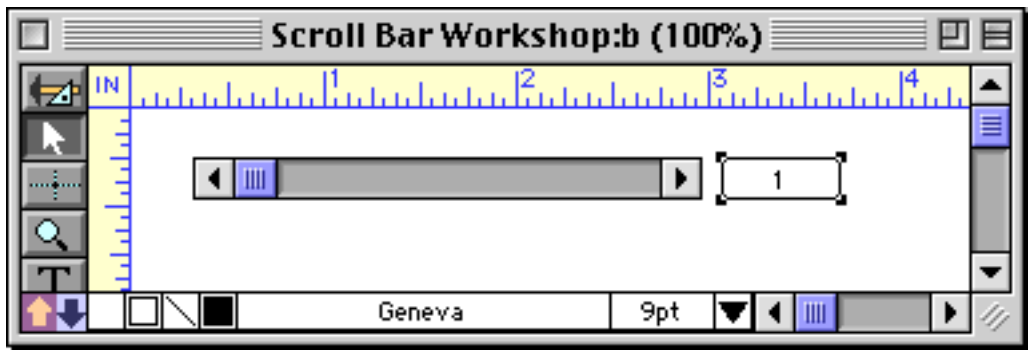
When you release the mouse, the Scroll Bar configuration dialog appears.



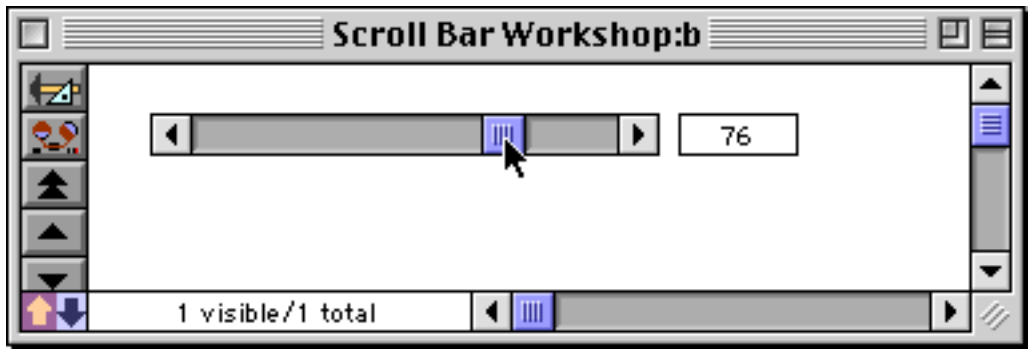
Select a field (must be a numeric field) or type in a global variable name (see “Variables” on page 1199 and “Variables” on page 1370) and then press **OK**. The new scroll bar appears.



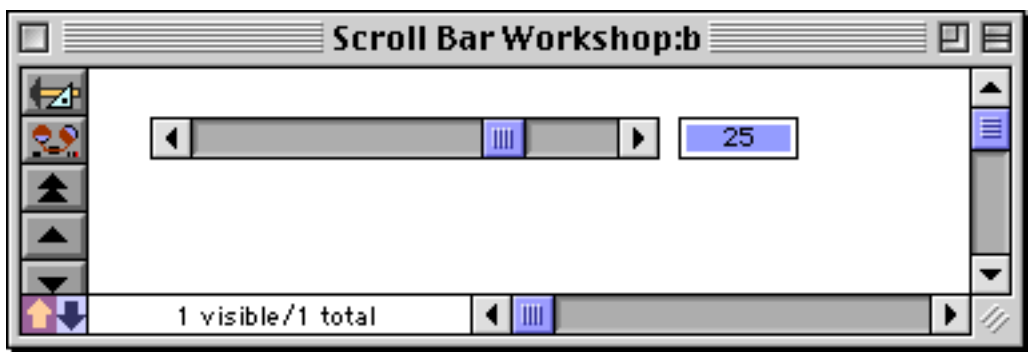
In this case we linked the scroll bar to a numeric field: H. We can add a Text Editor SuperObject to display the value of that field (see “Creating and Modifying Text Editor SuperObjects” on page 639).



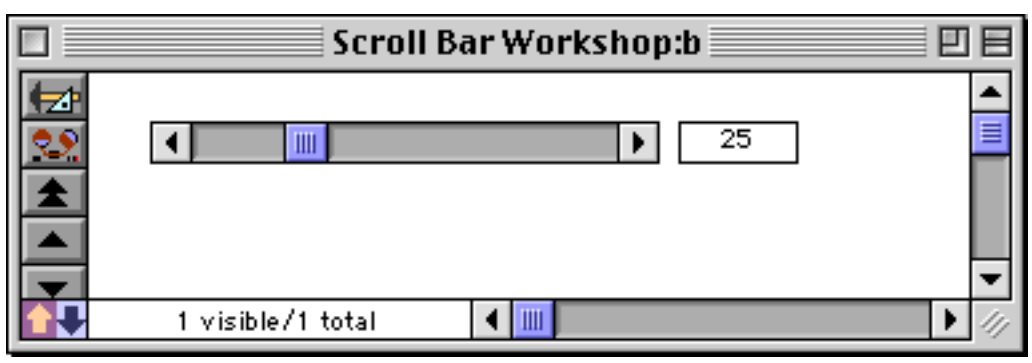
Now switch to Data Access Mode to try out the scroll bar. You can drag the thumb to any spot to set the value of the H field.



You can also move the position of the scroll bar thumb by typing in a value from 1 to 100. For example, you could type 25 into the Text Editor SuperObject.

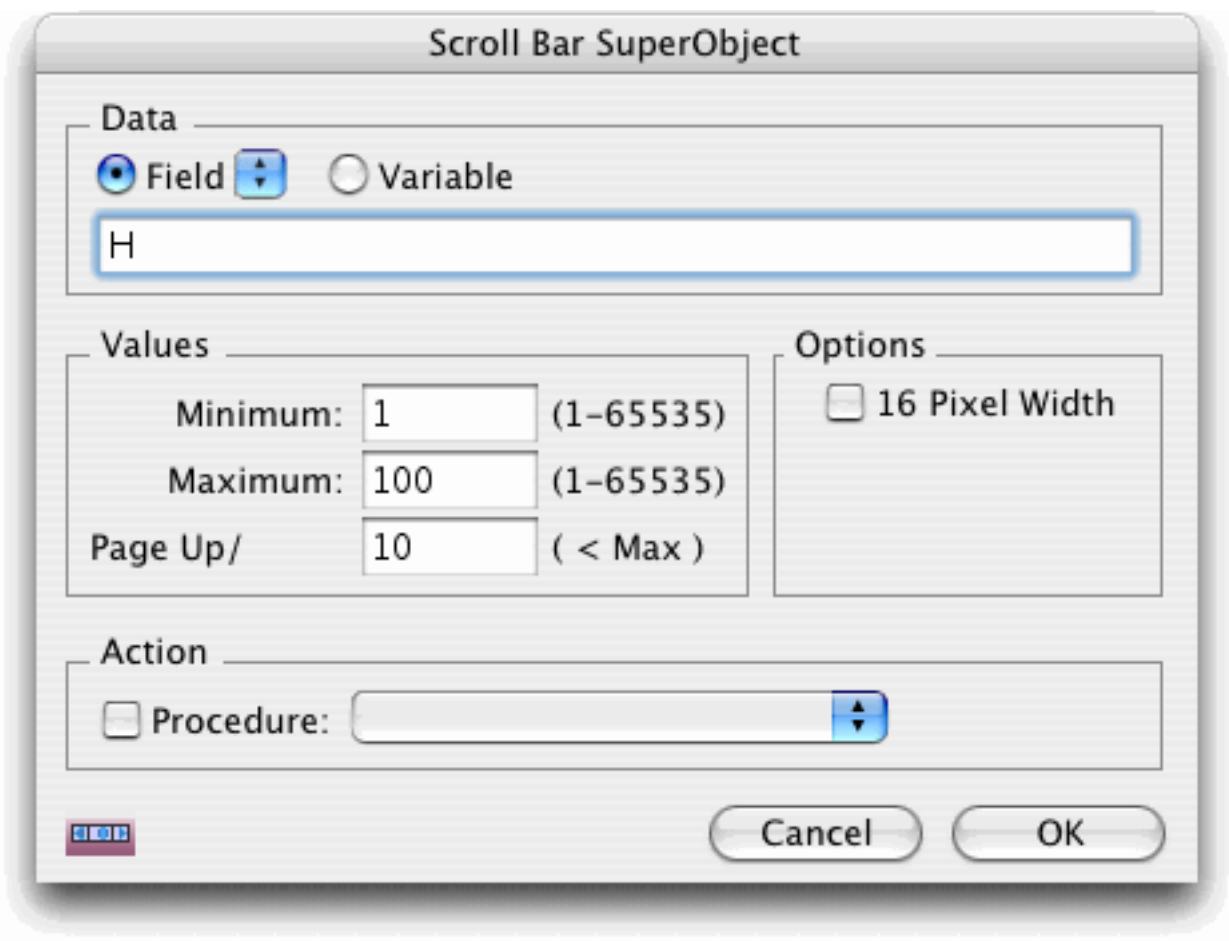


When you press the **Enter** key the scroll bar thumb will hop to the 25% position.



Scroll Bar Options

The SuperObject™ Scroll Bar dialog is divided into several sections. To re-open this dialog for a scroll bar you have already created, select the **Pointer** tool and then double click on the scroll bar object.



Data

This section of the dialog specifies the field or variable associated with this scroll bar. Type the name of the field or variable into the box (or select the field name from the pop-up menu next to the Field checkbox). If you choose a field, the field must be numeric with zero digits (see “Numeric Data” on page 289). If the scroll bar is associated with a variable that has not been created with a procedure, Panorama will automatically create a global variable with this name whenever the scroll bar appears. This global variable can be used in formulas and procedures just like any other global variable.

Min

This is the minimum value for the scroll bar, which corresponds to the scroll bar thumb position at the far left or top. The minimum value must be between 1 and 65535, and must be less than the maximum value.

Max

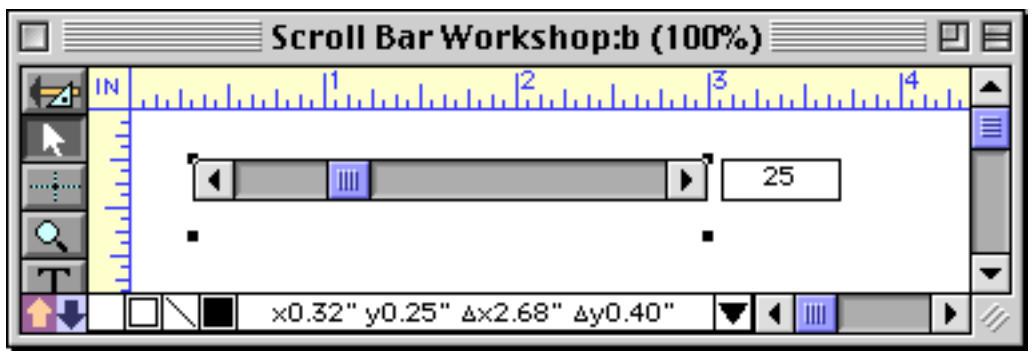
This is the maximum value for the scroll bar, which corresponds to the scroll bar thumb position at the far right or bottom. The maximum value must be between 2 and 65535, and must be greater than the minimum value. Setting the maximum value close to the minimum will produce a “grainy” scroll bar with big jumps as you press the arrows. Setting the maximum value far from the minimum will produce a “precise” scroll bar with tiny or even imperceptible movement as you press the arrows.

Page Up/Down

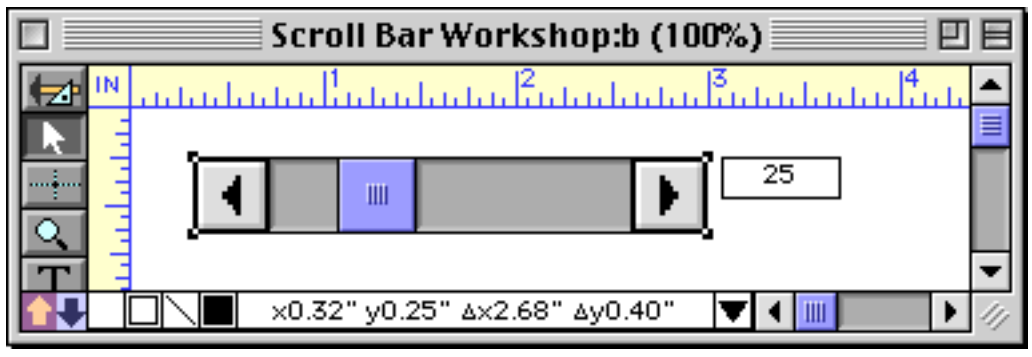
This is the amount the scroll bar value will change when the user presses on the gray area above and below the thumb. This value must be less than the difference between the minimum and maximum values, and is often about 1/10th of that difference.

16 Pixel

If this checkbox is turned on, Panorama will limit the scroll bar width to 16 pixels, even if the object is larger. This is the normal size for most scroll bars. The illustration below shows a scroll bar with the 16 Pixel option enabled. Even though the scroll bar object is 29 pixels high, the scroll bar is only 16 pixels high.



Here is the same scroll bar with the 16 Pixel option turned off.

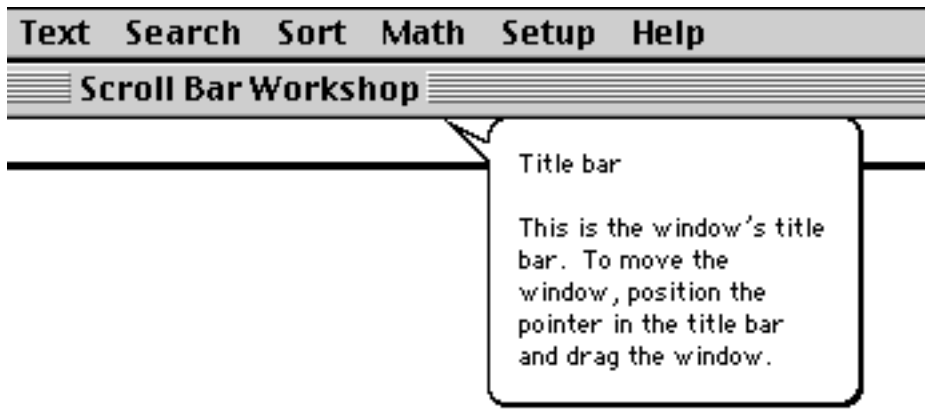


Procedure

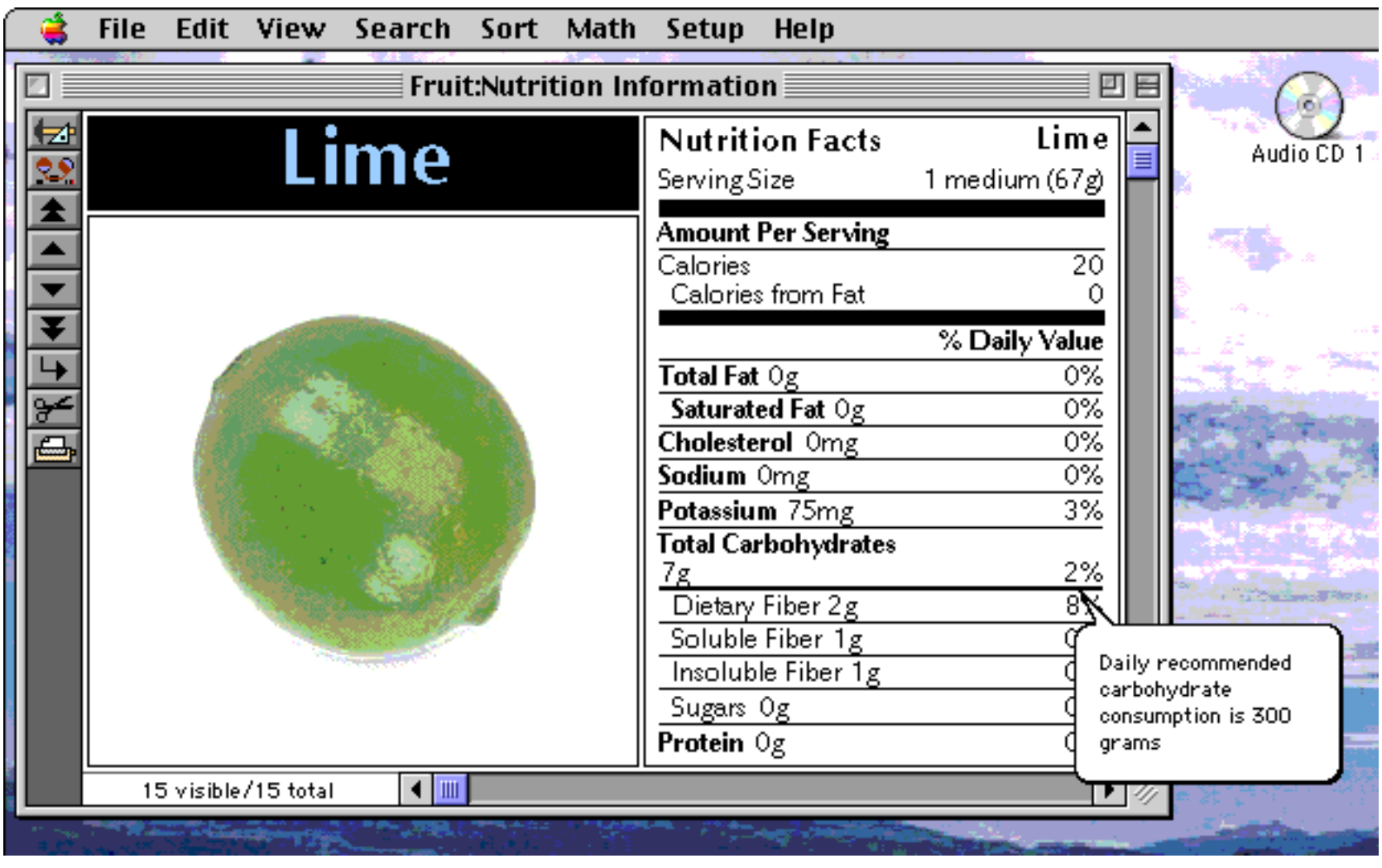
This section of the dialog allows you to specify a procedure that will be triggered every time the user clicks on the scroll bar.

Balloon Help

The Macintosh has a feature called **balloon help** that can help in learning how to use a program. When the **Show Balloons** option is turned on (Help menu), little pop-up balloons with help messages appear as you move the mouse across the screen.



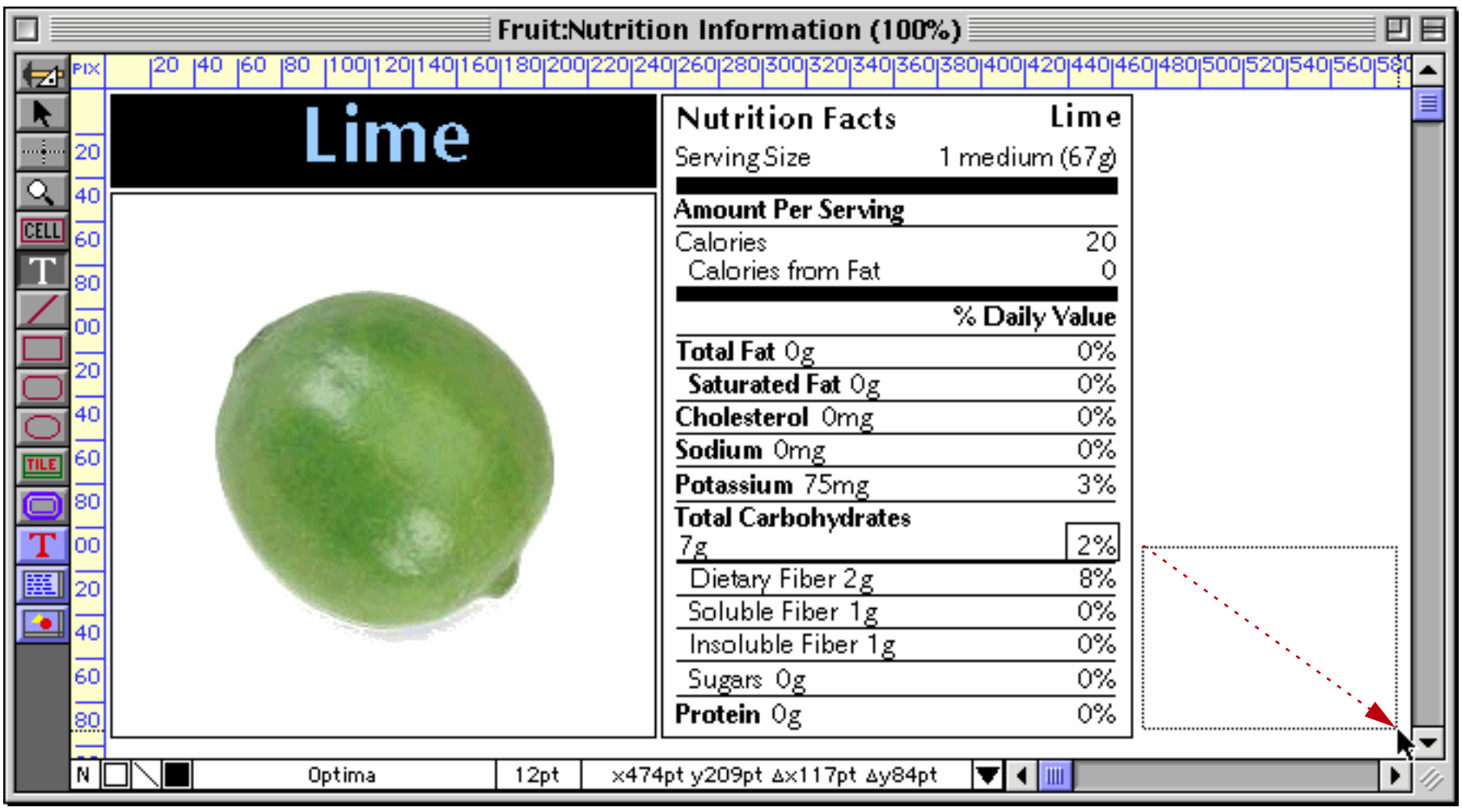
Panorama allows balloon help to be added to any form. You can make any message you want pop-up over different areas of a form.



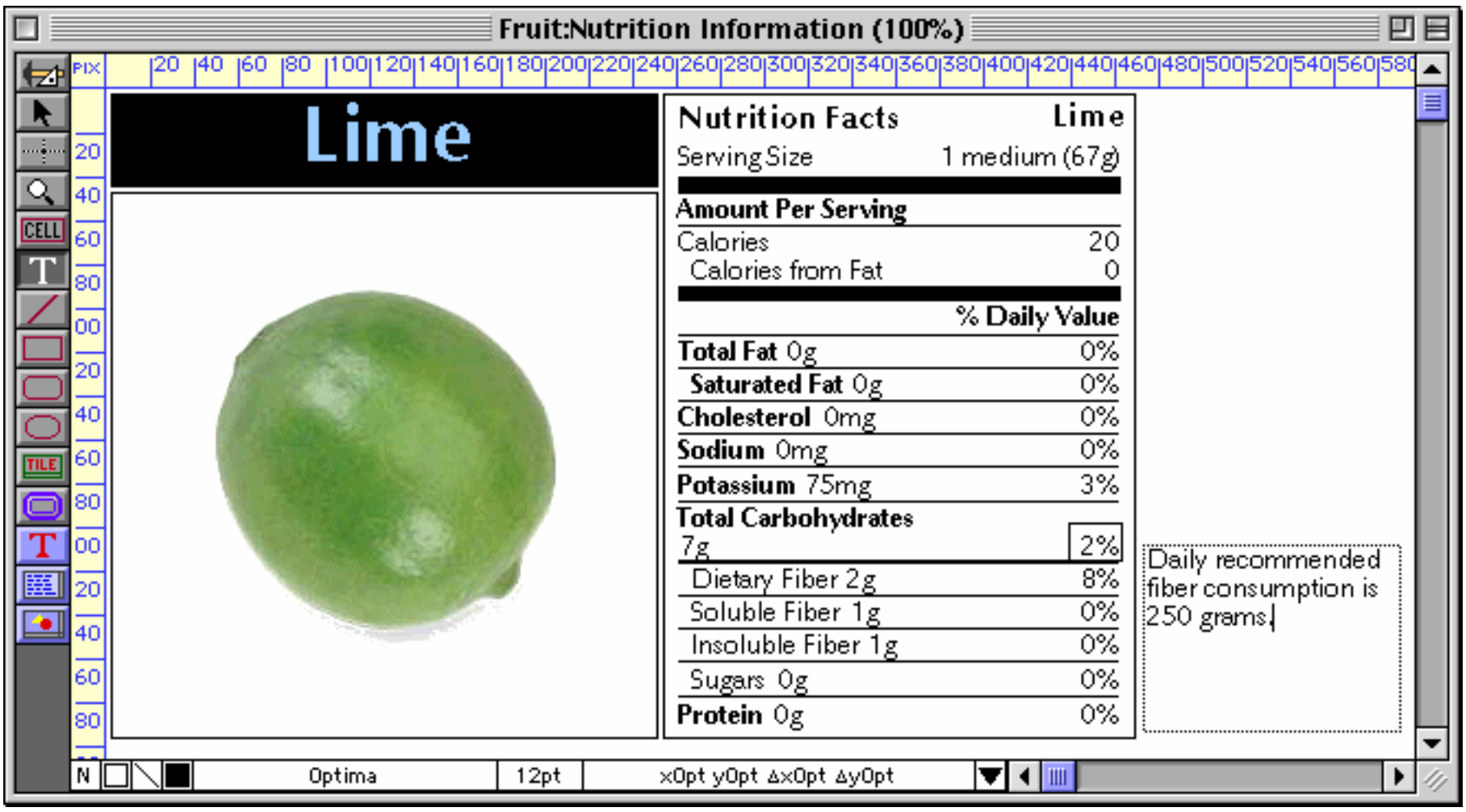
Creating Balloon Help Objects

Balloon help is added to a form by creating balloon help objects. You must create a separate balloon help object for each area where you want a different balloon help message to appear. For example, if you want to create balloon help for a button, you must create a balloon help object in the same location as the button (either on top of or behind the button).

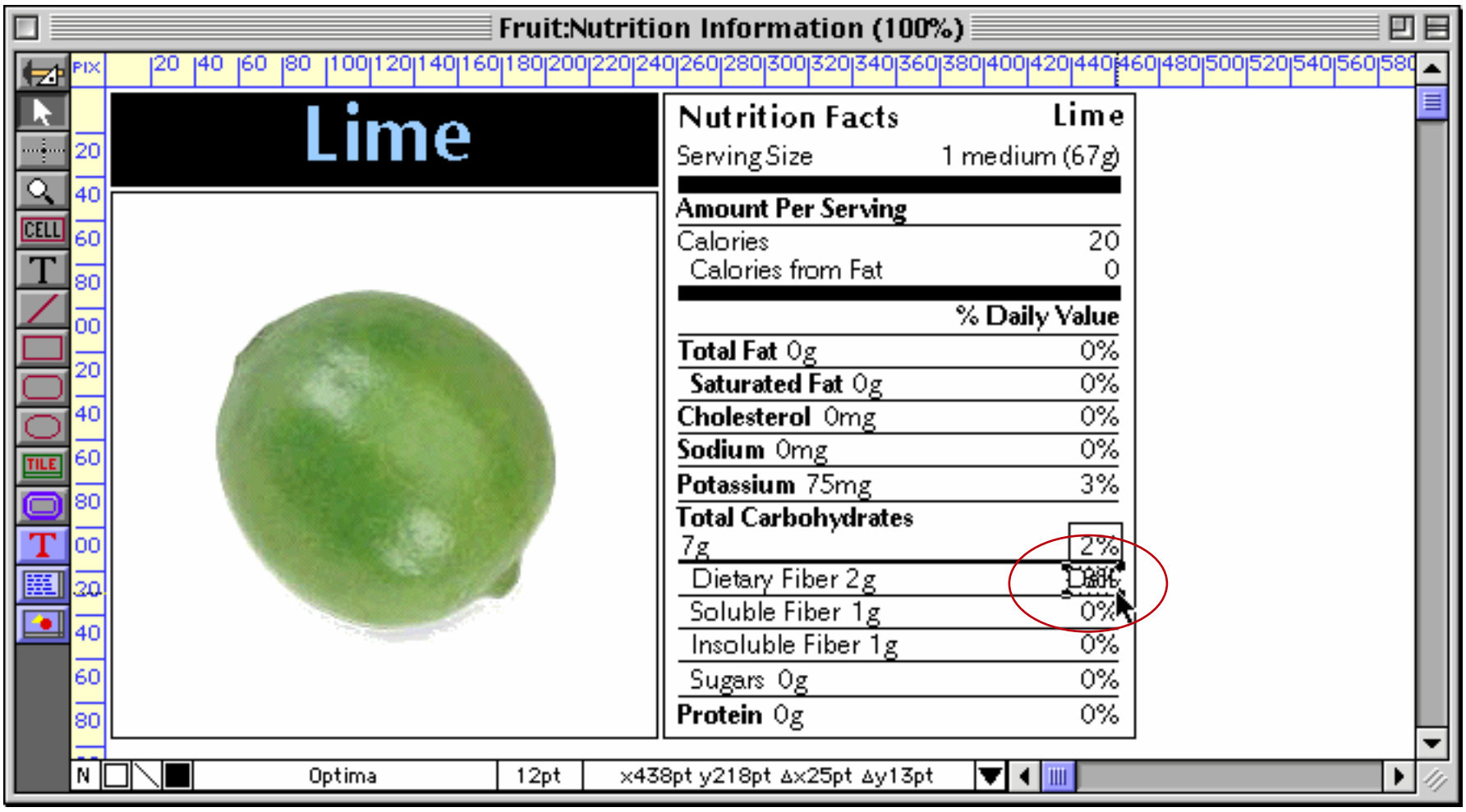
Balloon help objects are created in a different way than any other kind of object. First, you create a text object with the **Text** tool.



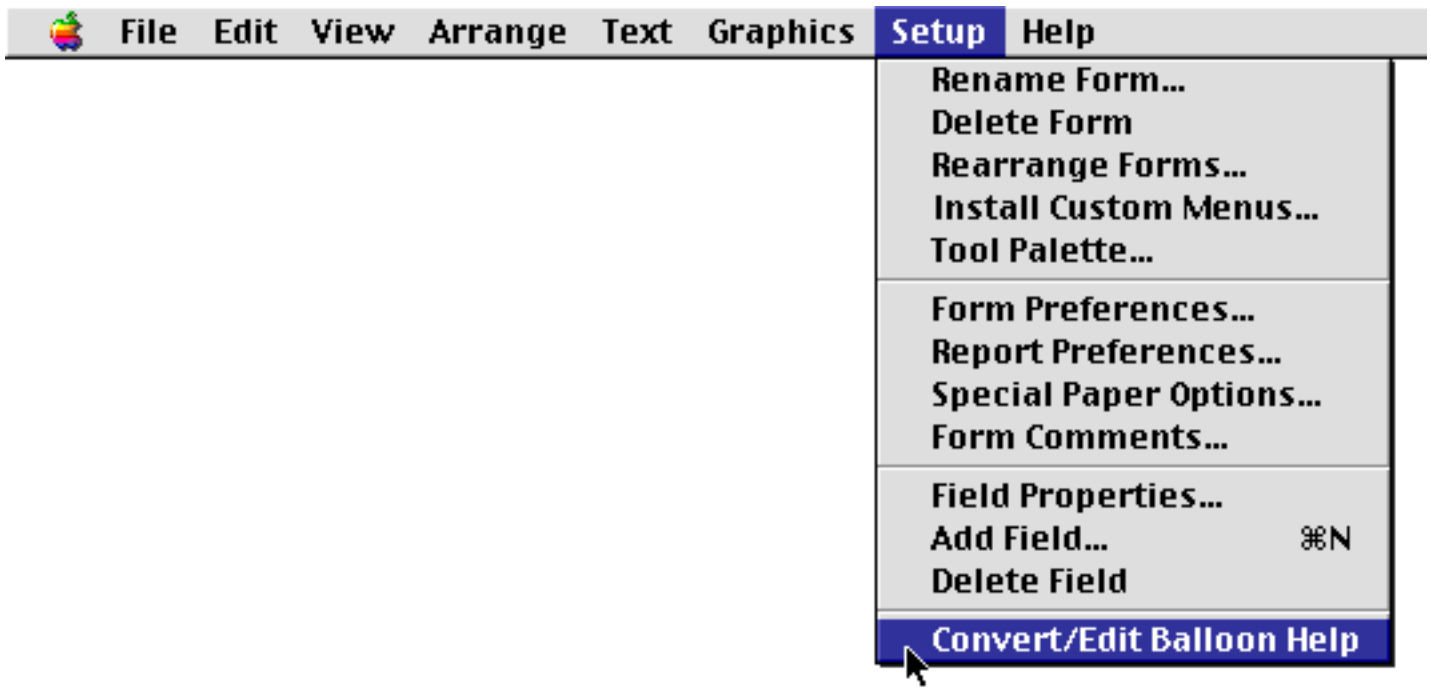
Type in the text that you want to appear in the balloon (you can edit this later).



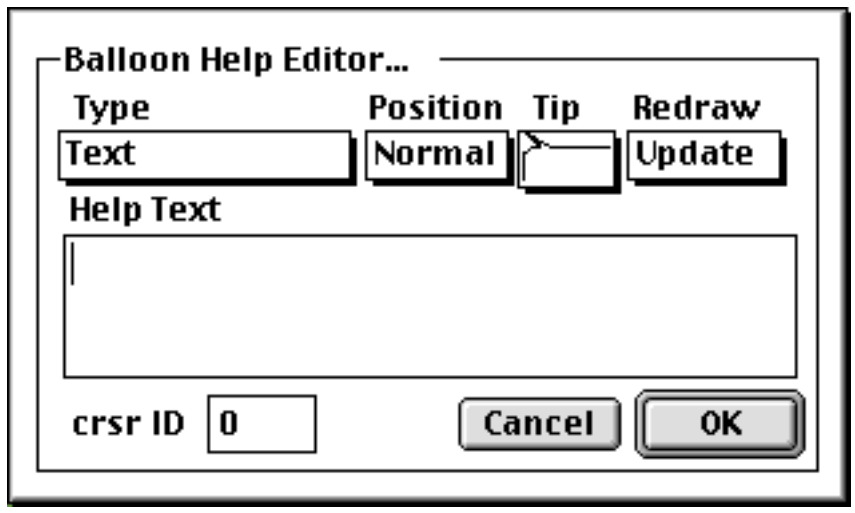
Once the text is created, select the text object with the **Pointer** tool. If necessary, move and resize the object to position it over the appropriate area of the form.



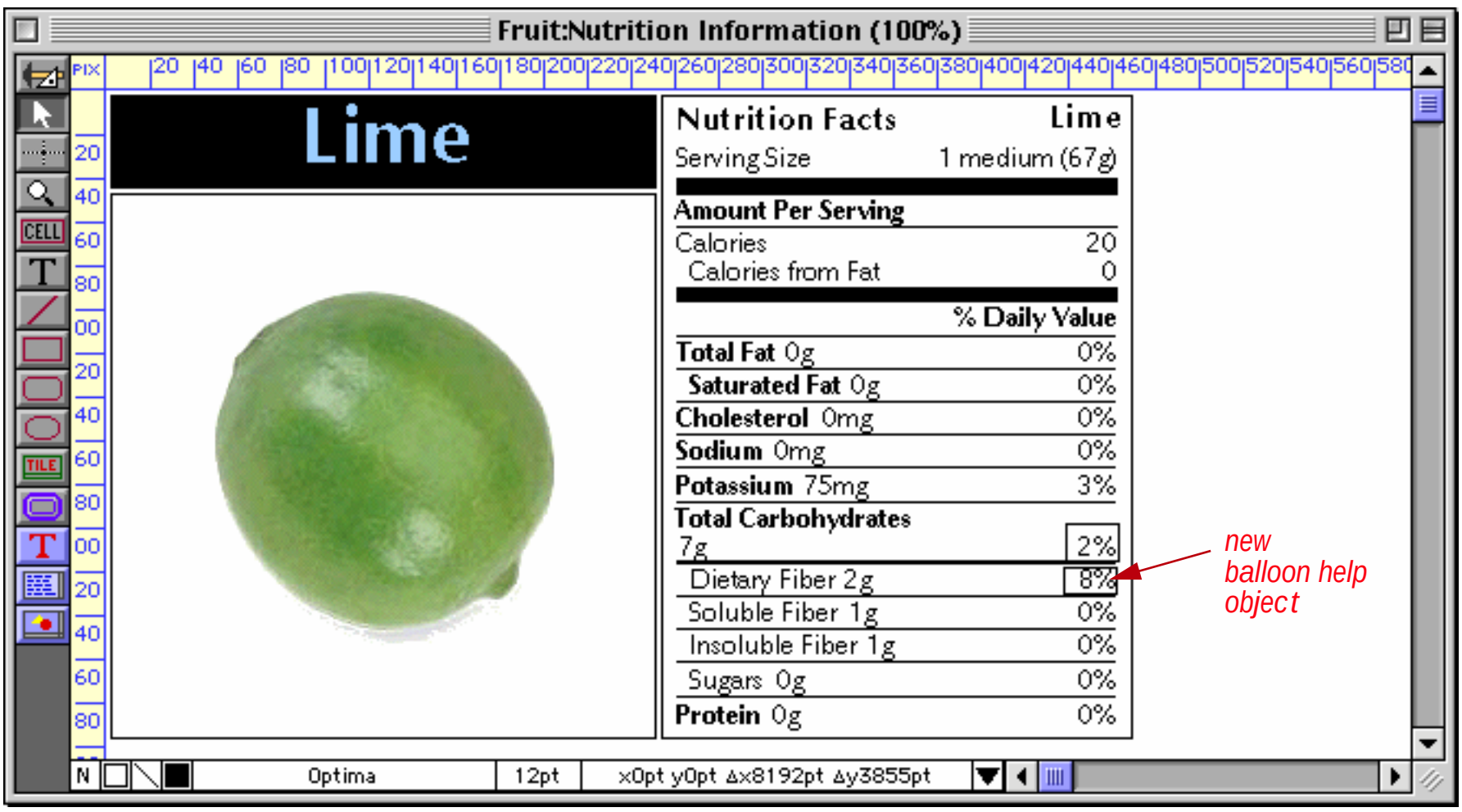
With the auto-wrap text object selected, choose **Convert/Edit Balloon Help** from the Setup menu.



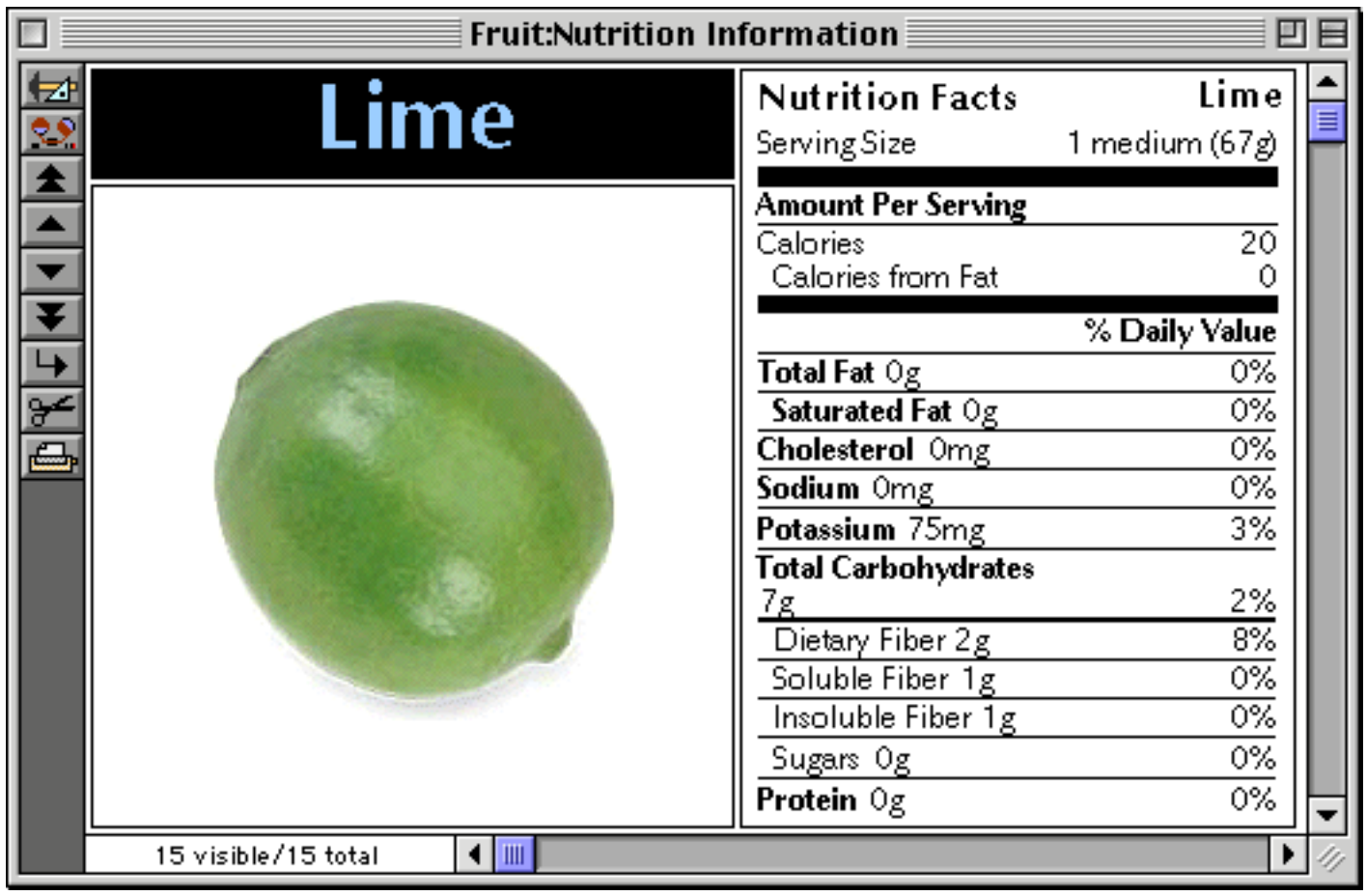
Panorama will convert the text object into a balloon help object, and at the same time it opens a dialog that allows you to specify balloon help options.



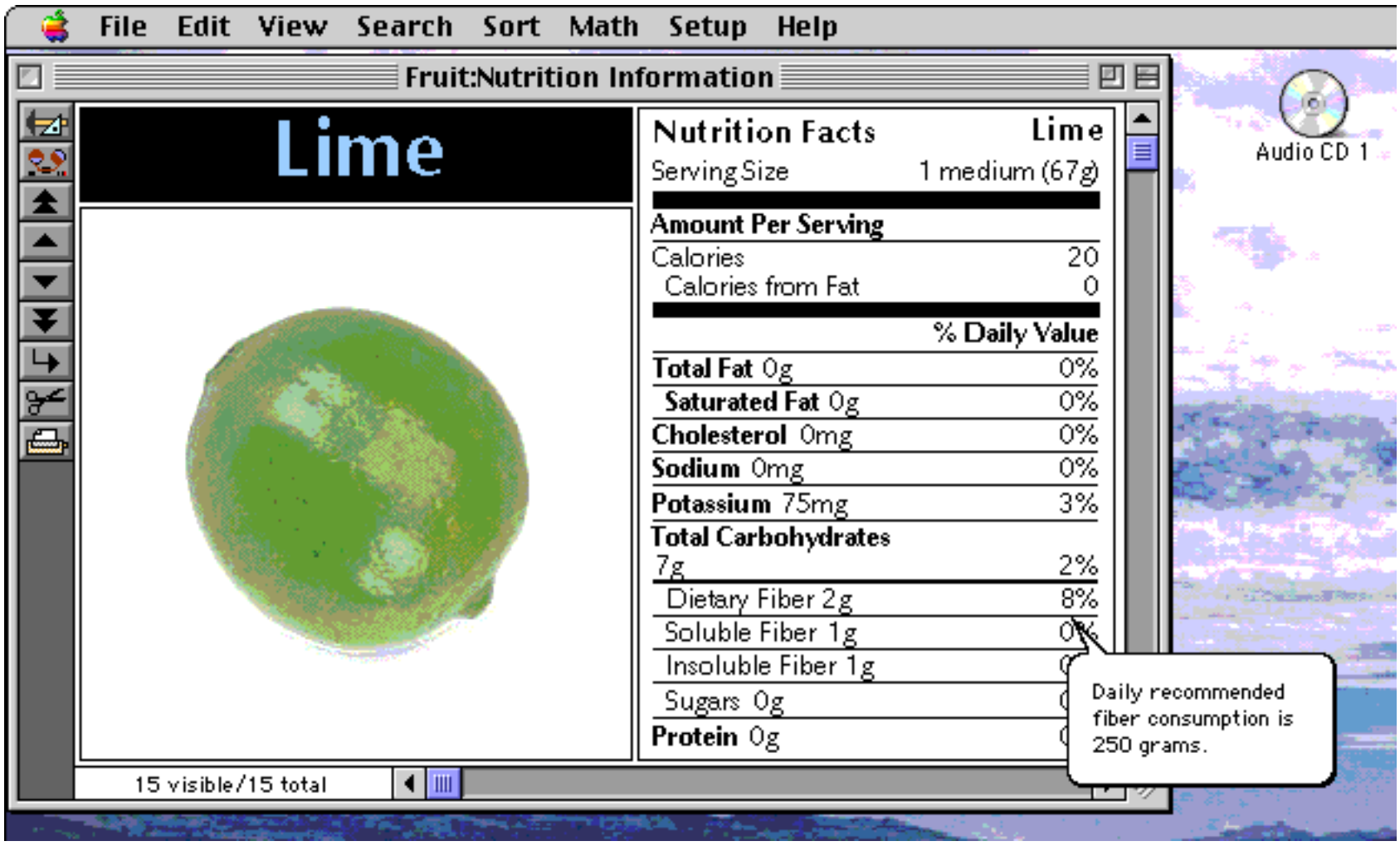
Don't worry about the fact that the Help Text are is empty. Just leave it blank and press **OK**. The text object has been converted into a balloon help object, which appears with a black line around it.



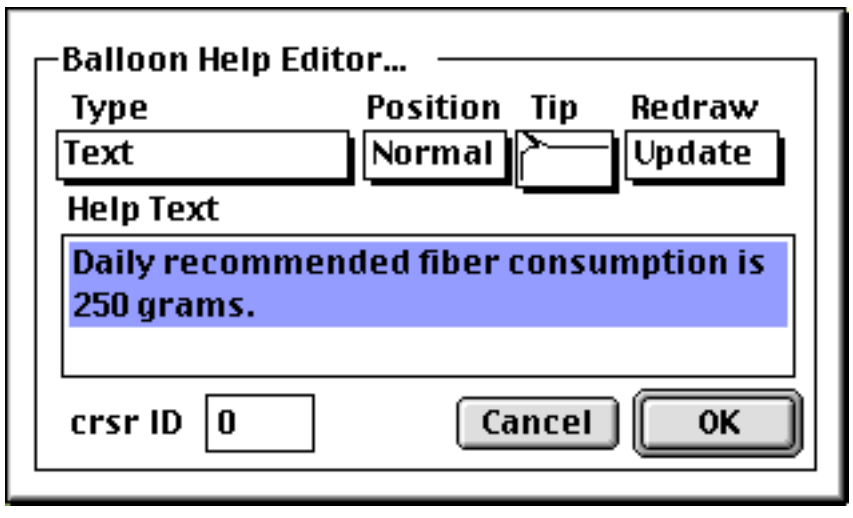
When you go to Data Access Mode the black line disappears, and the balloon help object becomes completely invisible.



Turn on the **Show Balloons** option (Help menu) to try out your new help balloon.

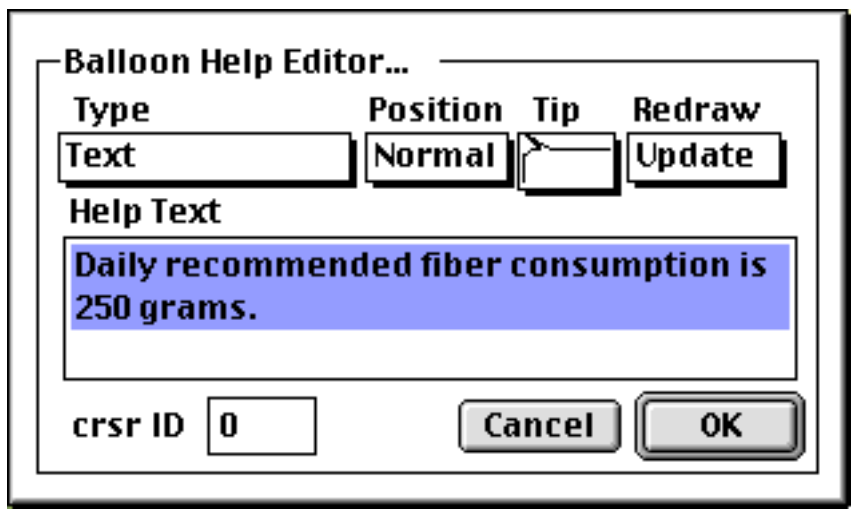


If you want to change the options later, go back to Graphics Mode. Select the object and choose **Convert/Edit Balloon Help** again to re-open the configuration dialog. This time the help text will appear, and you can edit it.



Balloon Help Options

The Balloon Help configuration dialog has four pop-up menus that allow you to specify various balloon help options. To open this dialog for a balloon help object that has already been created select the object and then choose **Convert/Edit Balloon Help** from the Setup menu. (Note: Unlike virtually every other type of Panorama object, you cannot open the configuration dialog by double clicking on the object with the **Pointer** tool selected. You must use the **Convert/Edit Balloon Help** command.)



The **Type** pop-up menu allows you to control whether a text or picture appears inside the balloon. If you select Text (the default option), the balloon will display the text in the Help Text section of the dialog. If you choose any of the resource options, the dialog will change to allow you to enter a resource ID number. The Picture Resource option will display a picture with the specified resource ID number. The String Resource option will display text from a STR resource. The String# Resource option will display text from a STR# resource. The TEXT styl Resource option will display styled text (which may include bold, italic, etc.) from a styl resource. These resources must be created with ResEdit or your favorite resource editor.

The **Position** pop-up menu controls where the tip of the balloon will appear. The Normal option makes the tip appear at the current mouse location. The Center option makes the tip appear in the center of the balloon help object.

The **Tip** pop-up menu controls the default position of the balloon. If possible, the computer will place the tip at the corner of the balloon that you specify. However, if the mouse is near the edge of the screen this may be impossible.

The **Redraw** pop-up menu controls how the screen is redisplayed when the balloon disappears. The Update option will always work, but is usually slower. The BitCopy option works by saving the pixels under the balloon and then replacing the exact same pixels when the balloon disappears. This option is much faster, but won't work if the pixels underneath might change while the balloon is visible (for example if the balloon is over a clock). The Both option uses both methods simultaneously. I'm not sure why you would want to do this, but Apple made this option available to us so we made it available to you.

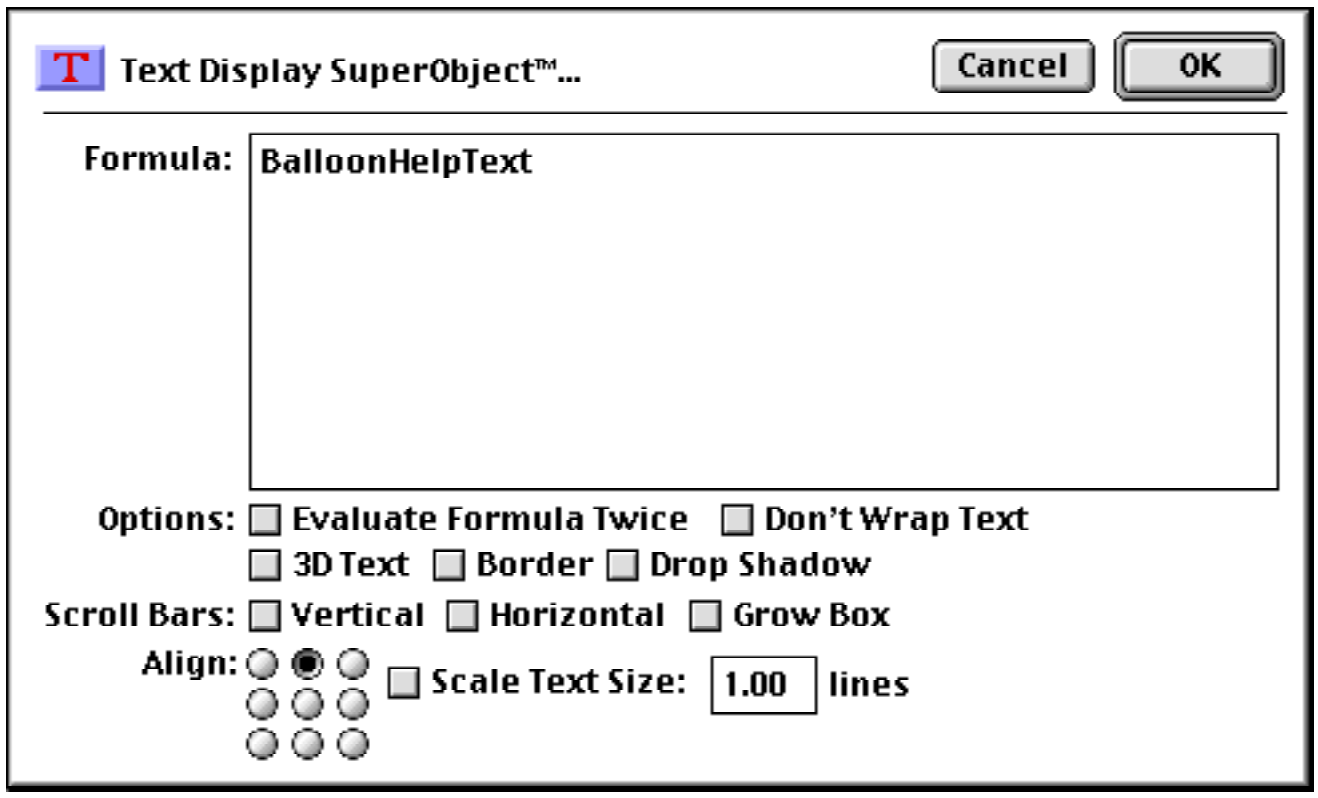
Changing the Cursor Shape Over Different Areas

Many applications change the shape of the cursor as the mouse moves over different areas of a window. For example, it might change to a finger when over a button, to a crosshair when over a table, and to a pointer everywhere else. You can make the cursor change as the mouse moves over your Panorama forms by using Balloon Help objects. The Balloon Help dialog allows you to enter the resource ID of a cursor in the Cursor ID option. The cursor must be stored in a resource file that has been opened with the `openresource` statement.

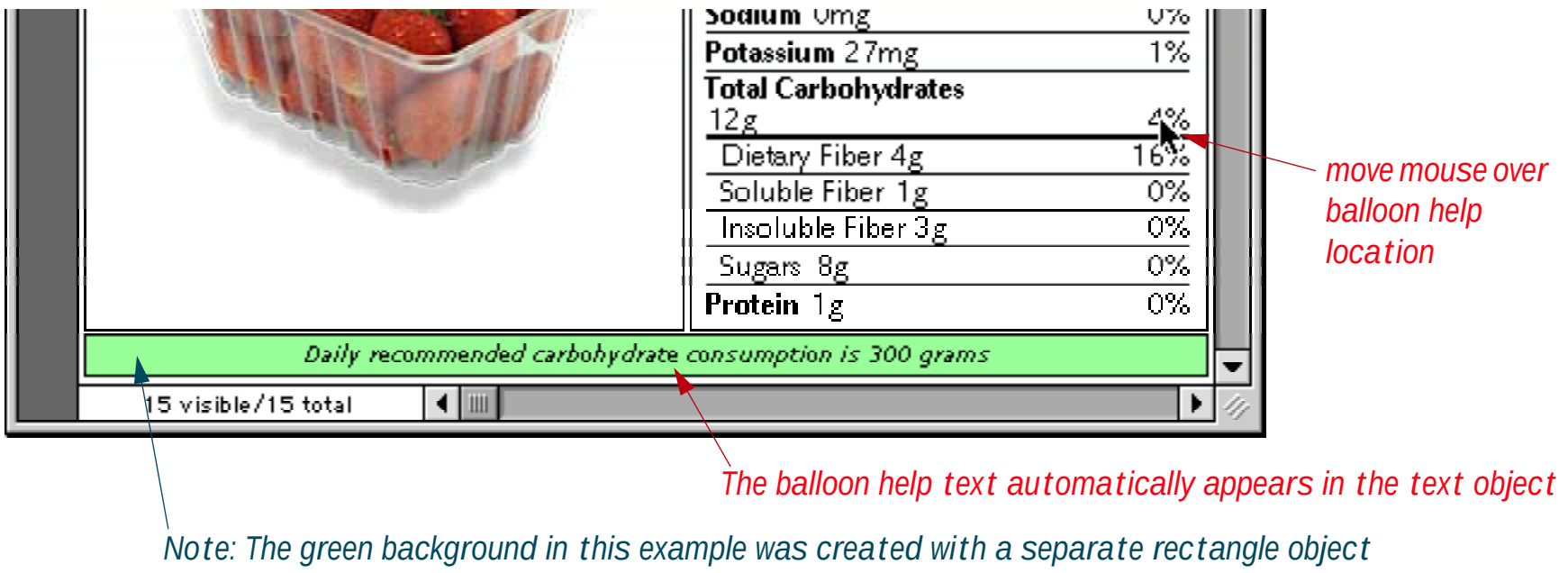
Displaying Balloon Help Text Directly on the Form

Balloon Help text can be displayed on the form itself using a special variable. When used this way the text is visible even if the **Show Balloons** option is not turned on. This option also allows the balloon help text to be displayed on PC systems (kind of like tool tips). Note: This option was added in the Panorama 4.0.2 release, and will not work with earlier versions.

To display balloon help text directly on a form first set up balloon help objects on various locations on the form (as described earlier in this chapter, see "Creating Balloon Help Objects" on page 975). Once the balloon help objects are set up you'll need to set up a Text Display Objector Auto-Wrap Text Object that displays the variable **BalloonHelpText** (see "Using Formulas to Display Text" on page 621). You don't need to create this variable yourself, Panorama will do it for you. Here is an example of how to set up a Text Display SuperObject to display the balloon help text.



There's nothing more to it. To see the help text simply move the mouse over a location where you have set up balloon help.



The help text will automatically update as you move the mouse over different locations in the form. The help text will appear whether or not the **Show Balloons** option is turned on, and will also appear when viewing the form on a Windows PC system.

Chapter 19: Charts



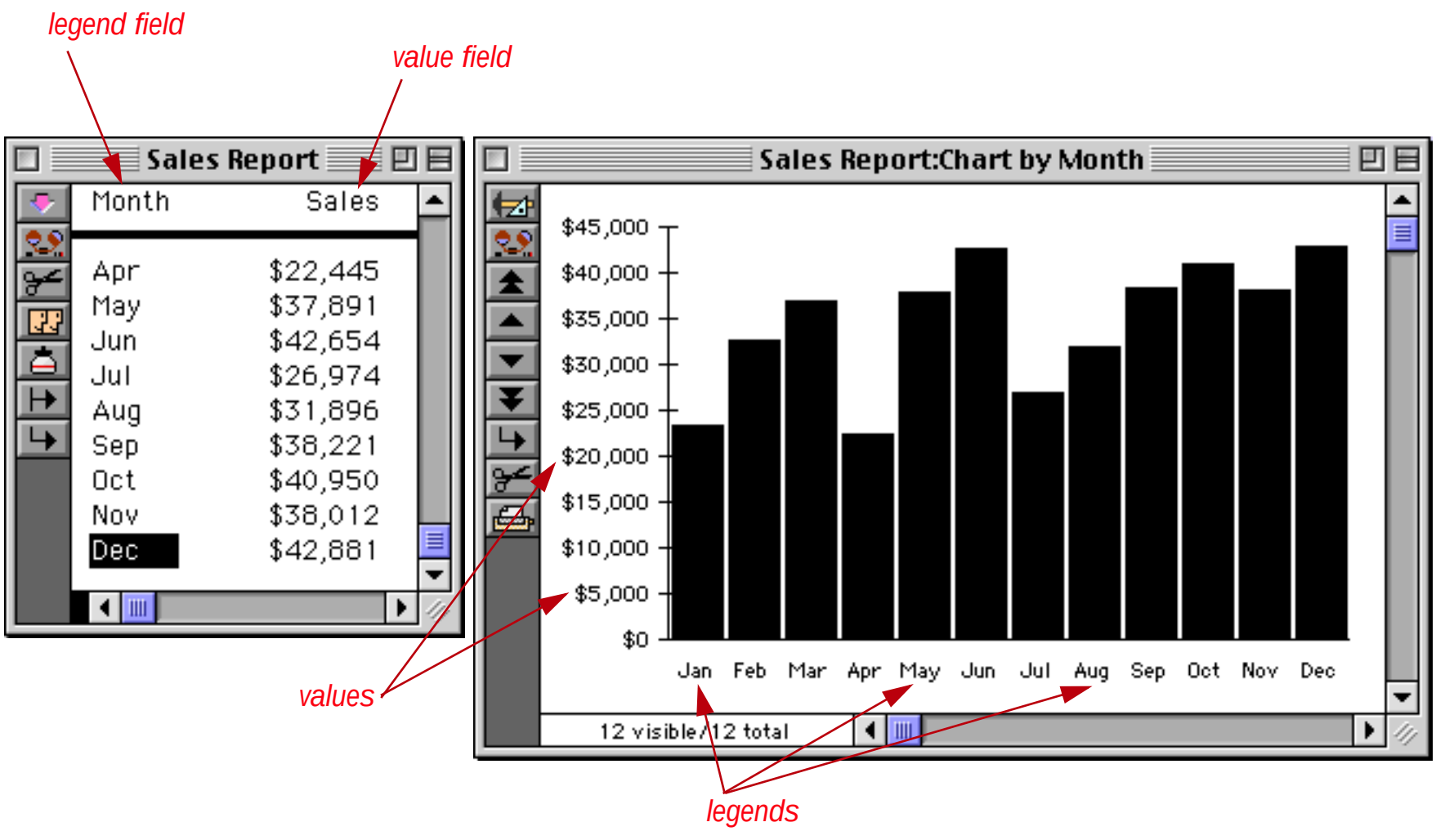
Many databases are filled with numbers. In this chapter you'll learn how to bring those numbers to life by turning them into charts and graphs. Panorama can draw five different kinds of charts—Bar, Line, Area, Pie, and Scatter.

Drawing a chart involves two steps—setting up the chart itself and preparing the database. You only have to set up the chart once, but you usually have to prepare the database each time you want to draw the chart. Fortunately, preparing the data is easy, and the entire process can be automated with a procedure to make it a real "no brainer."

Chart Data

The job of a chart is to display numeric information graphically. Before Panorama can draw a chart it needs to know what numbers you want to draw, and what those numbers represent.

The illustration below shows a simple data sheet along with a form containing a bar chart. Each bar corresponds to a record in the database. The database has 12 visible records, so this chart has 12 bars.



Of course, most databases have hundreds or thousands or records—far too many records to chart directly like this. Before you can draw a chart of a database with hundreds or thousands of records, you must create a summary of the database. The chart will display the summary instead of the entire database. See "Preparing the Database for Drawing a Chart" on page 996 later in this chapter.

Each bar has a legend that tells what the bar represents. The chart grabs the legend from a field in the database—the legend field. In the illustration above the legends come from the Month field. The legend is usually drawn across the X (horizontal) axis of the chart.

The height of each bar shows its value. The chart grabs the value from a field in the database—the value field. Any numeric field can be used as a value field. In the illustration above the values come from the Sales field. The values are usually drawn along the Y (vertical) axis of the chart.

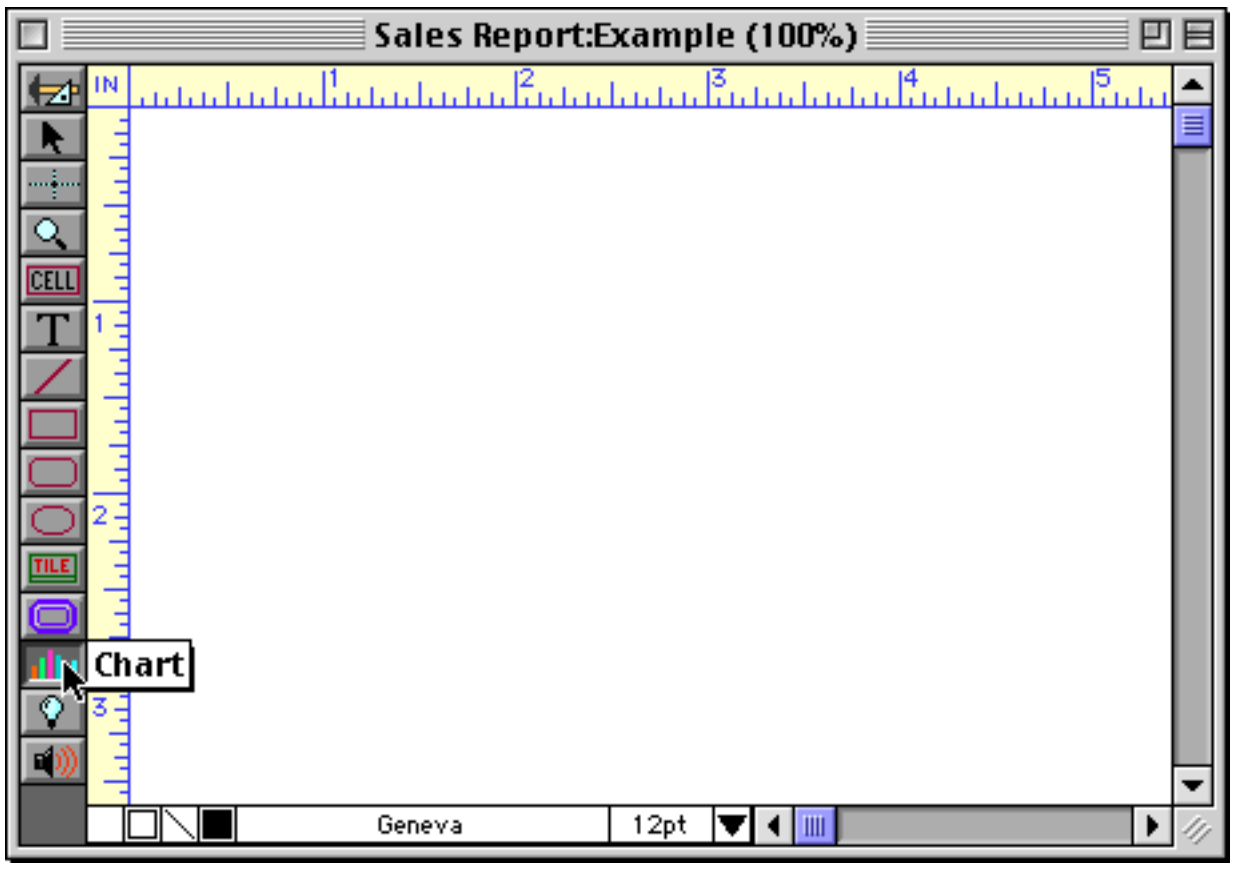
The chart doesn't just use any random field for the legend and value—you must specify these fields using pop-up menus. You'll learn how to do that later in this chapter.

Tip: Remember that each legend and value comes from a separate record. When you set up your database, make sure that the data to be charted is in separate records—not spread across a single record.

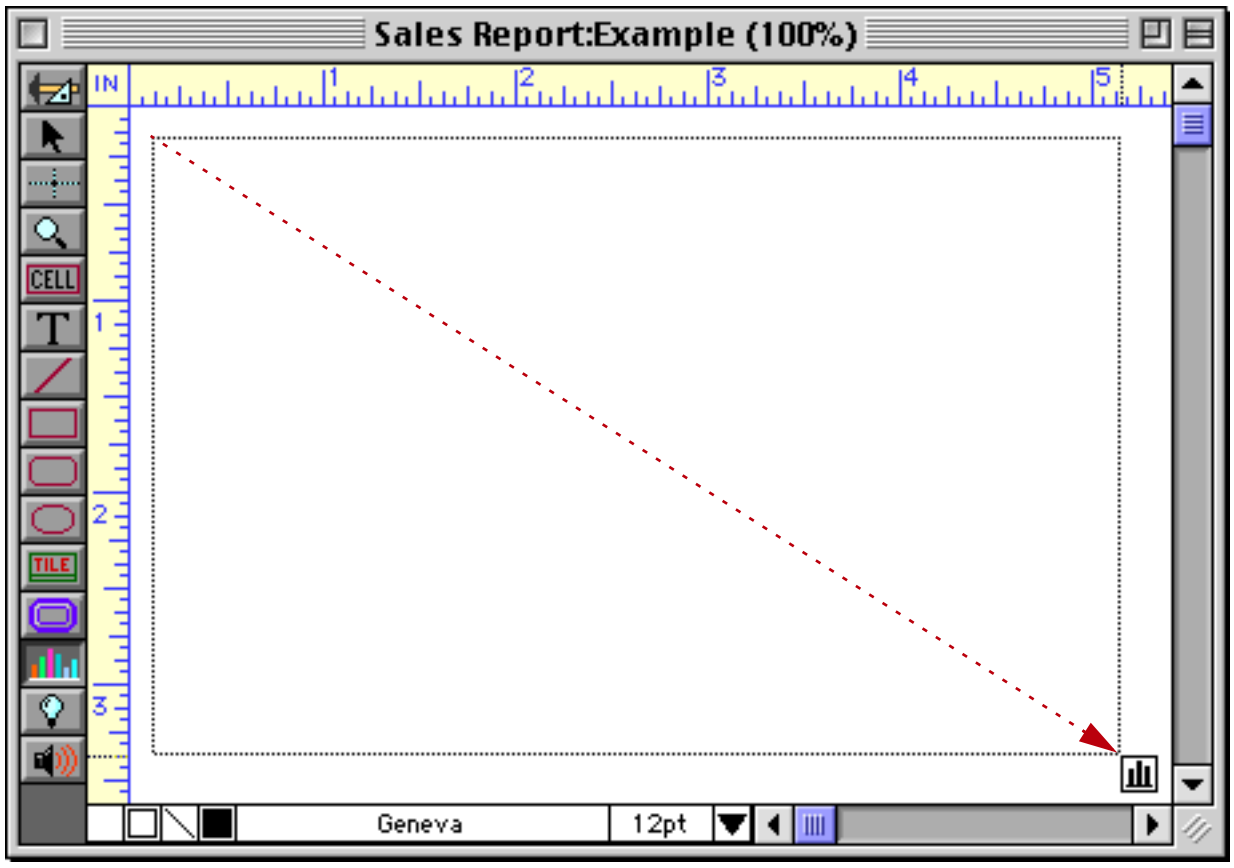
Creating a New Chart

Charts are graphic objects that are set up and manipulated with the same graphic editing tools you use to create forms and reports (see "Graphic Design" on page 491). Be sure the form is in graphic design mode before you try to create a chart.

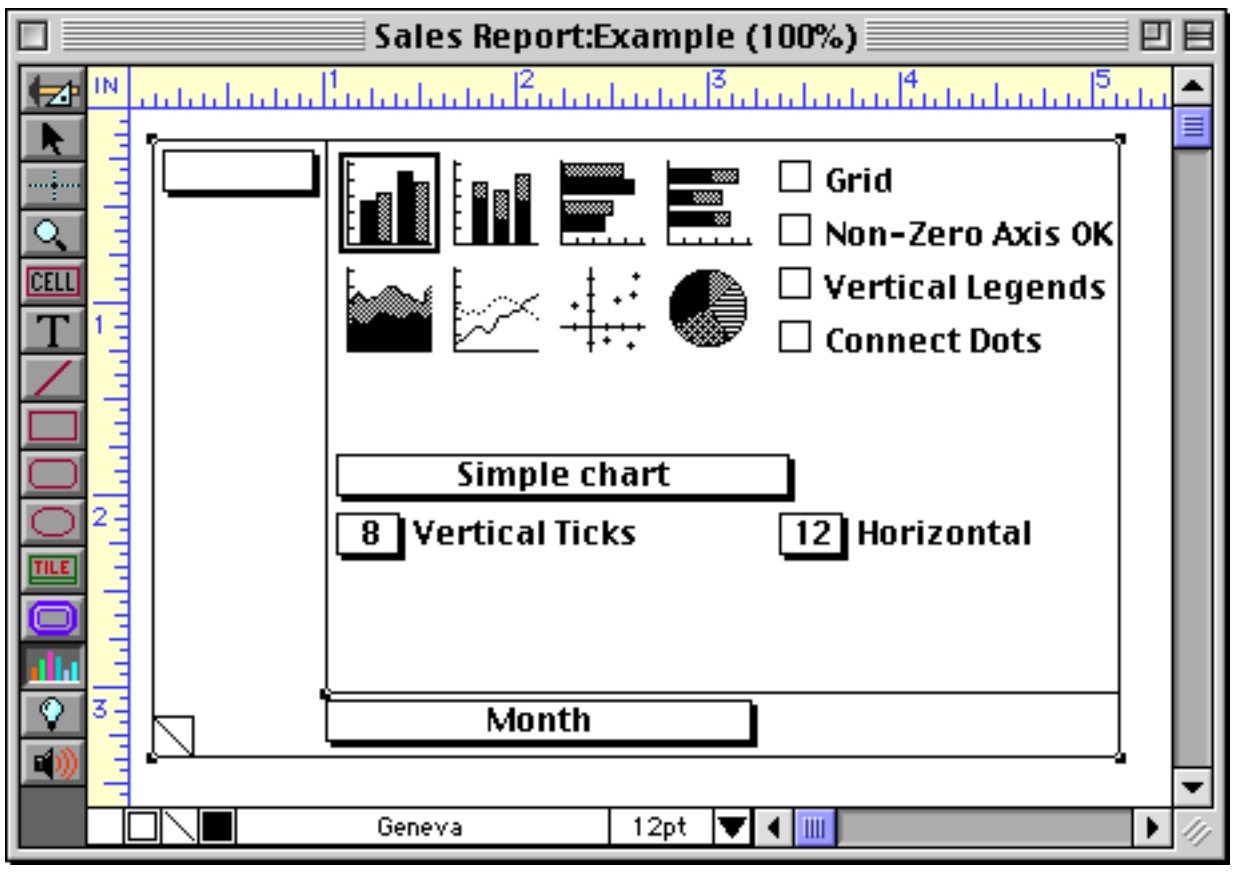
To create a chart, start by selecting the **Chart** tool.



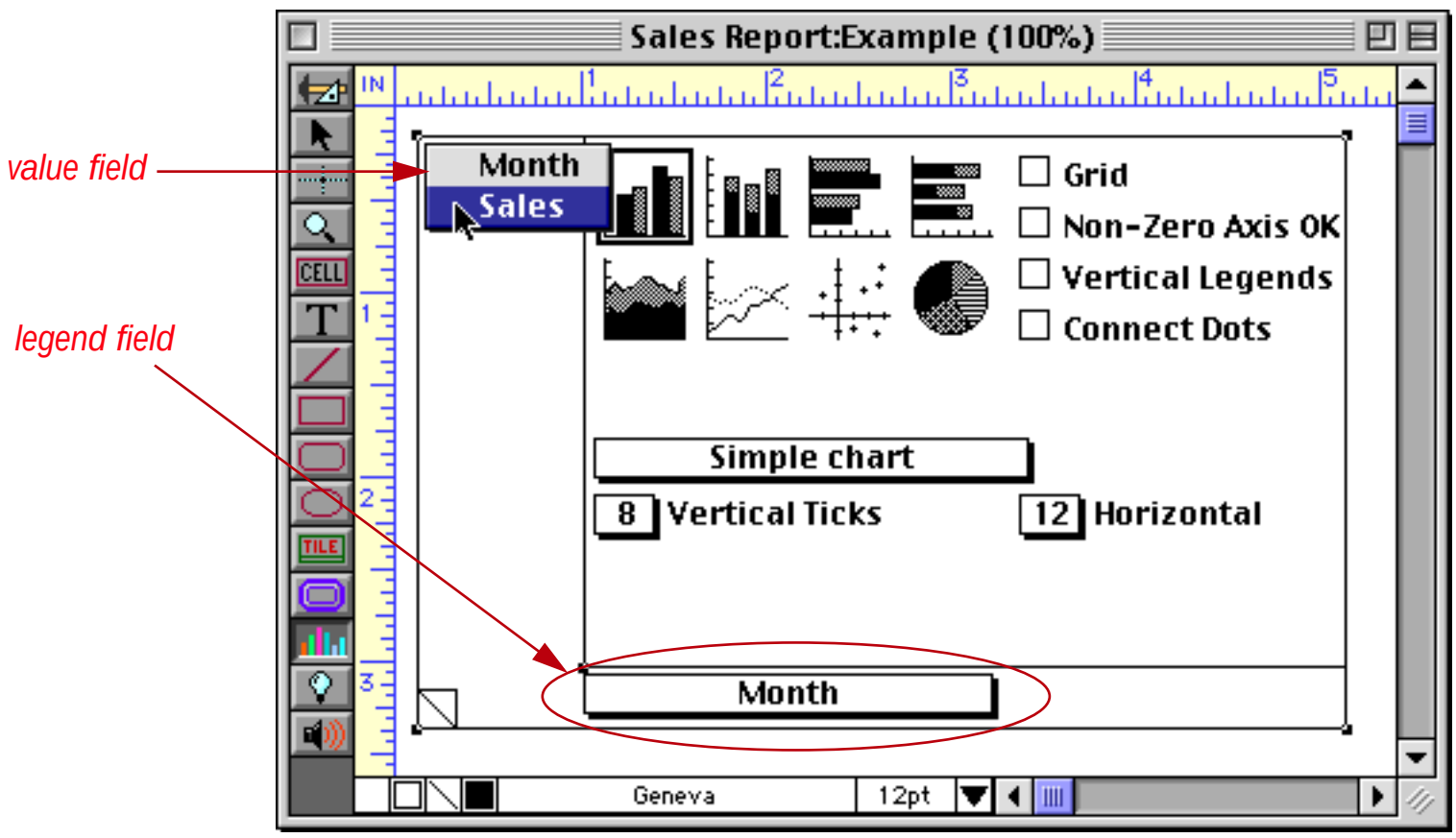
Next, drag the mouse across the form to define the corners of the chart.



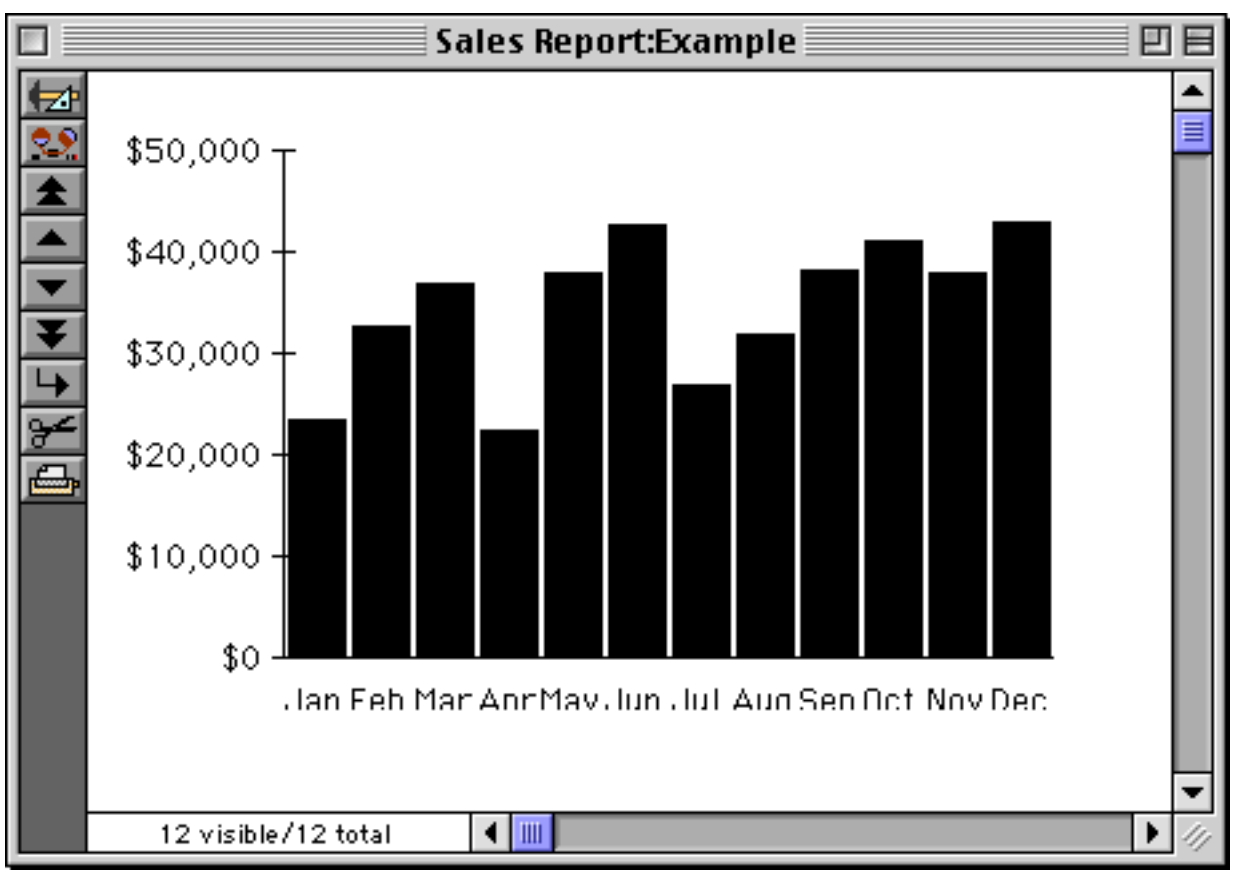
The new chart doesn't look much like a chart. In fact, it looks more like a dialog!



The buttons and pop-up menus in the dialog allow you to configure the chart. At a minimum, you must select a legend field and at least one value field. In this case the legend field is already set to Month. Use the pop-up menu to set the value field to Sales.

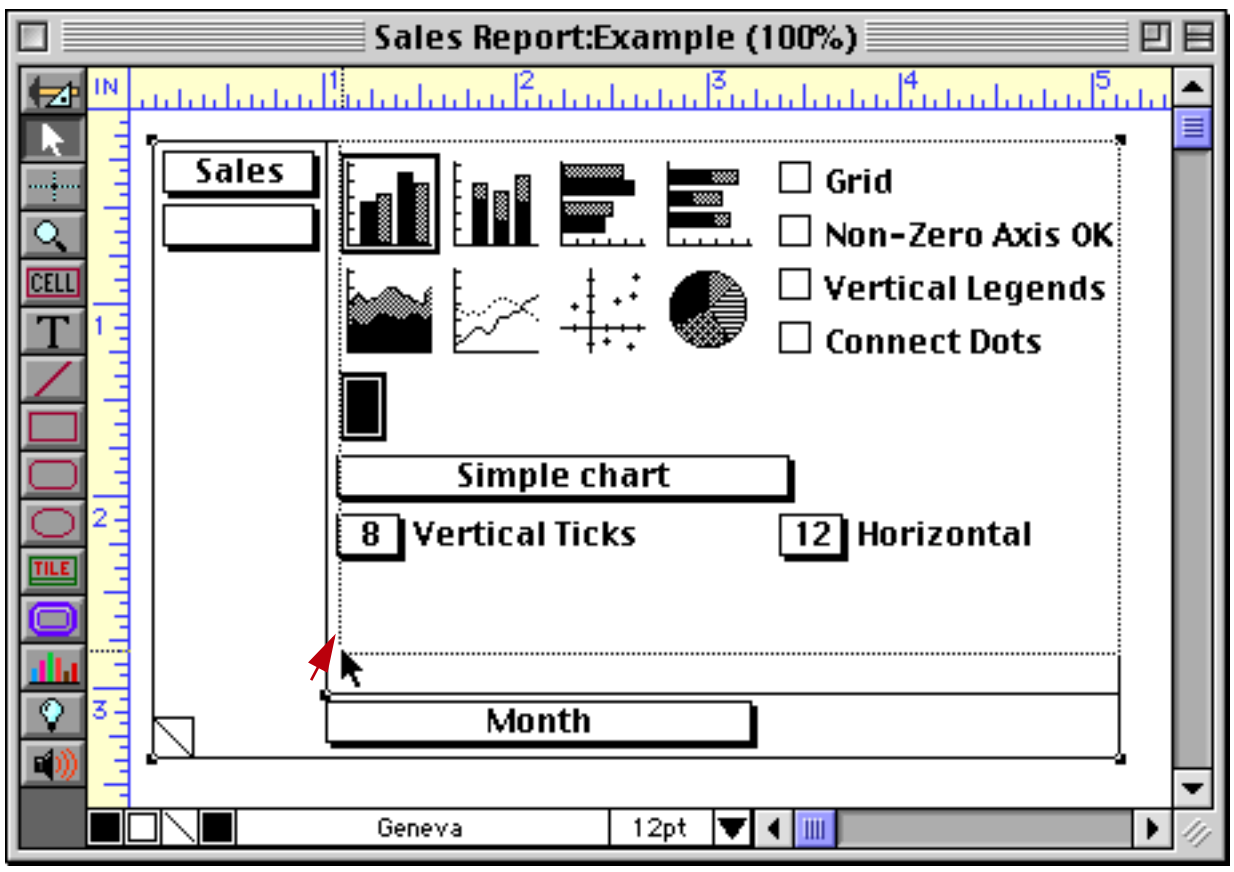


When you switch to Data Access Mode, the chart options disappear and Panorama draws the actual chart based on the information in the legend and value fields.

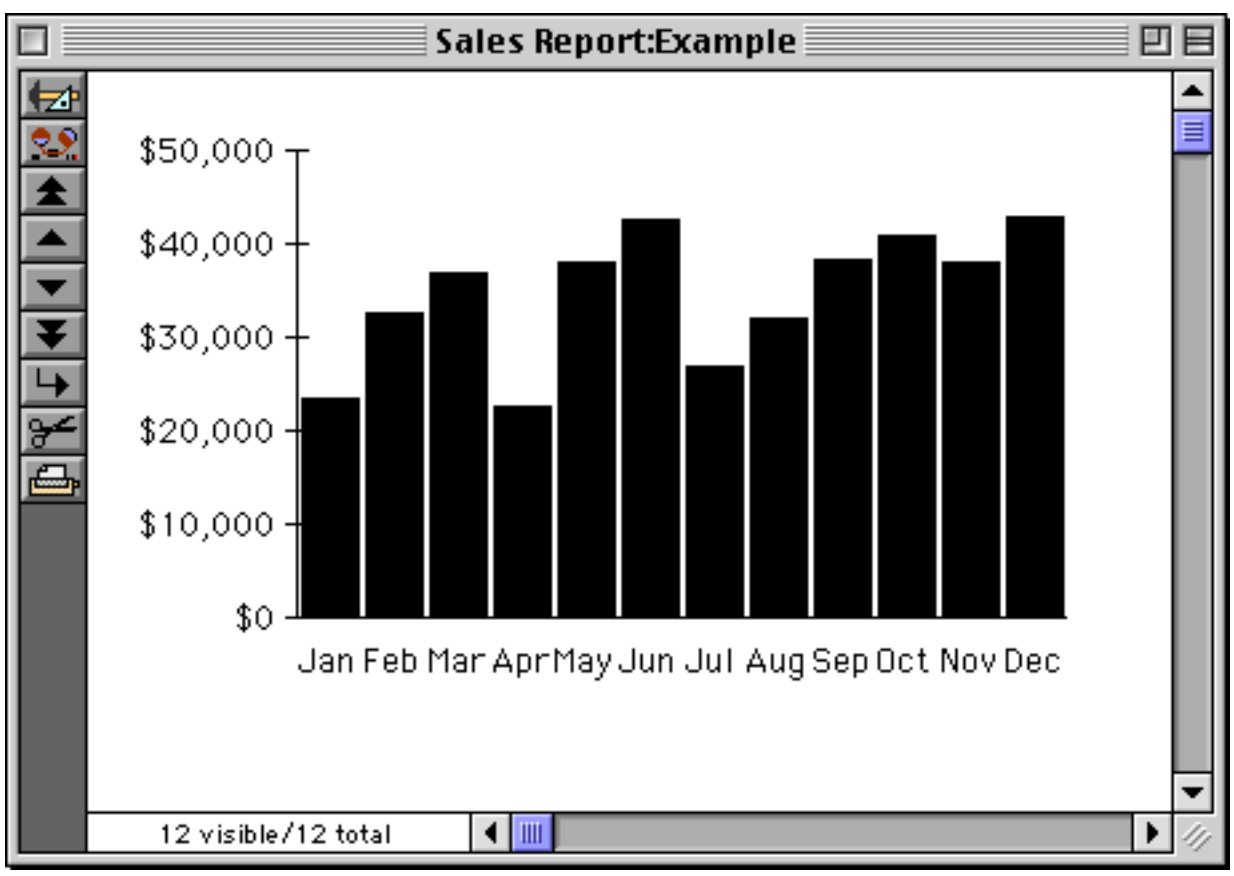


You can move the chart by dragging it, or change the size by dragging one of the handles on the corners (just like any other graphic object). When you drag the chart, be sure you press on one of the empty areas within the chart, not on a button.

In addition to the four handles on the corners, charts have an extra handle at the intersection of the X and Y axes. You can drag this handle to change the position of the X and Y axis within the chart. In this illustration the extra handle is being dragged upward to leave more room for the legends.

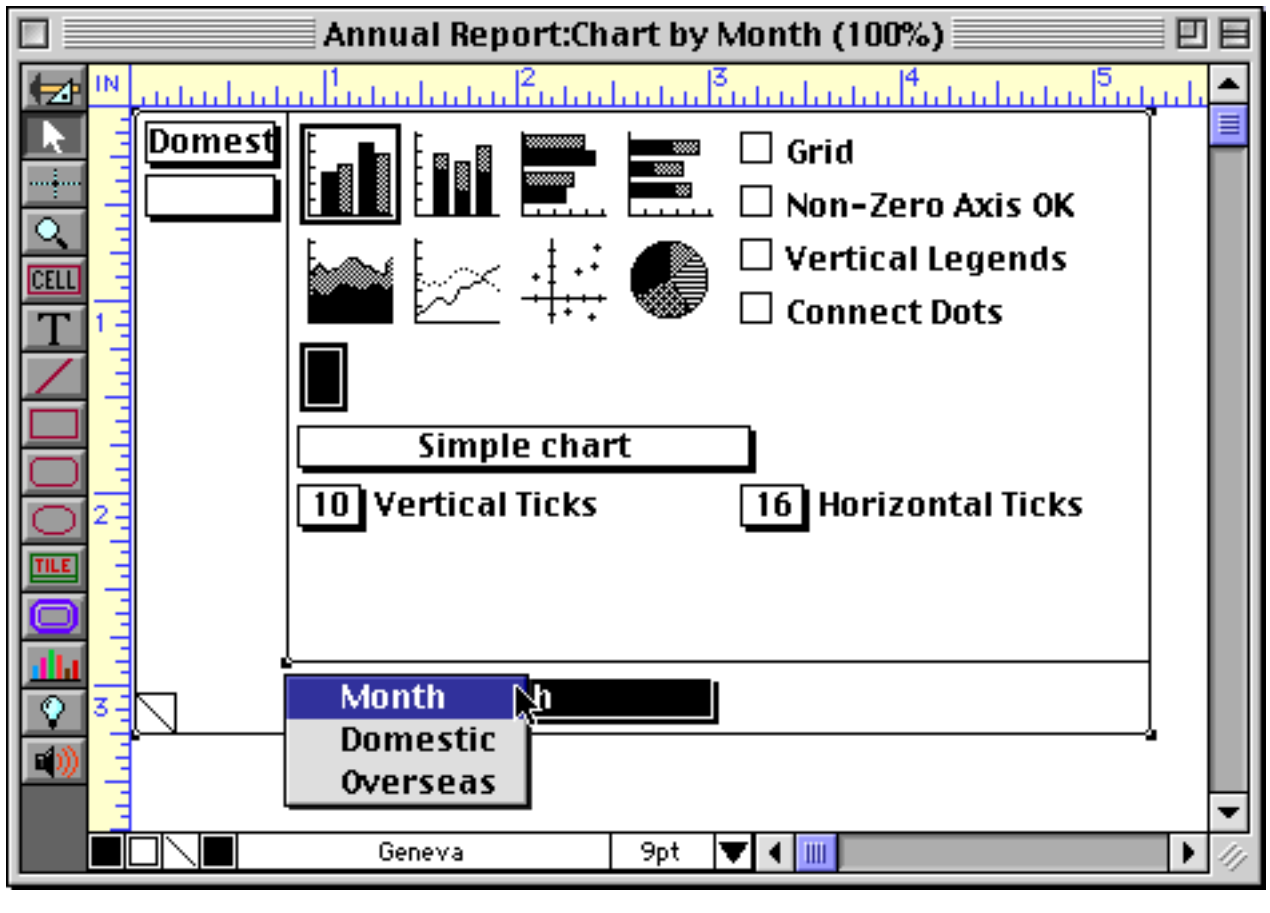


Switching back to Data Access Mode shows that there is now enough room to display the legends without cropping.



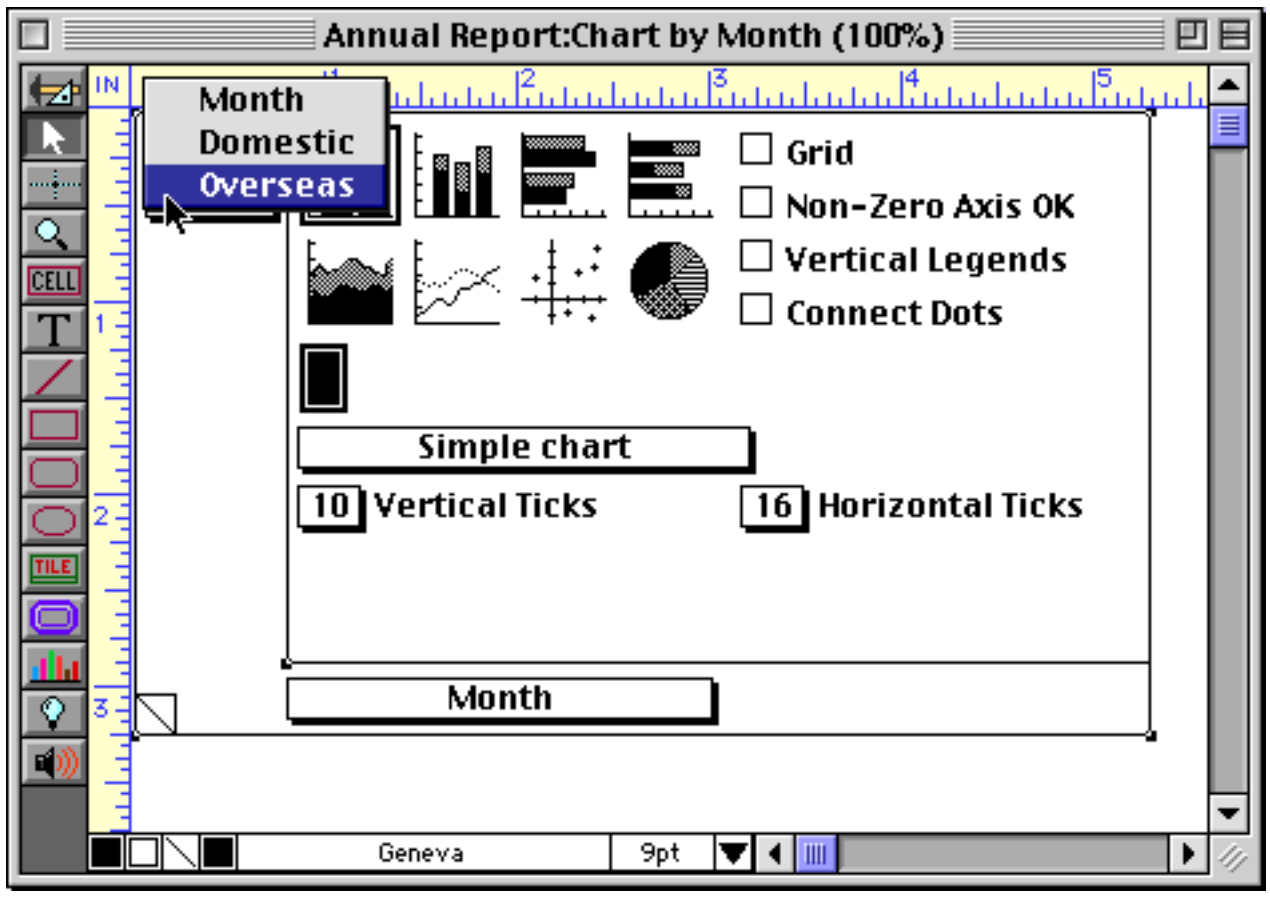
Setting Up Legend and Value Fields

To set the legend field, use the pop-up menu at the bottom of the chart object.



If the chart is being designed to display summary records, the legend field is usually the field that was used to group the database (see "STEP 1 - GROUP" on page 399). Each chart has one (and only one) legend field.

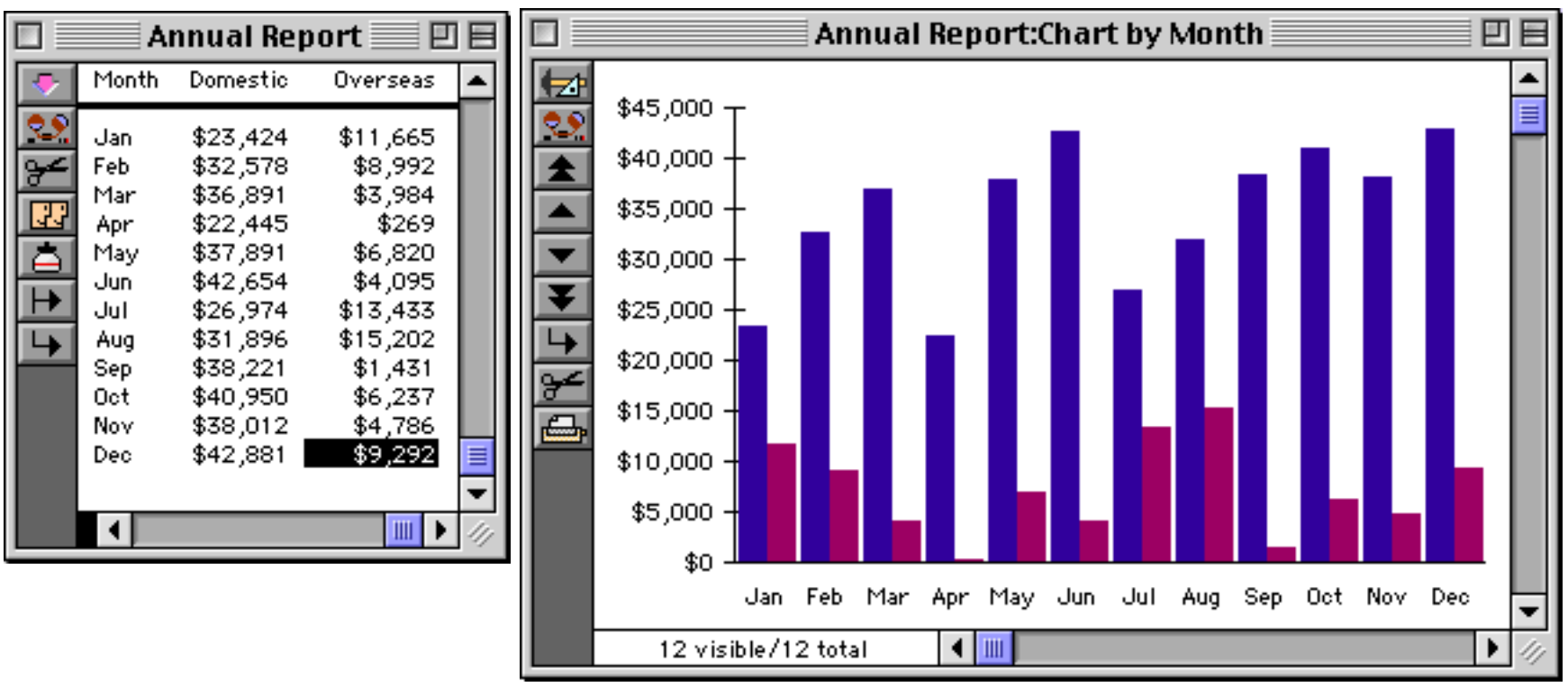
To set the value field, use the pop-up menu in the upper left corner of the chart object.



Keep in mind that the value field must contain numeric data (see "Numeric Data" on page 289).

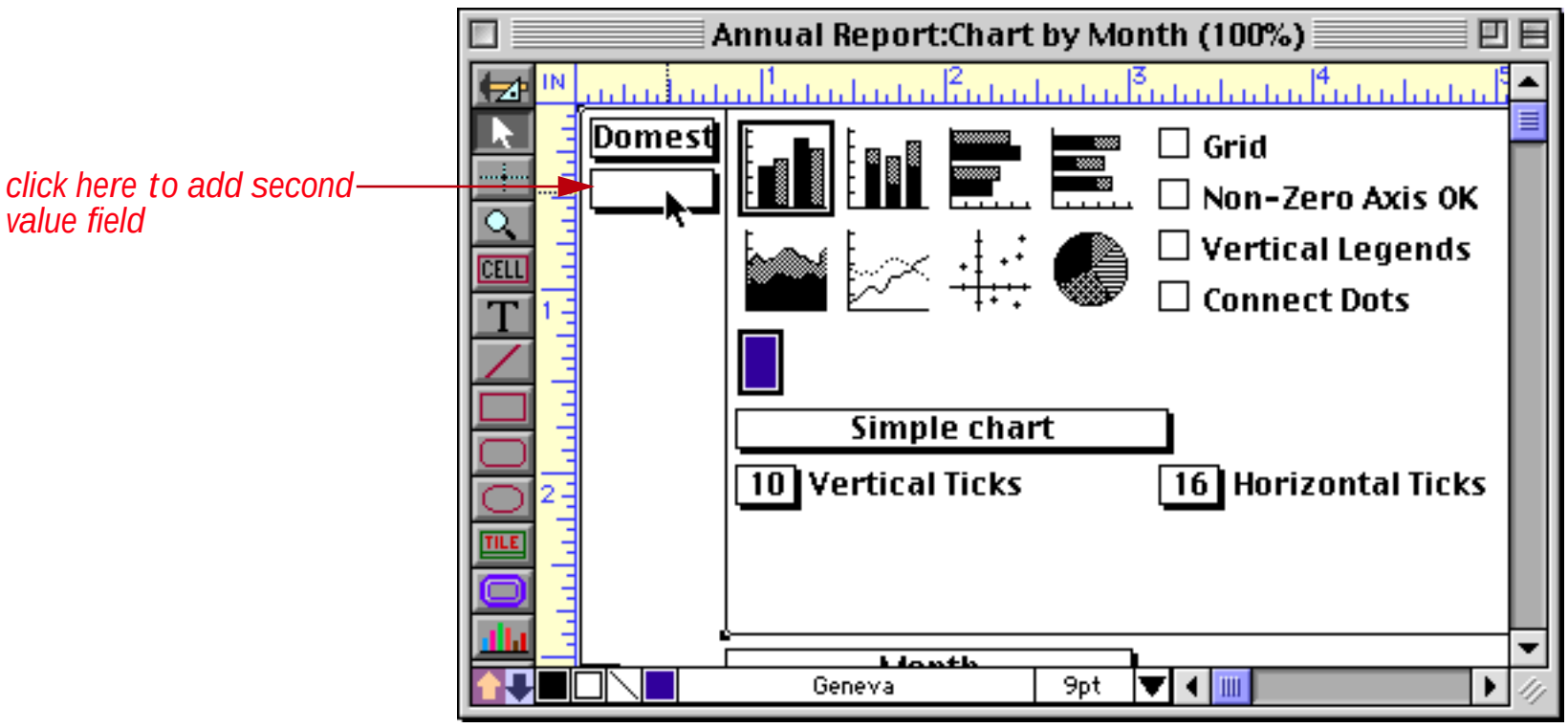
Setting Up Additional Value Fields

The simple bar chart shown on the previous pages displays one value per record. More complex charts can display several values per record by adding additional bars or lines.



A chart can display up to 10 values per record. To differentiate between the values, each value field can be given its own color and pattern.

After you specify the first value field, a second pop-up menu appears below the first value field. To add another value field, use this pop-up menu.



You can add up to 10 value fields to a chart.

To remove a value field, choose **Remove Field** from the value field pop-up menu. However, you cannot remove the first value field.

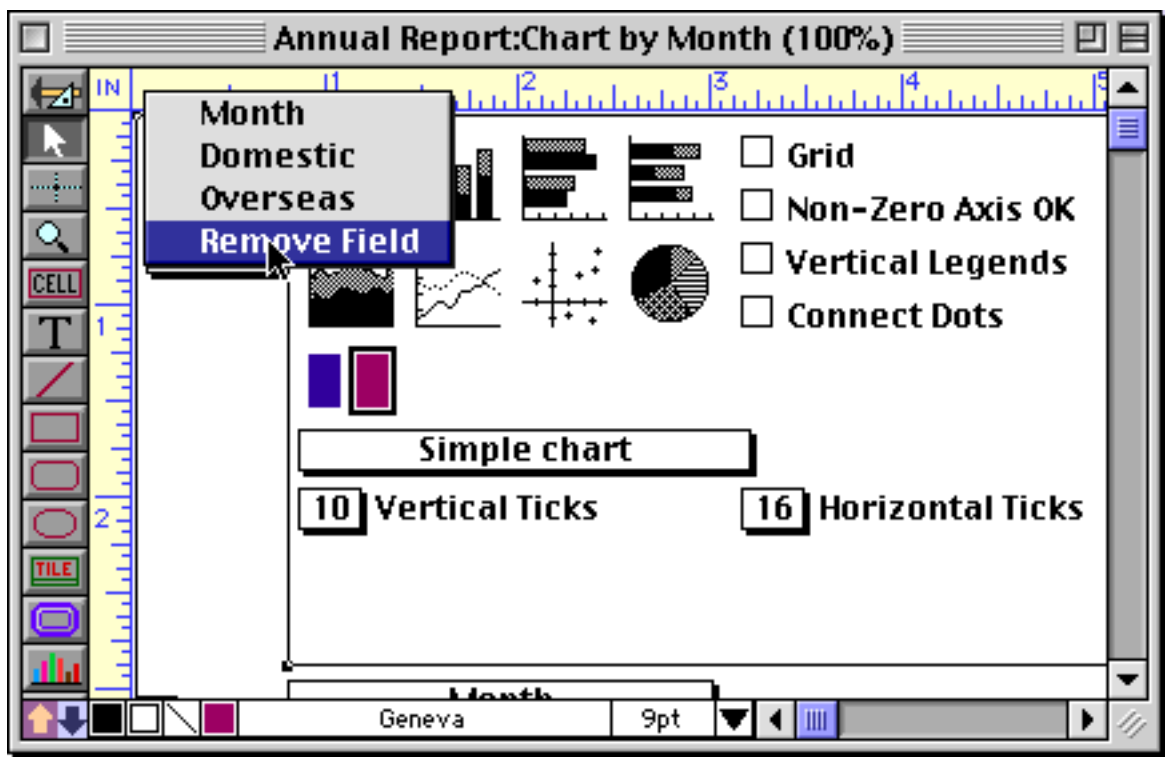
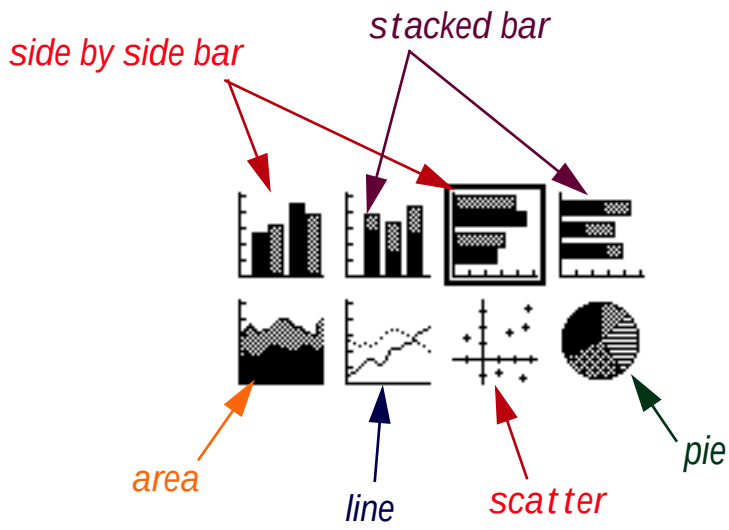


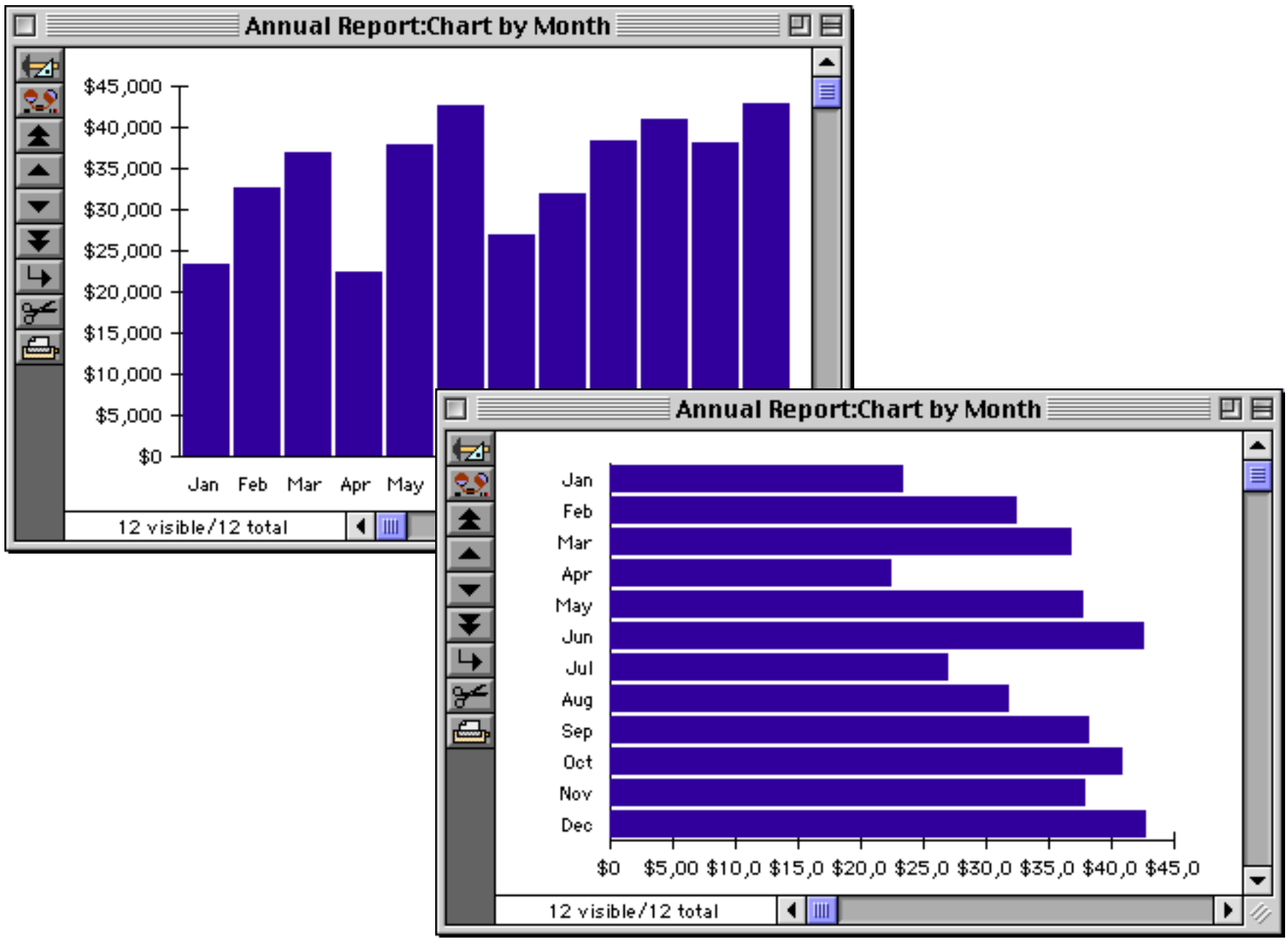
Chart Types

The chart dialog has eight icons representing the kinds of charts Panorama can draw, including bar charts, line charts, area charts, pie charts, and scatter diagrams. The following section contains a brief description of each type of chart. Select the chart type by clicking on the appropriate icon. A box will appear around your selection.



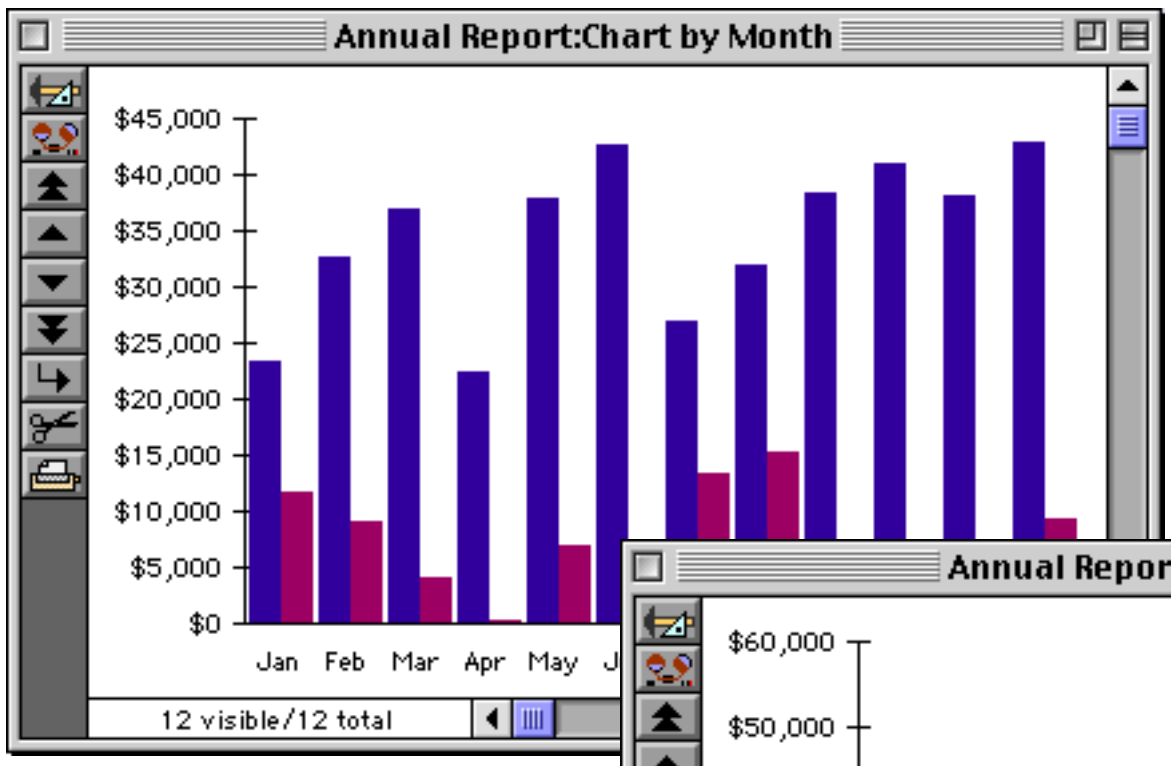
Bar Charts

Bar charts are probably the most common type of chart. Panorama can draw bar charts with either vertical or horizontal bars. Horizontal bars are especially useful when you have lots of data or long legends.

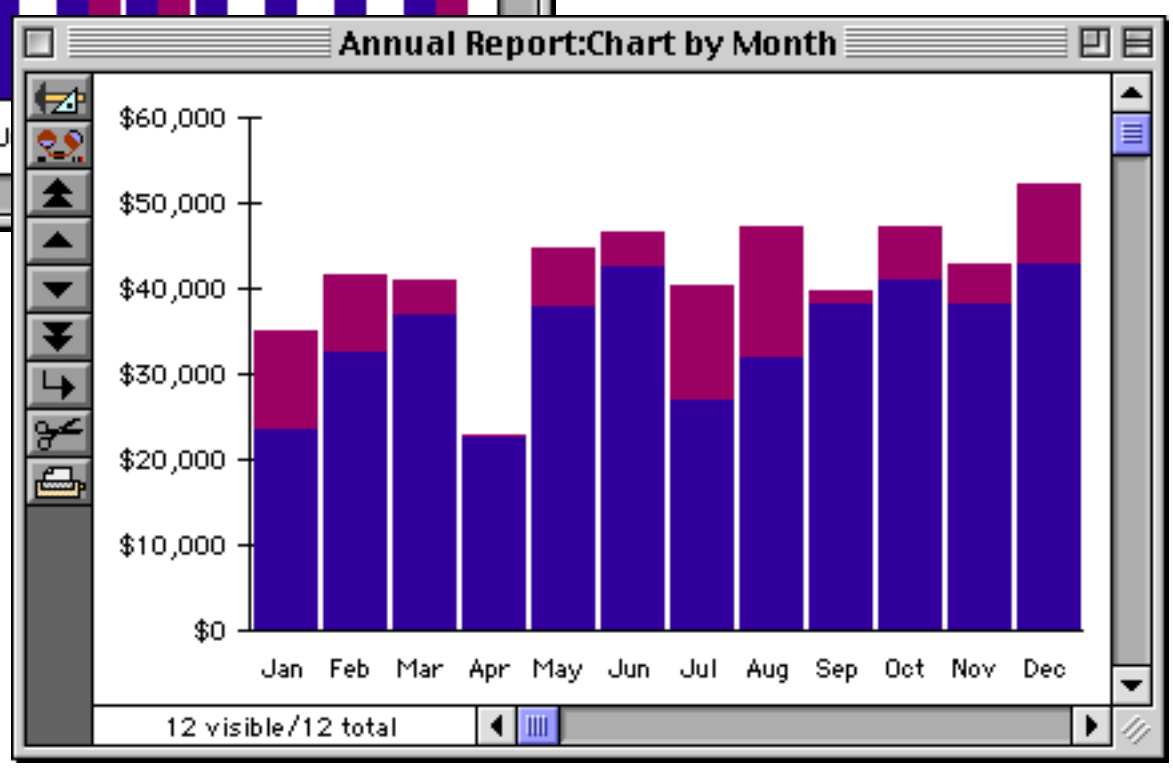


If the chart has multiple value fields, the bars can appear either side-by-side or stacked. If the values are side-by-side, they are displayed from left to right. If the values are stacked, they are displayed from bottom to top.

side by side

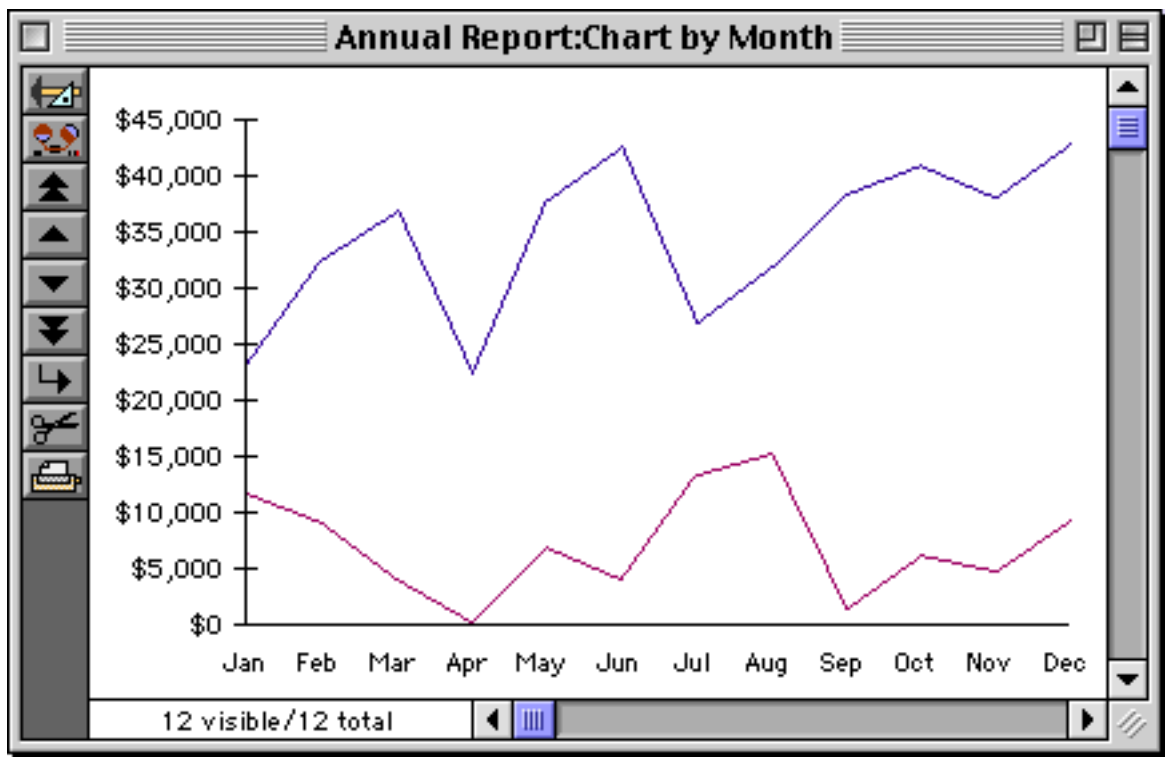


stacked



Line Charts

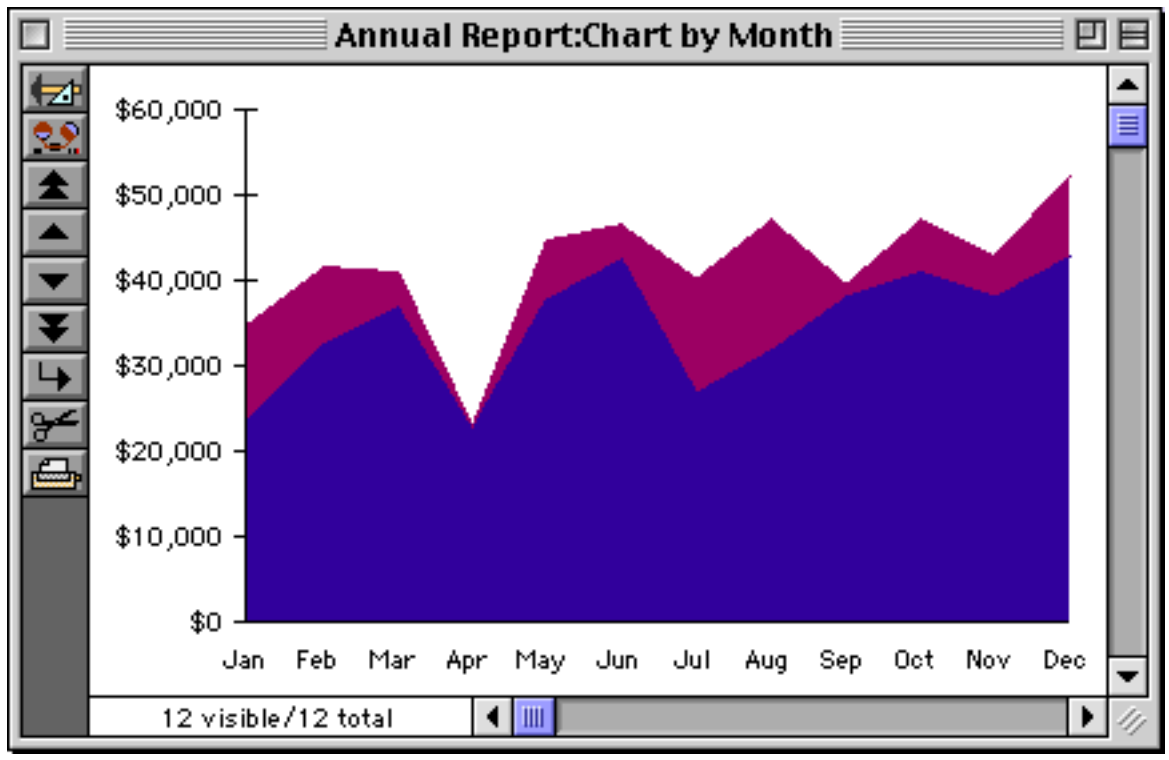
Line charts are also popular, especially when large amounts of data need to be displayed.



See “Dressing Up Chart Appearance” on page 1004 to learn how to control the pattern, width, and color of each line.

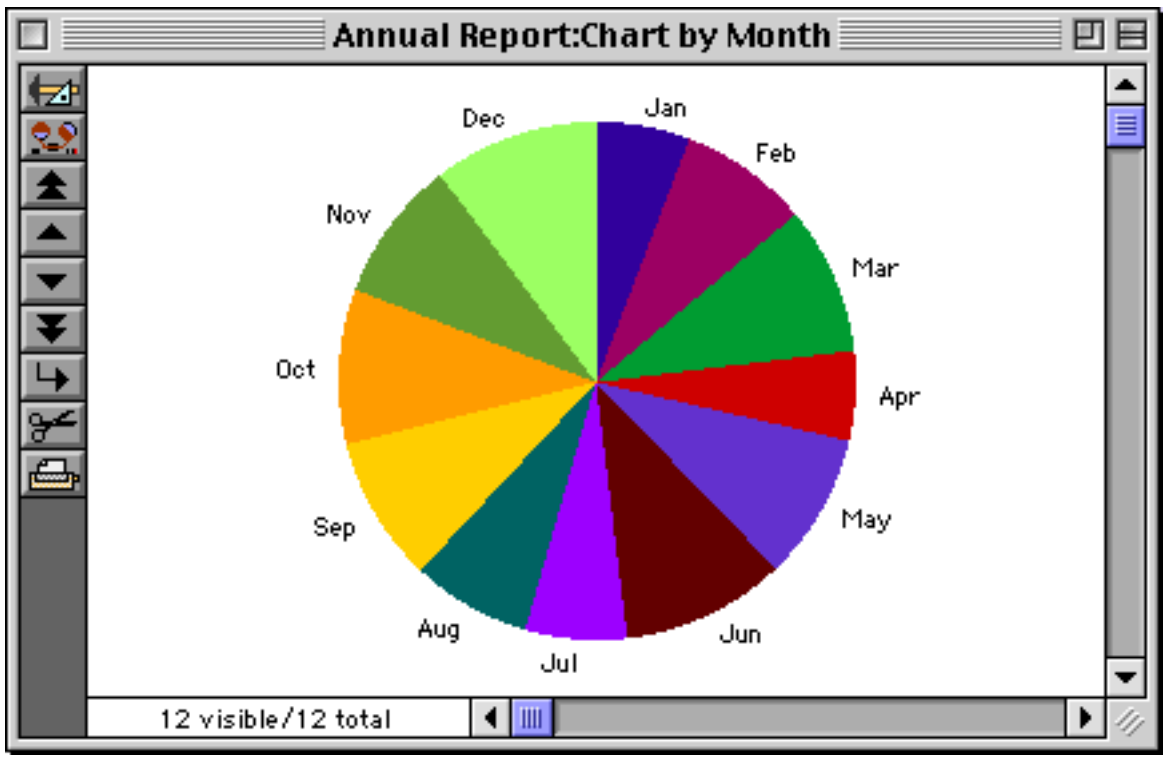
Area Charts

Area charts are very similar to line charts. The areas for each value are stacked on top of one another.

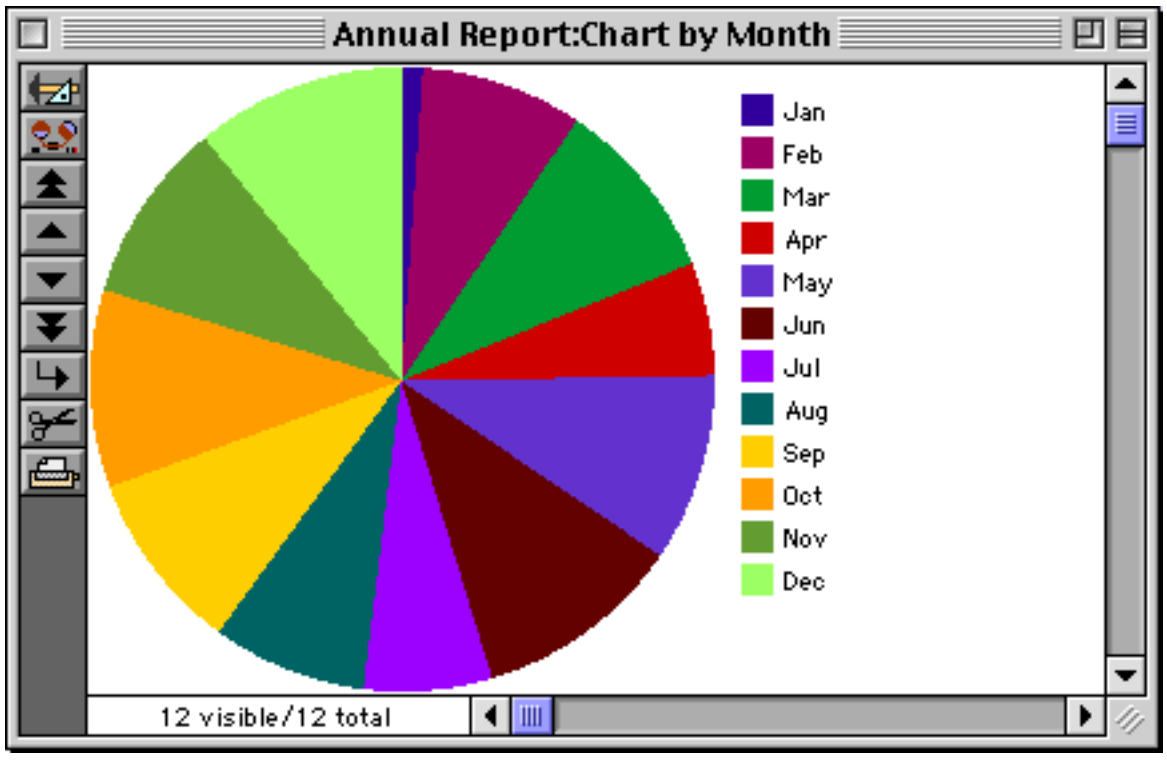


Pie Charts

Pie charts display the data as slices of a pie. The size of each slice is proportional to size of each value in relation to the whole. If possible, Panorama will draw the legends around the pie.



If some of the slices are too thin, Panorama will draw a separate legend, either to the right or below the pie. The exact location of the legend depends on the shape of the chart object.

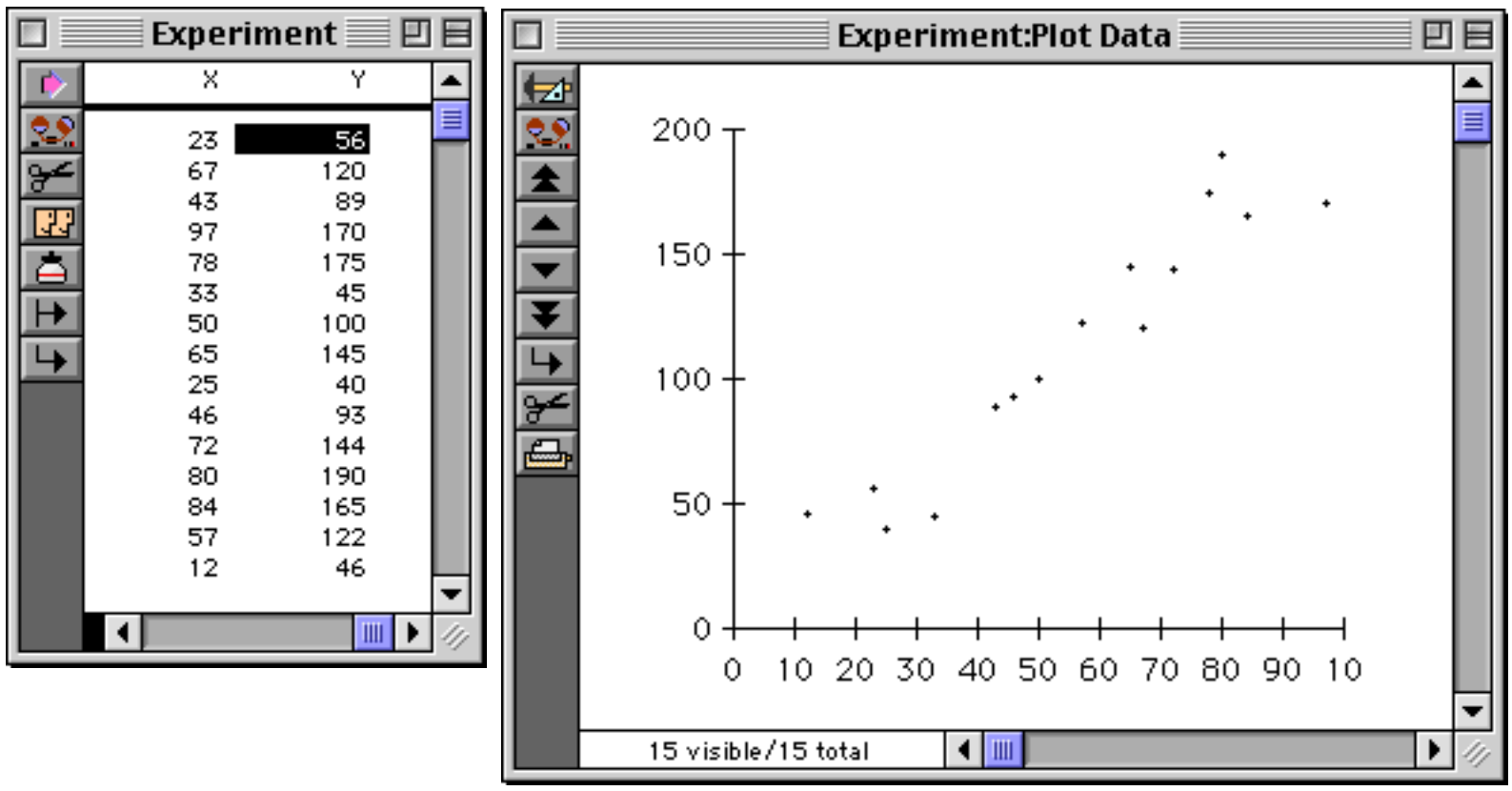


Pie charts should only be used for showing the relationships between a few values. Most graphic designers suggest that pie charts with more than seven slices should be avoided. Although Panorama can draw a pie chart with dozens of slices, a bar or line chart will probably convey the message more clearly.

A pie chart can only display a single value field. Any additional value fields will be removed from the chart when the pie chart icon is selected.

Scatter Diagrams

Scatter diagrams plot two numeric fields against each other. Each record contains a pair of numbers (Legend,Value) that Panorama uses as Cartesian (x,y) co-ordinates.

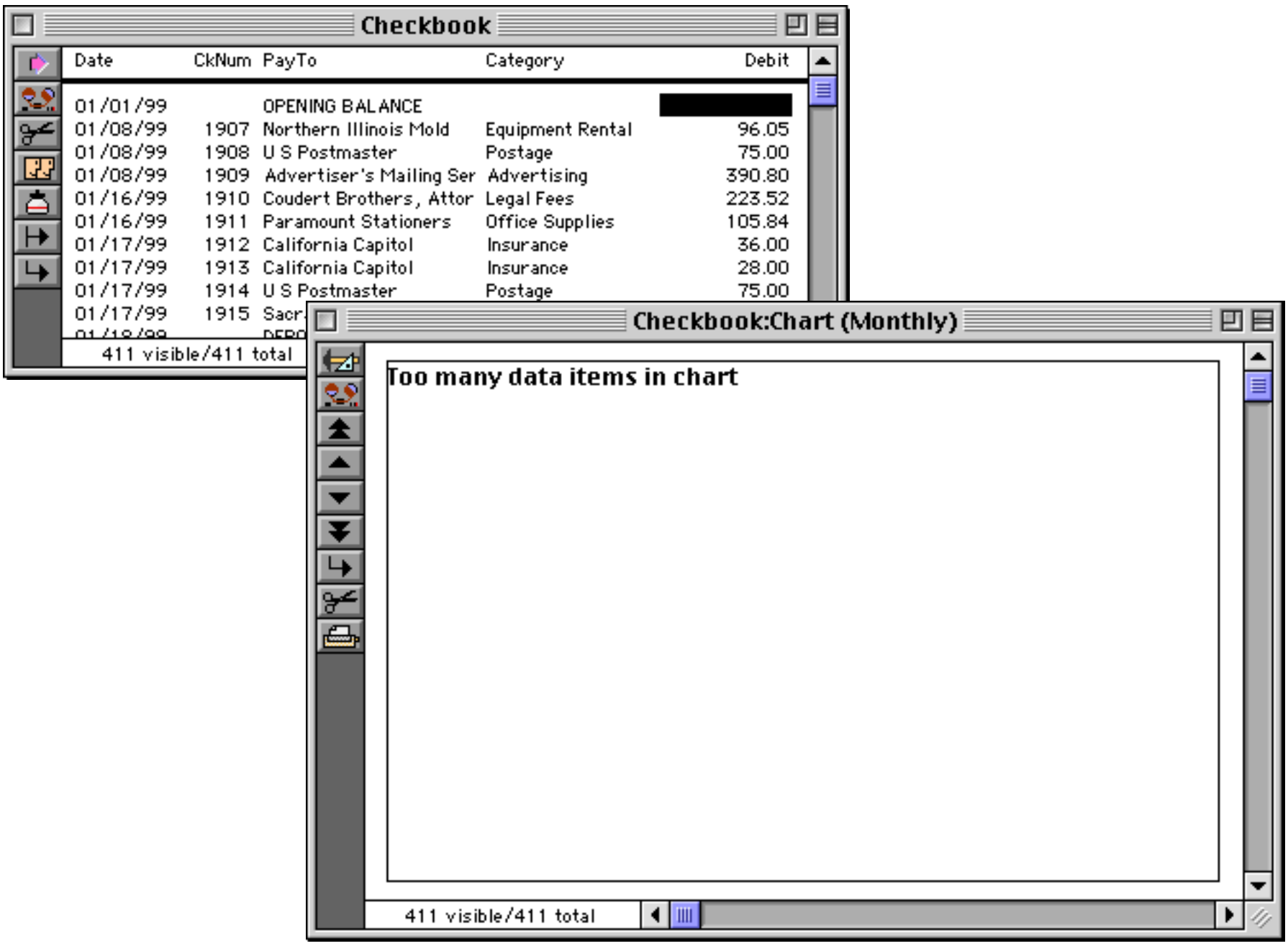


Each pair of values is normally plotted as a separate dot, but you can also connect the dots or change the dots into symbols or pictures (see “Scatter Diagram Flash Art” on page 1028 and “Connect Dots” on page 1032).

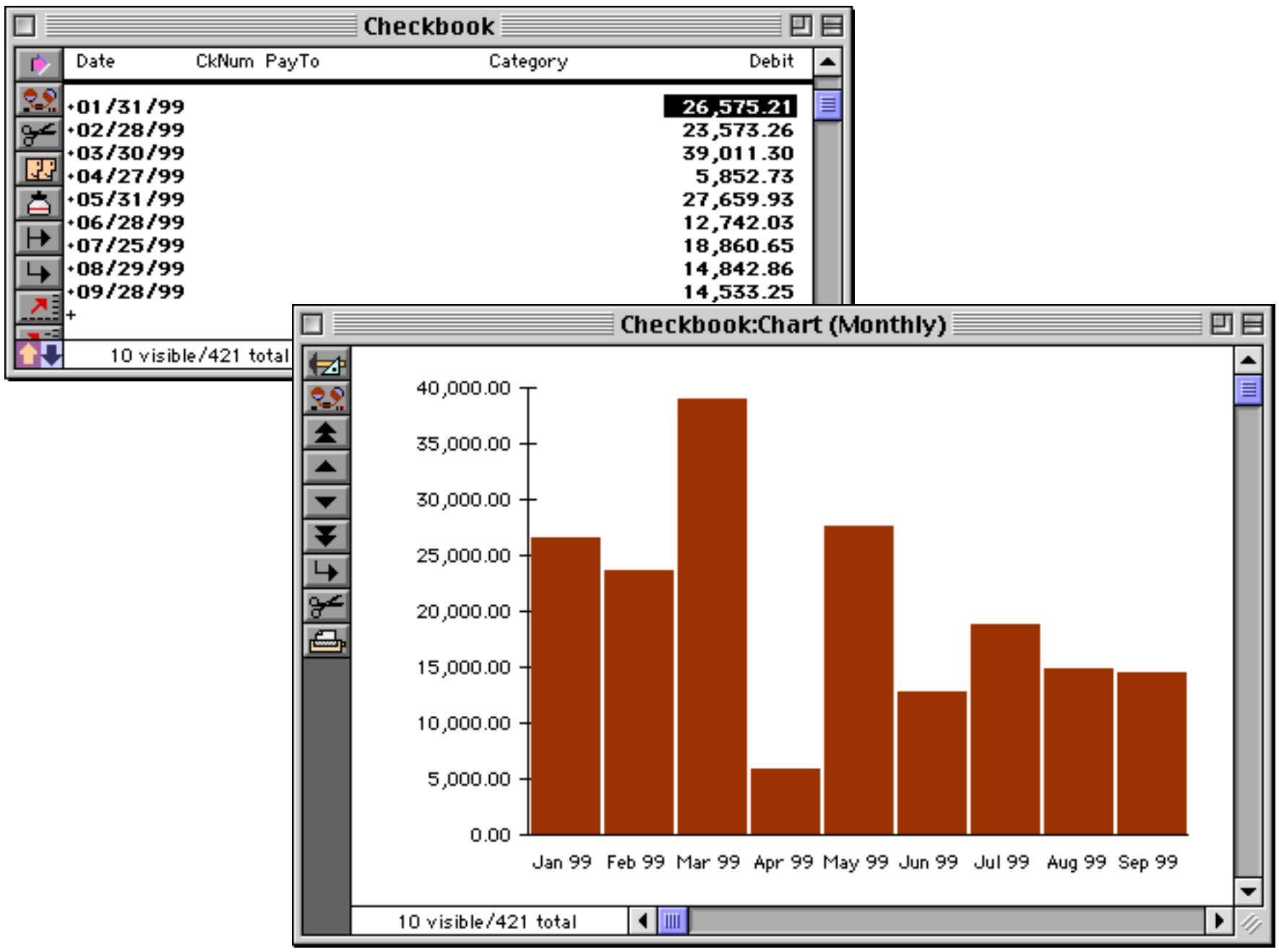
Unlike other charts, a scatter diagram requires that both the legend and the value fields contain numeric data.

Preparing the Database for Drawing a Chart

Most databases contain far too much raw data to chart directly, record for record. For example, this checkbook database has several hundred records. Panorama can't draw a chart with 411 bars, and even if it could, you wouldn't be able to make much sense out of it.

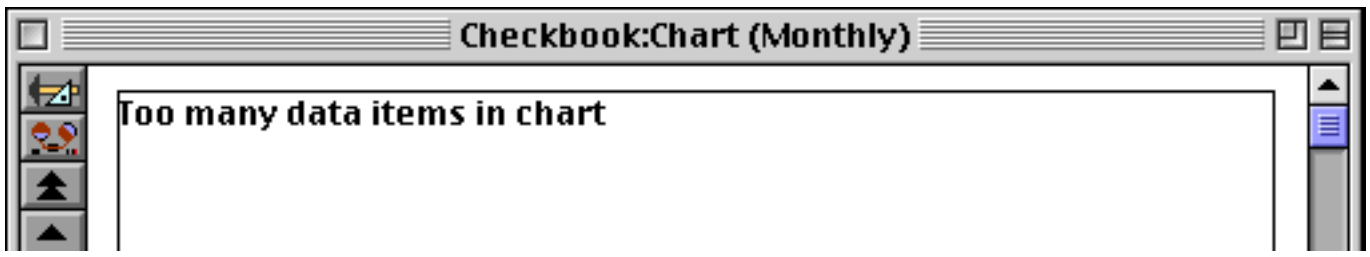


Before the chart can be drawn from this database the data must be reduced to a few summary records. To recap those techniques, first group (usually **Group Up**, see "STEP 1 - GROUP" on page 399), then calculate (**Total**, **Average**, etc., see "STEP 2 - CALCULATE" on page 403), then use **Outline Level** to collapse the database so only the summary records are visible (see "STEP 3 - OUTLINE" on page 409). The result is that hundreds or even thousands of data records are converted into a few summary records that can be charted easily like this. In this example the 411 data records have been summarized into nine monthly summary records. (The tenth summary record, the grand total, is not included in the chart).



A chart will only display data from a single summary level. Before it starts drawing a chart, Panorama checks the summary level of the first visible record in the database. It only charts information that is at the same summary level as that first record. This means that you do not have to remove the grand total from the bottom of the database—the chart will ignore it automatically. It also means that you must collapse the data in addition to grouping it. If you do not collapse the data with the **Outline Level** command (see "STEP 3 - OUTLINE" on page 409), the chart will try to display the raw data.

If you forget to group and collapse the database before opening the chart, Panorama will probably not be able to display any chart at all. Instead of drawing the chart, the chart will display the message Too many data items in chart.



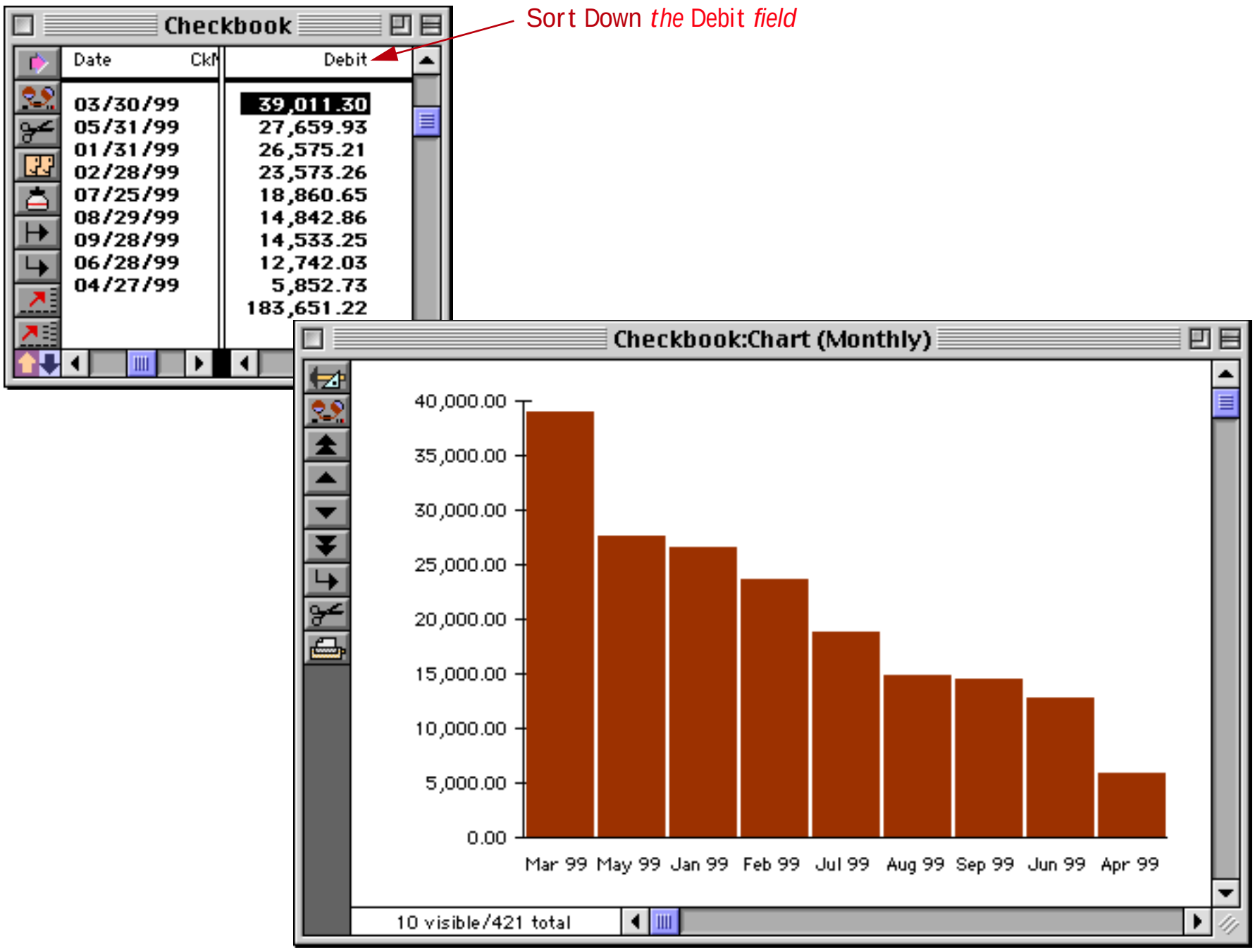
Don’t panic—just go back and prepare the database. As soon as the database is collapsed, the chart will appear.

Tip: You can also reduce the scope of a chart with the **Find/Select** command (see “The Find / Select Dialog” on page 369). For example, you may only want to chart data in the current year, or transactions on the west coast. You should perform selections like these before you group the database.

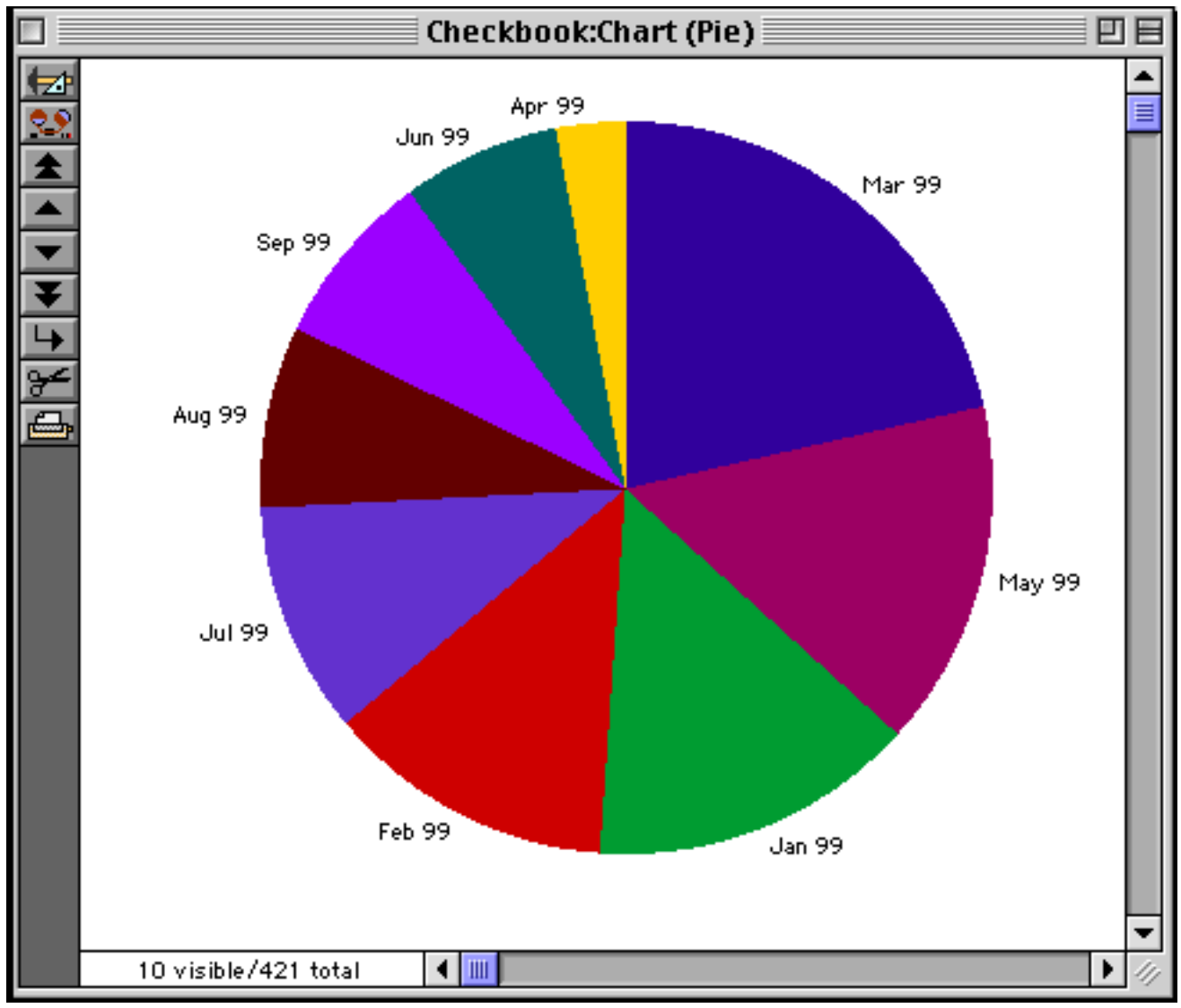
You may want to create a procedure that automatically groups, calculates, collapses, and then opens the form window containing the chart. For information about creating procedures see “Procedures” on page 1340.

Ranking (Sorting) the Chart Values

Sometimes you may want a chart to display the values in order from smallest to largest, or largest to smallest. To display a chart this way, sort the database by value after it has been collapsed (see "Sorting by Summary Value" on page 417). Click on the value field you want to sort by and choose **Sort Up** to display values from smallest to largest, or **Sort Down** to display values from largest to smallest. Here is a chart that shows the result of **Sort Down** on the value column. The months are no longer in chronological order, but in order of spending per month from highest to lowest.

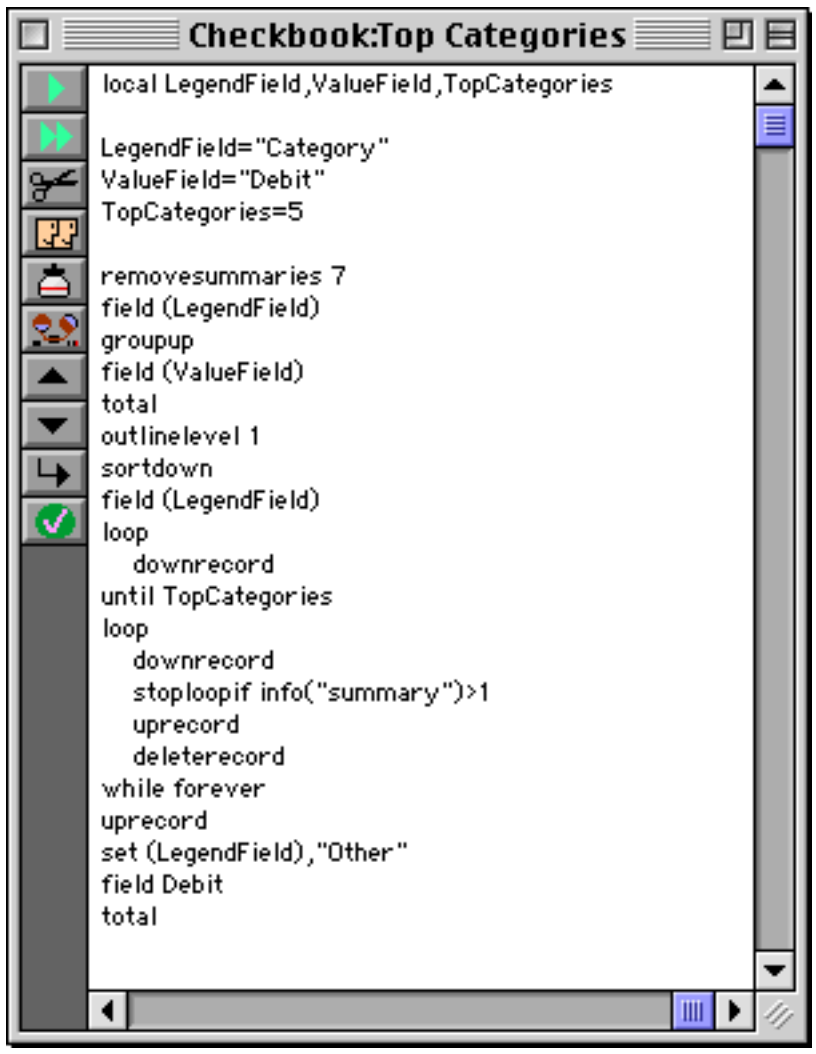


Pie charts are often displayed with the largest pie first (at 12 o'clock), then each smaller pie in order. Use **Sort Down** to sort the collapsed database in this order.

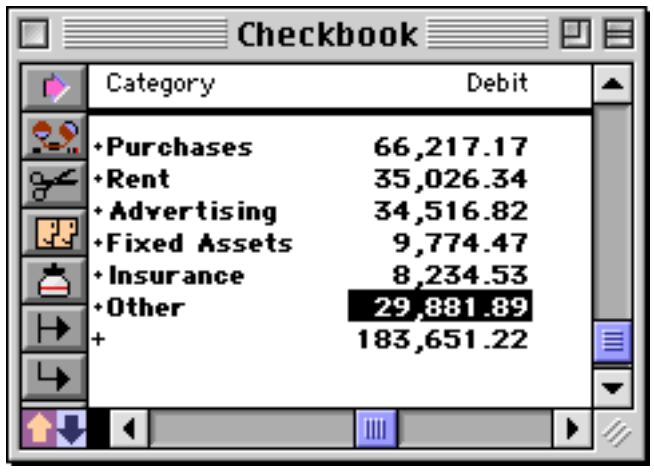


Charts with “Other”

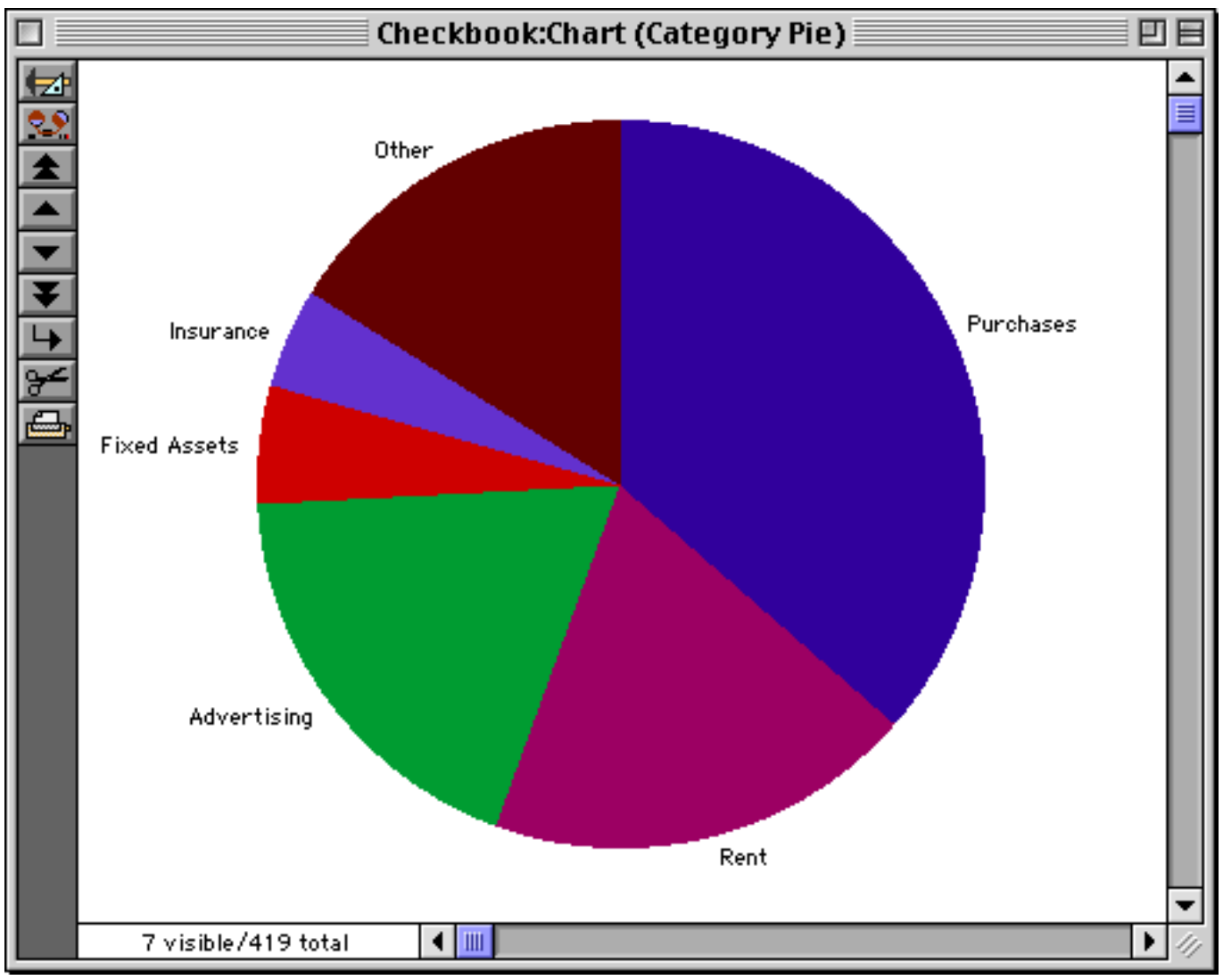
Often you’ll want to display a chart with all the small values lumped together under the legend Other. The best way to do this is with a procedure. Here is an example procedure that creates totals for the top three checkbook categories, with all other categories lumped together into an Other category.



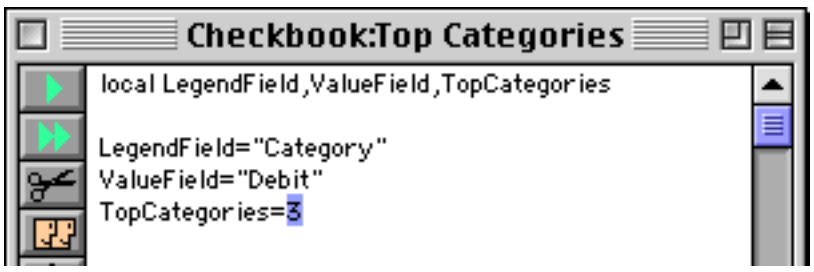
Running this procedure will produce summaries that look like this —



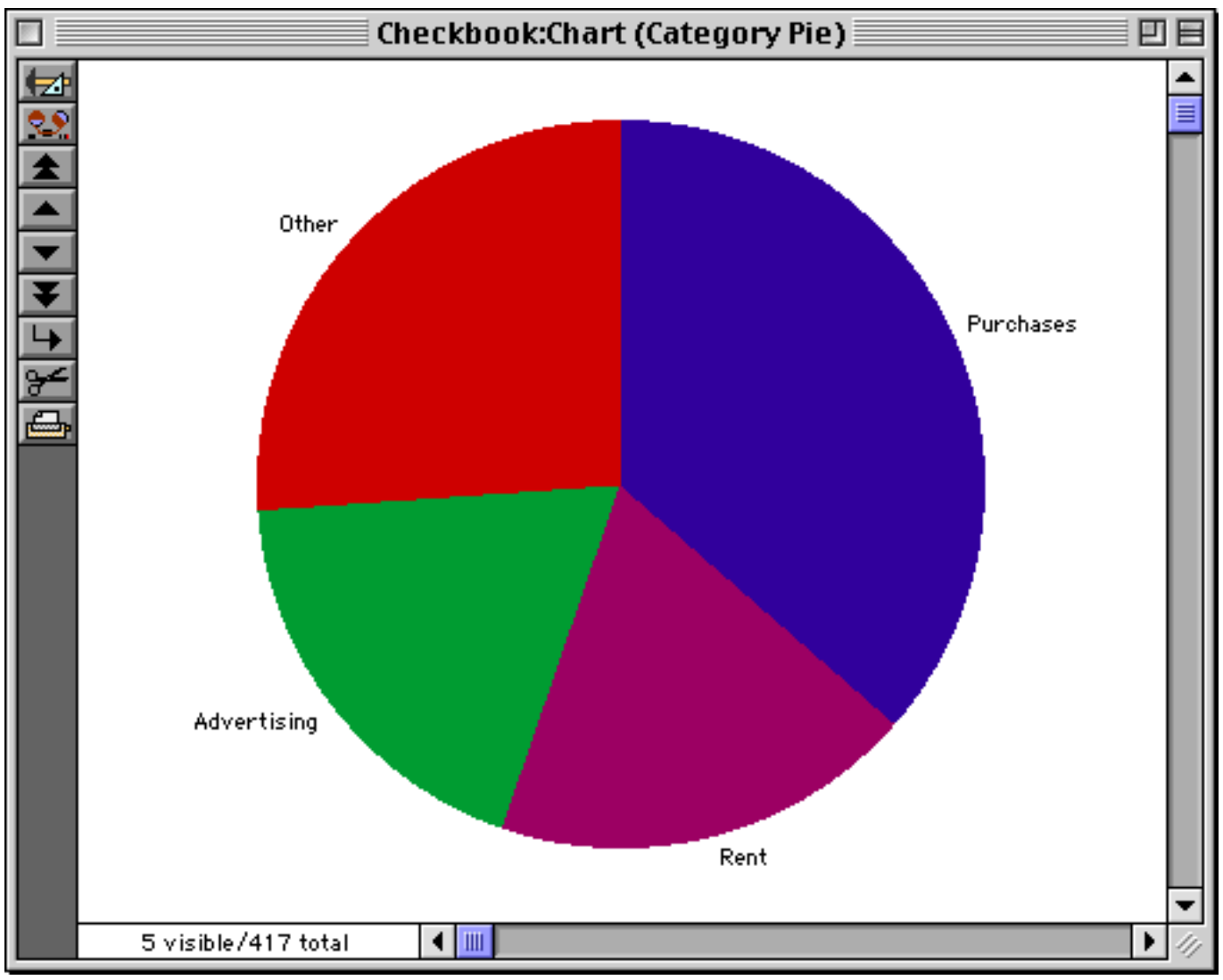
which can be easily charted like this!



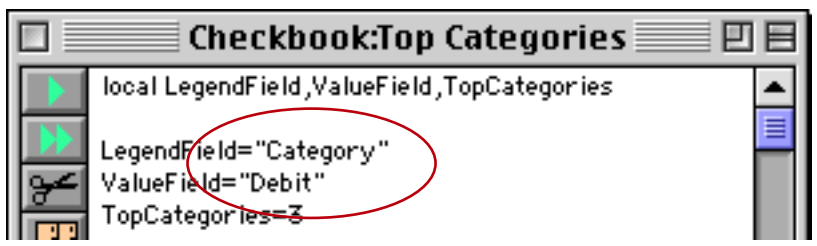
This procedure is designed to be easily adaptable to any database or situation. For example, to lump everything into only 3 categories plus other you only need to change one number.



Here's the revised chart.



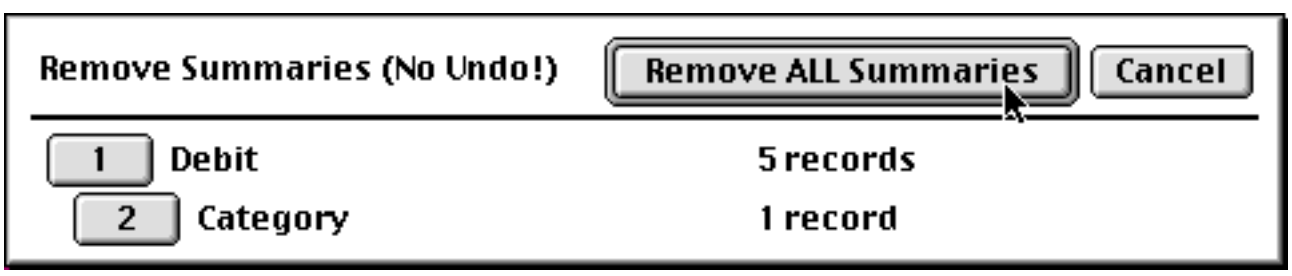
To adapt this procedure to a different database just change the field names.



By the way, it's kind of fun to run this procedure with the chart visible. Check it out!

Restoring the Original Data

When you are done with your chart, use the **Remove Summaries** dialog (Sort Menu) to remove the summary records (see "Getting Rid of Summary Records" on page 421). Simply press the **Remove All Summaries** button and all the summary records will vanish.



(Remember, if the chart is still visible, the message Too many data items in chart will re-appear when the summaries are deleted.)

Maximum Number of Chart Points

Most bar or pie charts only display a few values. But a complex line chart or scatter diagram may display hundreds of data values. You must tell Panorama in advance that you plan to display such a complex chart.

The complexity pop-up menu (below the chart type icons) allows you to specify one of three complexity levels—simple, medium, or very complex. A simple chart takes the least amount of memory and can display up to 50 data values—more than adequate for most applications. The table below lists the attributes of each complexity level.

Type of Chart	# of Values	# of Bytes
Simple	50	2100
Medium	200	3400
Medium	500	5000
Complex	2,000	20,000
Very Complex	5,000	41,000

If memory is tight, you'll want to use the simple chart option whenever possible. On the other hand, in today's world of machines with 32 mb, 64 mb, 128 mb or more memory there's not much reason to avoid using the more complex chart types.

Dressing Up Chart Appearance

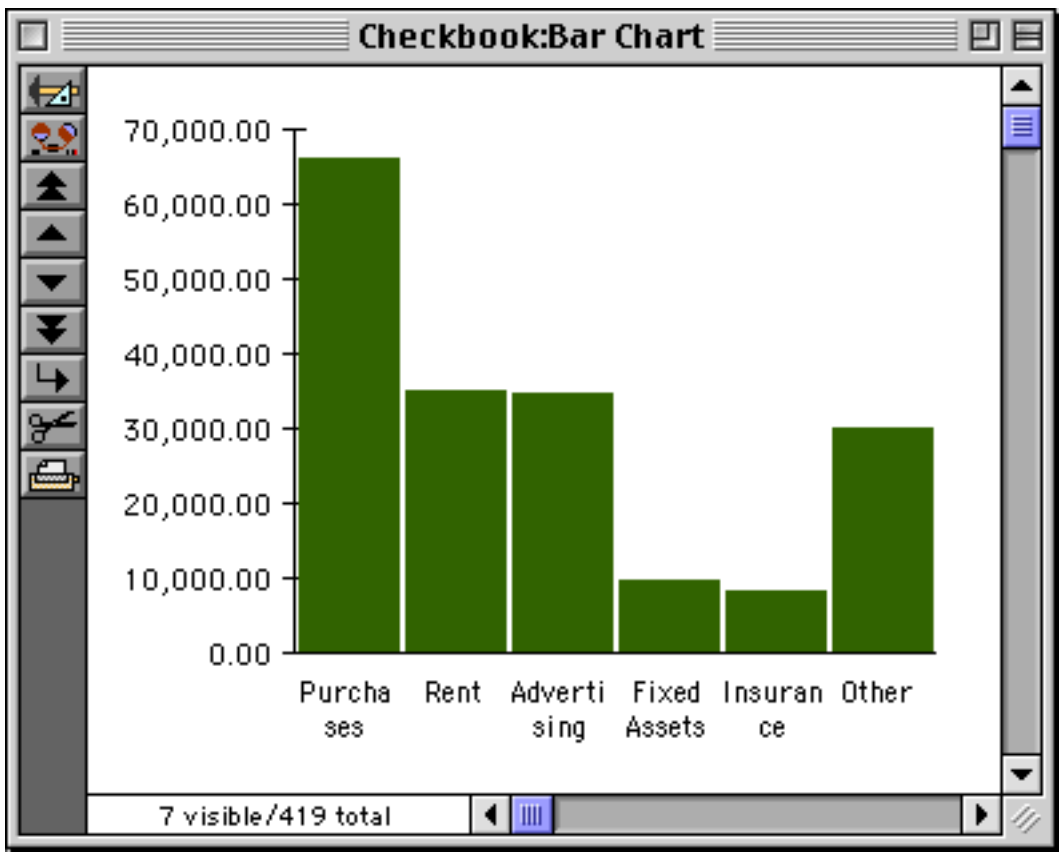
Panorama tries to display an attractive chart automatically. However you are not stuck with the chart Panorama provides. You can change the fonts, patterns, colors and many other chart attributes to suit your tastes.

Chart Font, Size, and Style

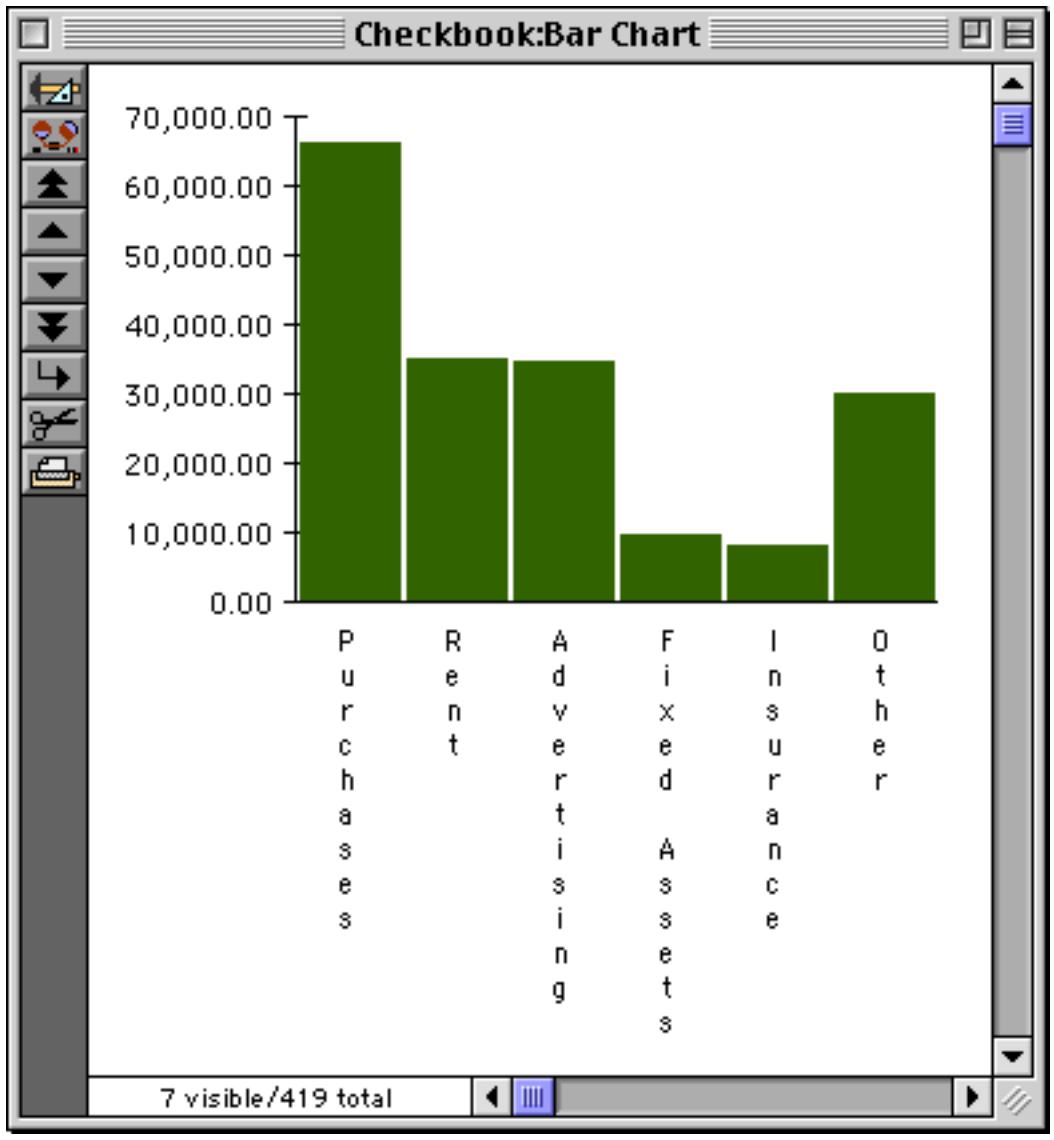
You can change the font, size, and style of the text in a chart just as you would for any text object. Simply select the chart and choose from the **Font** (see "Font" on page 529), **Size** (see "Text Size" on page 531), and **Style** (see "Text Style" on page 532) menus.

Vertical Legends

Legends are normally displayed horizontally under bar, line and area charts. However, as in this example, sometimes the legends are just too wide to fit under the bars!



The Vertical Legends check box allows the chart to display legends vertically instead of horizontally.



Adjust the fifth handle of the chart to allow enough room for the vertical legends.

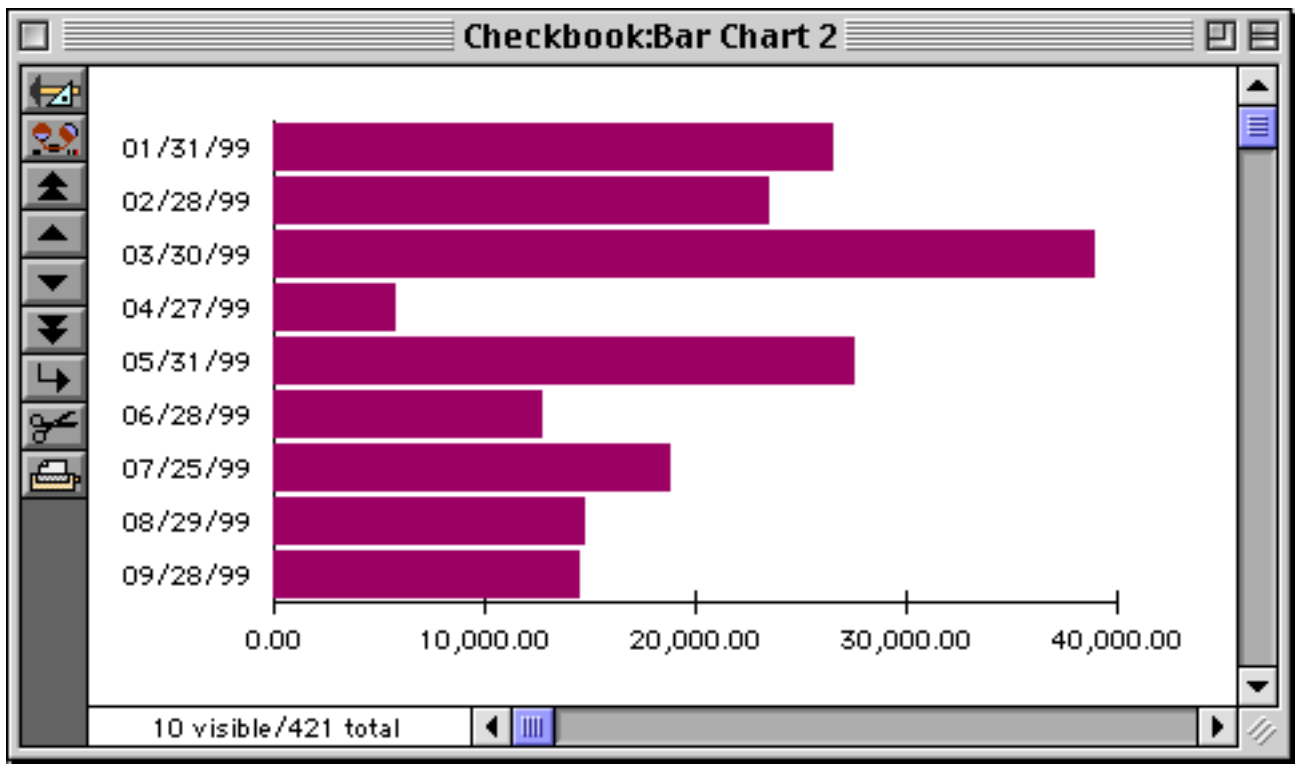
Note: If a chart uses horizontal legends, there may not be enough room on the chart for a long legend. When this happens, the legend is not displayed. Four solutions are possible: 1) use vertical legends, 2) use shorter legends, 3) use a smaller font size, or 4) increase the size of the chart.

Output Patterns

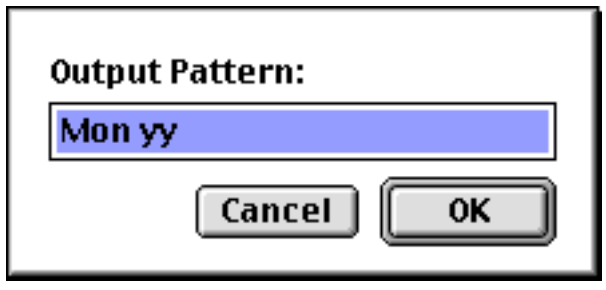
Panorama displays the numbers along the axis of the chart using the output pattern (see "Numeric Output Patterns" on page 290) set up for the first value field. For example if the first value field's output pattern is `$#.##`, the chart will display a dollar sign in front of each tick mark value.

If the legend field contains numbers or dates, you can use the **Output Pattern** command (Text Menu) to format the legend. For example, if the chart is designed to display data grouped by month, the output pattern could be set to `mm-YY` or `Mon yy`.

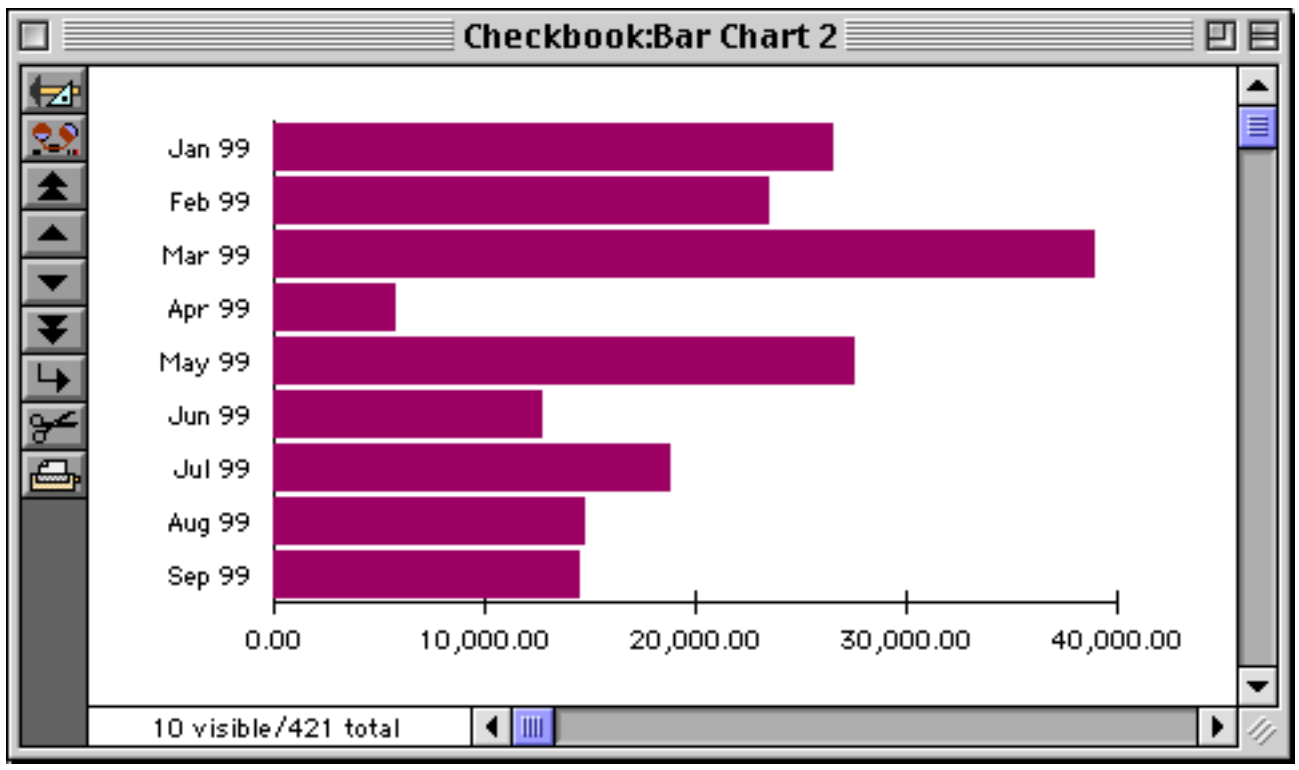
Here's an example of a chart without any output pattern set. The database has been grouped by month, so the chart legend shows the date of the last transaction in each month.



We can make a much better chart by selecting the chart object with the **Pointer** tool, choosing the **Output Pattern** command, and setting the pattern to `Mon yy`.



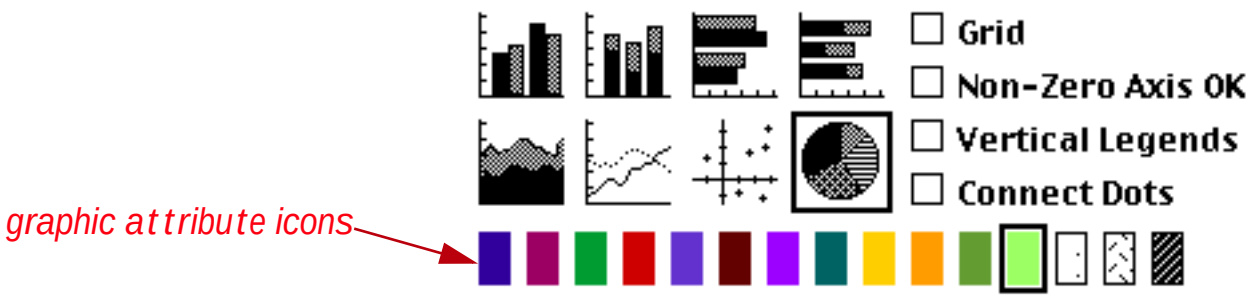
Here's the revised chart.



Graphic Attribute Icons

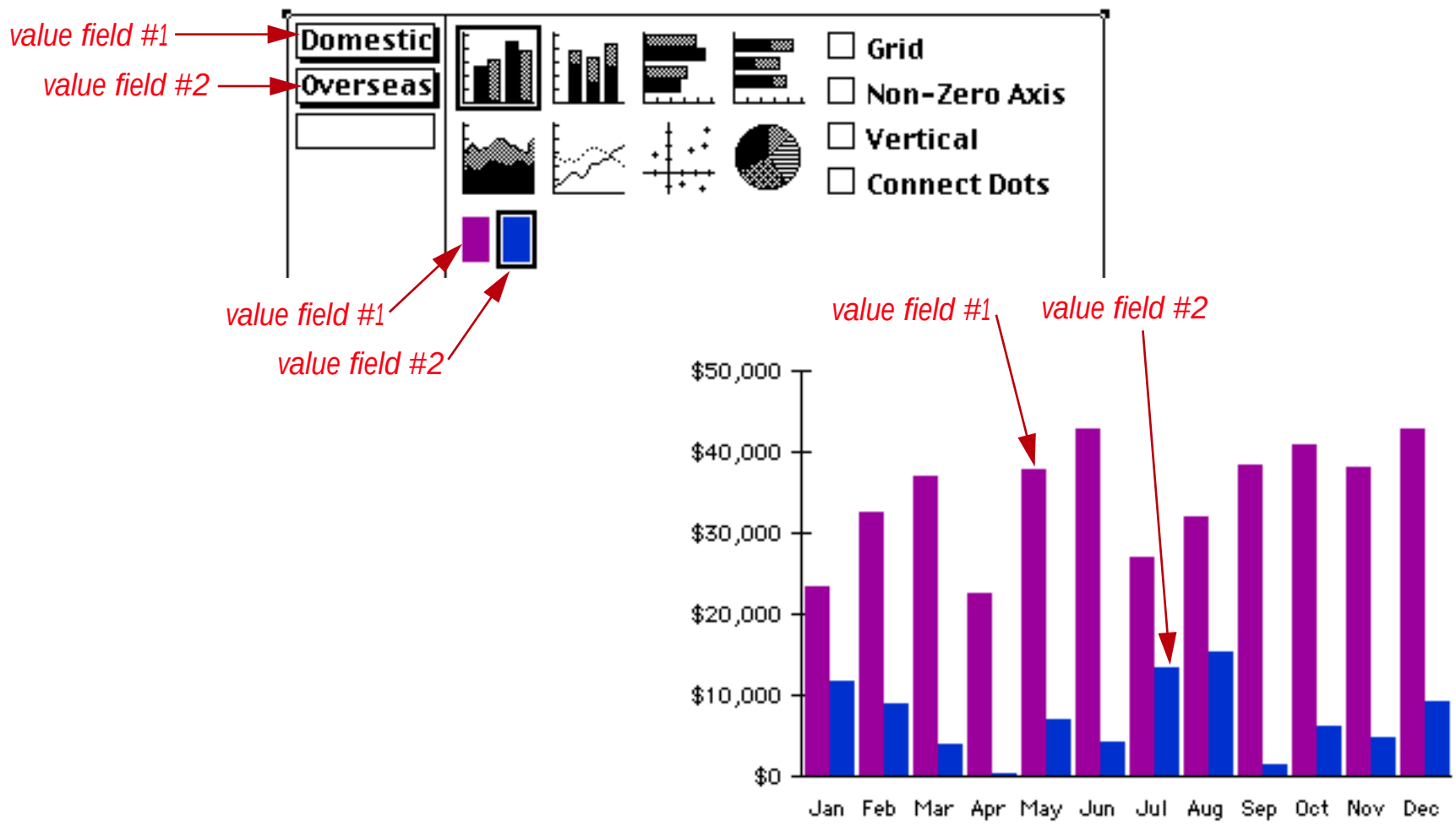
Panorama normally lets you select the graphic attributes of an object (pattern, border, color, etc.) by selecting the object and then choosing the attributes from the Graphic Control Strip or Graphic sub-menus. A chart, however, can have several different components that need to have separate graphic attributes. For example, you must be able to set the attributes for each slice of a pie chart independently.

To allow you to assign different attributes to different components, the chart object contains a row of graphic attribute icons just below the chart type buttons.

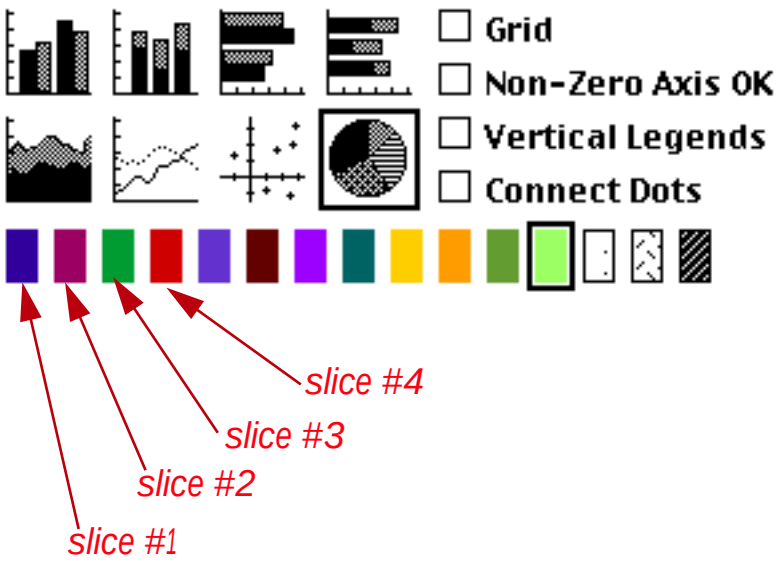


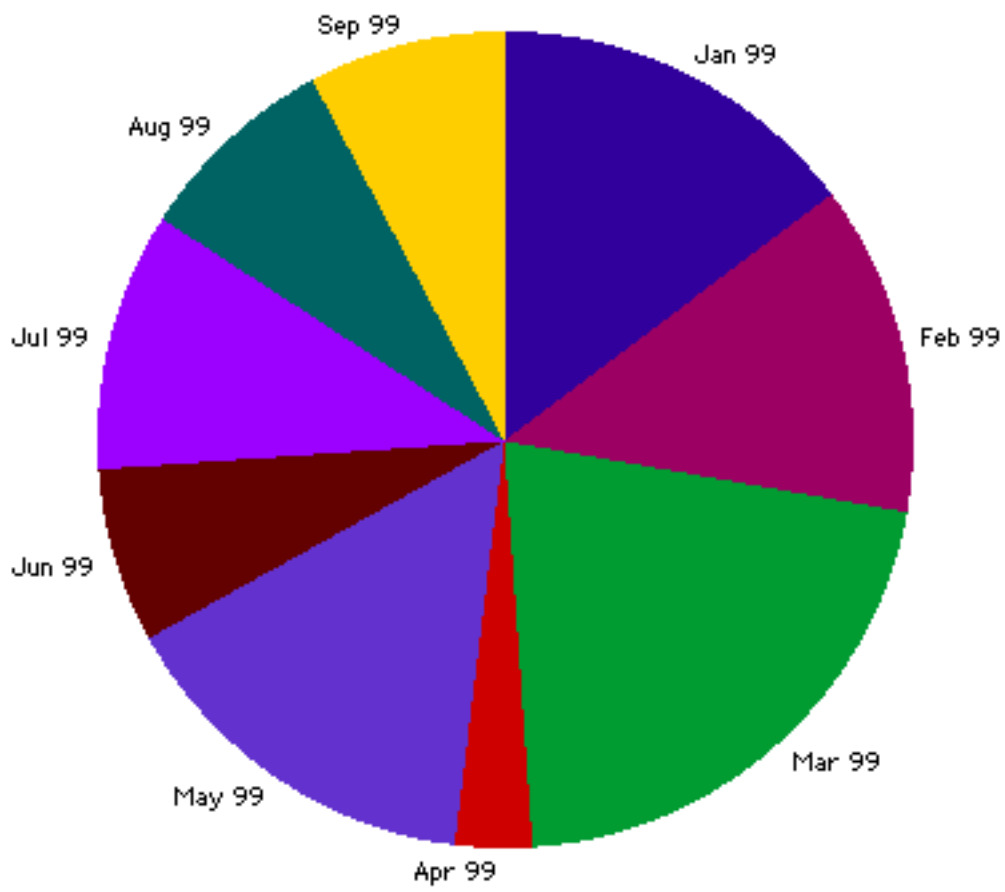
The number of graphic attribute icons available depends on the type of chart and the number of value fields.

For most types of charts the number of graphic attribute icons is the same as the number of value fields. The leftmost icon is used to set the attributes for the first value field, the next icon represents the second value field, etc.



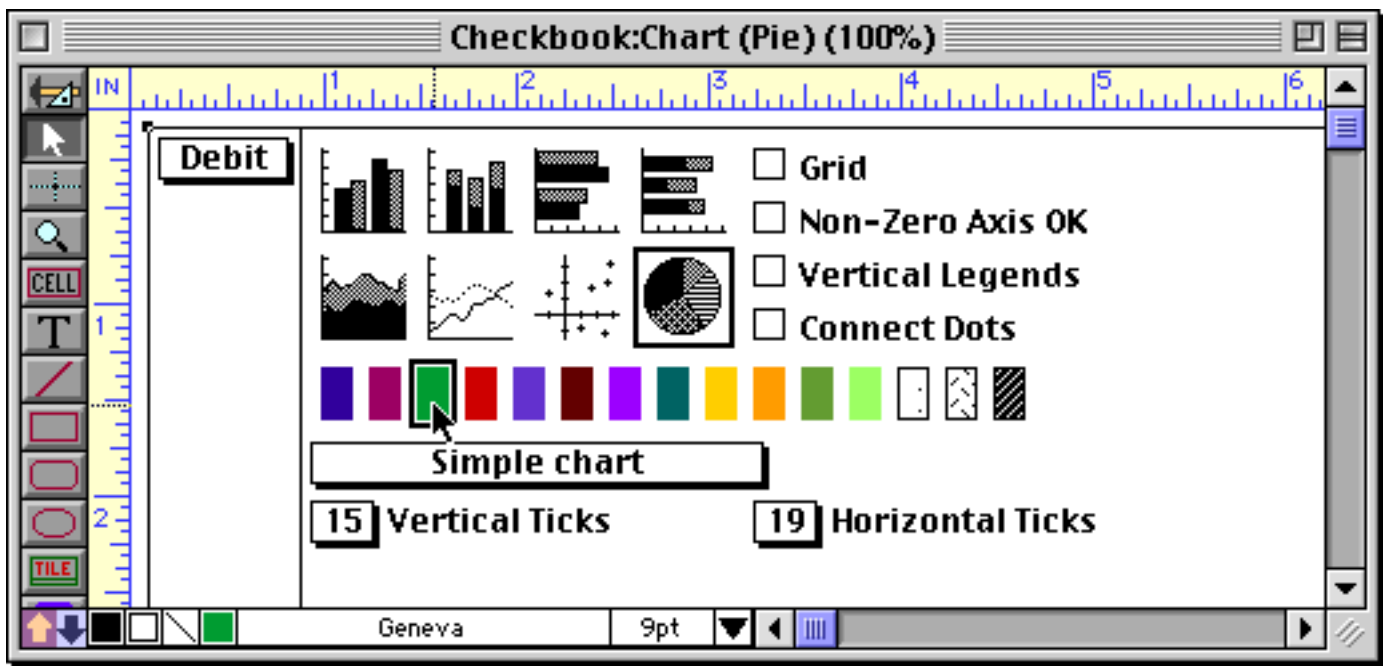
A pie chart always has 15 graphic attribute icons (although some may be invisible if the chart object is too small). The leftmost icon is used to set the attributes of the first pie slice (12 o'clock), the second icon represents the next slice, etc. If the pie has more than 15 slices, the 16th slice will wrap back to the beginning and use the first graphic attribute icon. The 17th slice will use the second icon, etc.



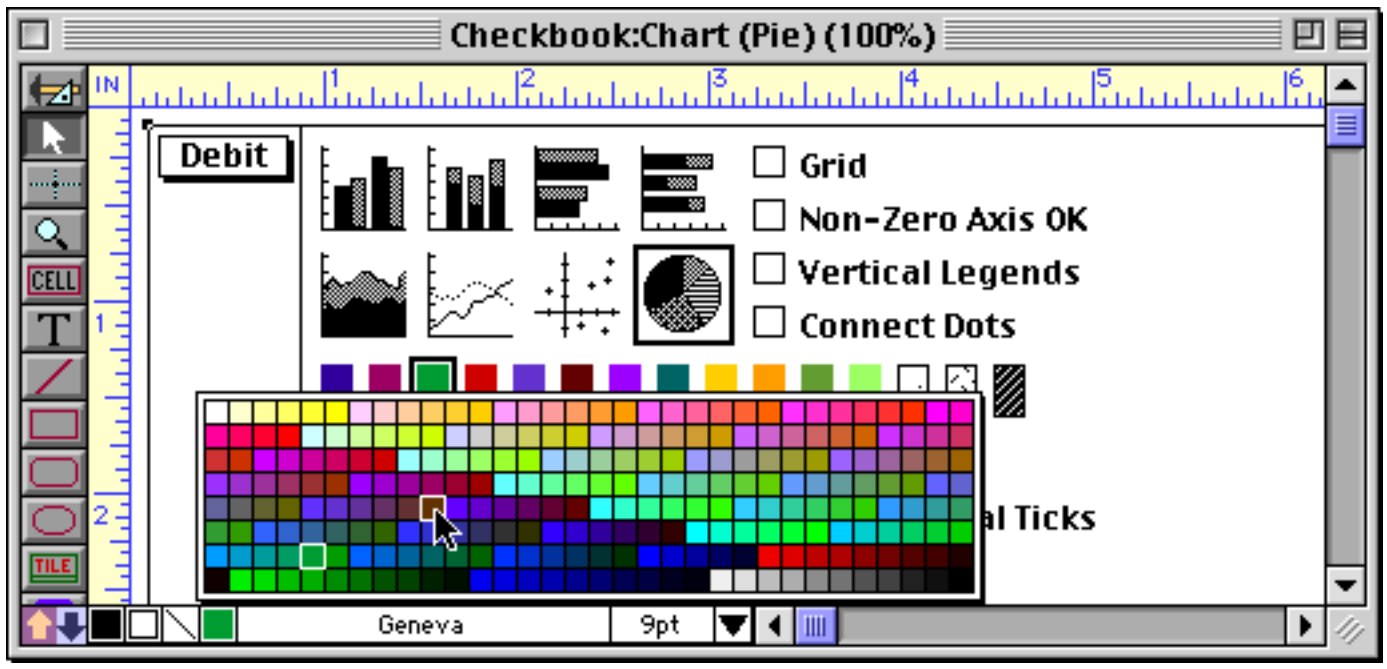


To change the graphic attributes of one of these icons, simply click on the icon to select it and then choose the attributes from the Graphic Control Strip or the Graphic sub-menus (Fill Pattern, Line Pattern, Line Width, Color). The icon will change to show the effect of the new attributes. Panorama draws a box around the selected icon to show that it is active.

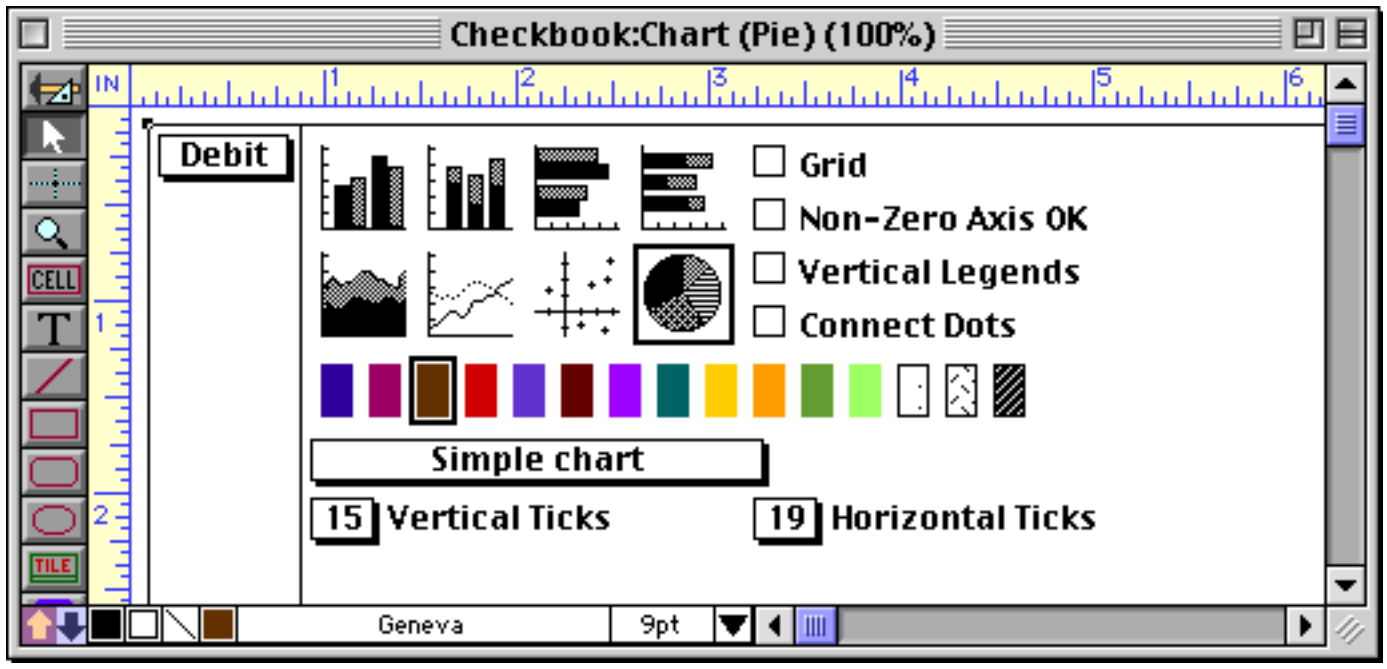
For example, to change the color of the third slice of a pie chart, click on the third graphic attribute icon.



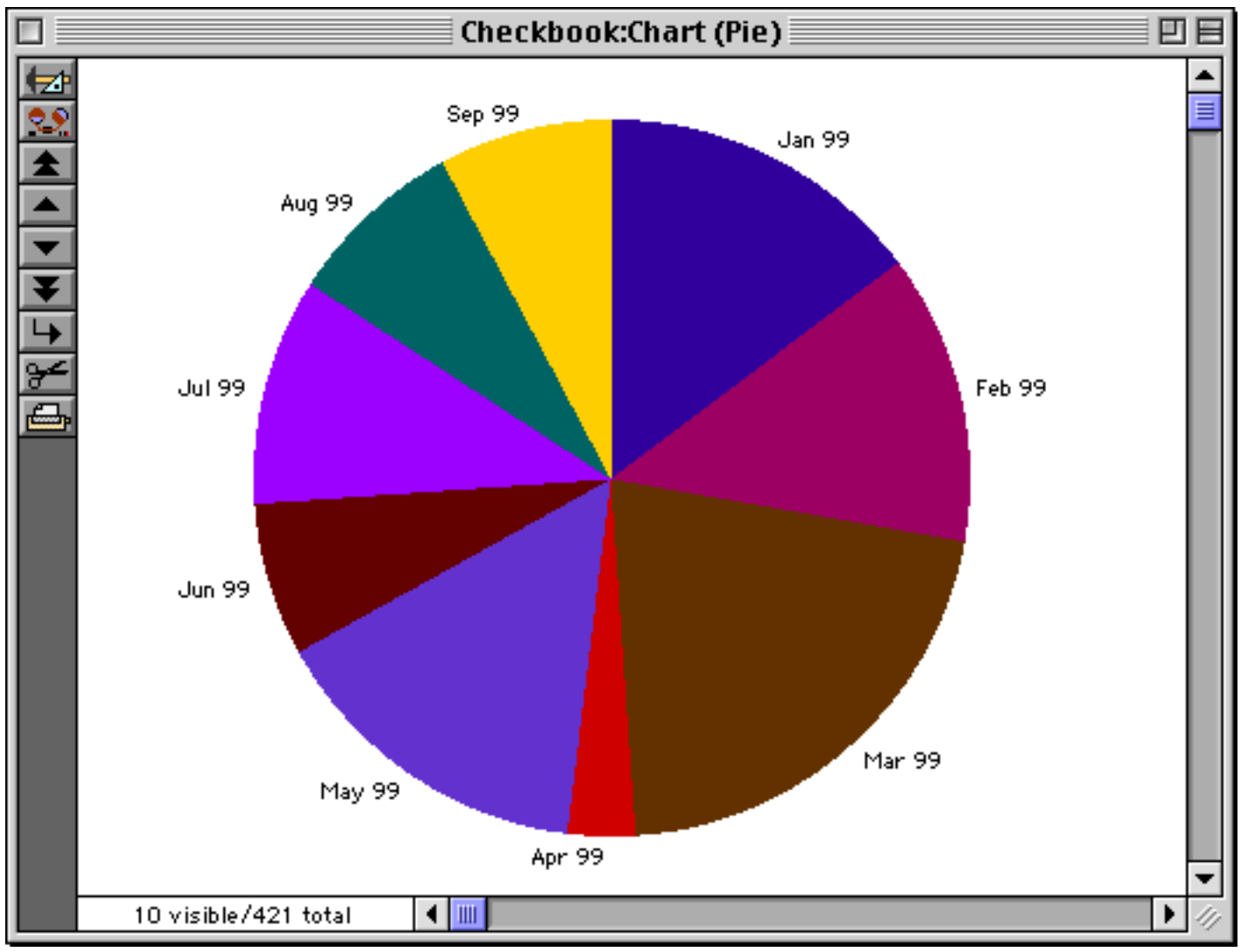
Once the icon is selected, pick the color from the Color box in the Graphic Control Strip.



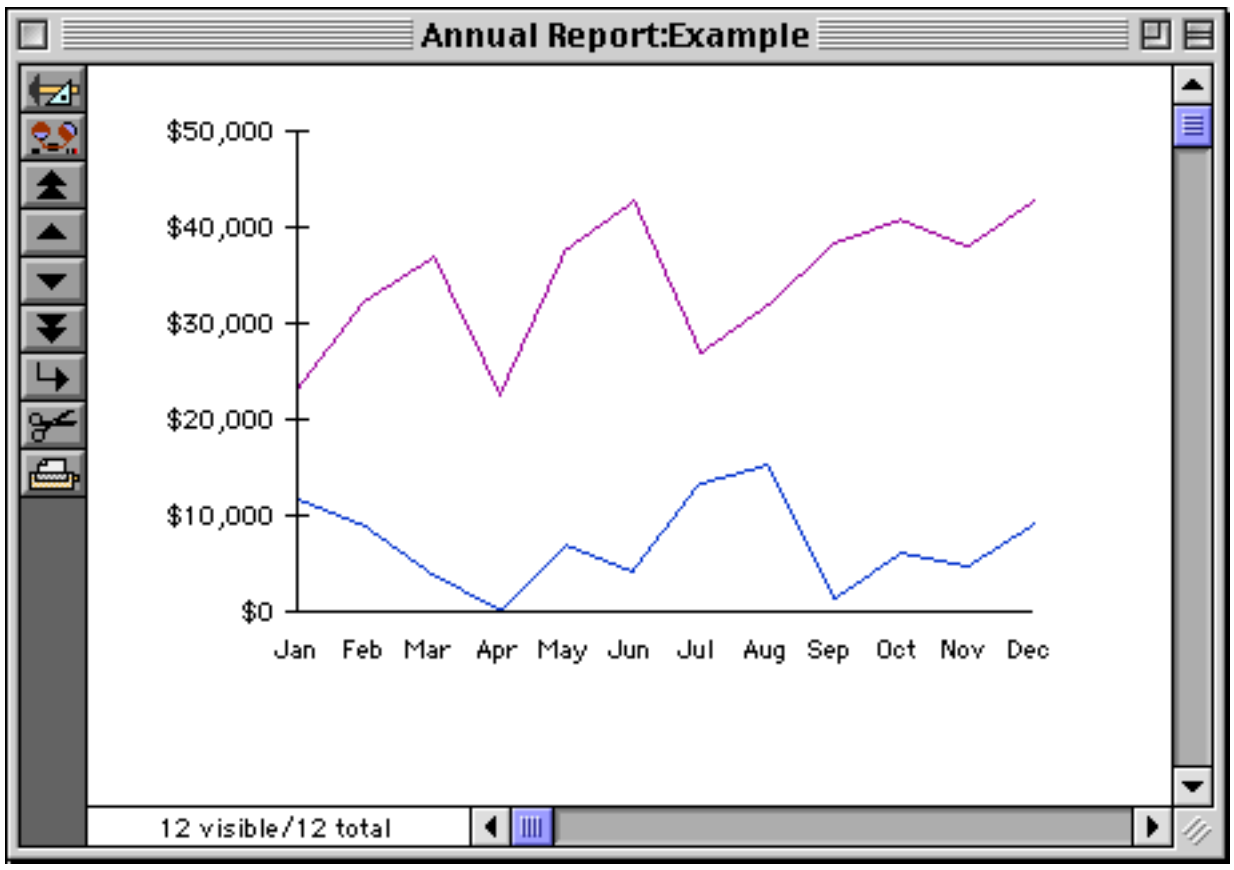
When you release the mouse the icon updates to show the new color.



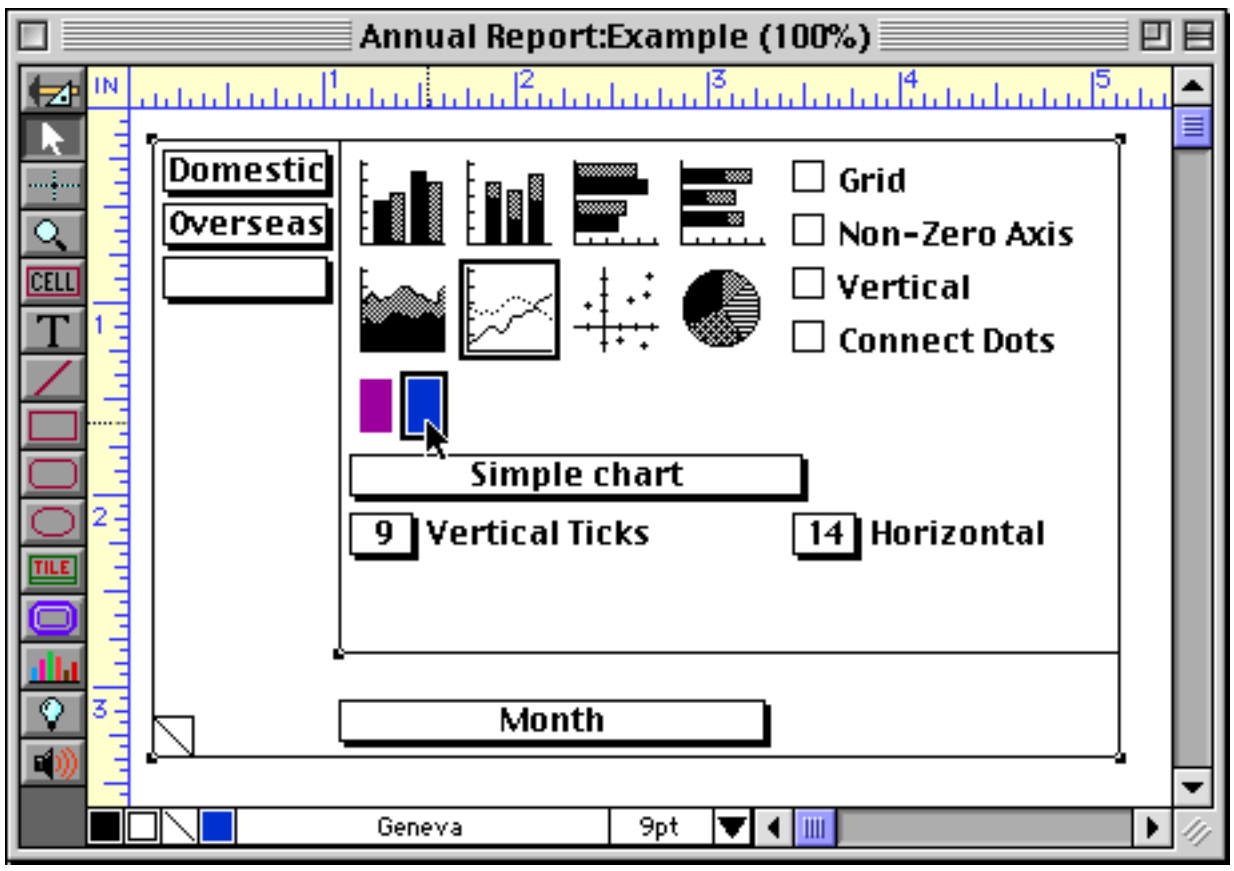
Here's the updated chart with a brown third slice.



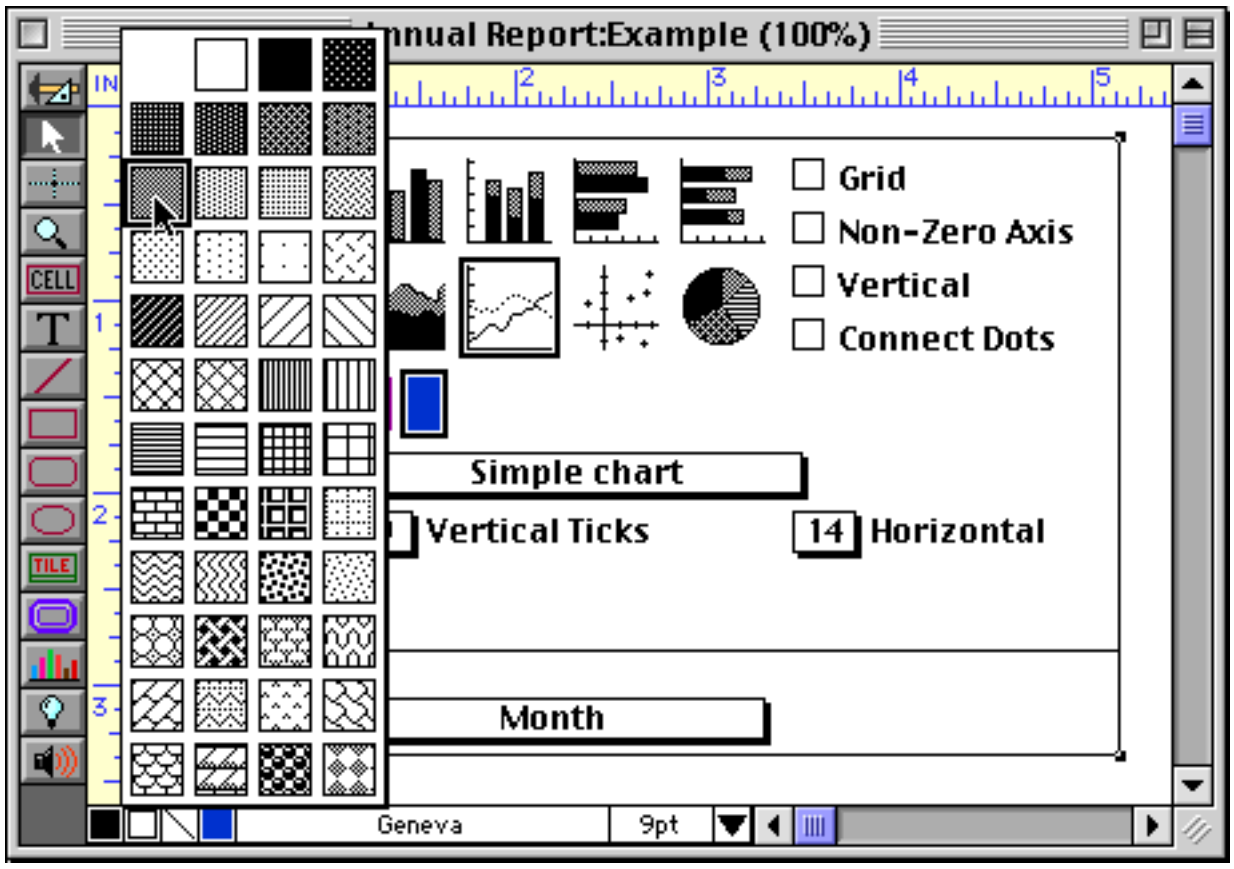
Here is a line chart with two lines — domestic (purple) and overseas (blue) sales.



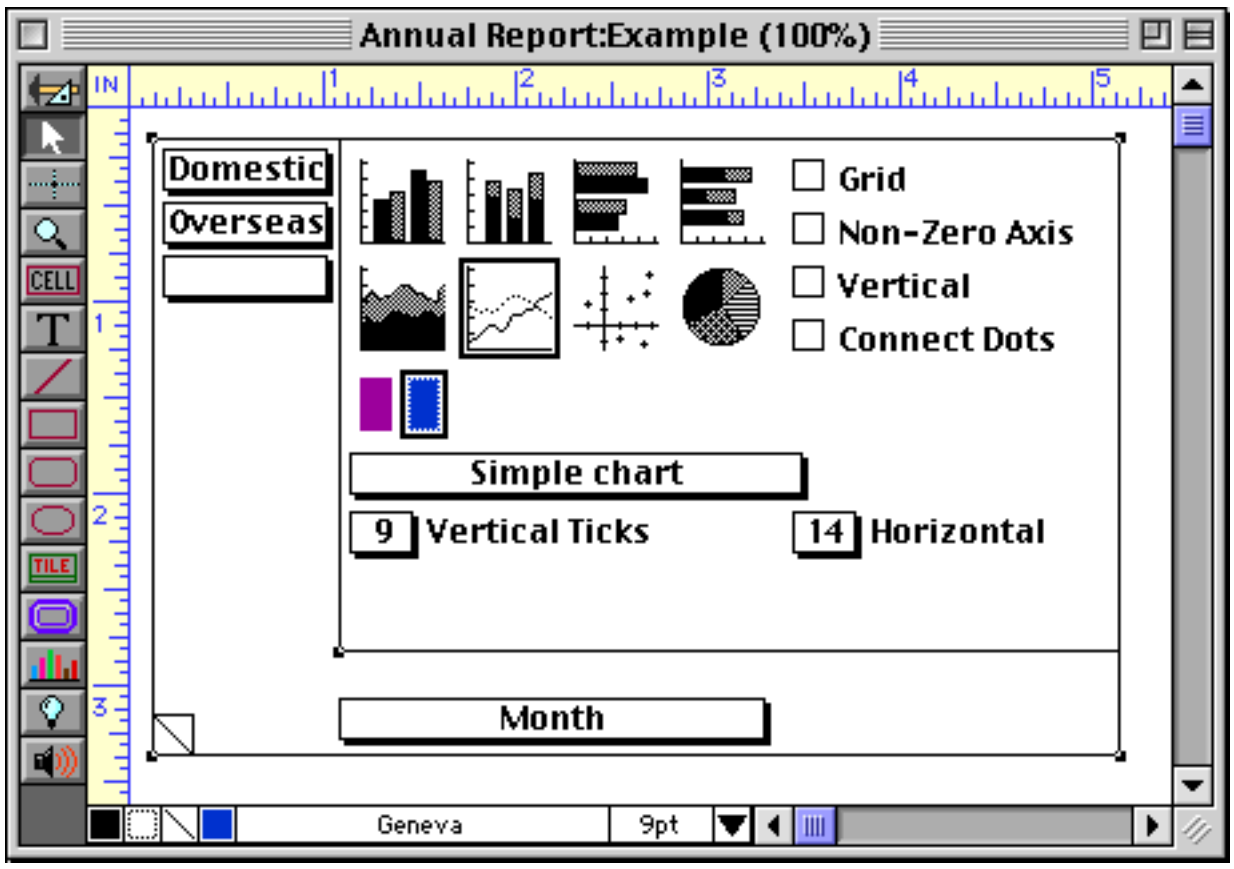
Suppose you wanted the overseas line to be dotted instead of solid. Switch to Graphic Design mode and click on the second graphic attribute icon.



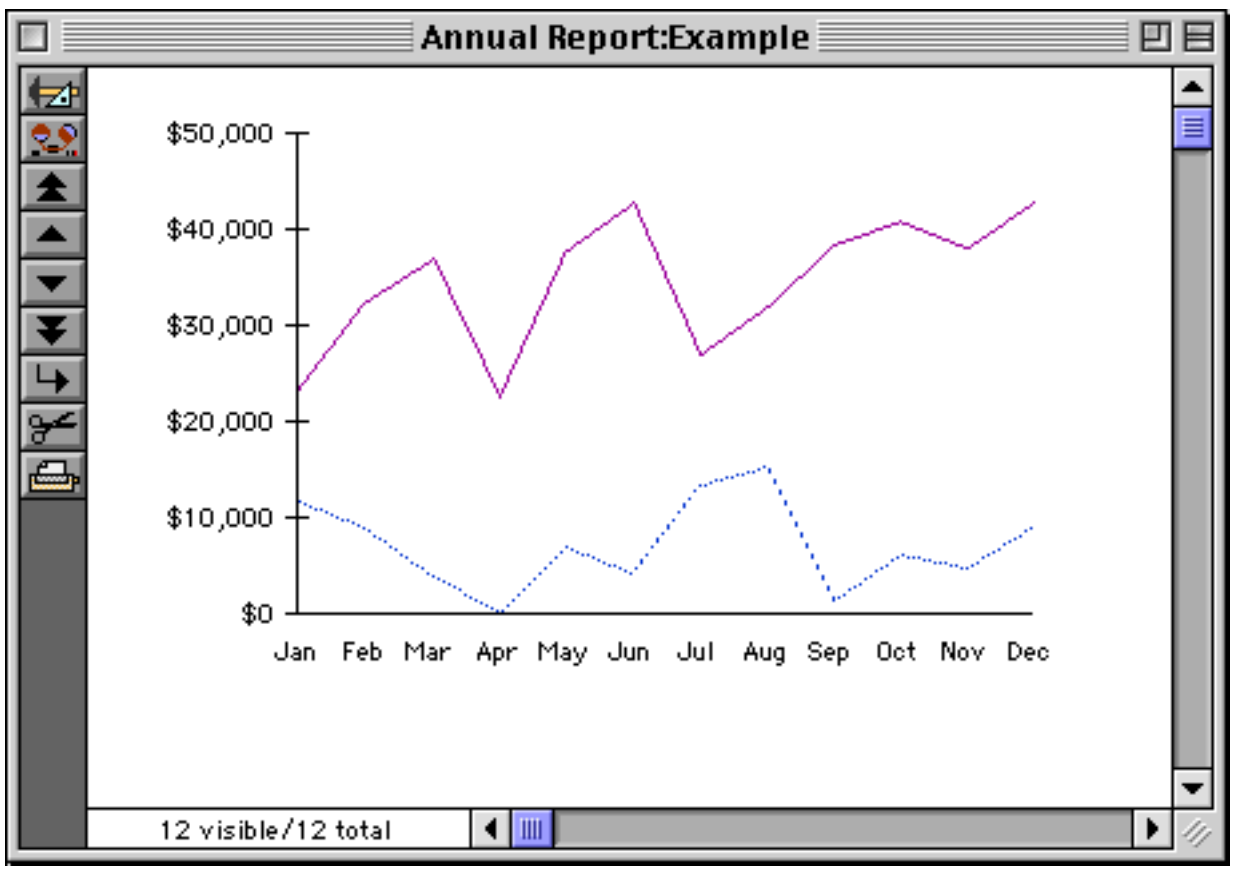
Now choose a gray pattern from the line pattern menu.



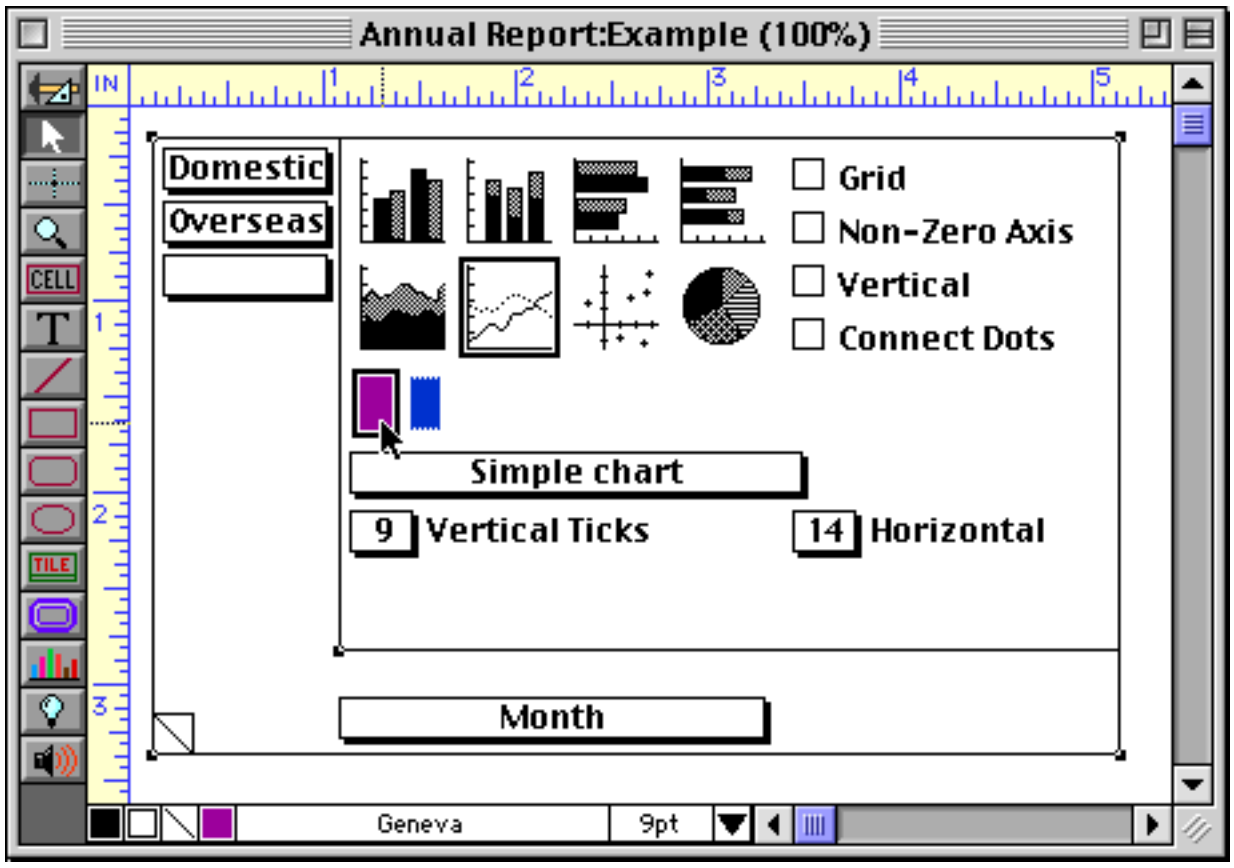
When you release the mouse you will see a dotted border around the graphic attribute icon.



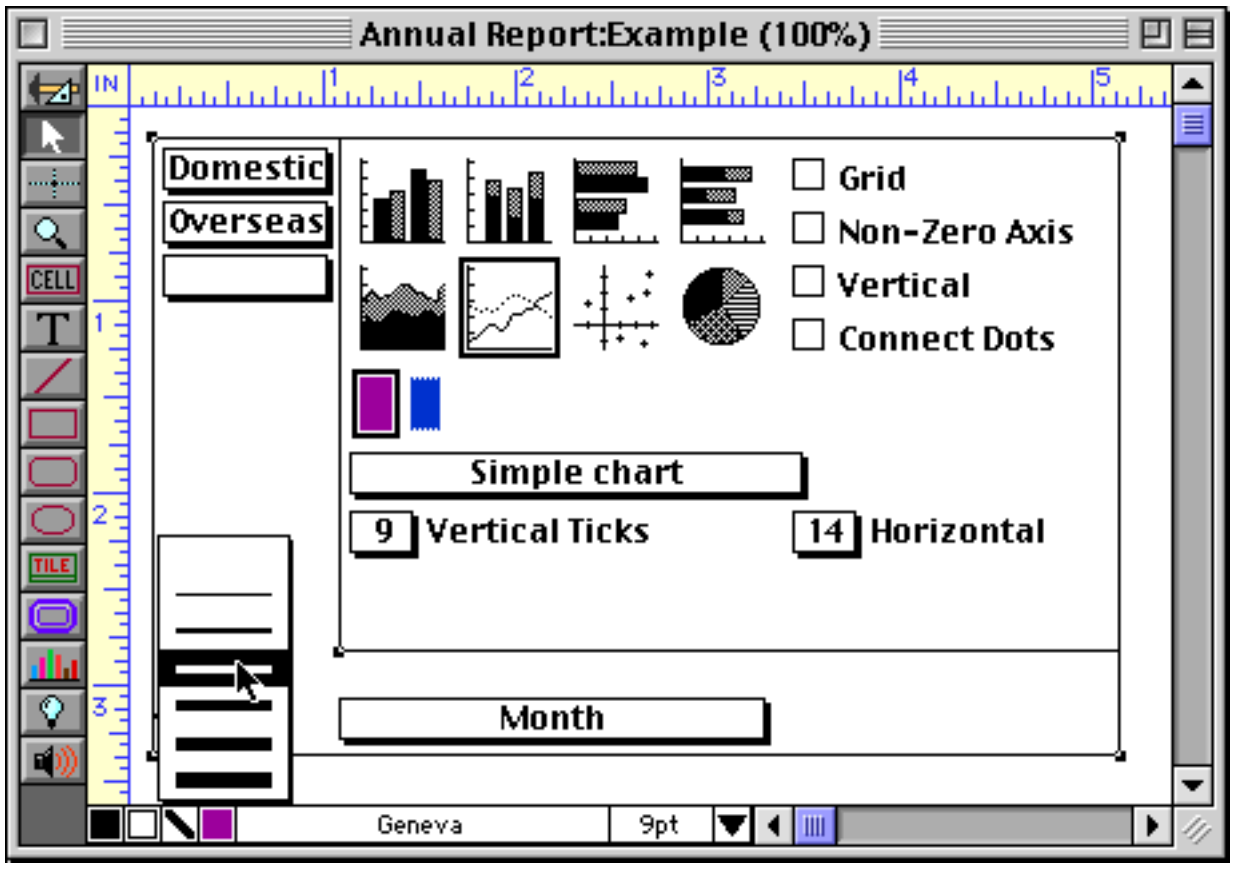
Switch to Data Access Mode to see the revised chart.



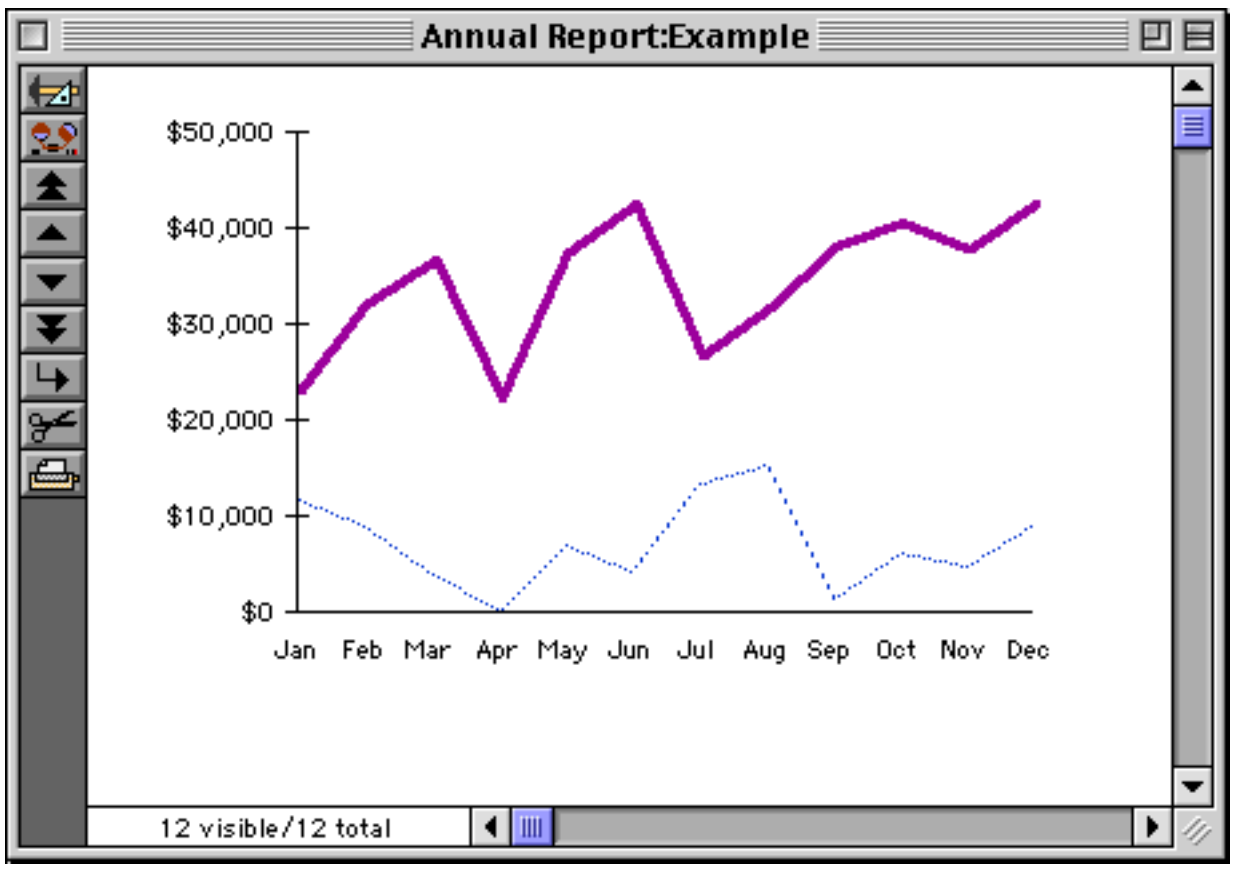
Let’s make one further revision. Go back to Graphic Design Mode and select the first graphic attribute icon.



Now choose a double width line from the Line Width menu.

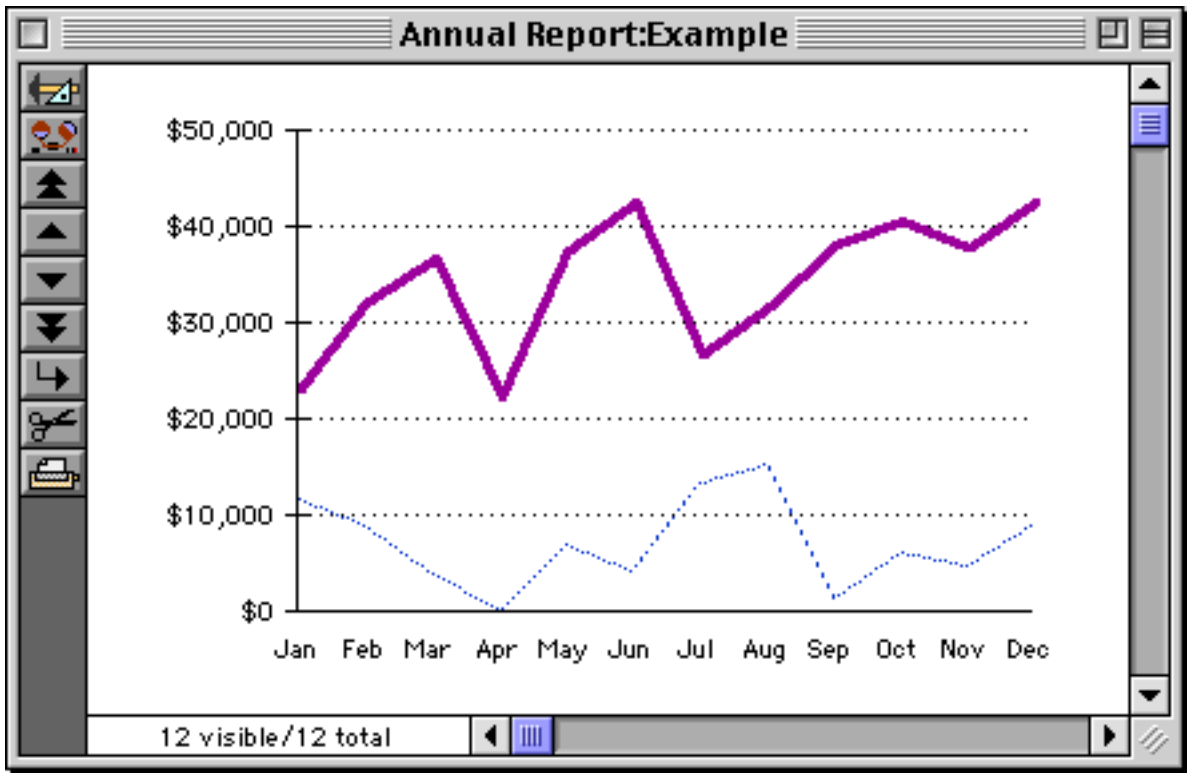


Here's the result of this modification.



Grid

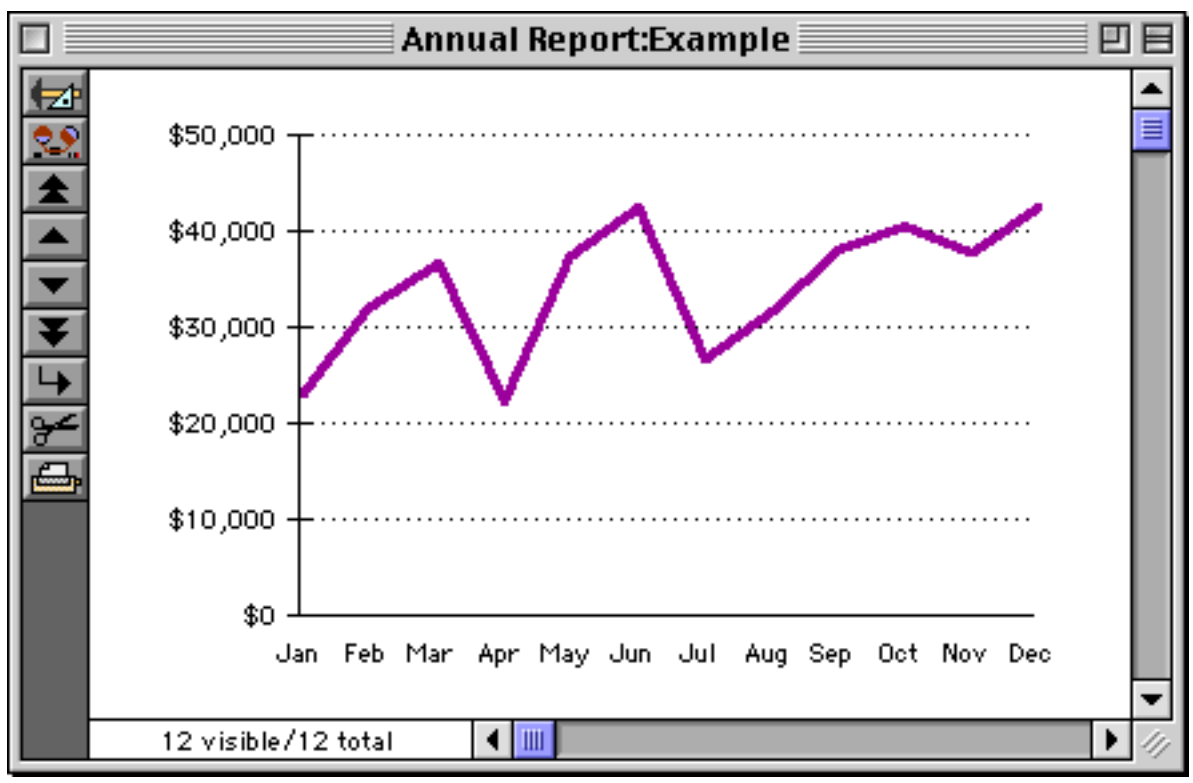
The Grid checkbox tells the chart to display grid lines from each tick mark.



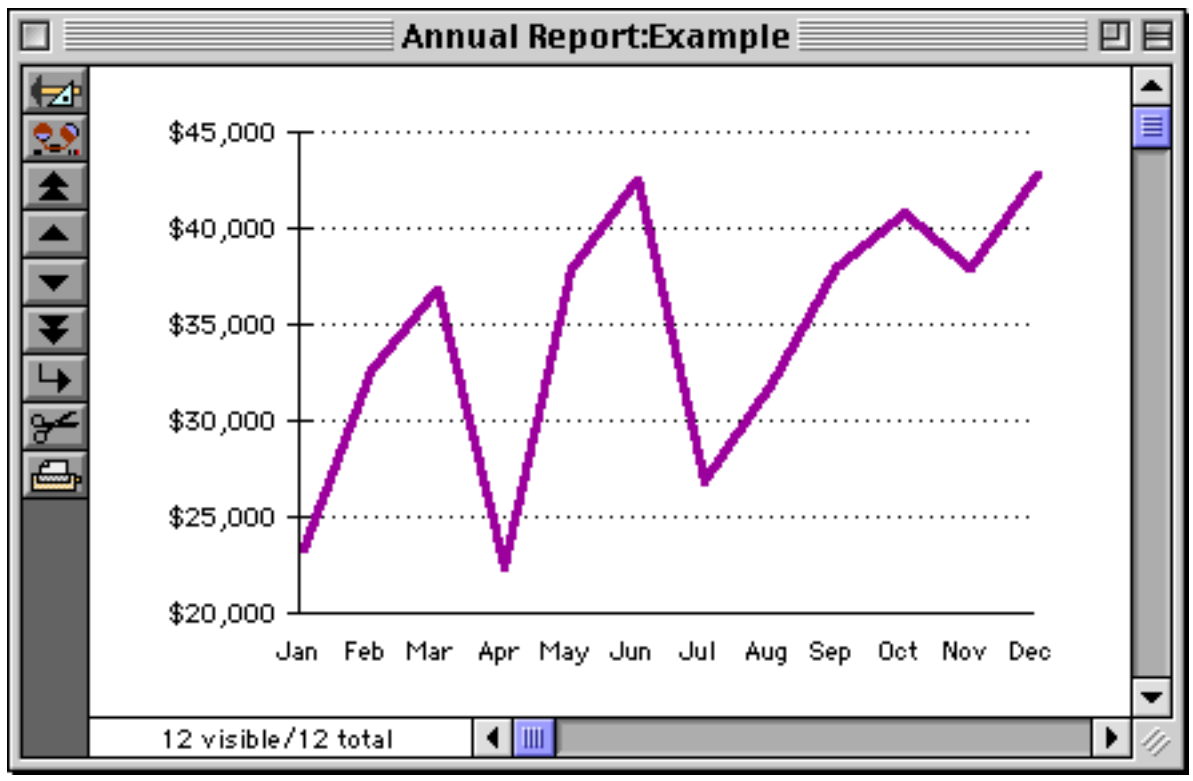
You can also turn grids on or off by double clicking any of the chart type icons. The Grid checkbox is ignored when displaying pie charts.

Non-Zero Axis OK

Panorama normally displays all charts from zero. If you check the Non-Zero Axis OK option, Panorama is allowed to display charts that don't go all the way to zero. Here's a normal chart that includes zero even though the lowest value is over $20,000.



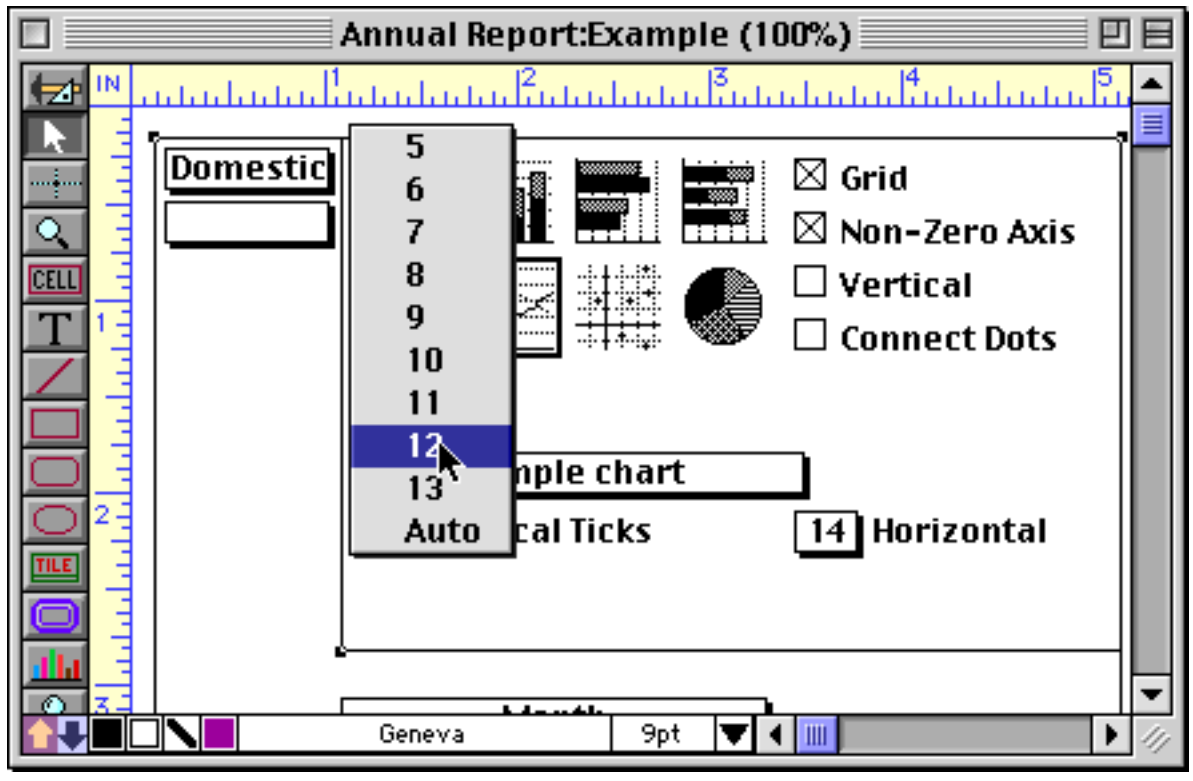
Here's the same chart with the Non-Zero Axis OK option enabled.



As you can see, a chart with a non-zero axis can greatly exaggerate the differences between values. Because charts that are not based on zero can be very misleading, they should be avoided if possible.

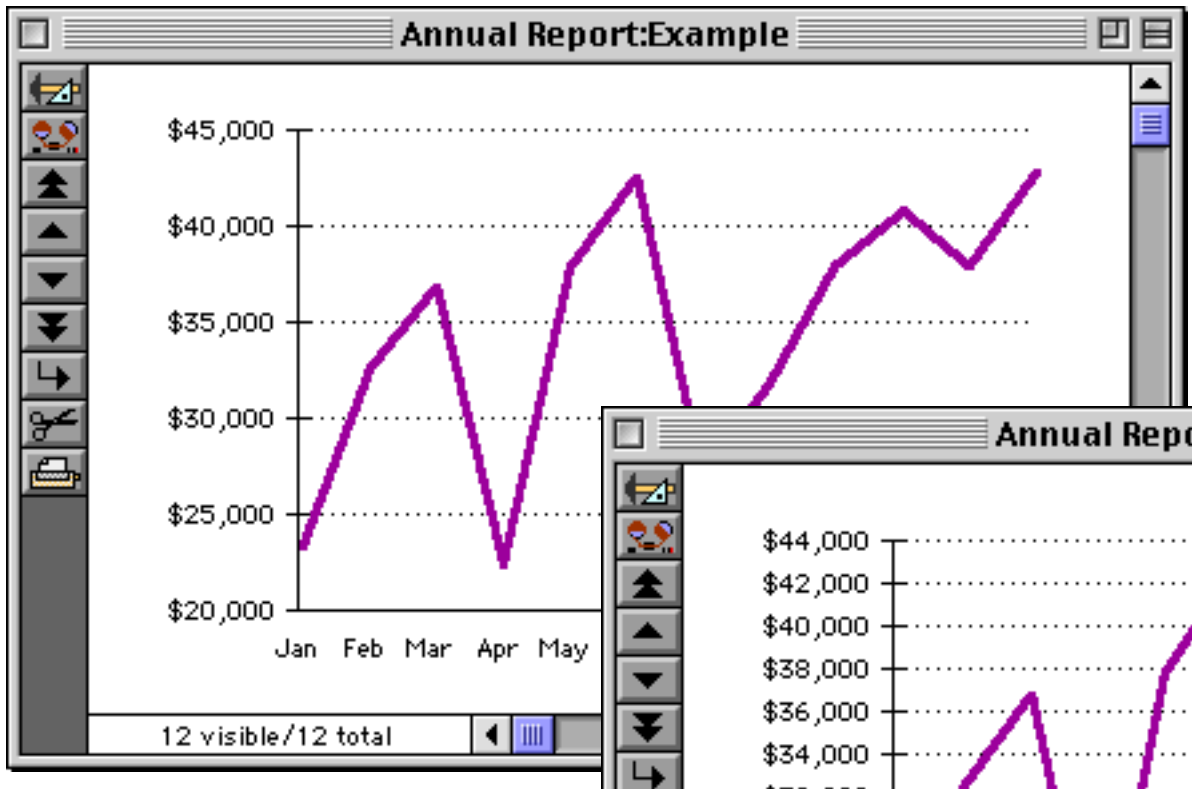
Tick Mark Spacing

Panorama normally sets the number of tick marks automatically depending on the data values and the chart size. You can override the automatic tick mark spacing using the two pop-up menus near the bottom of the chart object.



Here's what this chart looks like with automatic tick marks and set to 12 tick marks.

automatic tick marks



12 tick marks

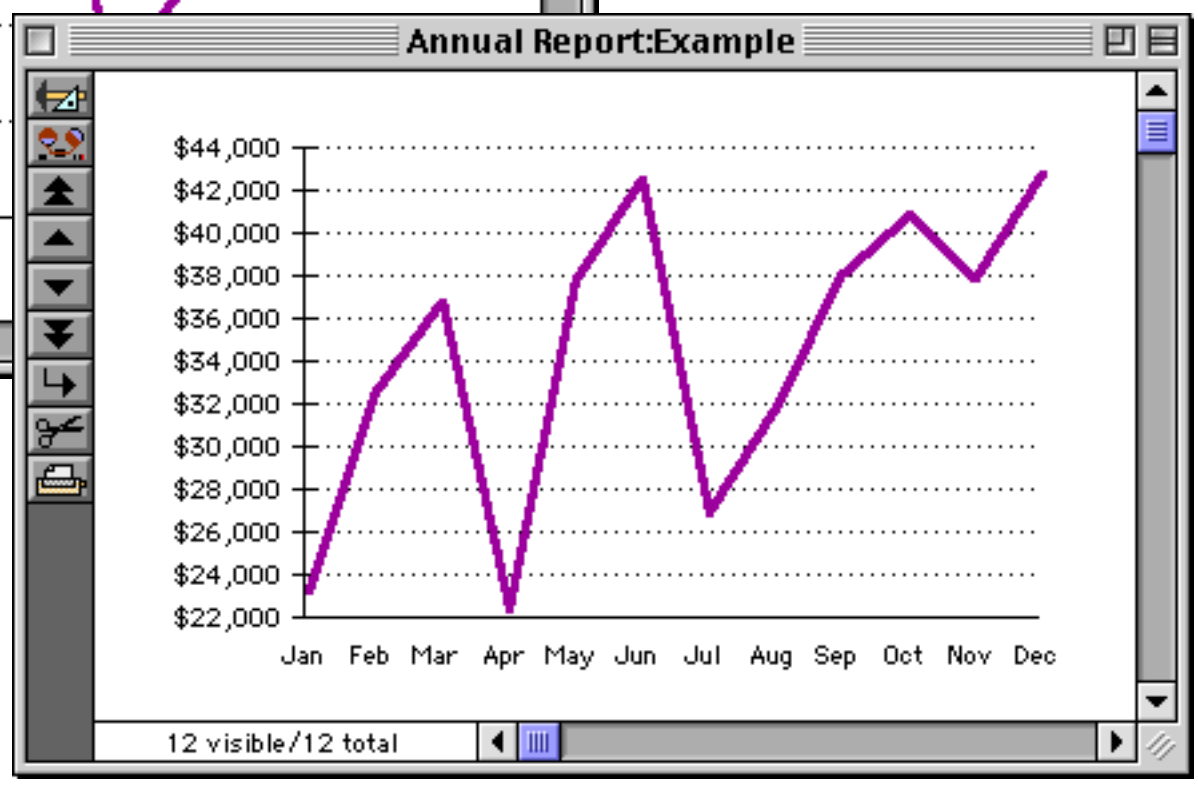
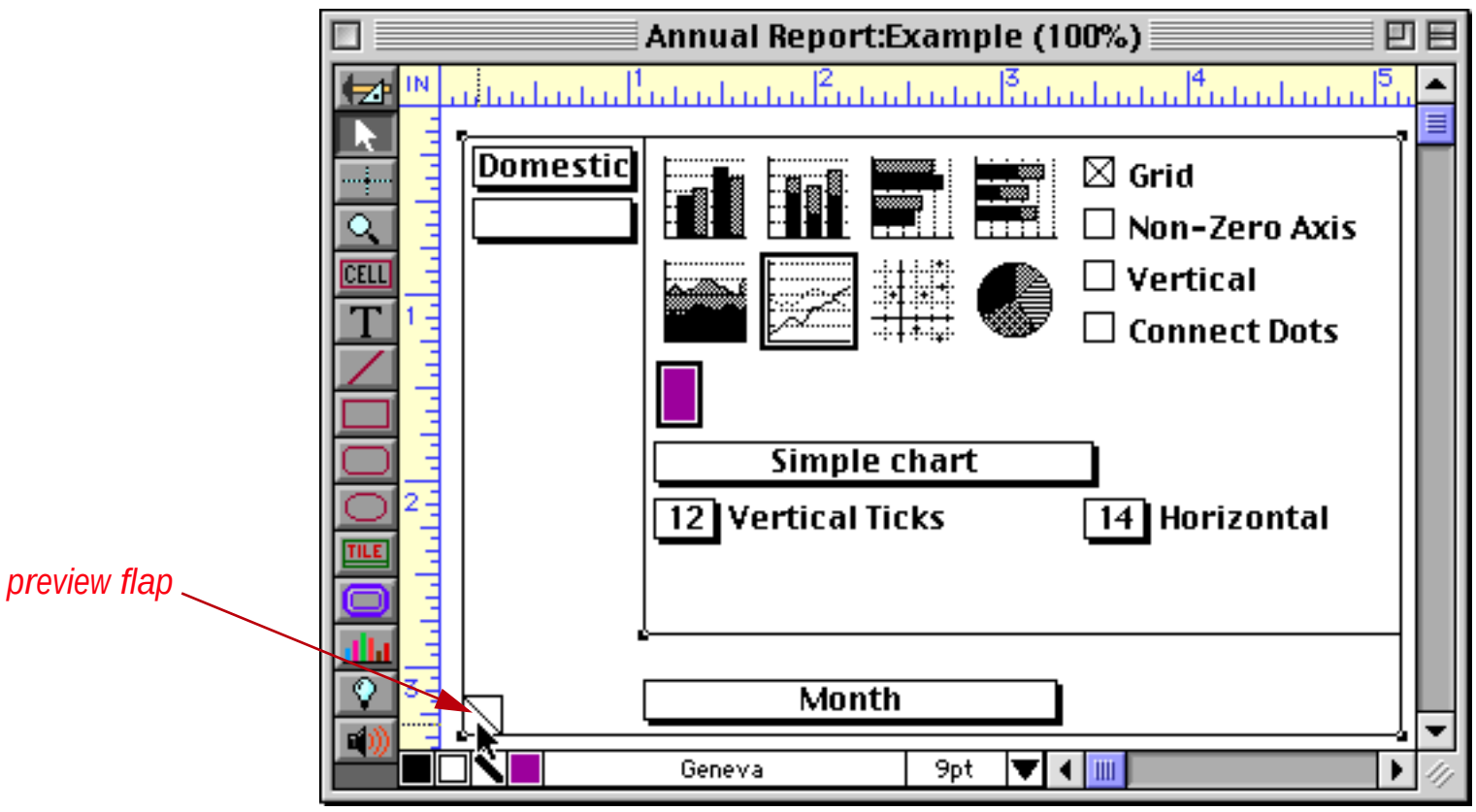
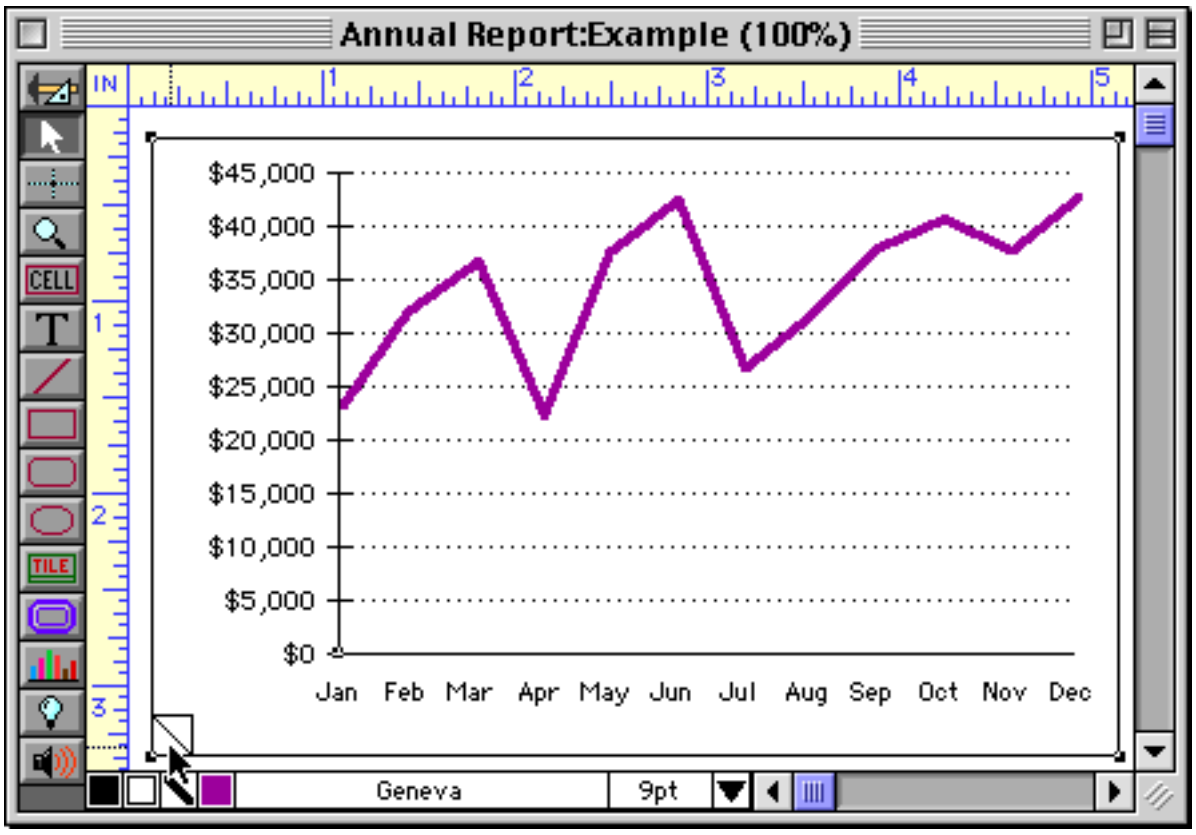


Chart Preview

You can see the effect of a change in chart setup by switching from Graphic Mode to Data Access Mode. You can also preview the chart by clicking on the preview flap in the lower left hand corner of the chart object. The preview flap is the small triangle that looks like a turned up page corner.

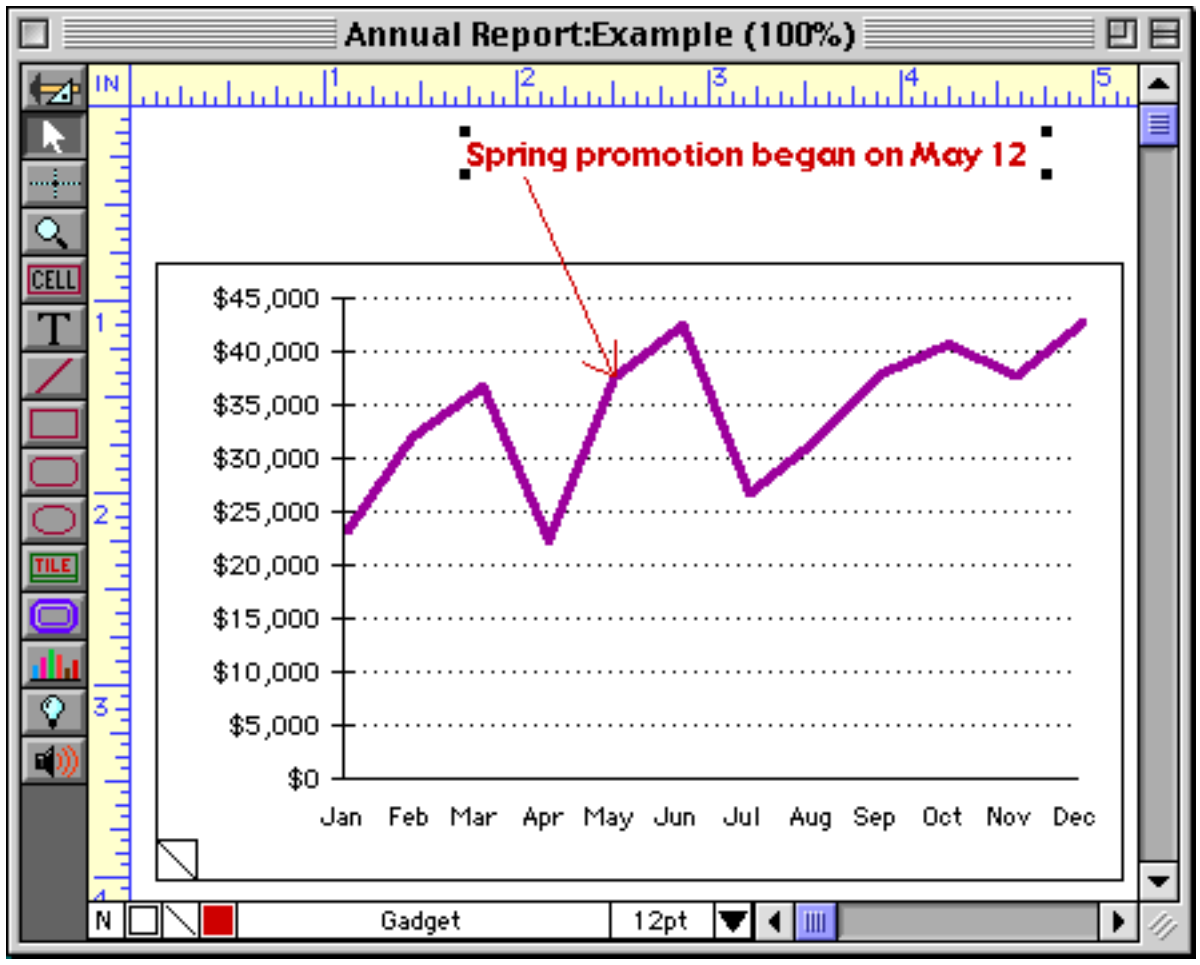


Click the preview flap to flip to a preview of the chart.



Click the preview flap again to flip back to the chart dialog. You can flip back and forth as many times as you like.

Chart preview is especially handy when you want to draw graphics on the chart itself. For example, you may want to draw a line to a data point to highlight the point. Chart preview lets you see the chart while you draw objects on top of it. Of course if the data changes, these objects may no longer be in the correct position!



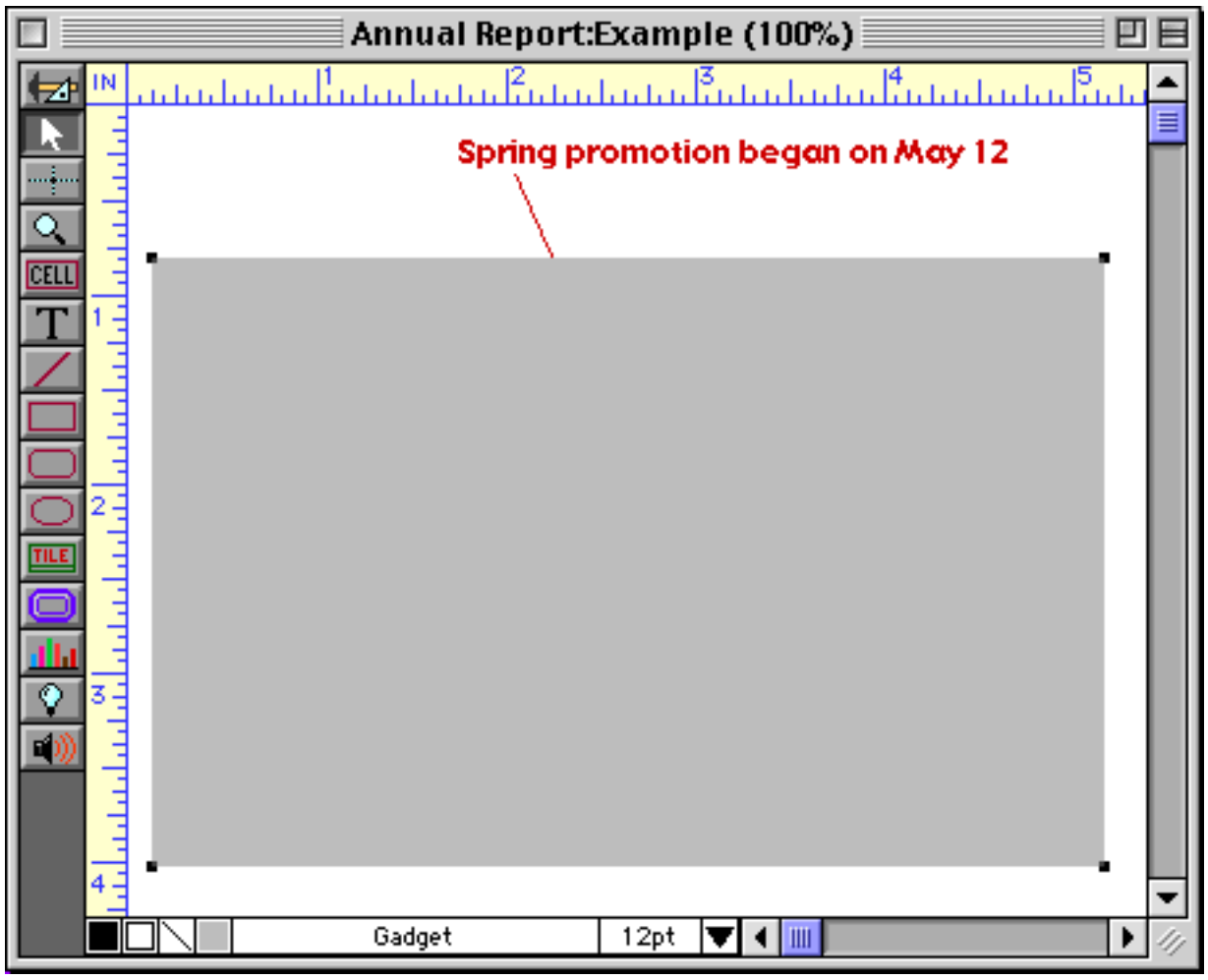
Copying a Chart to Another Application

Chart preview is also useful for copying a chart to a separate graphics or page layout program. To copy a chart to the clipboard, first preview the chart, then select the chart object and use the Copy command. You can then paste the chart into another program (as a picture).

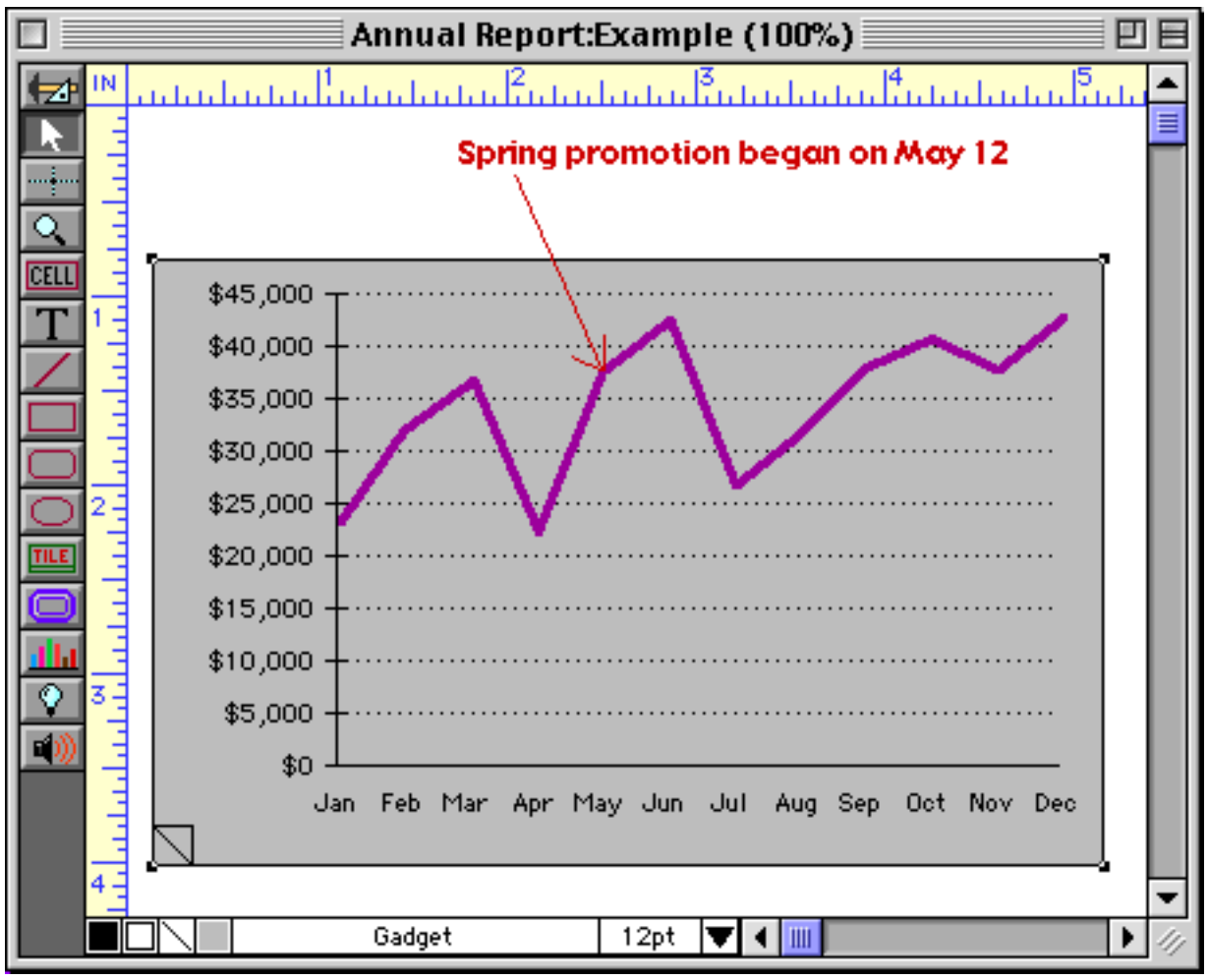
Graphic Embellishments (Titles, Legends, Drop Shadows, etc.)

Since a chart object is part of a form, you can use Panorama's graphic tools to spruce up the form any way you wish. You can add a title to a form with the Text tool. Since charts are transparent, you can add a backdrop by putting a gray or colored rectangle behind the chart. You can even create a complete legend for the chart showing which pattern or color corresponds to which data value.

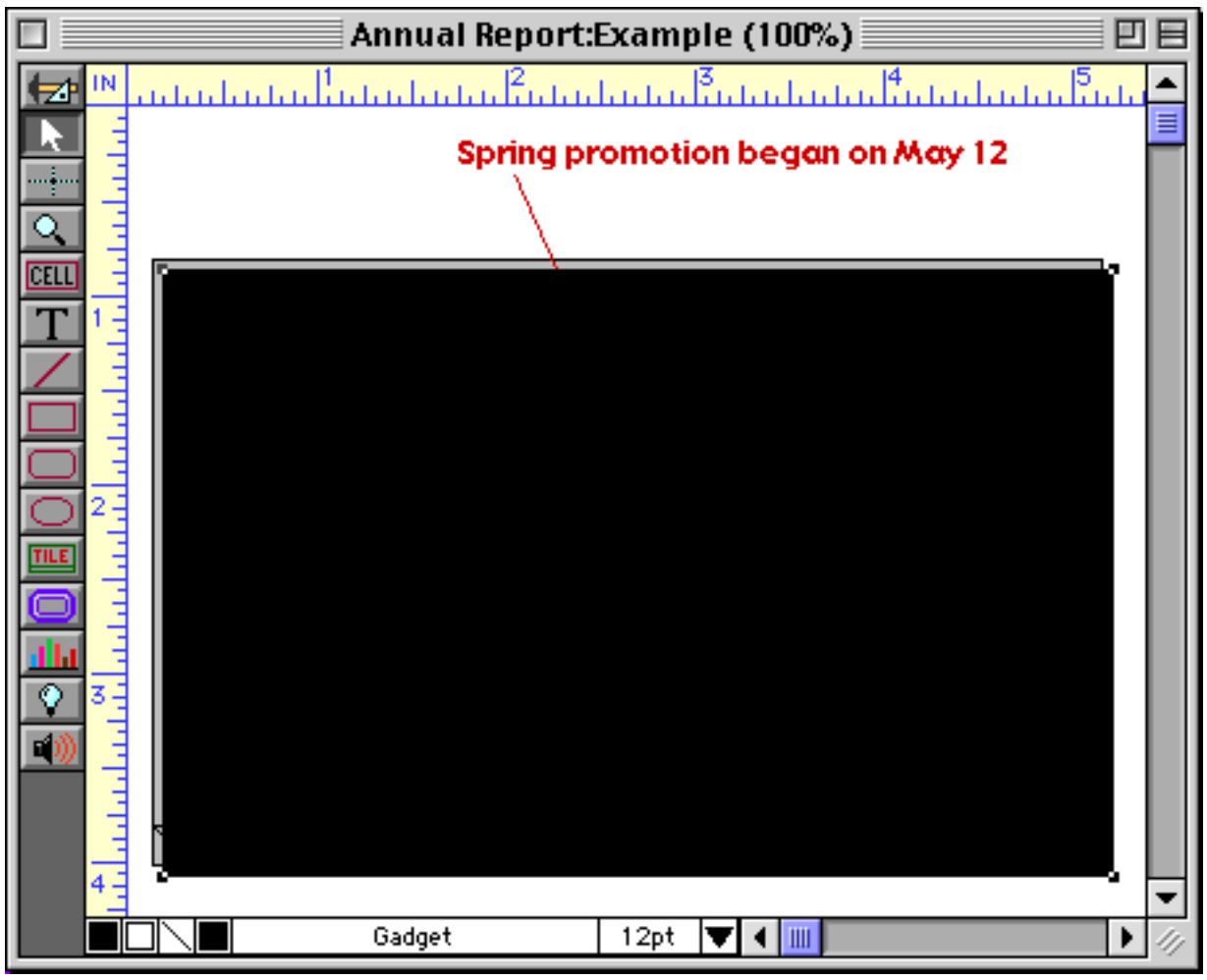
Adding a drop shadow to a chart requires two rectangles. First create a white or light colored rectangle over the chart (see "Creating a Graphic Object" on page 494).



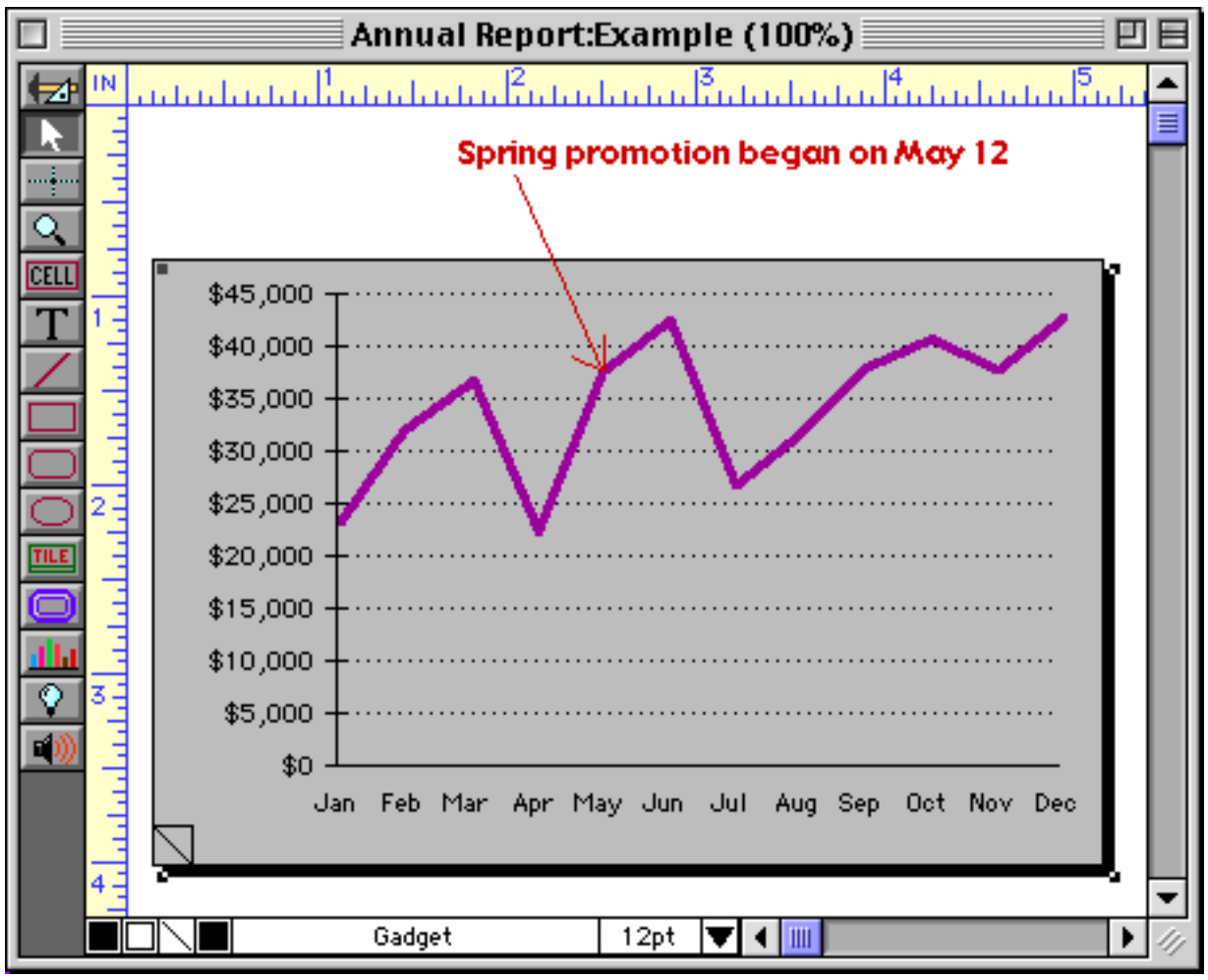
Use **Send to Back** to move the rectangle behind the chart (see "Changing the Stacking Order" on page 569).



Next, duplicate this rectangle (see "Duplicate" on page 561) and change it to a dark color (see "Color" on page 526).



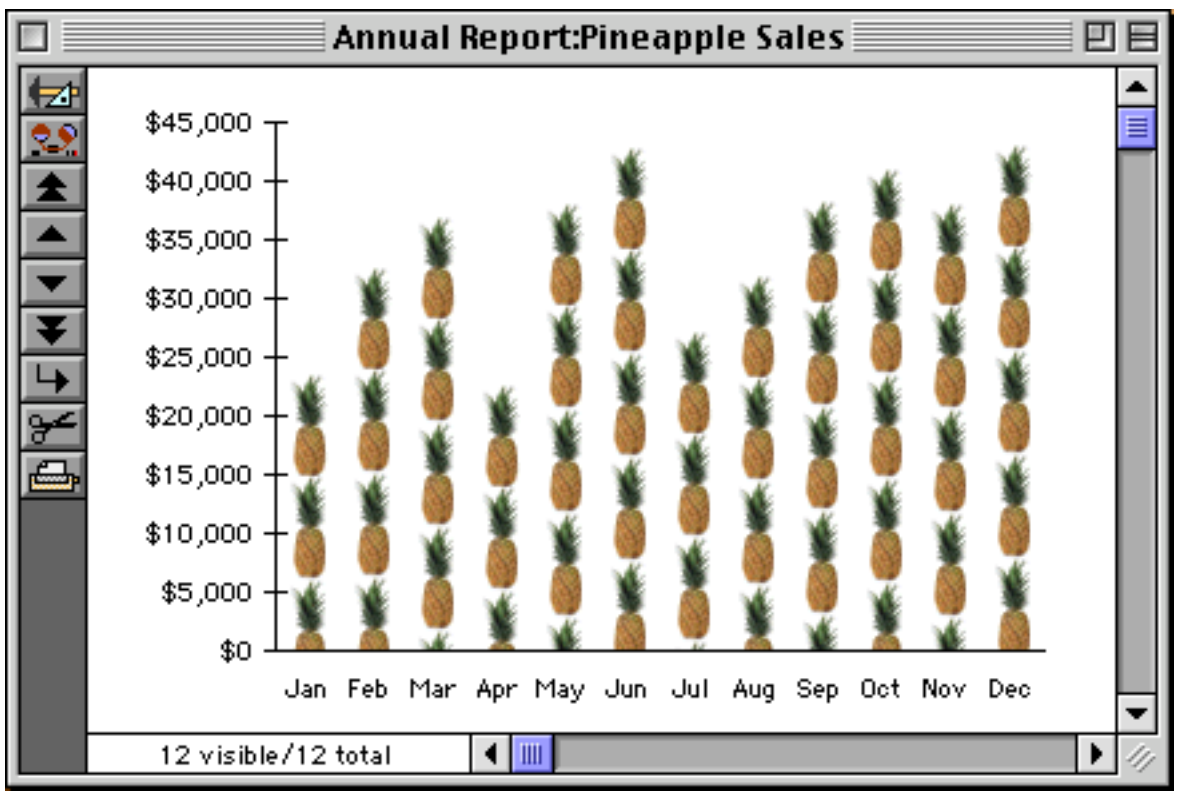
Use **Send to Back** to move this new rectangle behind everything else. Voila! A drop shadow!



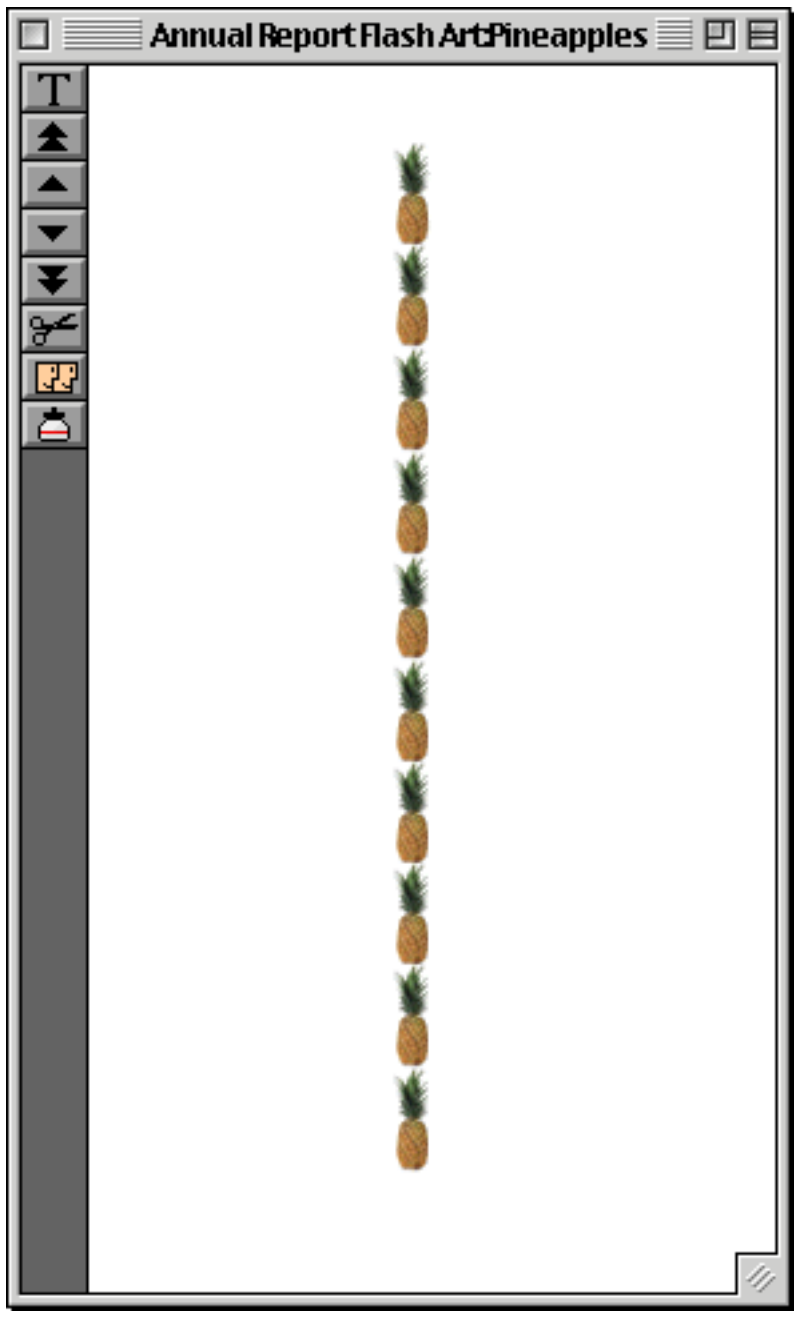
If necessary, move the shadow into position behind the chart, using the arrow keys if necessary for fine adjustments (see "Nudging an Object (or Objects)" on page 509). (The white or light colored rectangle is needed because the chart is transparent. This rectangle blocks the shadow rectangle below it.)

Chart Flash Art

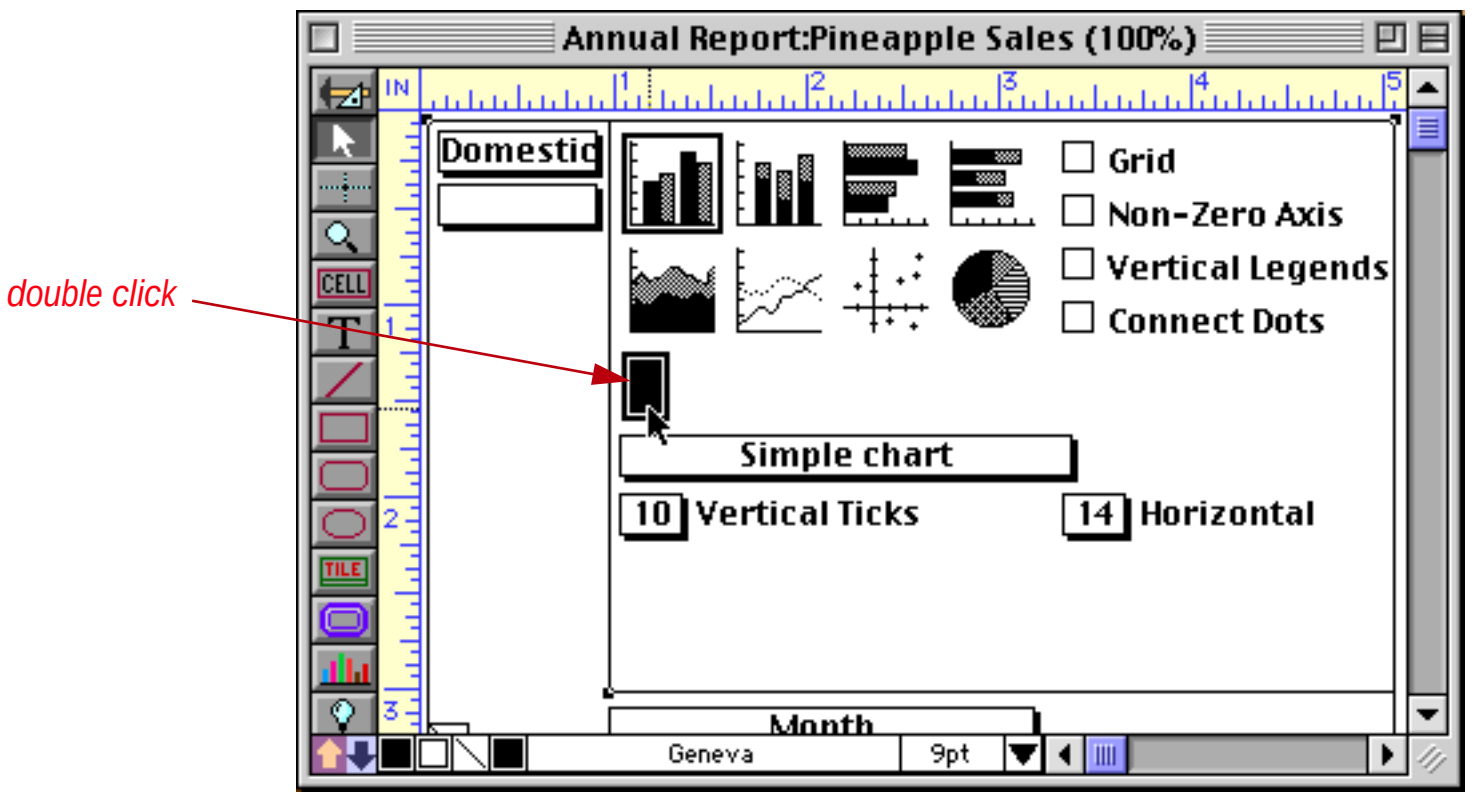
Charts are usually displayed using solid patterns and colors. With a little extra effort, you can create a truly unusual charts by using Flash Art to draw the values in a bar, area, or scatter chart.

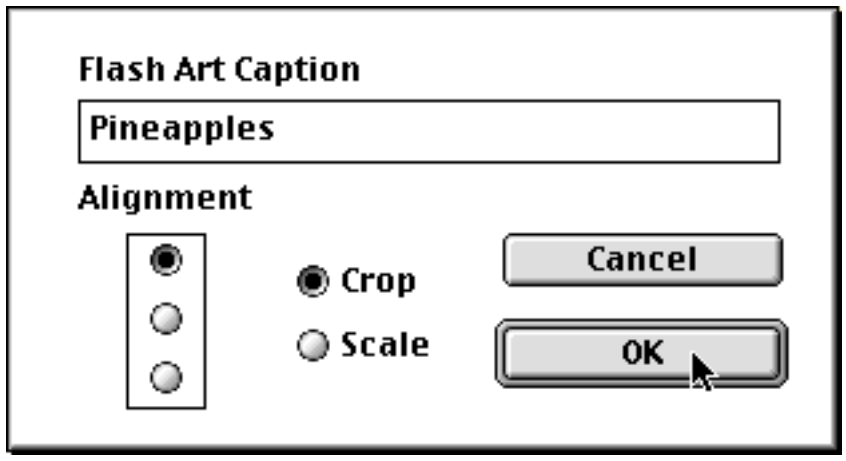


Before you can use Flash Art in a chart, you must build up a scrapbook of one or more Flash Art pictures. Each Flash Art picture is identified by a name. For instructions on building a Flash Art Scrapbook, see “The Flash Art Scrapbook (Gallery)” on page 762. The image used for the chart above was created in Adobe Photoshop and consists of a series of pineapples stacked on top of each other.



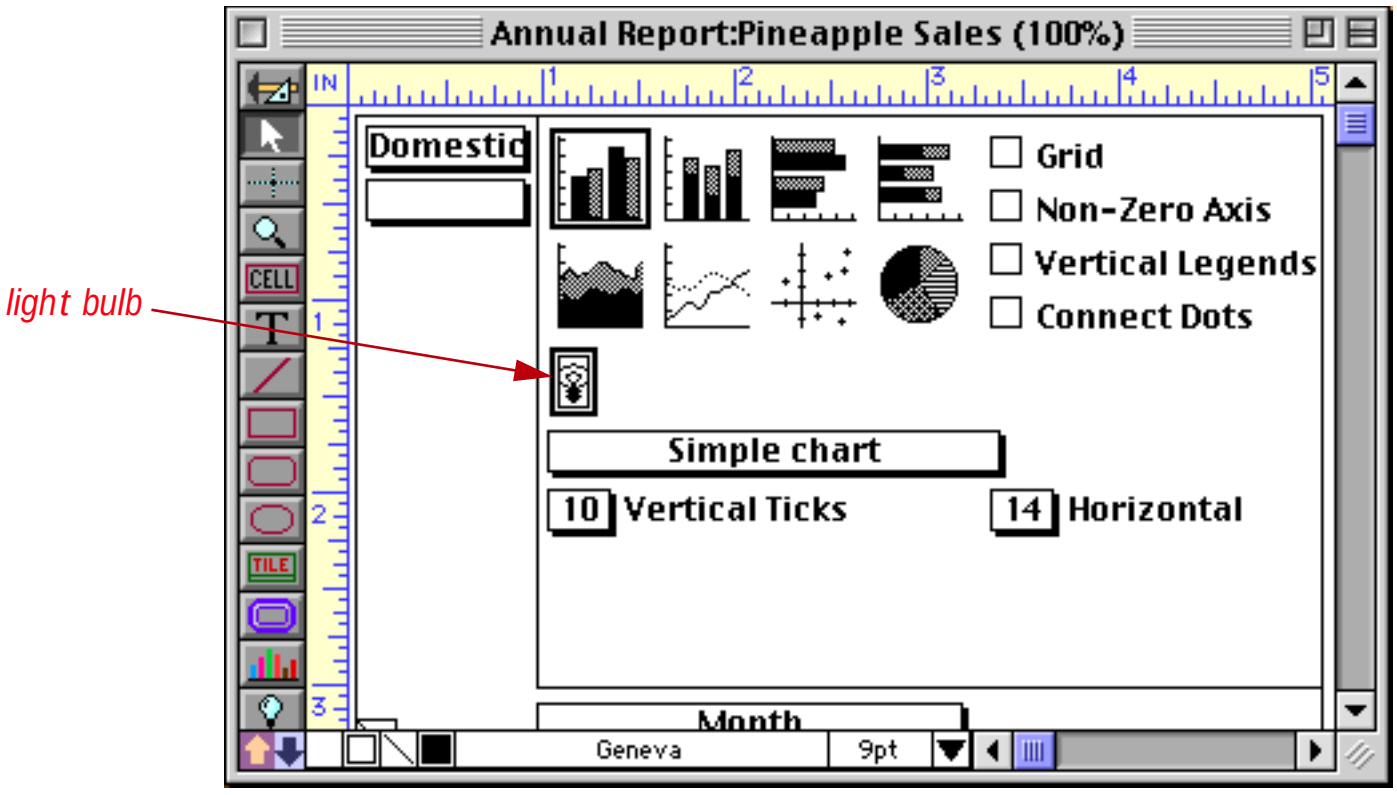
To use Flash Art instead of a standard pattern or color, double click on one of the graphic attribute icons (see “Graphic Attribute Icons” on page 1007).



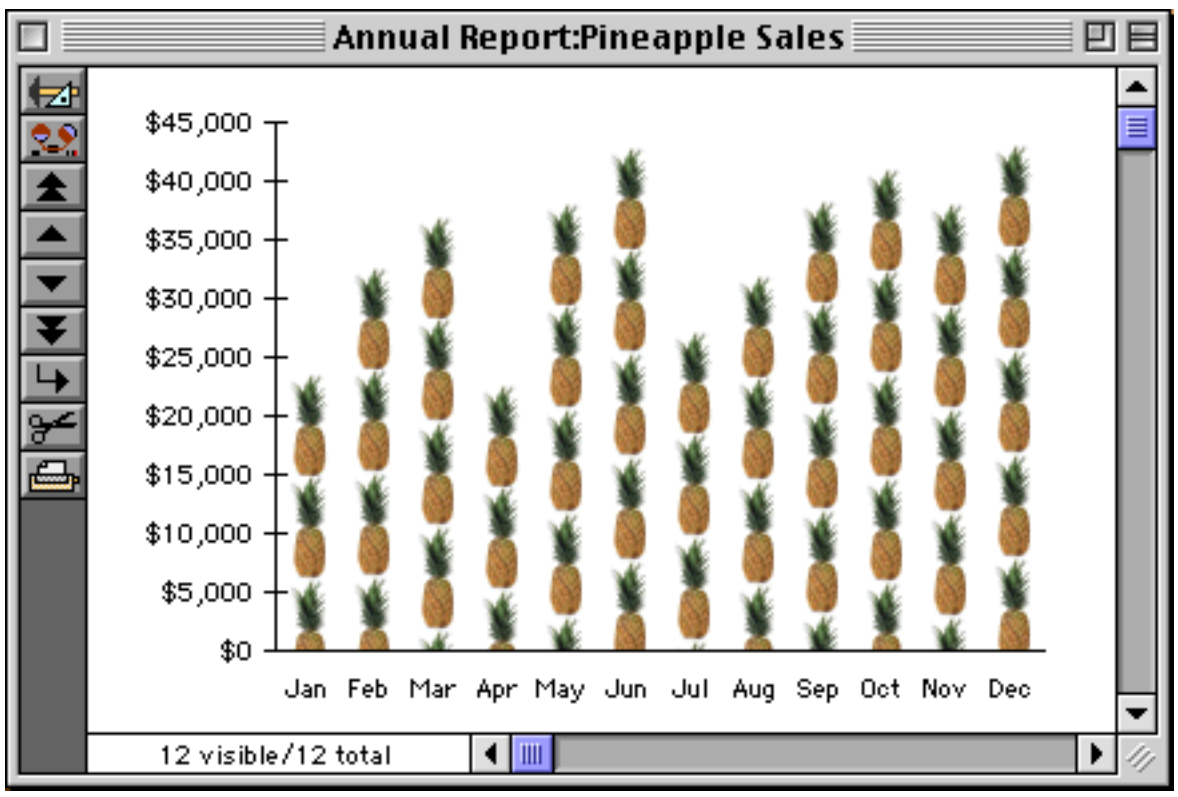


Double clicking on a graphic attribute icon opens a dialog for setting up the Flash Art options.Type in the name of the Flash Art picture you want to use (up to 28 characters).

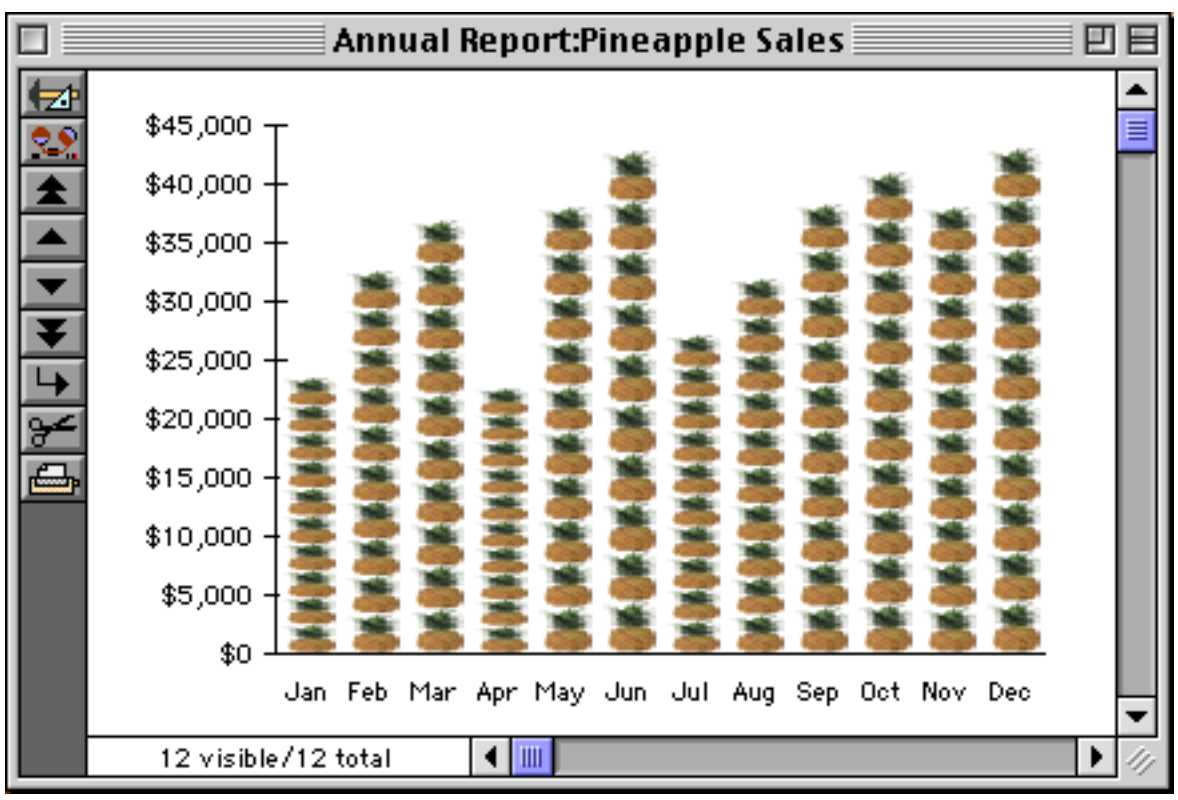
When you click **OK**, the icon will display a tiny flash art light bulb.



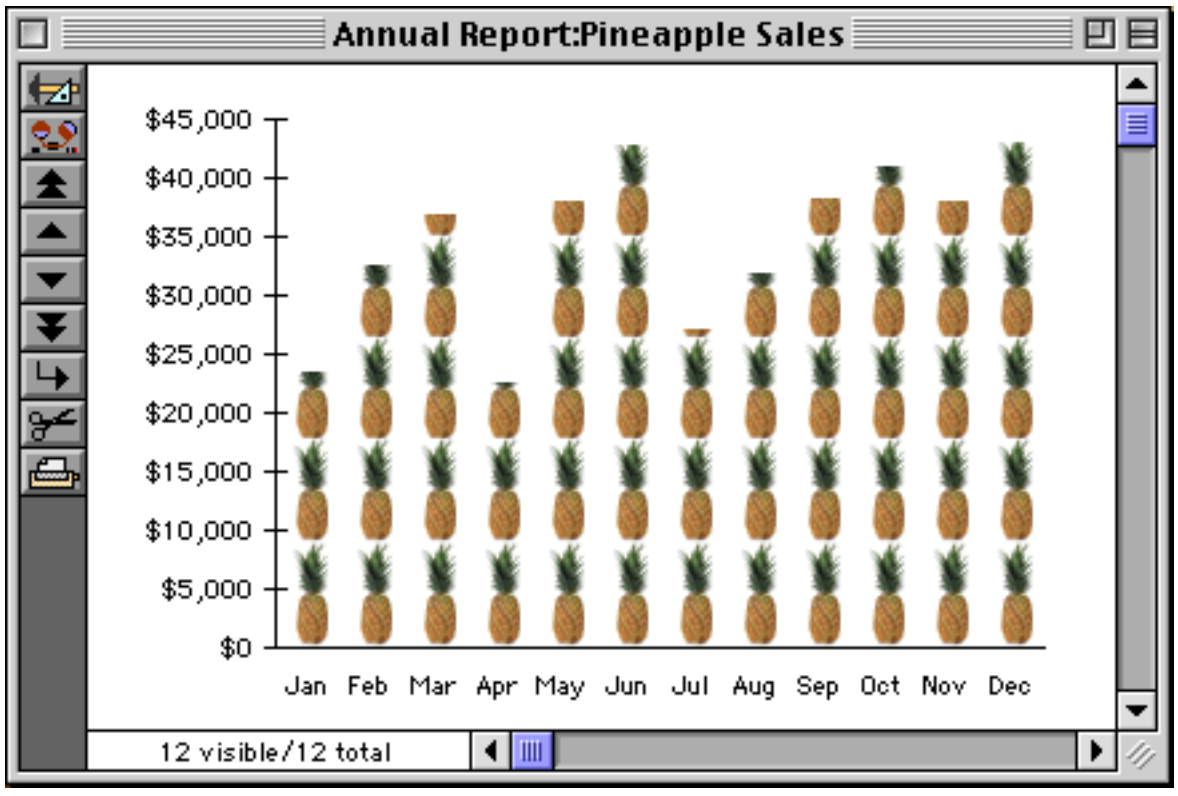
Switch to Data Access Mode (or preview) to see the Flash Art picture in the chart.



The Flash Art dialog allows you to specify whether you want to scale or crop the picture to fit in each bar of a bar chart. Here's an example of the same chart using the Scale option.



If the picture is cropped, you can specify whether it should be cropped from the center or from either end. Here's an example that is cropped from the bottom instead of the top. Notice how the tops of the pineapples are cut off.

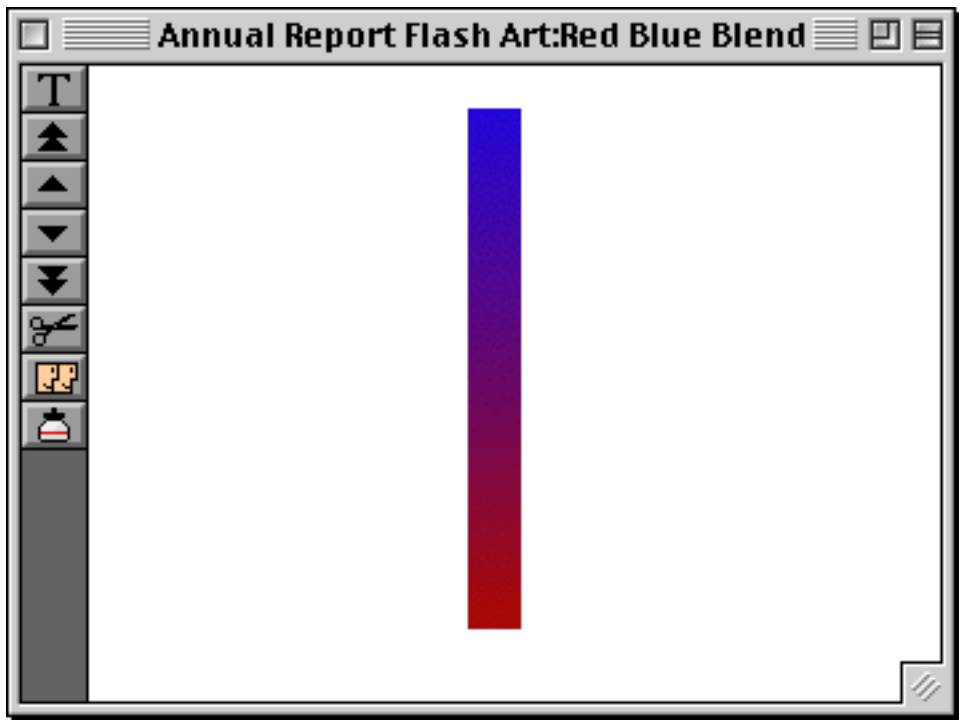


Flash Art cannot be used in line or pie charts.

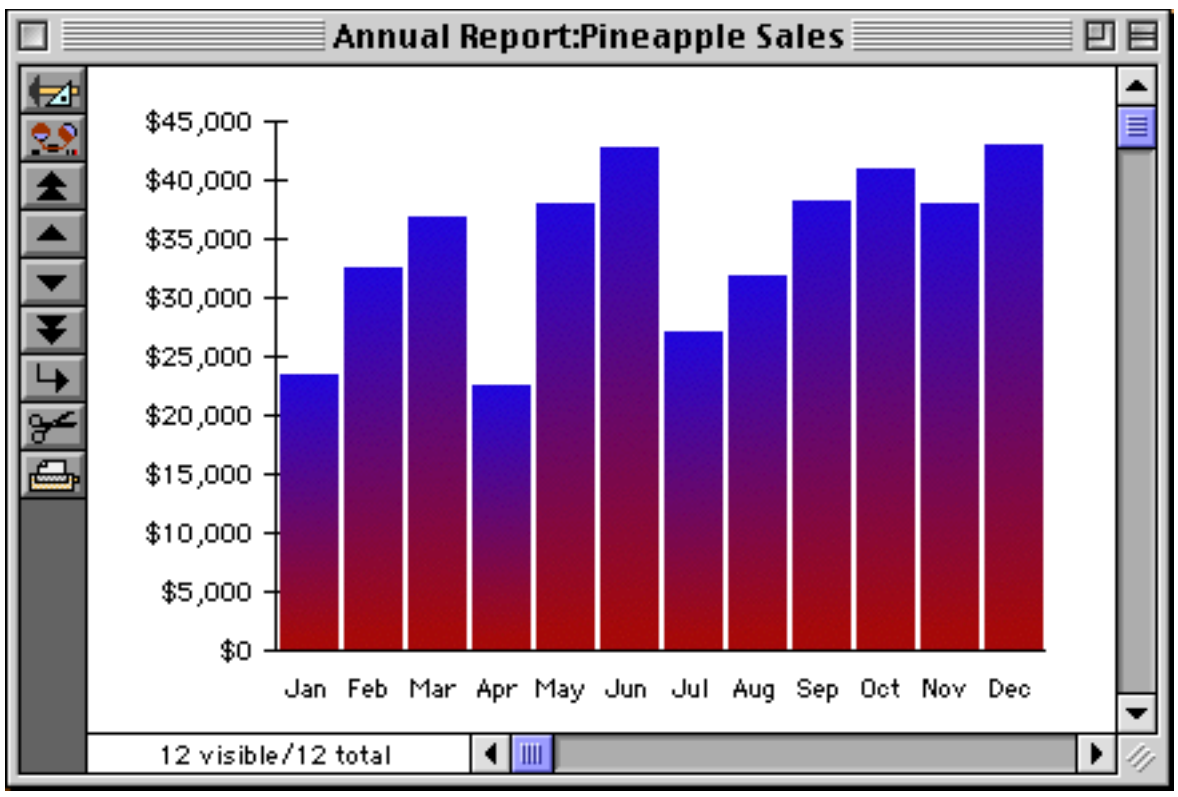
Using Flash Art for Color or Blends

Beyond the obvious application of pict-o-grams, Flash Art can also be used to display charts using graphic effects that cannot normally be created with Panorama. For example, Panorama is normally limited to 256 colors, but using Flash Art you can use any color available on your system. Simply create a swatch of the color in a color paint or draw program, then copy the swatch into the Flash Art Scrapbook. Once the color swatch is in the scrapbook, it can be used in a chart.

You can also use a program like PhotoShop to create a blend, then paste the blend into the Flash Art Scrapbook.



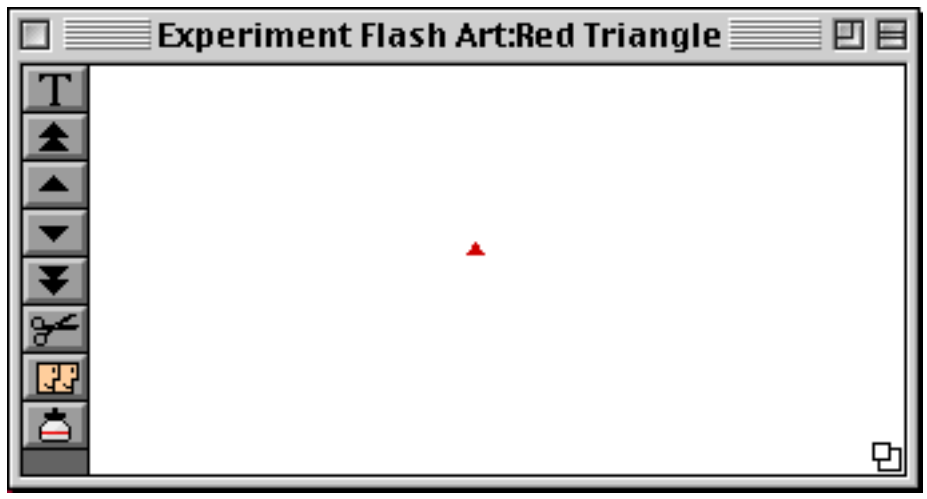
Once the blend is in the scrapbook, it can be used in a chart. This chart was created using the Scale option, but you can also get nice effects using the Crop option.



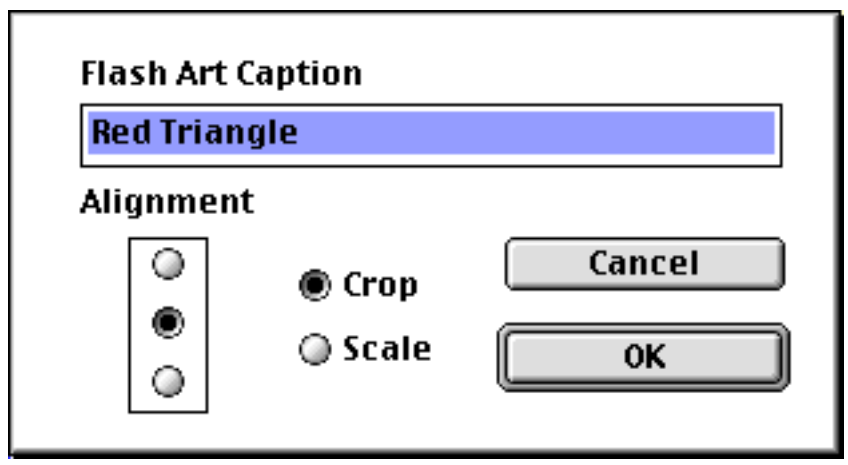
Scatter Diagram Flash Art

Scatter diagrams don't have to be just a collection of dots. You can spruce up scatter diagrams with Flash Art, and by playing connect the dots. (See "Scatter Diagrams" on page 995 for basic information about scatter diagrams.)

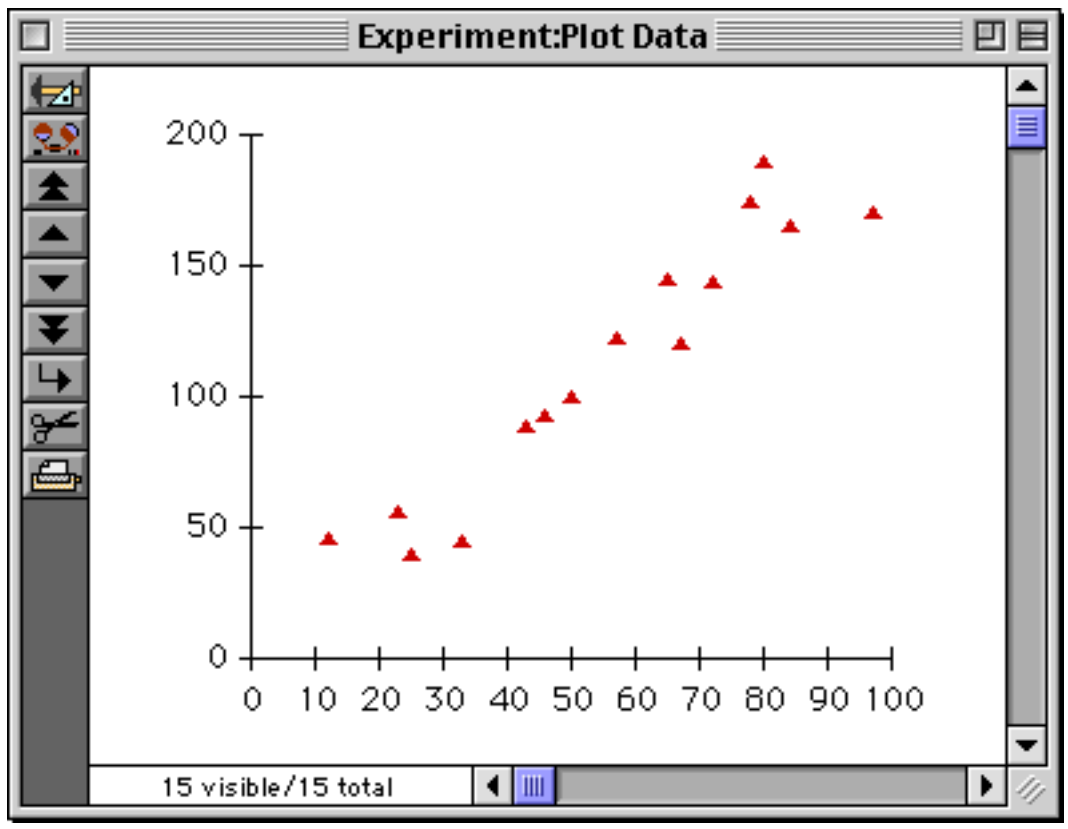
To use a Flash Art picture instead of a simple dot, start by creating an image you want to use, then paste that image into the Flash Art Scrapbook (see "Adding a New Image to the Scrapbook" on page 763).



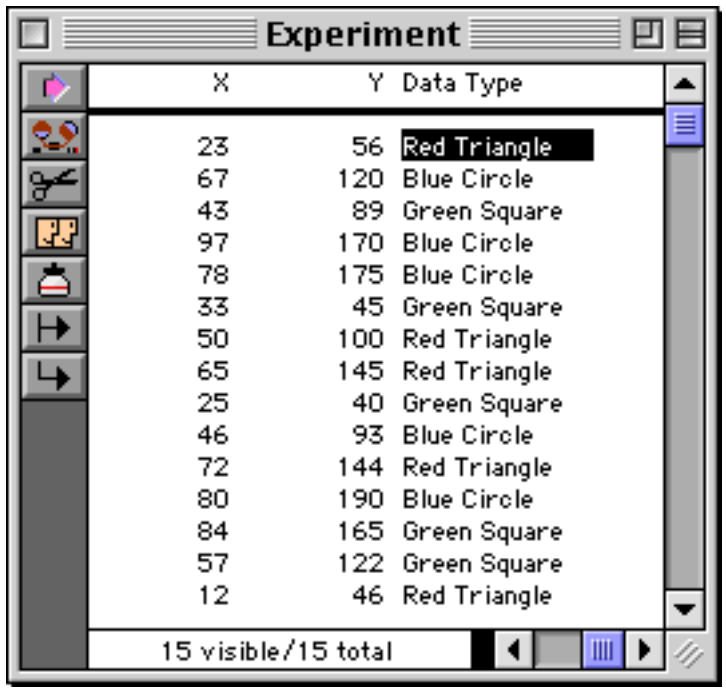
Next double click the graphic attributes icon and set up the Flash Art option (see "Chart Flash Art" on page 1022).



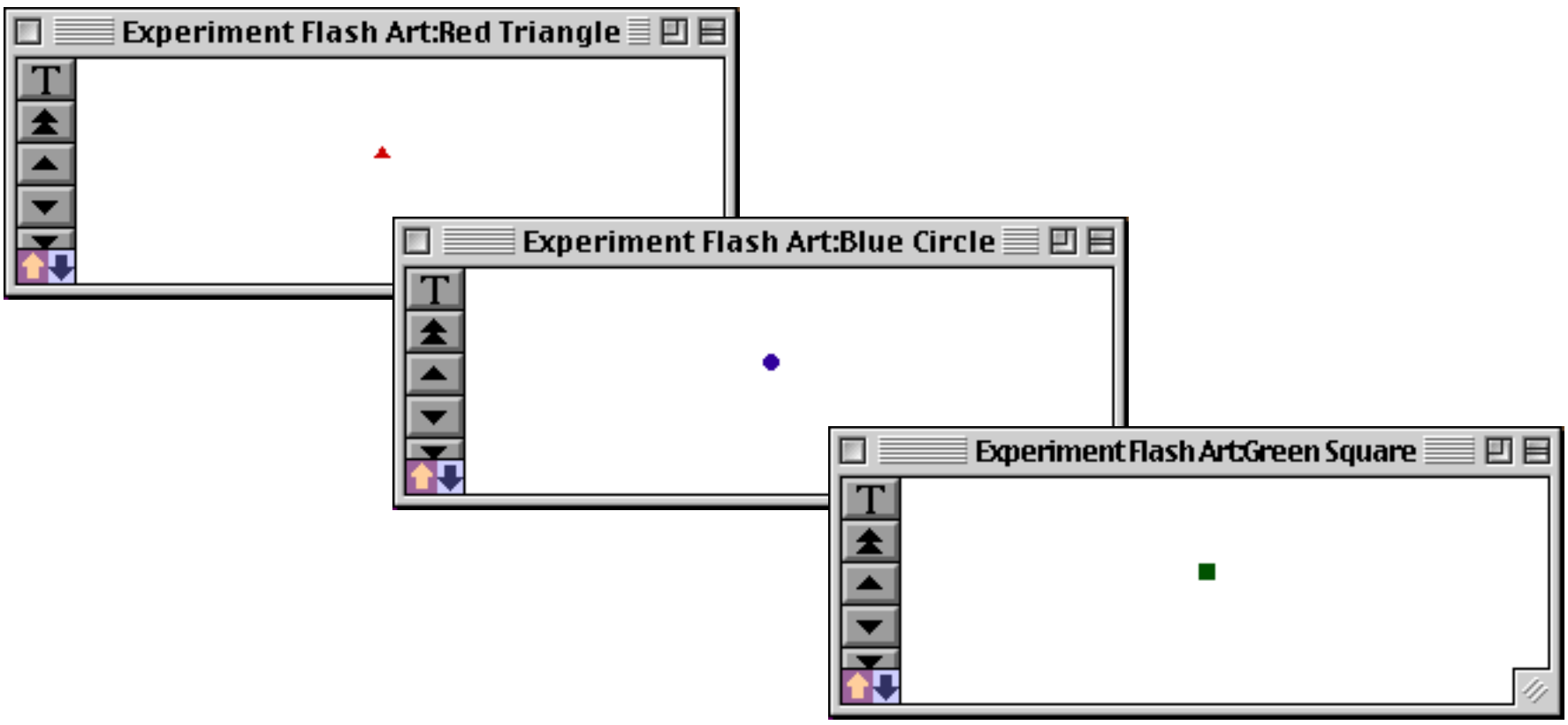
Now Panorama will draw the scatter diagram using red triangles.



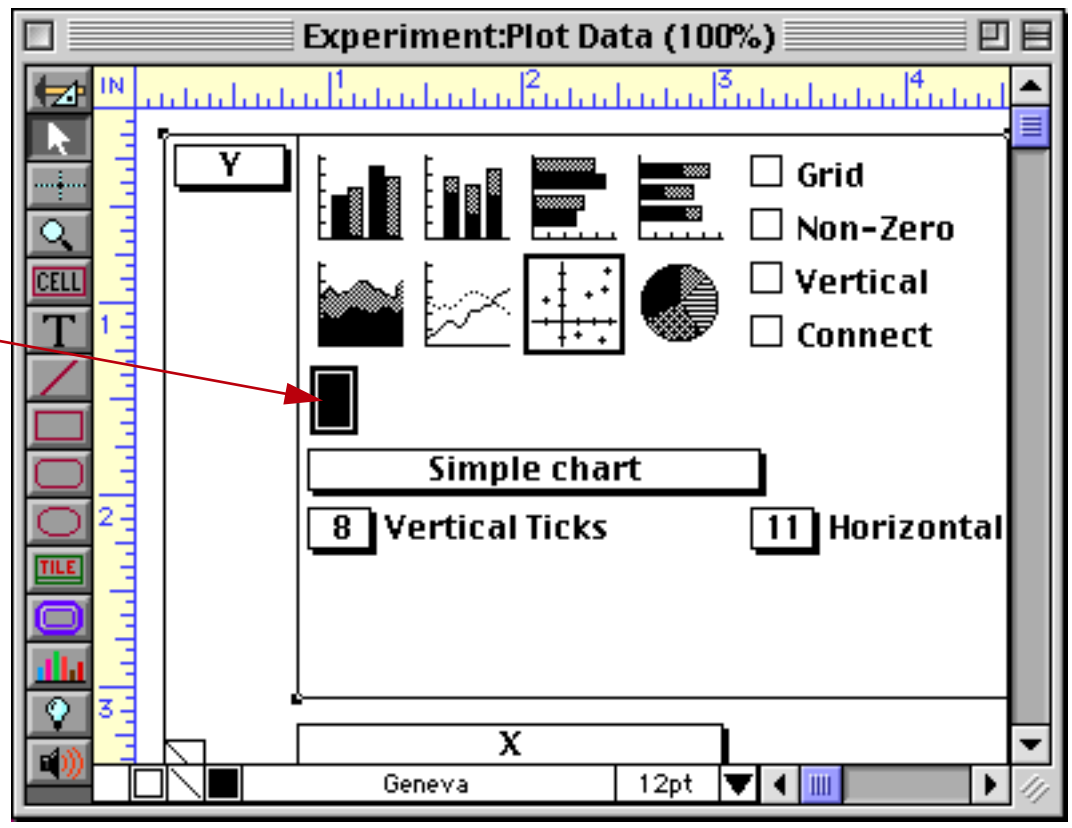
It’s possible to create a scatter diagram in which different points are represented with different images. To create a chart like this, start by adding an extra field to the database that contains the name of the image associated with each point. In this example a field named Data Type has been added (see “Add Field” on page 249).



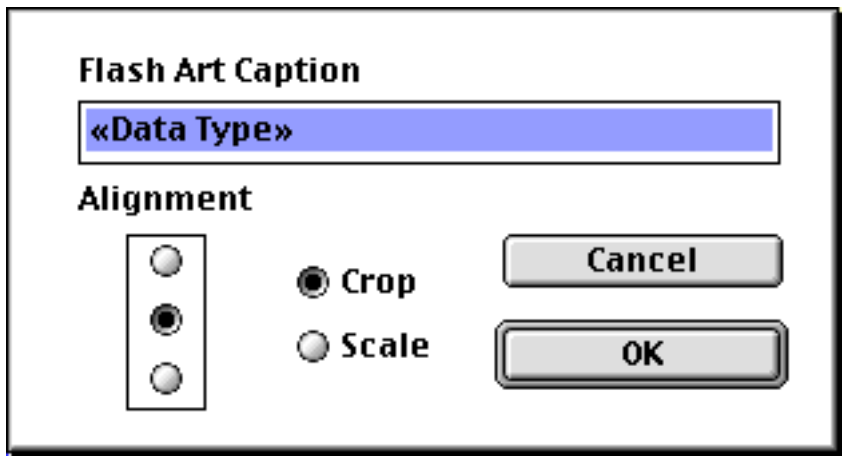
The next step is to add the images to the Flash Art Scrapbook (see "Adding a New Image to the Scrapbook" on page 763).



Next double click on the graphic attribute icon for the chart (see "Chart Flash Art" on page 1022).

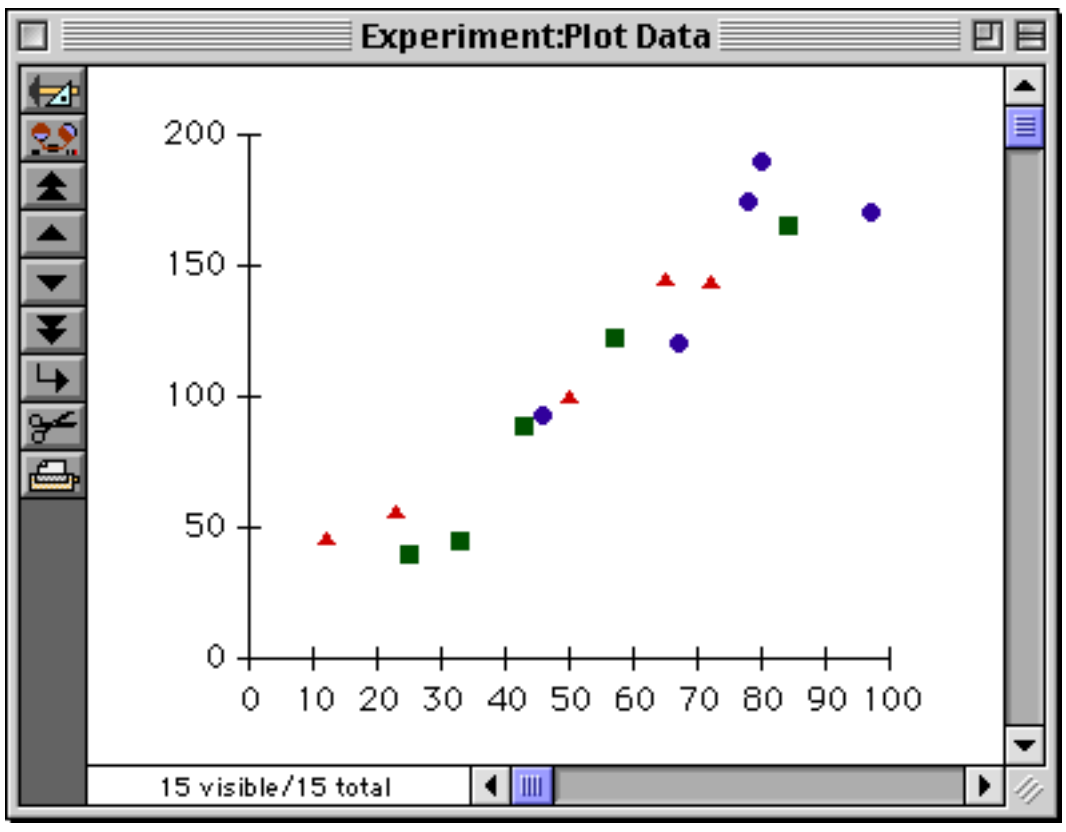


Enter the name of that field surrounded by the « » chevron characters. On the Macintosh these characters are produced by typing **Option-** and **Shift-Option-**. On PC systems these characters are produced by typing **Alt-0171** and **Alt-0187**.



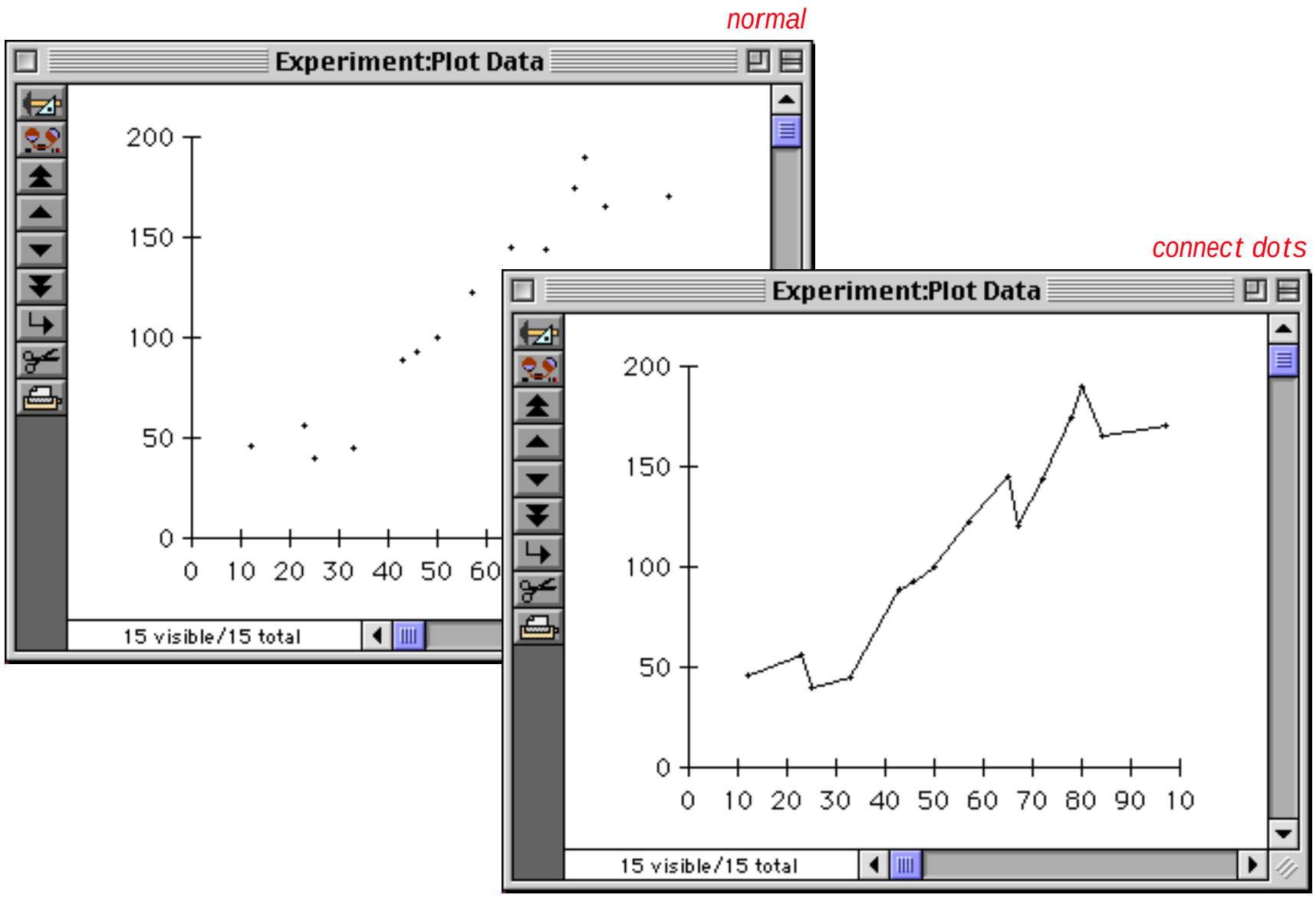
(Note: Although the example may make it appear as if you could enter an entire formula, only a field name is allowed.)

When the chart is displayed it shows the appropriate symbol for each point.

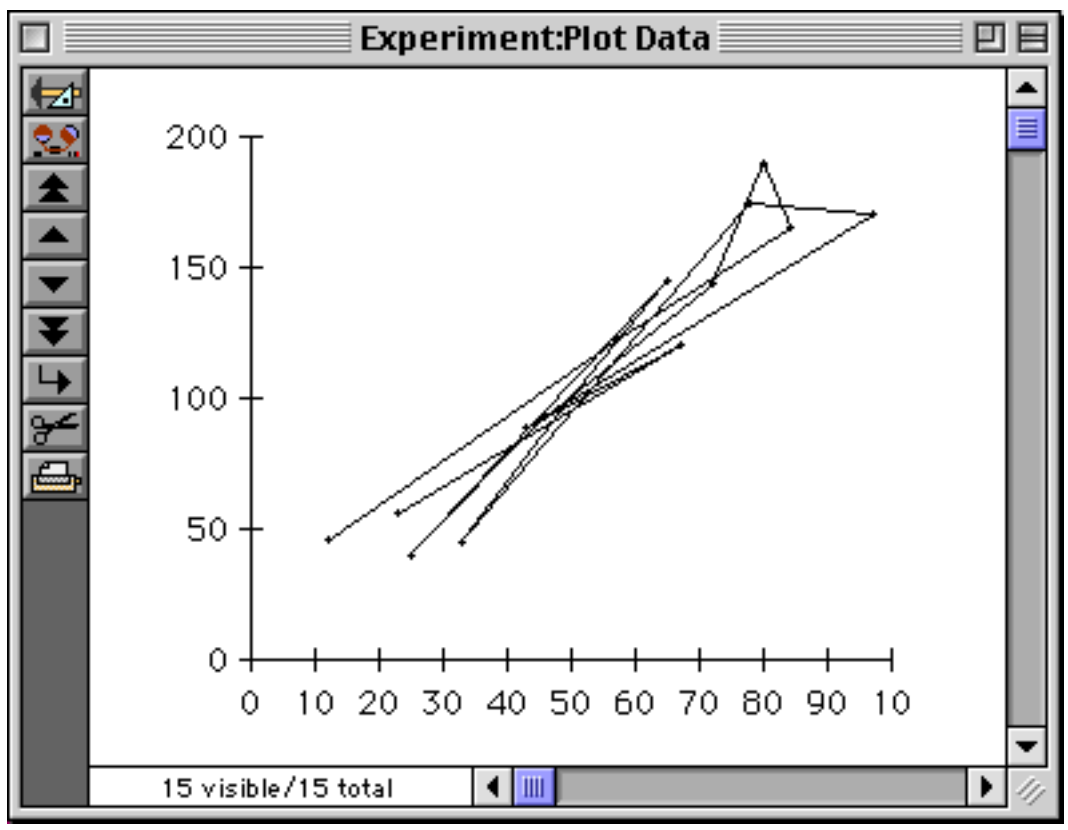


Connect Dots

To draw a line between the dots in a scatter diagram, check the Connect Dots option. The Connect Dots option only affects scatter diagrams. All other chart types ignore this option.

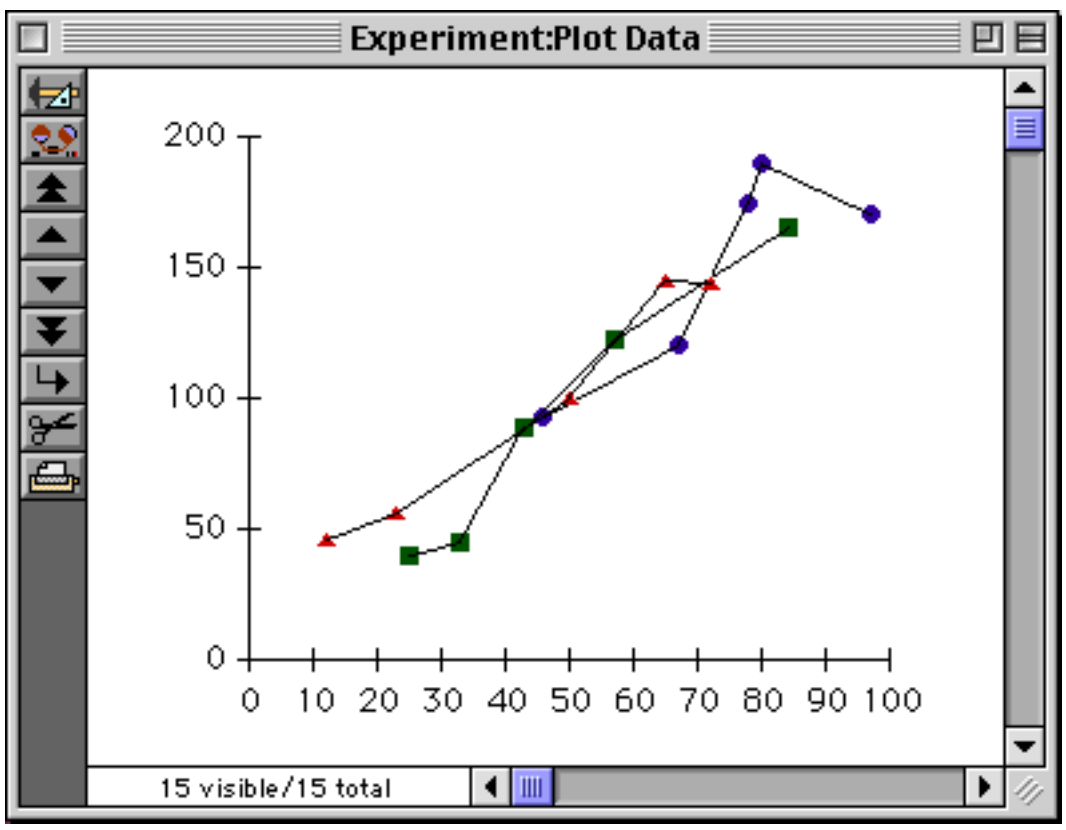


When a chart is connecting the dots, the order of each point is very important. If the points are out of order, the result can be a mess.



The points can be re-arranged into order by **Sort**ing the X value, and then using **Sort Within** on the Y value (see “Sorting By More Than One Field” on page 360). (This assumes, of course, that you don’t want the lines to be scrambled up.)

If the chart uses different Flash Art symbols for different dots (see “Scatter Diagram Flash Art” on page 1028), Panorama will only draw a line between points that are the same shape.

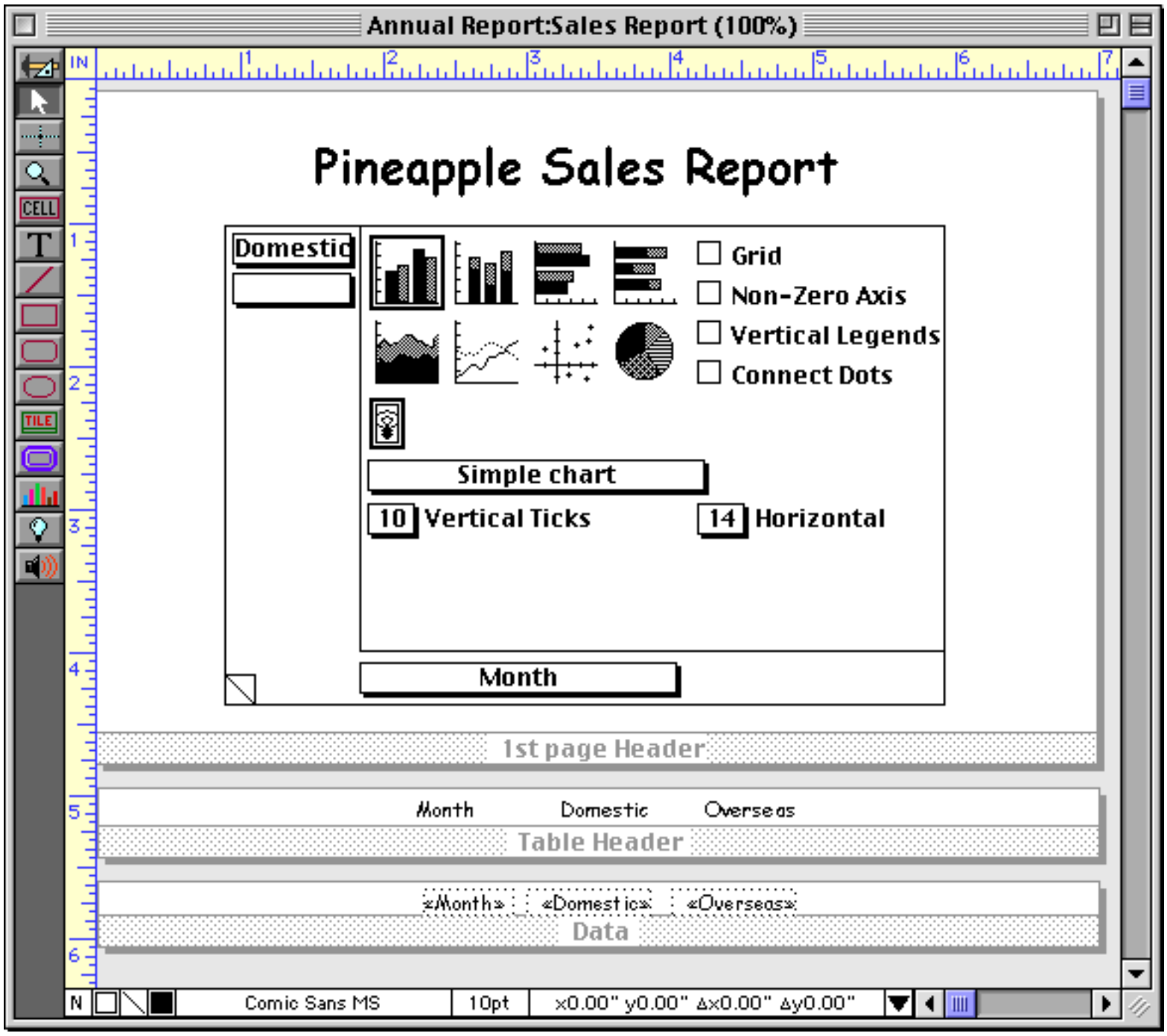


To make sure that all points of each symbol are connected together, sort the database by the type of symbol. You may also want to **Sort Within** by the X and Y values.

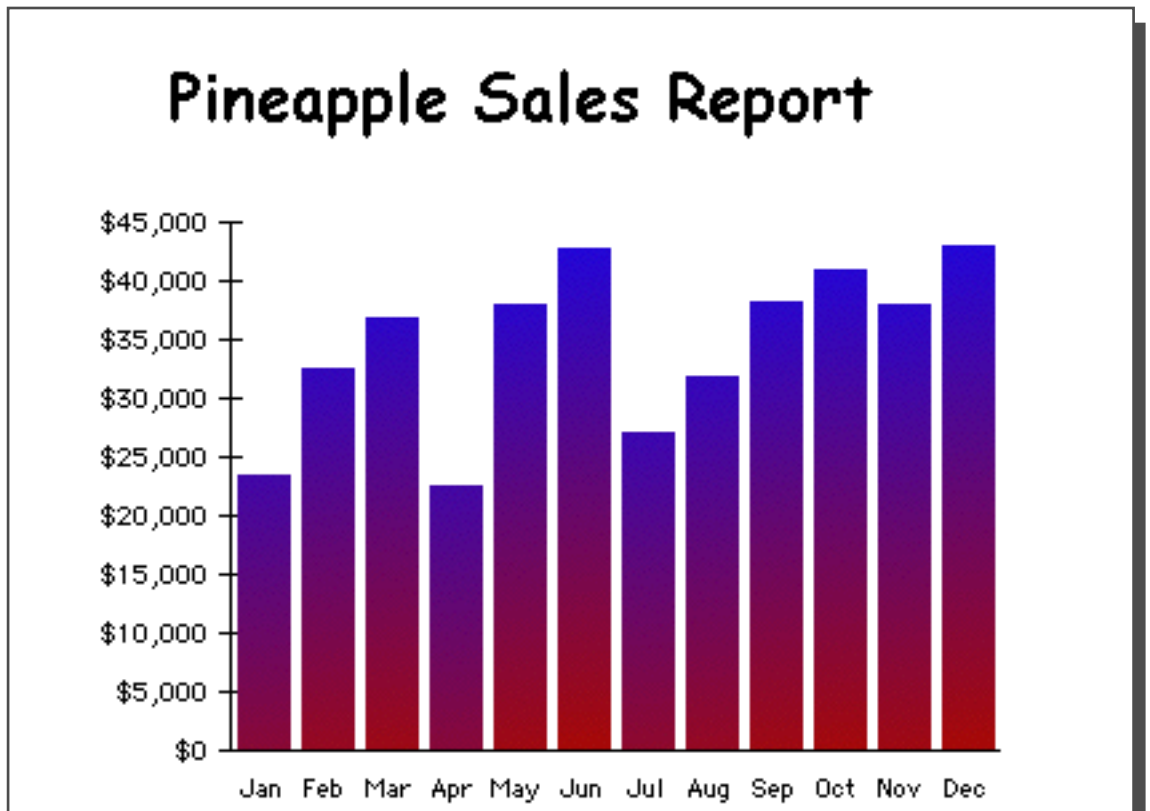
Printing a Chart

A chart can be printed just like any other graphic element. A simple way to print a chart is to place it on a data tile and use the **Print One Record** tool to print it (see “Print One Record” on page 1048). (If you use the **Print** command you will get many identical copies of the chart—one for each visible record.)

Another method is to place the chart on the **First Page Header** tile (see "First Page Header Tile" on page 1093).



This allows you to combine the chart with a listing of the data on the same page. Here's what the final printed page looks like.



Month	Domestic	Overseas
Jan	$23,424	$11,665
Feb	$32,578	$8,992
Mar	$36,891	$3,984
Apr	$22,445	$269
May	$37,891	$6,820
Jun	$42,654	$4,095
Jul	$26,974	$13,433
Aug	$31,896	$15,202
Sep	$38,221	$1,431
Oct	$40,950	$6,237
Nov	$38,012	$4,786
Dec	$42,881	$9,292

Chapter 20: Printing Basics



Since we haven't quite arrived at the age of the totally paperless office, printing is still an important function of any computer program—including Panorama. This chapter covers the basics of printing. In the next chapter you'll learn how to design and use custom reports.

Printing Different Views

You can print any of Panorama's six kinds of views—data sheet, design sheet, flash art gallery, form, crosstab, or procedure. To print a view, you must make it visible in the top window. Use the View Menu if the view you want to print is not currently visible and on top (see "Switching Between Views" on page 222).

Once the view is visible in the top window, use the **Print** command in the File Menu to print the contents of the view.

Printing the Data Sheet

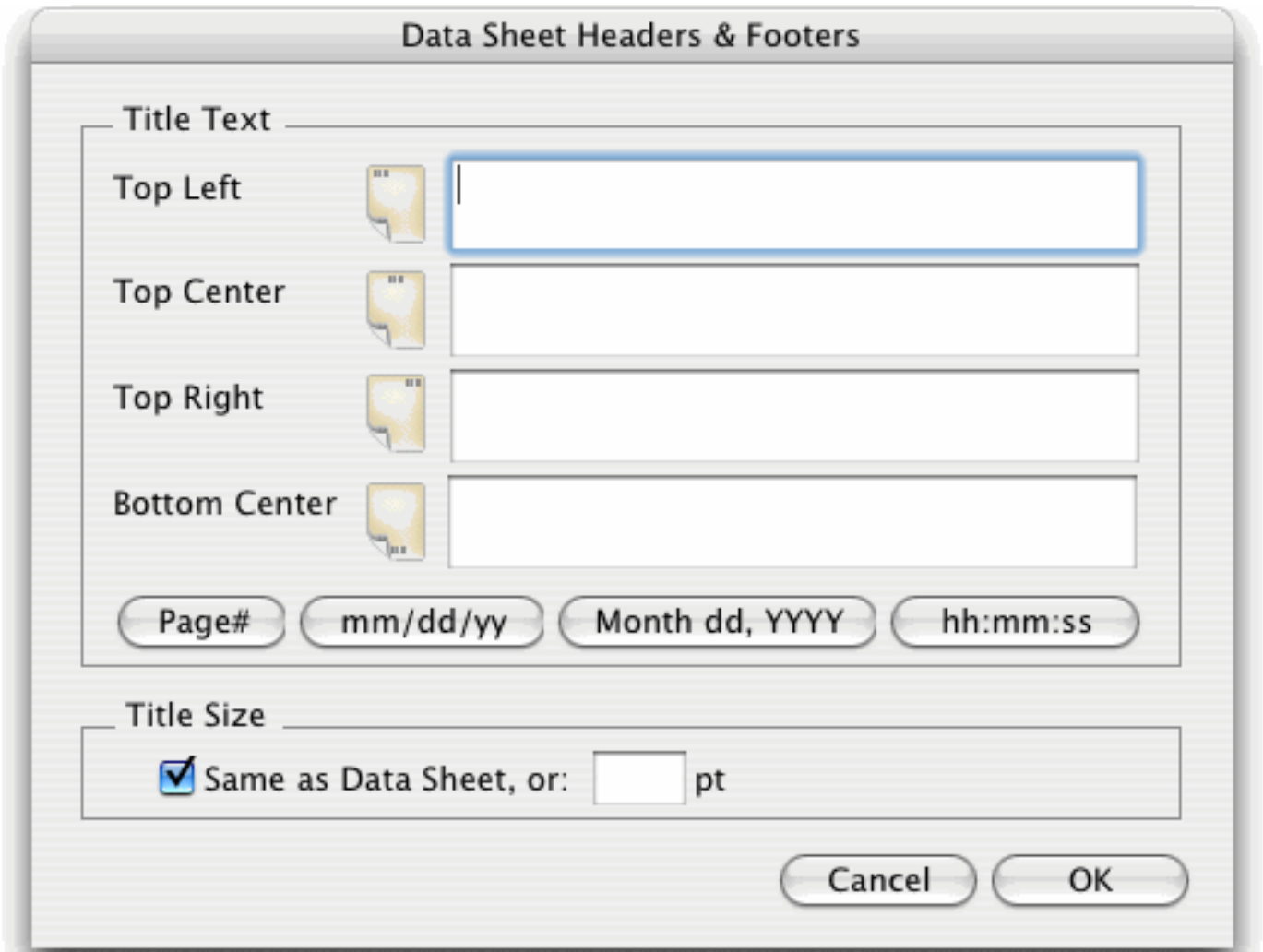
Panorama prints the data sheet exactly as it appears on the screen. If the data sheet is too wide to fit on the page, Panorama will print extra pages until all the columns are printed.

You can get more data sheet columns on a page several ways. One method is to use the **Page Setup** dialog to switch to a wide paper orientation (sideways or landscape), or to reduce the printout to a smaller size (Macintosh only). You can also use a smaller font.

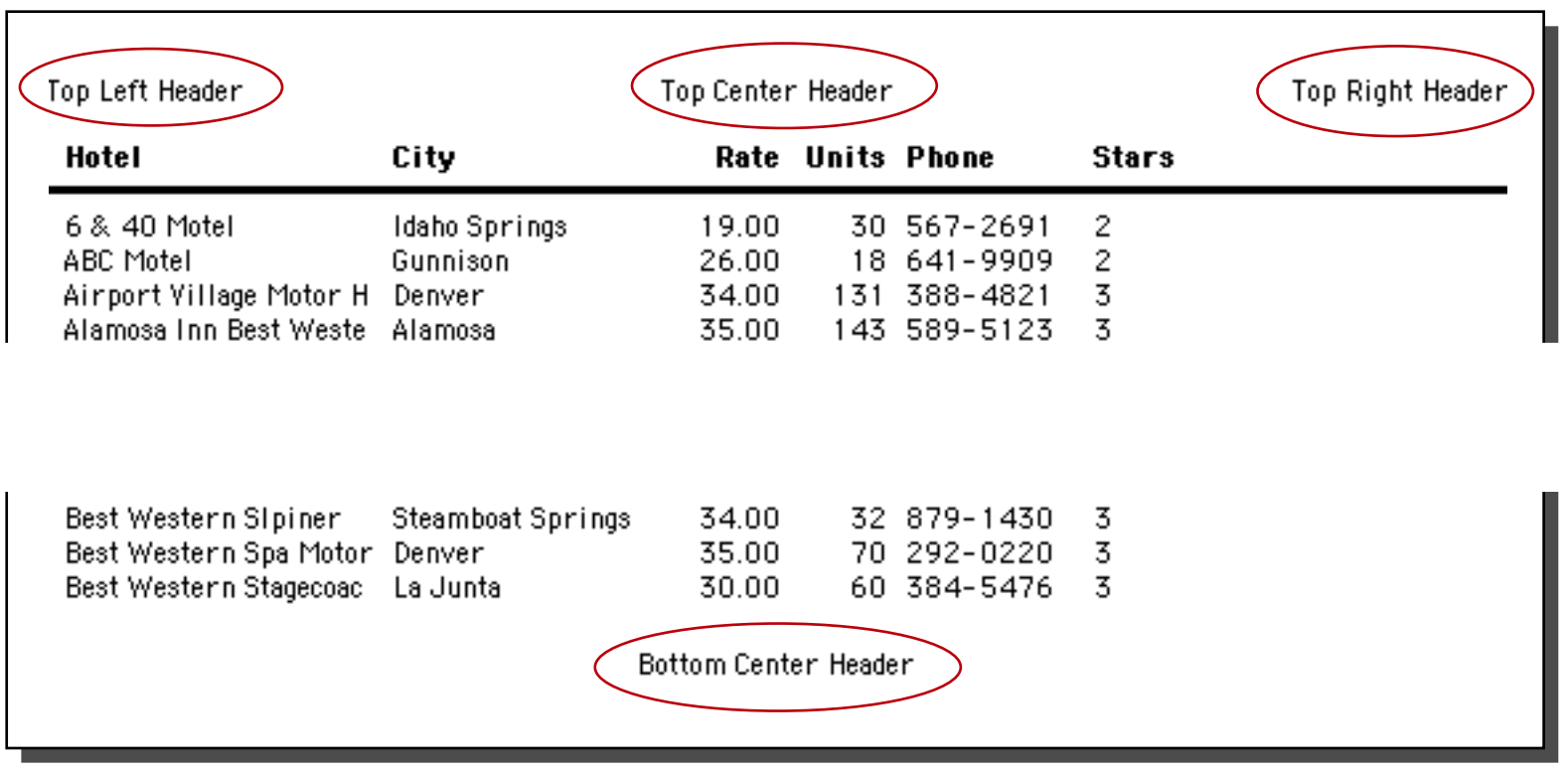
You can also print the design sheet or a crosstab sheet. These views will also print extra pages if they contain more columns than will fit on a single page.

Printing Data Sheet Headers & Footers

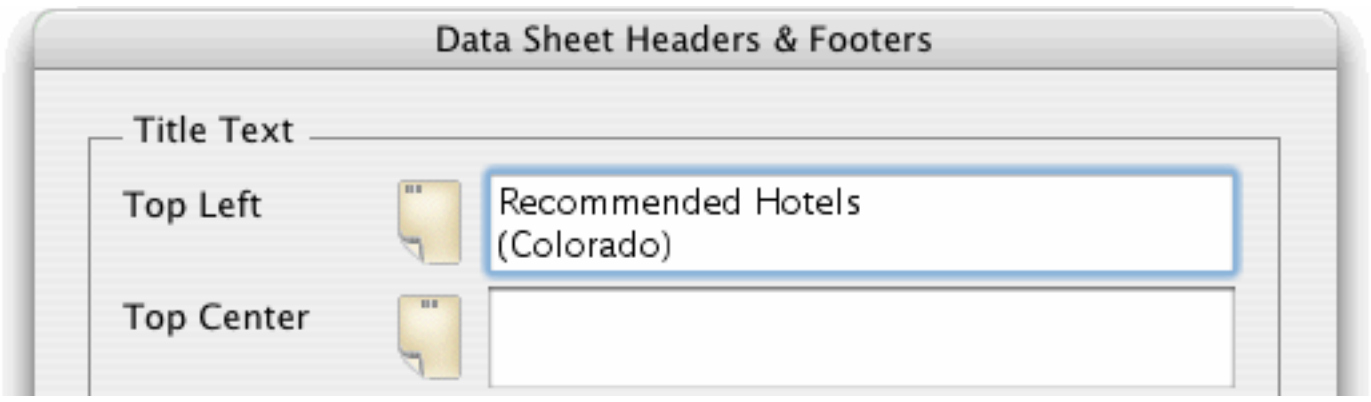
The **Headers/Footers** dialog (File Menu) sets up headers and footers for the data sheet, design sheet, or any crosstab. (You can set up separate headers and footers for each of these windows.)



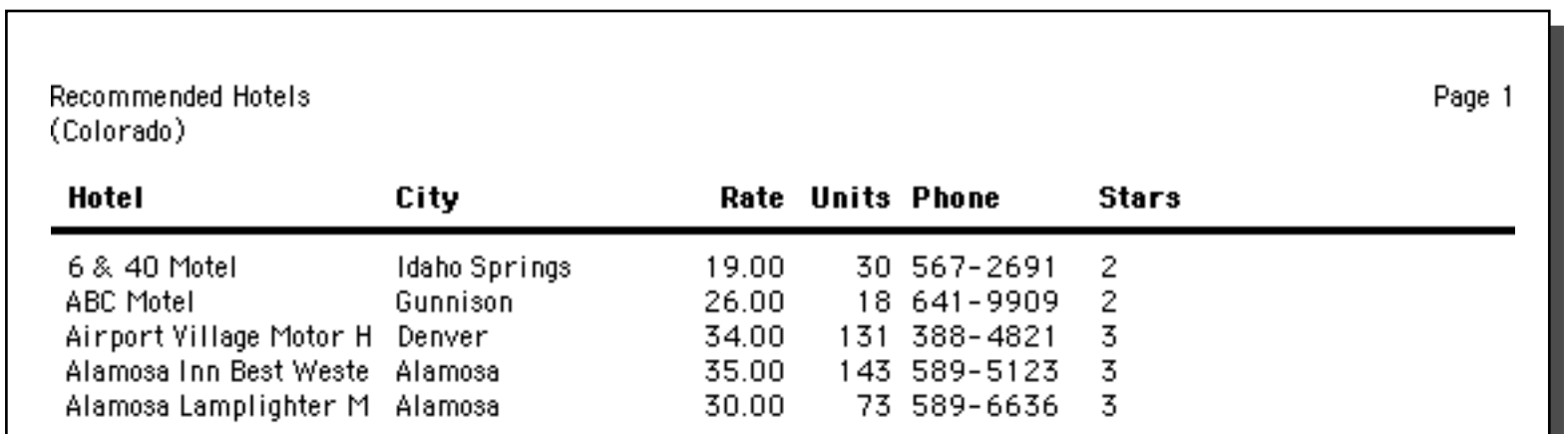
The **Headers/Footers** dialog allows you to position headers and footers in four locations on the printed page: top left, top center, top right, and bottom center.



If you want to create a header or footer that is more than one line high, just press the **Return** key and type in the additional lines, like this.



When printed this header will look like this.



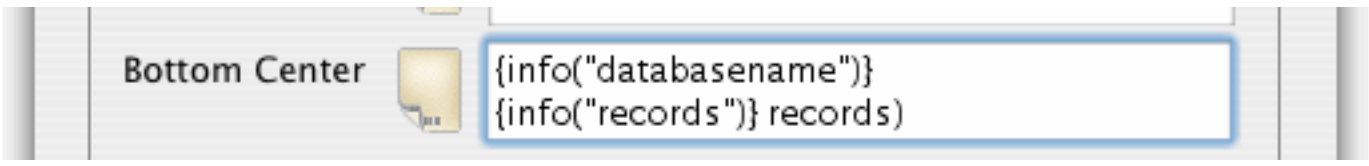
You can insert special codes into the header/footer text to print the page number, date, and time. Here's an example of how to insert the page number (the printed result is shown above). On the Macintosh the « » chevron characters are produced by typing **Option**-\ and **Shift-Option**-\. On PC systems these characters are produced by typing **Alt-0171** and **Alt-0187**.



The table below lists some of the special codes that can be inserted into a header or footer:

Description	Code	Example
Page Number	«page #»	1
Date	«date:mm/dd/yy»	3/7/02
Date	«date:Month ddnth, yyyy»	April 8th, 2003
Time	{timepattern(now(),"hh:mm:ss am/pm"}	2:23:12 PM
Database Name	{info("databasename")}	Hotels

As the last two entries in this table show, you can actually insert any formula into a header or footer by surrounding the formula with { and }. Here's an example that uses two formulas to display the database name and number of records in the footer.

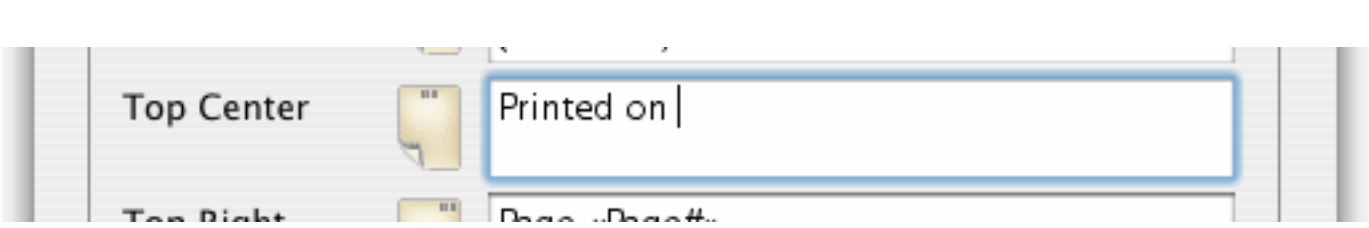


Here's what the finished footer looks like. See "Formulas" on page 1165 to learn more about formulas.

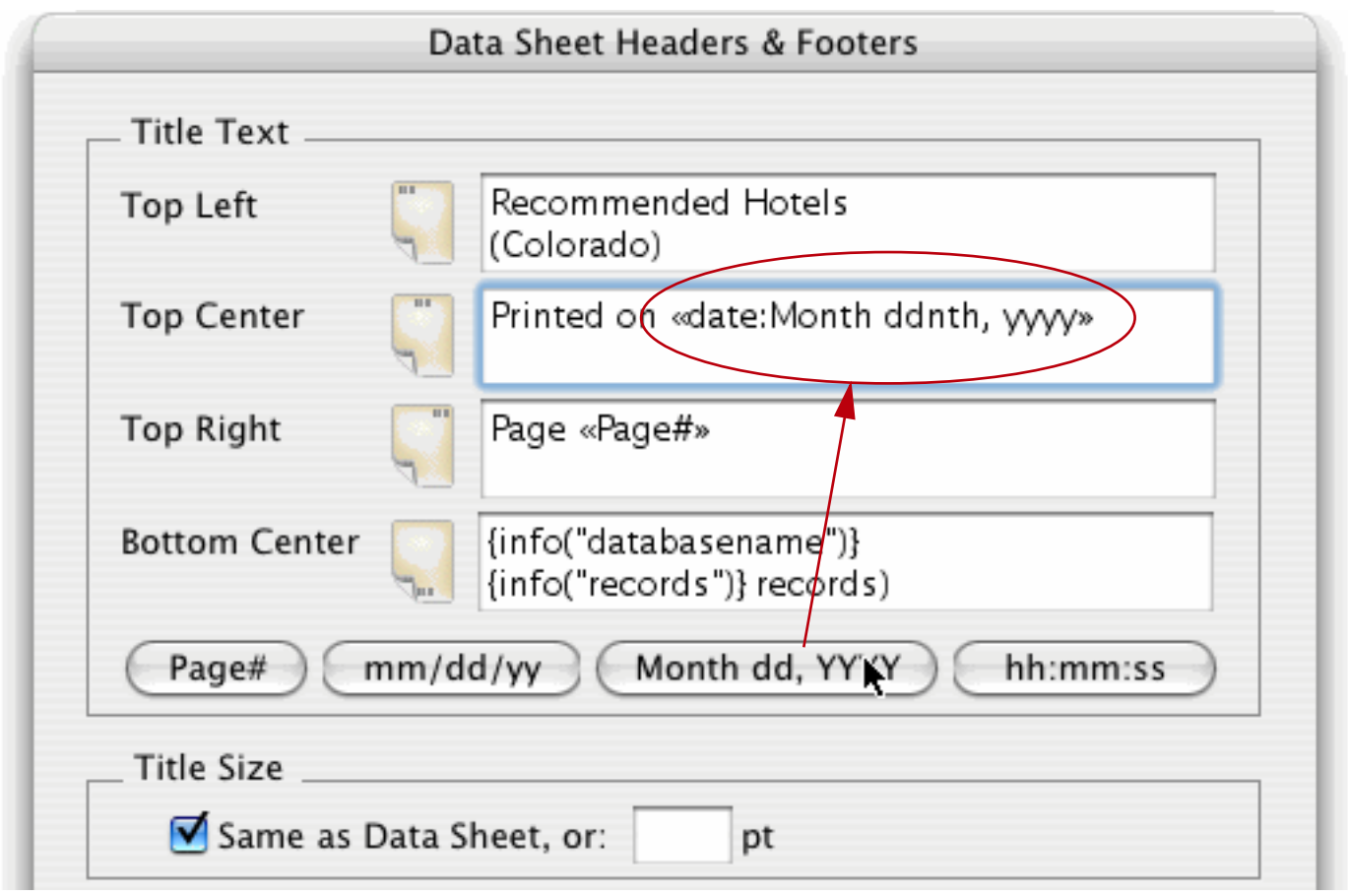
Best Western Sands	Cortez	30.00	81	565-3761	4
Best Western Shangri L	Breckenridge	38.00	41	453-2225	3
Best Western Silver Ki	Leadville	33.00	62	486-2610	3
Best Western Slpiner	Steamboat Springs	34.00	32	879-1430	3
Best Western Spa Motor	Denver	35.00	70	292-0220	3

Colorado Hotels (439 records)

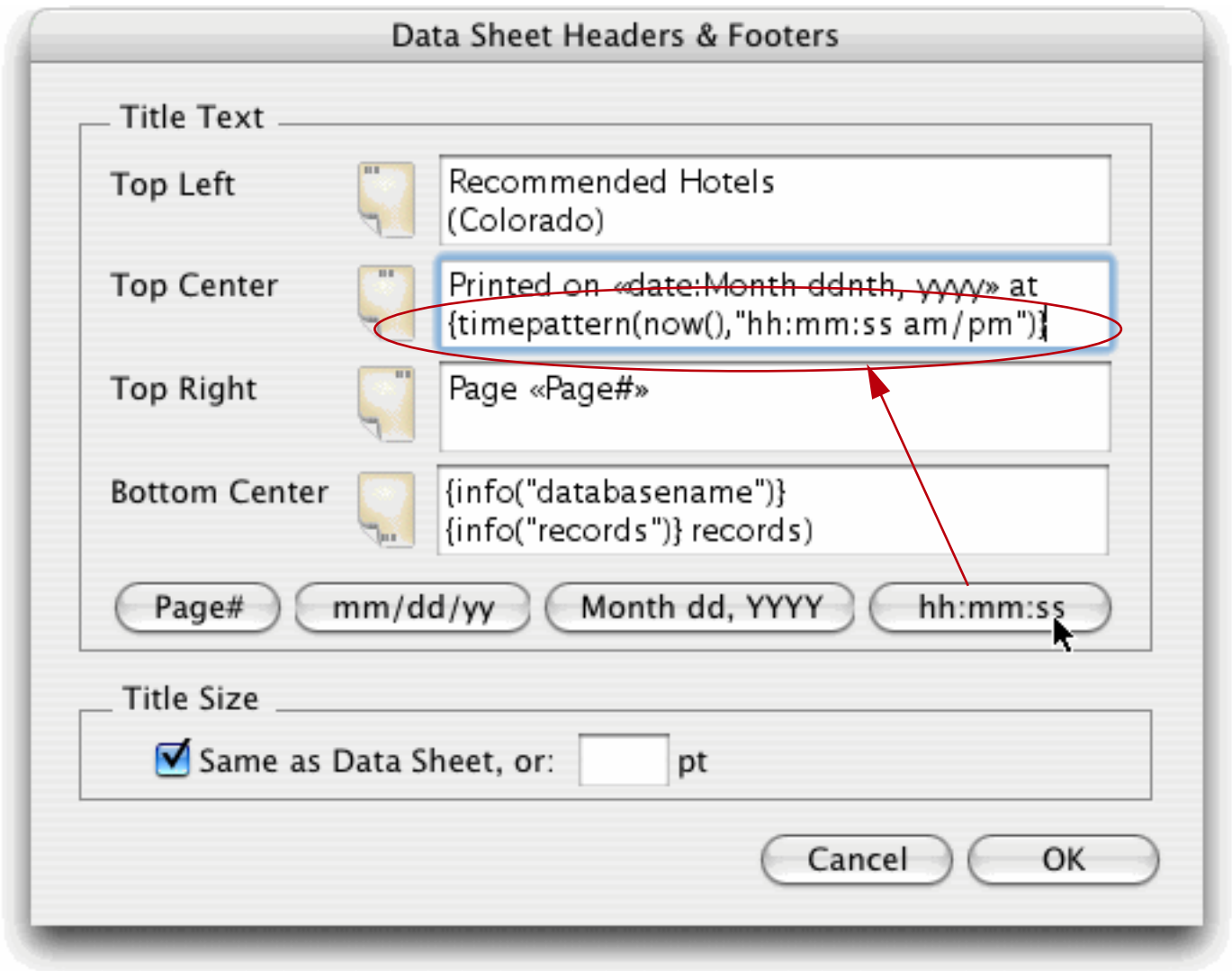
The four buttons just below the header/footer area will insert the most common codes into a header or footer for you. For example, suppose you wanted the top center header to show the time and date the database was printed, like this: Printed on May 23rd, 2000 at 4:21 PM. Start by opening the **Headers/Footers** dialog. Type Printed on into the Top Center header.



Now press the **Month dd, YYYY** button. Panorama will insert the code for you.



Next type in at and press the **hh:mm:ss** button.



Now press **OK**, and print or preview the data sheet. The top of the printed page will look like this:

Recommended Hotels (Colorado) Printed on July 6th, 2000 at 8:04:39 PM Page 1

Hotel	City	Rate	Units	Phone	Stars
6 & 40 Motel	Idaho Springs	19.00	30	567-2691	2
ABC Motel	Gunnison	26.00	18	641-9909	2
Airport Village Motor H	Denver	34.00	131	388-4821	3
Alamosa Inn Best Weste	Alamosa	35.00	143	589-5123	3
Alamosa Lamplighter M	Alamosa	30.00	73	589-6636	3
Alpenglo Motor Lodge	Winter Park	44.00	12	726-5294	2
Alpine Motel	Ouray	28.00	12	325-4546	2
Alpine North Motel	Durango	34.00	21	247-4042	3

Printing a Form

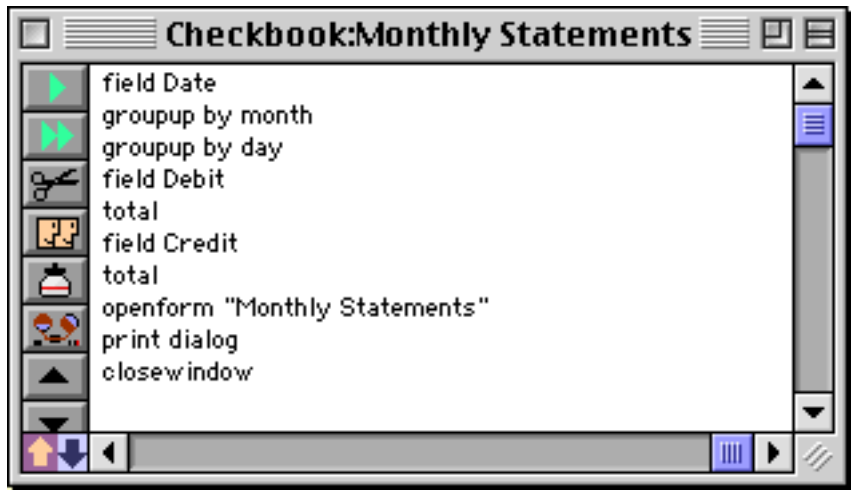
When you ask Panorama to print a data sheet, it prints an exact line-for-line replica—with one line for every visible record in the database. But when a form view is printed it would not be practical to print the entire form for every record. Instead, Panorama lets you specify what part of the form should be printed where on the report. To do this you use special graphic objects called **report tiles**. Each report tile defines a section of the form that is to be included in a printed report. By combining several tiles you can create a complex report with headers, footers, subtitles, summaries, etc. Report tiles and their applications are covered in detail in the following chapter, see “Custom Reports” on page 1049.

If you ask Panorama to print a form that has no report tiles, it will simply print the upper left hand corner of the form (8" by 10"), one record per page. In other words if your database contains 187 visible records, Panorama will print 187 pages. For most reports you will want to use report tiles so that more than one record is printed per page.

Preparing Data For Printing

Before you print the database, you may want to prepare it for printing. If you want the data printed in a certain order (for example alphabetical by name), you must sort the database before you print it (see “Sorting” on page 359). If you want to print only a portion of the database (for example, only zip codes in California), you must use the **Find/Select** command to make the rest of the database invisible (see “Searching and Selecting” on page 367). If you want to print subtotals or other summary information, you must group and total the database before printing (see “Summaries and Outlines” on page 393).

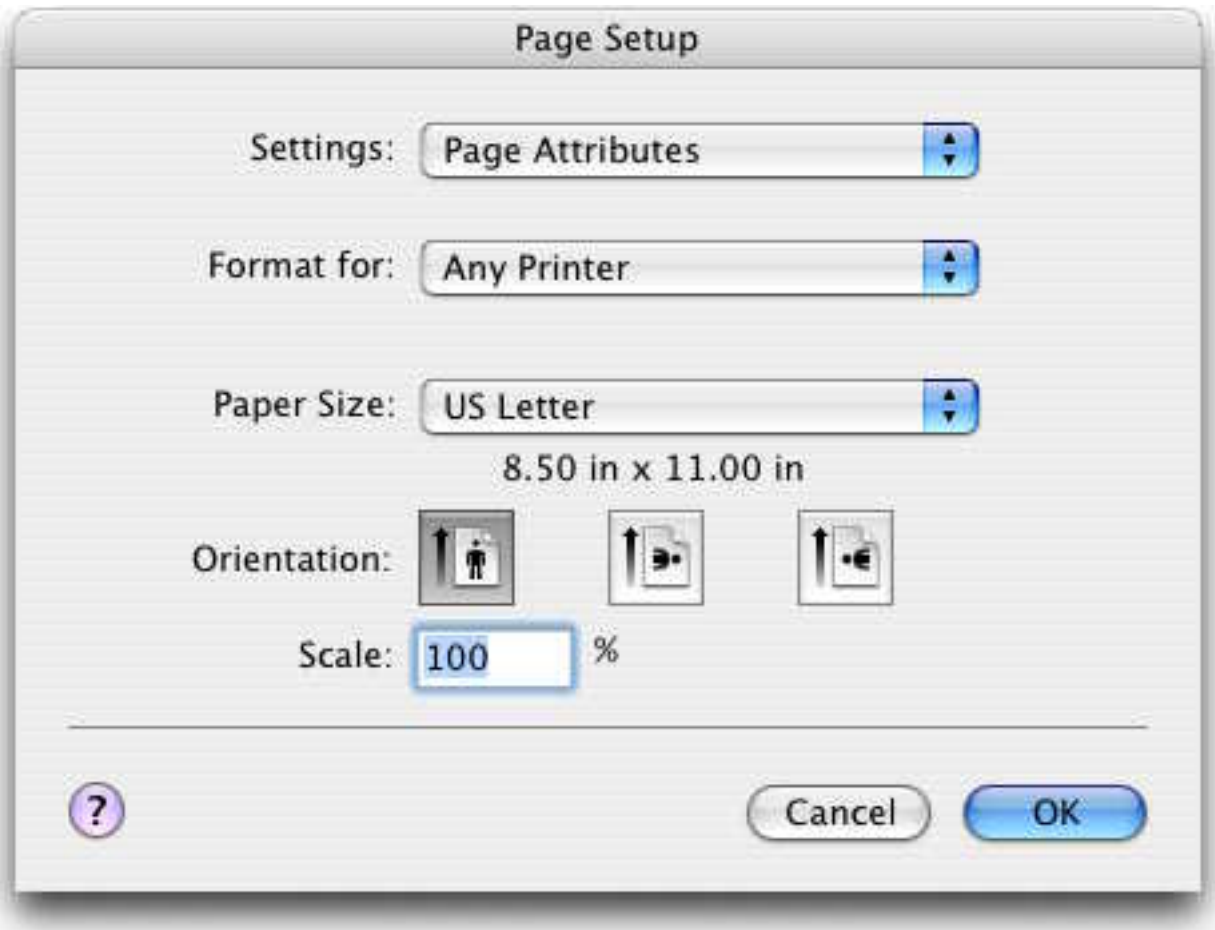
If you are going to print a report often, you may want to create a procedures to automatically prepare the data before printing. For example, here is a procedure that groups the database by month and by day, calculates the totals, then opens the form Monthly Statements and prints the database.



For more information on procedures see “Procedures” on page 1339.

The Page Setup Dialog

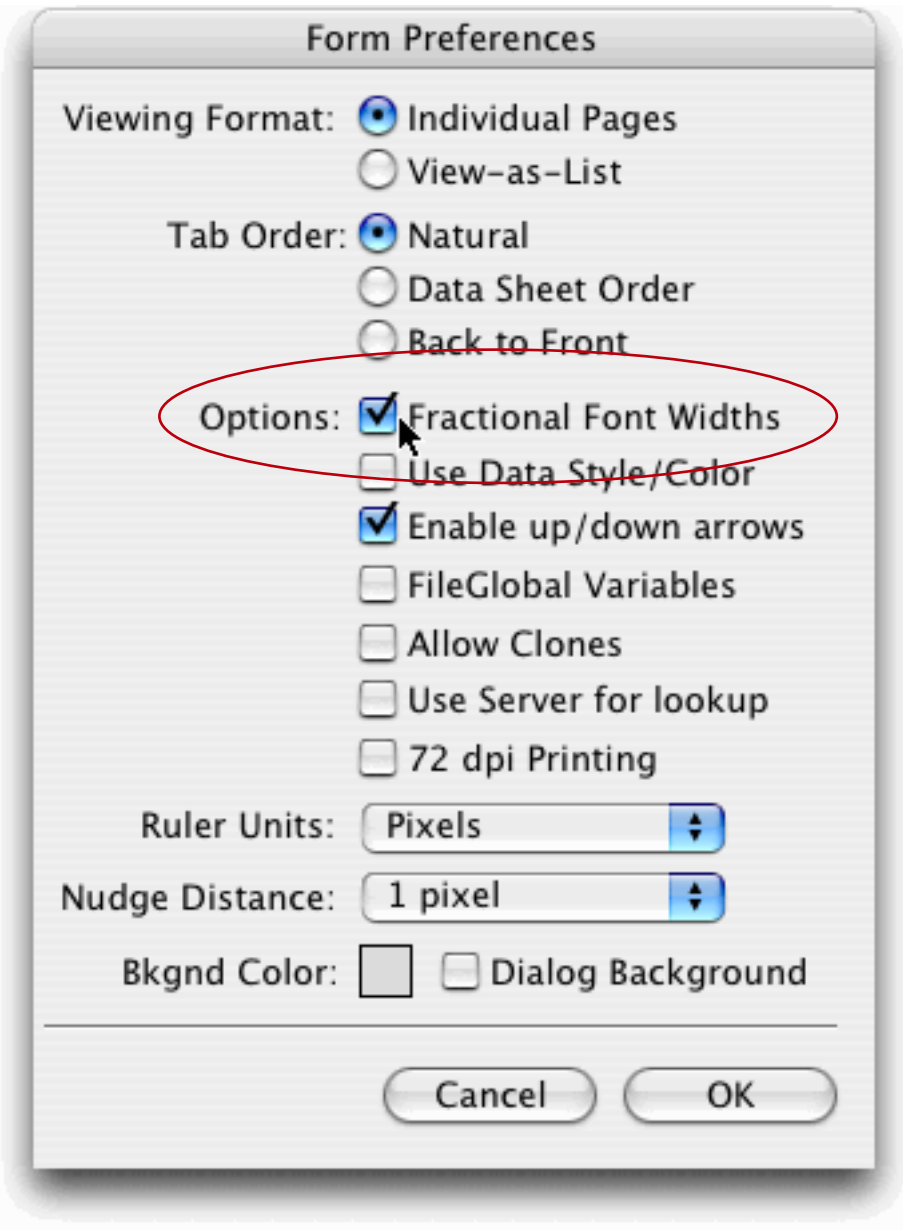
The **Page Setup** command in the File Menu displays a dialog that allows you to specify several printing options. The exact options available depend on the operating system and what kind of printer you are using, but in general you can control the page size, orientation (tall or wide), and print reduction factor. Here is a typical **Page Setup** dialog.



Each form view has its own separate page setup. The page setup is remembered as part of the form. For example, a single database can have an invoice that is printed using the tall orientation, and a report that is printed using the wide orientation (sideways). You don’t have to remember to switch the page setup when you switch forms—Panorama will do it for you. Incidentally, be sure to save the file after you change the page setup. If you save the file, Panorama will remember the page setup the next time the file is opened. (However, not all print options are saved as part of the database. The exact options that are saved vary from printer to printer.)

Fractional Fonts

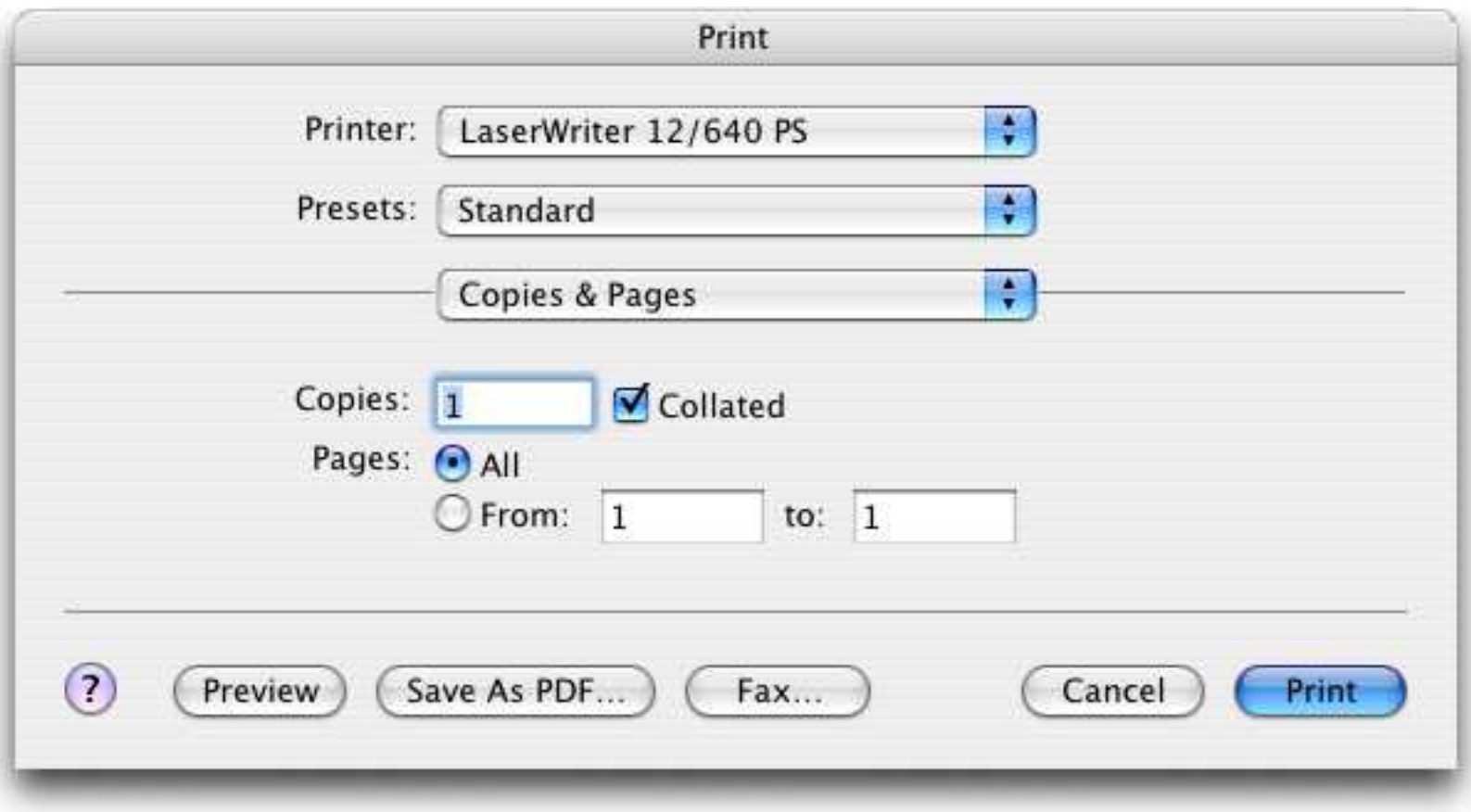
The Fractional Fonts option is in the **Form Preferences** dialog (Setup Menu).



If this option is checked, Panorama will print using the most accurate character spacing. This option should only be checked when you are printing using a Postscript or TrueType font (which these days, is almost always true).

The Print Dialog

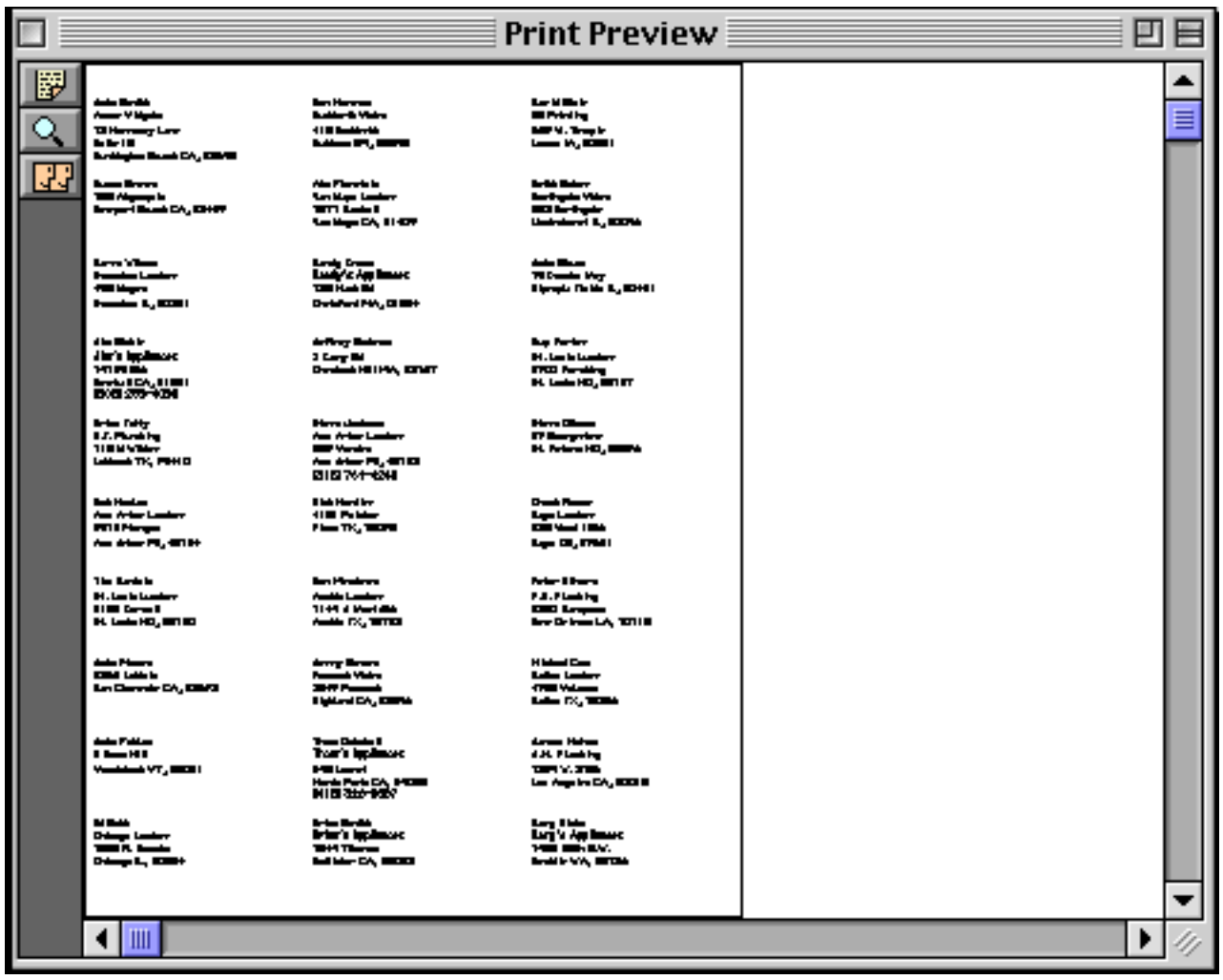
The **Print** command also displays a dialog box allowing you to choose printing options. You can choose which pages to print, how many copies to print, and whether you want to manually feed the paper. The exact options will depend on the operating system and printer you are using. Here is a typical **Print** option dialog.



For the exact details on the operation of this dialog see the documentation that came with your printer.

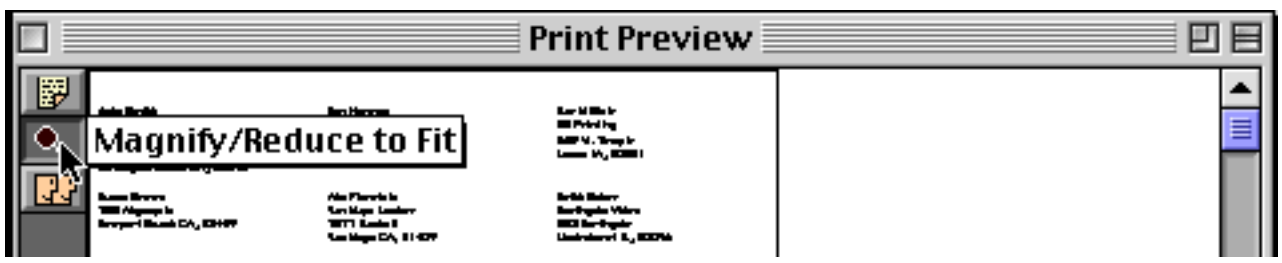
Print Preview

The **Print Preview** command (File Menu) allows you to see what a report will look like without using any paper. This command opens a new window called Print Preview.

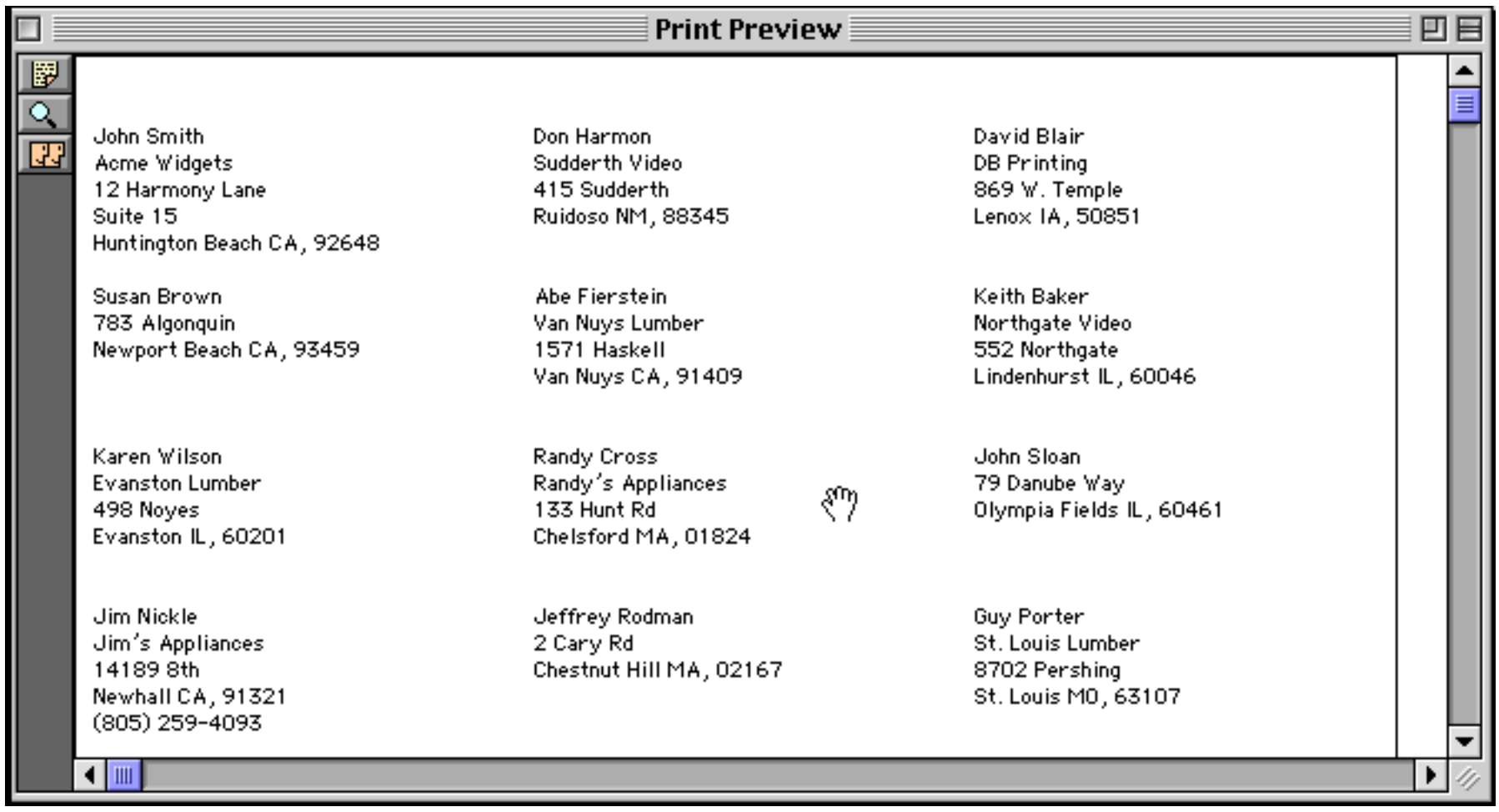


The preview window normally fills the entire screen, but you can change the size and location of the preview window after it is opened using the grow box and drag bar. Only one preview window can be open at a time, and all other Panorama windows are disabled when the preview window is open. You can still see the other windows, but you cannot bring them to the front or click on them.

The image of the printed page is reduced so that the entire page fits in the window. To expand the image to full size, click on the magnifying glass tool.

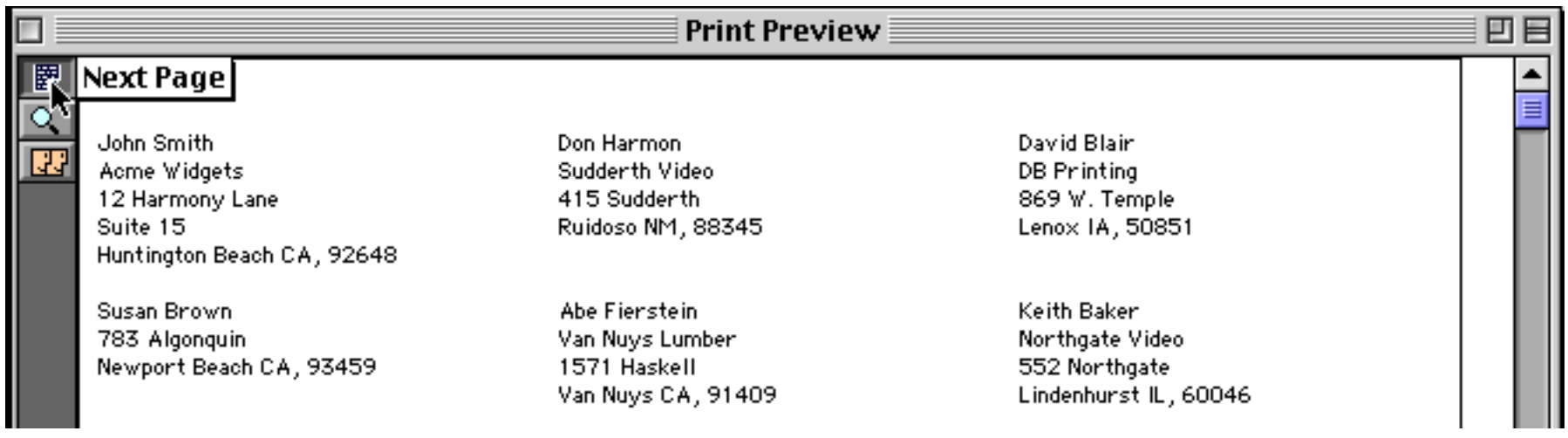


Each time you click on this tool, the window toggles between reduce to fit and 100% magnification, like this.

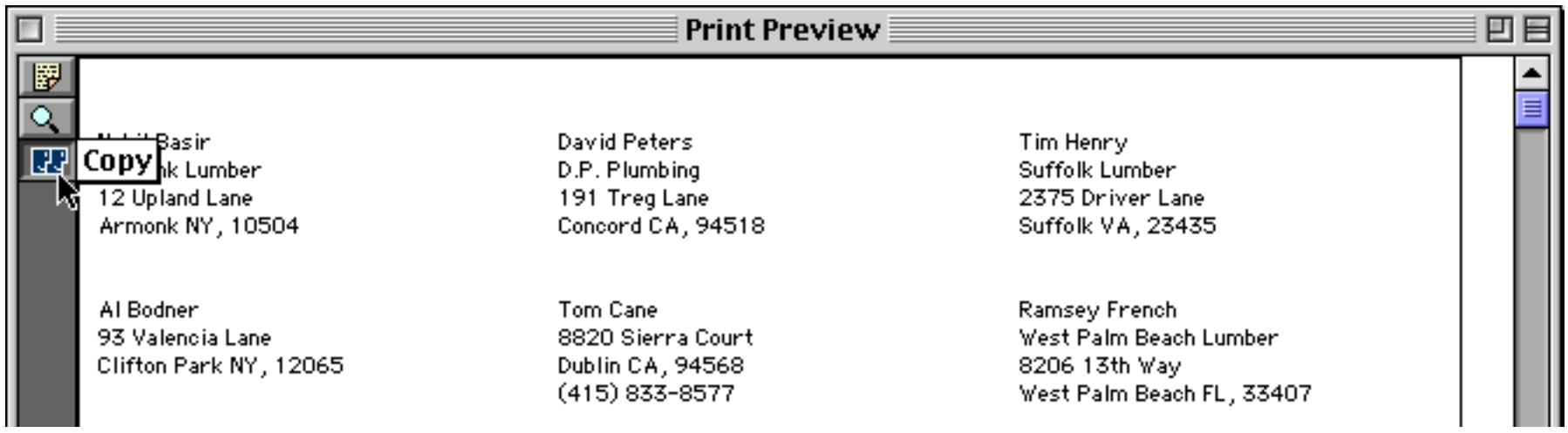


When the preview is magnified to 100%, you can use the scroll bars to shift to different locations within the page. You can also use the mouse to push the preview to a different spot. Simply press the mouse (which looks like a hand, see illustration above) on the preview window, then drag the image to a new location.

To preview the next page, click on the **Next Page** tool.



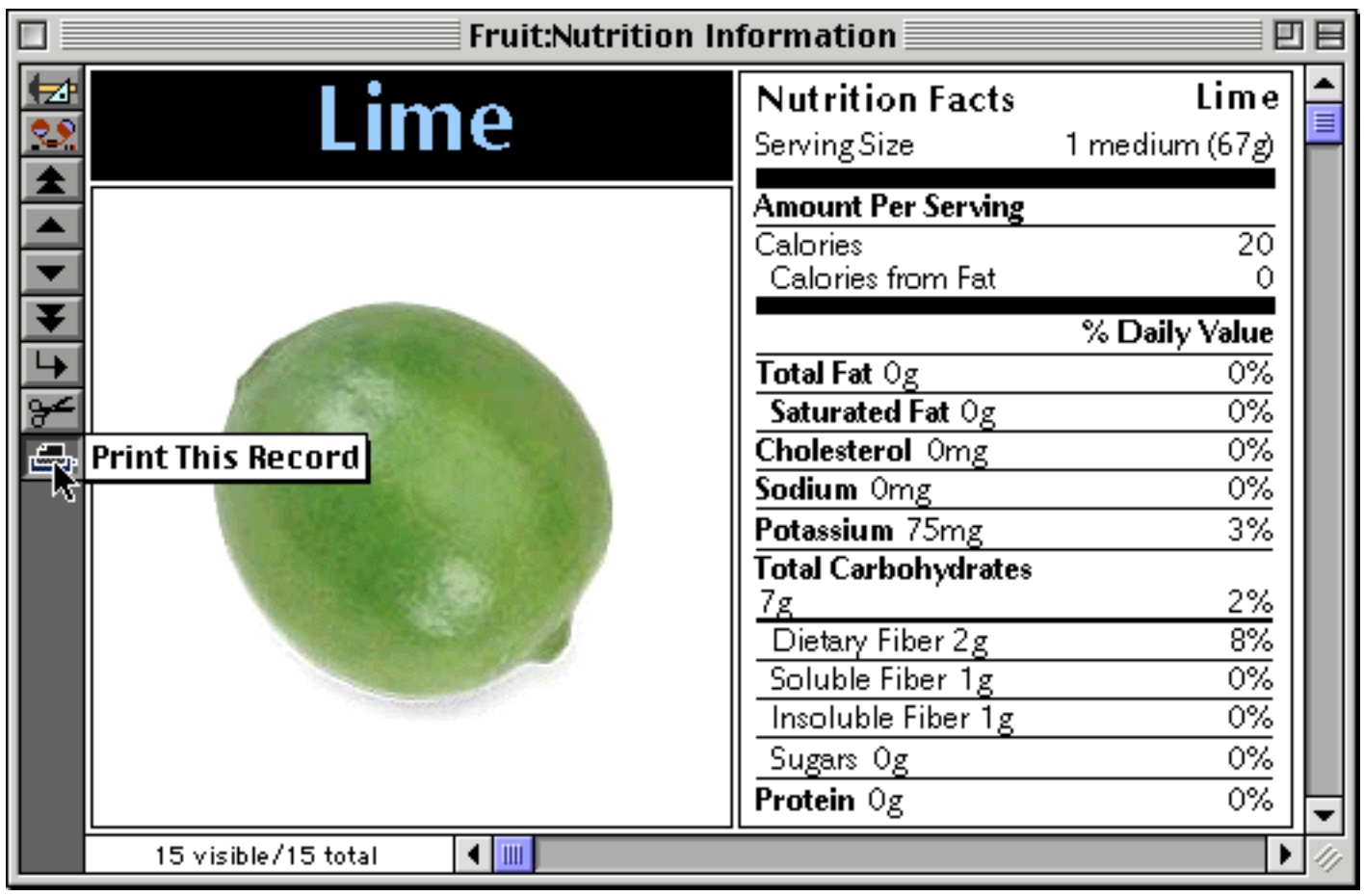
To copy an image of the page to the clipboard, use the **Copy Page** tool.



Once the image is on the clipboard, you can copy it into a graphic or page layout program.

Print One Record

The **Print** command normally prints every visible record in a database. If you want to print just one record, use the **Print One Record** tool at the bottom of the tool palette.



This tool only appears in the tool palette of form windows. It is not available for any other view (data sheet, design sheet, etc.).

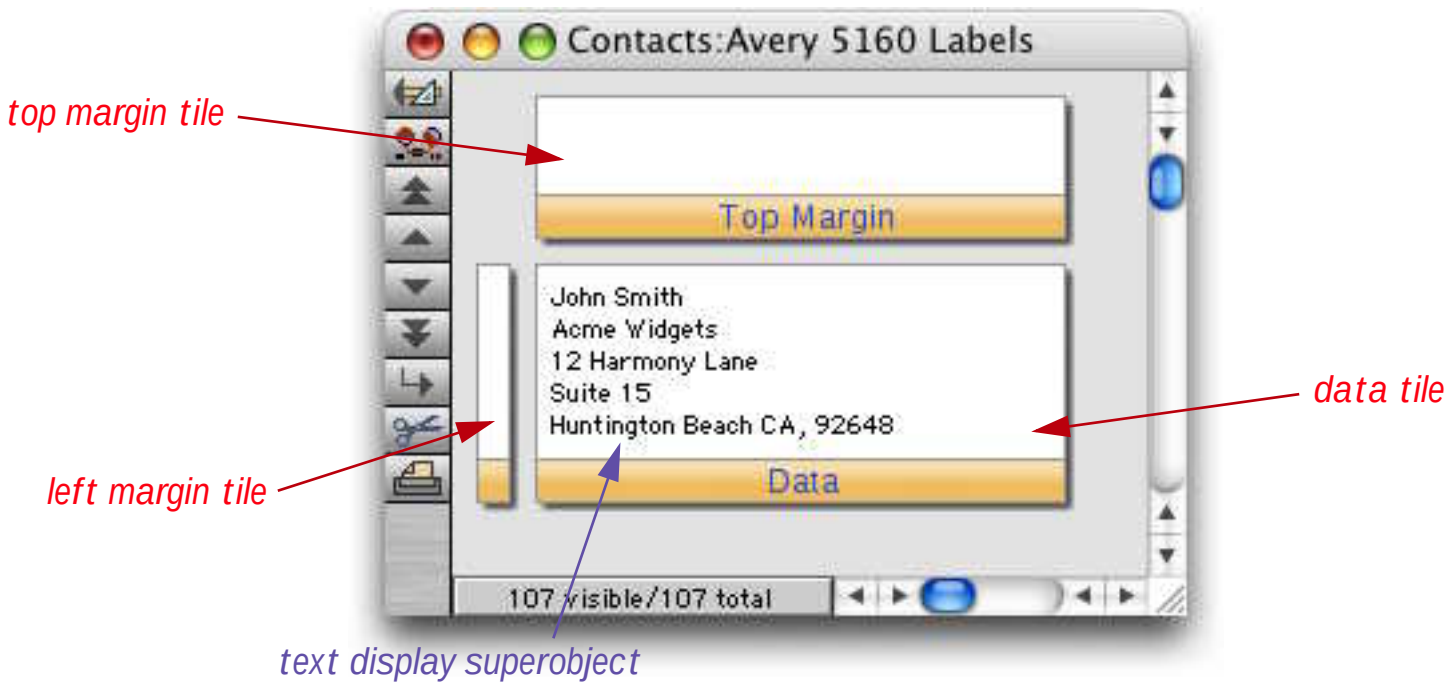
Chapter 21: Custom Reports



Panorama has a very flexible system for printing custom reports. Panorama assembles each report page by taking pre-defined components from your form and sliding them into position on the page. Because the report components fit together like floor tiles, these components are called **report tiles**. Each tile in the form corresponds to a section of real estate on the printed page.

A form for generating a report may contain only a single report tile, or it may contain dozens of tiles. Panorama checks for the presence of each type of tile as it is building the report, and if found, uses the tile to build one section of the report. The size and shape of the tiles determines the overall layout of the report.

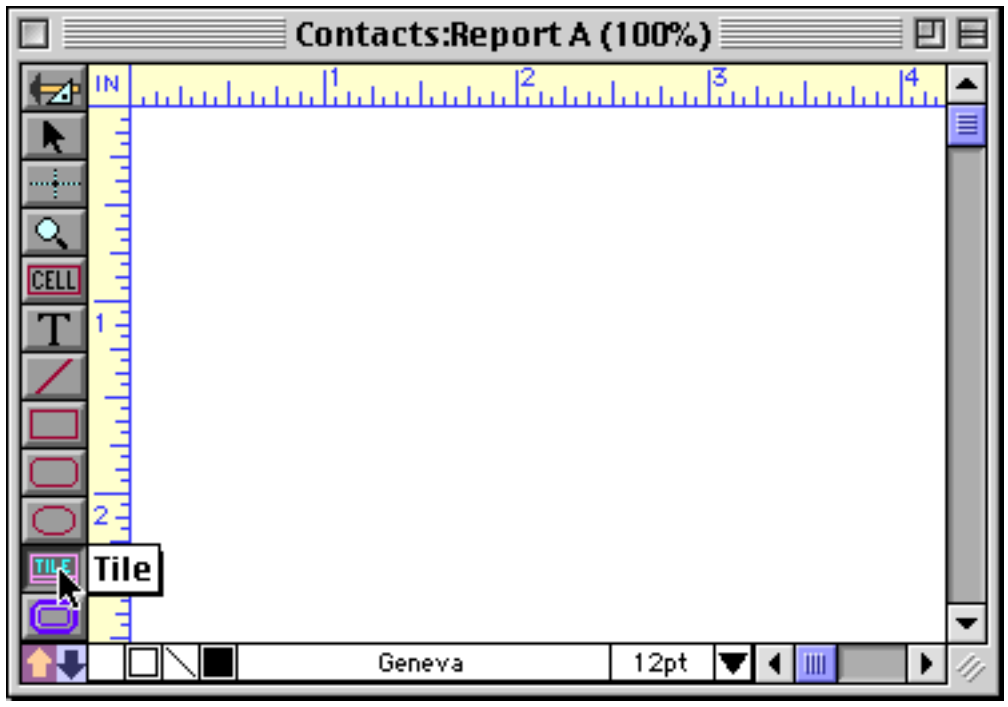
Report tiles are graphic objects that are part of a form. Here is an example of a typical form that contains three report tiles—top margin tile, left margin tile, and data tile. (This form also contains a Text Display SuperObject that has been placed on top of the data tile—more on that later.)



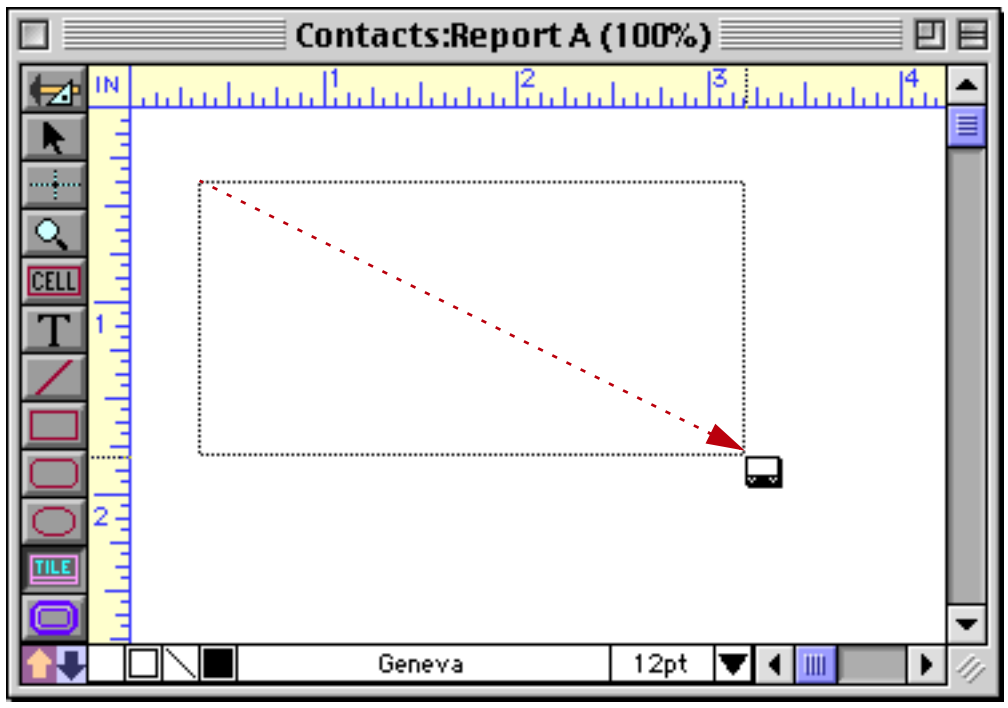
You can manipulate report tiles just as you would any other graphic object—drag, nudge, copy, etc. (see “Graphic Design” on page 491). Each form can be used to generate a unique report. Since a database can contain an unlimited number of different forms you can also have an unlimited number of custom reports. To print a particular custom report simply open the appropriate form (see “Switching Between Views” on page 222) and print. (You may also want to prepare the data before you print by sorting, selecting, and/or summarizing the data. See “Sorting” on page 359, “Searching and Selecting” on page 367, and “Summaries and Outlines” on page 393).

Working with Tiles

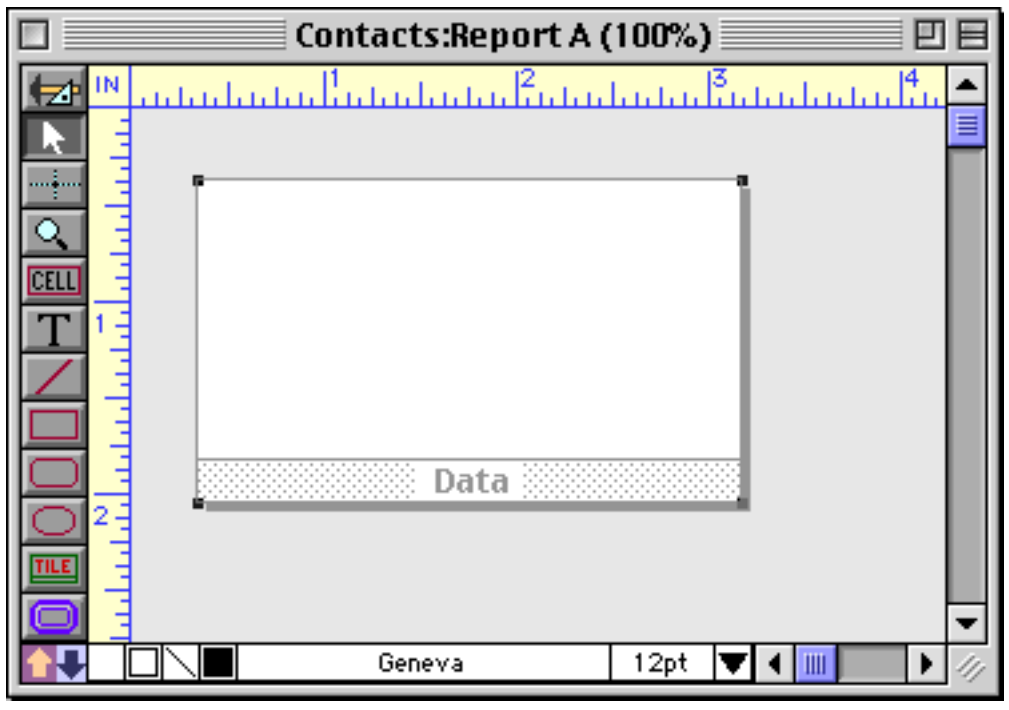
To create a new tile, start by selecting the **Tile** tool.



Next, drag the mouse across the form in the spot where you want to place the new tile. Any empty spot will do. The position of a tile on the form does not affect how it is printed—only the size and shape.

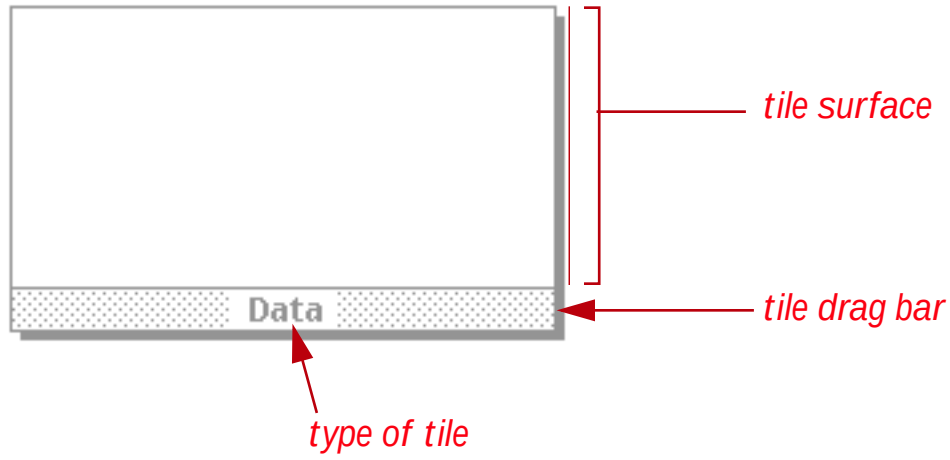


When you release the mouse, Panorama creates the new tile. If this is the first tile on the form, Panorama automatically creates a **data tile** (see “Data Tiles” on page 1064). Later, Panorama will give you a choice.

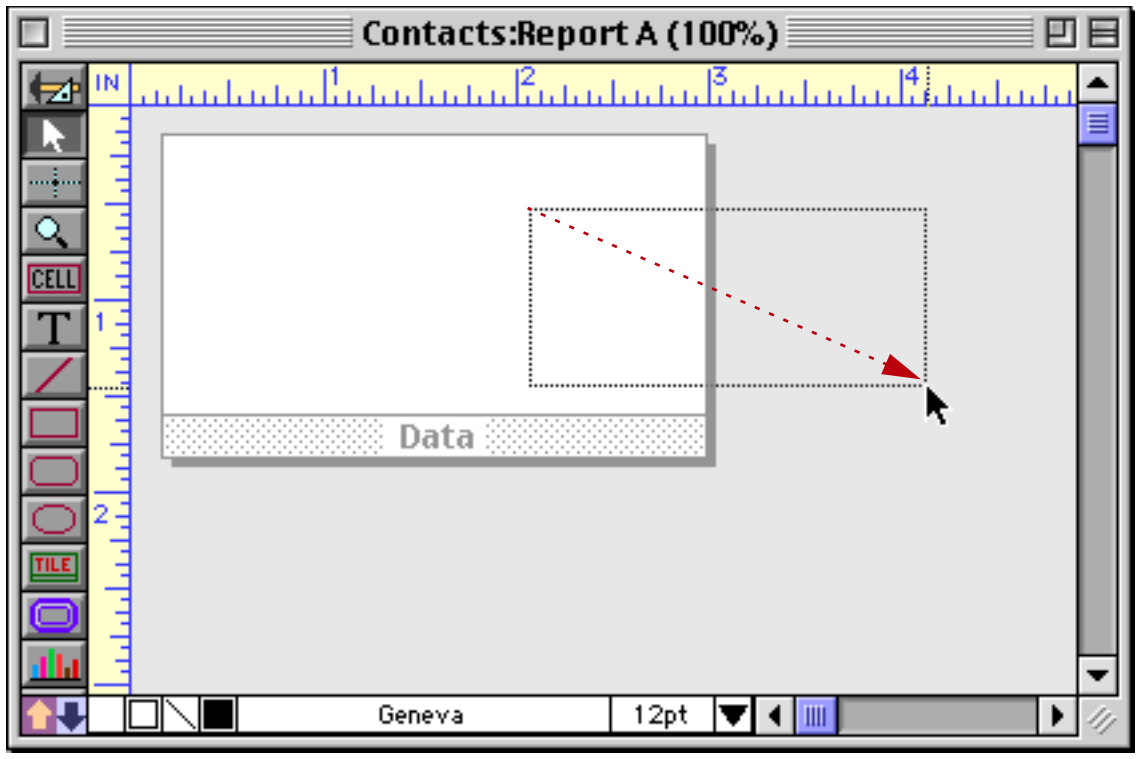


When you add the first tile to the form (the data tile), the background (outside the tile) will turn gray, as shown above. Only graphics or text that are on top of a tile will be printed — any graphics or text in the gray area will not be printed.

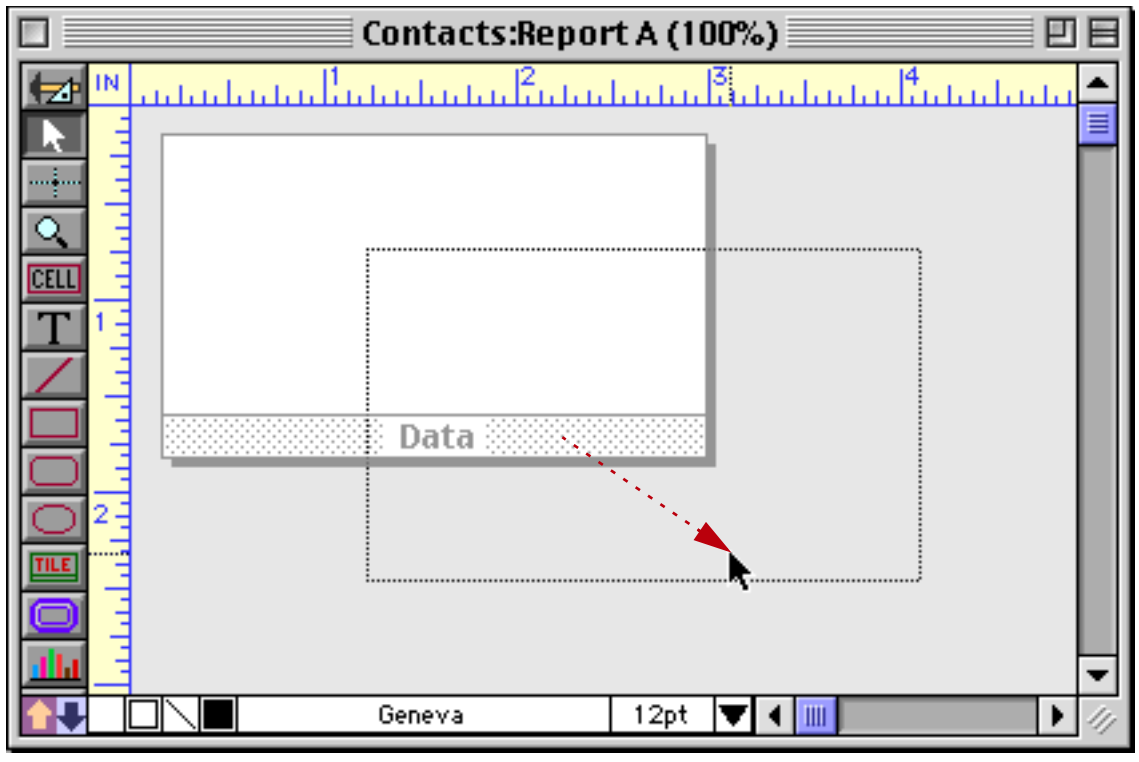
On the screen, a tile looks sort of like an upside down window. Tiles are divided into two parts: the surface and the drag bar. The surface is the actual printed area of the tile.



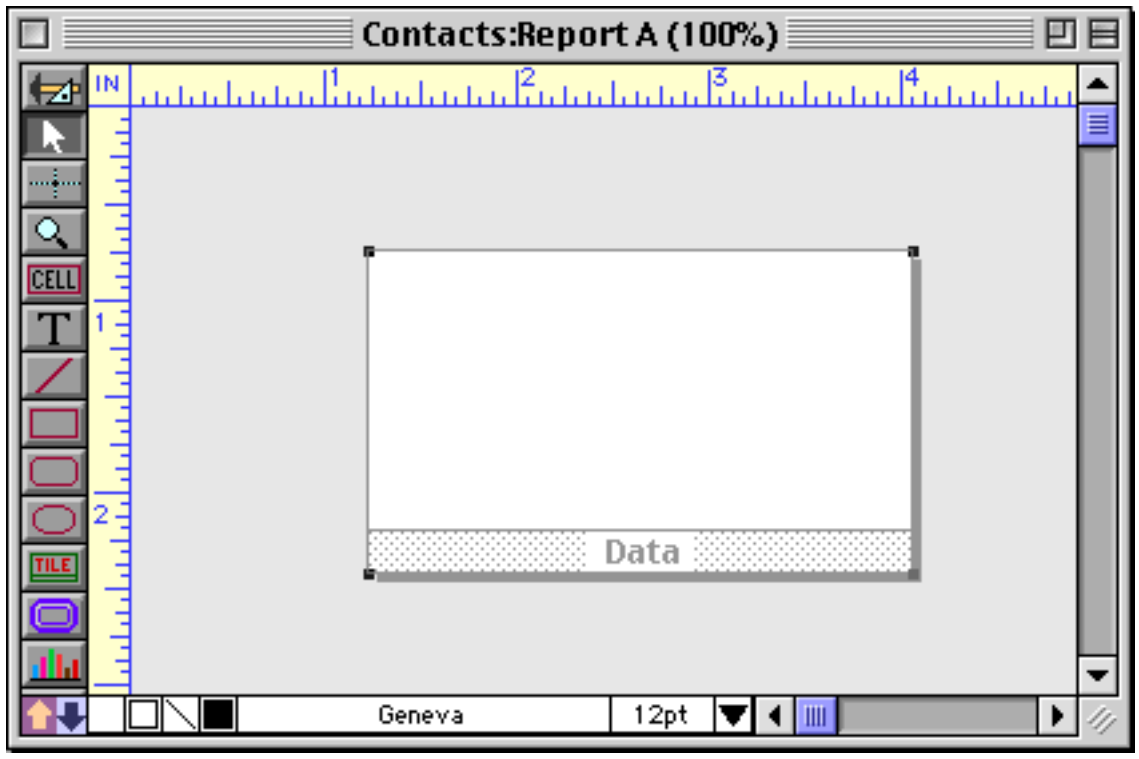
Unlike other graphic objects that can be manipulated by clicking anywhere in the object, a tile is only sensitive to clicks on its drag bar. If you click and/or drag on the surface of a tile, the object is not selected and does not move. Instead, a selection marquee appears, just as if you had dragged on an empty spot in the form (see "Selecting Multiple Objects at Once" on page 502).



To move a tile, press the mouse on the drag bar and drag the tile to the new position.

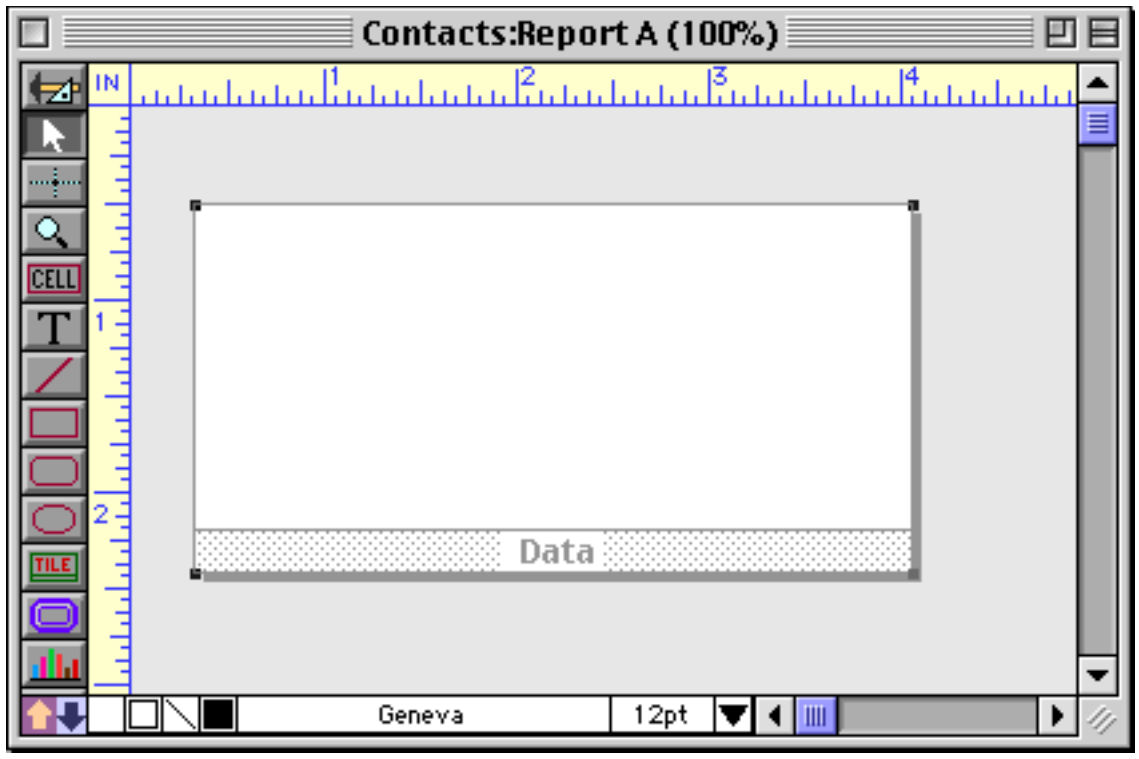


When you release the mouse, the tile moves to the new position, just as with dragging any other kind of object.



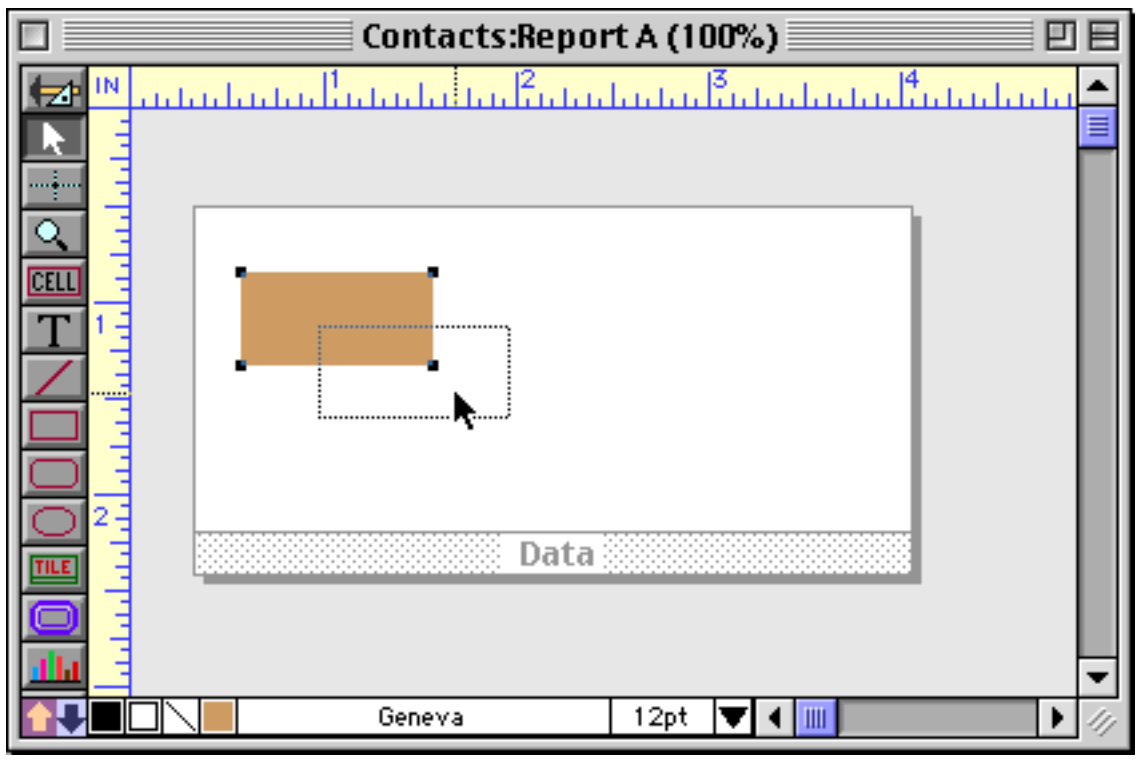
To select a tile, click on the drag bar. When the tile is selected, four handles appear around the corners of the tile, as shown above. You can use these grips to change the size of the tile, again, just like any other kind of object.

Release the mouse to see the new size.



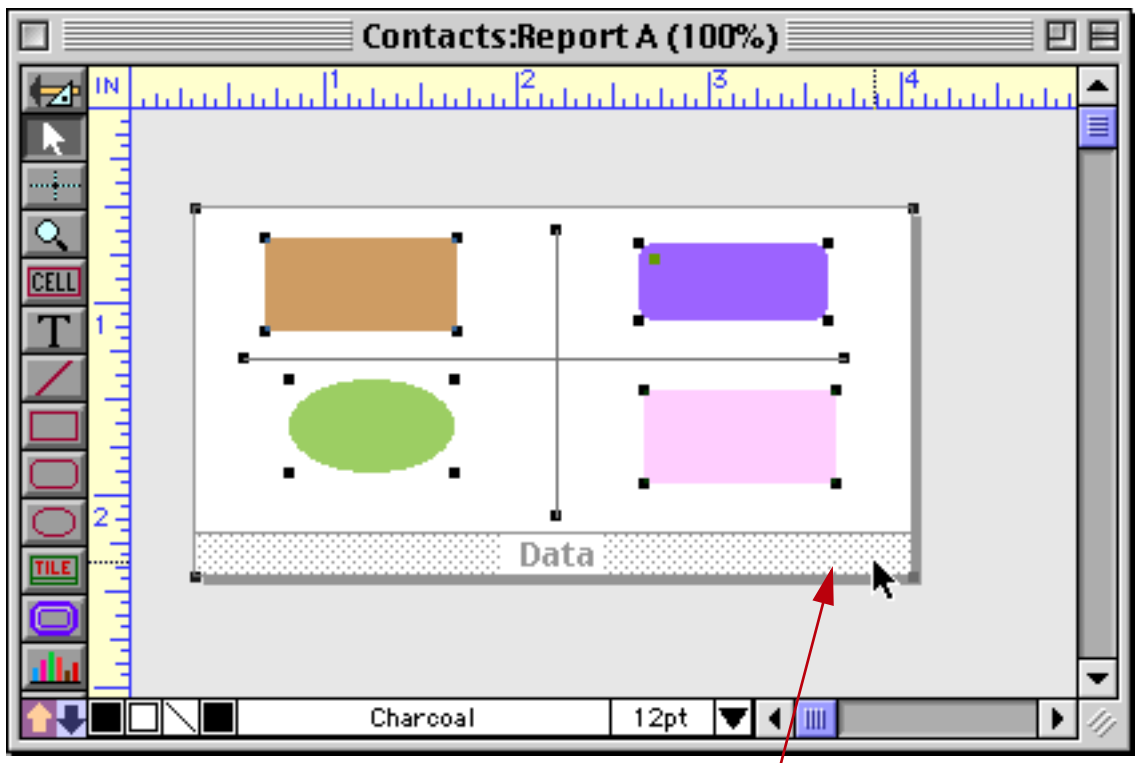
Tiles can also be moved or resized with the **Dimensions** command (see "Viewing and Setting Exact Object Dimensions" on page 512) and by nudging with the arrow keys (see "Nudging an Object (or Objects)" on page 509 and "Nudging the Size of an Object" on page 513). Note: When using the **Dimensions** command the dimensions are for the surface of the tile only, not including the drag bar along the bottom of the tile.

As mentioned above, the surface of the tile is not sensitive to the mouse. In other words, clicking on the surface area does not select the tile, and you cannot move the tile by dragging on the surface. However, you can place objects on top of the tile and move or select them.



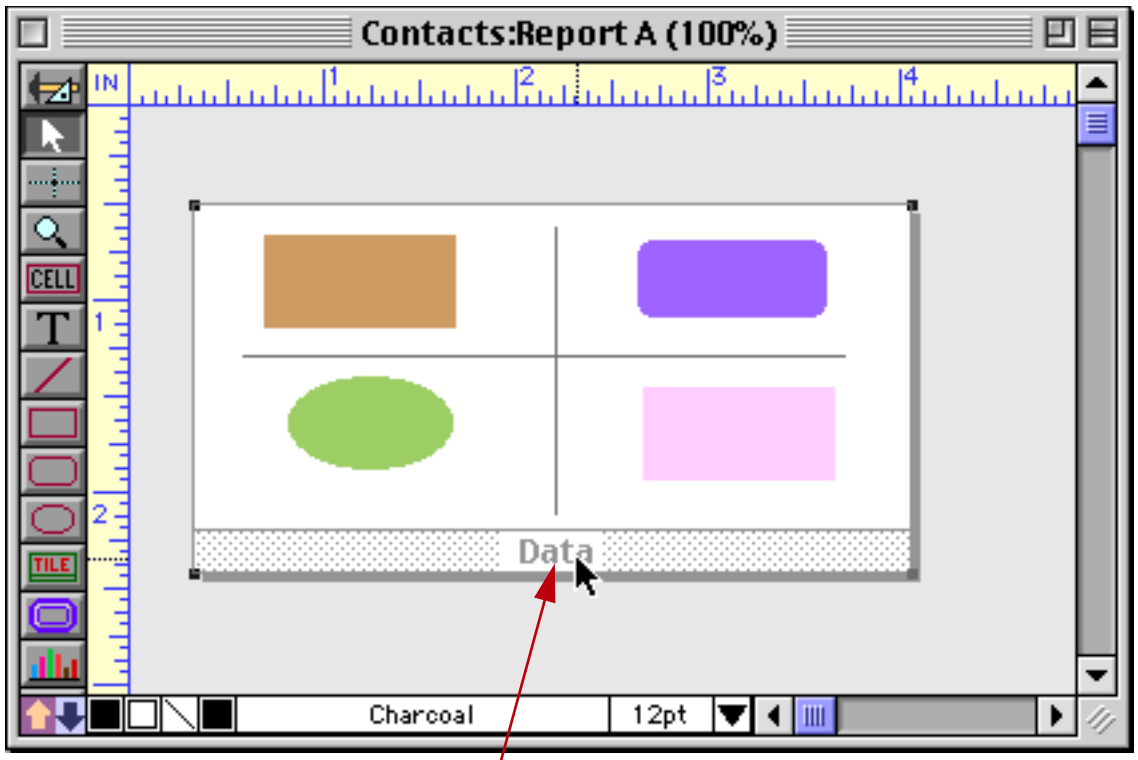
In fact, as you will see later, most tiles must have other objects placed on top of them to build a complete report.

If a report tile has one or more objects placed on top of it, double clicking on the tile's drag bar will select both the tile and all of the objects on top of the tile. This is convenient if you want to move or copy the tile and the objects to a new position or to a different form.



double click drag bar to select tile and all objects on the tile

However, double clicking on the type of tile does not select the objects.

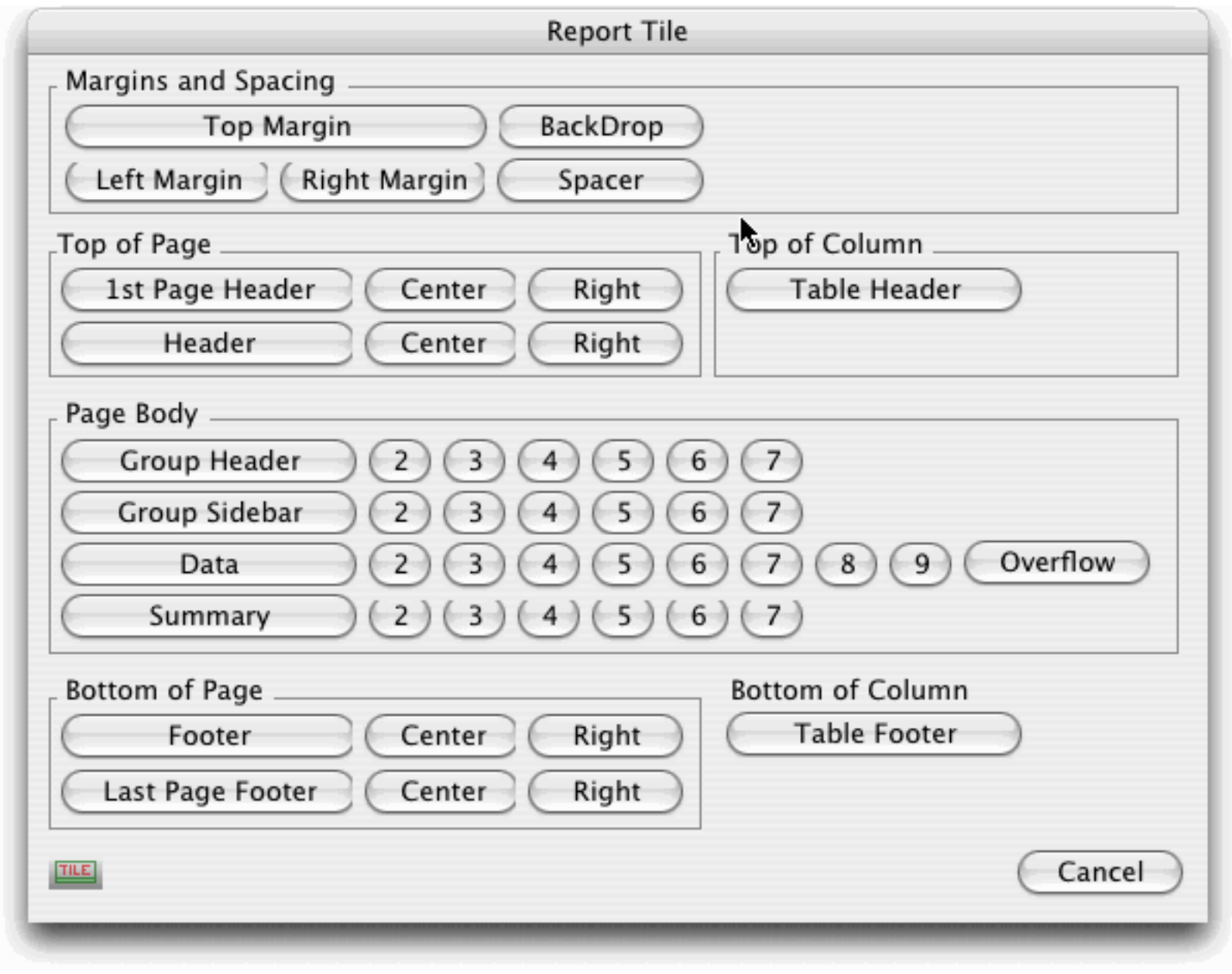


double click here to open the tile configuration dialog

Instead it opens the tile configuration dialog, allowing you to change the type of a tile. Depending on the number and type of tiles that are already on your form you may get the simple dialog shown here:



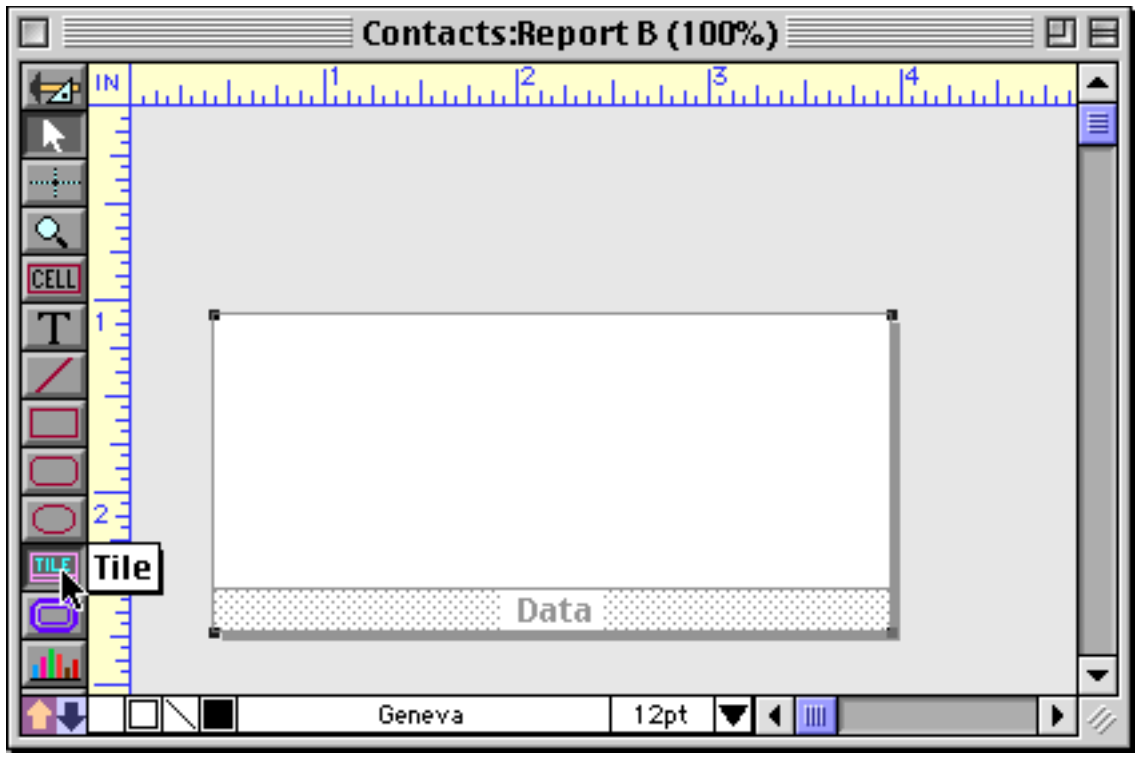
or the more complex dialog shown here (the *Specialized Tile* dialog).



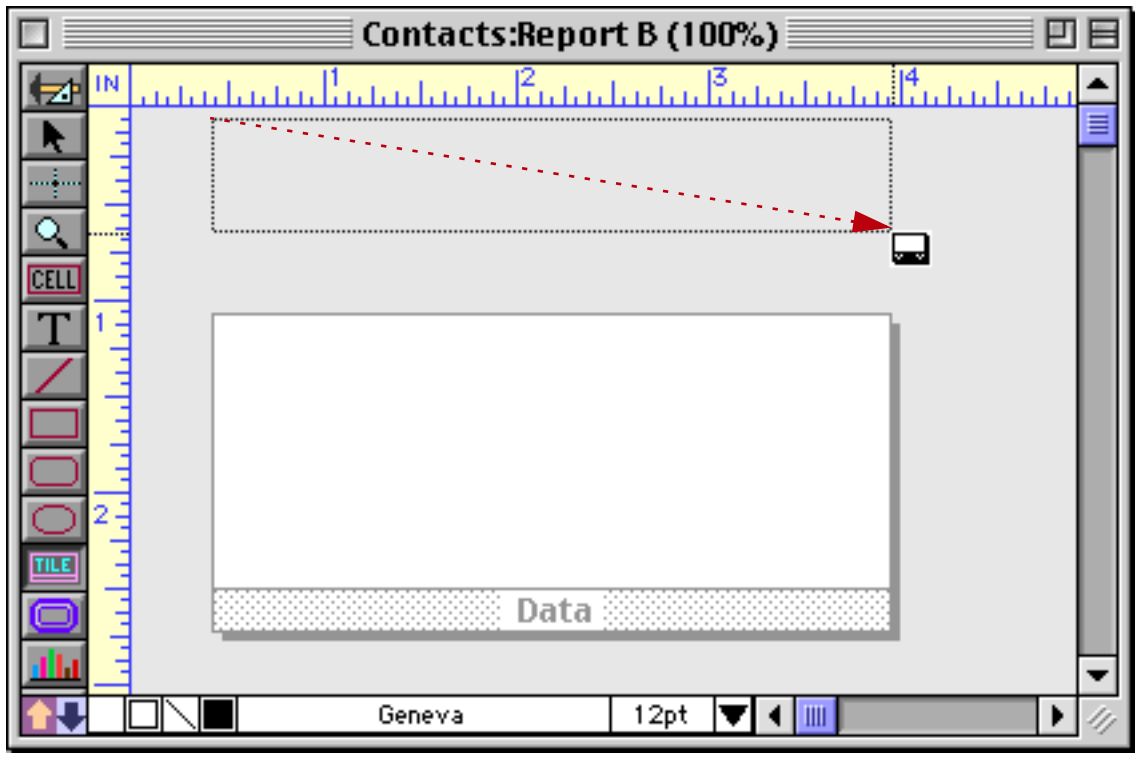
To change the tile's type simply pick the appropriate button, or press **Cancel** to keep the same type.

Creating Additional Tiles

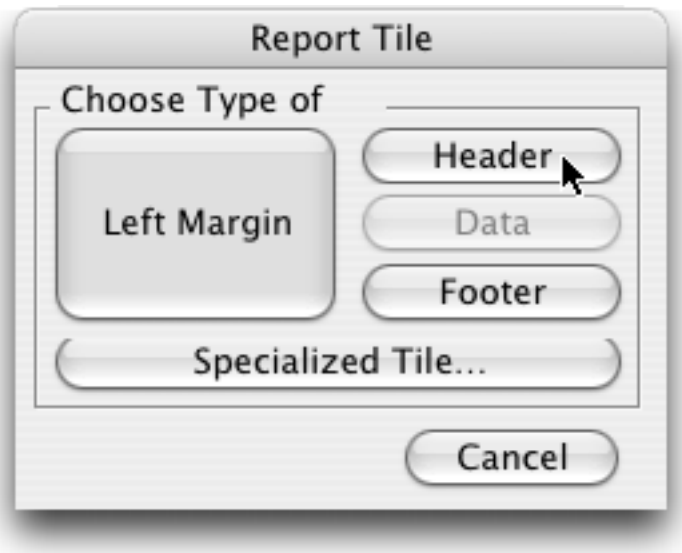
A minimum report contains at least a data tile. Many reports will contain additional tiles. To create an additional tile, start by selecting the **Tile** tool.



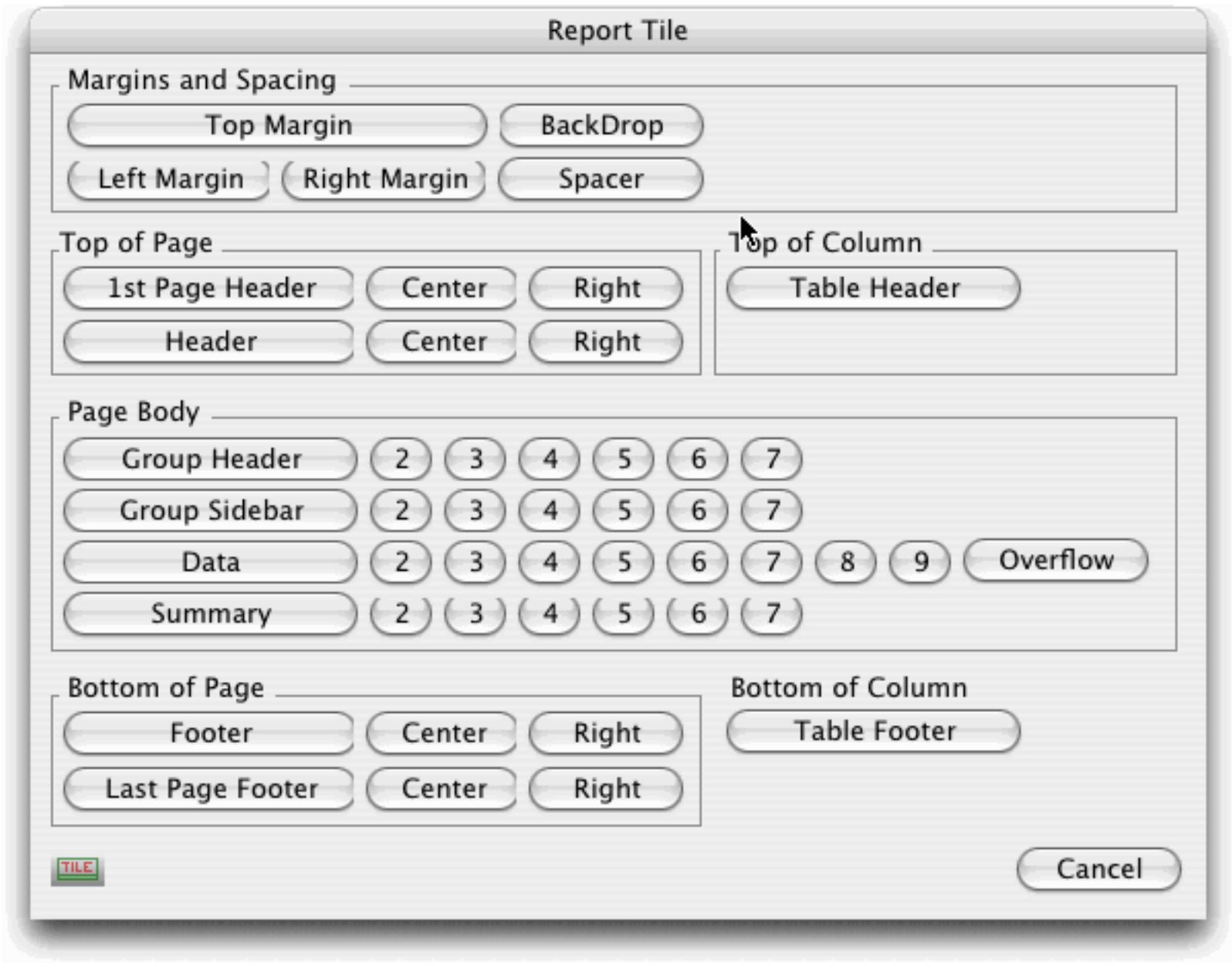
Next, drag the mouse across an empty (gray) spot on the form. Remember, the exact position of the tile is not important, only the size and shape.



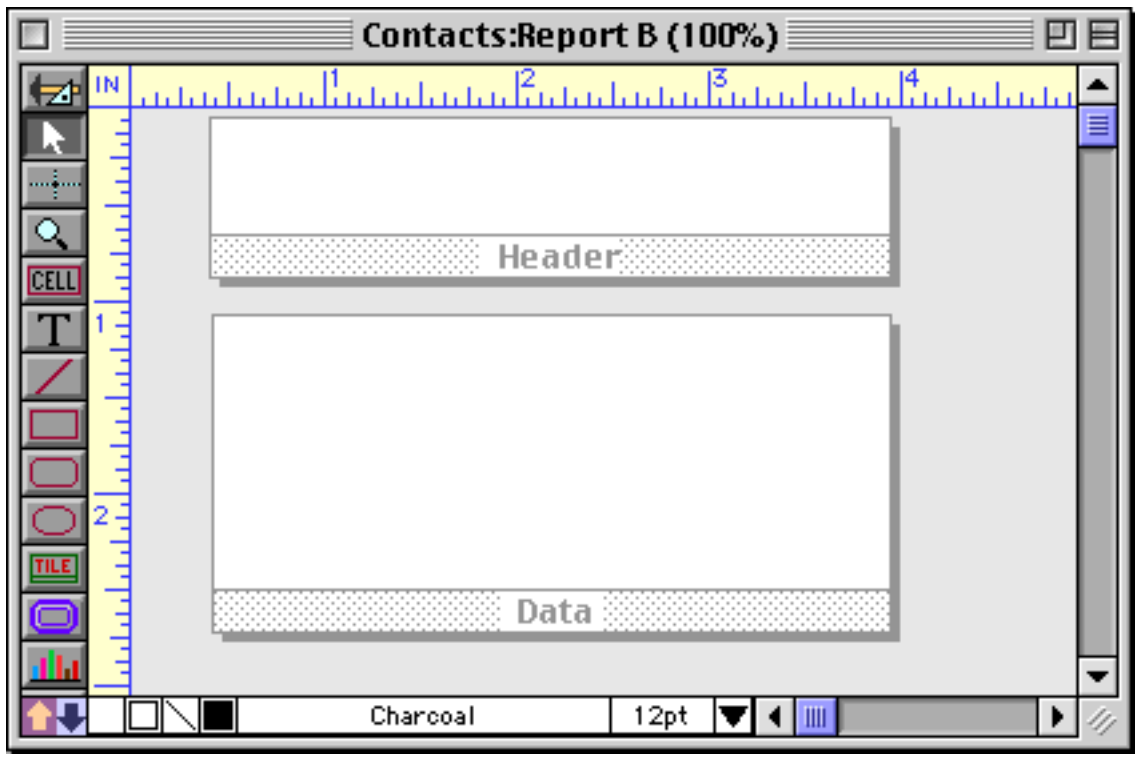
When you release the mouse the tile configuration dialog appears. Since this form currently has only one tile, the simple version of the configuration dialog will appear.



To create a **Header** tile, **Footer** tile, or **Left Margin** tile simply press the appropriate button. To create any other type of tile press the Specialized Tiles button, which makes the “long” version of the tile configuration dialog appear.

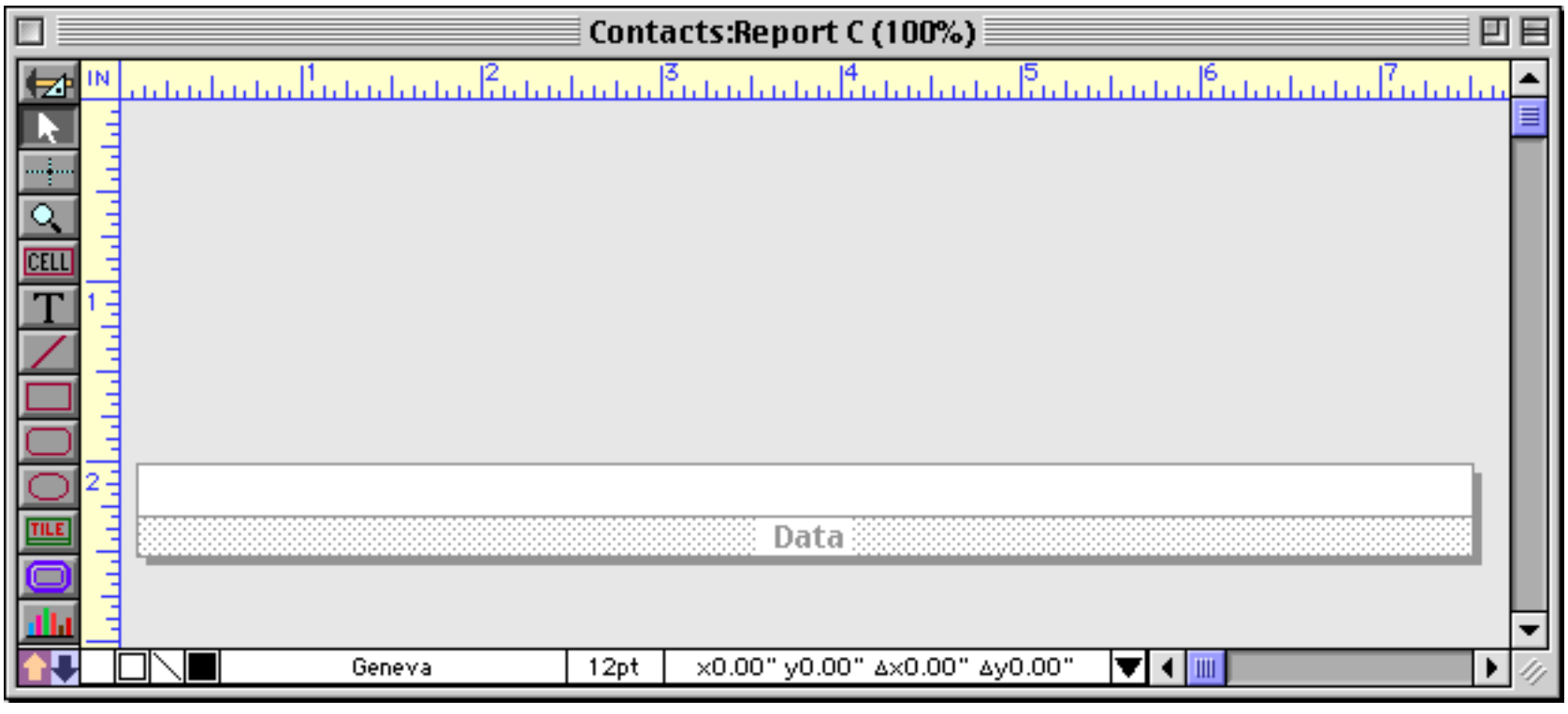


The multitude of choices in this dialog will be discussed later. For now we'll simply press the Header button to create a header tile.

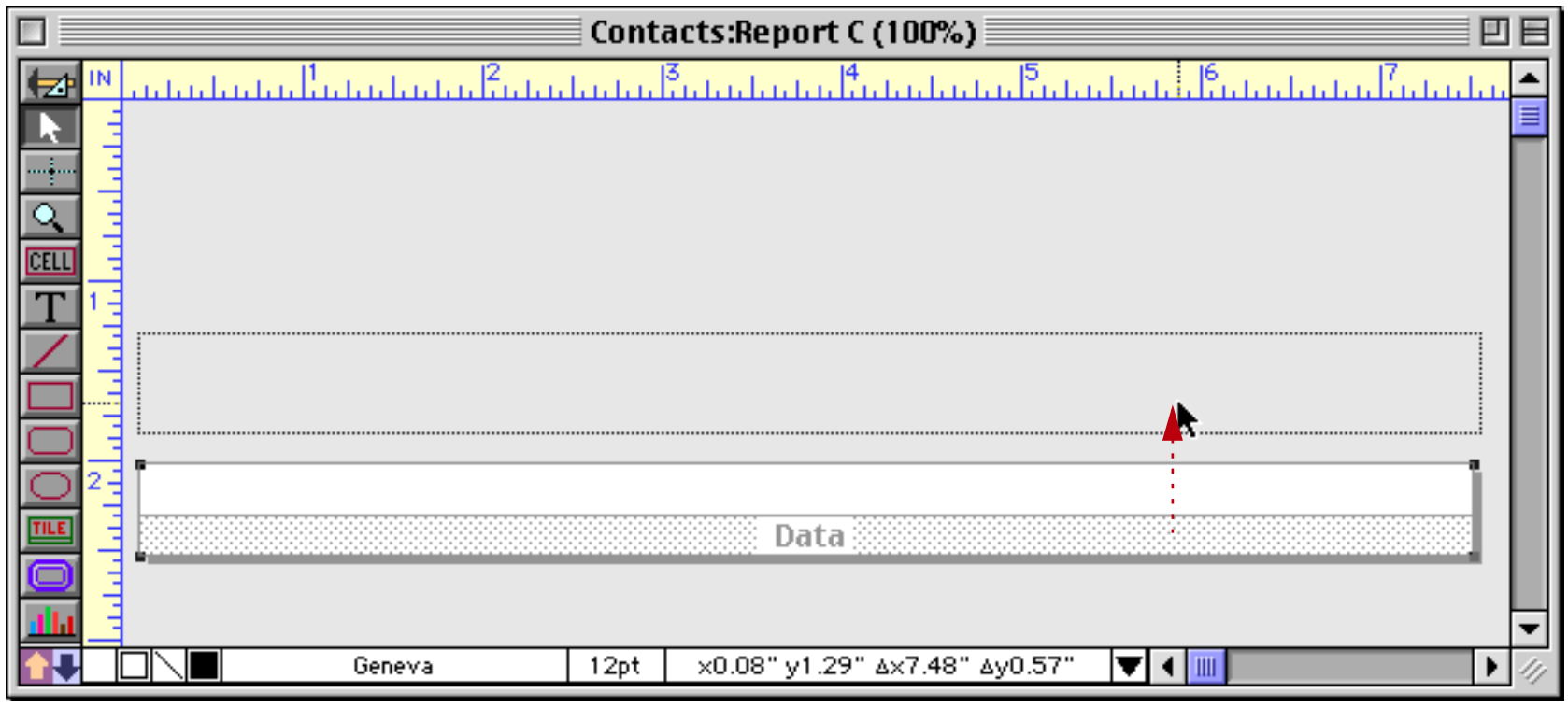


Creating A New Tile By Duplicating

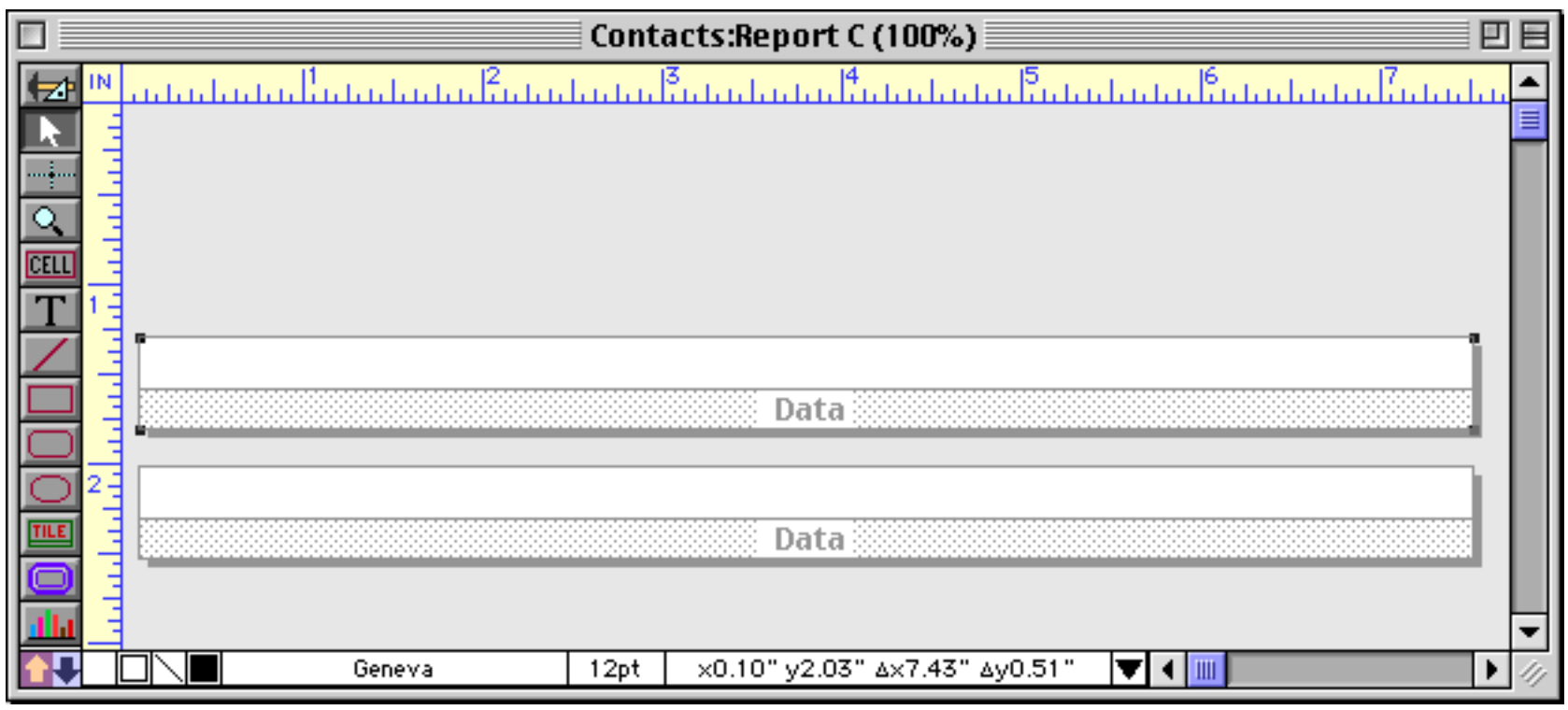
Another way to create a new tile is by duplicating an existing tile. For example, suppose you started with a data tile like this an wished to create a matching header tile.



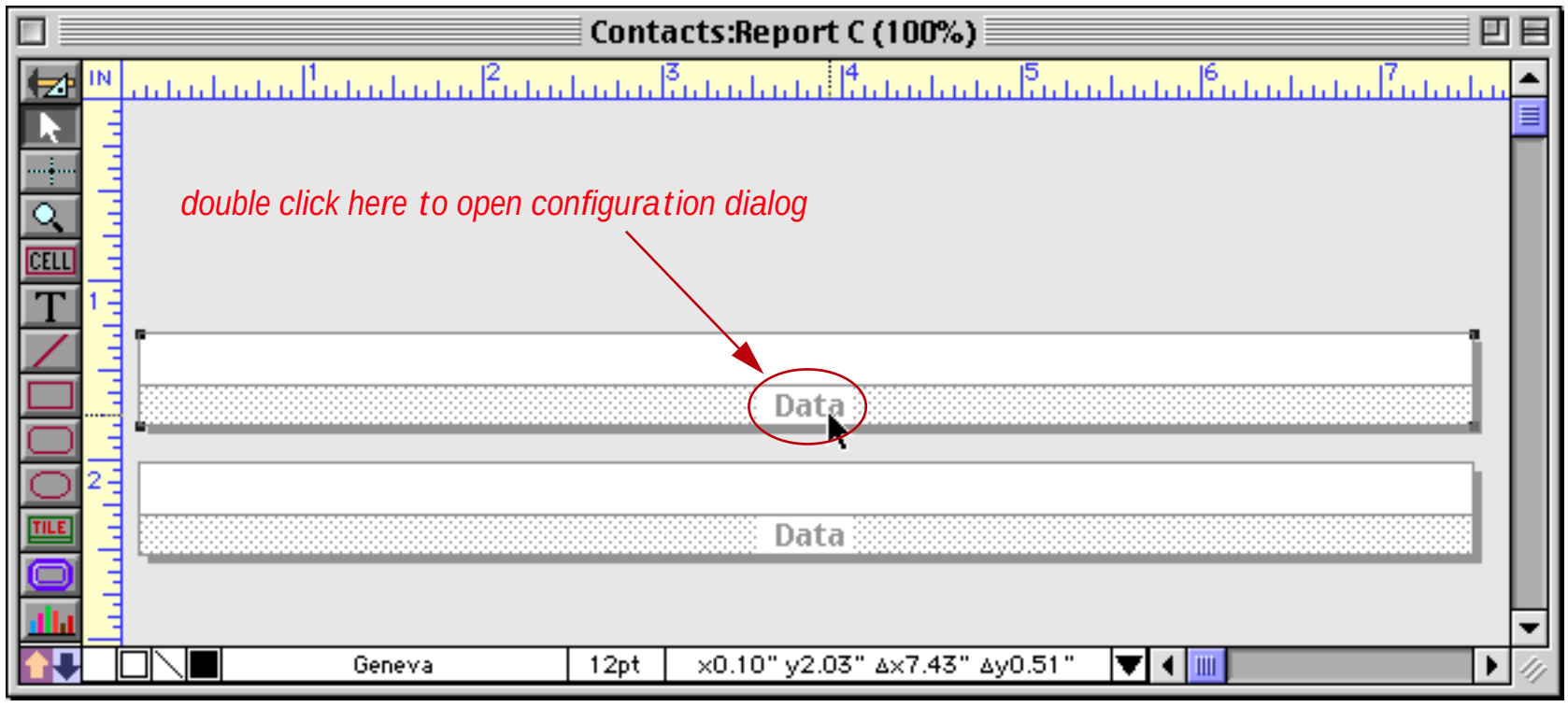
To duplicate this tile, hold down the Option key (Macintosh) or Alt key (PC), then click on the tile's drag bar and drag it (see "Drag Duplicating" on page 561). You may also want to hold down the **Shift** key to keep the new tile perfectly aligned with the original.



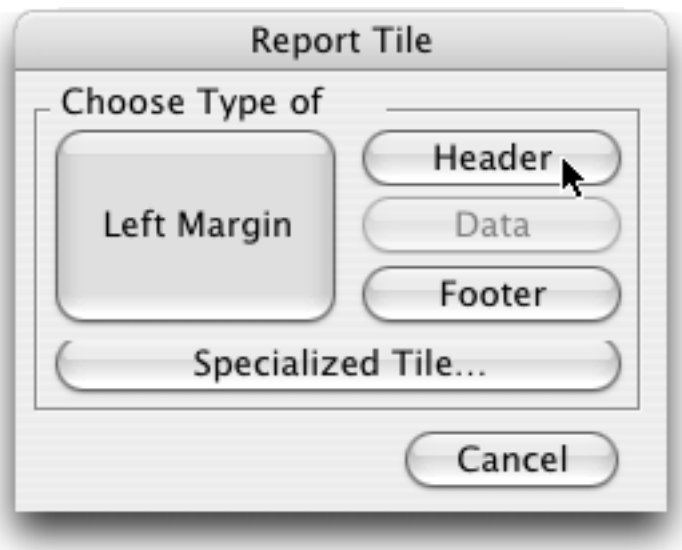
When you release the mouse a second data tile will appear.



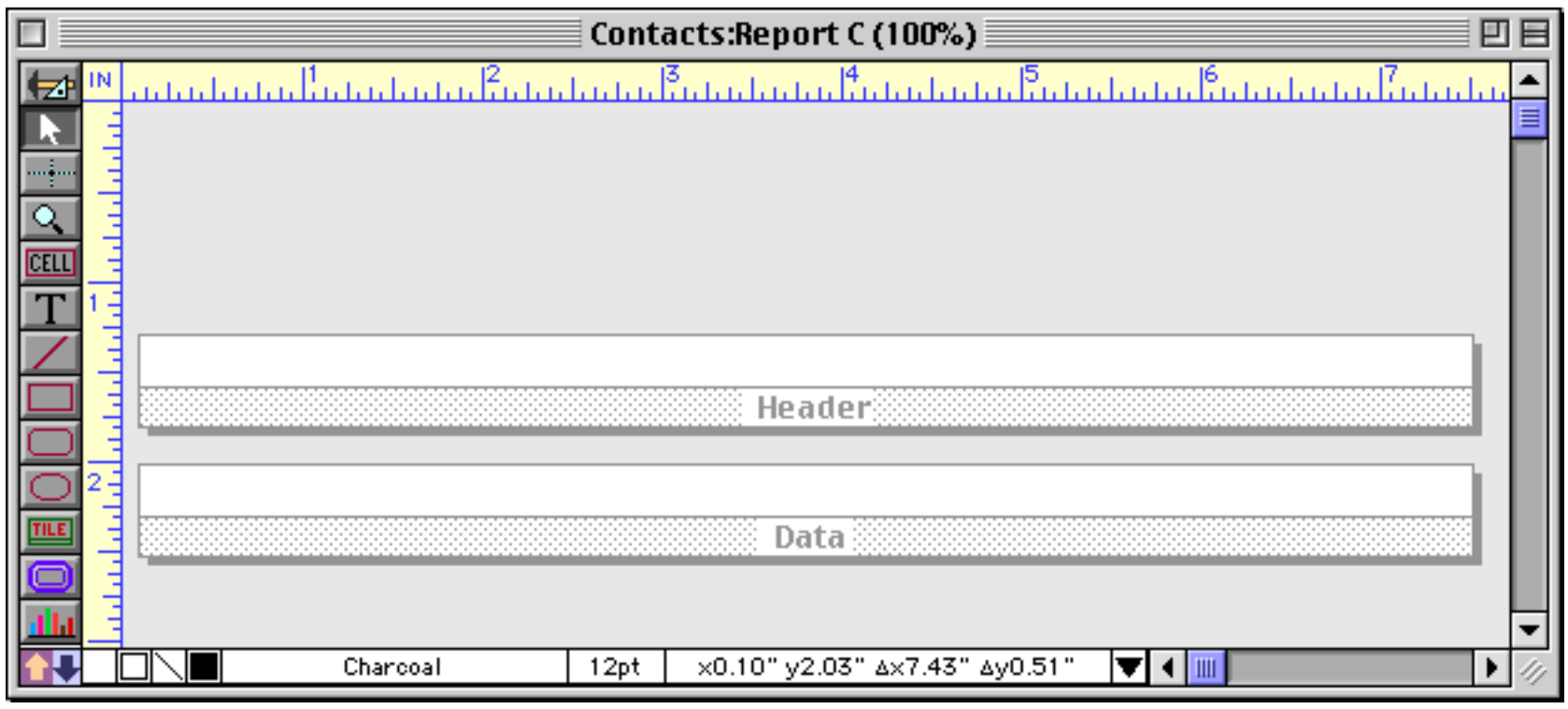
To change the type of the new tile, double click on the type of tile.



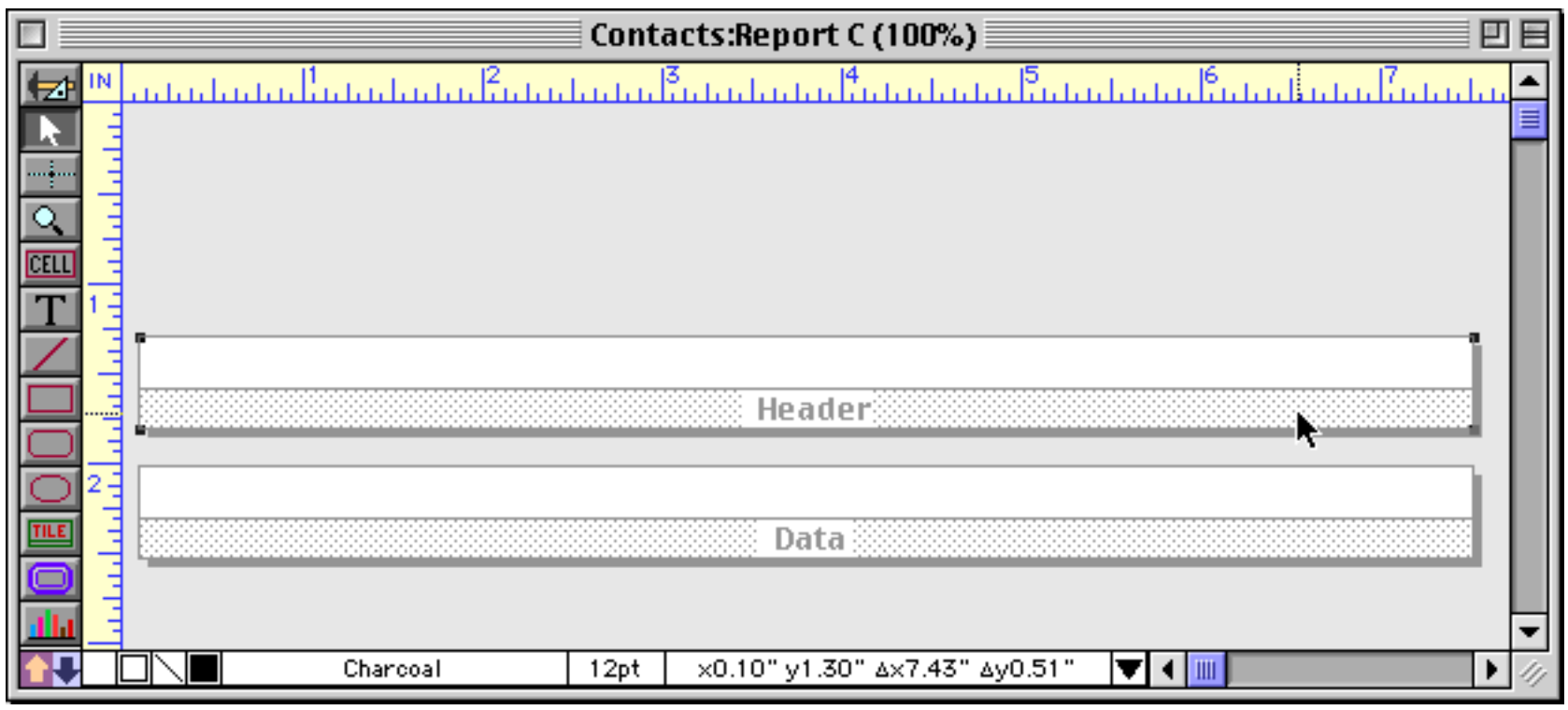
This opens the tile configuration dialog.



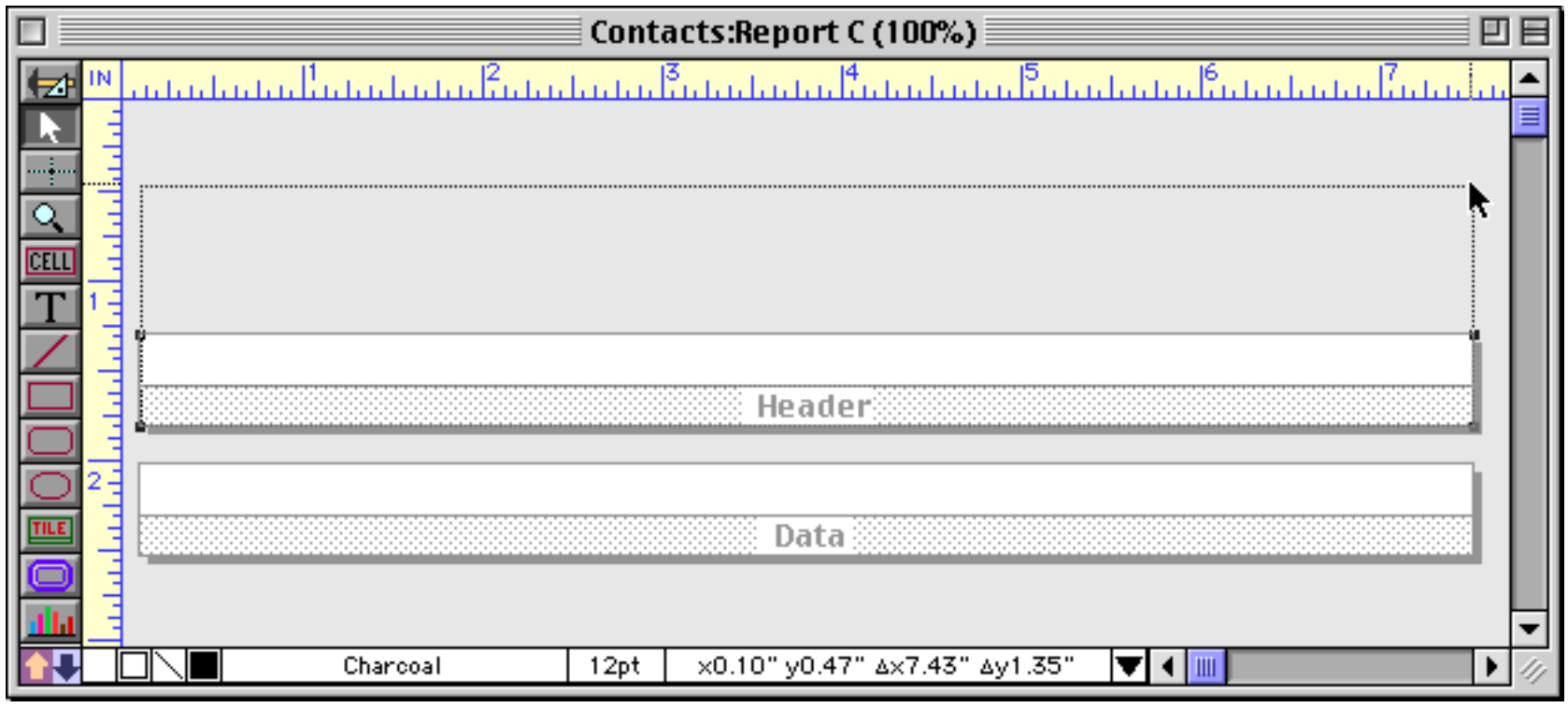
Press **Header** to switch the type of the new tile.



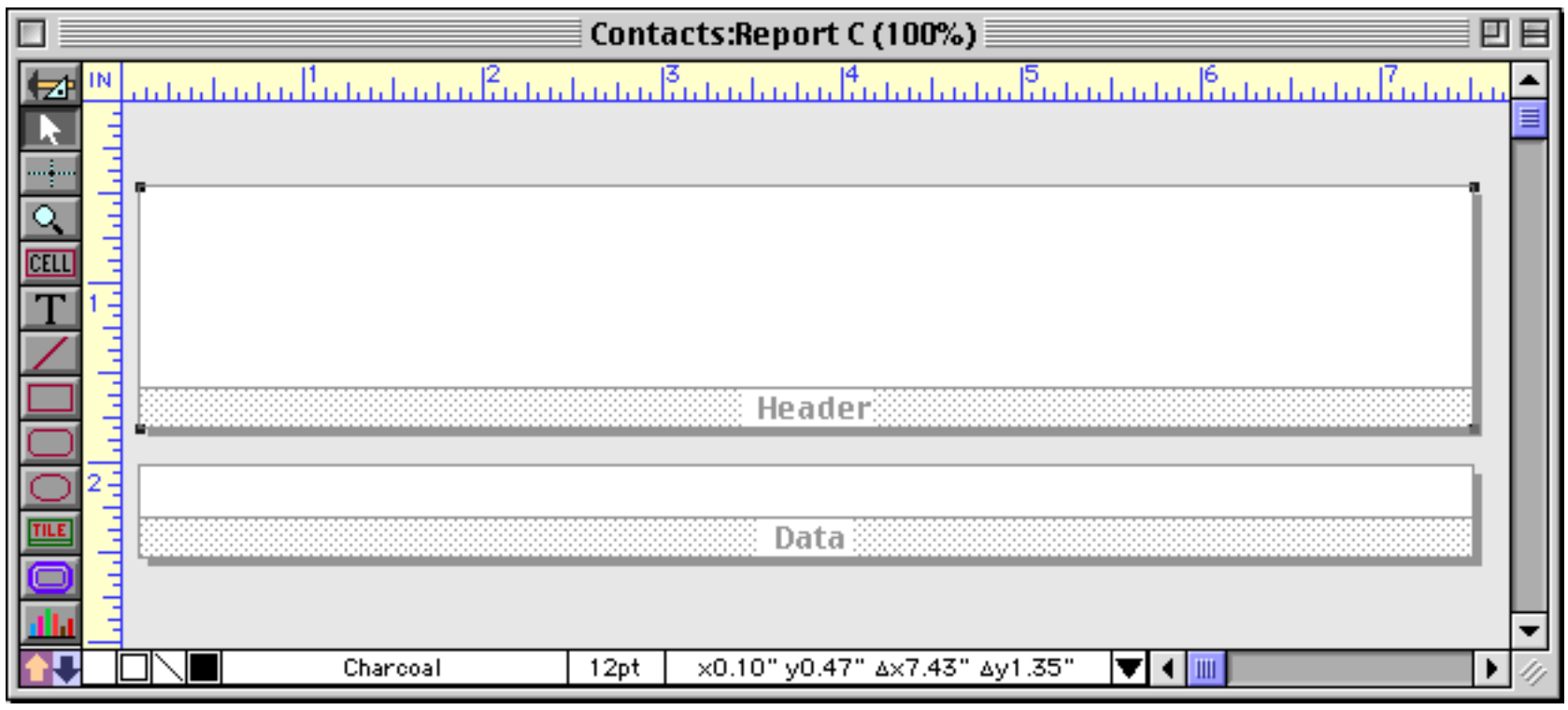
You may want to increase the height of the header tile. Here's one way to do that. Start by clicking on the drag bar to select the tile.



Now drag one of the handles to set the new size. Hold down the **Shift** key if you want to keep the width constant.



When you release the mouse button the tile snaps to the new height.



Of course the tile can also be moved or resized with the **Dimensions** command (see "Viewing and Setting Exact Object Dimensions" on page 512) and by nudging with the arrow keys (see "Nudging an Object (or Objects)" on page 509 and "Nudging the Size of an Object" on page 513).

Tiles In Action

Panorama has many different types of tiles. Each type of tile has its own rules that tell Panorama where the tile should be printed on the page. For example, a header tile is always printed at the top of the page. You can make the header bigger or smaller, add fancy graphics, even move it to a different spot in the form window—but no matter what you do, the header will always print at the top of the page.

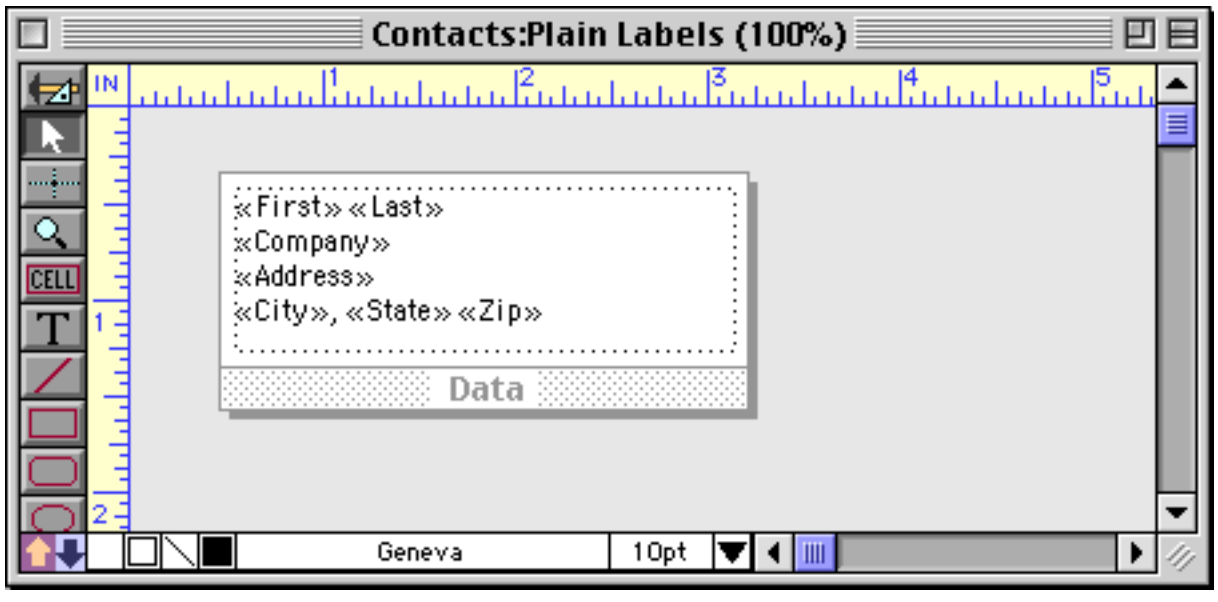
Panorama's basic rules for printing tiles are simple. First it prints the header (if any) at the top of the page, then the footer at the bottom of the page. The space left in the middle is available for data. Panorama starts in the upper left hand corner and prints a column of data—one data tile at a time. Once Panorama reaches the bottom, it checks to see if it should print another column. If another column is needed, Panorama goes back to the top and prints it.

More complex reports can have variations on this basic theme. Panorama has many different specialized types of tiles for automatically creating almost any report format. But the basic idea is the same—each tile slides into place on the page according to its rules.

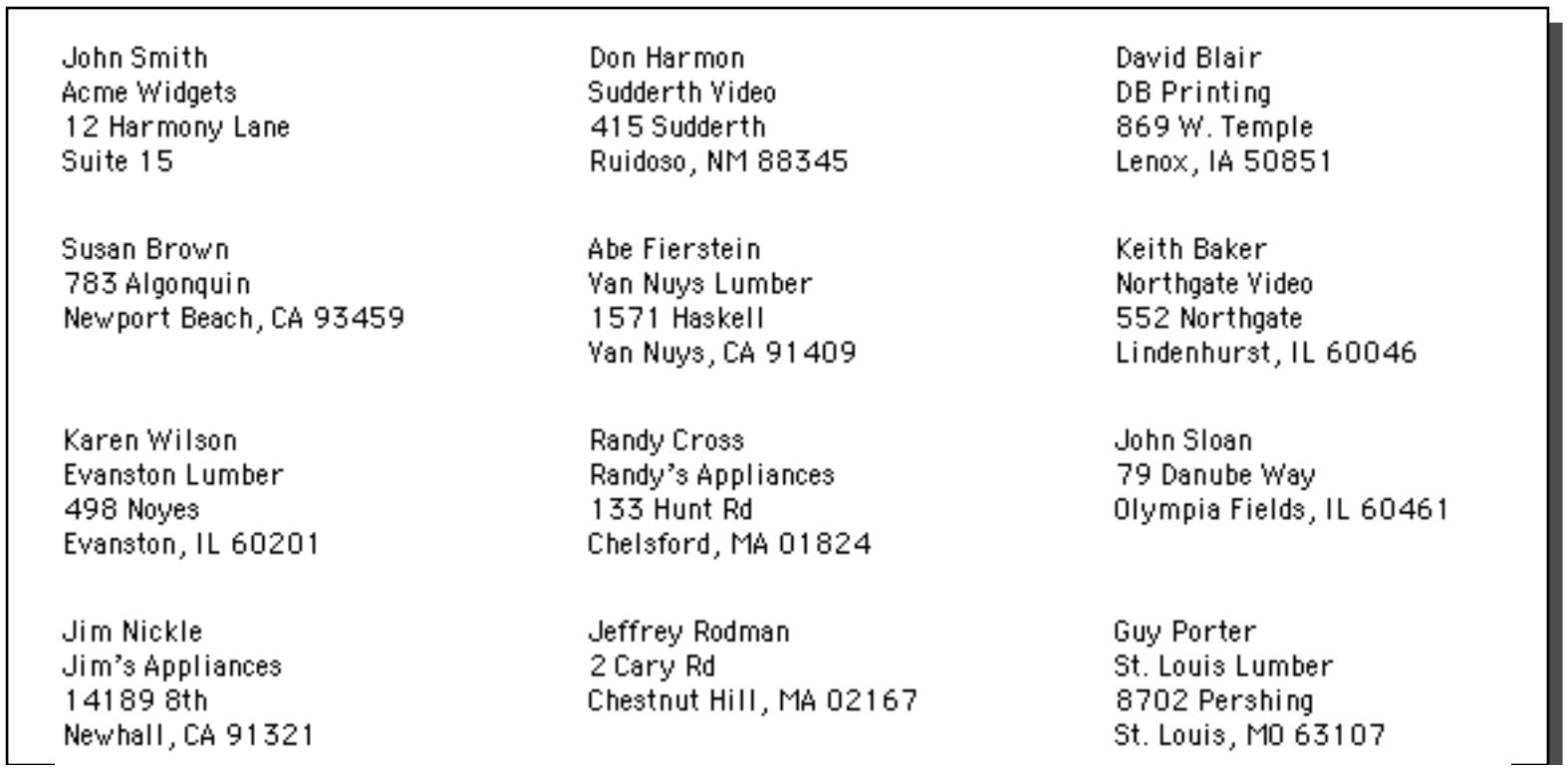
A form should contain no more than one tile of each type. If a form contains two header tiles, for example, Panorama will not know which one to print at the top of the page. If you attempt to print using a form that has duplicate tiles, Panorama will display an alert.

Data Tiles

The data tile is the cornerstone of every custom report. Because the data tile is used to print each data record, the size and shape of the data tile has a major impact on the overall look of the report. For instance, if you want to print labels, the data tile should be the same size and shape as a single label plus the gaps between the labels, like this.



When you print this form it will look like this.



John Smith
Acme Widgets
12 Harmony Lane
Suite 15

Don Harmon
Sudderth Video
415 Sudderth
Ruidoso, NM 88345

David Blair
DB Printing
869 W. Temple
Lenox, IA 50851

Susan Brown
783 Algonquin
Newport Beach, CA 93459

Abe Fierstein
Van Nuys Lumber
1571 Haskell
Van Nuys, CA 91409

Keith Baker
Northgate Video
552 Northgate
Lindenhurst, IL 60046

Karen Wilson
Evanston Lumber
498 Noyes
Evanston, IL 60201

Randy Cross
Randy's Appliances
133 Hunt Rd
Chelsford, MA 01824

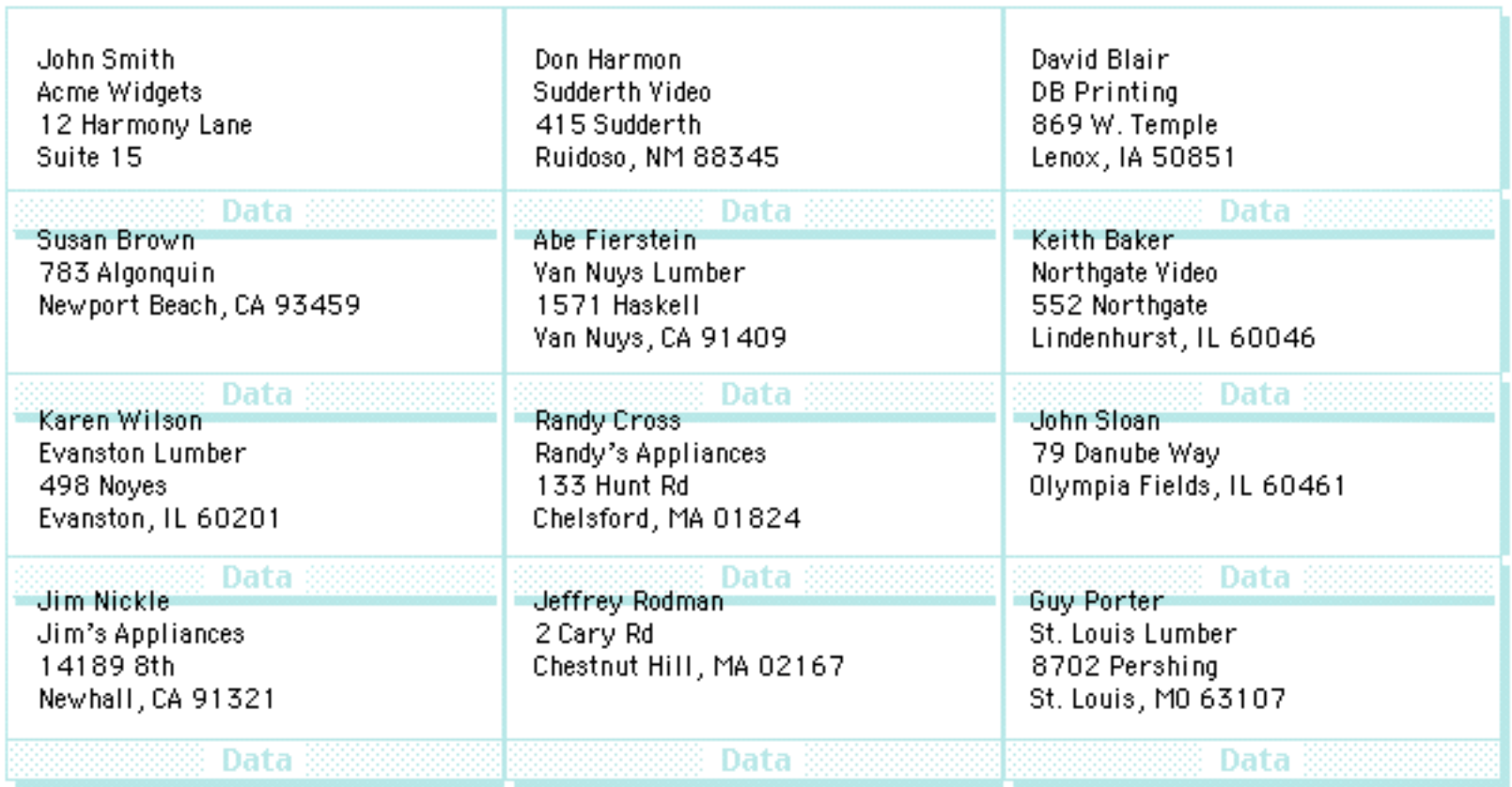
John Sloan
79 Danube Way
Olympia Fields, IL 60461

Jim Nickle
Jim's Appliances
14189 8th
Newhall, CA 91321

Jeffrey Rodman
2 Cary Rd
Chestnut Hill, MA 02167

Guy Porter
St. Louis Lumber
8702 Pershing
St. Louis, MO 63107

This illustration shows how Panorama stacked the tiles together to create the report shown above. In this illustration, the tiles, which are actually invisible, are shown in light blue. As you can see, Panorama slides the tile surfaces together as closely as possible. (The tile drag bars are not counted as part of the tile when the report is printed.)



John Smith
Acme Widgets
12 Harmony Lane
Suite 15

Data

Don Harmon
Sudderth Video
415 Sudderth
Ruidoso, NM 88345

Data

David Blair
DB Printing
869 W. Temple
Lenox, IA 50851

Data

Susan Brown
783 Algonquin
Newport Beach, CA 93459

Data

Abe Fierstein
Van Nuys Lumber
1571 Haskell
Van Nuys, CA 91409

Data

Keith Baker
Northgate Video
552 Northgate
Lindenhurst, IL 60046

Data

Karen Wilson
Evanston Lumber
498 Noyes
Evanston, IL 60201

Data

Randy Cross
Randy's Appliances
133 Hunt Rd
Chelsford, MA 01824

Data

John Sloan
79 Danube Way
Olympia Fields, IL 60461

Data

Jim Nickle
Jim's Appliances
14189 8th
Newhall, CA 91321

Data

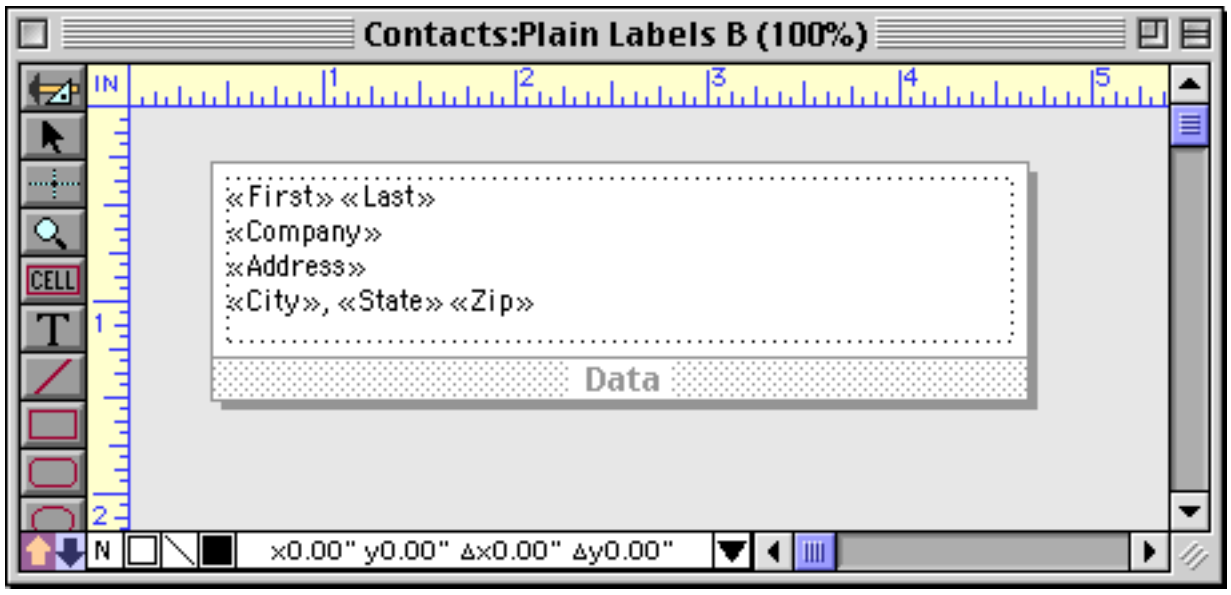
Jeffrey Rodman
2 Cary Rd
Chestnut Hill, MA 02167

Data

Guy Porter
St. Louis Lumber
8702 Pershing
St. Louis, MO 63107

Data

Simply changing the size or shape of the data tile can completely change the printed report.



The only change is that the data tile (and the text object on it) has been made wider. Now Panorama can only stack two data tiles side by side.

John Smith
Acme Widgets
12 Harmony Lane
Suite 15

Don Harmon
Sudderth Video
415 Sudderth
Ruidoso, NM 88345

Susan Brown
783 Algonquin
Newport Beach, CA 93459

Abe Fierstein
Van Nuys Lumber
1571 Haskell
Van Nuys, CA 91409

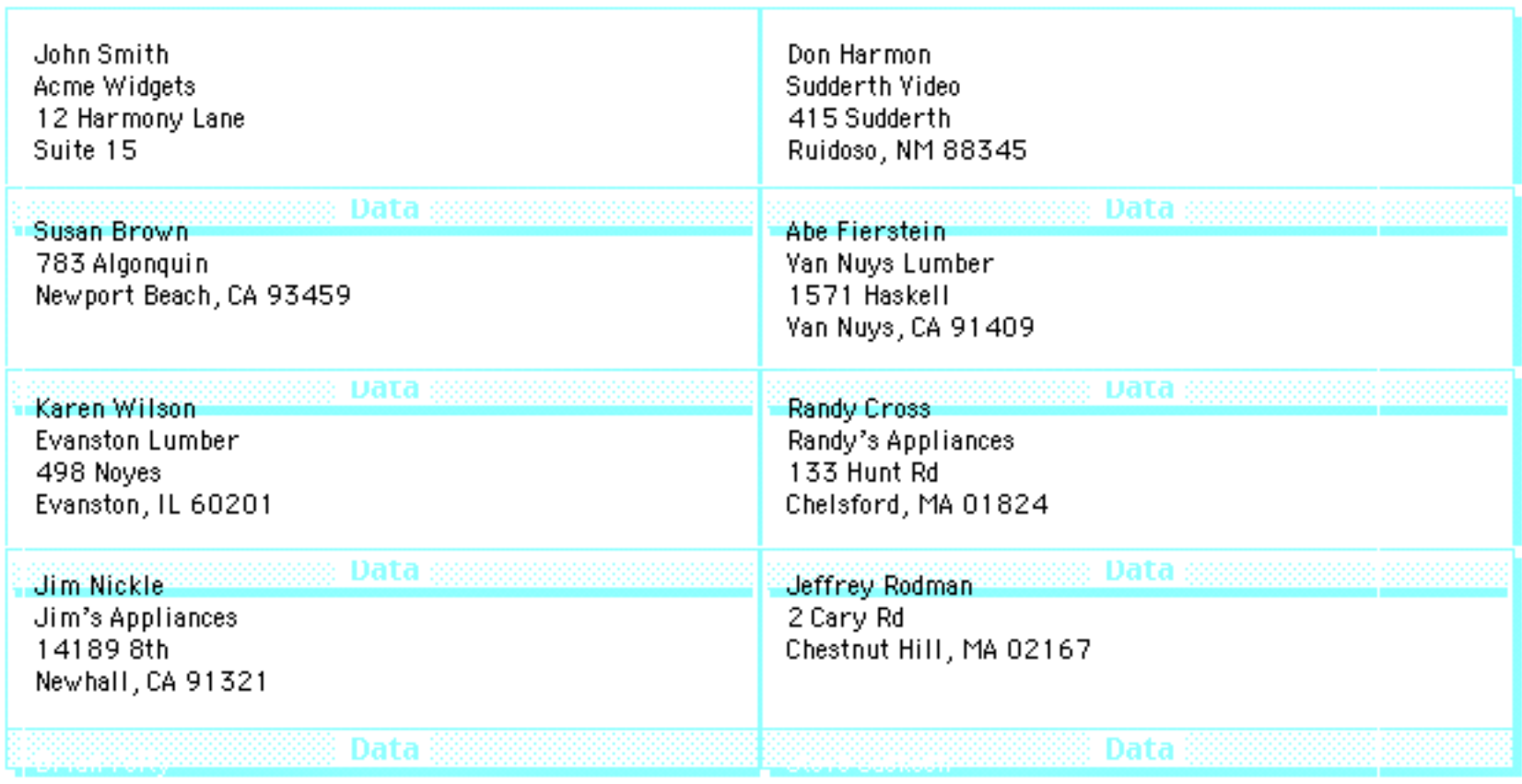
Karen Wilson
Evanston Lumber
498 Noyes
Evanston, IL 60201

Randy Cross
Randy's Appliances
133 Hunt Rd
Chelsford, MA 01824

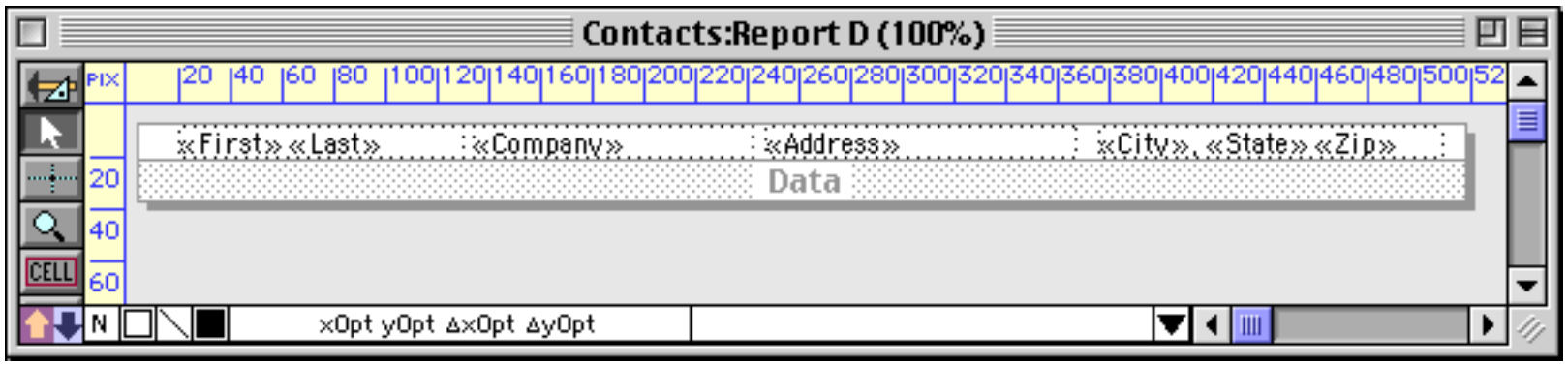
Jim Nickle
Jim's Appliances
14189 8th
Newhall, CA 91321

Jeffrey Rodman
2 Cary Rd
Chestnut Hill, MA 02167

Once again here is an illustration that shows how Panorama fits the invisible tiles together to make the finished report.



To make a columnar report with one line per record, make the data tile as wide as the page and one line high, like this.



Here's the finished report printed from this tile.

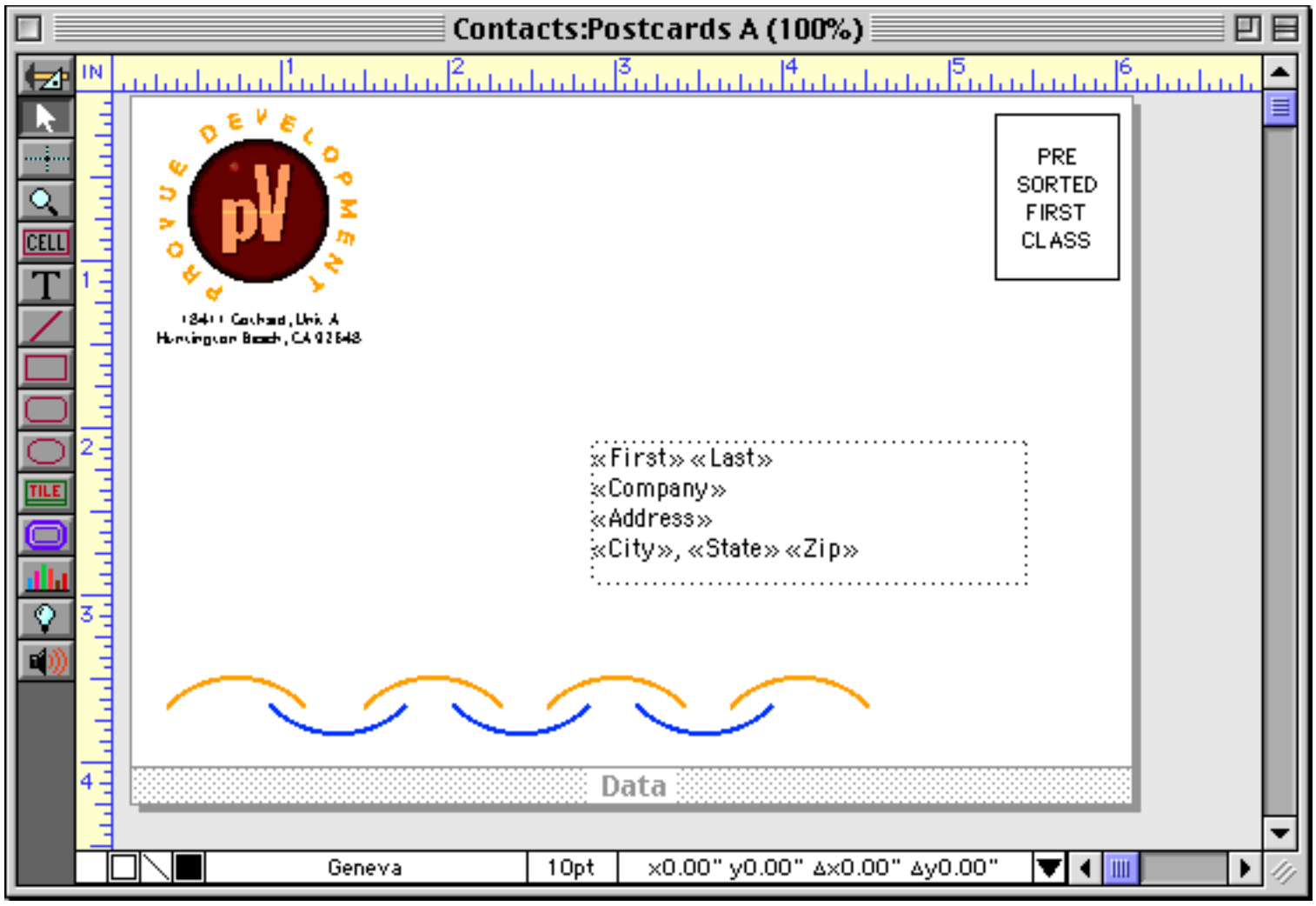
John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091
Ed Ruth	Chicago Lumber	1580 N. Oconto	Chicago, IL 60634
Don Harmon	Sudderth Video	415 Sudderth	Ruidoso, NM 88345

This data tile is too wide to stack side-by-side, so Panorama simply stacks them vertically.

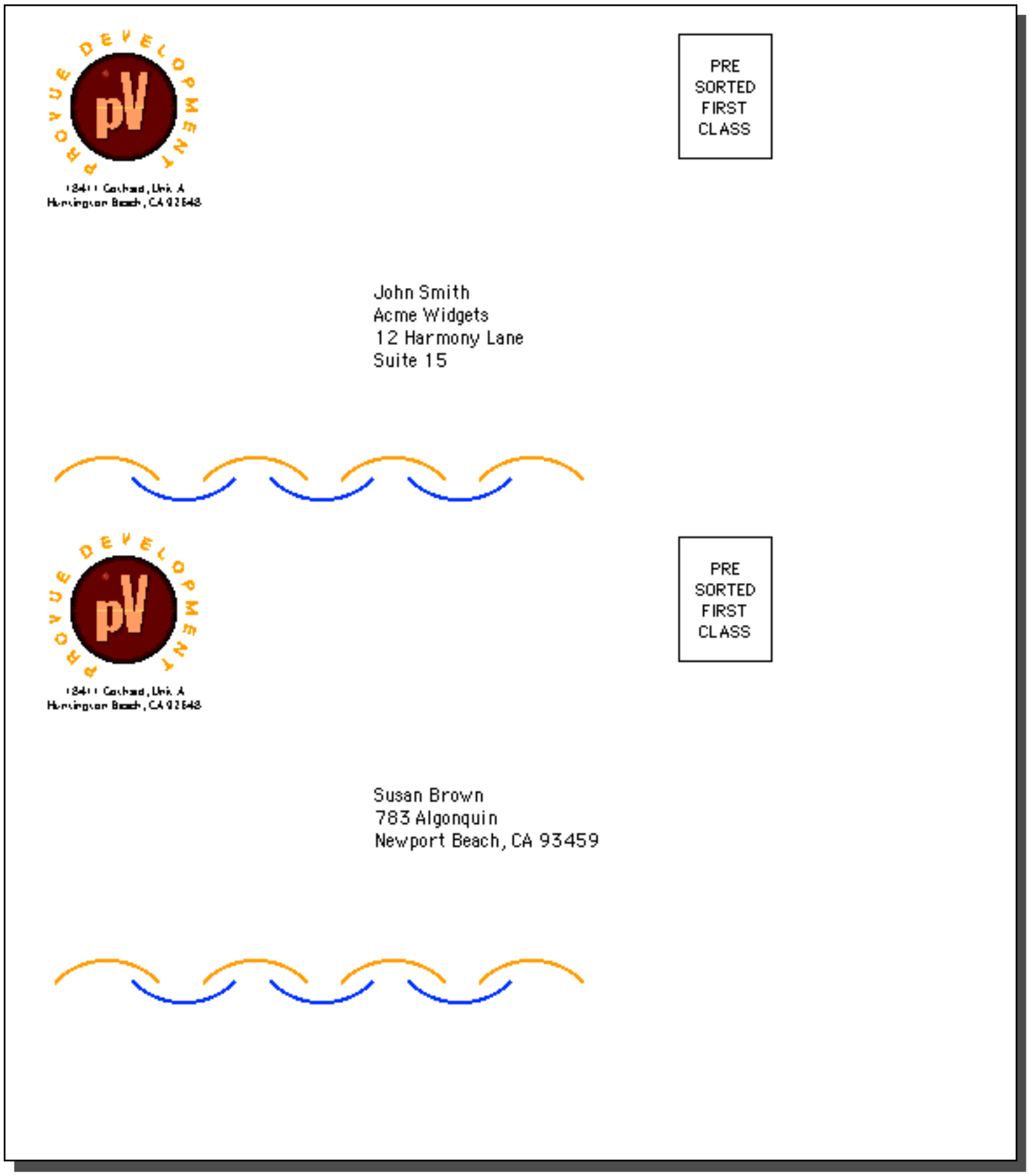
John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091
Ed Ruth	Chicago Lumber	1580 N. Oconto	Chicago, IL 60634
Don Harmon	Sudderth Video	415 Sudderth	Ruidoso, NM 88345

Data

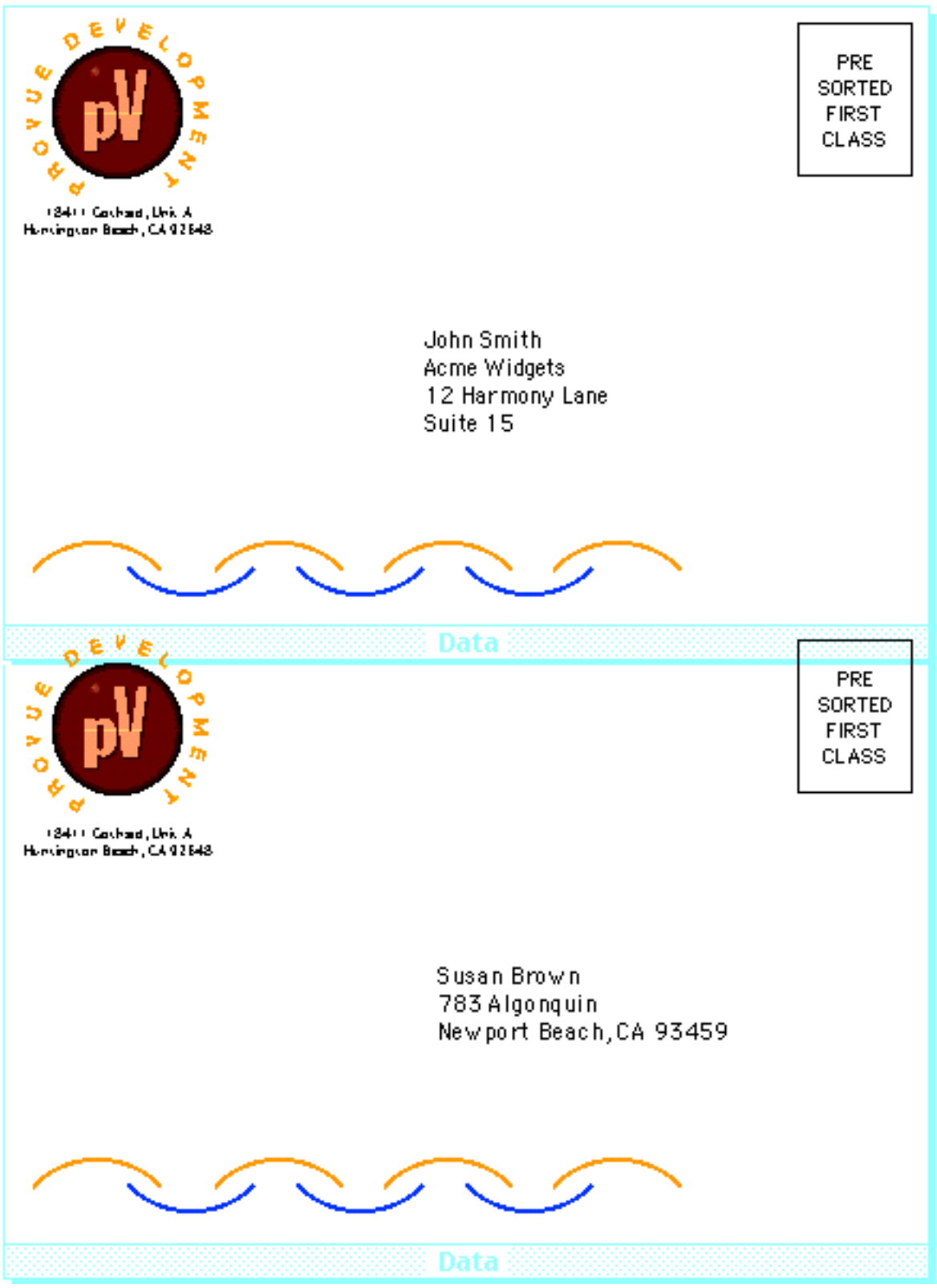
At the other extreme you can make the data tile very large so that only one or two records will be printed per page. This form is designed to print two postcards per page.



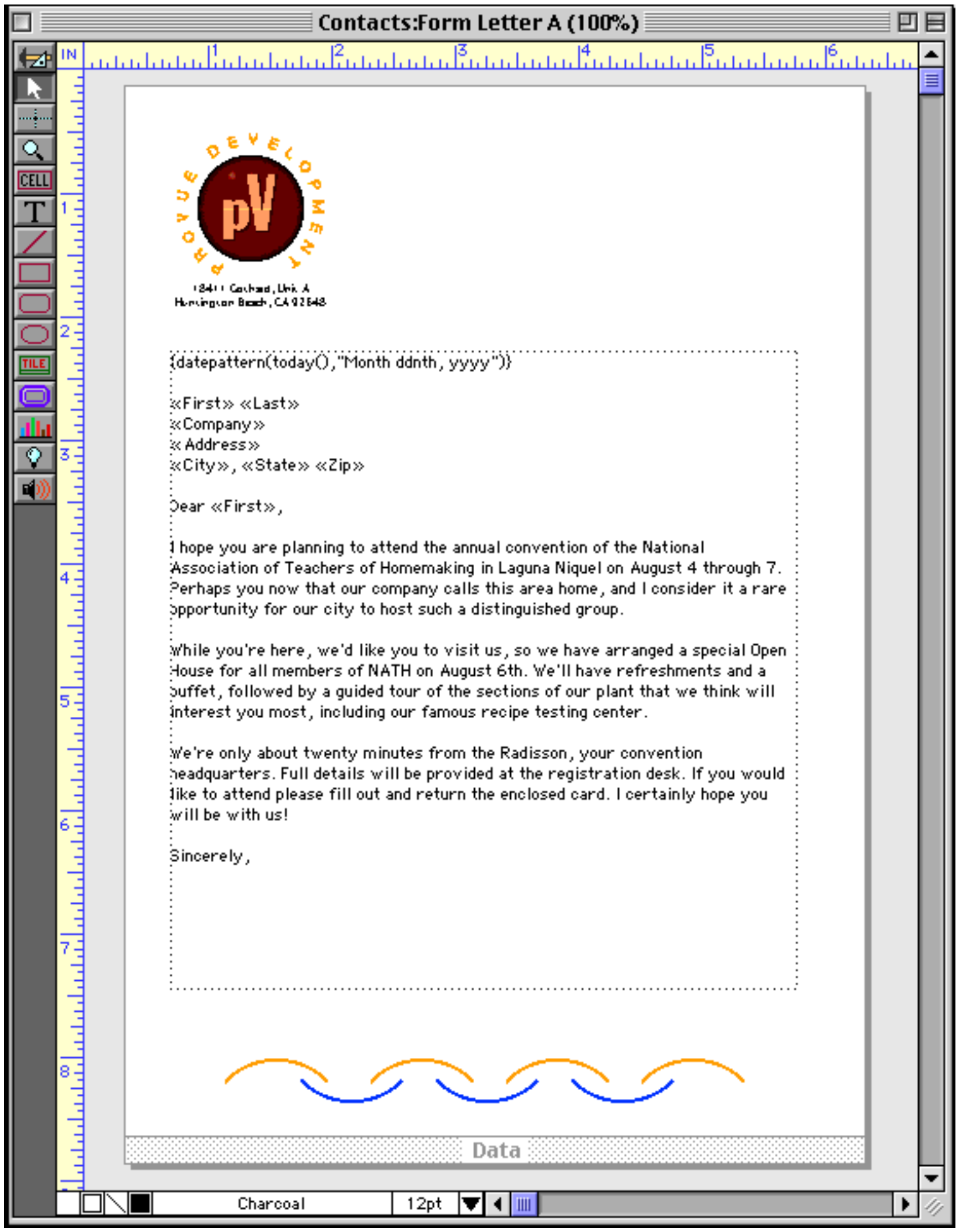
Here's what the postcards look like when printed on an 8.5 x 11 inch page.



Just as with the other reports, Panorama prints this one by stacking the data tiles as close together as possible.



If the data tile is large enough Panorama will only print one record per page. Common applications include invoices, statements, tax returns, and form letters like the one shown here.



Here's the printed result of this form. Panorama will print one complete page for each selected record in the database, each with a customized letter. (Or, if you wish, you can use the **Print One Record** tool to print a single letter. See "Print One Record" on page 1048)



PROVUE DEVELOPMENT

pV

12411 Cochran, Unit A
Huntington Beach, CA 92648

July 7th, 2000

John Smith
Acme Widgets
12 Harmony Lane
Suite 15
Huntington Beach, CA 92648

Dear John,

I hope you are planning to attend the annual convention of the National Association of Teachers of Homemaking in Laguna Niquel on August 4 through 7. Perhaps you now that our company calls this area home, and I consider it a rare opportunity for our city to host such a distinguished group.

While you're here, we'd like you to visit us, so we have arranged a special Open House for all members of NATH on August 6th. We'll have refreshments and a buffet, followed by a guided tour of the sections of our plant that we think will interest you most, including our famous recipe testing center.

We're only about twenty minutes from the Radisson, your convention headquarters. Full details will be provided at the registration desk. If you would like to attend please fill out and return the enclosed card. I certainly hope you will be with us!

Sincerely,

By the way, it's also possible to print multi-page letters. See "Printing Data that Overflows a Page" on page 1104.

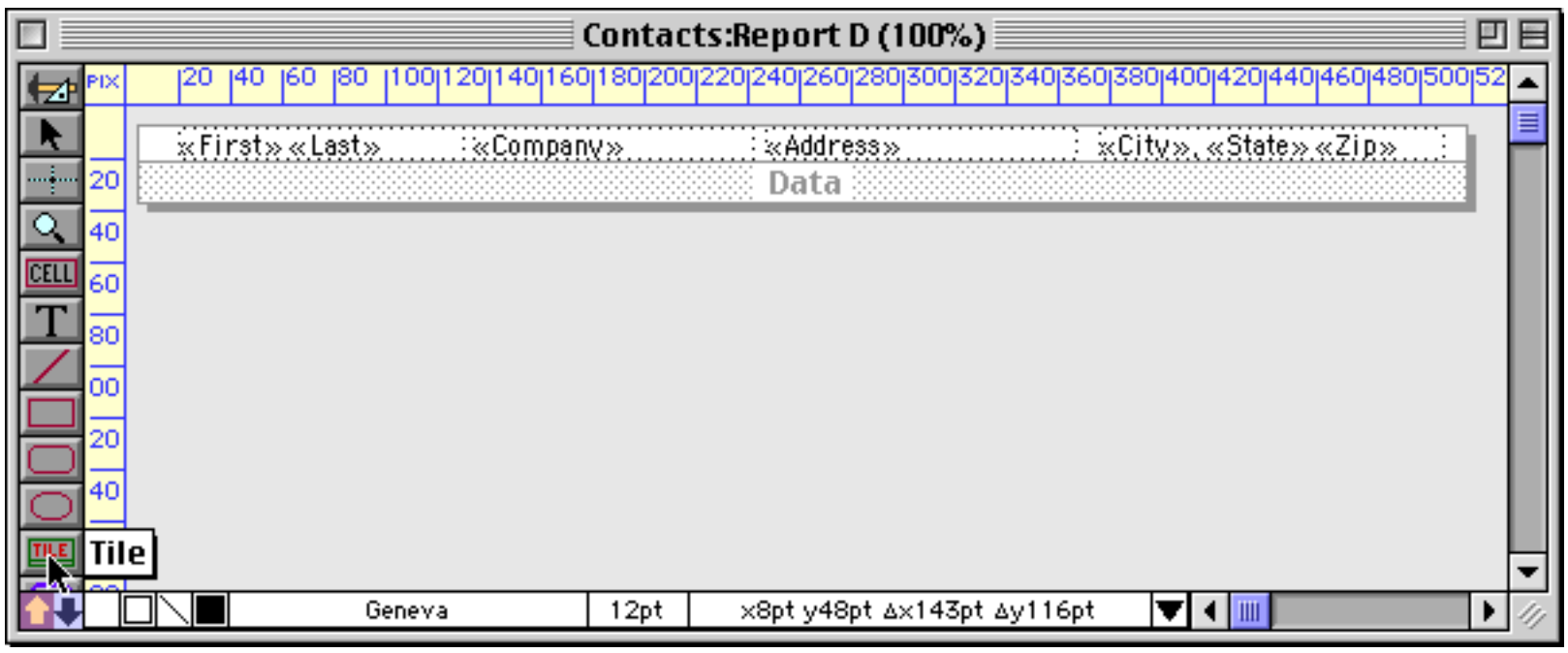
Margins

If a form doesn't contain any margin tiles (like the examples in the previous section) Panorama will automatically start printing as far up and to the left as possible on the current printer. In this case the exact margins will depend on the printer and on the Page Setup options you have chosen.

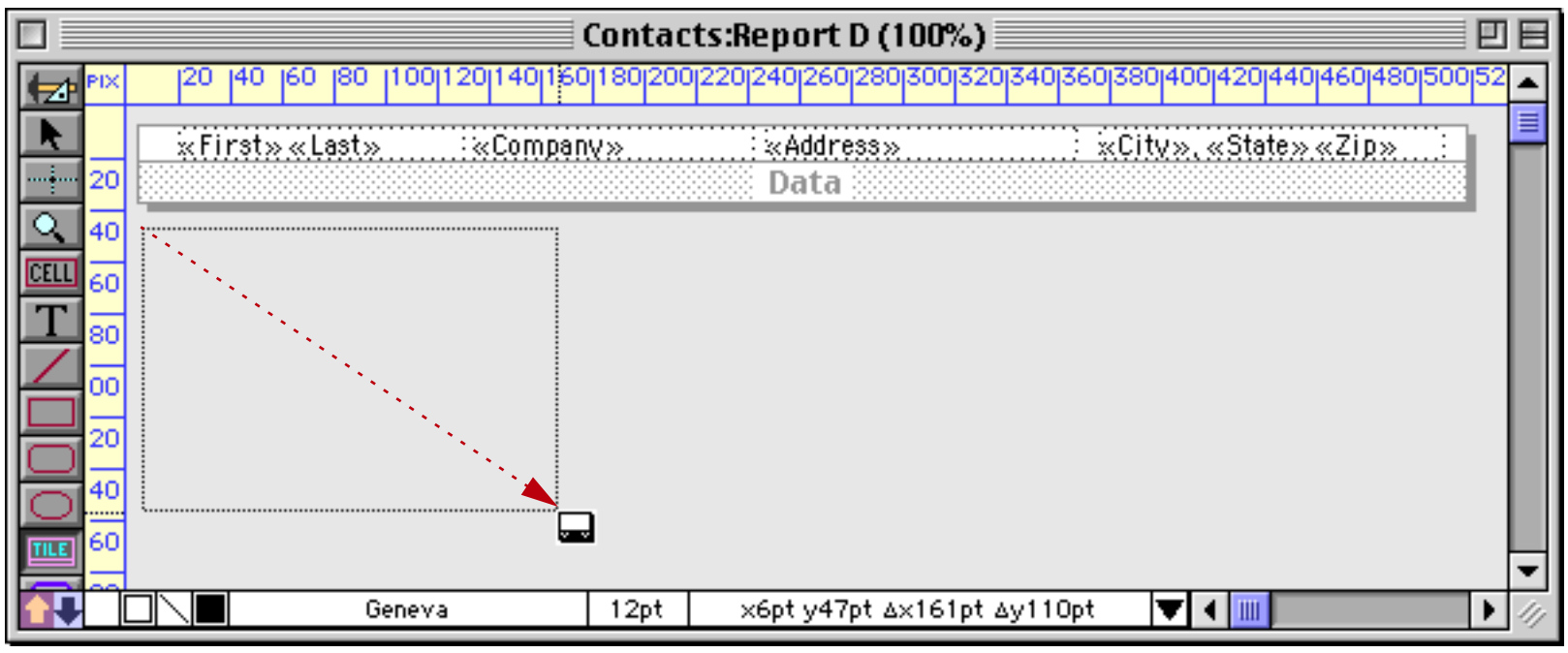
Top Margin Tile

The top margin tile allows you to modify the distance from the top of the page to the top of the first tile printed. Unlike most other tiles where both dimensions are important, Panorama only cares about the height of the top margin tile. If the report contains a top margin tile, the height of that tile becomes the top margin of the report.

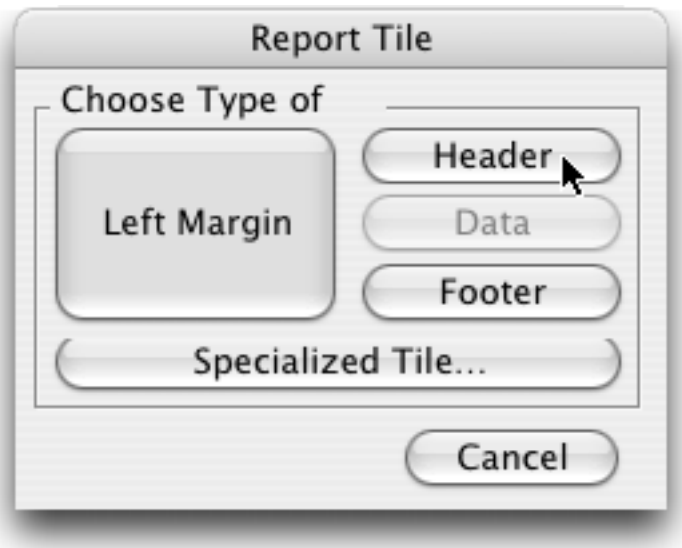
To create a top margin, start by selecting the **Tile** tool.



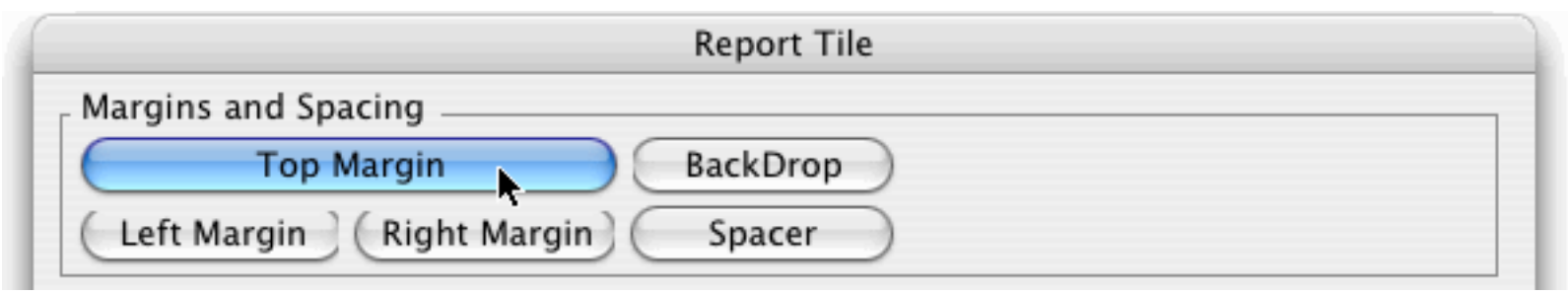
Drag the mouse over an empty spot on the form. You don't need to worry about the width, but the height of this object will become the height of the top margin. Of course you can adjust the dimensions later.



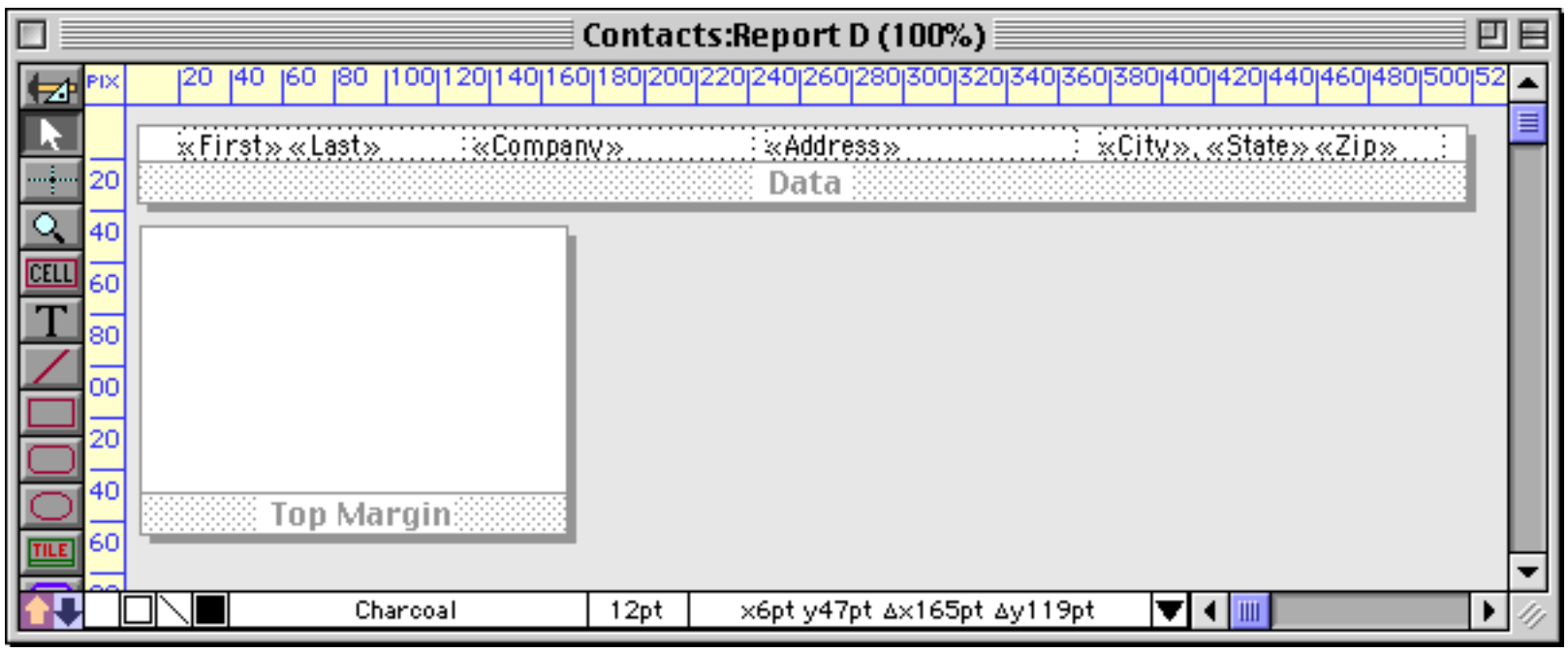
When you release the mouse the tile configuration dialog appears. If the small configuration dialog appears (as shown below) then click on the **Specialized Tiles** button.



The **Top Margin** button is near the top of the dialog.



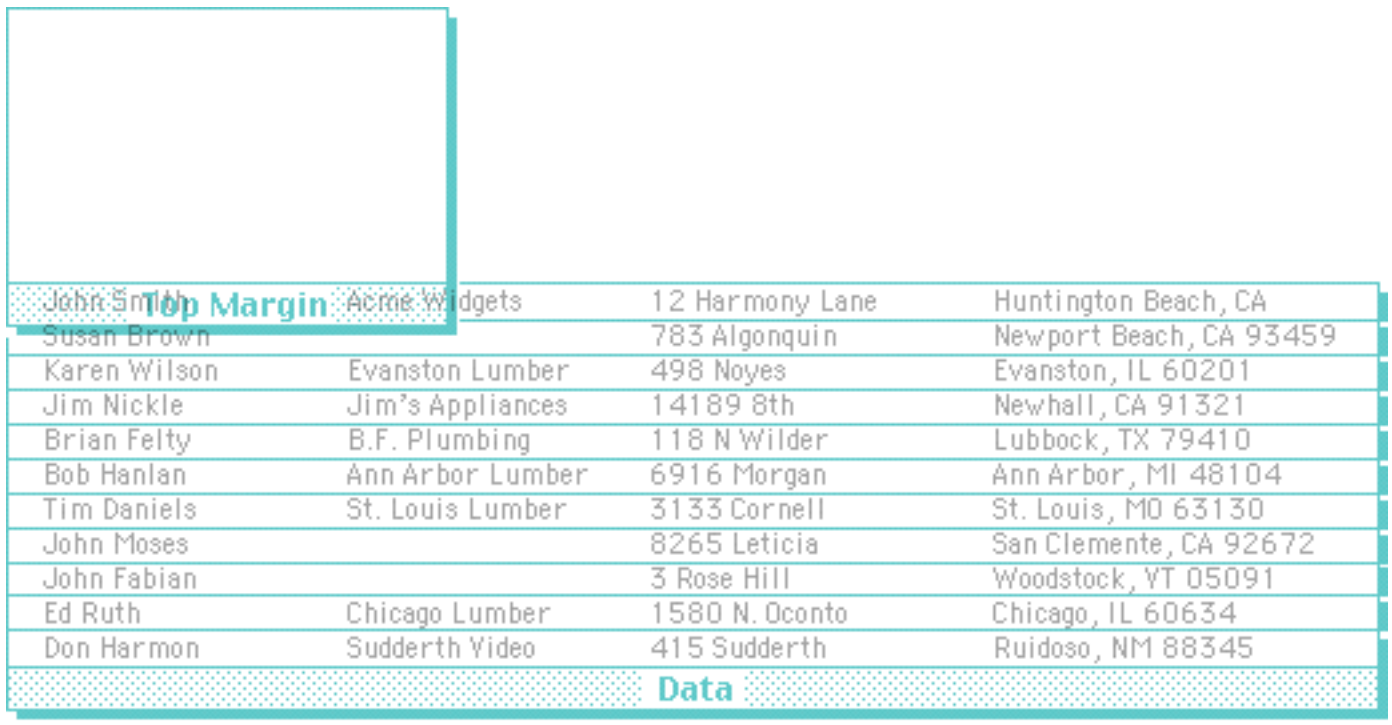
Press this button to create the new tile.



When it's printed this report will look like this.

John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091
Ed Ruth	Chicago Lumber	1580 N. Oconto	Chicago, IL 60634
Don Harmon	Sudderth Video	415 Sudderth	Ruidoso, NM 88345

This illustration shows how Panorama assembles the report using the top margin and data tiles. The top margin is always the top tile on the page, with any other tiles stacked below it.



Unlike other tiles, Panorama will not print text or graphics that are placed on the surface of the top margin tile. The top margin tile has only one purpose—to set the top margin.

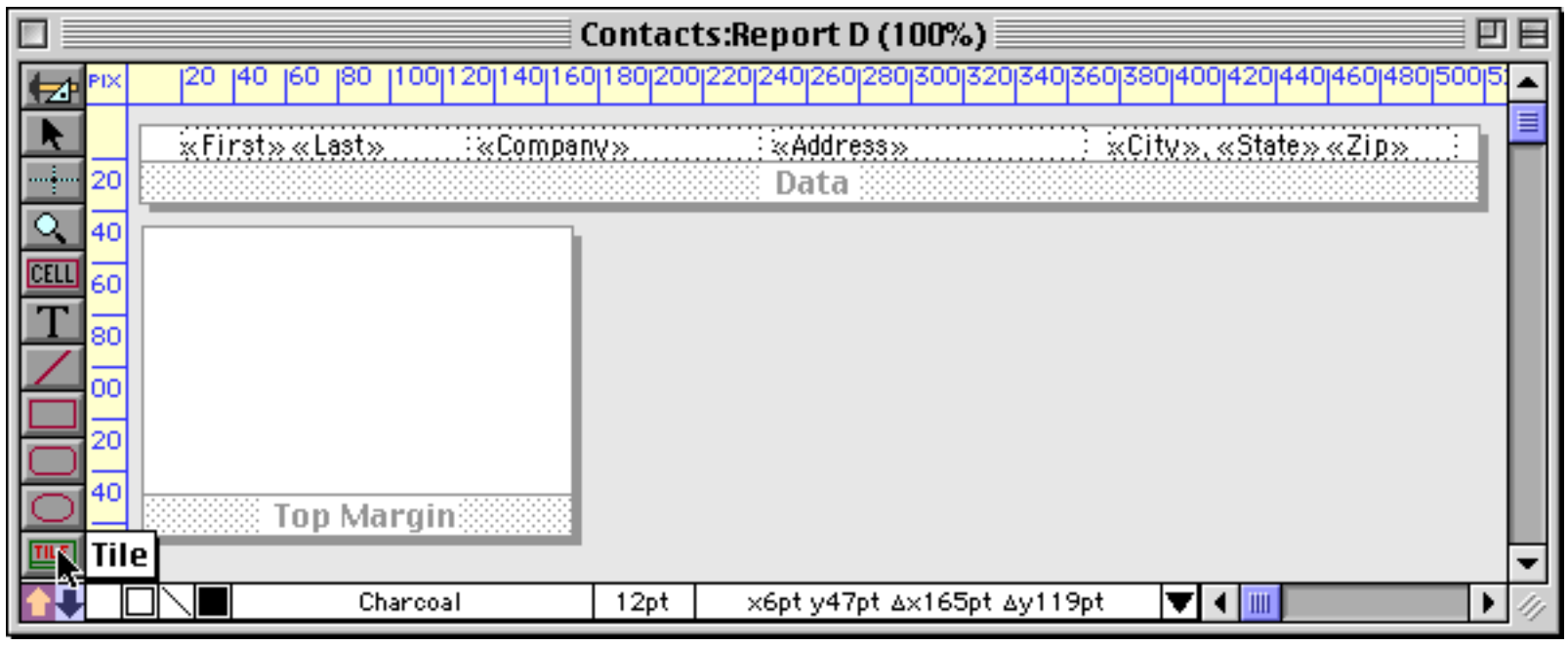
If the height of the top margin tile is less than the minimum printer margin the text may be cut off.

Left Margin Tile

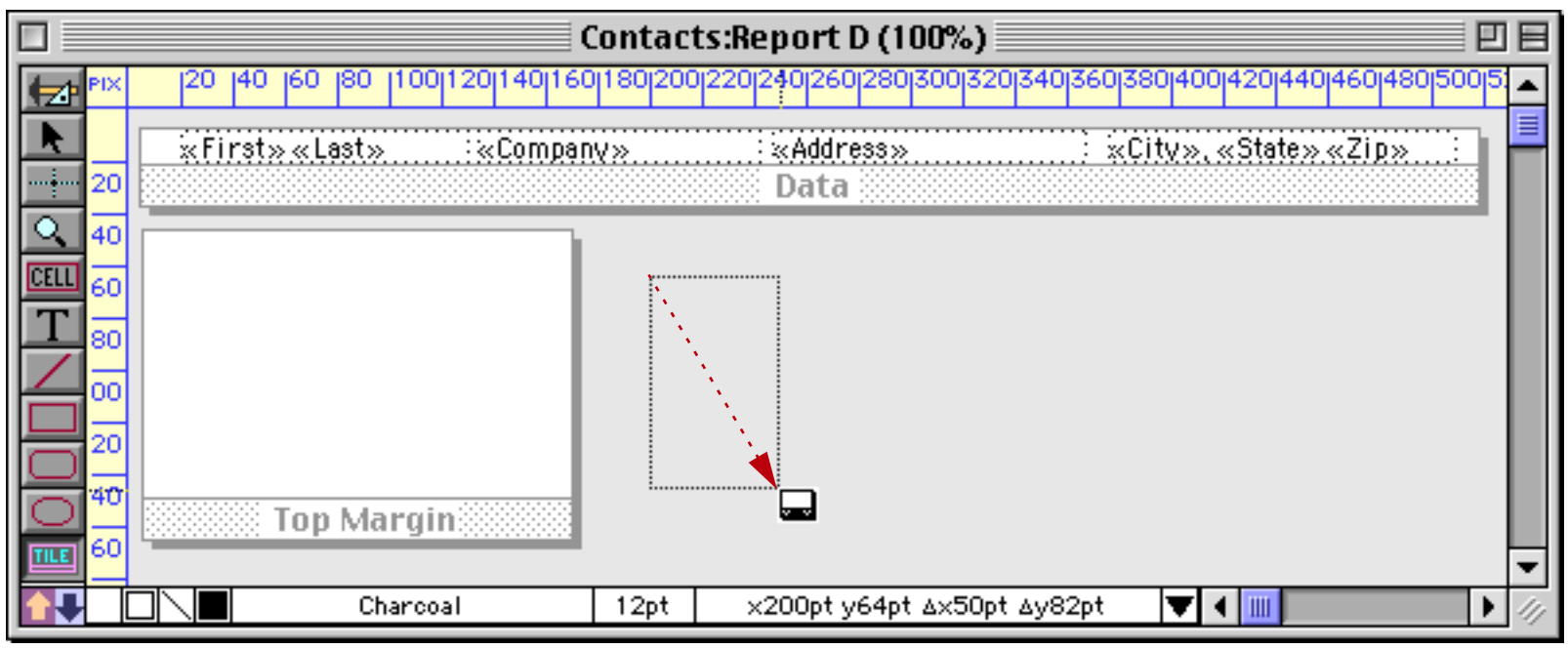
Panorama normally prints the header, footer, and data tiles with the smallest possible left margin, usually about 1⁄4 inch. (The actual minimum left margin depends on the type of printer you are using, and on the current **Page Setup**.)

You can use the left margin tile to change the left margin. Unlike other tiles where both dimensions are important, Panorama only cares about the width of the left margin tile. If the report contains a left margin tile, the width of that tile becomes the left margin of the report.

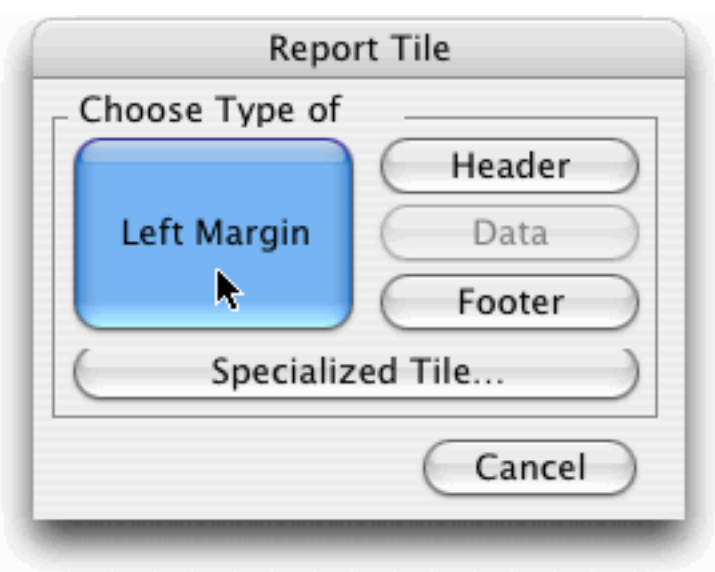
To create a left margin, start by selecting the **Tile** tool.



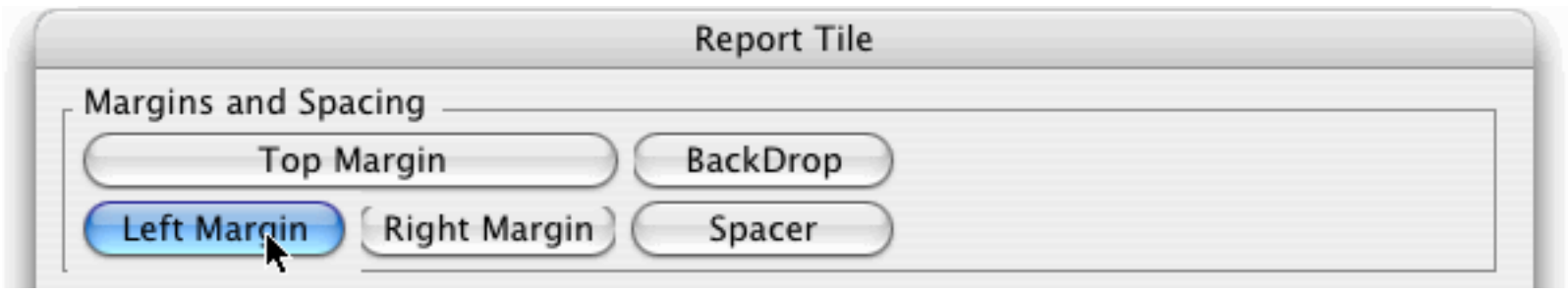
Drag the mouse over an empty spot on the form. You don't need to worry about the height, but the width of this object will become the width of the left margin. Of course you can adjust the dimensions later.



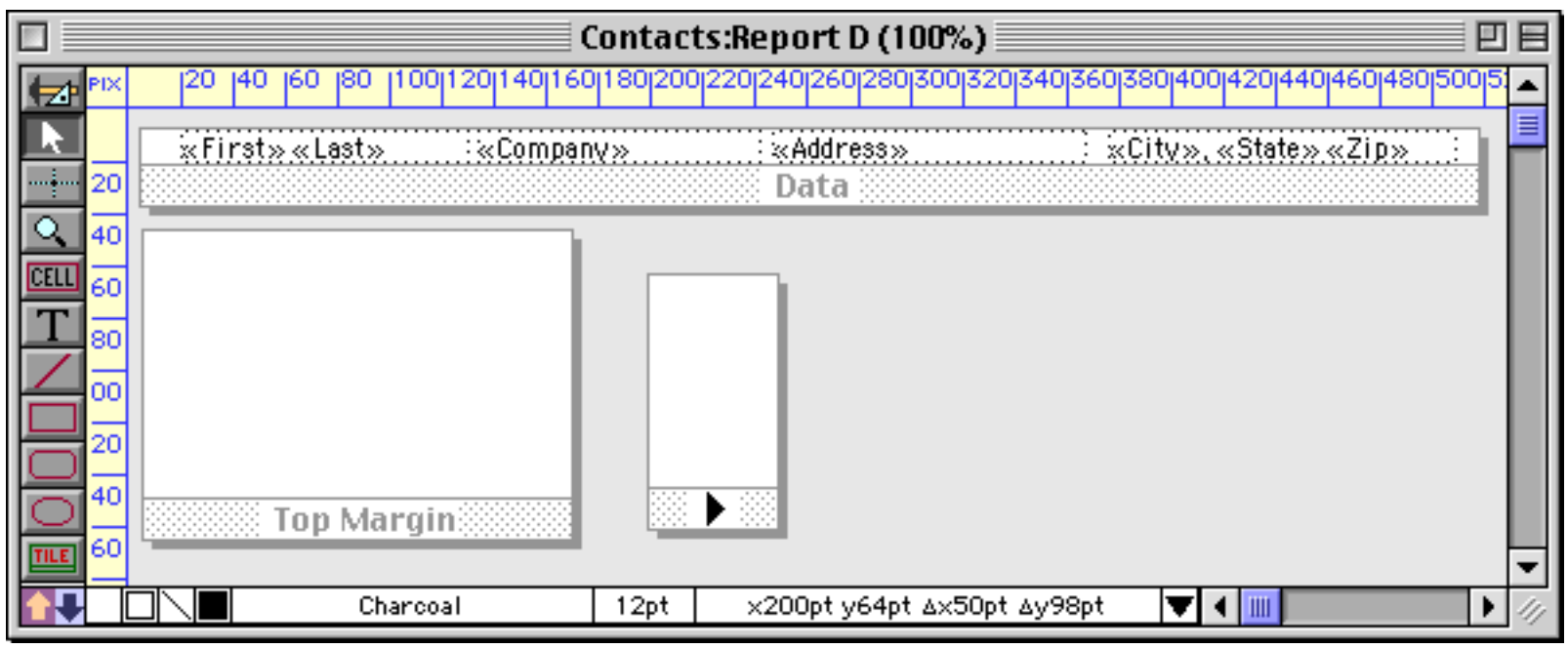
When you release the mouse the tile configuration dialog appears. If the small configuration dialog appears (as shown below) then click on the **Left Margin** button.



If the large configuration dialog appears press the **Left Margin** button, which is near the top of the dialog.



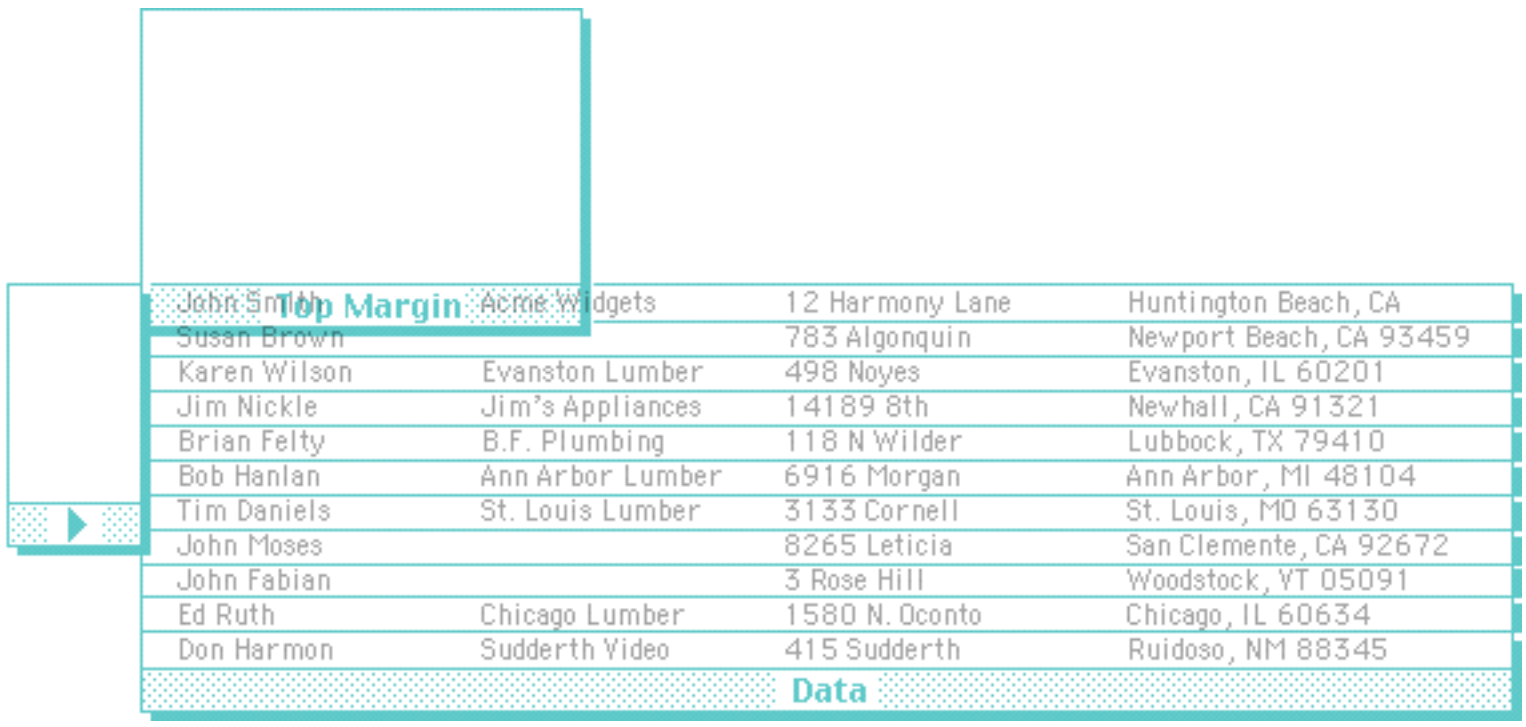
Since the Left Margin tile is often quite narrow, Panorama simply displays a triangle in the tile's drag bar.



When it's printed this report will look like this.

John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091
Ed Ruth	Chicago Lumber	1580 N. Oconto	Chicago, IL 60634
Don Harmon	Sudderth Video	415 Sudderth	Ruidoso, NM 88345

This illustration shows how Panorama assembles the report using the top margin and data tiles. The top margin is always the top tile on the page, with any other tiles stacked below it.



Unlike other tiles, Panorama will not print text or graphics that are placed on the surface of the left margin tile. The left margin tile has only one purpose—to set the left margin.

Right Margin Tile

The right margin tile is rarely used. Panorama normally tries to use the entire printed area of the page, right up to the minimum right margin. The usual right margin is about 1⁄4 inch, but like the left margin, it can vary depending on the type of printer you are using and the current Page Setup.

You can use the right margin tile to set a larger right margin. Like the left margin, Panorama only cares about the width of the right margin tile. If the report contains a right margin tile, the width of that tile becomes the right margin of the report. However, there is one exception—if your right margin tile is less than the minimum width, the report will ignore the tile and use the minimum right margin.

Increasing the right margin has two effects. First, when you are using automatic multiple columns (see "Controlling the Number of Columns" on page 1127), increasing the right margin can reduce the number of columns printed. Panorama will stop printing columns when the right margin is reached. However, it is usually simpler not to use a right margin and instead set the number of columns explicitly using the **Report Preferences** dialog (see "Controlling the Number of Columns" on page 1127).

The right margin also affects the position of the center and right header / footer tiles (see "Designing Headers and Footers For Changing Page Sizes" on page 1097). The centered tiles will be centered between the left and right margins, while right flush tiles will be flush against the right margin.

The right margin does not have any effect on normal header / footer tiles, or on the first column of data tiles. For example, suppose that the header tile is wider than the space between the left and right margins. The header will overlap and print in the right margin area.

Unlike other tiles, Panorama will not print text or graphics that are placed on the surface of the right margin tile. The right margin tile has only one purpose—to set the right margin.

Bottom Margin

Panorama does not have a special tile for setting the bottom margin. Instead, you should include space for the bottom margin in the footer tile (see "Footer Tile" on page 1087).

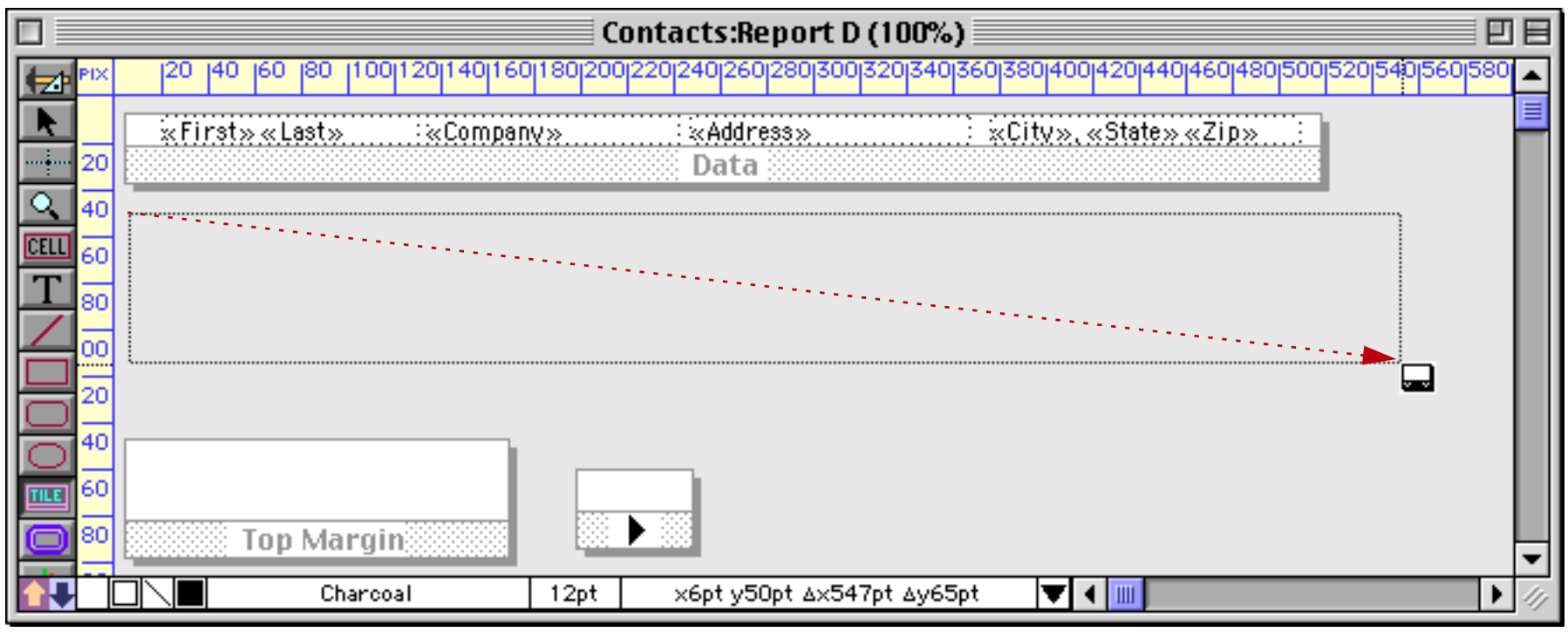
Headers and Footers

Many reports have headers at the top of each page and footers at the bottom. As you might guess, headers and footers are set up with report tiles!

Header Tile

The header tile is printed at the top left of each page. The header can be used to print the report title, page number (see “Page Numbers” on page 1088), date (see “Printing the Current Date and Time” on page 1091) or anything else you want.

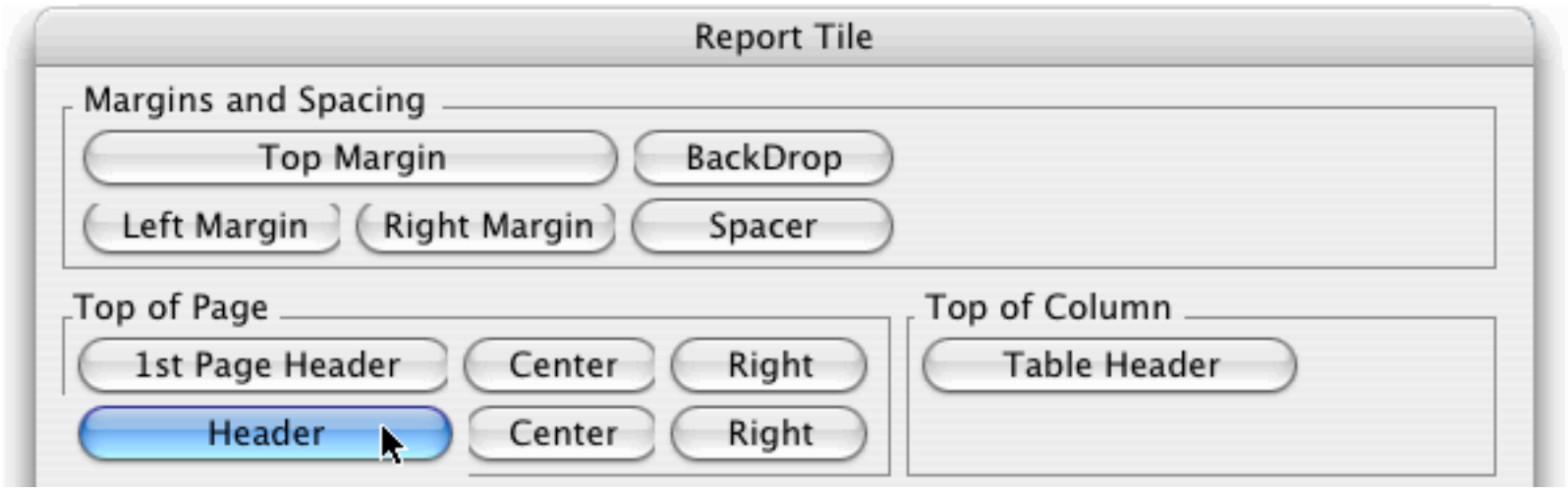
To create a header, start by selecting the **Tile** tool, just as with any other tile. Drag the mouse over an empty spot on the form, leaving the proper amount of space for the header you want to create. Usually the header is approximately the entire width of the page. (Of course you can adjust the dimensions later.) The header tile can be in any empty spot on the form. In this example we are creating the header tile below the data tile, even though the header will eventually print above the data.



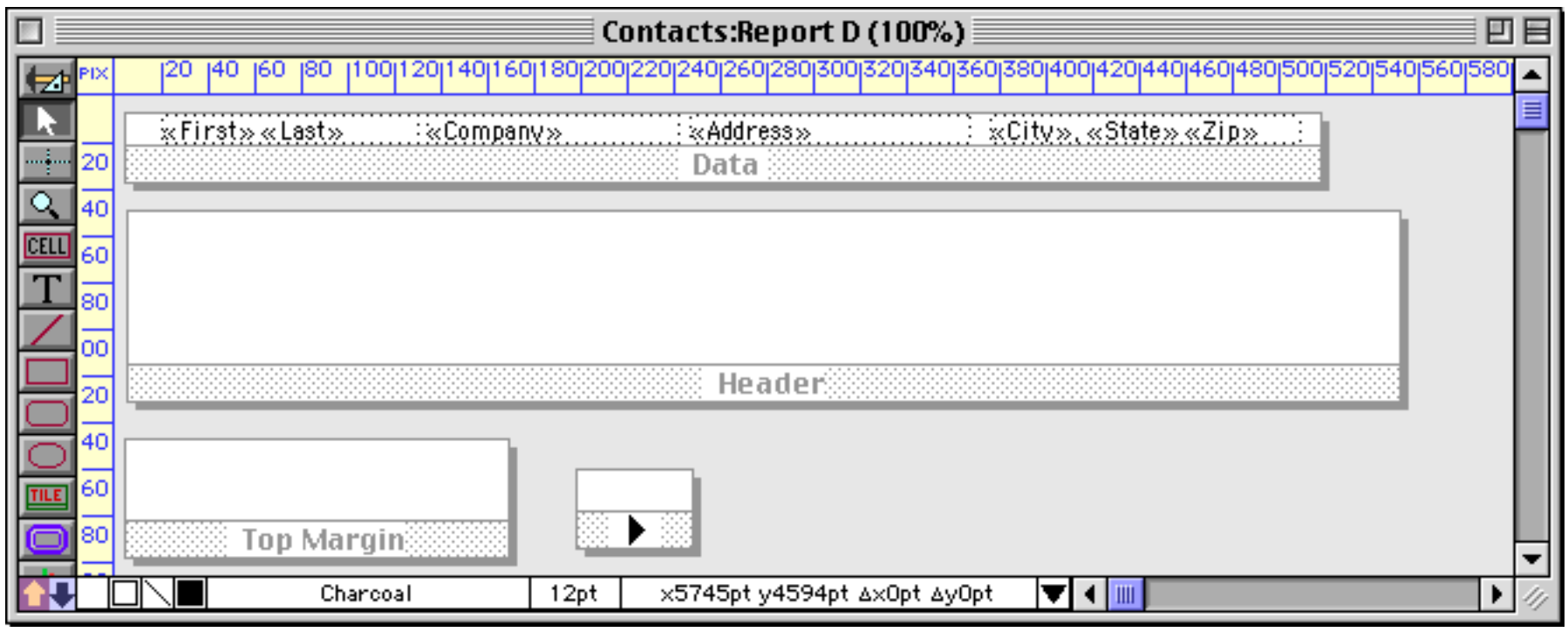
When you release the mouse the tile configuration dialog appears. If the small configuration dialog appears (as shown below) then click on the **Header** button.



If the large configuration dialog appears press the **Header** button, in the *Top of Page* section.



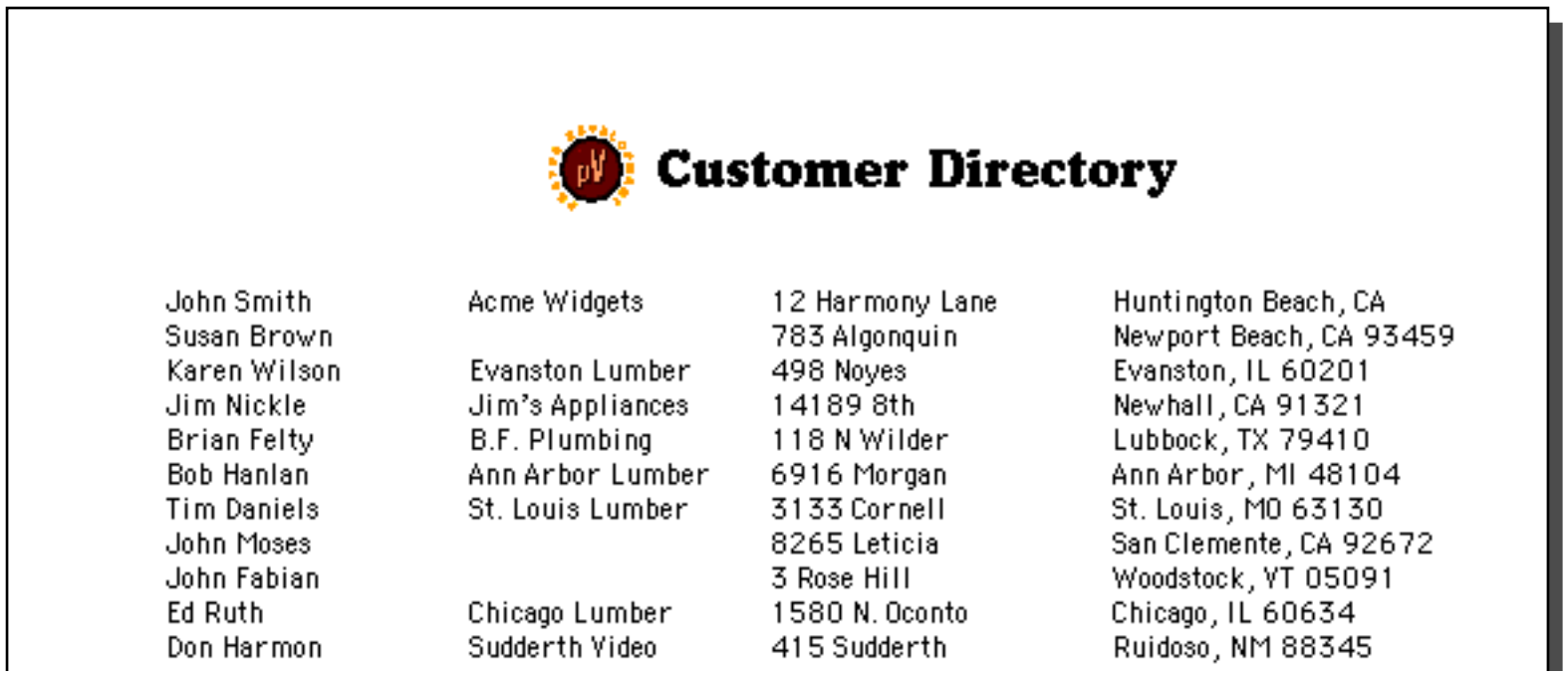
Here's the new tile.



A blank header tile isn't much use, so we'll add some text and graphics to the header.



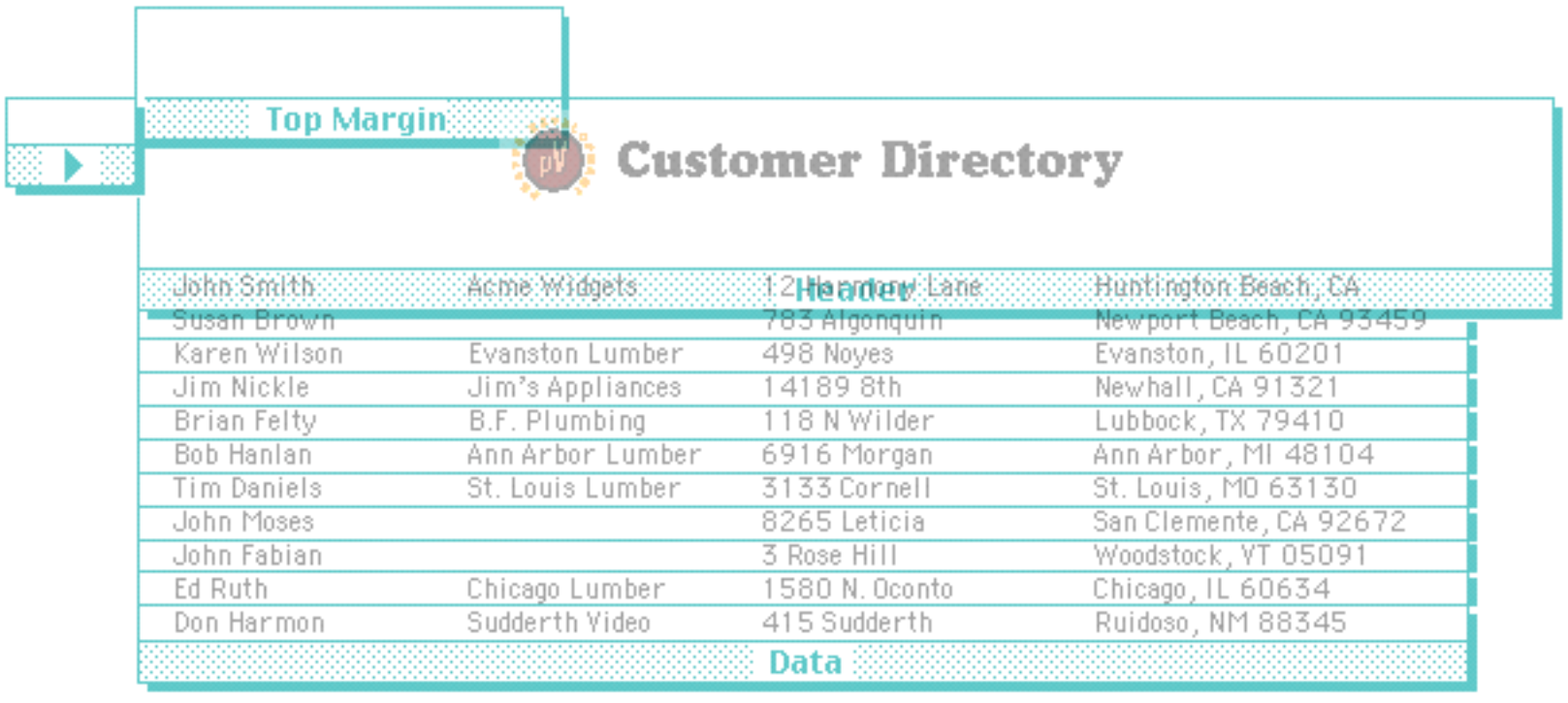
When it's printed this report will look like this.



Customer Directory

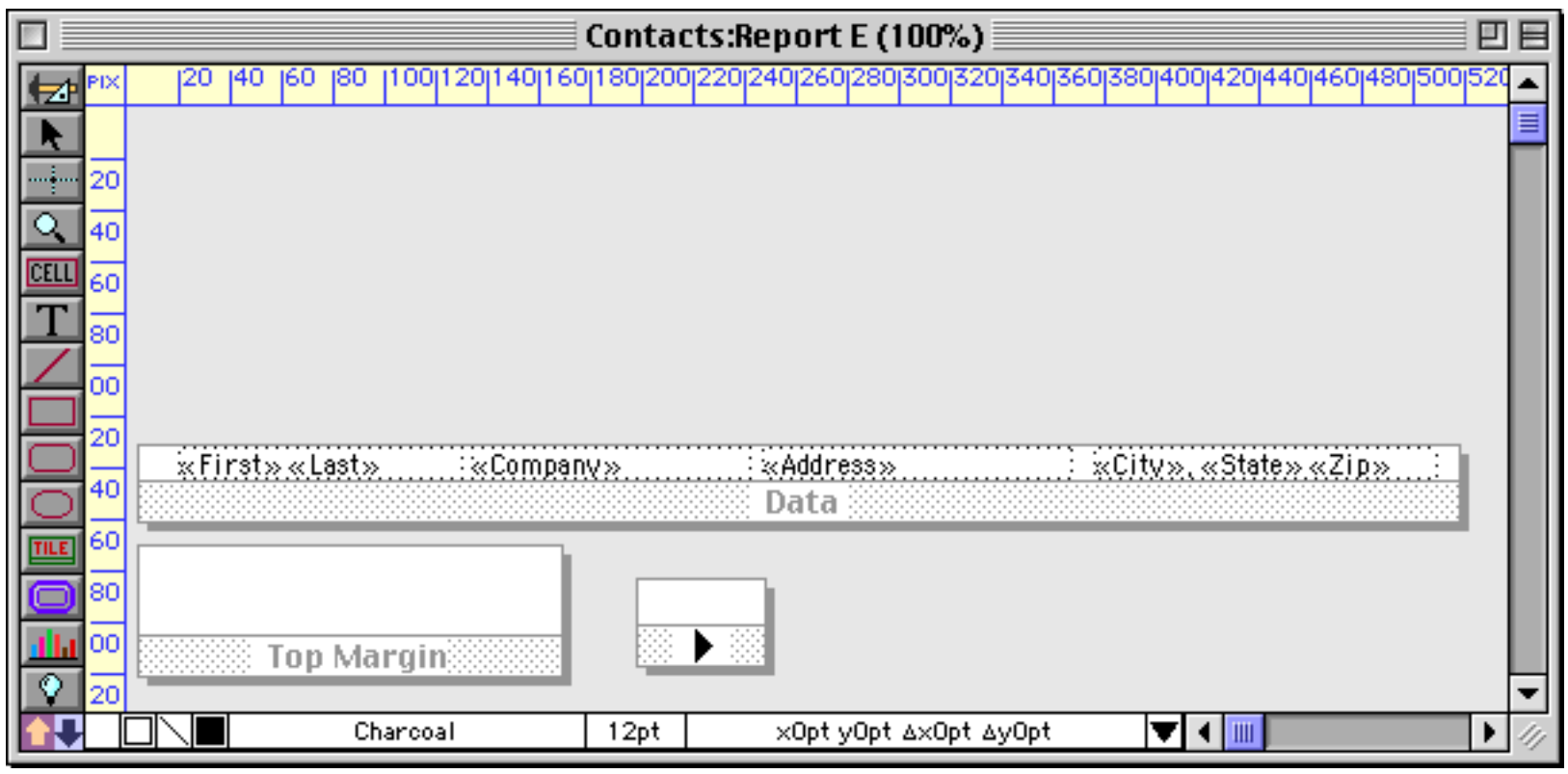
John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091
Ed Ruth	Chicago Lumber	1580 N. Oconto	Chicago, IL 60634
Don Harmon	Sudderth Video	415 Sudderth	Ruidoso, NM 88345

This illustration shows how Panorama assembles the report using the margin, header and data tiles.

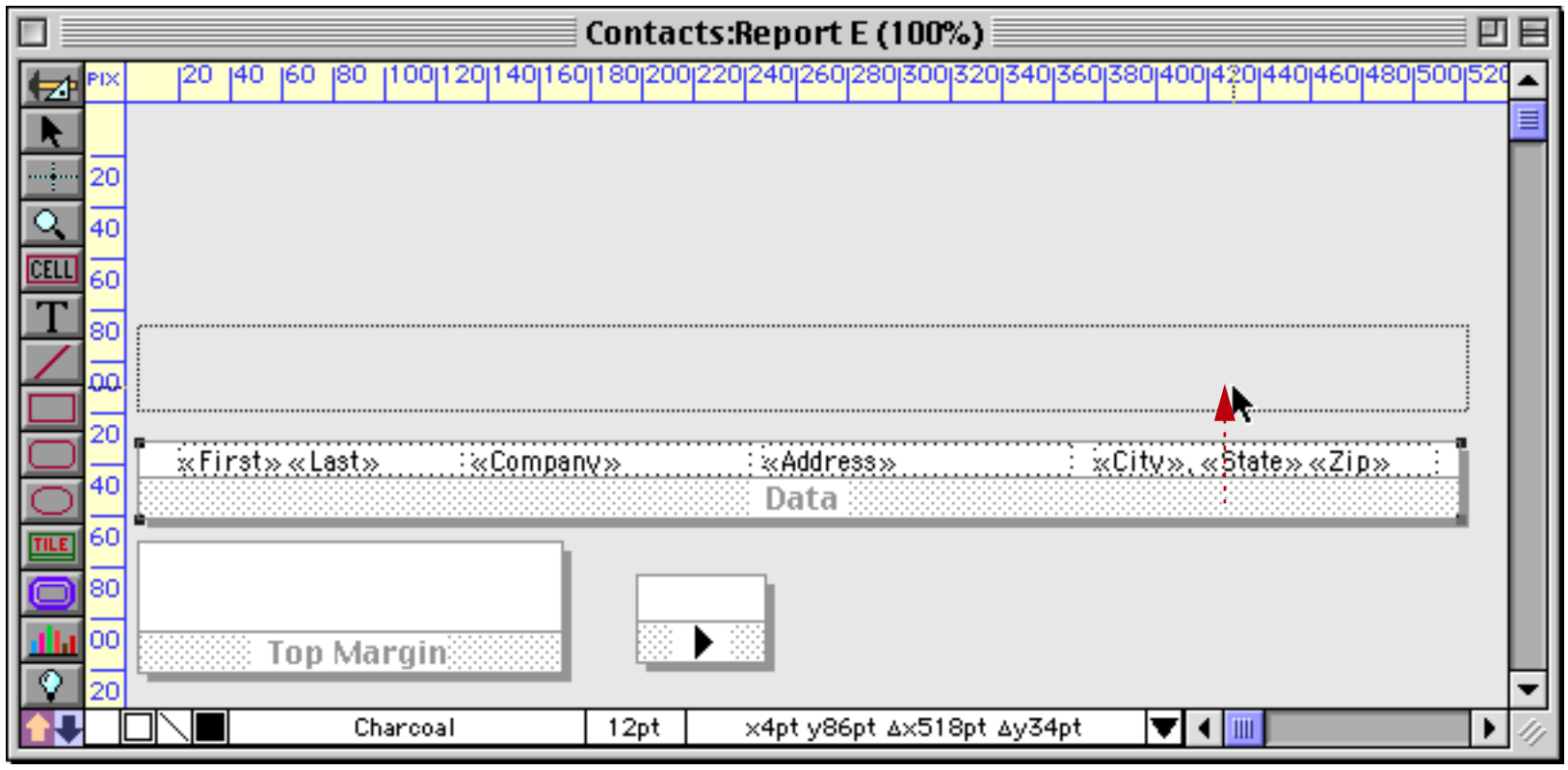


Creating a Header Tile by Duplicating the Data Tile

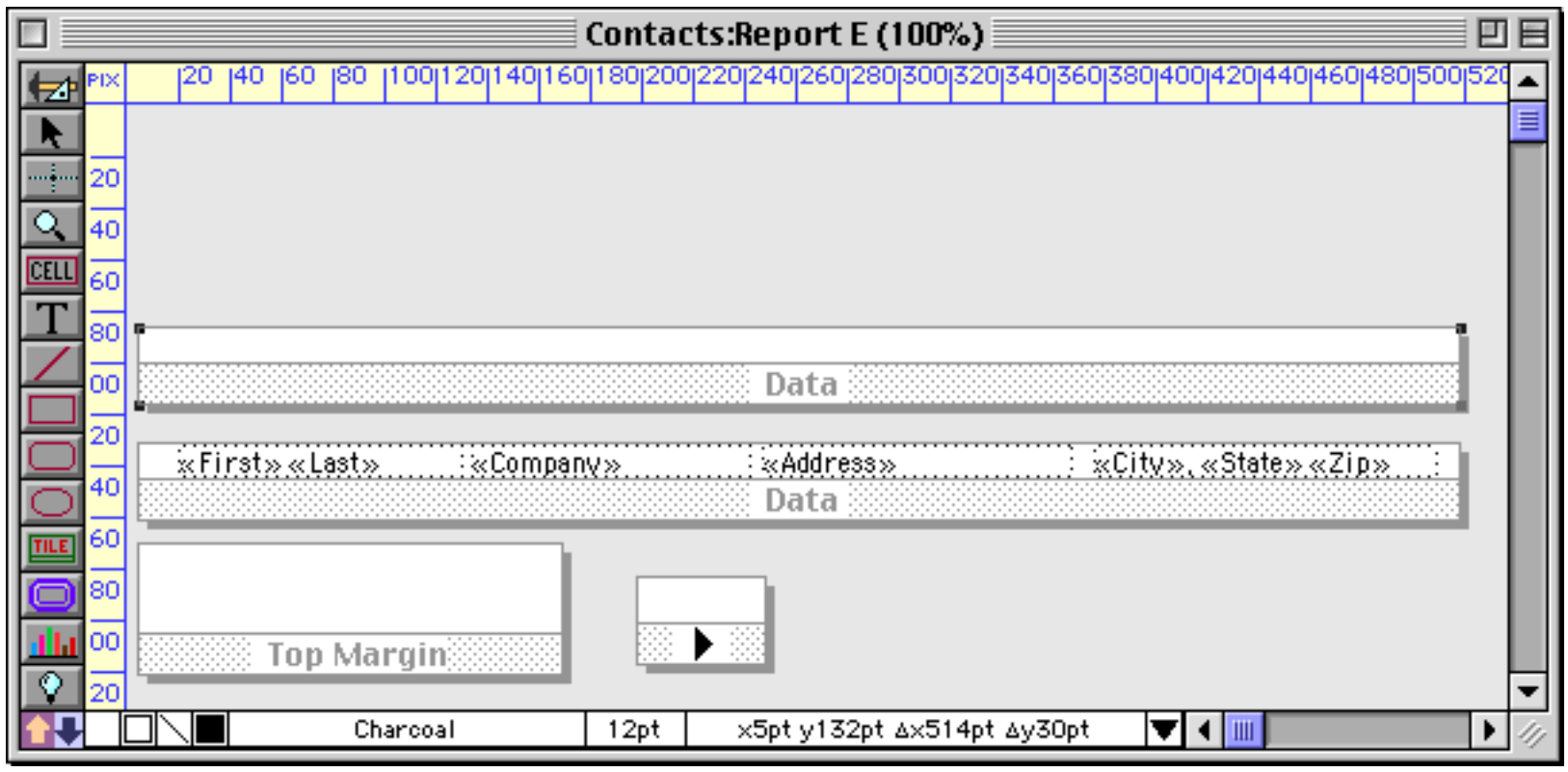
An alternate way to create the header tile is to duplicate the data tile (see "Drag Duplicating" on page 561). This makes it easy to line up items between the two tiles. Start with just a data tile.



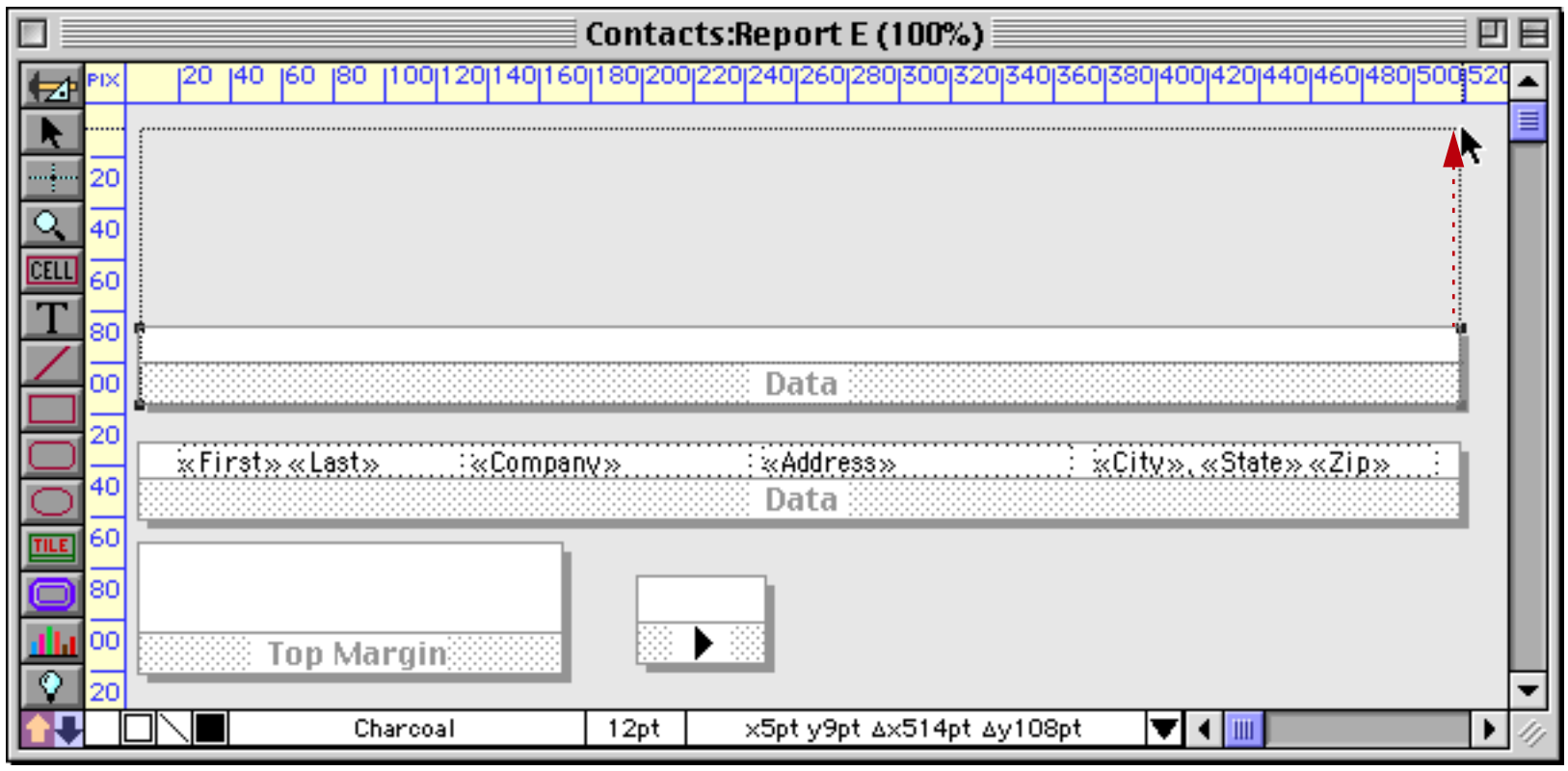
To duplicate the tile, hold down the **Option** key (Mac) or **Alt** key (Windows) and drag the tile (see "Drag Duplicating" on page 561). You probably also want to hold down the **Shift** key at the same time to make sure that the two tiles stay in perfect alignment.



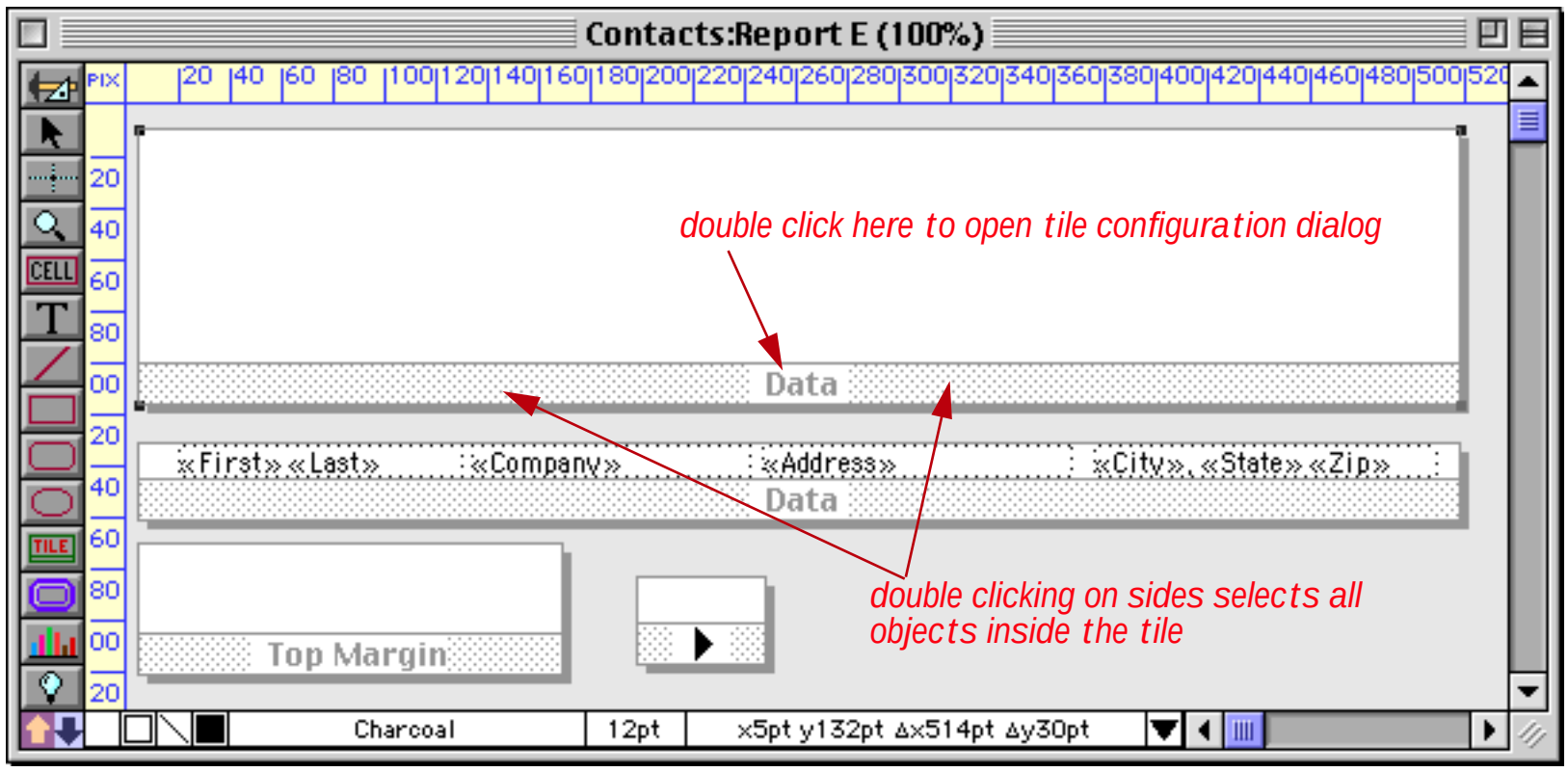
When you release the mouse your form will contain two data tiles.



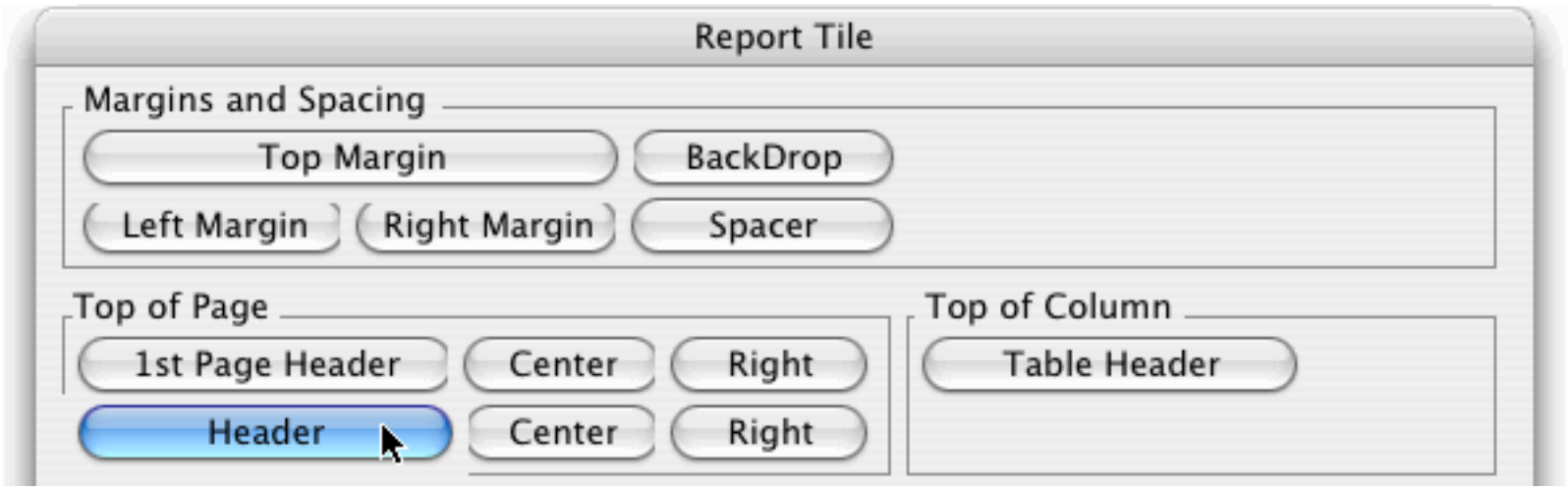
Grab one of the handles to adjust the height of the new tile.



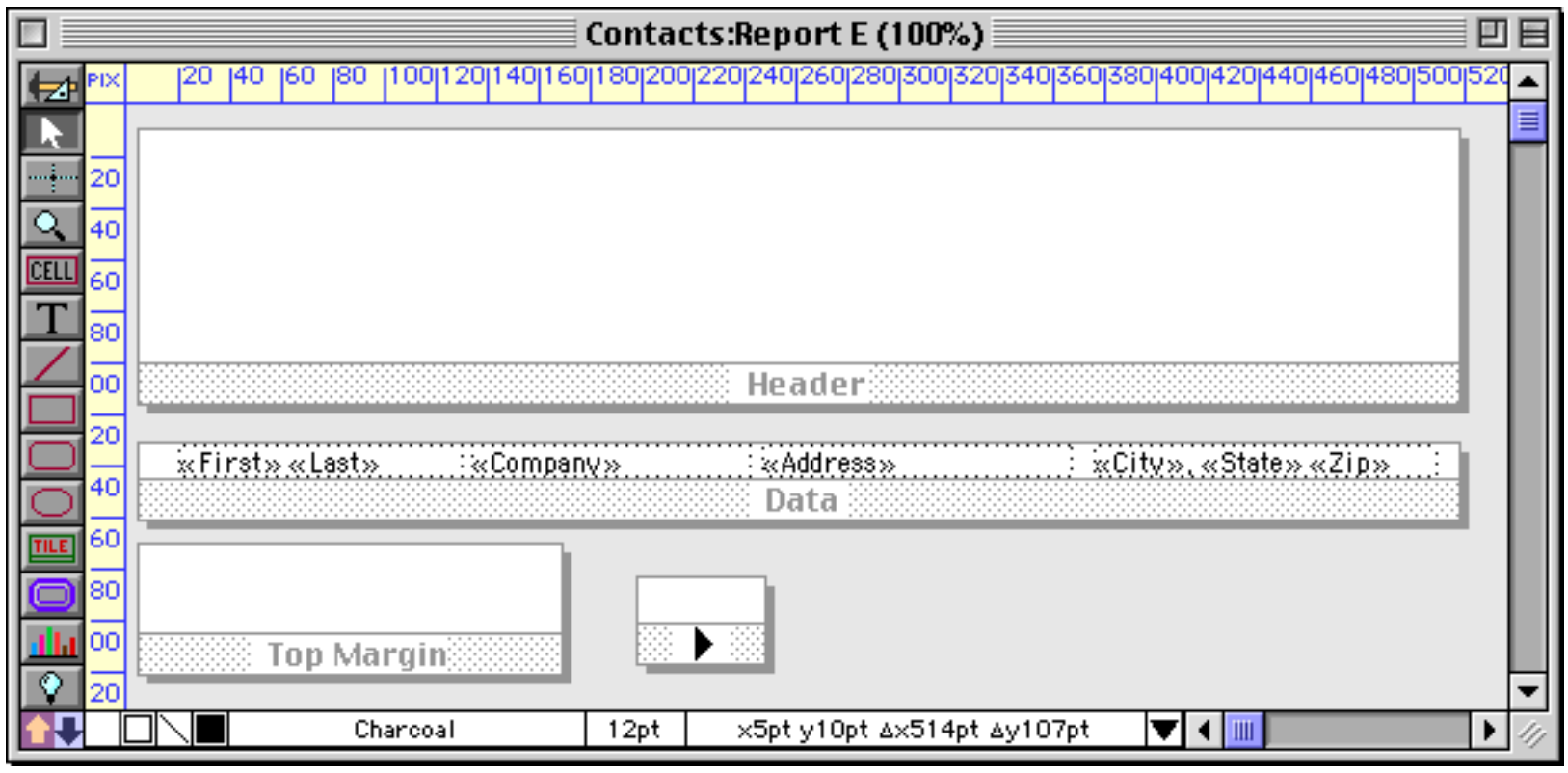
To convert the new tile from a data tile into a header tile, double click on the word **Data**.



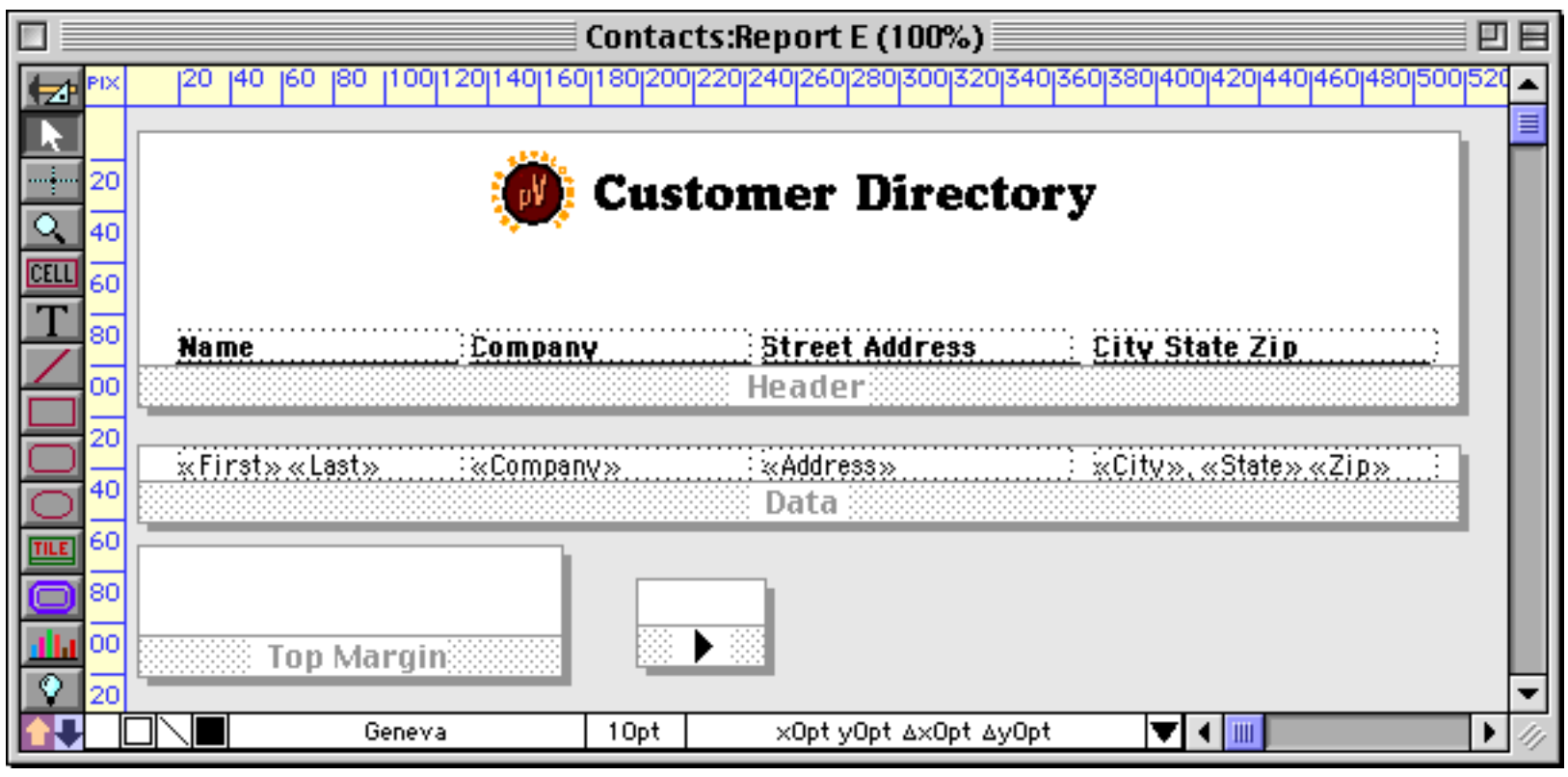
This opens the tile configuration dialog.



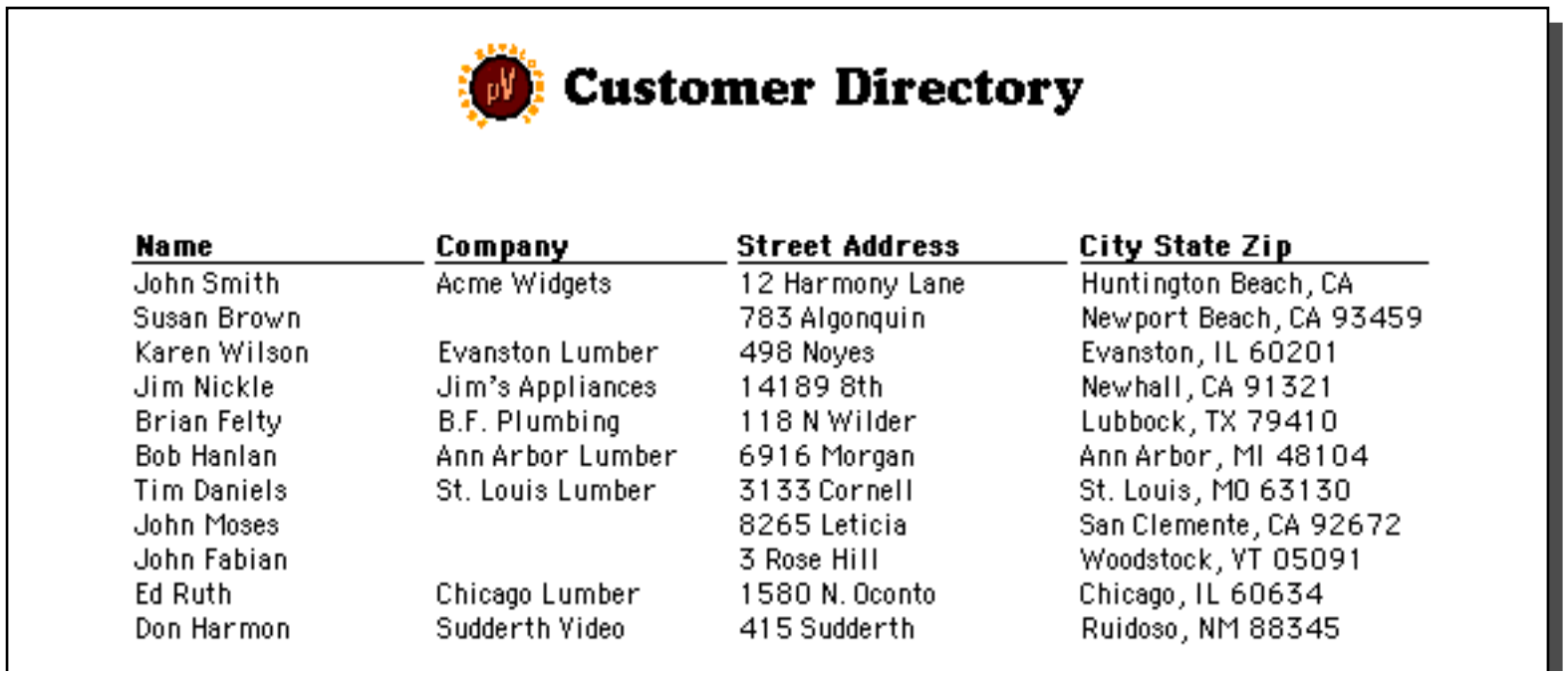
Press the **Header** button to convert the tile.



Now we can add graphics and text to the header. Anything placed on the header will line up vertically with objects in the same position on the data tile.



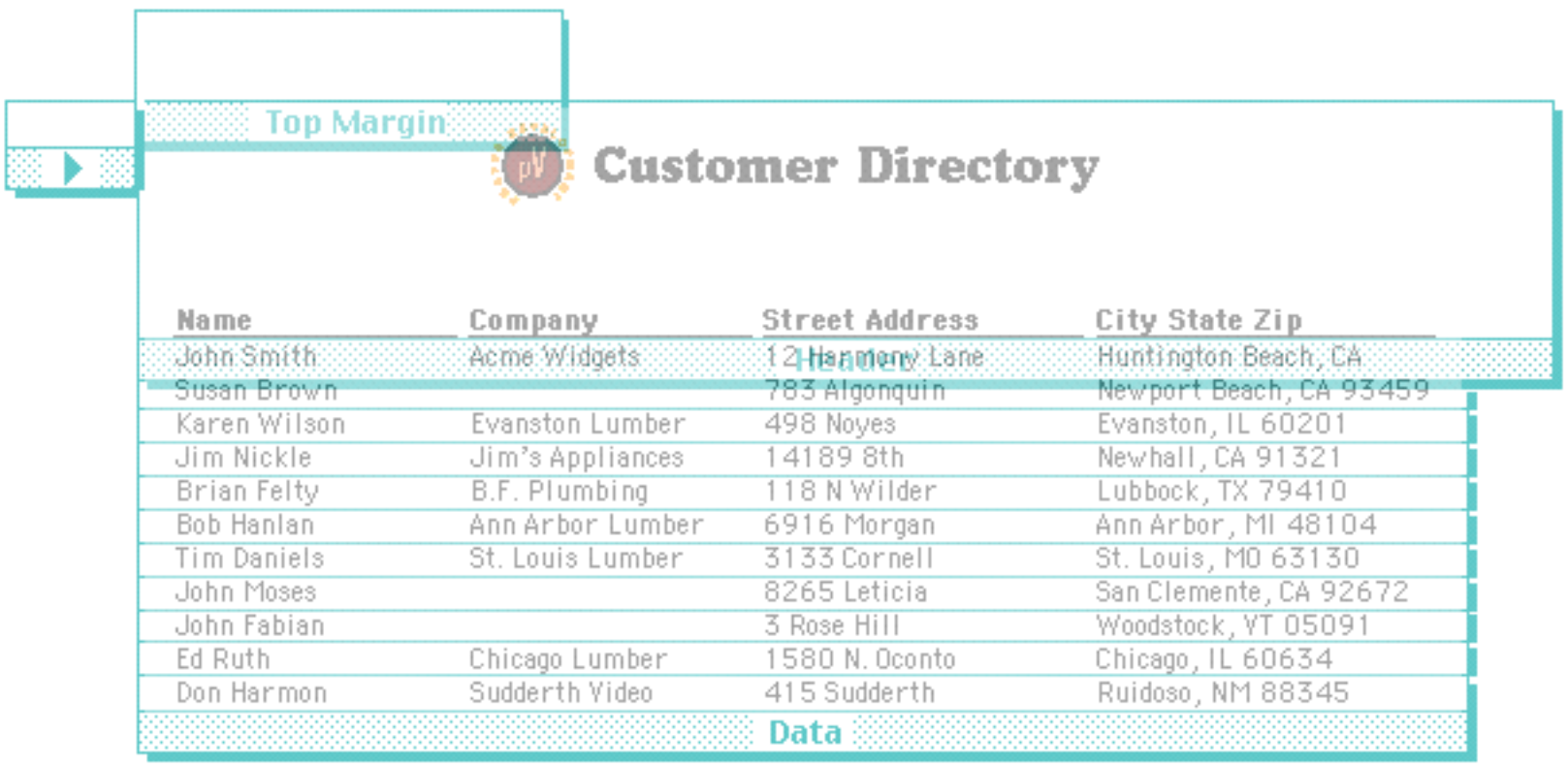
Here's the printed report.



Customer Directory

Name	Company	Street Address	City State Zip
John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091
Ed Ruth	Chicago Lumber	1580 N. Oconto	Chicago, IL 60634
Don Harmon	Sudderth Video	415 Sudderth	Ruidoso, NM 88345

Once again Panorama builds these reports by sliding the tile's into place.

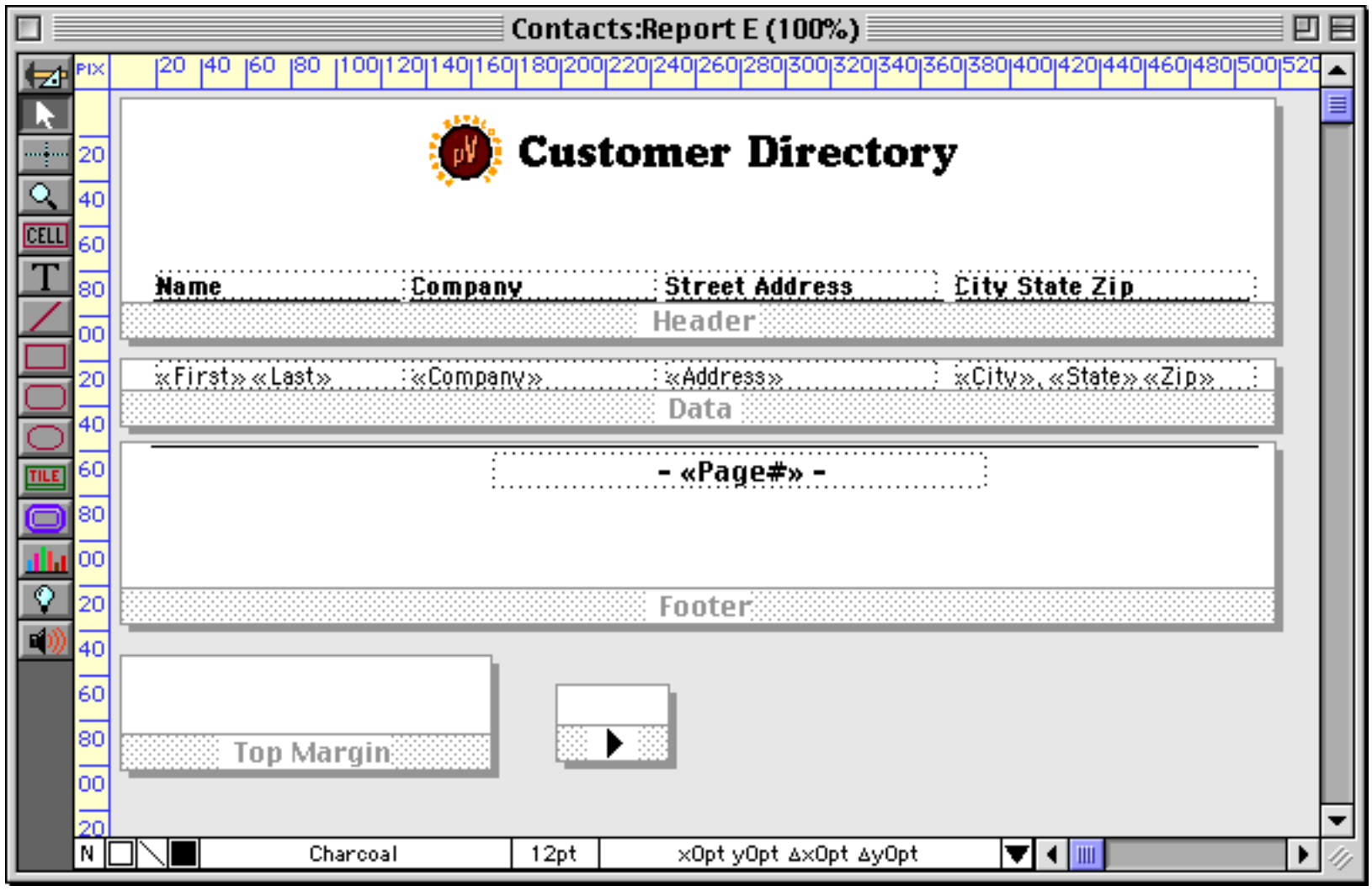


You can also build a header using the **Table Header** tile. See "Table Header and Table Footer Tiles" on page 1123.

Footer Tile

The footer tile is printed at the bottom left of each page. The footer can be used to print the report title, page number (see "Page Numbers" on page 1088), date (see "Printing the Current Date and Time" on page 1091) or anything else you want.

You can create a footer with the **Tile** tool (see "Header Tile" on page 1079) or by duplicating another tile (see "Creating a Header Tile by Duplicating the Data Tile" on page 1082). In this example the footer was created by drag duplicating the data tile (see "Drag Duplicating" on page 561).

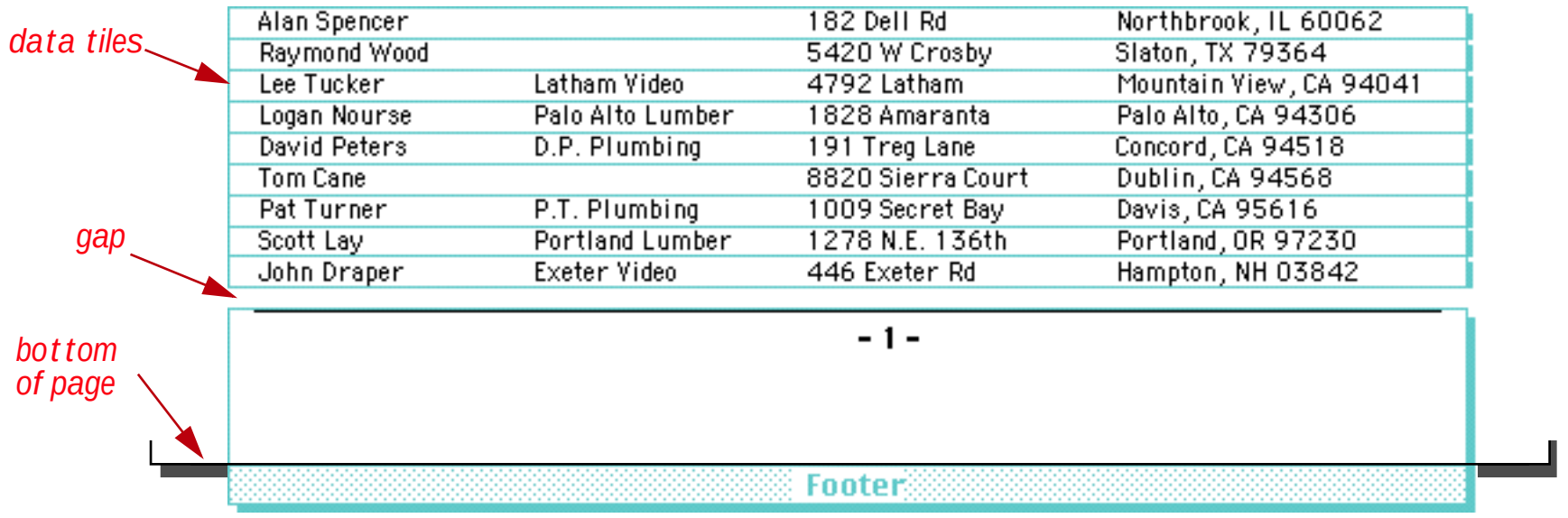


Here's what the bottom of this page looks like.

Alan Spencer		182 Dell Rd	Northbrook, IL 60062
Raymond Wood		5420 W Crosby	Slaton, TX 79364
Lee Tucker	Latham Video	4792 Latham	Mountain View, CA 94041
Logan Nourse	Palo Alto Lumber	1828 Amaranta	Palo Alto, CA 94306
David Peters	D.P. Plumbing	191 Treg Lane	Concord, CA 94518
Tom Cane		8820 Sierra Court	Dublin, CA 94568
Pat Turner	P.T. Plumbing	1009 Secret Bay	Davis, CA 95616
Scott Lay	Portland Lumber	1278 N.E. 136th	Portland, OR 97230
John Draper	Exeter Video	446 Exeter Rd	Hampton, NH 03842

- 1 -

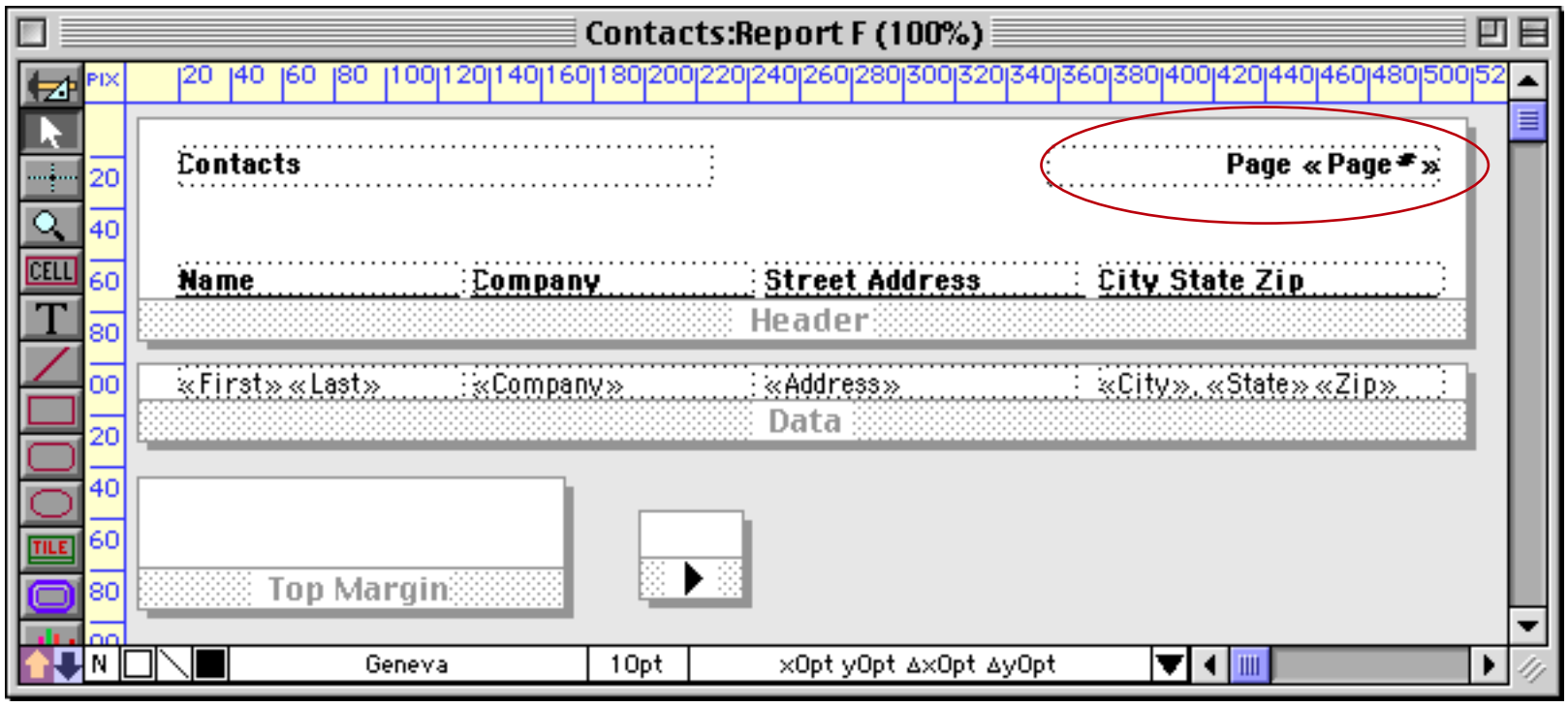
Notice that there is a gap between the bottom of the last line of data and the footer tile. This is because the Panorama always aligns the footer with the bottom of the page. The footer is printed as close to the bottom of the page as possible.



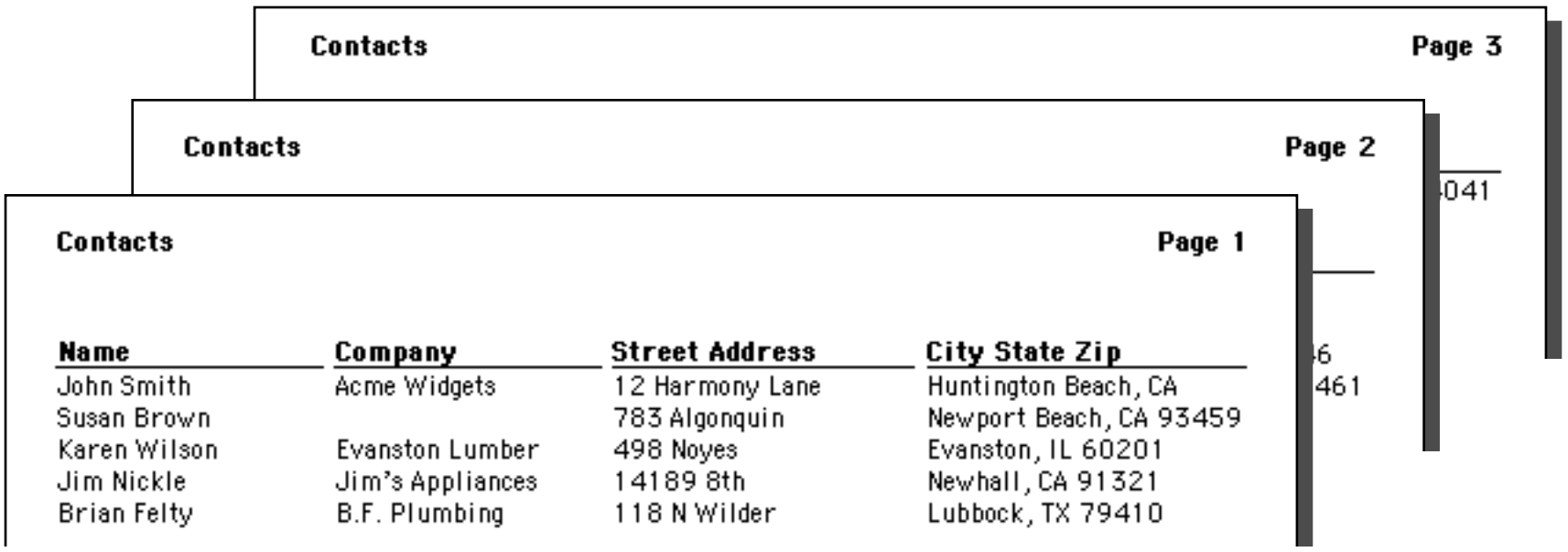
If you want to create a footer that is right below the data tiles with no gap, use the **Table Footer** tile (see "Table Header and Table Footer Tiles" on page 1123).

Page Numbers

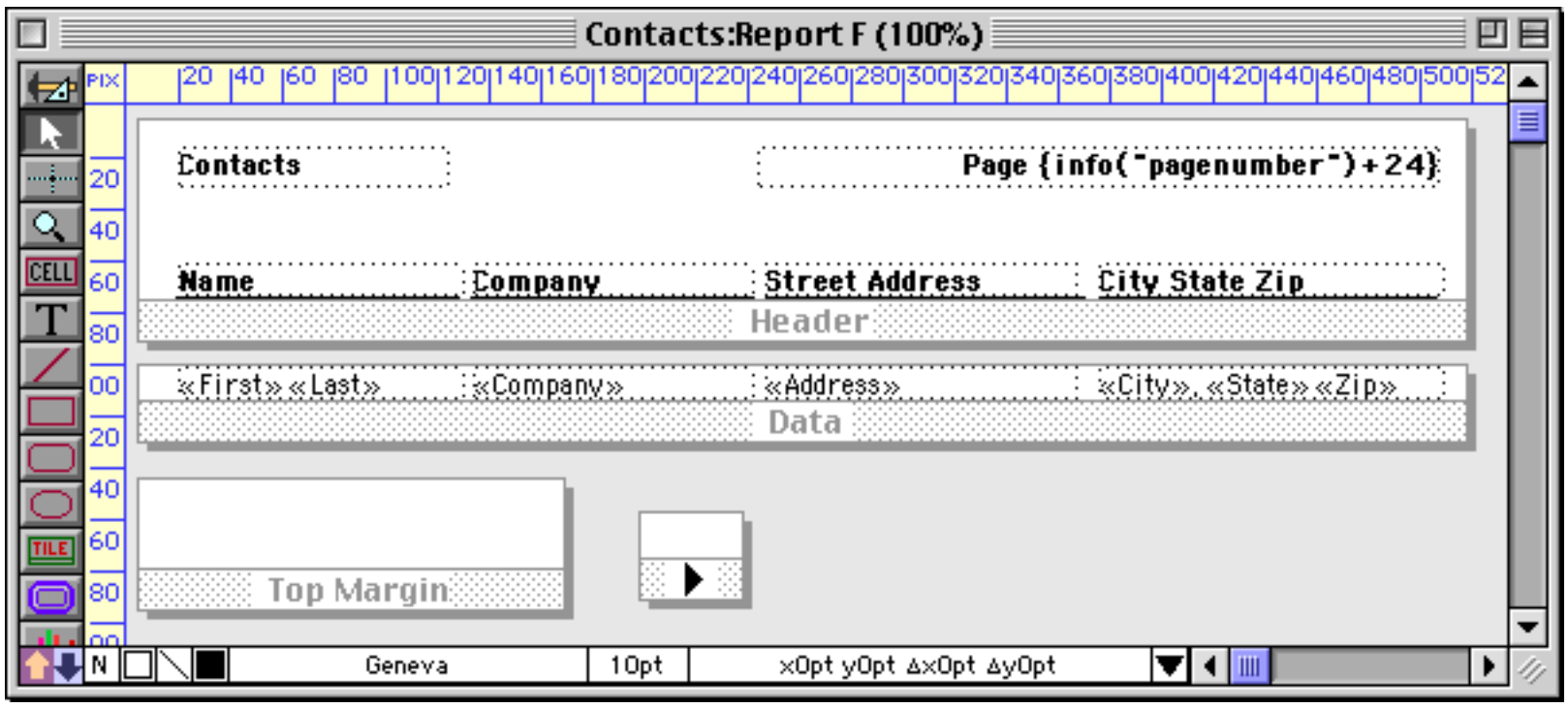
Panorama can automatically calculate and print a page number on each page of your report. One technique for printing the page number is to use an auto-wrap text object (see "Displaying Data in Auto-Wrap Text" on page 595). The page number is merged into the text by typing «page#» into the object. (On a Macintosh the « chevron is **Option**-\ and the » chevron is **Shift-Option**-\. On Windows systems the « chevron is **Alt-0171** and the » chevron is **Alt-0187**.) The illustration below shows how to print page numbers on the upper right hand corner of a report. Notice that the text object has been set to right justify (Style Menu) so that the page number will be flush with the right edge of the report.



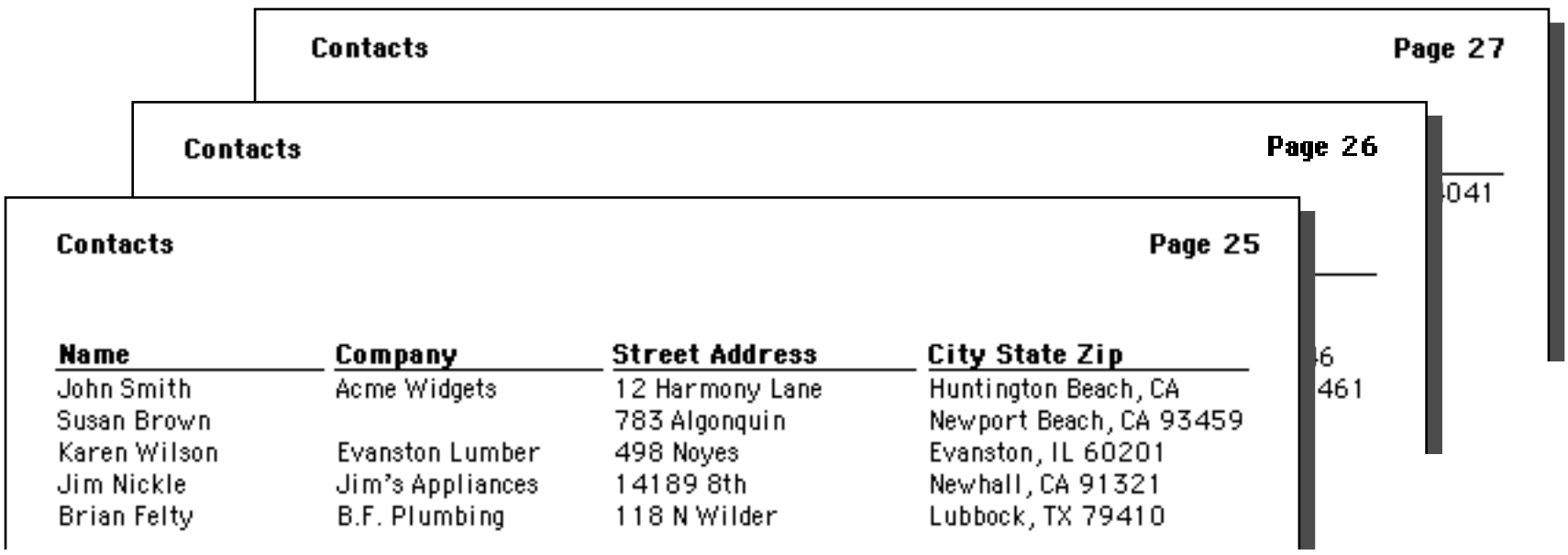
Here is what the first three pages of this report look like.



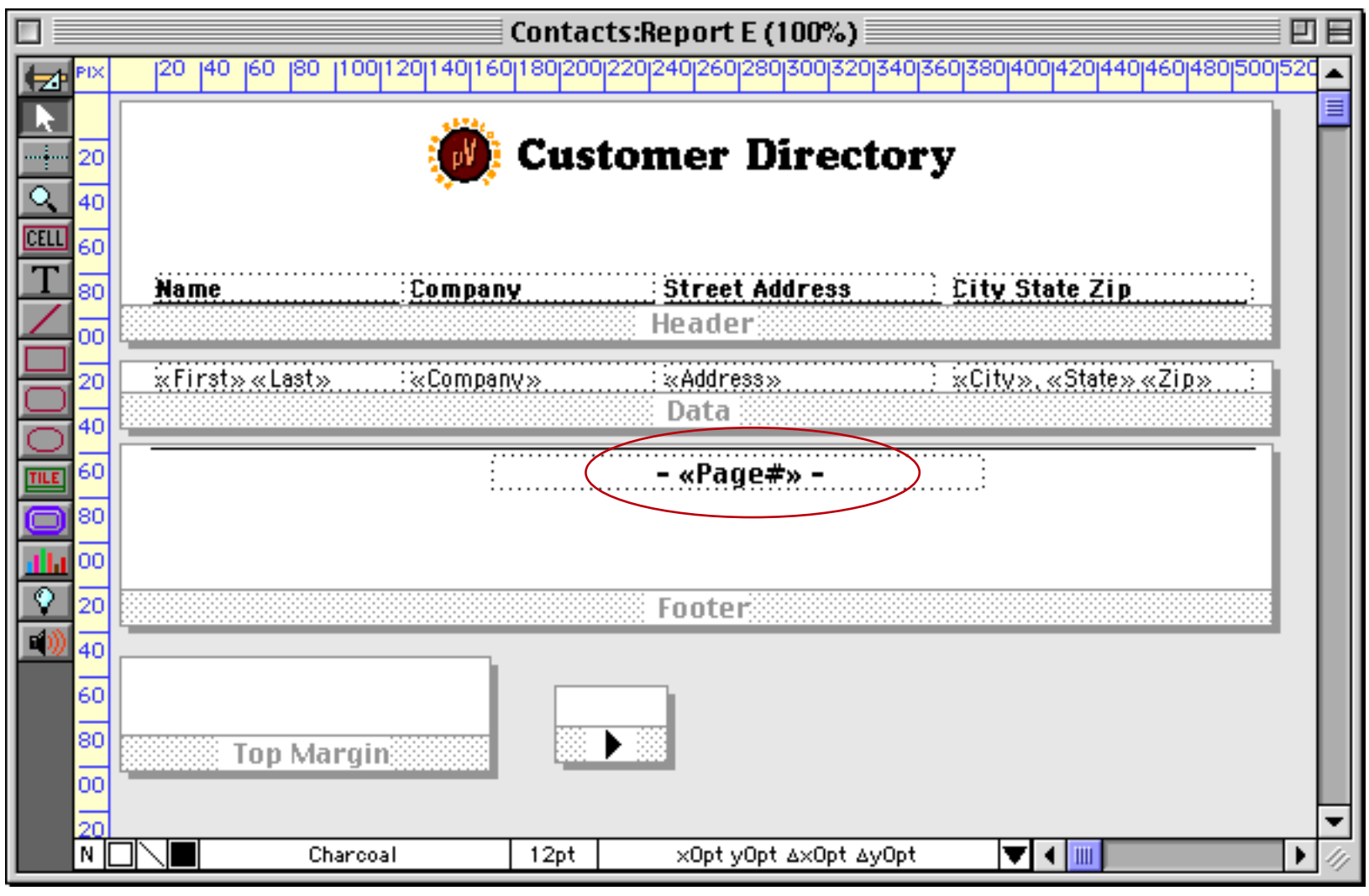
If you want the page number to start with a page number other than one you must use the `info("pagenumber")` function. You can either merge this function into an auto-wrap text object (see "Displaying Formulas in Auto-Wrap Text" on page 602) or use a Text Display SuperObject (see "Text Display SuperObjects™" on page 608). The example below shows how to create a report with page numbers starting at 25.



Here's the printed report, with page numbering beginning at 25.



To center the page number at the bottom of each page, place an auto-wrap text or Text Display SuperObject in the center of the footer tile. To make sure that the page number is centered, the text object should be centered within the footer tile. You also need to check the center justify option in the Style Menu.



You may want to display each page number in relation to the total number of pages in the report—for instance Page 2 of 5. Panorama cannot automatically calculate the total number of pages in a report, but you can find out this number manually with the Preview command (see "Print Preview" on page 1046). Use the **Next Page** tool in the preview window to count the number of pages, then edit the title to display the correct number.

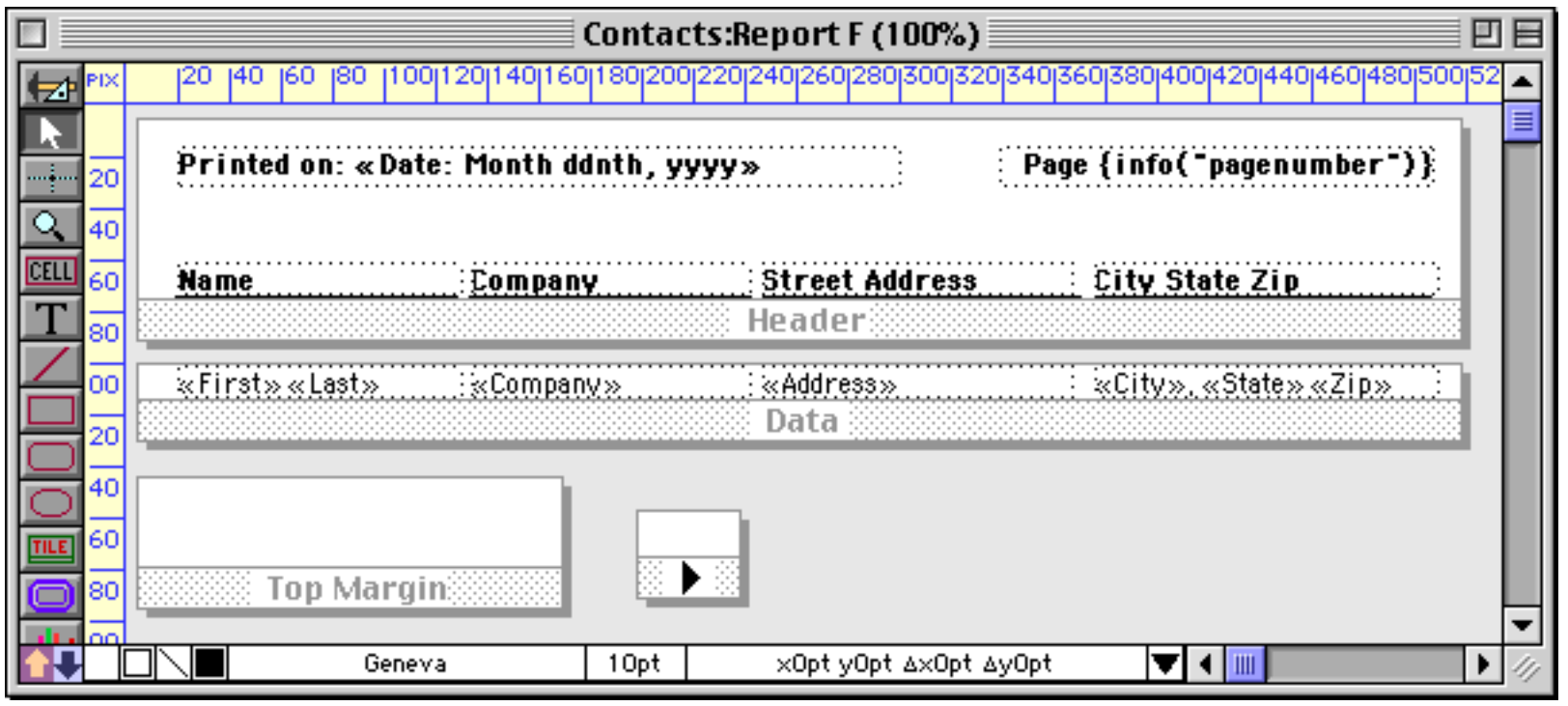
The examples shown have all assumed that odd and even pages are identical. It is also possible to make odd and even pages mirror images, so that the page number always prints on the outside (on the right for odd pages and on the left for even pages). See “Even and Odd Page Layout” on page 1152 to learn how to set this up.

Printing the Current Date and Time

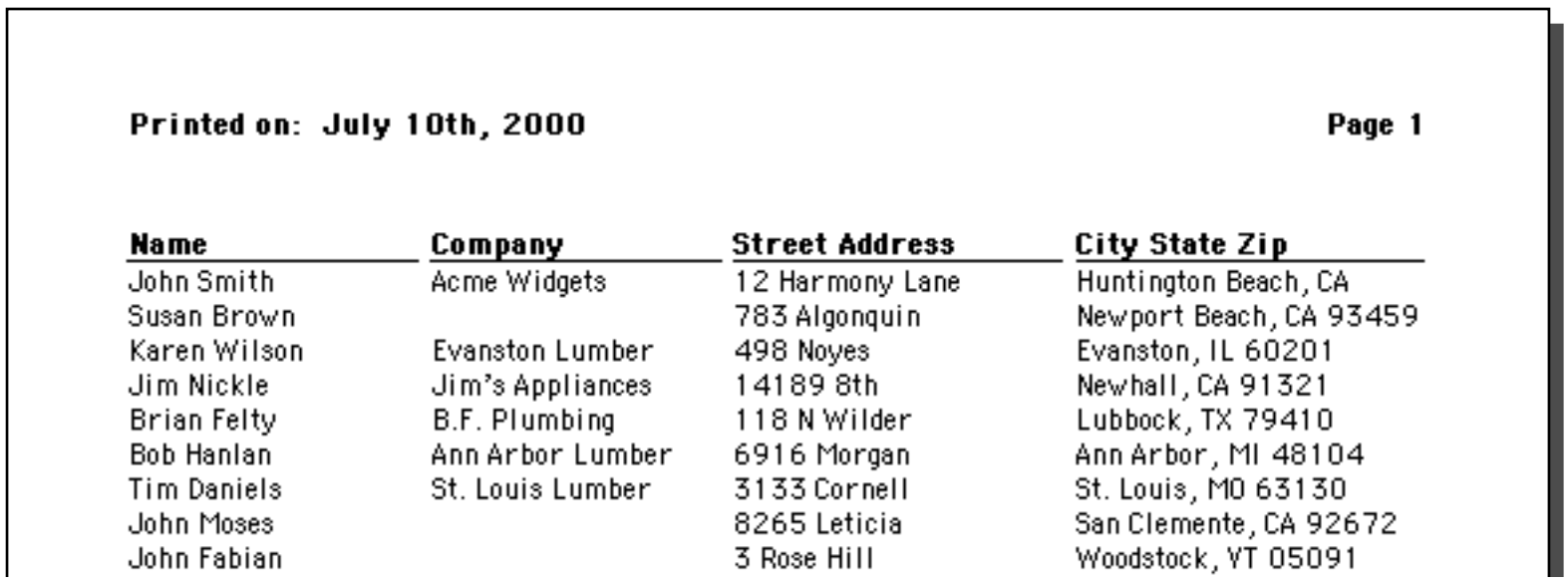
Panorama can automatically print the current date and time on each page of your report. One technique for printing the date is to use an auto-wrap text object (see “Displaying Data in Auto-Wrap Text” on page 595). The page number is merged into the text by typing «date:pattern» into the object. (On a Macintosh the « chevron is **Option-** and the » chevron is **Shift-Option-**. On Windows systems the « chevron is **Alt-0171** and the » chevron is **Alt-0187**.) Don’t actually type in the word pattern, instead choose one of the patterns from the table below.

Pattern	Example
mm/dd/yy	3/9/04
MM/DD/YY	03/09/04
mm-dd-yyyy	3-9-2004
dd-MON-yy	9-MAR-04
dd-Month-yy	9-March-04
Month dd, yyyy	March 9, 2004
Month ddnth, yyyy	March 9th, 2004
DayOfWeek, Month dd	Thursday, March 9

The illustration below shows how to print the current date in the upper left hand corner of a report.



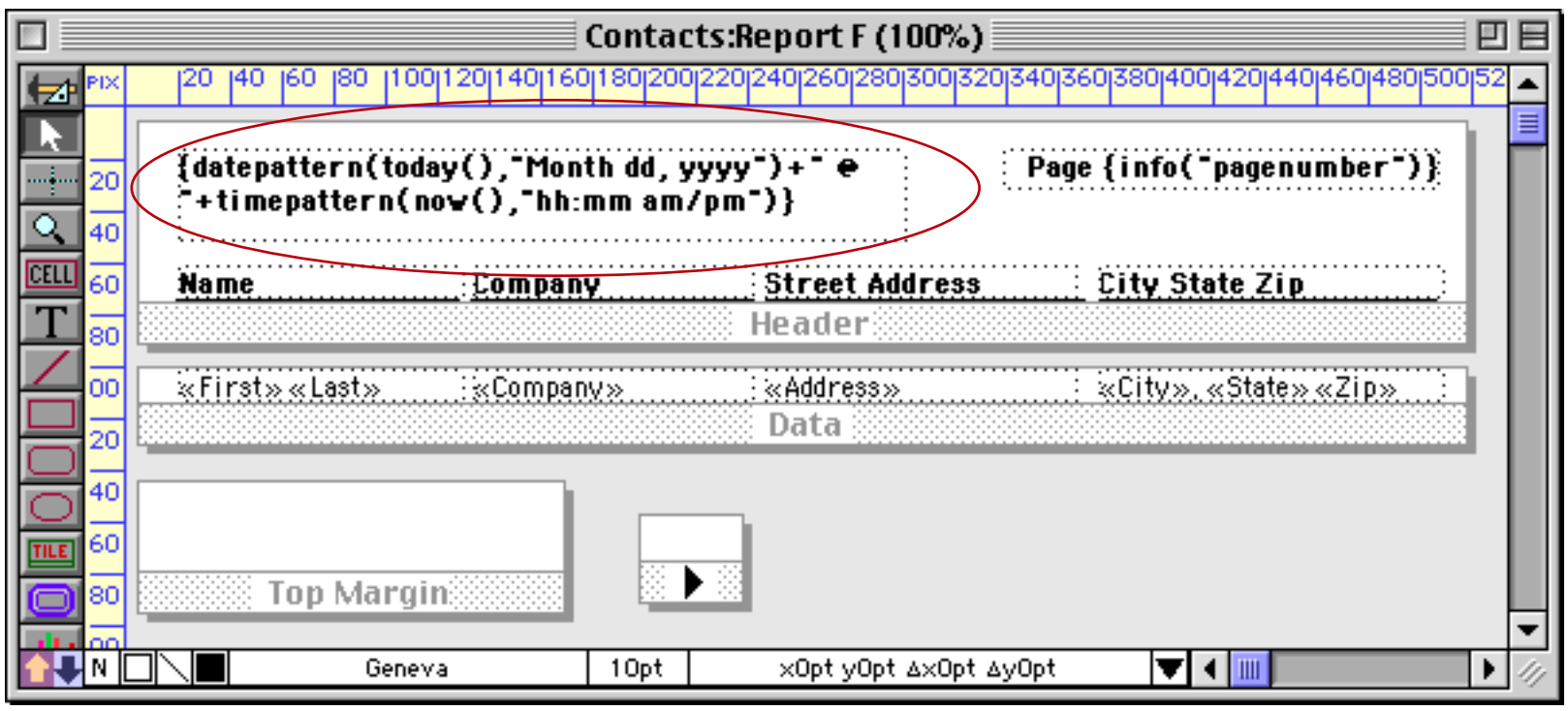
Here's the top of the first page of this report.



Printed on: July 10th, 2000 **Page 1**

Name	Company	Street Address	City State Zip
John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091

To include the current time in a printed report you must use the `now()` and `timepattern(` functions. You can either merge these function into an auto-wrap text object (see "Displaying Formulas in Auto-Wrap Text on page 602") or use a Text Display SuperObject (see "Text Display SuperObjects™ on page 608"). The example below prints both the date and time using a formula.



Here is the printed report.

July 10, 2000 @ 2:31 PM Page 1

Name	Company	Street Address	City State Zip
John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091

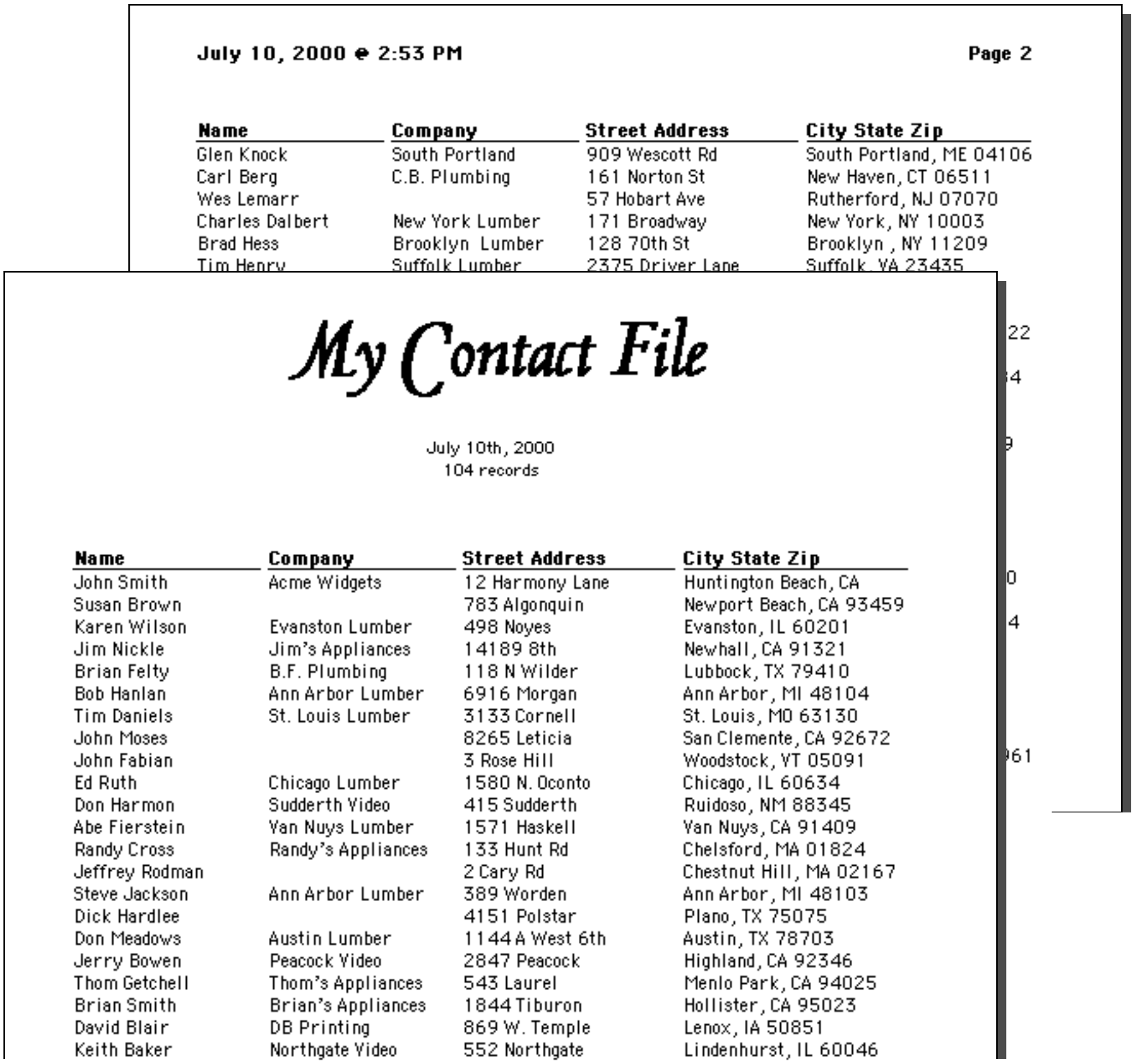
You can actually merge in any formula you want into the header and / or footer.

First Page Header Tile

Panorama normally prints the header tile (or tiles) at the top of each page (see above). However, if the report includes a first page header tile, that tile will be printed at the top of the first page. All subsequent pages will print using the normal header tile (or tiles). Here is an example of a form that contains both a first page header and a regular header that will print on the second, third, and any additional pages.



Here are the first two pages of the printed report generated by this form. The first page includes the **First Page Header**, while the second (and subsequent pages) use the regular Header tile.



July 10, 2000 @ 2:53 PM Page 2

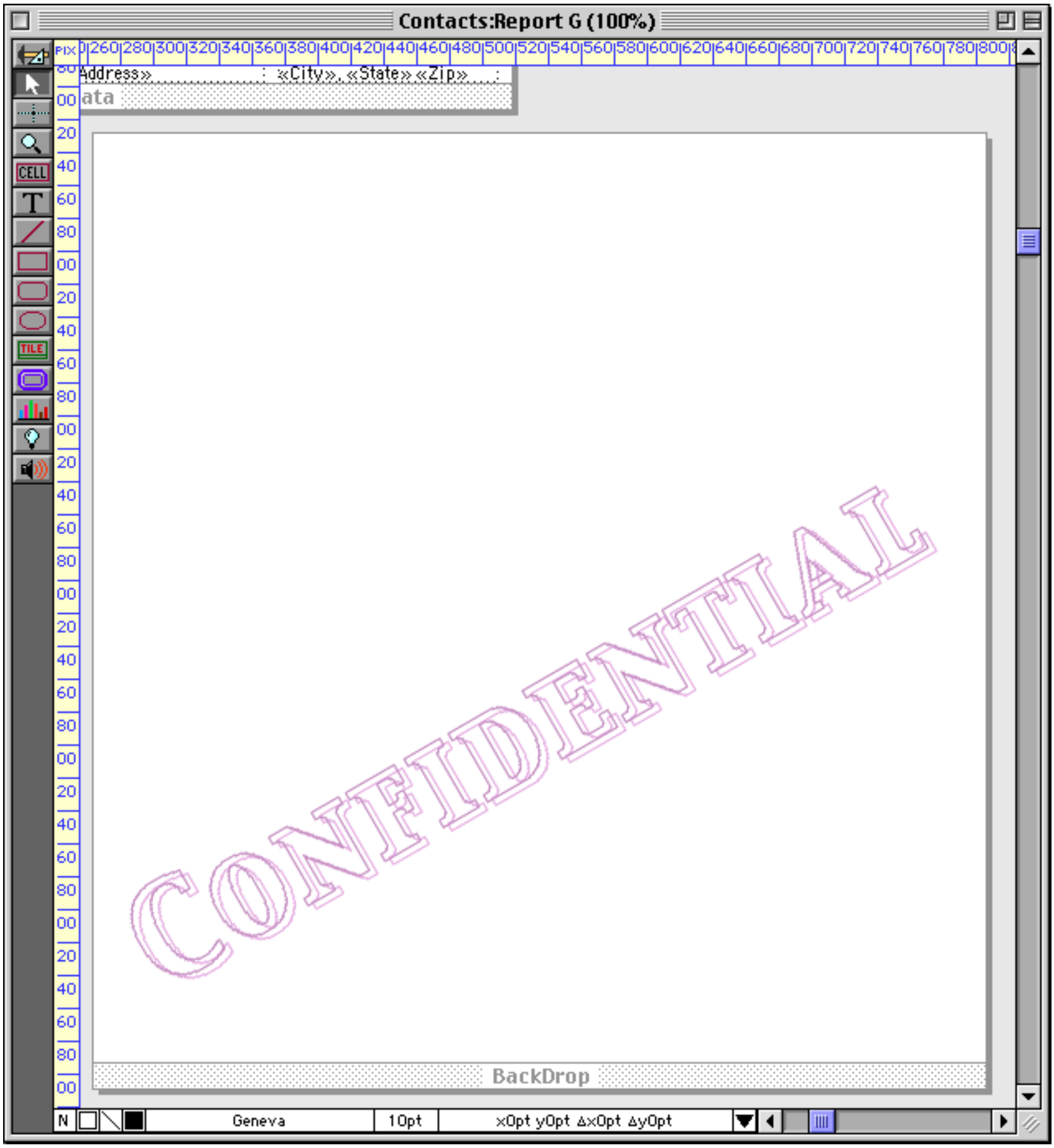
Name	Company	Street Address	City State Zip
Glen Knock	South Portland	909 Wescott Rd	South Portland, ME 04106
Carl Berg	C.B. Plumbing	161 Norton St	New Haven, CT 06511
Wes Lemarr		57 Hobart Ave	Rutherford, NJ 07070
Charles Dalbert	New York Lumber	171 Broadway	New York, NY 10003
Brad Hess	Brooklyn Lumber	128 70th St	Brooklyn , NY 11209
Tim Henry	Suffolk Lumber	2375 Driver Lane	Suffolk, VA 23435

Name	Company	Street Address	City State Zip
John Smith	Acme Widgets	12 Harmony Lane	Huntington Beach, CA
Susan Brown		783 Algonquin	Newport Beach, CA 93459
Karen Wilson	Evanston Lumber	498 Noyes	Evanston, IL 60201
Jim Nickle	Jim's Appliances	14189 8th	Newhall, CA 91321
Brian Felty	B.F. Plumbing	118 N Wilder	Lubbock, TX 79410
Bob Hanlan	Ann Arbor Lumber	6916 Morgan	Ann Arbor, MI 48104
Tim Daniels	St. Louis Lumber	3133 Cornell	St. Louis, MO 63130
John Moses		8265 Leticia	San Clemente, CA 92672
John Fabian		3 Rose Hill	Woodstock, VT 05091
Ed Ruth	Chicago Lumber	1580 N. Oconto	Chicago, IL 60634
Don Harmon	Sudderth Video	415 Sudderth	Ruidoso, NM 88345
Abe Fierstein	Van Nuys Lumber	1571 Haskell	Van Nuys, CA 91409
Randy Cross	Randy's Appliances	133 Hunt Rd	Chelsford, MA 01824
Jeffrey Rodman		2 Cary Rd	Chestnut Hill, MA 02167
Steve Jackson	Ann Arbor Lumber	389 Worden	Ann Arbor, MI 48103
Dick Hardlee		4151 Polstar	Plano, TX 75075
Don Meadows	Austin Lumber	1144 A West 6th	Austin, TX 78703
Jerry Bowen	Peacock Video	2847 Peacock	Highland, CA 92346
Thom Getchell	Thom's Appliances	543 Laurel	Menlo Park, CA 94025
Brian Smith	Brian's Appliances	1844 Tiburon	Hollister, CA 95023
David Blair	DB Printing	869 W. Temple	Lenox, IA 50851
Keith Baker	Northgate Video	552 Northgate	Lindenhurst, IL 60046

The **First Page Header** tile can also be used to create a title page for the report. To do this simply enlarge the **First Page Header** tile until it is large enough to cover an entire page. In that case Panorama will print only that tile on the first page. The regular report will begin on the second page.

BackDrop Tile

The backdrop tile can be used to print an overall graphic design on each page of the report. The backdrop tile actually overlays all of the other tiles on the page. For example, the backdrop tile could be used to print the word “Confidential” across each page of the report or to print a border around the page. Here’s an example of a backdrop tile.



The contents of the **Backdrop** tile will print on top of every page in the report.

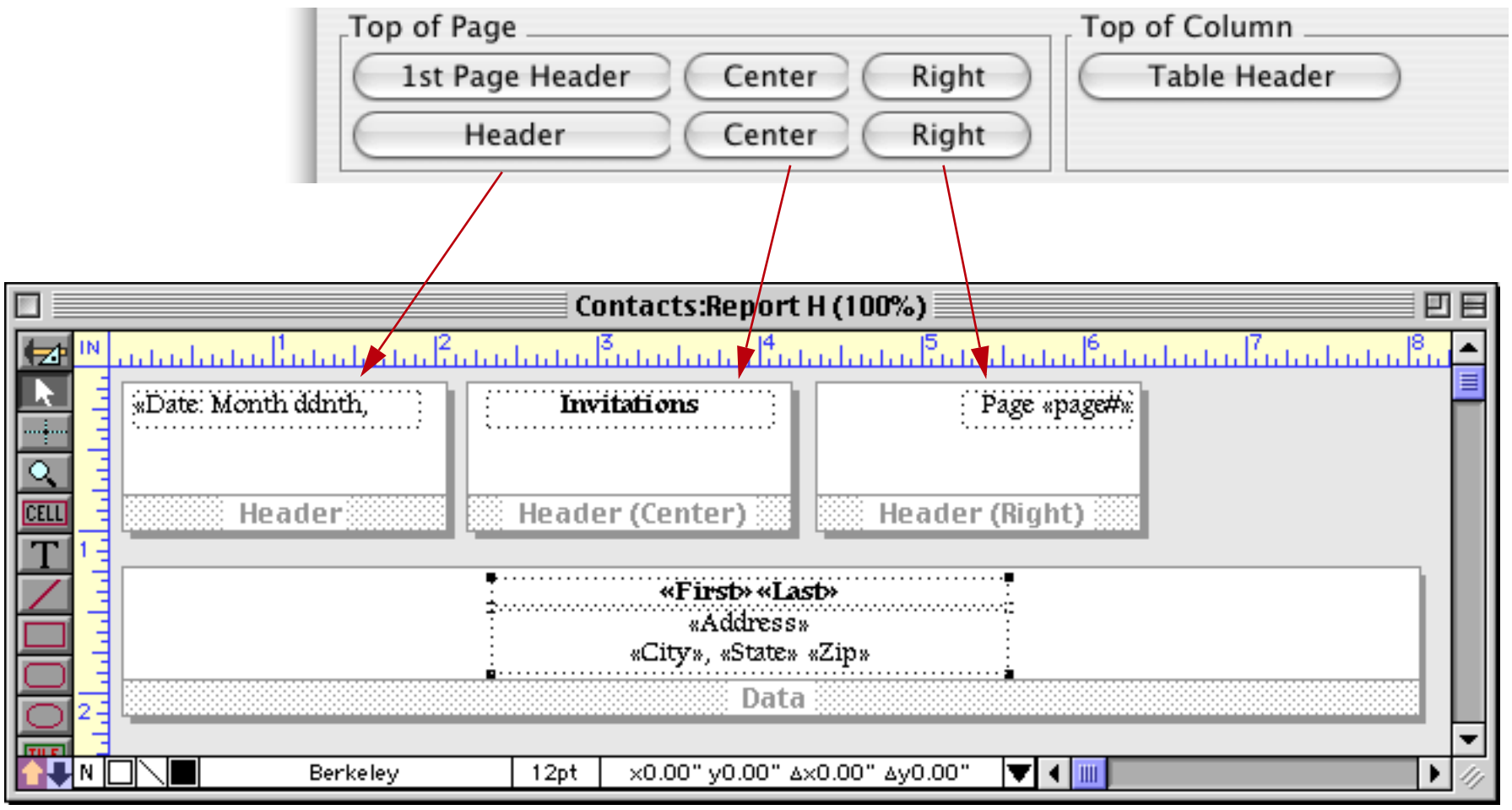
July 10, 2000 @ 3:47 PM Page 2

Name	Company	Street Address	City State Zip
Glen Knock	South Portland	909 Wescott Rd	South Portland, ME 04106
Carl Berg	C.B. Plumbing	161 Norton St	New Haven, CT 06511
Wes Lemarr		57 Hobart Ave	Rutherford, NJ 07070
Charles Dalbert	New York Lumber	171 Broadway	New York, NY 10003
Brad Hess	Brooklyn Lumber	128 70th St	Brooklyn , NY 11209
Tim Henry	Suffolk Lumber	2375 Driver Lane	Suffolk, VA 23435
Ramsey French	West Palm Beach	8206 13th Way	West Palm Beach, FL
Steve Miller	SM Printing	3894 11th Court	Jupiter, FL 33458
Joseph Doll	Joseph's Appliances	2650 Helen Rd	Shaker Heights, OH 44122
John Maguire	Akron Lumber	39 Beck Ave	Akron, OH 44302
Jerry Boone		6125 Park Drive	Traverse City, MI 49684
John Bath	J.B. Plumbing	8864 Ave	Mendota Heights, MN
Sam Pack		6051 Pheasant	Inverness, IL 60067
David Cohn		307 Ronnie Drive	Buffalo Grove, IL 60089
Tim Moran		220 East Parkway	Wheaton, IL 60187
Ed Swanson	D.S. Plumbing	683 Elm St	Batavia, IL 60510
Steve Dallas	Chaminade Video	1 Chaminade	Creve Coeur, MO 63141
Charles Pierce	Midland Lumber	1662 Durant	Midland, TX 79705
Steve West	S.W. Plumbing	1175 Wilson Rd	Fountain, CO 80817
Mary Bilbury	M.B. Plumbing	2754 Parkway	Beverly Hills, CA 90210
Charles Michaels		5238 Quince	Upland, CA 91786
Herb Dang	Herb's Appliances	206 Phelps St	San Francisco, CA 94124
Sari Rattner	S.R. Plumbing	495 N.E. 62Rd	Seattle, WA 98115
Leslie Bianchi		23 Oak St	Lexington, MA 02173
Peter Yarensky	Peter's Appliances	41 Elm St	Dover, NH 03820
Cheryll Howell	Gray Lumber	4 Fran Circle	Gray, ME 04039
Tom Love		53 Clubhouse Drive	Woodbury, CT 06798
Patrick Dowd		26 Catalpa Rd	Convent Station, NJ 07961
Jeffrey Funk	Jeffrey's Appliances	7 Elwood Ave	Flemington, NJ 08822
Craig Resth	H.H. Plumbing	54 Parkway Drive	North Chili, NY 14514
Jules Silk	J.S. Plumbing	9338 Waltham Rd	Cheltenham, PA 19012
Bela Hackman	Bela's Appliances	3132 Glengarry	Memphis, TN 38128
Jerry Levan		883 Boone Trail	Richmond, KY 40475
Frank Stelle	Jim's Appliances	58272 Auburn Rd	Fort Wayne, IN 46825
Thomas Cupal	Ann Arbor Lumber	8 Medford Court	Ann Arbor, MI 48104
Anne Crane	Grosse Pointe Shores	11 Moorland Drive	Grosse Pointe Shores, MI
Joseph Bizzarri	JB Printing	7045 Mandel	Westchester, IL 60153
Janel Rundlett	J.R. Plumbing	8601 Fairfax	Kansas City, KS 66115
Henry Hultquist	Lincoln Lumber	1197 S. 17th	Lincoln, NE 68502
Jerry Wilson		3050 North Main	Sand Springs, OK 74063

CONFIDENTIAL

Designing Headers and Footers For Changing Page Sizes

Most reports are designed for a specific page size. It is possible, however, to design headers and footers that automatically adjust for different page sizes. The key is to divide the header and footer into three components: left flush, centered, and right flush. Here is a report with three headers.



On a standard 8 1/2 by 11 page this report will look like this.

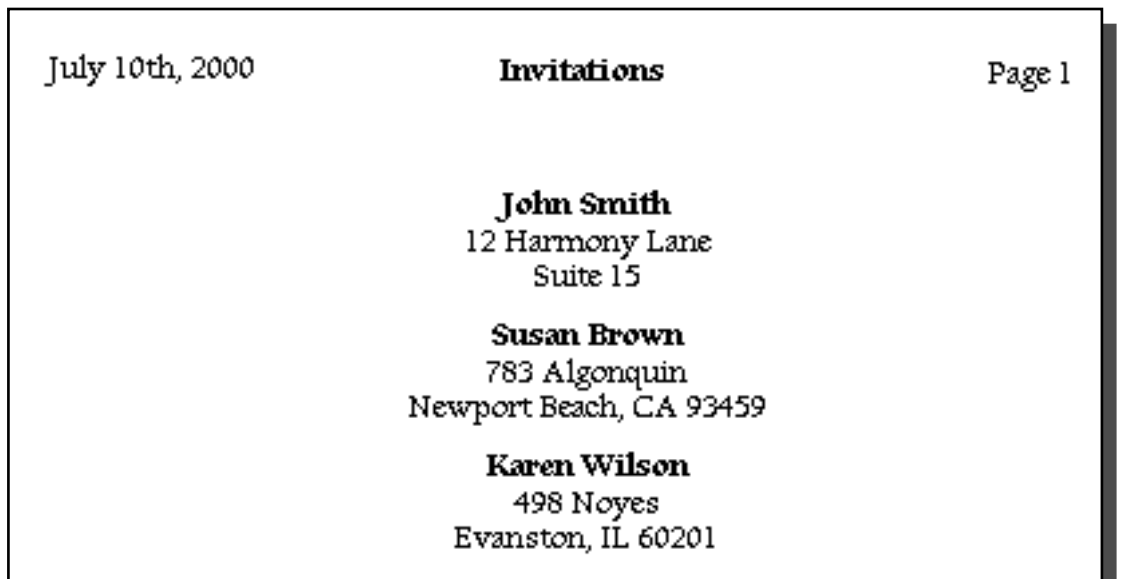
July 10th, 2000 **Invitations** Page 1

John Smith
12 Harmony Lane
Suite 15

Susan Brown
783 Algonquin
Newport Beach, CA 93459

Karen Wilson
498 Noyes
Evanston, IL 60201

If you print this report on a different size page (or switch from portrait to landscape orientation or use a different page reduction factor) the titles will adjust automatically.


July 10th, 2000 **Invitations** Page 1

John Smith
12 Harmony Lane
Suite 15

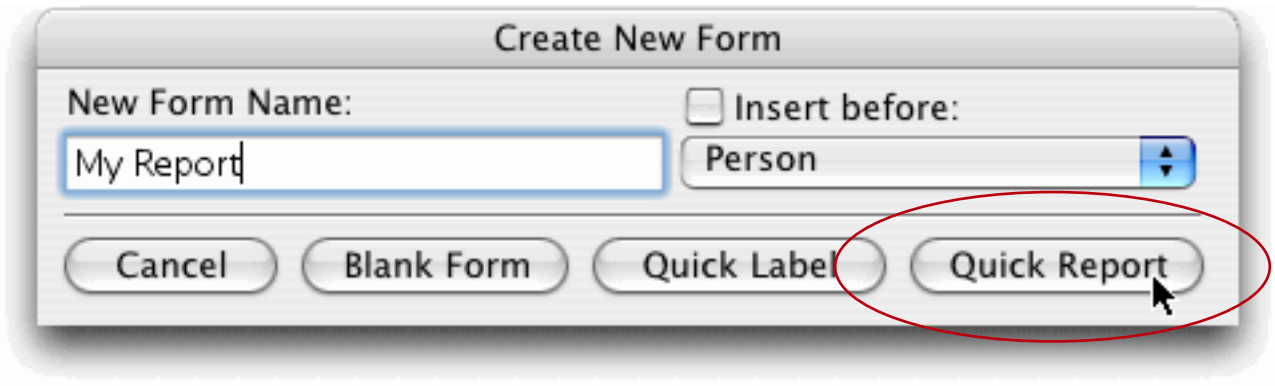
Susan Brown
783 Algonquin
Newport Beach, CA 93459

Karen Wilson
498 Noyes
Evanston, IL 60201

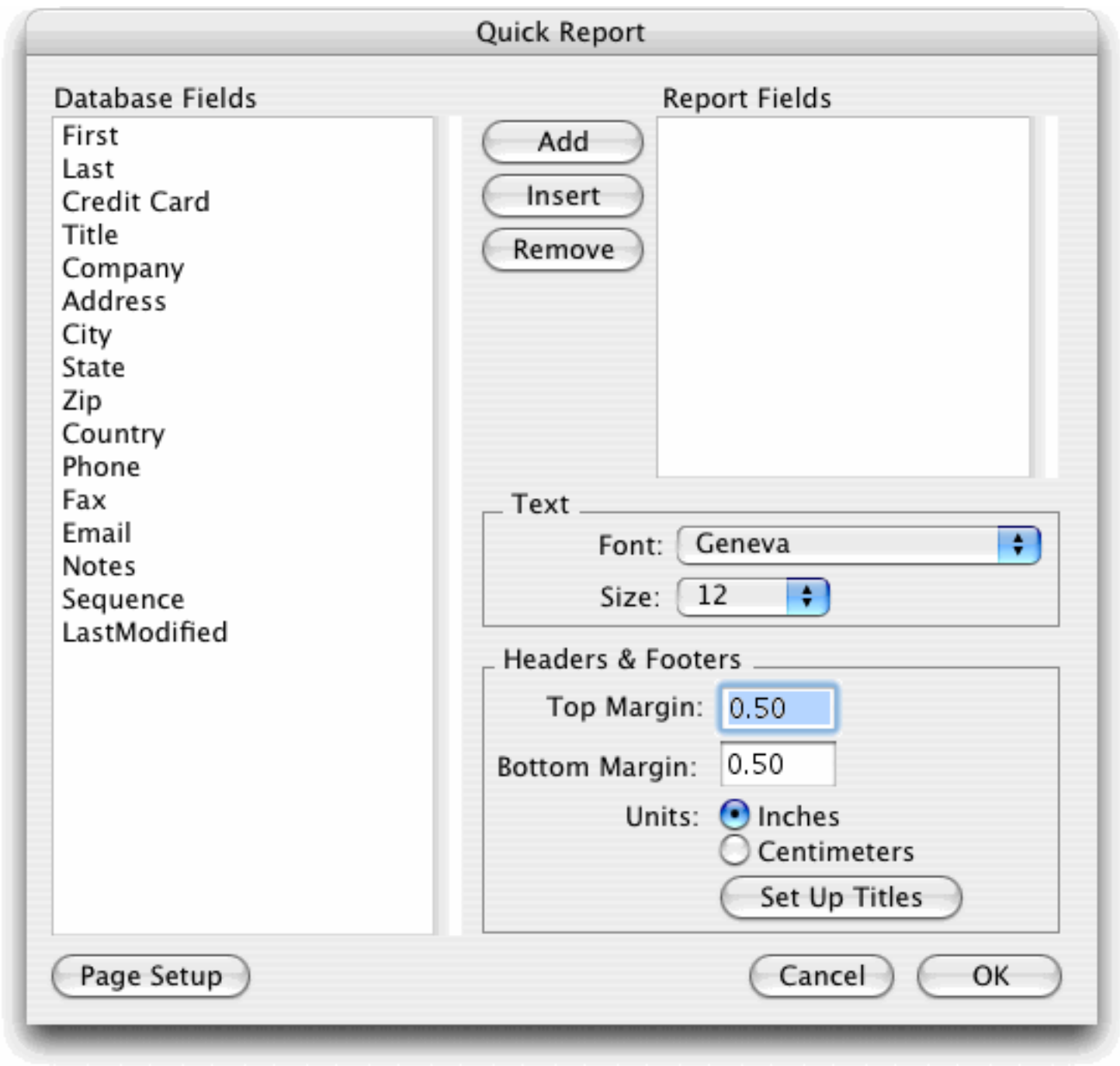
You don't need to use this technique unless you think that the page size may change. If the page size is fixed, just use the regular header and footer tiles.

The QuickReport Dialog

When you create a new form, Panorama gives you the option of creating a blank form or automatically creating a label or report.



The **QuickReport** button opens a dialog that can do most or all of the work of setting up a report for you. The QuickReport dialog allows you to automatically create a tabular report.



On the left is a list of all the fields in the database. On the right is a list of the fields that will be included in the report. In the dialog the report fields are listed from top to bottom, in the actual report they will be printed from left to right. To set up the report, copy the fields you want to include in the report from the list of fields on the left to the report list on the right.

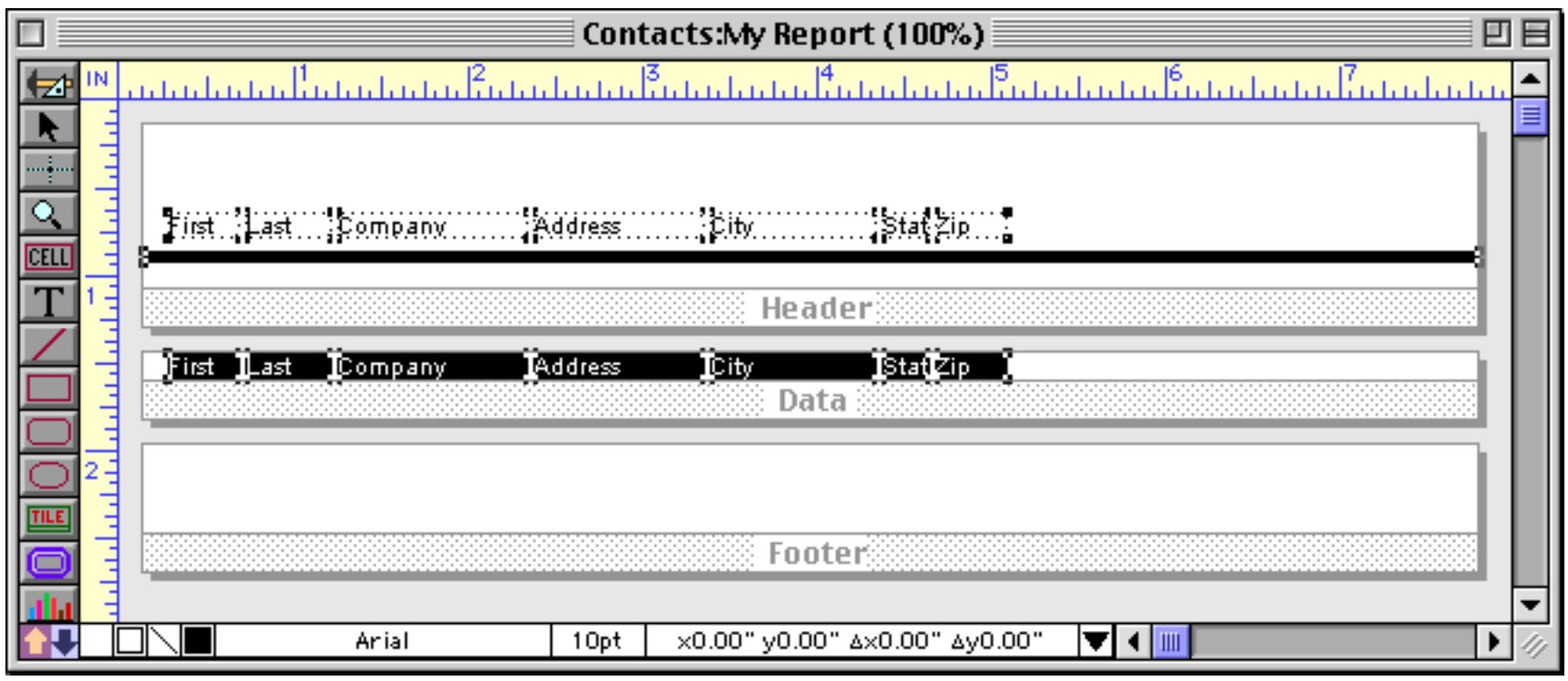
The easiest way to copy a field from the left to the right is to double click on the field name. Double clicking on a field name adds the field to the end of the list on the right. If a field name on the right is already selected, then double clicking on a field name on the left replaces that field. The **Add** button works the same as double clicking.

To insert a field into the middle of the list on the right, first select a field on the right, then select a field on the left. Press the **Insert** button to insert the new field into the list on the right.

To delete a field from the list on the right, select the field and press the **Remove** button. You can also delete a field from the list on the right by double clicking on it.

If you want to use a non-standard page size, you should use the **Page Setup** button to set it up. Be sure to set up the page size before you press the **OK** button. This lets QuickReport know how wide the page will be.

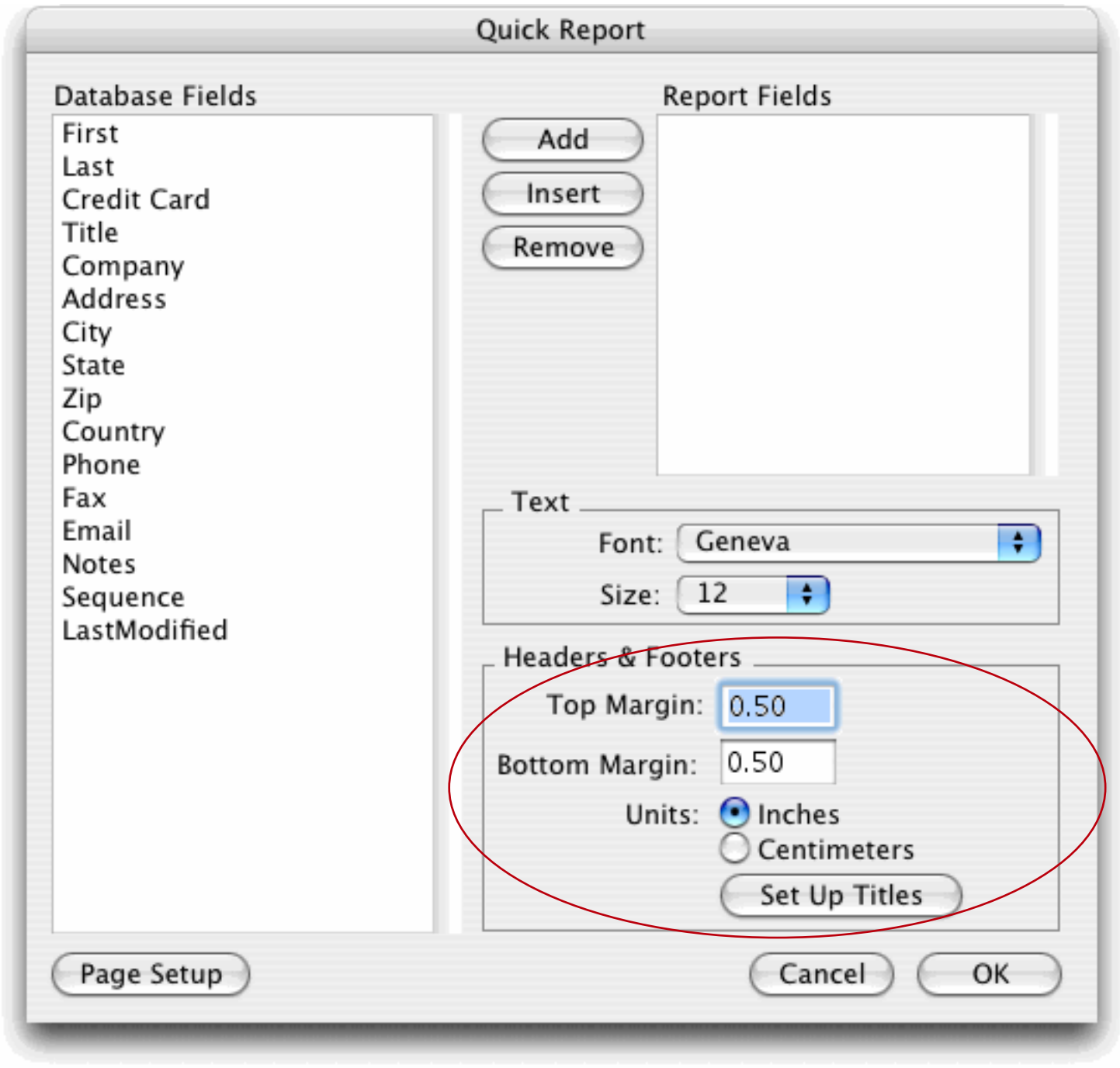
Press the **OK** button to generate the actual report. Panorama will generate all the tiles and cells needed for the report.



You can use this report as is, or you can use it as a starting point and modify it to suit your needs.

First	Last	Company	Address	City	Stat	Zip
John	Smith	Acme Widgets	12 Harmony	Huntington	CA	9264
Susan	Brown		783 Algonquin	Newport	CA	9345
Karen	Wilson	Evanston	498 Noyes	Evanston	IL	6020
Jim	Nickle	Jim's	14189 8th	Newhall	CA	9132
Brian	Felty	B.F. Plumbing	118 N Wilder	Lubbock	TX	7941
Bob	Hanlan	Ann Arbor	6916 Morgan	Ann Arbor	MI	4810
Tim	Daniels	St. Louis	3133 Cornell	St. Louis	MO	6313
John	Moses		8265 Leticia	San Clemente	CA	9267
John	Fabian		3 Rose Hill	Woodstock	VT	0509
Ed	Ruth	Chicago	1580 N.	Chicago	IL	6063
Don	Harmo	Sudderth Video	415 Sudderth	Ruidoso	NM	8834

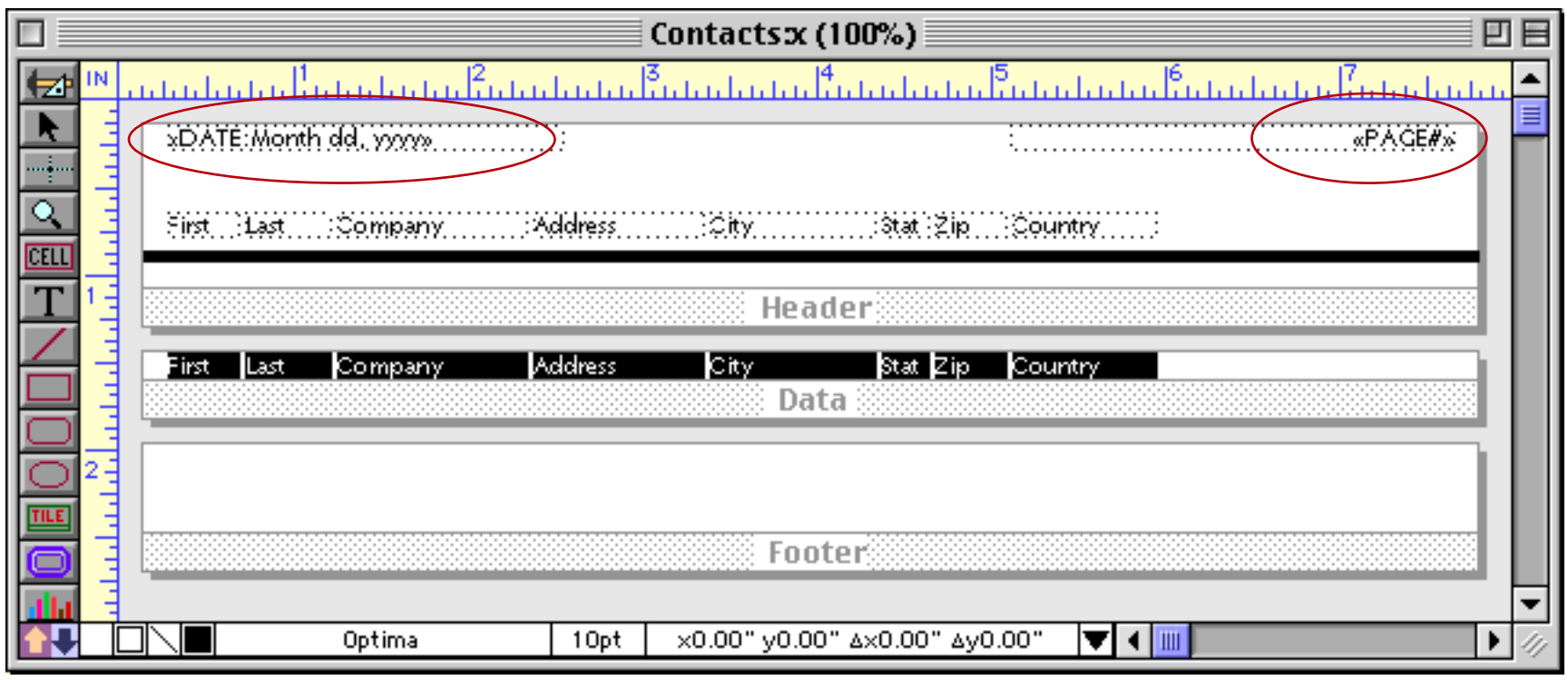
The QuickReport dialog also creates header and footer tiles. You can specify the height of the header and footer by specifying the top and bottom margins (in either inches or centimeters).



The QuickReport dialog can place page numbers and/or the date and time on the header and footer tiles. To set up the page numbers and date/time, press the **Set Up Titles** button. This button opens a small dialog with 16 checkboxes.



These checkboxes allows you to place any combination of page number, date, and time in up to four positions on the page: top left, top center, top right, and bottom center. Here is the form generated with the options shown above.

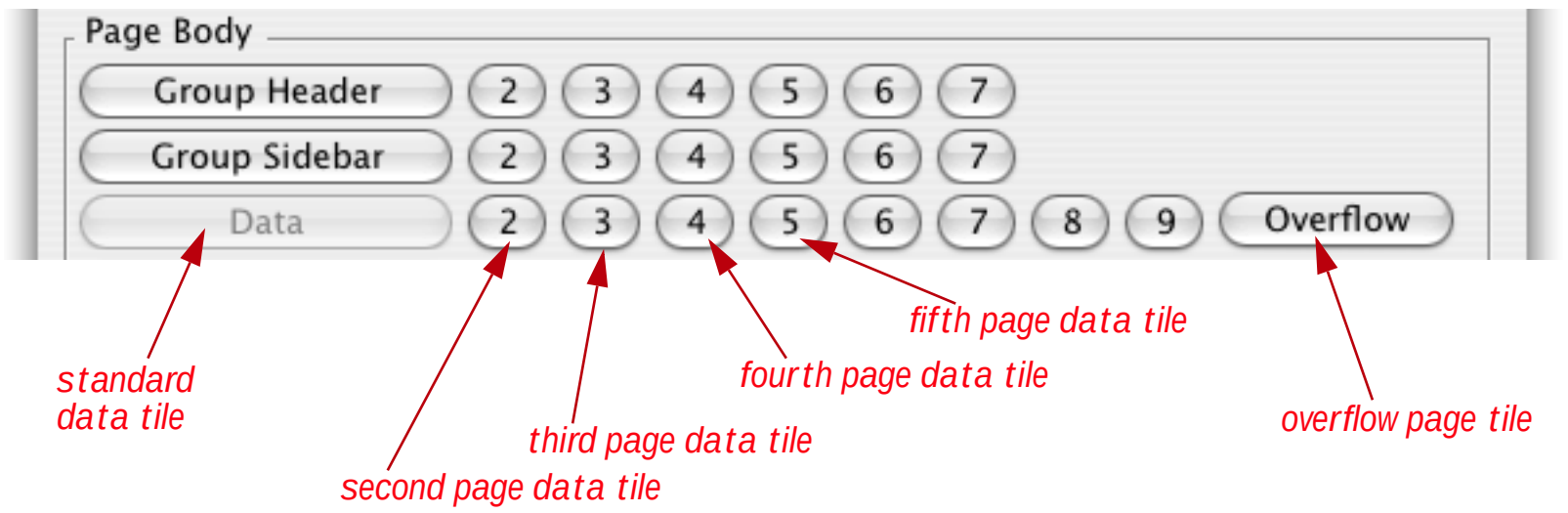


To learn more about printing the page number and date see “Page Numbers” on page 1088 and “Printing the Current Date and Time” on page 1091.

Printing Multiple Page Records

If the data tile is made large enough to fill the entire page then Panorama will print only one record per page. In some cases, a single page is not enough. For example, you may need to print an invoice, statement or tax return with several pages per record. Panorama can print up to 10 pages for each record.

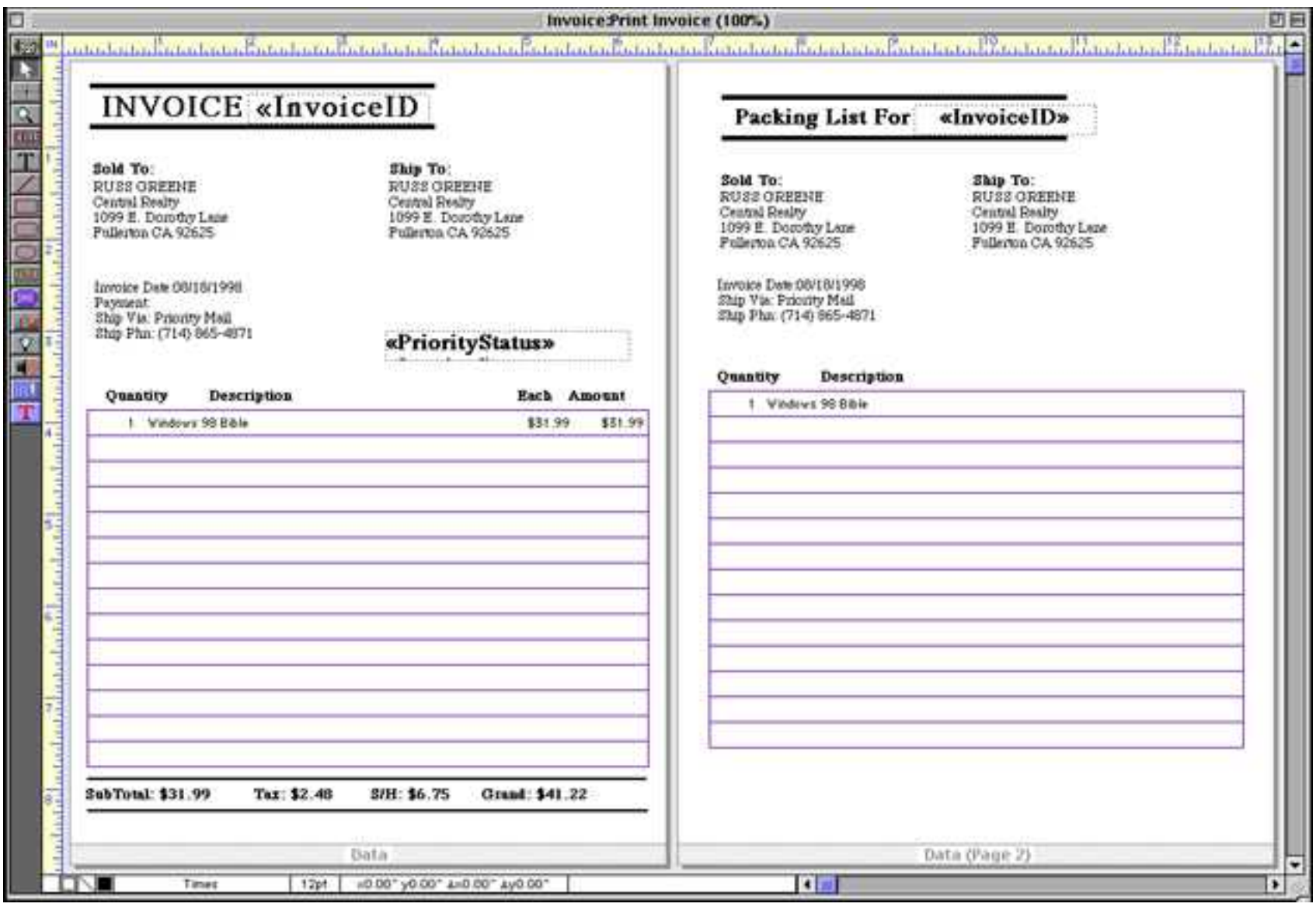
To print a multiple page form you will need to create multiple data tiles. These additional data tiles are created with the **Specialized Tile** configuration dialog.



The tile’s for additional pages can be identified by the page number in the drag bar of the tile.



When you create a data tile for an additional page it should be large enough to fill the entire sheet of paper, just like the first data tile. When a form contains data tiles for multiple records Panorama will ignore any header or footer tiles. Only the margins and data tiles will be printed. This illustration shows a form for simultaneously printing an invoice and packing slip. (Sorry — had to reduce this image to make both pages fit!)



When you print this form, both pages will print automatically for each record printed. If you use the **Print One Record** tool (see "Print One Record" on page 1048) Panorama will automatically print the current invoice and packing slip. If you use the regular **Print** command Panorama will print an invoice and packing slip for each selected record.

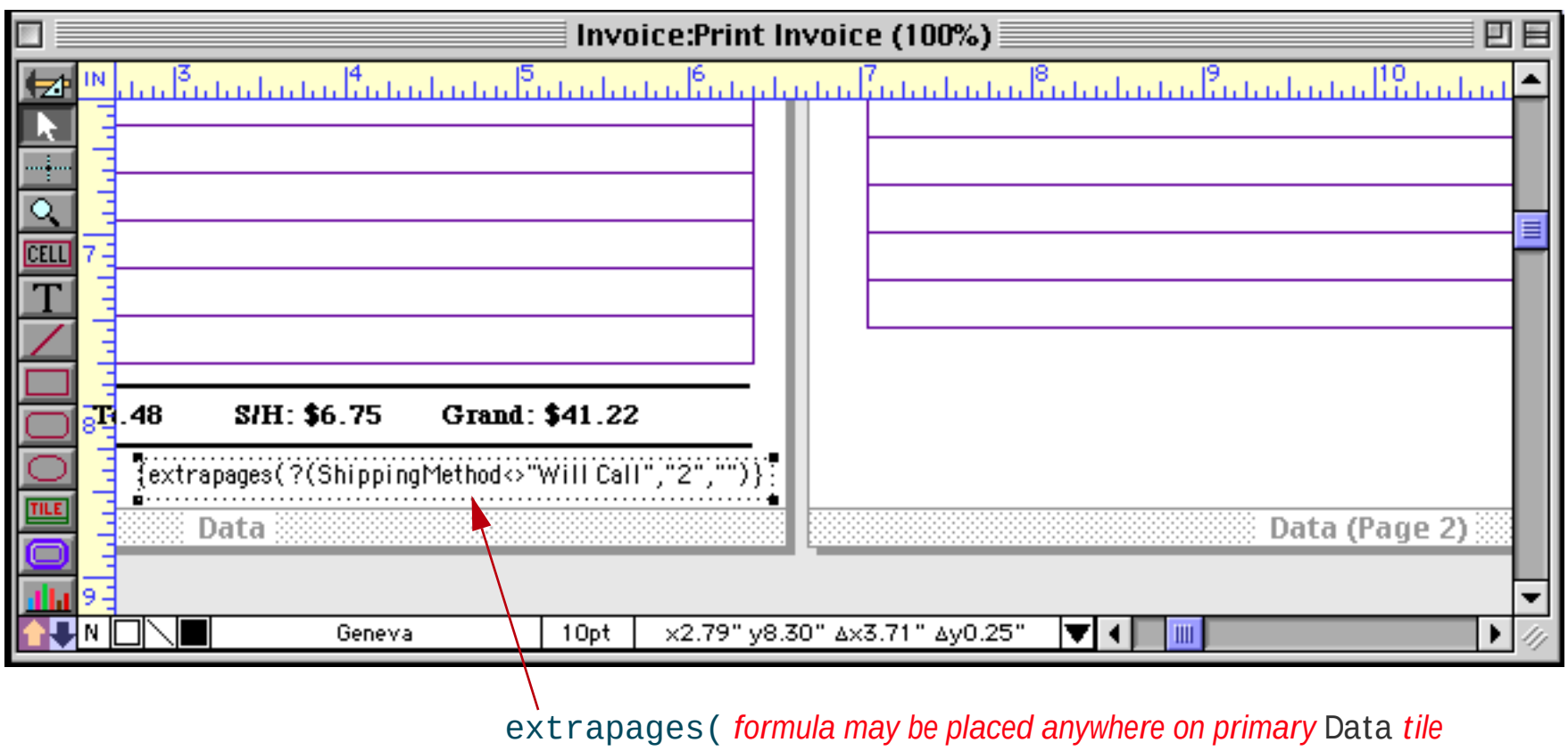
Selectively Printing Multiple Pages per Record

Panorama allows up to 9 pages to be printed per record using data tiles 1 through 9. However, you do not have to print all tiles for each record, you can print them selectively using a formula on the form. For example, you can easily ask Panorama to print pages 1, 2, and 5 for one record and 1 and 4 for another record (page one, the primary data tile, always prints). You could use this feature to selectively print an envelope, to print only selected portions of a tax return, to print one or two page invoices, etc.

To control which extra data tiles get printed, you must use the `extrapages(<pagelist>)` function (see "EXTRAPAGES(" on page 5213). This function must be put in a formula in an auto wrap text object somewhere on the main data tile. For example if you want to print pages 1, 2, and 5 you would need to use the formula

```
{extrapages("25")}
```

Usually you would use a field or variable to selectively control which pages print instead of a fixed string. To illustrate this let's modify the invoice example from the previous section. This form would always print both an invoice and a packing slip. By adding an `extrapages(` function we can modify the form so that a packing slip is not printed if the shipping method is Will Call.



The `extrapages(` function may be placed anywhere on the primary Data tile. The function has no result, so it won't cause anything to be printed. In this case if the shipping method is Priority Mail, FedEx, UPS, or anything except Will Call the `?(` function will produce "2" and the second page (the packing list) will print. If the shipping method is Will Call the `?(` function will produce "" and the packing list will not print.

Printing Data that Overflows a Page

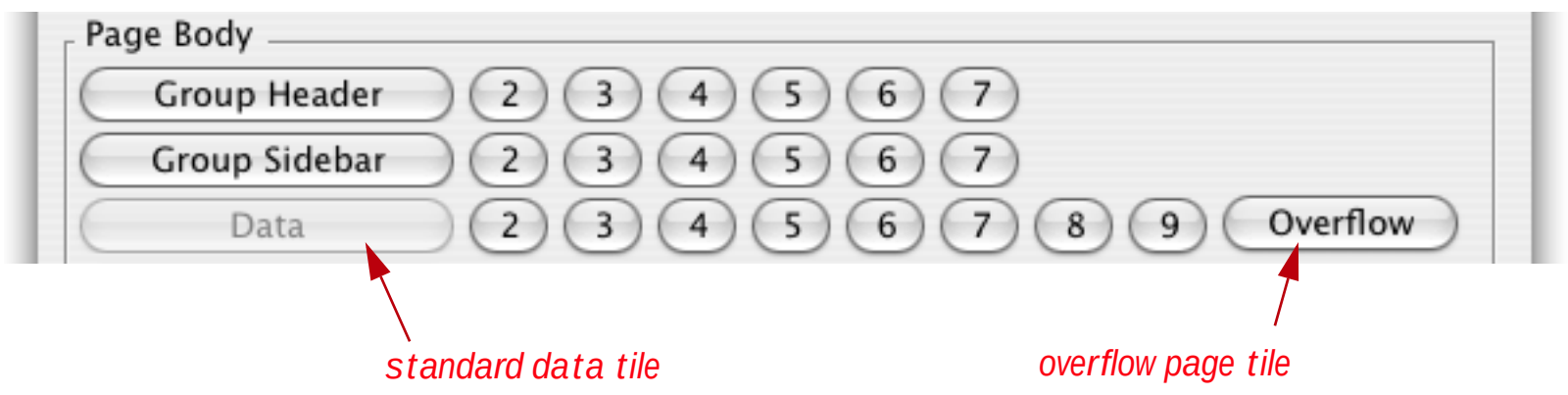
With some restrictions, Panorama allows a report to print data records that won't fit on a single page. For example, this feature could be used to print a multiple page letter. If the letter doesn't fit on the first page it spills over onto multiple pages using the data overflow tile.

To print data in a single record that doesn't fit on a single page, the data must meet the following criteria:

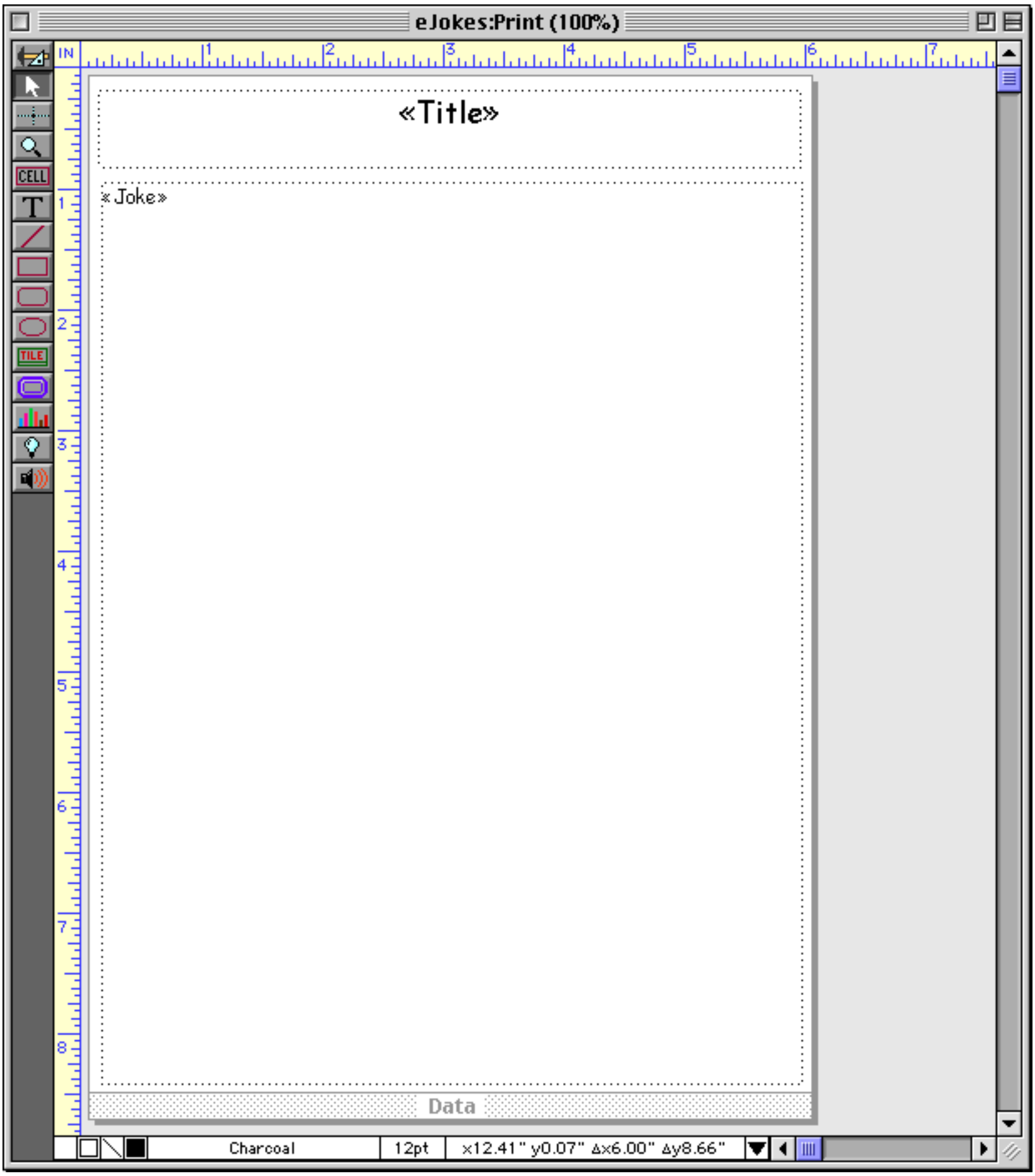
1) The data must be contained in a single graphic object.

2) The graphic object must be an auto-wrap text object, Word Processor SuperObject, or Super Flash Art object.

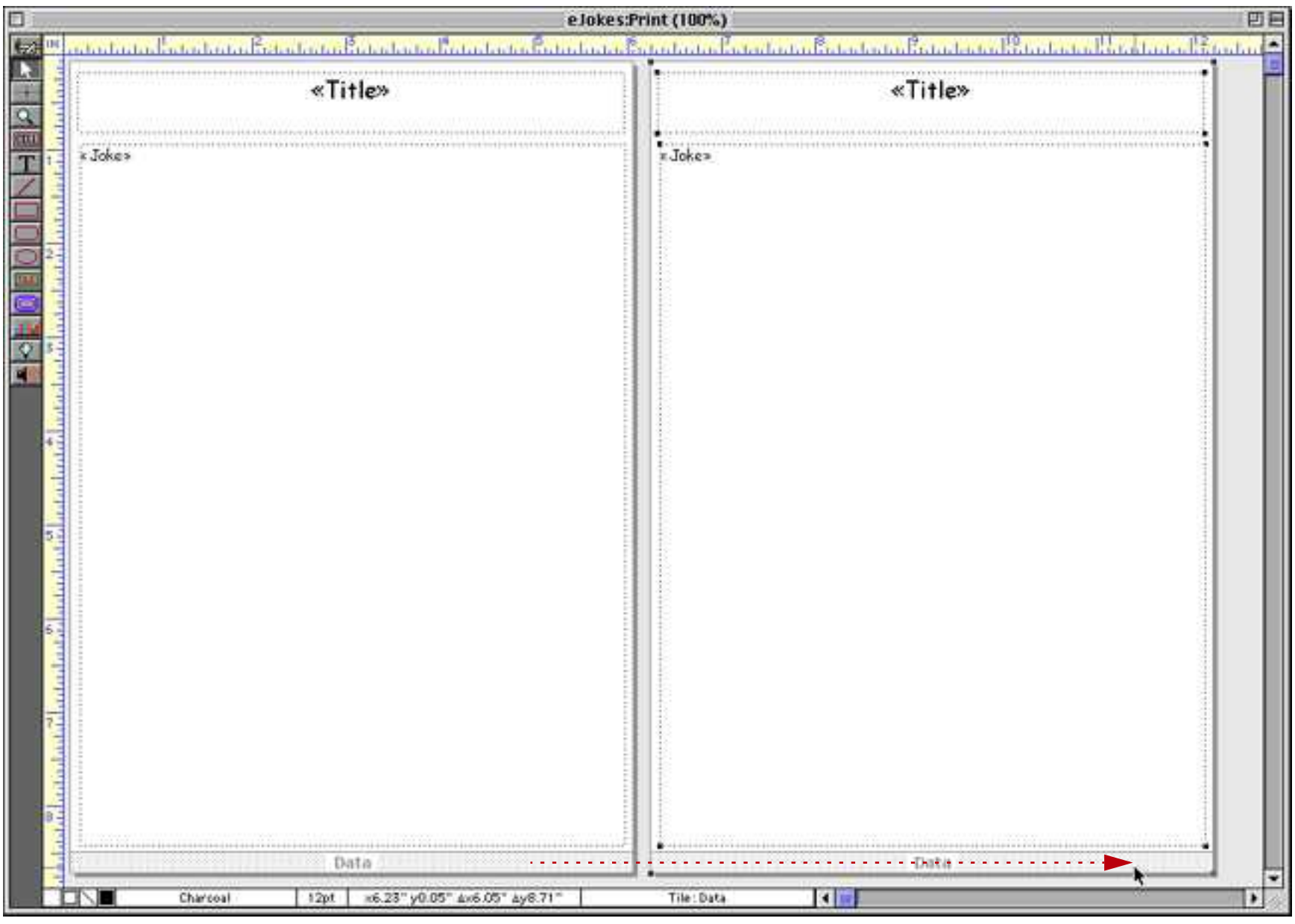
To print data that overflows a page you'll need two tiles: a data tile and an overflow tile. The overflow tile is labeled (…) in the Specialized tile configuration dialog.



Usually the best way to create a form with an overflow tile is to start with a single page form like this.



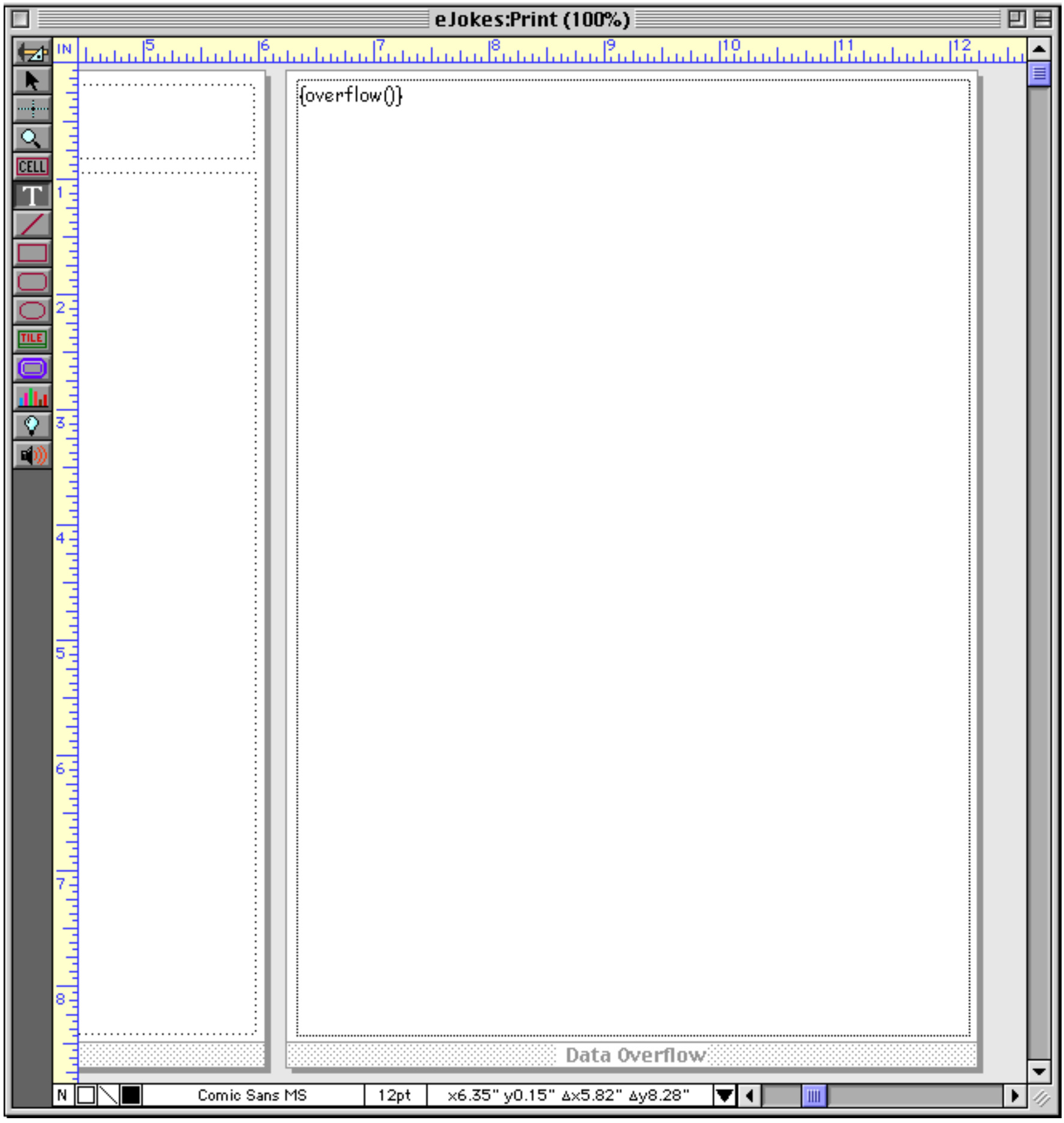
Double click the tile drag bar to select the tile and all the objects on it (see "Working with Tiles" on page 1050). Then duplicate these objects by holding down the Option key (Macintosh) or Alt key (PC), and clicking on the tile's drag bar and dragging it (see "Drag Duplicating" on page 561). You may also want to hold down the **Shift** key to keep the new tile perfectly aligned with the original. The result is two data tiles, like this.



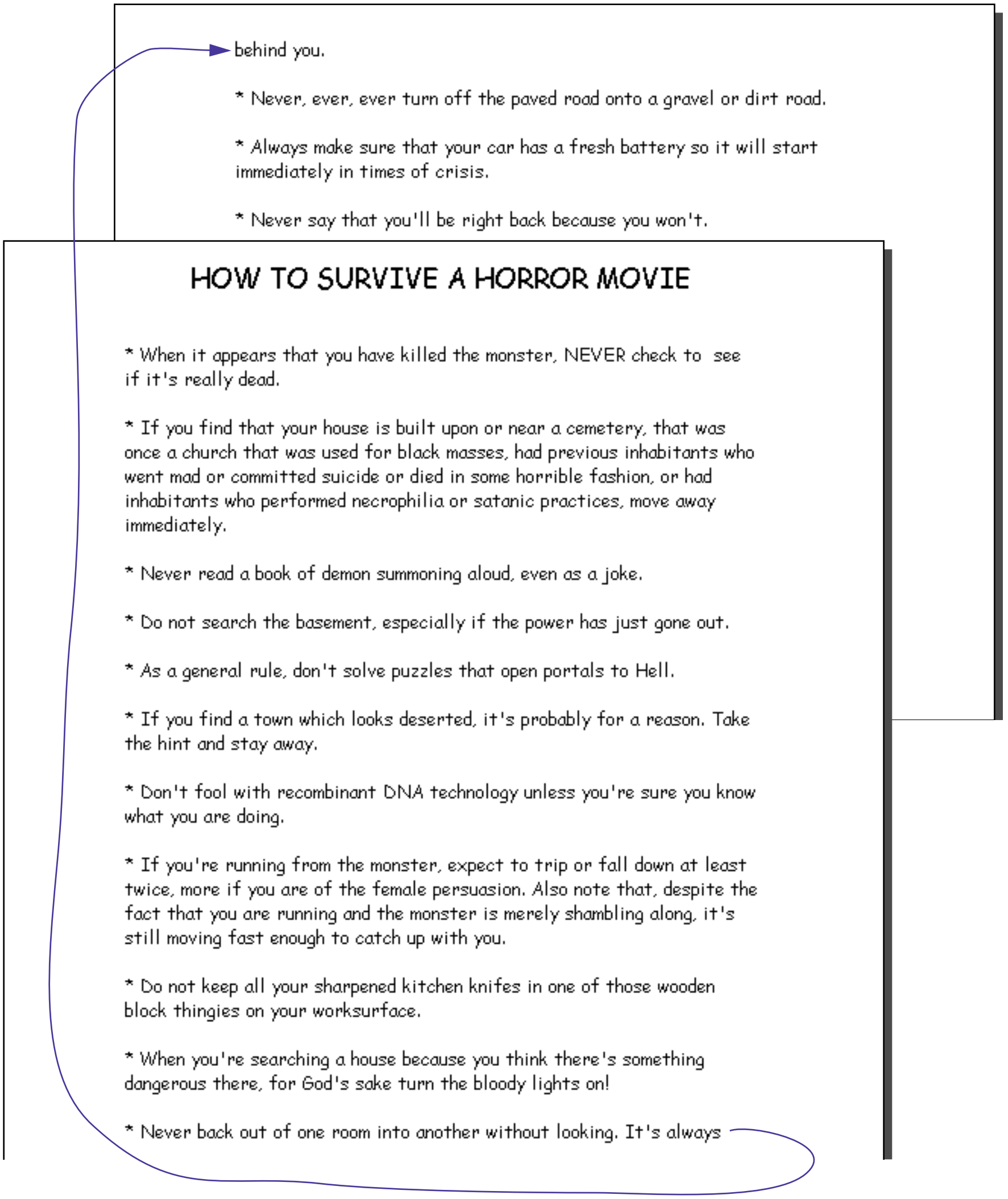
Double click on the title of the second data tile to change it to an overflow tile.



Now adjust the graphics on the overflow tile to the format you want for the second, third, and succeeding pages. If you are using auto-wrap text to display the text you'll need to type in the `overflow()` function as shown in the illustration below.



That's all there is too it! When you print a record that contains more than one page of data, Panorama will automatically print as many additional pages as is necessary to display all of the data. The blue arrow shows how the text overflows from one page to the next.



behind you.

* Never, ever, ever turn off the paved road onto a gravel or dirt road.

* Always make sure that your car has a fresh battery so it will start immediately in times of crisis.

* Never say that you'll be right back because you won't.

HOW TO SURVIVE A HORROR MOVIE

* When it appears that you have killed the monster, NEVER check to see if it's really dead.

* If you find that your house is built upon or near a cemetery, that was once a church that was used for black masses, had previous inhabitants who went mad or committed suicide or died in some horrible fashion, or had inhabitants who performed necrophilia or satanic practices, move away immediately.

* Never read a book of demon summoning aloud, even as a joke.

* Do not search the basement, especially if the power has just gone out.

* As a general rule, don't solve puzzles that open portals to Hell.

* If you find a town which looks deserted, it's probably for a reason. Take the hint and stay away.

* Don't fool with recombinant DNA technology unless you're sure you know what you are doing.

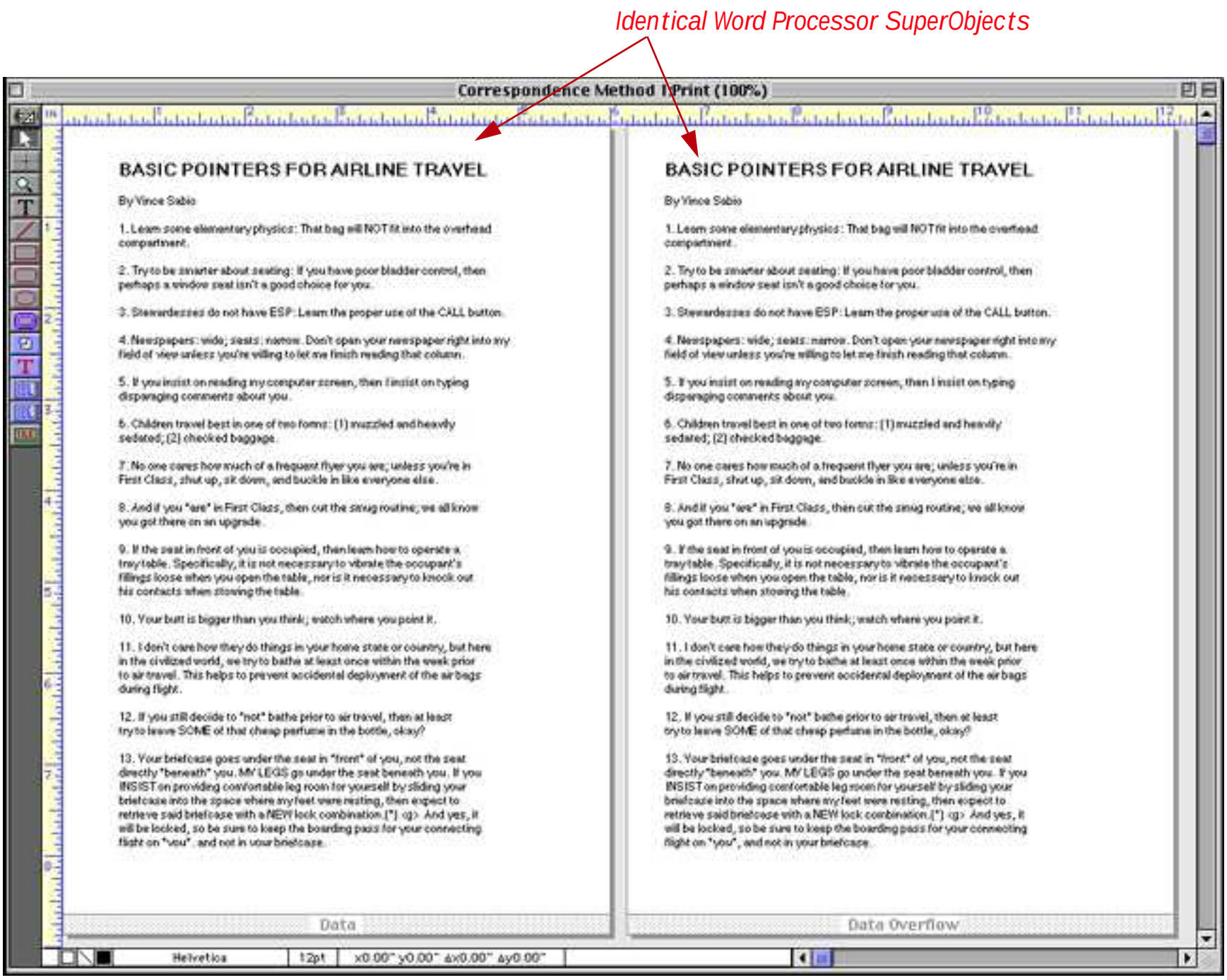
* If you're running from the monster, expect to trip or fall down at least twice, more if you are of the female persuasion. Also note that, despite the fact that you are running and the monster is merely shambling along, it's still moving fast enough to catch up with you.

* Do not keep all your sharpened kitchen knifes in one of those wooden block thingies on your worksurface.

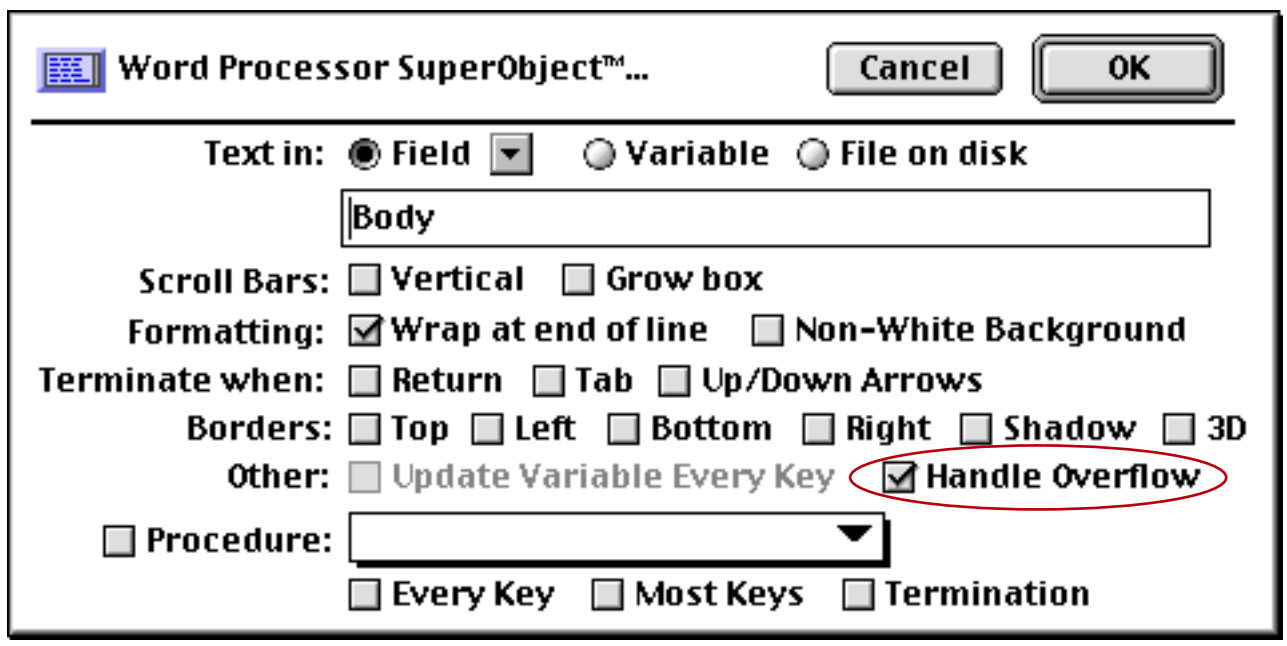
* When you're searching a house because you think there's something dangerous there, for God's sake turn the bloody lights on!

* Never back out of one room into another without looking. It's always

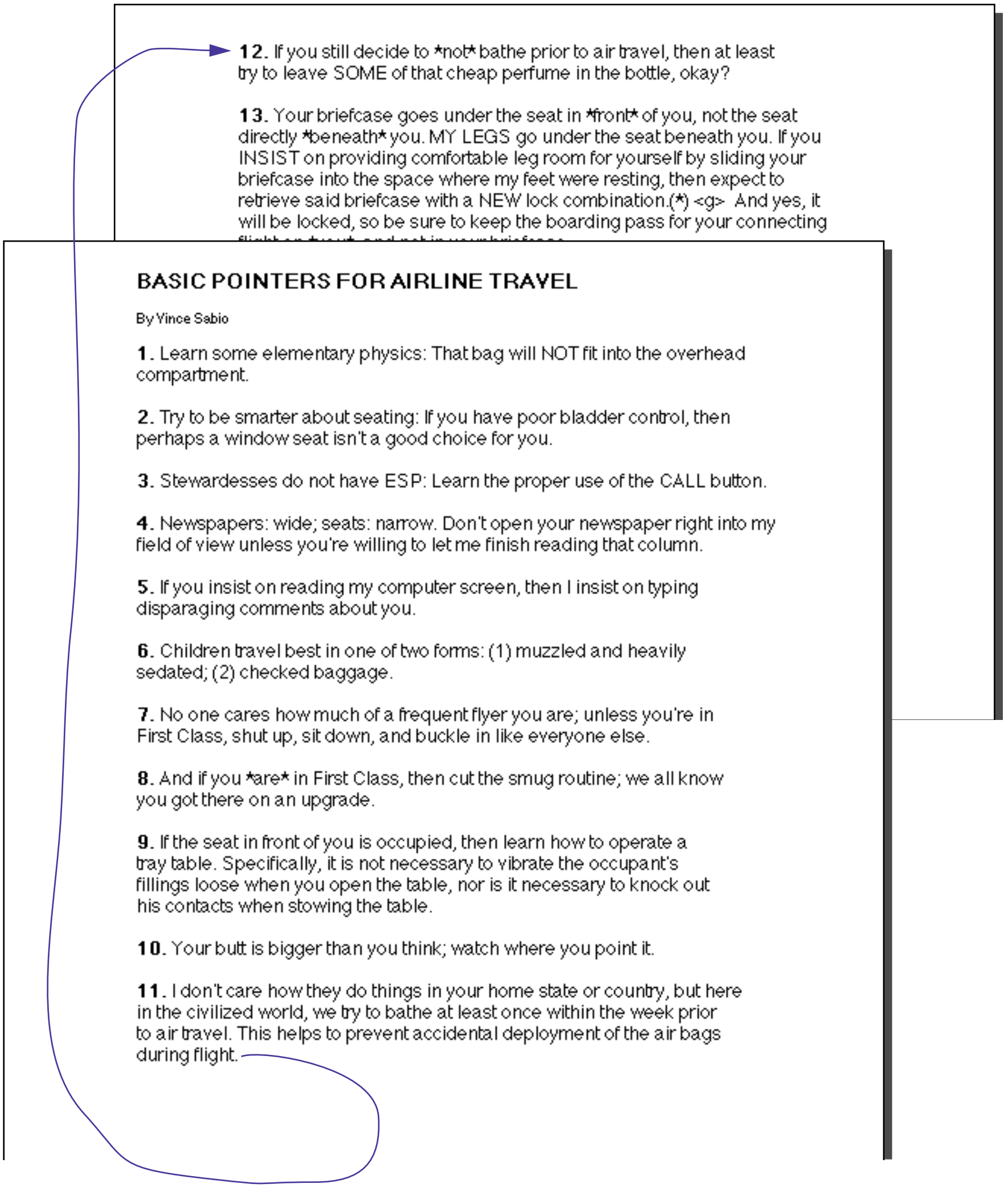
Printing overflow pages with the Word Processor SuperObject (see "Printing Word Processor Documents" on page 719) is almost the same as with the auto-wrap text object. Instead of two auto-wrap text objects, you have two Word Processor SuperObjects.



Both of these Word Processor objects should be configured exactly the same, and both must have the Handle Overflow option turned on.



Just as with the auto-wrap text objects, the text wraps from one page to the next.



12. If you still decide to *not* bathe prior to air travel, then at least try to leave SOME of that cheap perfume in the bottle, okay?

13. Your briefcase goes under the seat in *front* of you, not the seat directly *beneath* you. MY LEGS go under the seat beneath you. If you INSIST on providing comfortable leg room for yourself by sliding your briefcase into the space where my feet were resting, then expect to retrieve said briefcase with a NEW lock combination.(*) <g> And yes, it will be locked, so be sure to keep the boarding pass for your connecting

BASIC POINTERS FOR AIRLINE TRAVEL

By Vince Sabio

1. Learn some elementary physics: That bag will NOT fit into the overhead compartment.

2. Try to be smarter about seating: If you have poor bladder control, then perhaps a window seat isn't a good choice for you.

3. Stewardesses do not have ESP: Learn the proper use of the CALL button.

4. Newspapers: wide; seats: narrow. Don't open your newspaper right into my field of view unless you're willing to let me finish reading that column.

5. If you insist on reading my computer screen, then I insist on typing disparaging comments about you.

6. Children travel best in one of two forms: (1) muzzled and heavily sedated; (2) checked baggage.

7. No one cares how much of a frequent flyer you are; unless you're in First Class, shut up, sit down, and buckle in like everyone else.

8. And if you *are* in First Class, then cut the smug routine; we all know you got there on an upgrade.

9. If the seat in front of you is occupied, then learn how to operate a tray table. Specifically, it is not necessary to vibrate the occupant's fillings loose when you open the table, nor is it necessary to knock out his contacts when stowing the table.

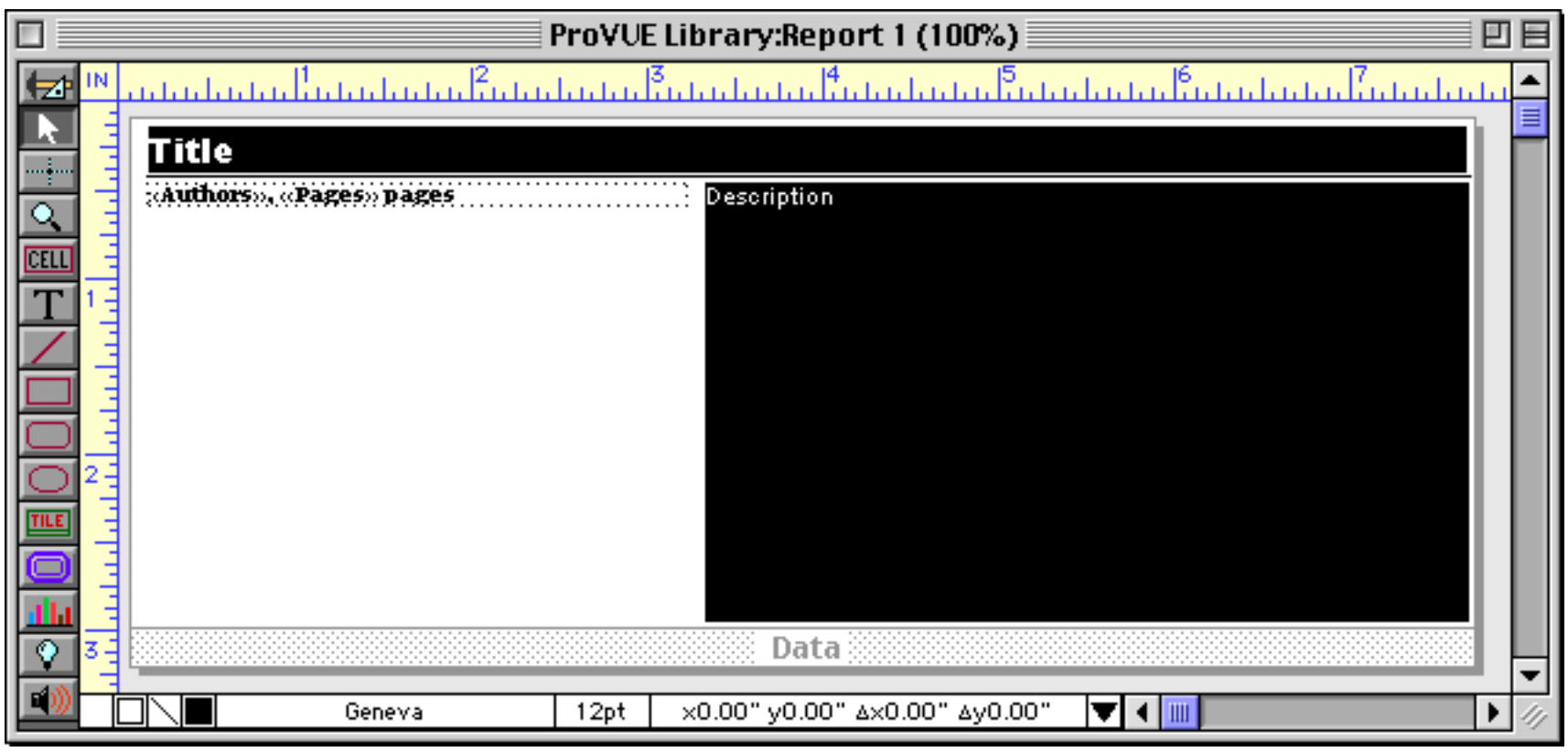
10. Your butt is bigger than you think; watch where you point it.

11. I don't care how they do things in your home state or country, but here in the civilized world, we try to bathe at least once within the week prior to air travel. This helps to prevent accidental deployment of the air bags during flight.

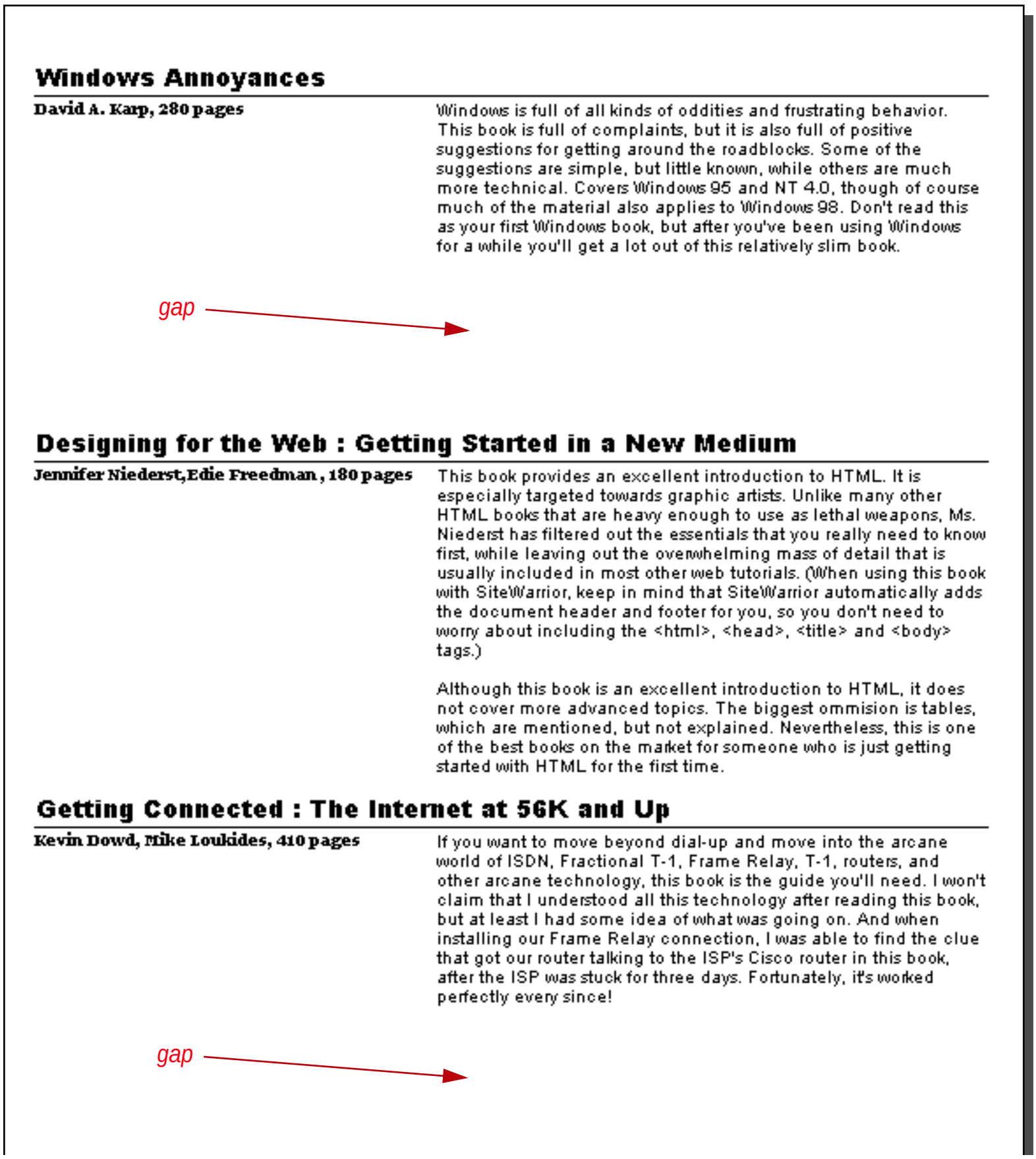
In the same way you can also wrap a tall image from page to page with the Super Flash Art object (see “Flash Art™” on page 748). This only works for images taller than a page - it does not work for images that are wider than a page.

Variable Height Records

In most reports each record is evenly spaced down the page. In some databases, however, the amount of data in each record may vary dramatically. For example, a database of catalog items may contain a description field whose contents vary dramatically. Some descriptions may be only a few words long, while others are several hundred words long. When a database like this is printed with a fixed height per record there will be large gaps between some of the records. For example, consider this form designed for printing a catalog of books.

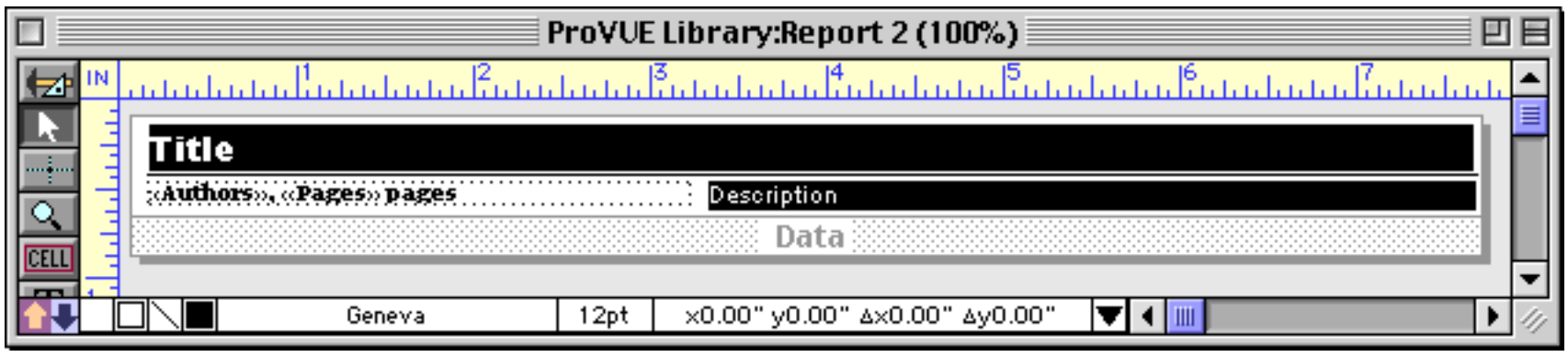


Since this form is designed to print the longest possible description, it will leave gaps when printing shorter book descriptions.



To print this type of database without leaving these large gaps between the records the report must be printed with variable height records. Contrary to what you might think at first, creating a report with variable height records does not involve using a special report tile. Instead, you design the report for the minimum possible size (in this case a one line description) and tell Panorama which objects may need to expand to accommodate extra data. As the report prints, Panorama will automatically expand these objects, and also adjust any other objects that may need to make room.

Here is a revised version of our book catalog form that is designed to print the minimum possible amount — a one line description.



Of course when this report is printed it isn't what we want at all - only the first line of each description is printed. (On the other hand, we have gotten rid of the gaps between the records!)

Windows Annoyances

David A. Karp, 280 pages Windows is full of all kinds of oddities and frustrating behavior.

Designing for the Web : Getting Started in a New Medium

Jennifer Niederst,Edie Freedman , 180 pages This book provides an excellent introduction to HTML. It is

Getting Connected : The Internet at 56K and Up

Kevin Dowd, Mike Loukides, 410 pages If you want to move beyond dial-up and move into the arcane

Frontier: The Definitive Guide

Matt Neuburg, 560 pages This book covers everything Frontier 4.2 from A to Z. It's not for the

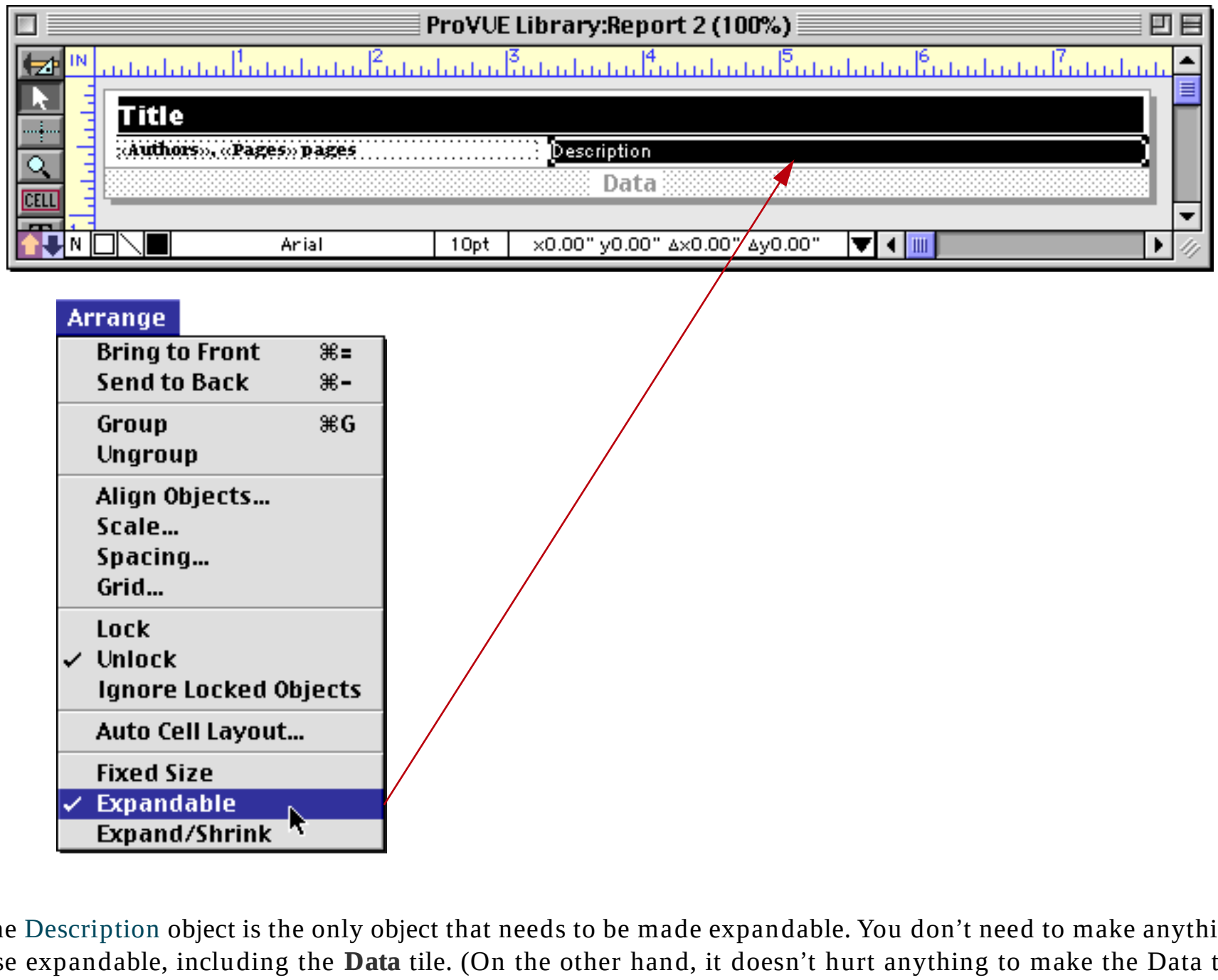
HTML : The Definitive Guide (Nutshell Handbook)

Chuck Musciano,Bill Kennedy, 552 pages This book contains a complete, informative reference to all

GIF Animation Studio: Animating Your Web Site

Richard Koman, 180 pages There's only one kind of animation that will work on virtually any

To fix this problem we need to select the Description object and make it **Expandable**.



The Description object is the only object that needs to be made expandable. You don't need to make anything else expandable, including the **Data** tile. (On the other hand, it doesn't hurt anything to make the Data tile expandable. Since the Data tile doesn't contain any data itself it cannot expand on it's own. When a data cell (or other variable height object) expands, the tile it is placed on expands automatically. Although you can make any type of object variable height, it only makes sense for objects that can contain variable height data—data cells, auto-wrap text, and flash art.)

Now that the Description object is expandable we get a very nice printed report that even Goldilock's could love — each record is not too long, not too short, but just right.

Windows Annoyances

David A. Karp, 280 pages

Windows is full of all kinds of oddities and frustrating behavior. This book is full of complaints, but it is also full of positive suggestions for getting around the roadblocks. Some of the suggestions are simple, but little known, while others are much more technical. Covers Windows 95 and NT 4.0, though of course much of the material also applies to Windows 98. Don't read this as your first Windows book, but after you've been using Windows for a while you'll get a lot out of this relatively slim book.

Designing for the Web : Getting Started in a New Medium

Jennifer Niederst,Edie Freedman , 180 pages

This book provides an excellent introduction to HTML. It is especially targeted towards graphic artists. Unlike many other HTML books that are heavy enough to use as lethal weapons, Ms. Niederst has filtered out the essentials that you really need to know first, while leaving out the overwhelming mass of detail that is usually included in most other web tutorials. (When using this book with SiteWarrior, keep in mind that SiteWarrior automatically adds the document header and footer for you, so you don't need to worry about including the <html>, <head>, <title> and <body> tags.)

Although this book is an excellent introduction to HTML, it does not cover more advanced topics. The biggest ommision is tables, which are mentioned, but not explained. Nevertheless, this is one of the best books on the market for someone who is just getting started with HTML for the first time.

Getting Connected : The Internet at 56K and Up

Kevin Dowd, Mike Loukides, 410 pages

If you want to move beyond dial-up and move into the arcane world of ISDN, Fractional T-1, Frame Relay, T-1, routers, and other arcane technology, this book is the guide you'll need. I won't claim that I understood all this technology after reading this book, but at least I had some idea of what was going on. And when installing our Frame Relay connection, I was able to find the clue that got our router talking to the ISP's Cisco router in this book, after the ISP was stuck for three days. Fortunately, it's worked perfectly every since!

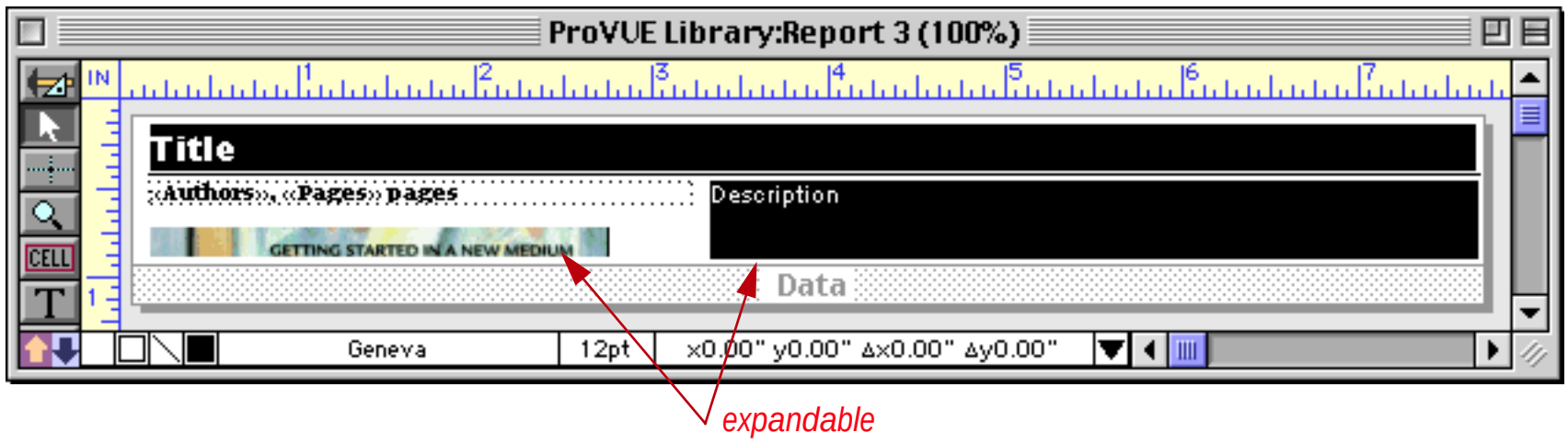
Frontier: The Definitive Guide

Matt Neuburg, 560 pages

This book covers everything Frontier 4.2 from A to Z. It's not for the faint of heart, but if you want to master Frontier, this book is complete and clear. The approach is much like a college class, starting with fundamentals and gradually building up to more advanced topics - leaving nothing out. In other words, this isn't a "Learning Frontier for Dummies" or "Build Web Sites With Frontier in 20 Minutes."

The Frontier web stuff is in a couple of chapters at the back. I was suprised at how little Frontier has in the way of web support. Though you could do many things with it, you really have to do a lot of programming to make web sites happen, especially compared to SiteWarrior. When I got a chance to meet Matt Neuburg at a trade show and show him SiteWarrior, he said "That's how Frontier should work!"

It's possible to place two or more expandable objects side-by-side in a form. This report contains two expandable objects: a Flash Art SuperObject and a Data Cell.



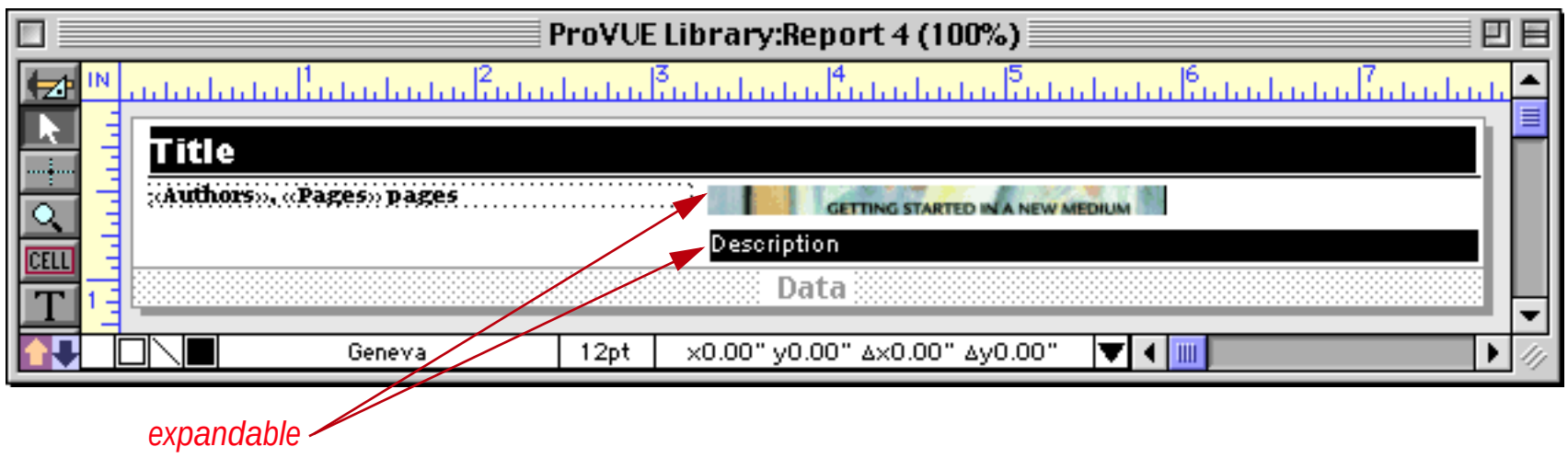
When you place expandable objects side-by-side it's important to make sure that the bottom edges of the expandable objects are perfectly aligned. You can make sure they are aligned properly with the **Align Objects** command (see "Aligning Objects" on page 553). When the objects are perfectly aligned Panorama will adjust the form for the largest object. In this example either the Flash Art or the text may be larger — the form will be adjusted for whichever is larger.



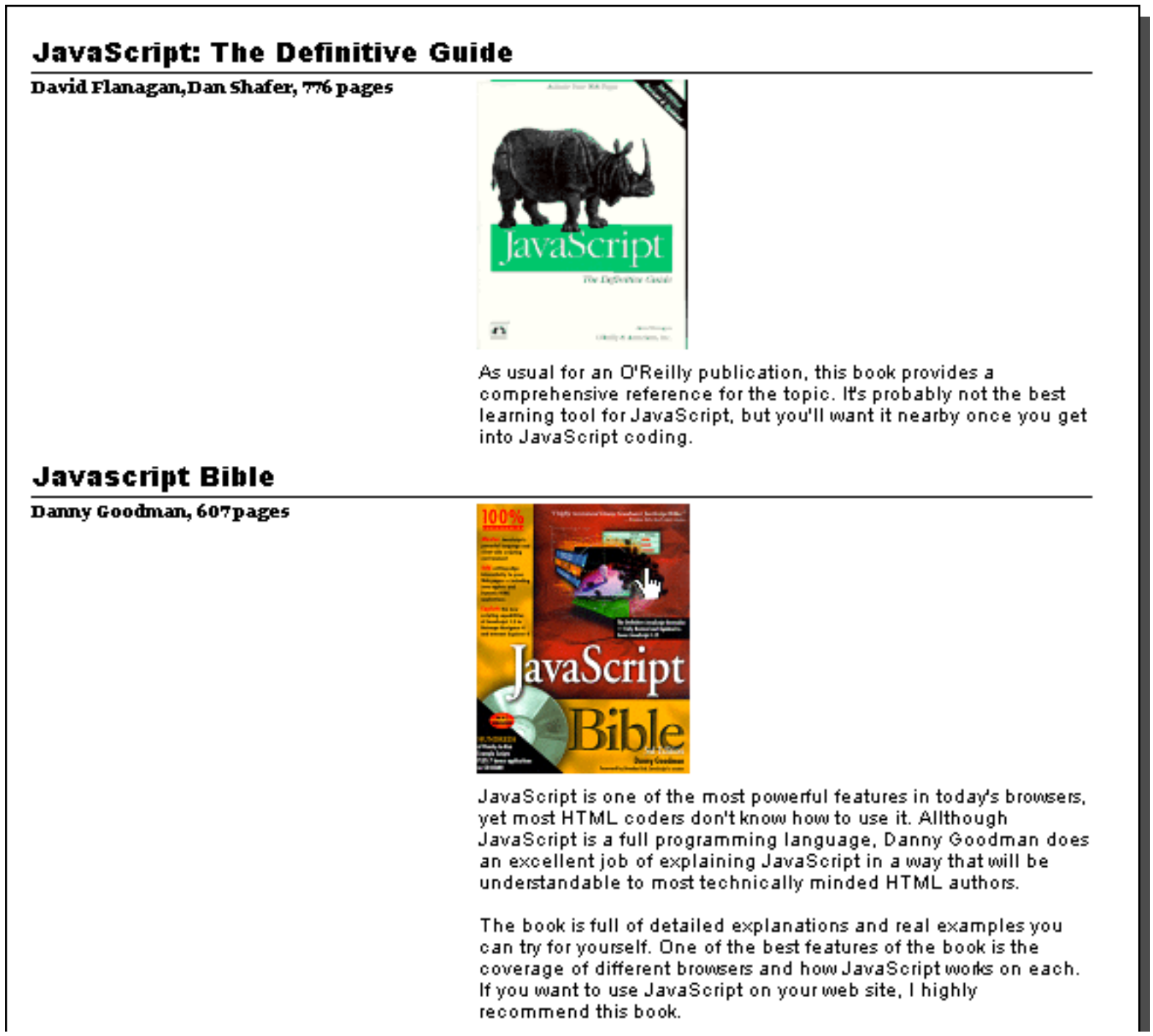
Any graphic element below an expandable object will shift downward as the expandable object expands. Any graphic element surrounding a variable height object will expand as that object expands. For instance, if a variable height cell has a box around it, the box will expand automatically as the cell expands. You don't need to check the **Expandable** option for the box.

Stacking Variable Height Objects

In the previous section expandable objects were arranged side by side. You can also stack variable height items vertically.



As each cell expands, it shifts the material below it, including any other expandable objects.



You can arrange expandable objects horizontally or vertically, but not both horizontally and vertically in the same report. The result of such a combination is usually unpredictable.

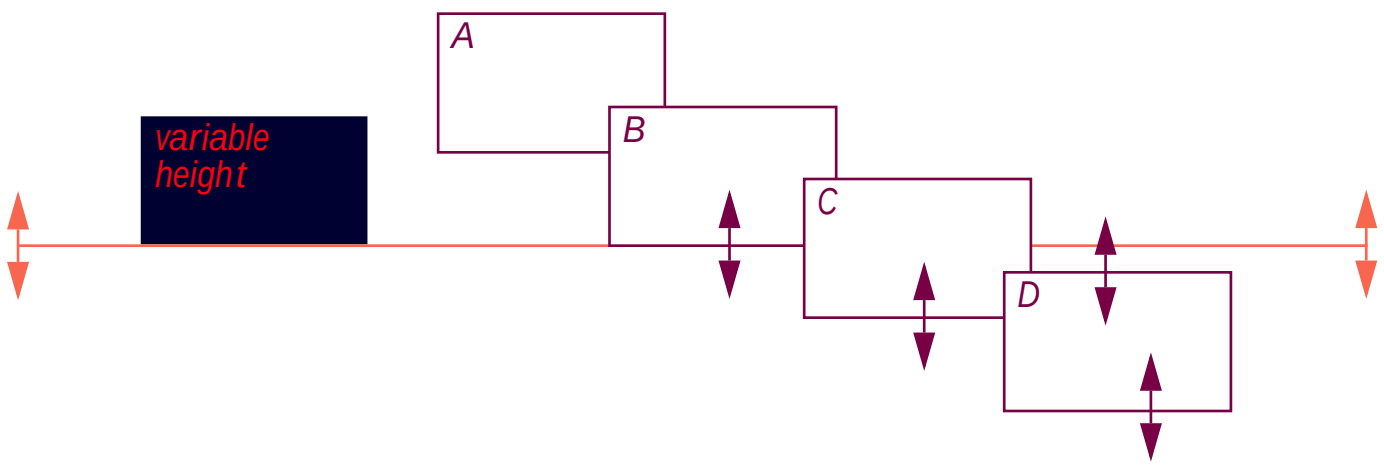
The Expand/Shrink Option

Expandable cells can grow to accommodate large amounts of data, but they can never get smaller. The **Expand/Shrink** option allows variable height objects to both grow and shrink as the data changes. If there is no data at all the object will shrink to nothing and disappear! In fact, this ability to make an object disappear completely is the main advantage of this option. If you don’t need the object to disappear completely we recommend using the **Expandable** option.

Expand/Shrink objects can be stacked vertically in a column, but they should never be placed side by side. When expand / shrink objects are placed side by side they fight each other and the result is unpredictable. In fact, you must be careful when placing an **Expand/Shrink** object next to anything else. You may find that an object placed next to an **Expand/Shrink** object also disappears, which is probably not what you had intended.

Mixing Variable Height Objects With Other Graphics

When variable height objects are mixed with other objects, the variable height objects can force the other objects to move or change size. The illustration below shows this effect. On the left is a variable height data cell. The illustration shows how four other graphic objects are affected by the variable height object.



When a variable height object expands or shrinks, everything in the area below the object (shown in gray) is affected. How the other objects are affected depends on their location relative to the gray area.

Object A is completely above the gray area. It isn't affected by the variable height data cell.

Object B touches the gray area, so it expands and shrinks in step with the variable height object.

Object C straddles the gray area, so it also expands and shrinks. If it shrinks too much it will disappear completely! (Of course that could be exactly what you want.)

Object D is completely in the gray area, so it slides up and down as the variable height object expands and contracts. It does not change size, however.

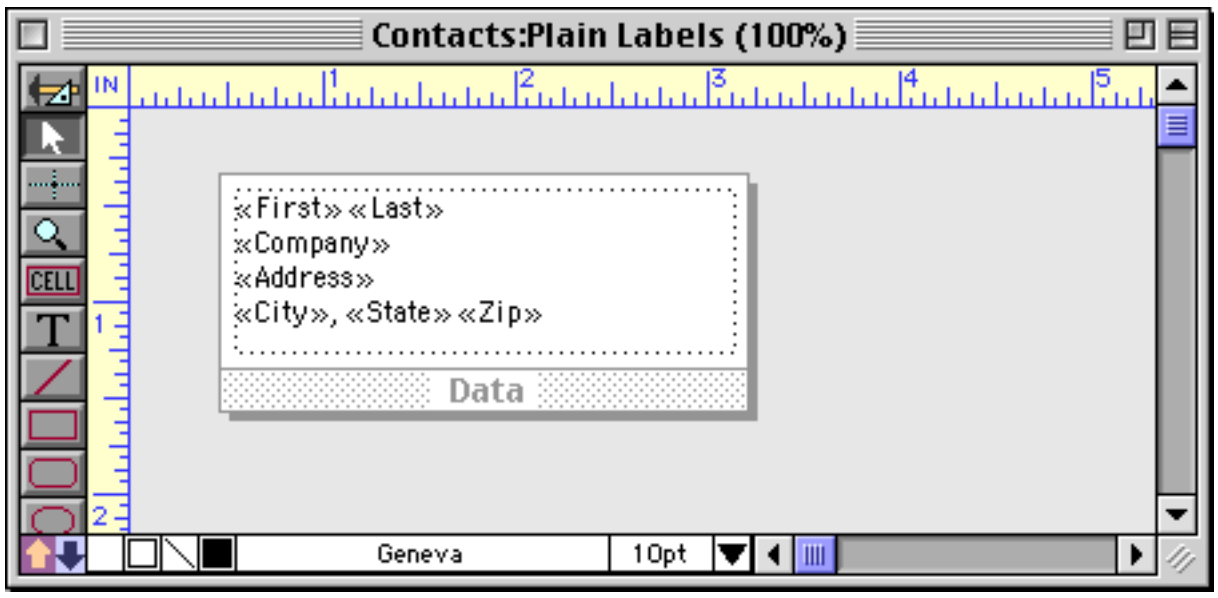
If there is more than one variable height object, the effect is cumulative. Panorama starts with the first variable height object (closest to the back) and works its way toward the front. For each variable height object the new height is calculated and all the other objects are adjusted.

These rules apply to other variable height objects as well as fixed height objects. If a second variable height object is partly or completely below another variable height object, it will be adjusted when the original variable height object expands or shrinks. You can use these rules to predict how unusual combinations of variable height objects will interact.

Printing Multiple Column Reports

If the data tile is less than half the width of the page Panorama can print a two column report. If the data tile is less than 1/3 of the page width Panorama can print a three column report. Multiple column reports can be used to print mailing labels, or to print multiple column catalogs, lists, and directories.

For example, the data tile on this form is just about 1/3 page.



When you print this form it will look like this.

John Smith
Acme Widgets
12 Harmony Lane
Suite 15

Don Harmon
Sudderth Video
415 Sudderth
Ruidoso, NM 88345

David Blair
DB Printing
869 W. Temple
Lenox, IA 50851

Susan Brown
783 Algonquin
Newport Beach, CA 93459

Abe Fierstein
Van Nuys Lumber
1571 Haskell
Van Nuys, CA 91409

Keith Baker
Northgate Video
552 Northgate
Lindenhurst, IL 60046

Karen Wilson
Evanston Lumber
498 Noyes
Evanston, IL 60201

Randy Cross
Randy's Appliances
133 Hunt Rd
Chelsford, MA 01824

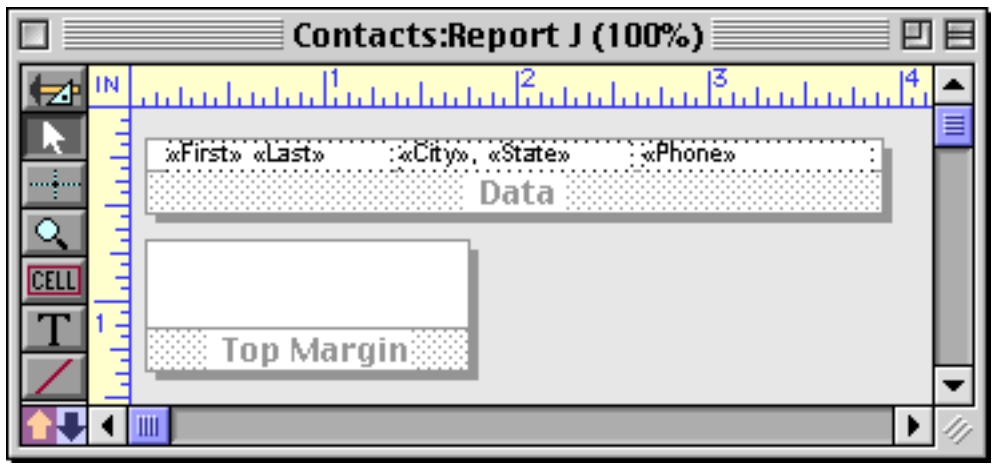
John Sloan
79 Danube Way
Olympia Fields, IL 60461

Jim Nickle
Jim's Appliances
14189 8th
Newhall, CA 91321

Jeffrey Rodman
2 Cary Rd
Chestnut Hill, MA 02167

Guy Porter
St. Louis Lumber
8702 Pershing
St. Louis, MO 63107

Here's another report where the Data tile is just less than half of the page width.



Panorama will print a two-up report, like this.

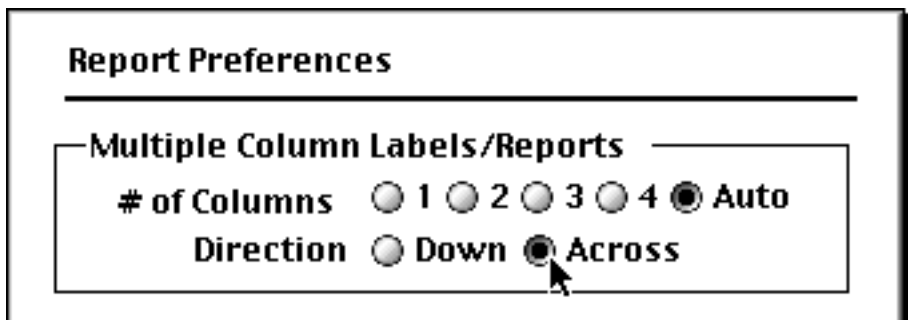
Keith Baker	Lindenhurst, IL		Jeannette Mulder	Irving, TX	
Nabil Basir	Armonk, NY		David Murray	Westport, CT	
John Bath	Mendota Heights,	(612) 451-1121	Jim Nickle	Newhall, CA	(805) 259-4093
Jack Beardsley	Toledo, OH		Logan Nourse	Palo Alto, CA	
Carl Berg	New Haven, CT	(203) 624-3367	Sam Pack	Inverness, IL	
Leslie Bianchi	Lexington, MA		Michael Paine	Pullman, WA	
Mary Bilbury	Beverly Hills, CA		David Peters	Concord, CA	
Joseph Bizzarri	Westchester, IL		Charles Pierce	Midland, TX	
David Blair	Lenox, IA		Guy Porter	St. Louis, MO	
Al Bodner	Clifton Park, NY		Jim Pyle	Roseville, CA	
Jerry Boone	Traverse City, MI		Sari Rattner	Seattle, WA	
Jerry Bowen	Highland, CA		Bud Roble	Riverside, CA	
Yvonne Broach	Houston, TX		Jeffrey Rodman	Chestnut Hill, MA	
Susan Brown	Newport Beach, CA		Chuck Rouse	Hays, KS	
Tom Cane	Dublin, CA	(415) 833-8577	Janel Rundlett	Kansas City, KS	
David Cohn	Buffalo Grove, IL		Ed Ruth	Chicago, IL	
Michael Cox	Dallas, TX		Peter Schug	Bronx, NY	
Anne Crane	Grosse Pointe		Jules Silk	Cheltenham, PA	
Randy Cross	Chelsford, MA		Peter Silvers	New Orleans, LA	
Thomas Cupal	Ann Arbor, MI		John Sloan	Olympia Fields, IL	
Charles Dalbert	New York, NY		John Smith	Huntington Beach,	(999) 555-1234
Steve Dallas	Creve Coeur, MO	(314) 993-4251	Brian Smith	Hollister, CA	

Across or Down?

When printing a multiple column report Panorama normally prints the entire first column, then goes back to the top of the second column, prints the entire column, goes to the top of the third column and prints it and on and on for all of the columns in the report.

Keith Baker	Lindenhurst, IL		Jeannette Mulder	Irving, TX	
Nabil Basir	Armonk, NY		David Murray	Westport, CT	
John Bath	Mendota Heights,	(612) 451-1121	Jim Nickle	Newhall, CA	(805) 259-4093
Jack Beardsley	Toledo, OH		Logan Nourse	Palo Alto, CA	
Carl Berg	New Haven, CT	(203) 624-3367	Sam Pack	Inverness, IL	
Leslie Bianchi	Lexington, MA		Michael Paine	Pullman, WA	
Mary Bilbury	Beverly Hills, CA		David Peters	Concord, CA	
Joseph Bizzarri	Westchester, IL		Charles Pierce	Midland, TX	
David Blair	Lenox, IA		Guy Porter	St. Louis, MO	
Al Bodner	Clifton Park, NY		Jim Pyle	Roseville, CA	
Jerry Boone	Traverse City, MI		Sari Rattner	Seattle, WA	
Jerry Bowen	Highland, CA		Bud Roble	Riverside, CA	
Yvonne Broach	Houston, TX		Jeffrey Rodman	Chestnut Hill, MA	
Susan Brown	Newport Beach, CA		Chuck Rouse	Hays, KS	
Tom Cane	Dublin, CA	(415) 833-8577	Janel Rundlett	Kansas City, KS	
David Cohn	Buffalo Grove, IL		Ed Ruth	Chicago, IL	
Michael Cox	Dallas, TX		Peter Schug	Bronx, NY	
Anne Crane	Grosse Pointe		Jules Silk	Cheltenham, PA	
Randy Cross	Chelsford, MA		Peter Silvers	New Orleans, LA	
Thomas Cupal	Ann Arbor, MI		John Sloan	Olympia Fields, IL	
Charles Dalbert	New York, NY		John Smith	Huntington Beach,	(999) 555-1234
Steve Dallas	Creve Coeur, MO	(314) 993-4251	Brian Smith	Hollister, CA	

If you want Panorama to print the data row by row instead of column by column, open the **Report Preferences** dialog (in the Setup menu) and switch to Across instead of Down.



Here is the exact same report but with the Across option enabled.

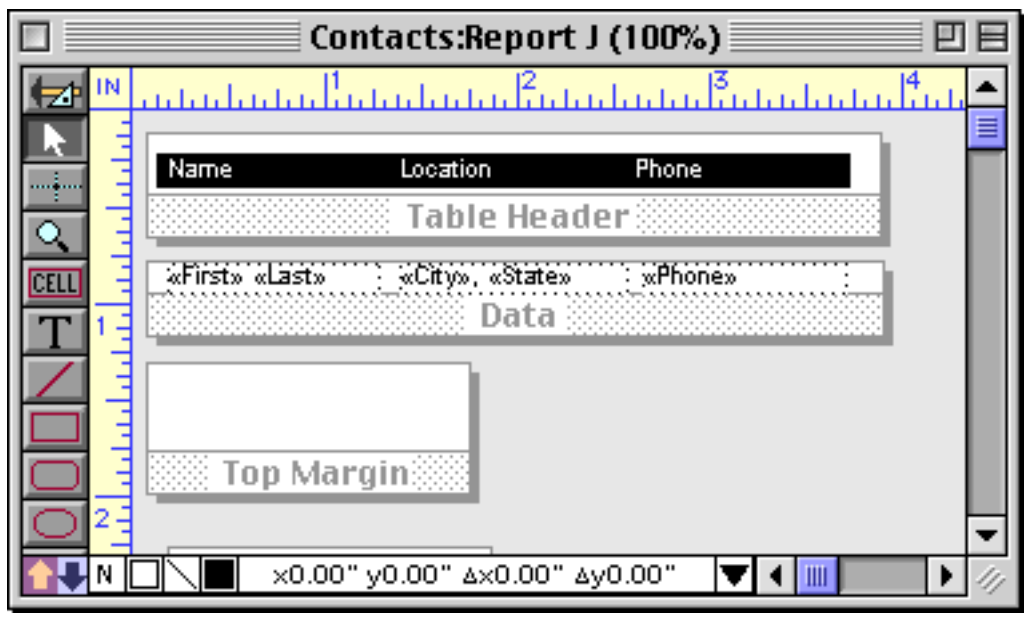
Keith Baker	Lindenhurst, IL		Nabil Basir	Armonk, NY	
John Bath	Mendota Heights,	(612) 451-1121	Jack Beardsley	Toledo, OH	
Carl Berg	New Haven, CT	(203) 624-3367	Leslie Bianchi	Lexington, MA	
Mary Bilbury	Beverly Hills, CA		Joseph Bizzarri	Westchester, IL	
David Blair	Lenox, IA		Al Bodner	Clifton Park, NY	
Jerry Boone	Traverse City, MI		Jerry Bowen	Highland, CA	
Yvonne Broach	Houston, TX		Susan Brown	Newport Beach, CA	
Tom Cane	Dublin, CA	(415) 833-8577	David Cohn	Buffalo Grove, IL	
Michael Cox	Dallas, TX		Anne Crane	Grosse Pointe	
Randy Cross	Chelsford, MA		Thomas Cupal	Ann Arbor, MI	
Charles Dalbert	New York, NY		Steve Dallas	Creve Coeur, MO	(314) 993-4251

The Across option is often the way you'll want to print mailing labels, and is automatically enabled when you create labels with the **QuickLabel** dialog (see "The QuickLabel Dialog" on page 1157).

Table Header and Table Footer Tiles

The Table Header and Table Footer tiles allow you to print a header and/or footer at the top and bottom of each column in a multiple column report. Unlike the regular header and footer tiles (see "Header Tile" on page 1079 and "Footer Tile" on page 1087) which are only printed once per page, the Table Header and Table Footer tiles are printed once for each column. You can combine the Table Header/Footer tiles with regular Header/Footer tiles if you wish, using the regular Header/Footer tiles for page numbers and overall page titles while using the Table Header/Footer tiles to create headers for each column.

Here is an example of a form with a table header.



When the form is printed, the table header appears at the top of each column.

Name	Location	Phone
Keith Baker	Lindenhurst, IL	
Nabil Basir	Armonk, NY	
John Bath	Mendota Heights,	(612) 451-1121
Jack Beardsley	Toledo, OH	
Carl Berg	New Haven, CT	(203) 624-3367
Leslie Bianchi	Lexington, MA	
Mary Bilbury	Beverly Hills, CA	
Joseph Bizzarri	Westchester, IL	
David Blair	Lenox, IA	
Al Bodner	Clifton Park, NY	
Jerry Boone	Traverse City, MI	
Jerry Bowen	Highland, CA	
Yvonne Broach	Houston, TX	
Susan Brown	Newport Beach, CA	
Tom Cane	Dublin, CA	(415) 833-8577
David Cohn	Buffalo Grove, IL	
Michael Cox	Dallas, TX	
Anne Crane	Grosse Pointe	
Randy Cross	Chelsford, MA	
Thomas Cupal	Ann Arbor, MI	

Name	Location	Phone
Mike Kuenning	Malvern, PA	
Scott Lay	Portland, OR	
Wes Lemarr	Rutherford, NJ	
Jerry Levan	Richmond, KY	
Tom Long	Austin, TX	(512) 474-1587
Tom Love	Woodbury, CT	
Nadine Lucas	San Francisco, CA	
John Maguire	Akron, OH	
James Mahan	Los Angeles, CA	
John Marshall	Jenison, MI	
Don Meadows	Austin, TX	
Charles Michaels	Upland, CA	
Steve Miller	Jupiter, FL	
Tim Moran	Wheaton, IL	
John Moses	San Clemente, CA	
Jeannette Mulder	Irving, TX	
David Murray	Westport, CT	
Jim Nickle	Newhall, CA	(805) 259-4093
Logan Nourse	Palo Alto, CA	
Sam Pack	Inverness, IL	

This illustration shows how Panorama assembles the Table Header and Data tiles into two columns.

Name	Location	Phone
Keith Baker	Lindenhurst, IL	
Nabil Basir	Armonk, NY	
John Bath	Mendota Heights,	(612) 451-1121
Jack Beardsley	Toledo, OH	
Carl Berg	New Haven, CT	(203) 624-3367
Leslie Bianchi	Lexington, MA	
Mary Bilbury	Beverly Hills, CA	
Joseph Bizzarri	Westchester, IL	
David Blair	Lenox, IA	
Al Bodner	Clifton Park, NY	
Jerry Boone	Traverse City, MI	
Jerry Bowen	Highland, CA	
Yvonne Broach	Houston, TX	
Susan Brown	Newport Beach, CA	
Tom Cane	Dublin, CA	(415) 833-8577
David Cohn	Buffalo Grove, IL	
Michael Cox	Dallas, TX	

Table Header

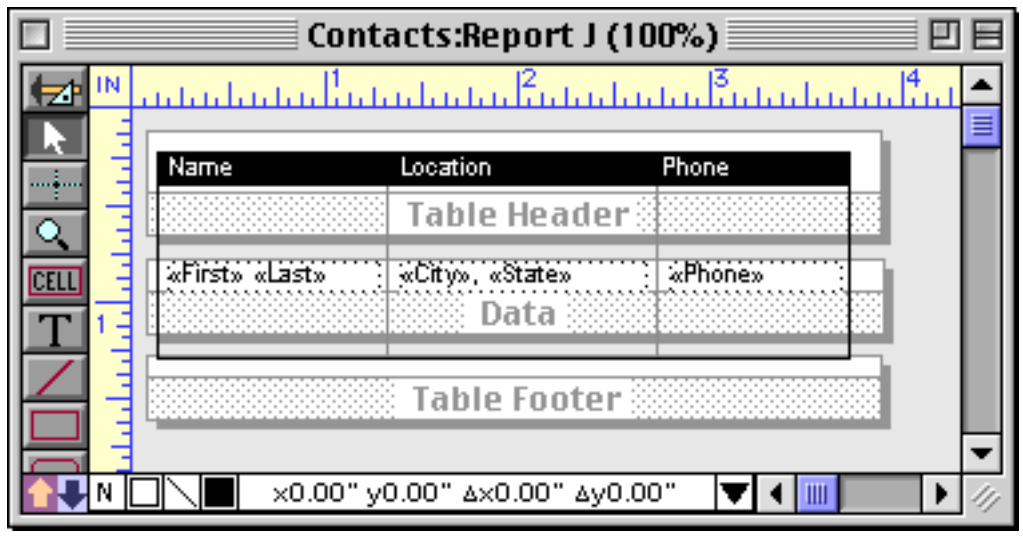
Data

Name	Location	Phone
Mike Kuenning	Malvern, PA	
Scott Lay	Portland, OR	
Wes Lemarr	Rutherford, NJ	
Jerry Levan	Richmond, KY	
Tom Long	Austin, TX	(512) 474-1587
Tom Love	Woodbury, CT	
Nadine Lucas	San Francisco, CA	
John Maguire	Akron, OH	
James Mahan	Los Angeles, CA	
John Marshall	Jenison, MI	
Don Meadows	Austin, TX	
Charles Michaels	Upland, CA	
Steve Miller	Jupiter, FL	
Tim Moran	Wheaton, IL	
John Moses	San Clemente, CA	
Jeannette Mulder	Irving, TX	
David Murray	Westport, CT	

Table Header

Data

Here's a modified version of the same report that includes both Table Header and Table Footer tiles.

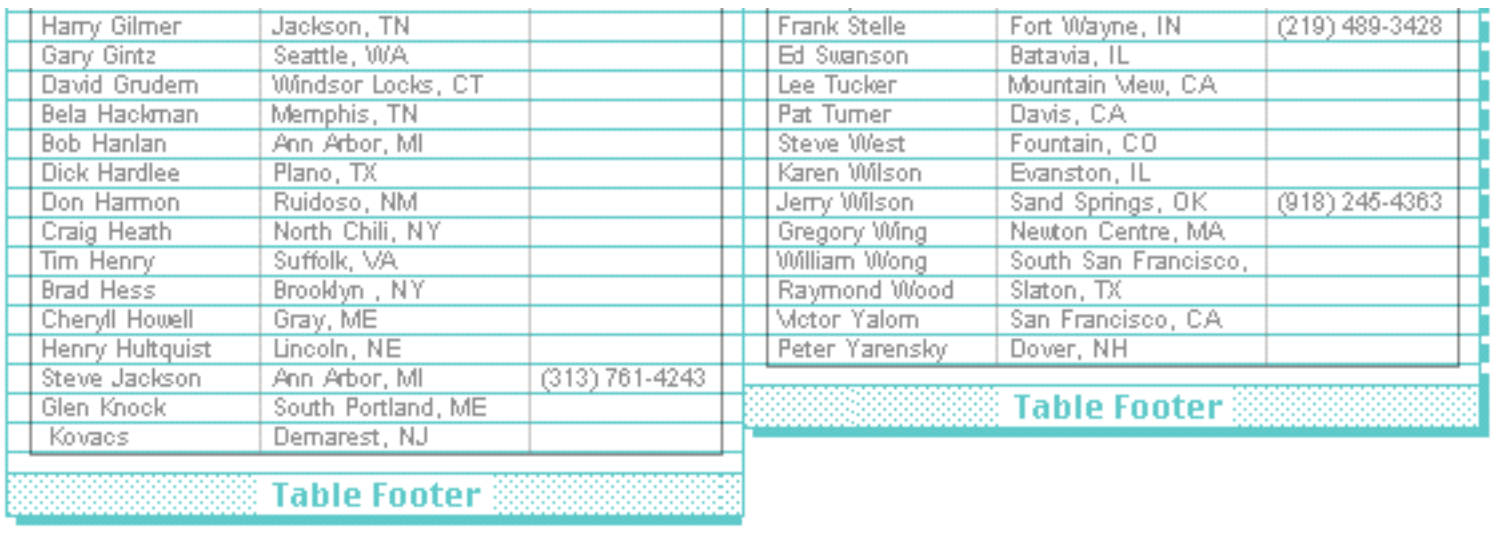


This form uses the Table Footer tile to print a bottom border below each column.

Name	Location	Phone
Keith Baker	Lindenhurst, IL	
Nabil Basir	Armonk, NY	
John Bath	Mendota Heights, MN	(612) 451-1121
Jack Beardsley	Toledo, OH	
Carl Berg	New Haven, CT	(203) 624-3367
Leslie Bianchi	Lexington, MA	
Mary Bilbury	Beverly Hills, CA	
Joseph Bizzarri	Westchester, IL	
David Blair	Lenox, IA	
Al Bodner	Clifton Park, NY	
Jerry Boone	Traverse City, MI	
Jerry Bowen	Highland, CA	
Yvonne Broach	Houston, TX	
Susan Brown	Newport Beach, CA	
Tom Cane	Dublin, CA	(415) 833-8577
David Cohn	Buffalo Grove, IL	
Michael Cox	Dallas, TX	
Anne Crane	Grosse Pointe	
Randy Cross	Chelsford, MA	
Thomas Cupal	Ann Arbor, MI	
Charles Dalbert	New York, NY	
Steve Dallas	Creve Coeur, MO	(314) 993-4251
Herb Dang	San Francisco, CA	
Tim Daniels	St. Louis, MO	
Stephen Dempsey	Demarest, NJ	
Mark Dockum	Camarillo, CA	
Joseph Doll	Shaker Heights, OH	(216) 751-1432
Patrick Dowd	Convent Station, NJ	
Mary Doyle	Redwood City, CA	
John Draper	Hampton, NH	
Joel Dye	Carrollton, TX	
John Fabian	Woodstock, VT	
Brian Felty	Lubbock, TX	
Abe Fierstein	Van Nuys, CA	
Ramsey French	West Palm Beach, FL	
Jeffrey Funk	Flemington, NJ	
Thom Getchell	Menlo Park, CA	(415) 326-3887
Steve Gibson	St. Peters, MO	
Harry Gilmer	Jackson, TN	
Gary Gintz	Seattle, WA	
David Grudem	Windsor Locks, CT	
Bela Hackman	Memphis, TN	
Bob Hanlan	Ann Arbor, MI	
Dick Hardlee	Plano, TX	
Don Harmon	Ruidoso, NM	
Craig Heath	North Chili, NY	
Tim Henry	Suffolk, VA	
Brad Hess	Brooklyn , NY	
Cheryll Howell	Gray, ME	
Henry Hultquist	Lincoln, NE	
Steve Jackson	Ann Arbor, MI	(313) 761-4243
Glen Knock	South Portland, ME	
Kovacs	Demarest, NJ	

Name	Location	Phone
Mike Kuenning	Malvern, PA	
Scott Lay	Portland, OR	
Wes Lemarr	Rutherford, NJ	
Jerry Levan	Richmond, KY	
Tom Long	Austin, TX	(512) 474-1587
Tom Love	Woodbury, CT	
Nadine Lucas	San Francisco, CA	
John Maguire	Akron, OH	
James Mahan	Los Angeles, CA	
John Marshall	Jenison, MI	
Don Meadows	Austin, TX	
Charles Michaels	Upland, CA	
Steve Miller	Jupiter, FL	
Tim Moran	Wheaton, IL	
John Moses	San Clemente, CA	
Jeannette Mulder	Irving, TX	
David Murray	Westport, CT	
Jim Nickle	Newhall, CA	(805) 259-4093
Logan Nourse	Palo Alto, CA	
Sam Pack	Inverness, IL	
Michael Paine	Pullman, WA	
David Peters	Concord, CA	
Charles Pierce	Midland, TX	
Guy Porter	St. Louis, MO	
Jim Pyle	Roseville, CA	
Sari Rattner	Seattle, WA	
Bud Roble	Riverside, CA	
Jeffrey Rodman	Chestnut Hill, MA	
Chuck Rouse	Hays, KS	
Janel Rundlett	Kansas City, KS	
Ed Ruth	Chicago, IL	
Peter Schug	Bronx, NY	
Jules Silk	Cheltenham, PA	
Peter Silvers	New Orleans, LA	
John Sloan	Olympia Fields, IL	
John Smith	Huntington Beach,	(999) 555-1234
Brian Smith	Hollister, CA	
Alan Spencer	Northbrook, IL	
Frank Stelle	Fort Wayne, IN	(219) 489-3428
Ed Swanson	Batavia, IL	
Lee Tucker	Mountain View, CA	
Pat Turner	Davis, CA	
Steve West	Fountain, CO	
Karen Wilson	Evanston, IL	
Jerry Wilson	Sand Springs, OK	(918) 245-4363
Gregory Wing	Newton Centre, MA	
William Wong	South San Francisco,	
Raymond Wood	Slaton, TX	
Victor Yalom	San Francisco, CA	
Peter Yarensky	Dover, NH	

Notice that unlike the regular Footer tile (see "Footer Tile" on page 1087) there is no gap between the last data tile and the column footer tile. (In this example the second column is shorter than the first because there weren't enough data records to fill the entire page. The height of each column will always be the same except for the last column on the last page.)



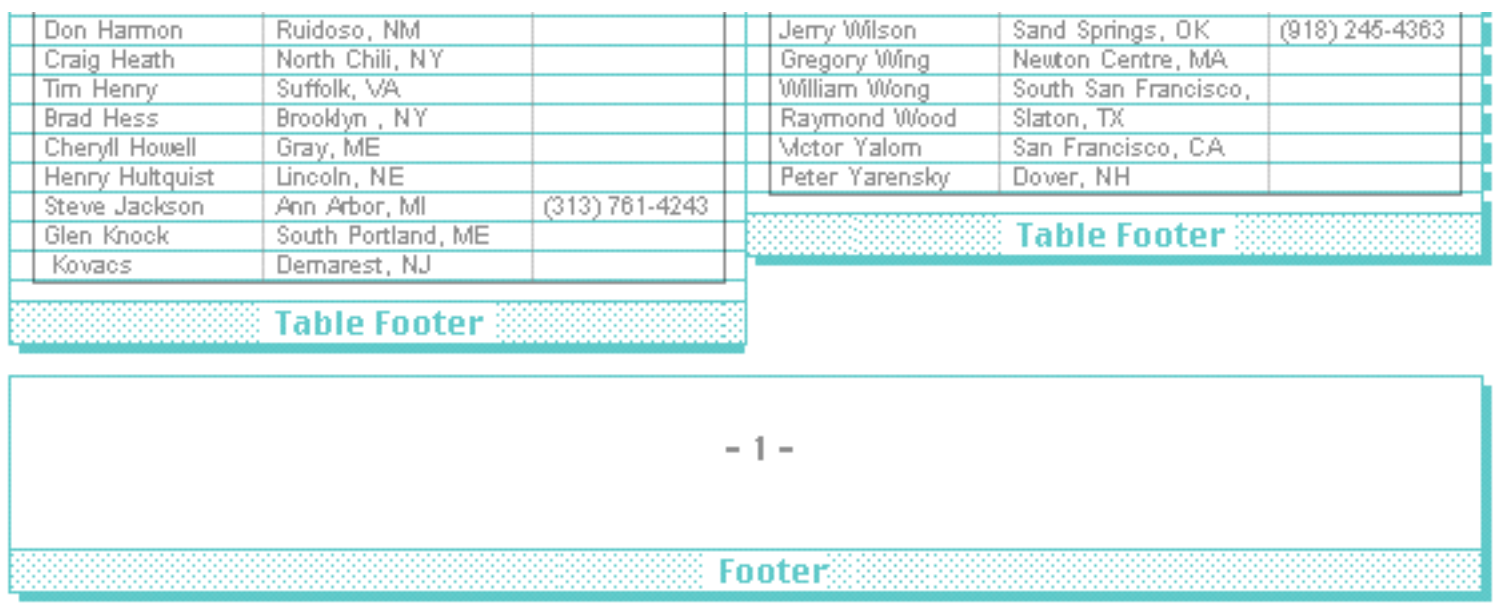
Harry Gilmer	Jackson, TN	
Gary Gintz	Seattle, WA	
David Grudem	Windsor Locks, CT	
Bela Hackman	Memphis, TN	
Bob Hanlan	Ann Arbor, MI	
Dick Hardlee	Plano, TX	
Don Harmon	Ruidoso, NM	
Craig Heath	North Chili, NY	
Tim Henry	Suffolk, VA	
Brad Hess	Brooklyn , NY	
Cheryll Howell	Gray, ME	
Henry Hultquist	Lincoln, NE	
Steve Jackson	Ann Arbor, MI	(313) 761-4243
Glen Knock	South Portland, ME	
Kovacs	Demarest, NJ	

Table Footer

Frank Stelle	Fort Wayne, IN	(219) 489-3428
Ed Swanson	Batavia, IL	
Lee Tucker	Mountain View, CA	
Pat Turner	Davis, CA	
Steve West	Fountain, CO	
Karen Wilson	Evanston, IL	
Jerry Wilson	Sand Springs, OK	(918) 245-4363
Gregory Wing	Newton Centre, MA	
William Wong	South San Francisco,	
Raymond Wood	Slaton, TX	
Victor Yalom	San Francisco, CA	
Peter Yarensky	Dover, NH	

Table Footer

However, if the form does contain a regular Footer tile there may be a gap between the bottom of the Table Footer tile and the top of the Footer tile. Here's an example showing a report with both a Table Footer and a regular Footer tile. The regular Footer tile is being used to print the page number (see "Page Numbers" on page 1088).



Don Harmon	Ruidoso, NM	
Craig Heath	North Chili, NY	
Tim Henry	Suffolk, VA	
Brad Hess	Brooklyn , NY	
Cheryll Howell	Gray, ME	
Henry Hultquist	Lincoln, NE	
Steve Jackson	Ann Arbor, MI	(313) 761-4243
Glen Knock	South Portland, ME	
Kovacs	Demarest, NJ	

Table Footer

Jerry Wilson	Sand Springs, OK	(918) 245-4363
Gregory Wing	Newton Centre, MA	
William Wong	South San Francisco,	
Raymond Wood	Slaton, TX	
Victor Yalom	San Francisco, CA	
Peter Yarensky	Dover, NH	

Table Footer

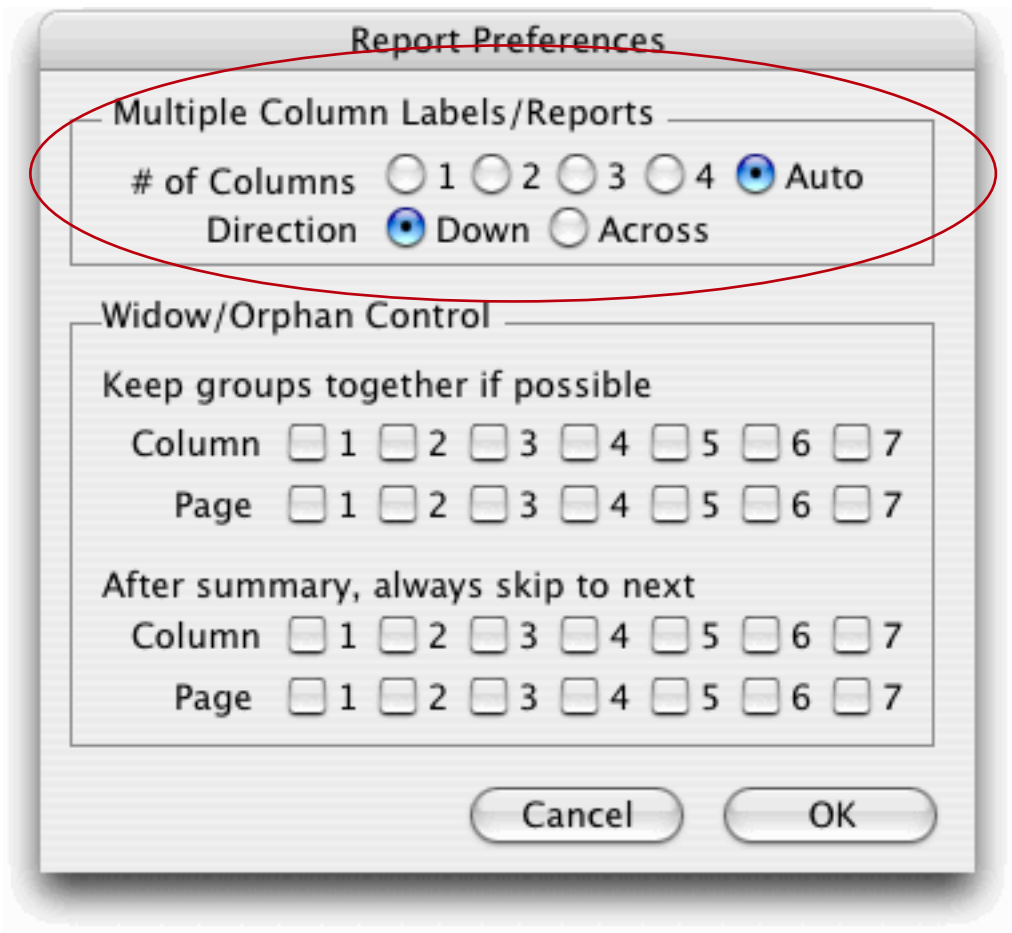
- 1 -

Footer

Usually table header and footer tiles are combined with the normal header and footer tiles for printing the report title, date, and page numbers.

Controlling the Number of Columns

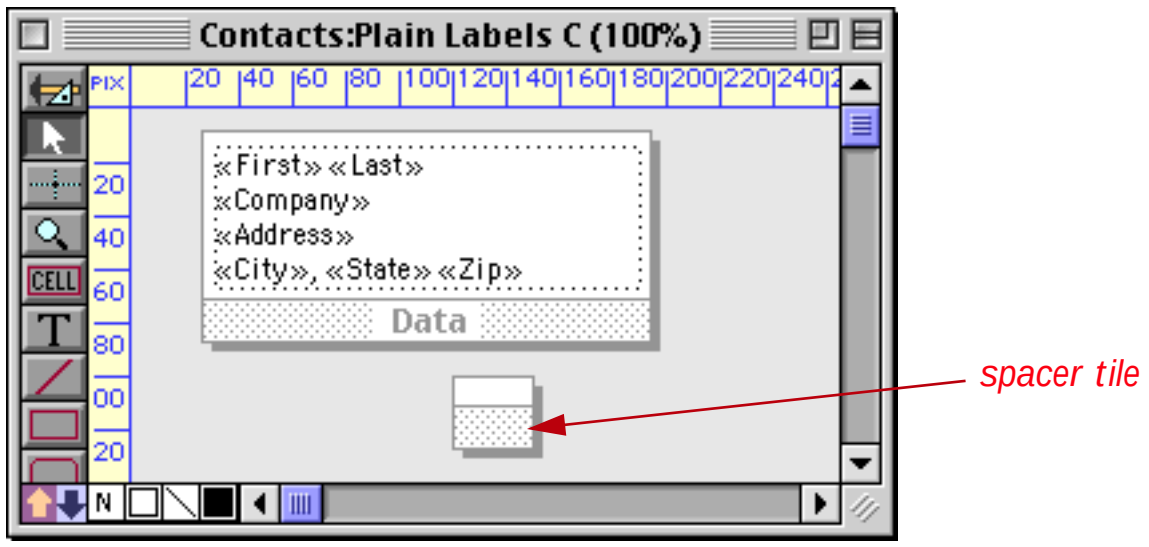
Panorama normally packs in as many columns as it can based on the width of the data tile. However, you can override the number of columns with the **Report Preferences** dialog in the Setup menu.



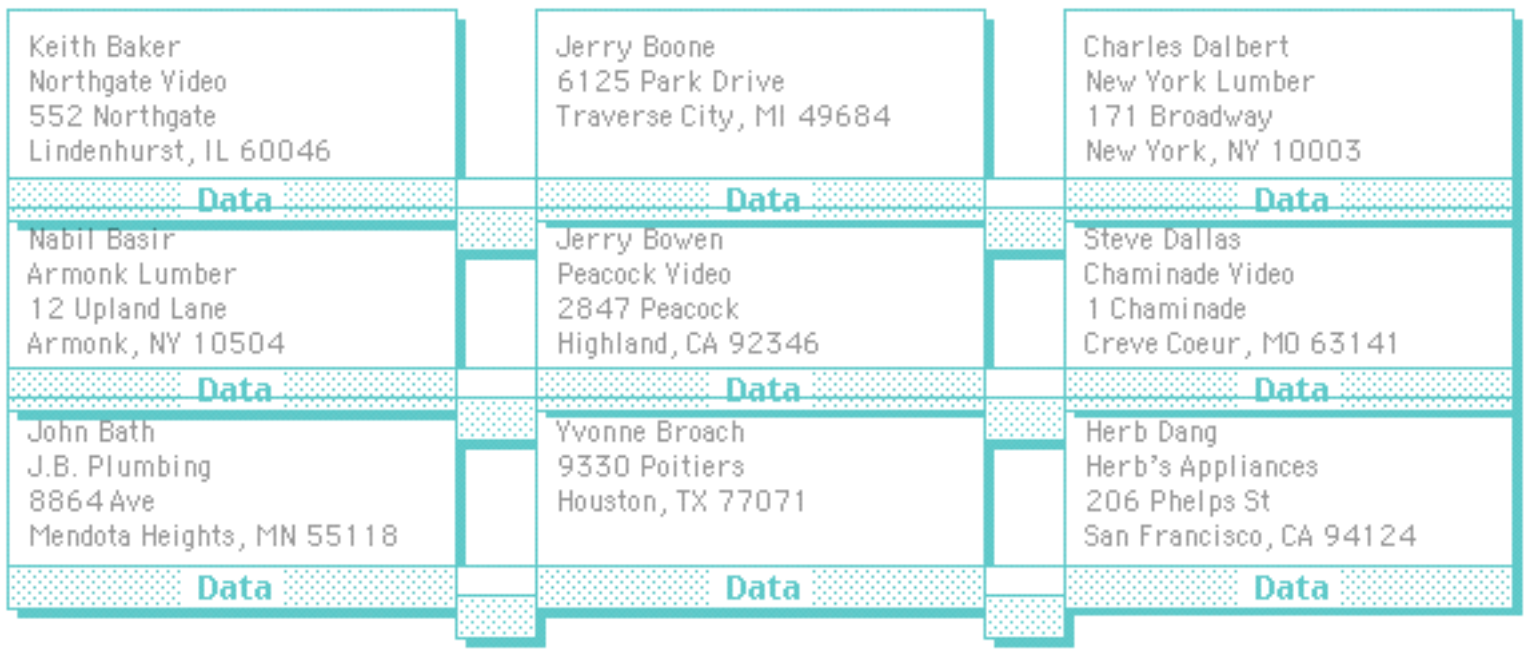
Picking 1, 2, 3, or 4 forces the number of columns. Panorama will use the number of columns you specify even if the columns won't fit on the page. This is usually useful when the number of columns you want to fit almost fits on the page, but not quite. There's no point in printing four columns when only three will fit — the extra column will simply be invisible. Pick Auto to let Panorama choose the number of columns based on the width of the data tile.

Spacer Tile

The spacer tile is rarely used. Panorama normally packs the data tiles as close to each other as possible. The spacer tile allows you to spread the data tiles apart. When printing a multiple column report, Panorama will spread the columns apart by the width of the spacer tile. Vertically, each data tile will be spread apart by the height of the spacer. Horizontally, each data tile will be spread apart by the width of the spacer. Here's an example of a form with a spacer tile.



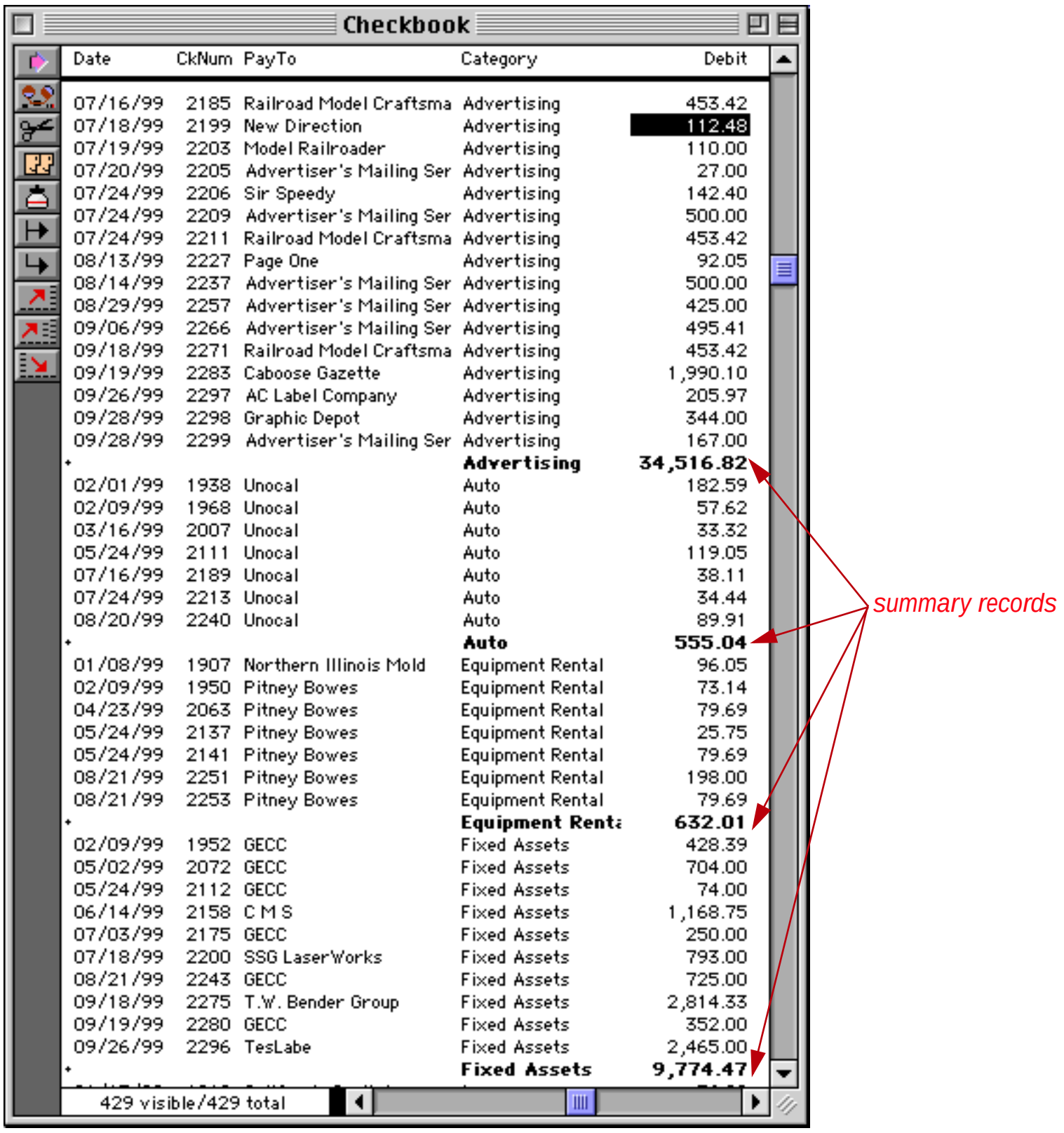
When the form is printed, Panorama inserts the Spacer tile in between the Data tiles. Notice, however, that there are no spacer tiles to the right of the last column. This slightly reduces the overall width of the report, possibly allowing you to sqeeze in that extra column.



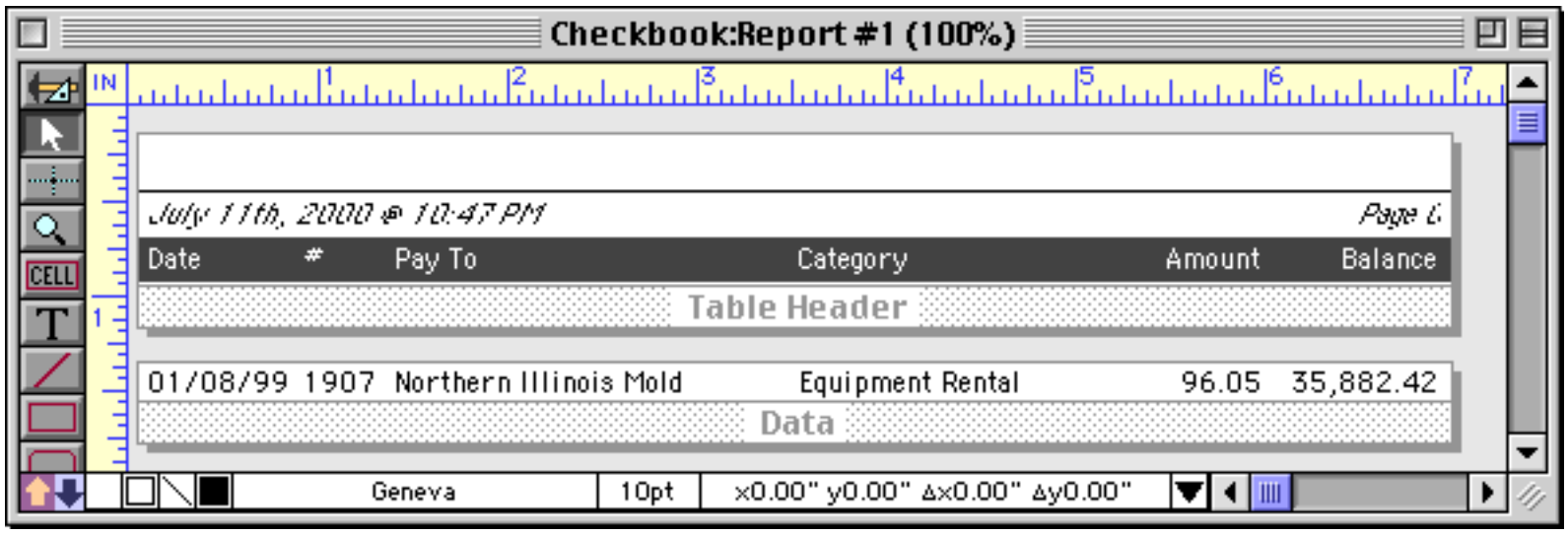
For most reports, this same spreading effect can be obtained by increasing the size of the data tile. If you are using automatic multiple columns however, increasing the size of the data tile doesn't always work, because it may force the last column into the right margin. If that happens, Panorama won't print the final column. The spacer tile allows you to spread the columns apart while still printing the maximum number of columns. Remember, the only time you may need the spacer tile is when you are printing reports with automatic multiple columns. You can also force the number of columns with the **Report Preferences** dialog (see previous section).

Printing Summary Information

In the chapter "Summaries and Outlines" on page 393 you learned how to take a database and group it into categories and then summarize those categories. The summary information (totals, averages, etc.) appears in temporary new summary records that temporarily become part of the database, like this.



In an ordinary report these summary records are printed just like any other record – in other words, they are printed with the data tile. Here's an example of a typical form with a data tile.



Here's a page from the printout, with the summary records circled in red. As you can see, the summary records are formatted exactly the same way as the regular data records.

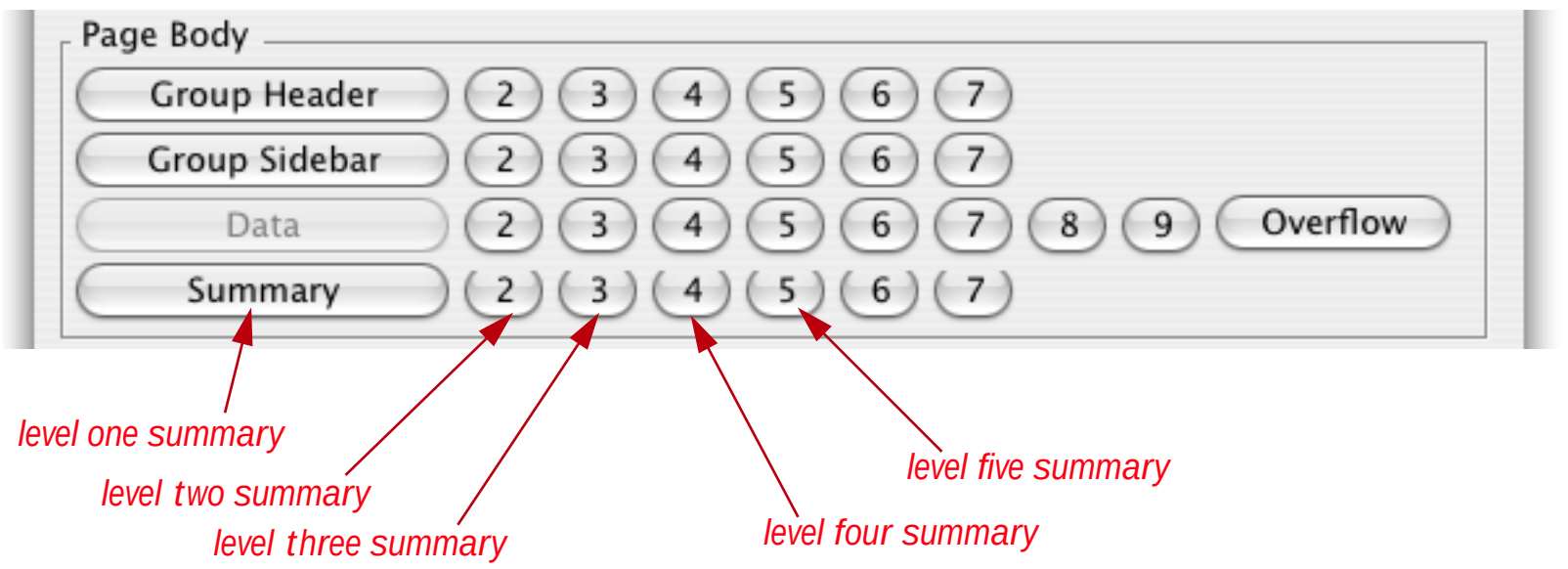
July 11th, 2000 @ 10:47 PM Page 4

Date	#	Pay To	Category	Amount	Balance
09/19/99	2285	Sherman Douglas Ins	Insurance	~~161.00~~	~~61,567.10~~
			Insurance	8,234.53	0.00
01/16/99	1910	Coudert Brothers, Attorney's	Legal Fees	~~223.52~~	~~35,193.10~~
02/09/99	1948	California Secretary Of State	Legal Fees	5.00	21,817.37
02/09/99	1949	City Of Caboose	Legal Fees	90.00	21,727.37
05/24/99	2121	Coudert Brothers/Attorney's	Legal Fees	799.55	49,799.79
06/05/99	2155	State Of California	Legal Fees	5.00	51,530.16
06/14/99	2159	Commonwealth Of	Legal Fees	10.00	58,056.21
07/16/99	2194	XL Corporation	Legal Fees	~~155.00~~	~~49,684.63~~
			Legal Fees	1,288.07	0.00
02/09/99	1964	Copierland	Maintenance	~~310.00~~	~~17,113.53~~
03/01/99	1986	Boyer & Ambrose Carpet	Maintenance	87.50	14,196.18
03/12/99	2002	Boyer & Ambrose Carpet	Maintenance	156.35	8,988.30
03/29/99	2037	Priority One Computers	Maintenance	496.40	21,349.77
03/29/99	2046	Executive Surveillance	Maintenance	132.00	19,370.06
04/24/99	2067	Boyer & Ambrose Carpet	Maintenance	132.00	31,336.38
05/08/99	2090	ServiceWorld	Maintenance	265.63	40,460.26
05/24/99	2114	E T S	Maintenance	49.00	51,641.24
05/24/99	2116	Sun Computers	Maintenance	282.00	51,091.39
05/24/99	2139	Pitney Bowes	Maintenance	140.00	43,941.69
05/29/99	2149	Sun Computers	Maintenance	276.00	52,038.36
06/05/99	2157	Sun Computers	Maintenance	101.25	51,408.96
06/21/99	2167	Dial One	Maintenance	267.13	56,654.66
07/16/99	2190	Executive Surveillance	Maintenance	168.00	50,229.71
07/24/99	2214	J & M Fire Extinguisher	Maintenance	38.19	41,952.07
08/08/99	2220	Newport Buidling &	Maintenance	120.00	49,321.22
08/21/99	2252	Vint Pest Control	Maintenance	120.00	54,797.48
09/18/99	2270	Computek Computer	Maintenance	100.00	69,777.93
09/18/99	2274	Boyer & Ambrose Carpet	Maintenance	~~432.54~~	~~68,392.35~~
			Maintenance	3,673.99	0.00
01/16/99	1911	Paramount Stationers	Office Supplies	~~105.84~~	~~35,087.26~~
01/22/99	1919	Cannon Astro	Office Supplies	145.72	24,693.23
01/25/99	1921	Nebs	Office Supplies	77.27	22,730.56
01/25/99	1922	Ramona Drinking Water	Office Supplies	98.10	22,632.46
02/01/99	1935	Copierland	Office Supplies	137.04	27,721.02

If you want the summary records to print differently from regular data records you'll need to add additional **summary report tiles** to your database (described in the next section). The use of these specialized tiles is strictly optional. If the report does not contain any summary tiles, Panorama will print summary records as if they were ordinary data records. For many reports this is just fine. If, however, you choose to use the specialized summary tiles, you can independently control how each summary level will be printed.

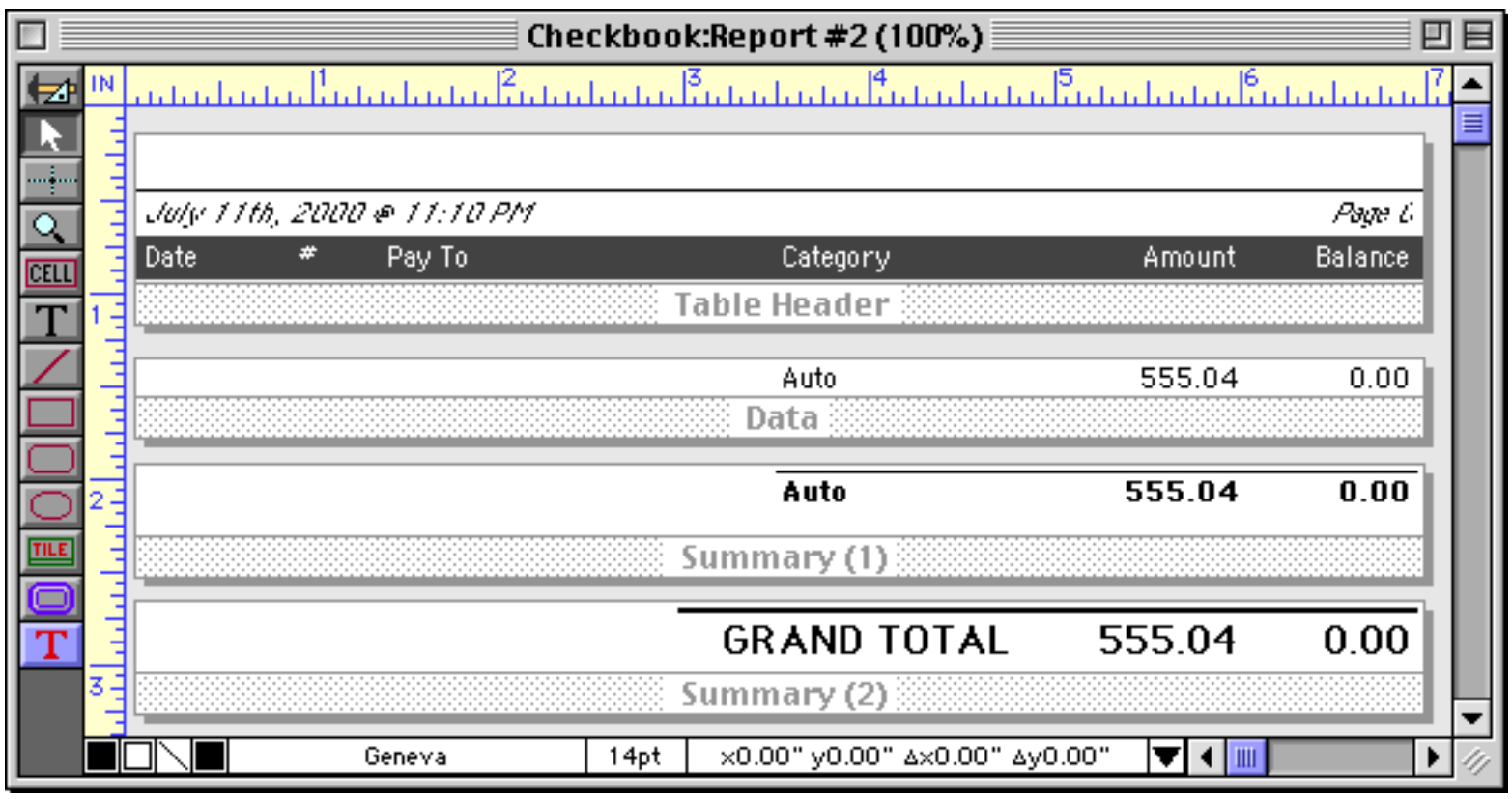
Summary Tiles

A summary tile works just like a data tile, except that the summary tile is used only when Panorama prints a summary record. Since a database may have up to 7 levels of summary records, a report may contain up to 7 summary tiles. For example, you could use summary tiles to print the data in 12 point Helvetica, subtotals (level 1) in 14 point and the grand total (level 2) in 18 point. Summary tiles are created with the **Specialized Tile** configuration dialog.



When Panorama is about to print a summary record, it checks to see if the report has a summary tile for that level. If it does, that summary tile will be placed in the report instead of the data tile. If there is no corresponding summary tile, Panorama will look for summary tiles corresponding to a lower summary level. If it finds such a tile, Panorama will use it to print the summary record. If Panorama doesn't find a lower level summary tile, it gives up and uses the data tile to print the summary record.

The easiest way to create summary tiles is dragging to make a copy of the data tile (see "Drag Duplicating" on page 561). Once the new tile is created, you can double click on the tile's name to open the configuration dialog and convert it from a data tile into the appropriate level summary tile. Here is an example of a report with two summary tiles - one for subtotals and one for the grand total.



Here is one page from the report that is printed by this form (assuming that the database has already been grouped by the Category field (see "STEP 1 - GROUP" on page 399) and totaled by the Amount field (see "STEP 2 - CALCULATE" on page 403).

July 11th, 2000 @ 10:55 PM *Page 4*

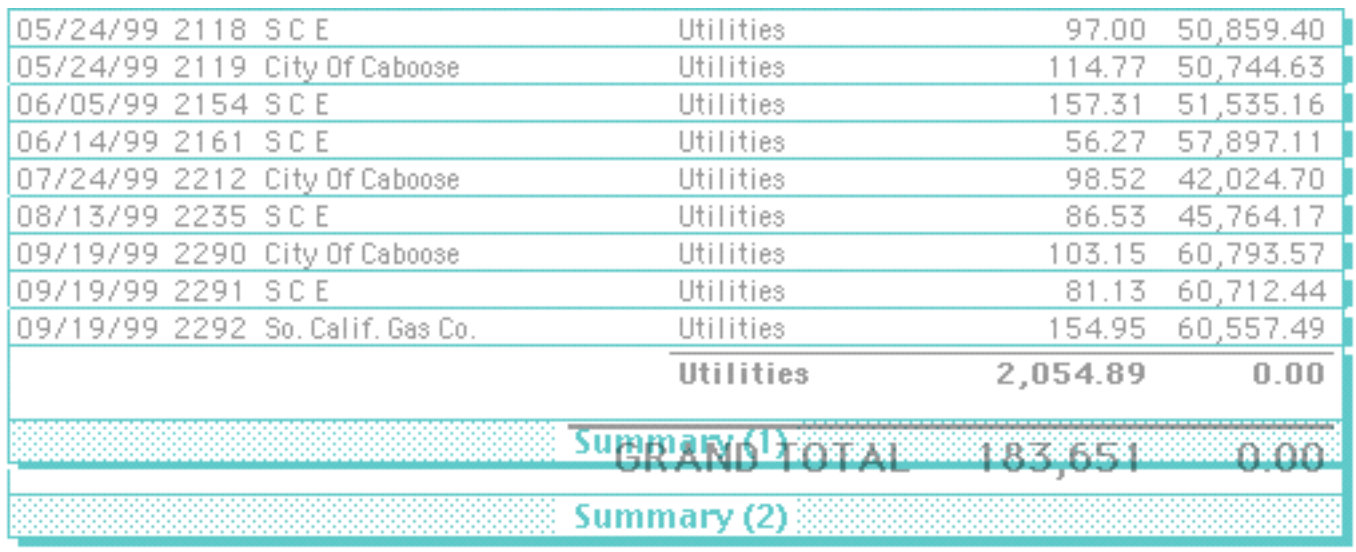
Date	#	Pay To	Category	Amount	Balance
08/21/99	2245	Maryland Casualty	Insurance	147.78	55,454.15
08/29/99	2260	Blue Cross Of Calif	Insurance	177.85	49,716.67
09/13/99	2269	Employers Health	Insurance	284.44	69,877.93
09/18/99	2272	Blue Cross Of Calif	Insurance	177.85	69,146.66
09/19/99	2285	Sherman Douglas Ins	Insurance	161.00	61,567.10
			Insurance	**8,234.53**	**0.00**
01/16/99	1910	Coudert Brothers, Attorney's	Legal Fees	223.52	35,193.10
02/09/99	1948	California Secretary Of State	Legal Fees	5.00	21,817.37
02/09/99	1949	City Of Caboose	Legal Fees	90.00	21,727.37
05/24/99	2121	Coudert Brothers/Attorney's	Legal Fees	799.55	49,799.79
06/05/99	2155	State Of California	Legal Fees	5.00	51,530.16
06/14/99	2159	Commonwealth Of	Legal Fees	10.00	58,056.21
07/16/99	2194	XL Corporation	Legal Fees	155.00	49,684.63
			Legal Fees	**1,288.07**	**0.00**
02/09/99	1964	Copierland	Maintenance	310.00	17,113.53
03/01/99	1986	Boyer & Ambrose Carpet	Maintenance	87.50	14,196.18
03/12/99	2002	Boyer & Ambrose Carpet	Maintenance	156.35	8,988.30
03/29/99	2037	Priority One Computers	Maintenance	496.40	21,349.77
03/29/99	2046	Executive Surveillance	Maintenance	132.00	19,370.06
04/24/99	2067	Boyer & Ambrose Carpet	Maintenance	132.00	31,336.38
05/08/99	2090	ServiceWorld	Maintenance	265.63	40,460.26
05/24/99	2114	E T S	Maintenance	49.00	51,641.24
05/24/99	2116	Sun Computers	Maintenance	282.00	51,091.39
05/24/99	2139	Pitney Bowes	Maintenance	140.00	43,941.69
05/29/99	2149	Sun Computers	Maintenance	276.00	52,038.36
06/05/99	2157	Sun Computers	Maintenance	101.25	51,408.96
06/21/99	2167	Dial One	Maintenance	267.13	56,654.66
07/16/99	2190	Executive Surveillance	Maintenance	168.00	50,229.71
07/24/99	2214	J & M Fire Extinguisher	Maintenance	38.19	41,952.07
08/08/99	2220	Newport Buidling &	Maintenance	120.00	49,321.22
08/21/99	2252	Vint Pest Control	Maintenance	120.00	54,797.48
09/18/99	2270	Computek Computer	Maintenance	100.00	69,777.93
09/18/99	2274	Boyer & Ambrose Carpet	Maintenance	432.54	68,392.35
			Maintenance	**3,673.99**	**0.00**
01/16/99	1911	Paramount Stationers	Office Supplies	105.84	35,087.26
01/22/99	1919	Cannon Astro	Office Supplies	145.72	24,693.23
01/25/99	1921	Nebs	Office Supplies	77.27	22,730.56
01/25/99	1922	Ramona Drinking Water	Office Supplies	98.10	22,632.46

The last page of this report contains both level 1 (subtotal) and level 2 (in this case, grand total) summaries.

July 11th, 2000 @ 11:08 PM *Page 16*

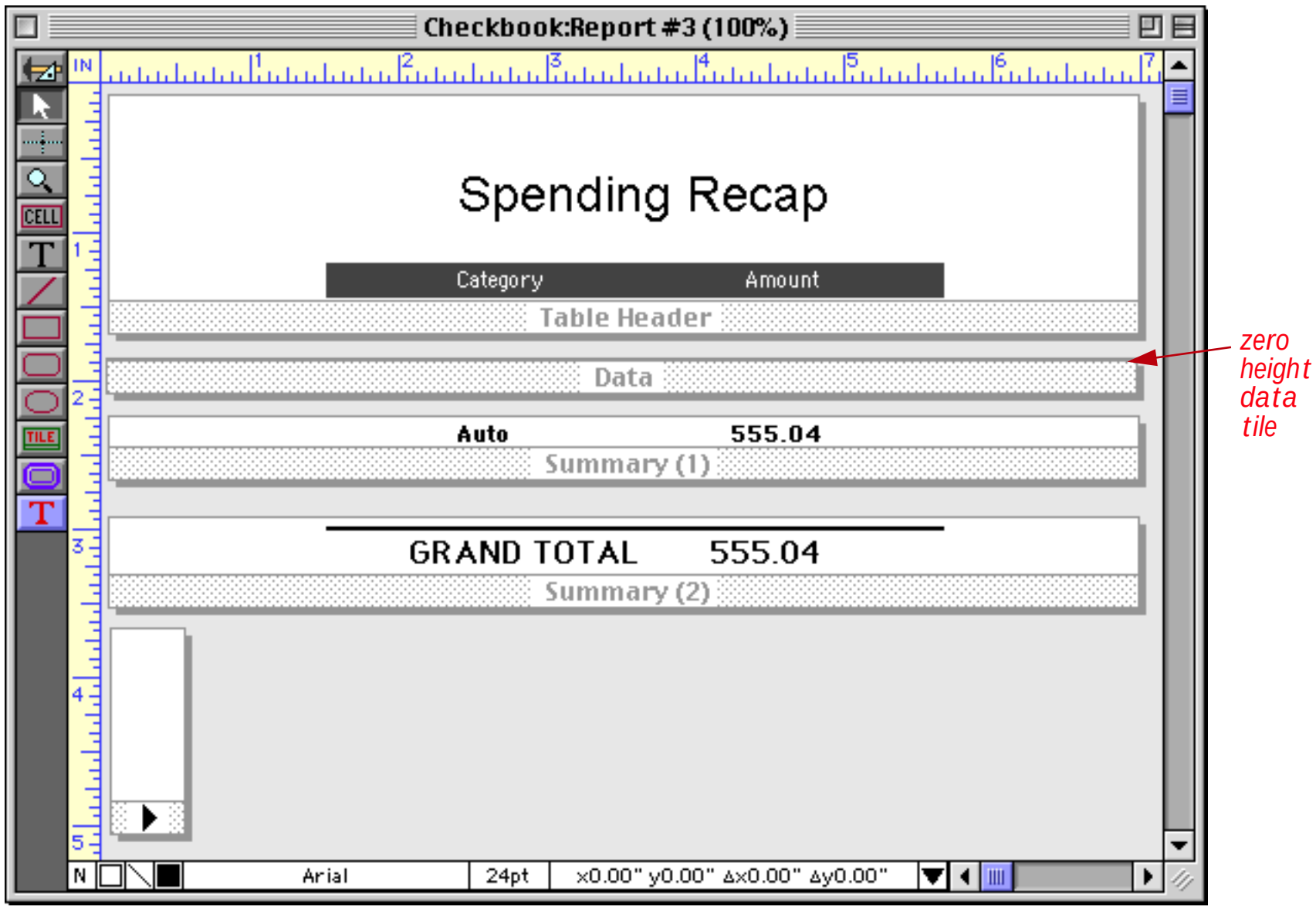
Date	#	Pay To	Category	Amount	Balance
02/09/99	1953	City Of Caboose	Utilities	9.64	21,151.17
02/09/99	1972	So. Calif. Gas Co.	Utilities	136.33	14,937.24
02/09/99	1973	S C E	Utilities	172.03	14,765.21
03/29/99	2041	City Of Caboose	Utilities	77.71	20,801.96
03/29/99	2042	So. Calif. Gas Co.	Utilities	217.32	20,584.64
03/29/99	2043	S C E	Utilities	89.46	20,495.18
05/07/99	2083	S C E	Utilities	96.26	41,775.05
05/07/99	2085	So. Calif. Gas Co.	Utilities	86.74	41,420.56
05/24/99	2117	So. Calif. Gas Co.	Utilities	134.99	50,956.40
05/24/99	2118	S C E	Utilities	97.00	50,859.40
05/24/99	2119	City Of Caboose	Utilities	114.77	50,744.63
06/05/99	2154	S C E	Utilities	157.31	51,535.16
06/14/99	2161	S C E	Utilities	56.27	57,897.11
07/24/99	2212	City Of Caboose	Utilities	98.52	42,024.70
08/13/99	2235	S C E	Utilities	86.53	45,764.17
09/19/99	2290	City Of Caboose	Utilities	103.15	60,793.57
09/19/99	2291	S C E	Utilities	81.13	60,712.44
09/19/99	2292	So. Calif. Gas Co.	Utilities	154.95	60,557.49
			Utilities	**2,054.89**	**0.00**
			GRAND TOTAL	183,651	0.00

This illustration shows how the report tiles at the end of this report are put together. Data records are printed with the Data tile, level 1 (subtotal) summary records are printed with the **Summary (1)** tile, and level 2 (grand total) summary records are printed with the **Summary (2)** tile.

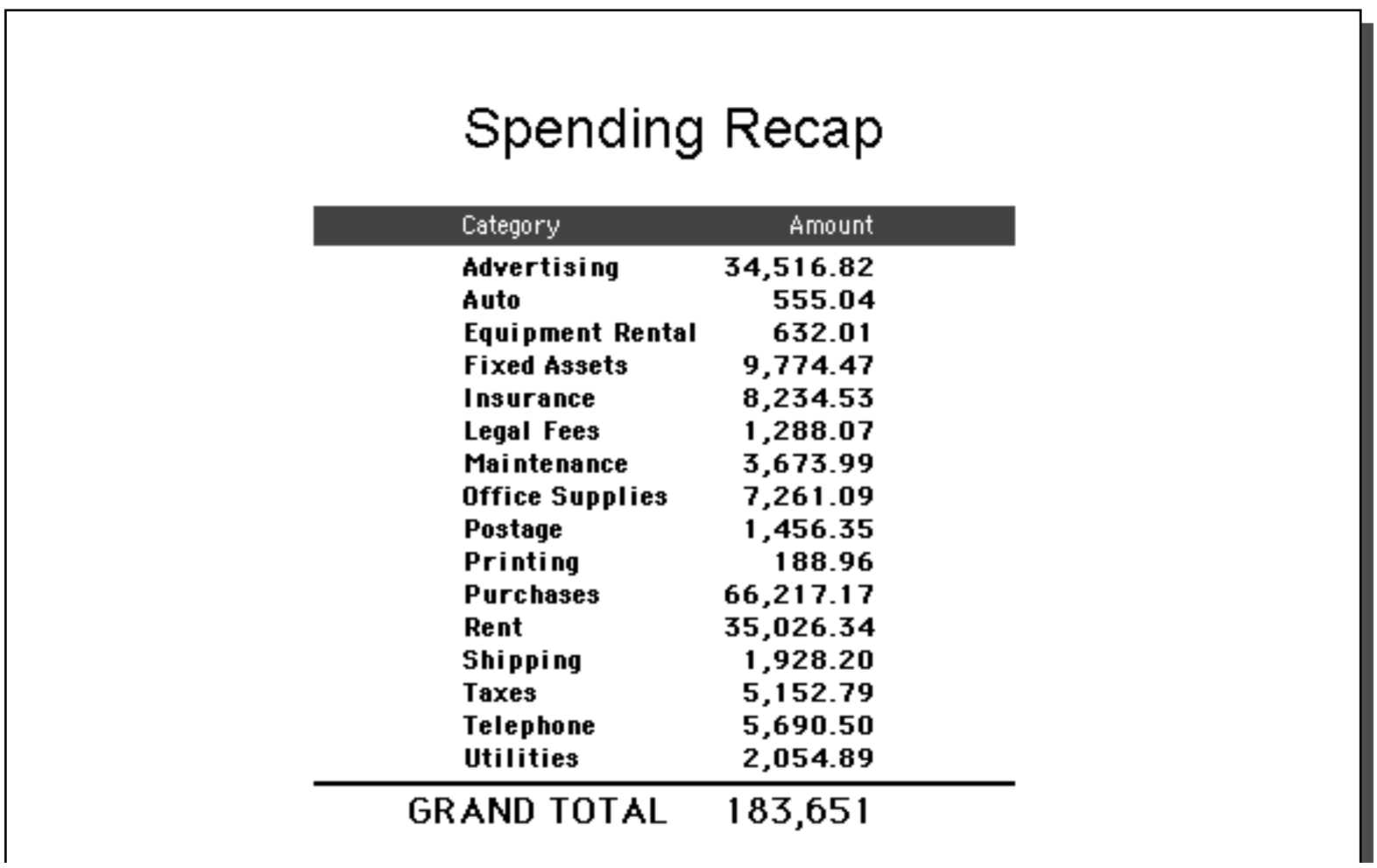


Printing Summaries Without Data

Sometimes you may want to print only summary records without printing any data records. The preferred technique for doing this is to use the **Outline Level** dialog to collapse the database before printing (see "STEP 3 - OUTLINE" on page 409). Another method is to create a zero-height data tile, as shown in this example.



The easiest way to make sure that the height of the **Data** tile is exactly zero is to use the **Dimensions** dialog (see "Viewing and Setting Exact Object Dimensions" on page 512). When this form is printed all of the data records are invisible because of the zero height Data tile and all you see are the summaries.



Spending Recap

Category	Amount
Advertising	34,516.82
Auto	555.04
Equipment Rental	632.01
Fixed Assets	9,774.47
Insurance	8,234.53
Legal Fees	1,288.07
Maintenance	3,673.99
Office Supplies	7,261.09
Postage	1,456.35
Printing	188.96
Purchases	66,217.17
Rent	35,026.34
Shipping	1,928.20
Taxes	5,152.79
Telephone	5,690.50
Utilities	2,054.89
GRAND TOTAL	183,651

Remember that all reports must have a data tile, even if (as in this case) it is not actually visible in the printed report.

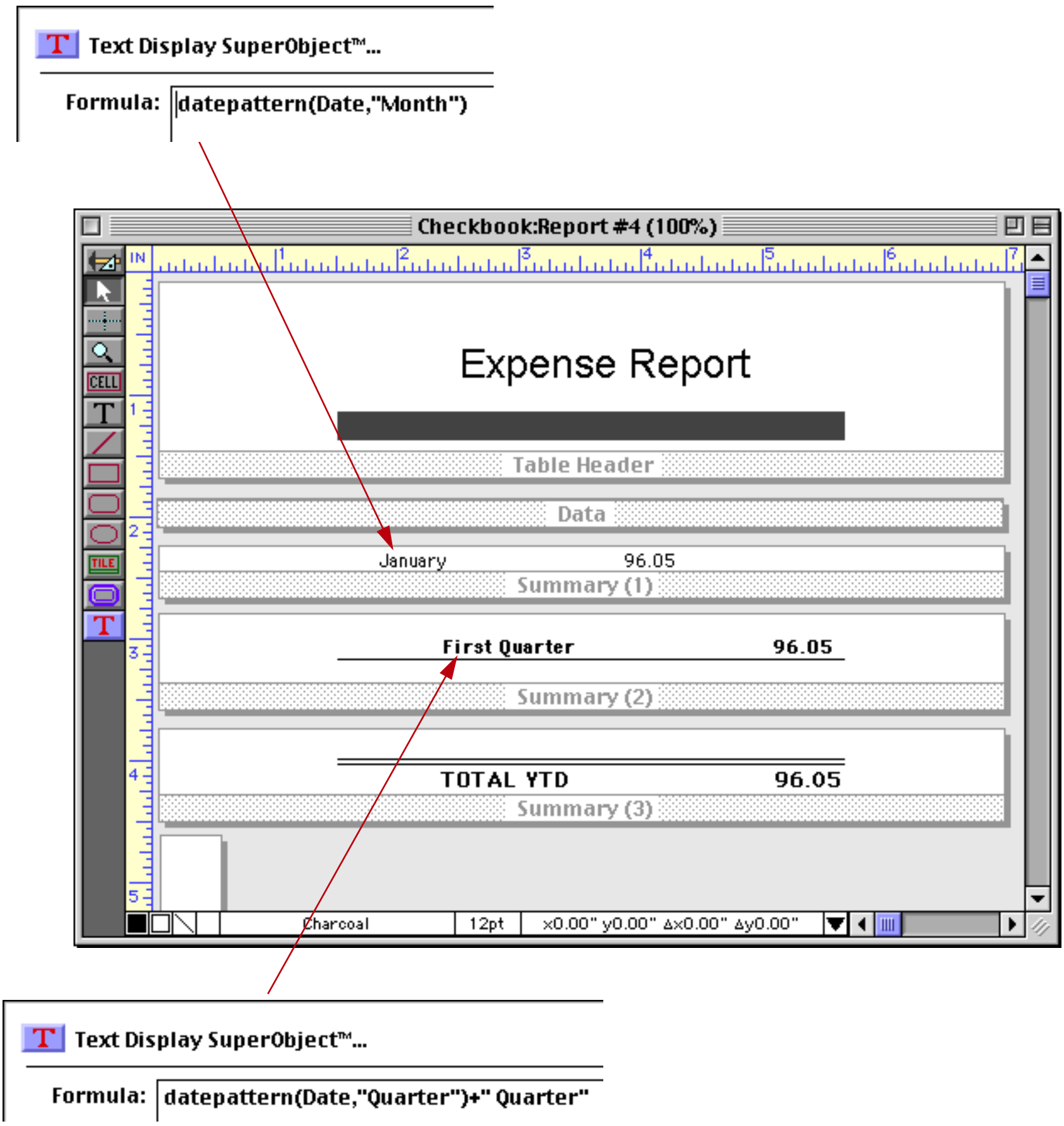
Printing Data Grouped by Month, Quarter or Year

When a database has been grouped by month, quarter or year, the dates should usually be formatted differently at each summary level. At the raw data level the entire date should be printed, while on the summary tiles only the month, quarter, or year should be printed.

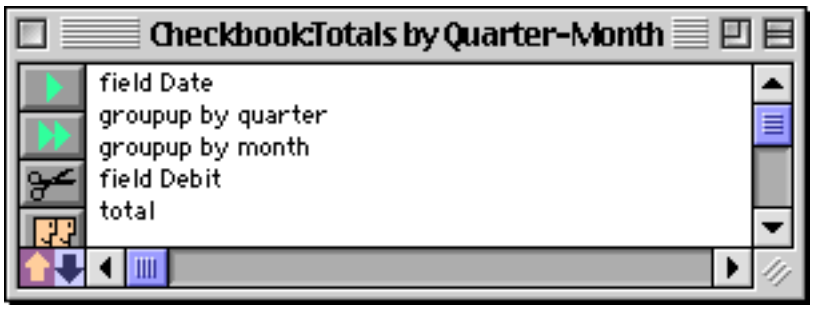
To format a Data Cell object to display only the month, quarter, or year, select the date cell object and choose the **Output Pattern** command in the Text Menu (see "Data Cell Custom Output Patterns" on page 638). This command sets the output pattern for just the individual selected object. The pattern for a month summary could be Month yyyy, Mon-yy or mm-yy. For a quarter summary use qqqyy or Qtr "Qtr" yy. For a year use yy or yyyy. For more information about date output patterns see "Date Output Patterns" on page 295.

To print only the month, quarter or year with an auto-wrap or Text Display object use the `datepattern(` function (see "Converting Between Dates and Text" on page 1249) with one of the patterns mentioned in the previous paragraph.

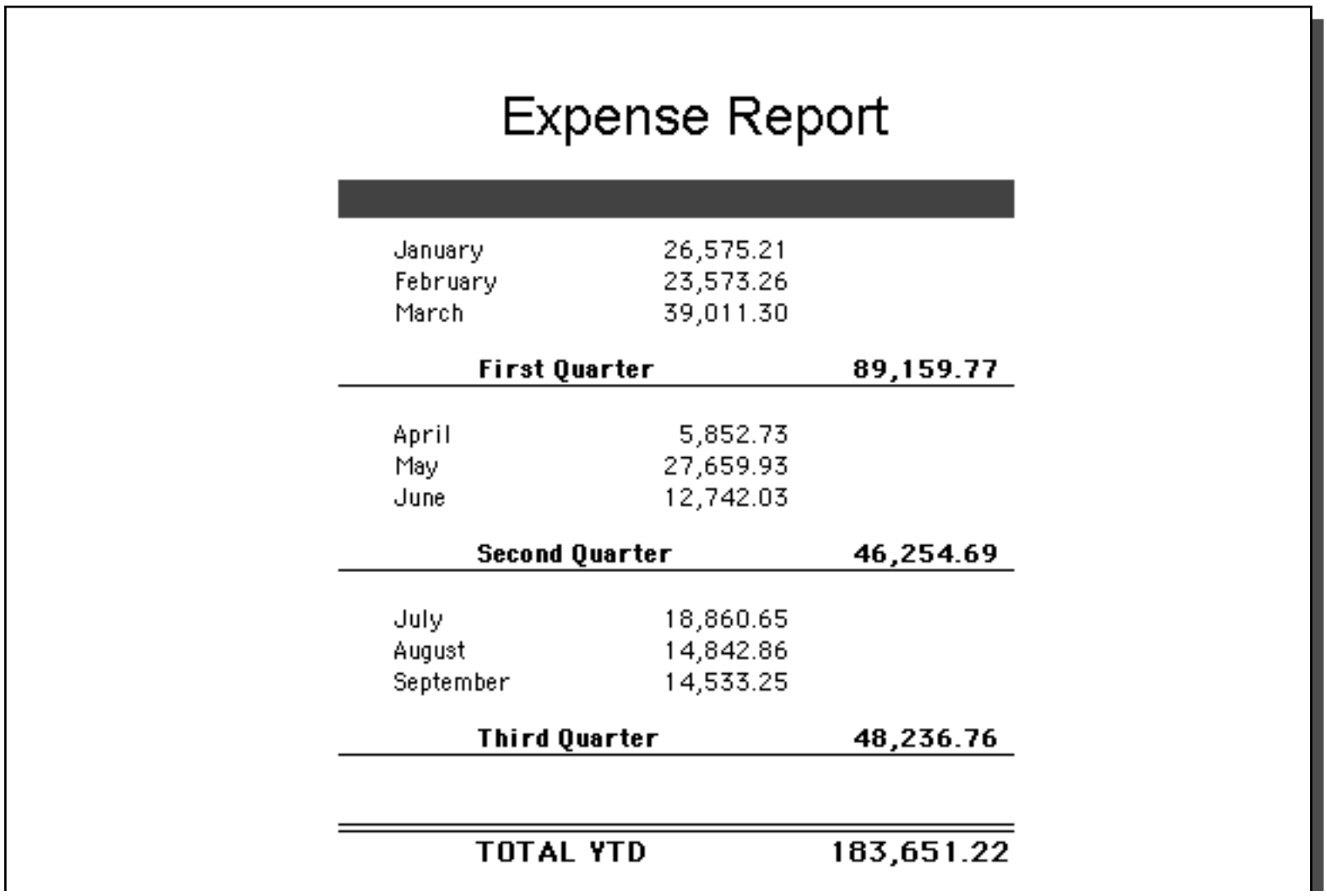
Here's an example that uses Text Display SuperObjects to display the date.



To prepare this report for printing you must **Group Up** by Quarter on the Date field (see "Grouping by Week, Month, Quarter, or Year" on page 400) then **Group Up** by Month, then **Total** the Debit field (see "Total" on page 403). Here is a procedure that will prep the database for you (see "Procedures" on page 1339).



Here is the printed report.

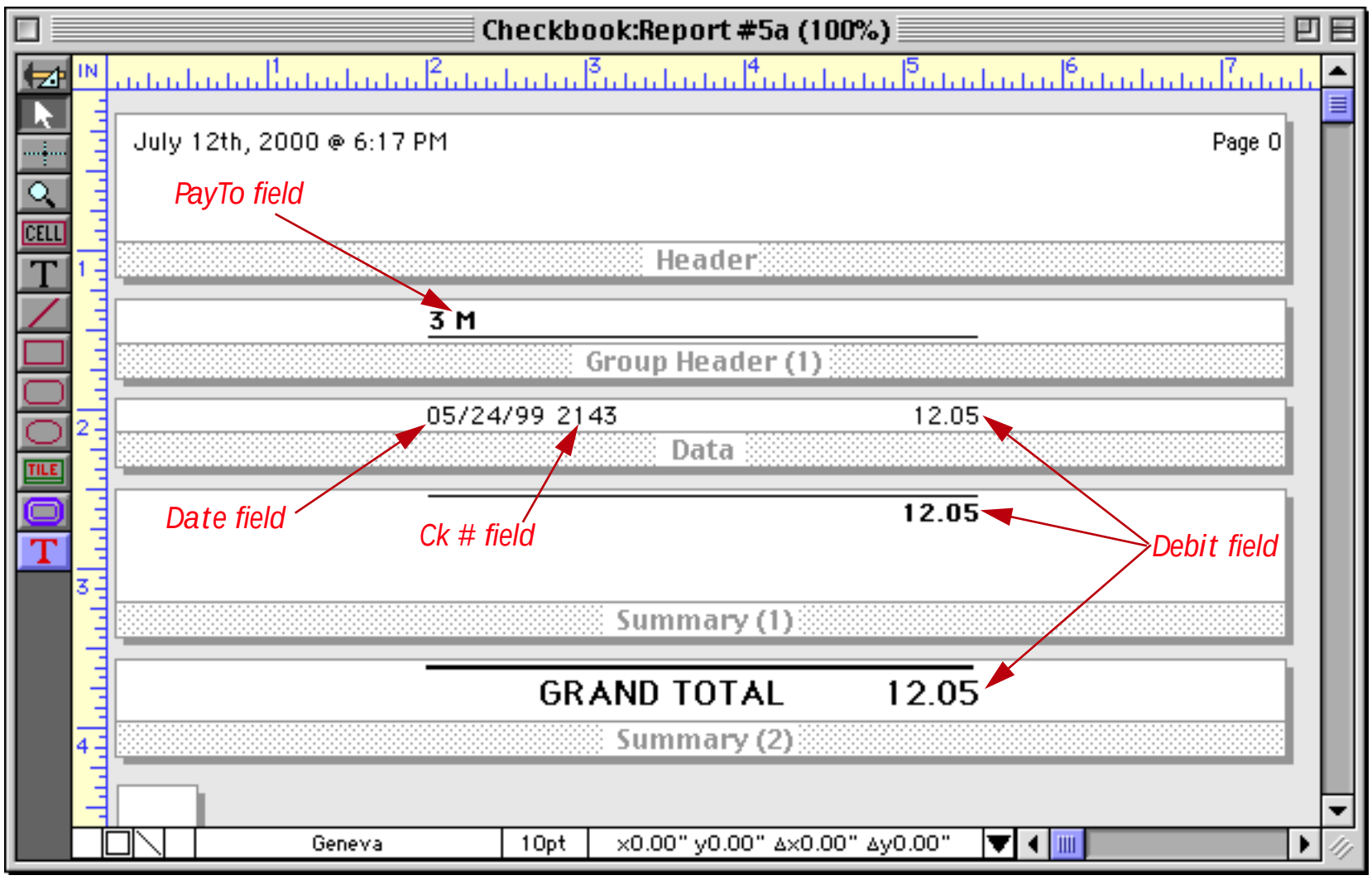


Expense Report

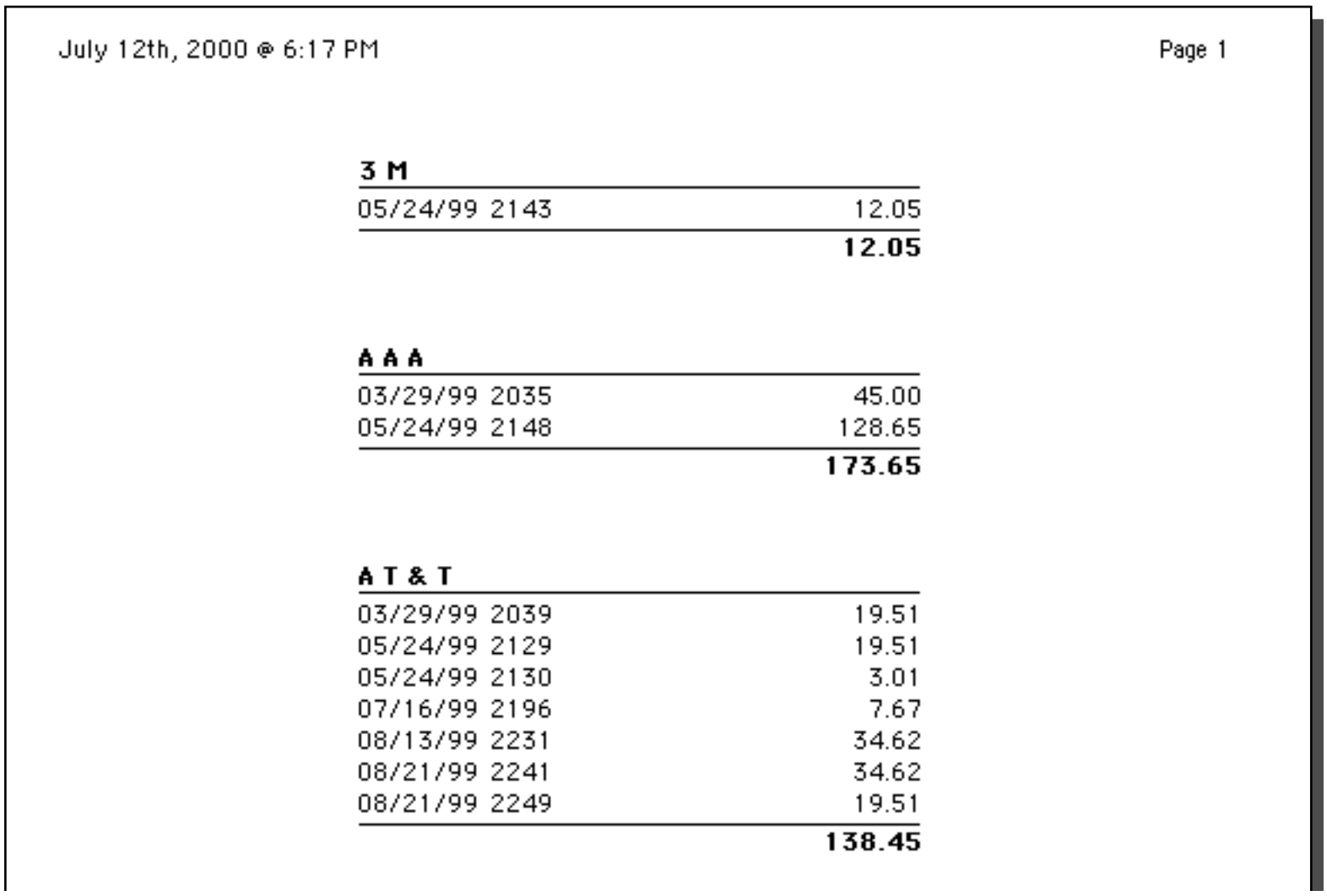
January	26,575.21	
February	23,573.26	
March	39,011.30	
First Quarter		**89,159.77**
April	5,852.73	
May	27,659.93	
June	12,742.03	
Second Quarter		**46,254.69**
July	18,860.65	
August	14,842.86	
September	14,533.25	
Third Quarter		**48,236.76**
TOTAL YTD		**183,651.22**

Group Headers

Panorama normally prints a header at the top of each report page. The **Group Header** tiles allow an individual header to be printed at the top of each group in the report. (Of course, the database must already be arranged into groups.) The **Group Header** tile can be used to print a title at the top of each group and also to provide extra spacing between groups. Here's an example of a report with a group header. This example assumes that the database has been grouped by the PayTo field.



When it's printed this report looks like this.



July 12th, 2000 @ 6:17 PM Page 1

3 M

05/24/99 2143	12.05
	12.05

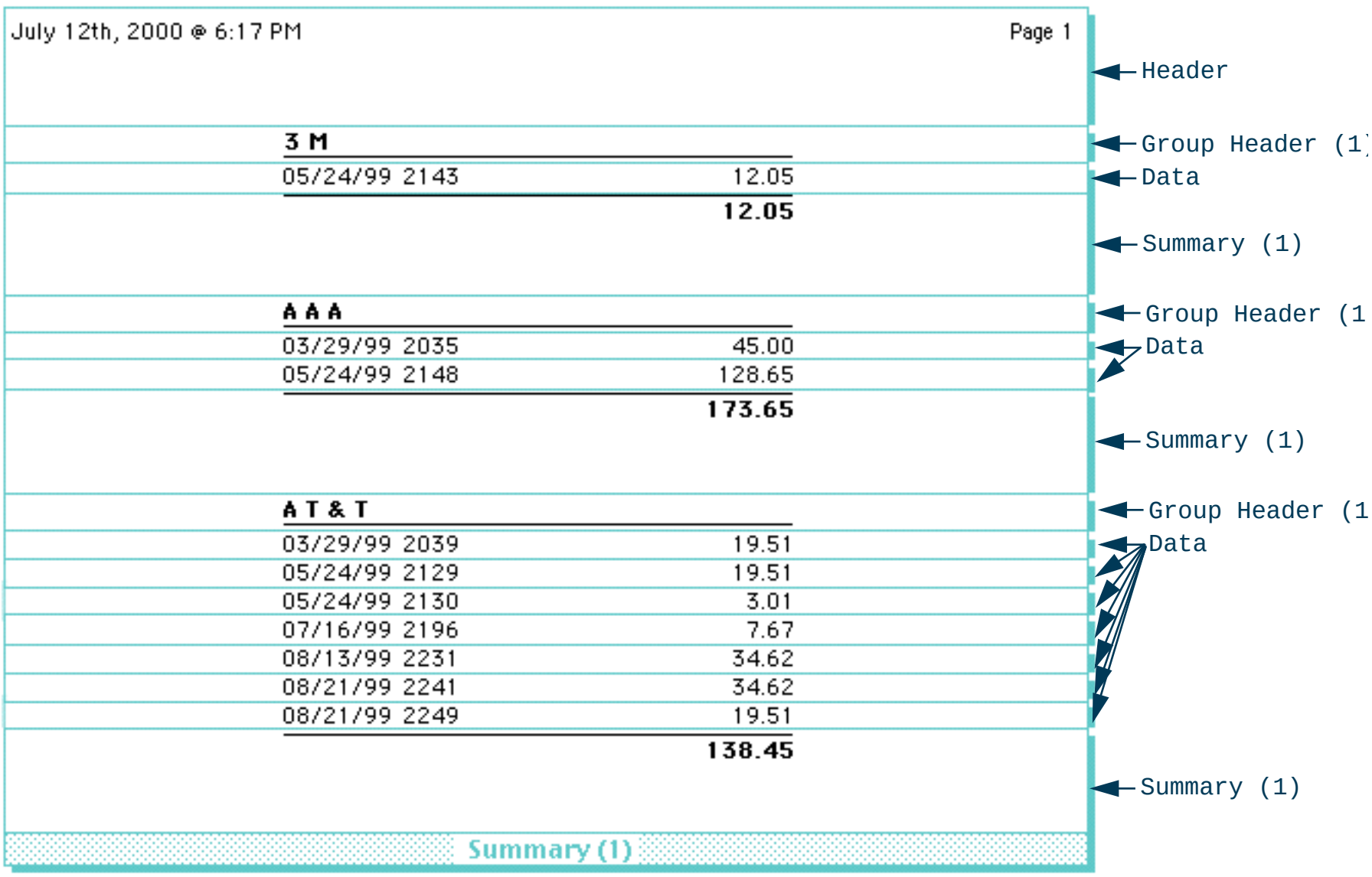
A A A

03/29/99 2035	45.00
05/24/99 2148	128.65
	173.65

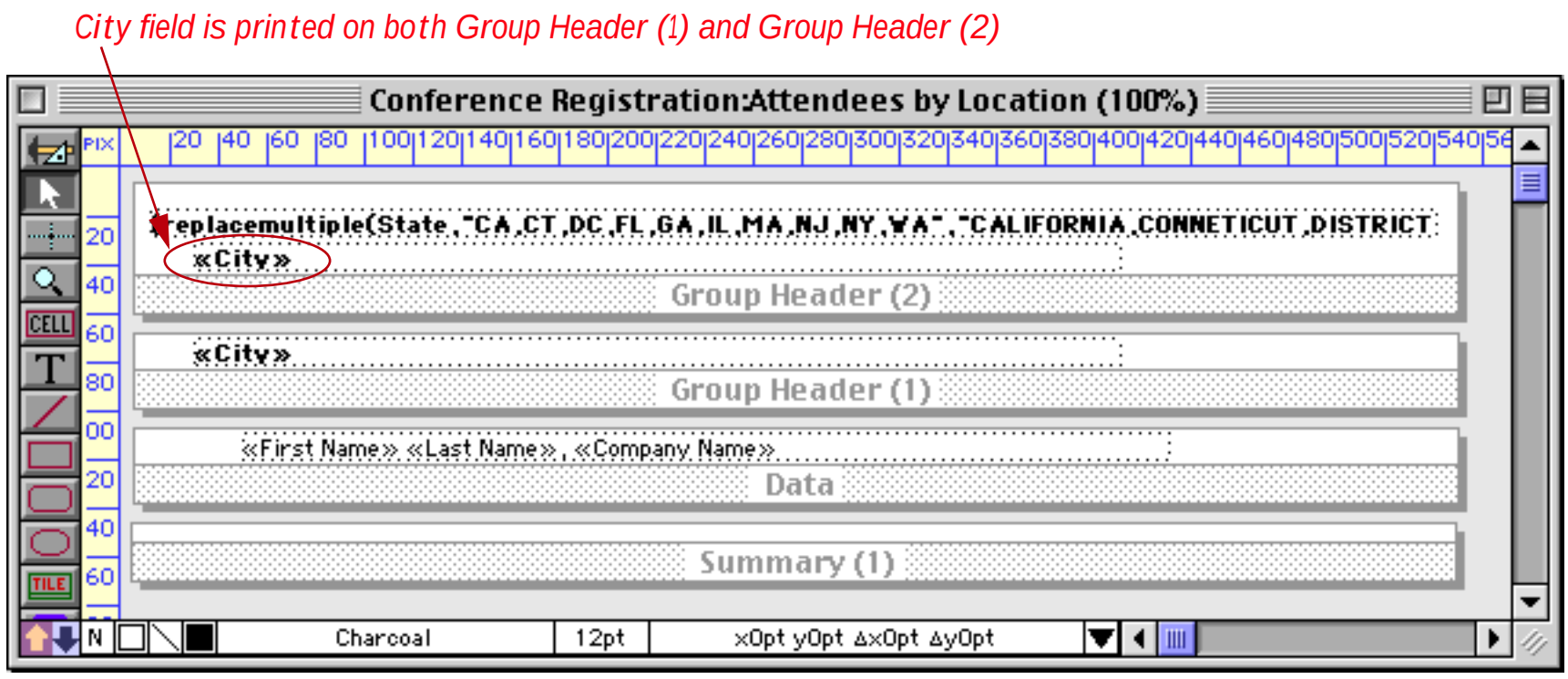
A T & T

03/29/99 2039	19.51
05/24/99 2129	19.51
05/24/99 2130	3.01
07/16/99 2196	7.67
08/13/99 2231	34.62
08/21/99 2241	34.62
08/21/99 2249	19.51
	138.45

This illustration shows how Panorama assembles the tiles to create the final report. At the beginning of each group it prints the group header, then the data tiles. At the end of the group it prints the summary tile.



Sometimes two groups start at the same spot. For example, in a database grouped by State and City, the start of the California group may also be the start of the Anaheim group. In this case the higher level wins, and Panorama will print the group header for California. Because of this, the higher level summary headers should include any graphics or data needed for each lower level, as shown in this illustration.



Here's the finished report,

CALIFORNIA

Alameda

Jim Abrahms, International Transportation Resources

Berkeley

Robert Sophie, Alvarado, Johnson, & Wright

Nancy Alexander, JPSA

Burbank

Keith Jacobs, Bath Co.

Canoga Park

Ken Taylor, Saticoy Litho

Compton

Steve Prendergass, Pacific Photo Labs

Costa Mesa

Joyce Ferrell, Church Aviation

Cupertino

Christy Alpert, Signal Research

Arthur Clairmont, South Coast Office Products

Harold Cobb, Cobb Associates

Sherry Grossman, Pablo Distribution

Peter Parks, Hamilton Press

Michelle Adams, Sceptre

Emeryville

Cynthia Knight, TBS Contracting, Inc.

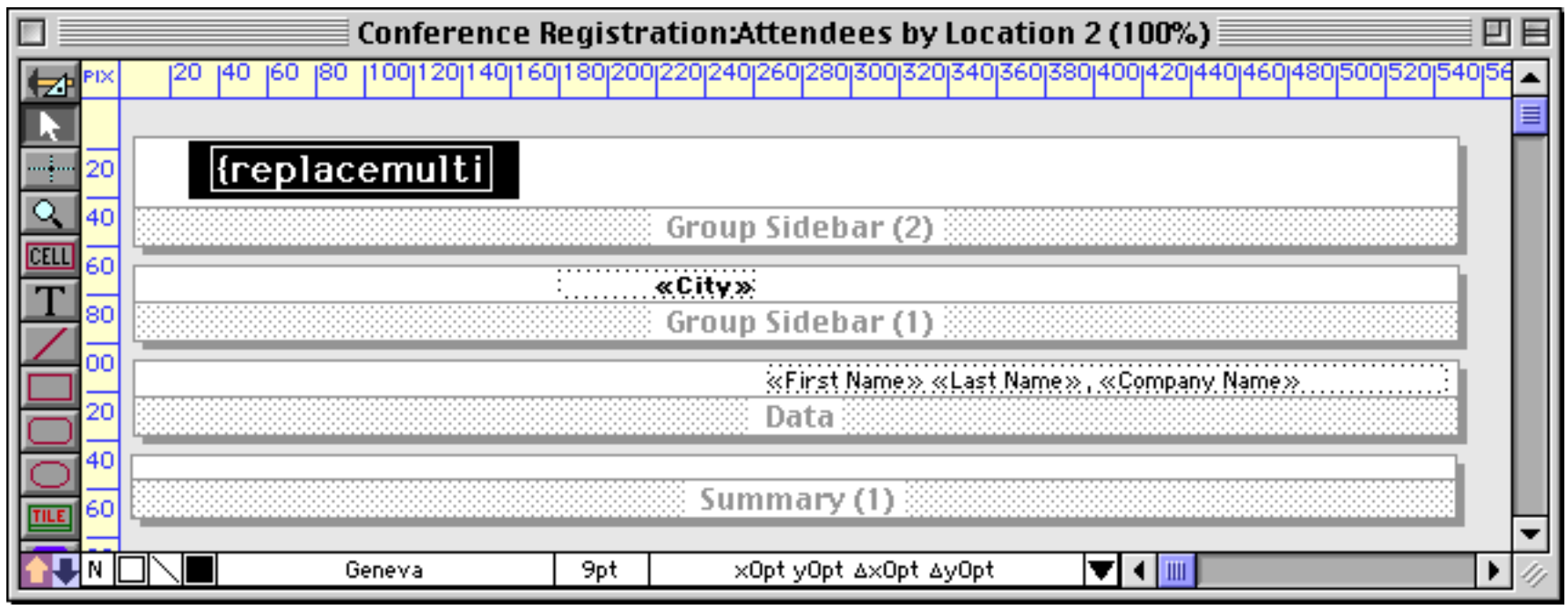
Here's how Panorama assembles the tiles to create this report.



Group Sidebars

The **Group Sidebar** tile allows you to print a “header” beside each group (either to the left or to the right) instead of above the group. The group sidebar tile is unusual because when it is printed, it will actually overlap the data tiles that follow. It is up to you to design the graphics on the group sidebar and data tiles so that they don’t interfere with each other. In other words, all the text and graphics on the group sidebar tile should be to the left with the text and graphics on the data tile to the right, or vice versa.

Here is a report with two Group Sidebar tiles. The report is designed to be printed after the database has been grouped by State and then by City.



Here is the printed report. Notice that at the beginning of each state and city the sidebar prints next to the data. The sidebar is actually overlaying the data (and in the case of the state, the other sidebar) but because of the way the graphics are laid out the sidebars don't interfere with each other or with the data.

CALIFORNIA

Cupertino Christy Alpert, Signal Research
Arthur Clairmont, South Coast Office Products
Harold Cobb, Cobb Associates
Sherry Grossman, Pablo Distribution
Peter Parks, Hamilton Press
Michelle Adams, Sceptre

Long Beach Kathy Schwartz, Wendover Insurance Group

Los Angeles Charles Arrow, Arrow, Inc.
Dave Elko, First Row Group

Palo Alto Cindy Blunden, Hot Lines, Inc.
Robert Dorn, Valley Services
Roxie Jacobsen, Alpha Pic
Donna Brady, Challenger Air Cargo

San Diego Jan Morgan, McCormick-Ridder
Frank Pearce, Taylor & Associates
Ray Cavalier, Stagg Instant Press

San Francisco Kris Frazee, Mariani Publishing
Mike Reynolds, Birch Catering
John Chelsie, British Consolate
George Kraus, Sand Hill Technology

San Mateo Roy Jones, Wheeler & Assoc
Donald Mentzinger, Sequoia Advertising
Marty Abrams, Minutemen Press
Karly Gersh, Hamilton Printing

San Rafael Isabel Alston, Leader Systems
Lew Farrell, Coast Label & Supply
Barry Church, Montbuild
Mark Kochs, Alexander Escrow

Santa Ana Tim Hill, Alameda Escrow
Charlene Stein, Borregas-Wilson Inc.
Jared Alexander, Kinetic Computing

FLORIDA

Naples Mark Lauing, Sherman-Davis

Tallahassee Robin Knight, Fico Appliance Service

Tampa Ralph Webster, Nordhoff Aviation

GEORGIA

Marietta Kay Desia, Industrial Development Board

ILLINOIS

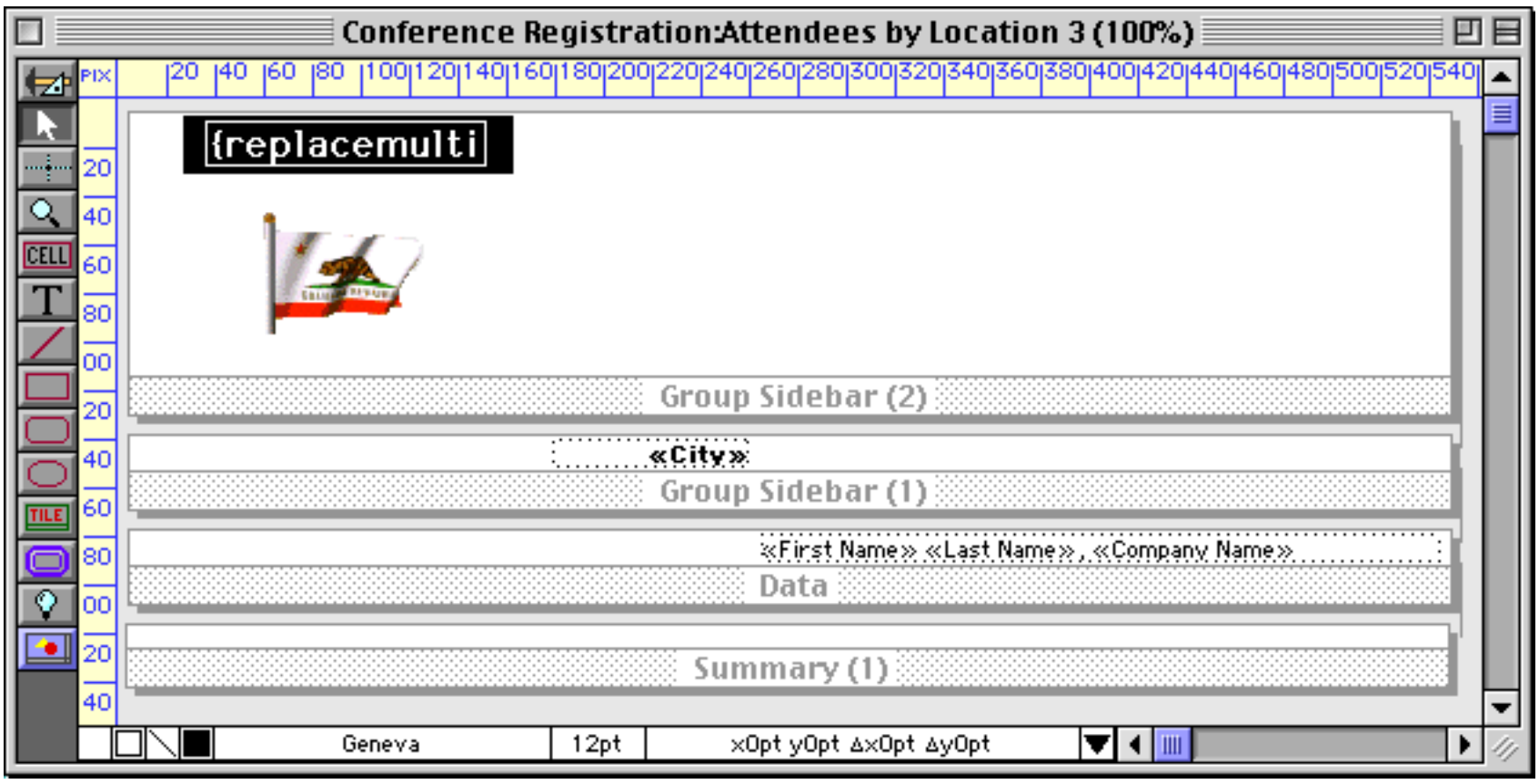
Chicago Anthony Campbell, Drake Inc.
Russ Malone, Northridge Bakeries

Here's how Panorama assembles the tiles to create this report. Because of the overlapping tiles this can be a bit confusing, so the Data tiles have been omitted from this illustration. The Group Sidebar(2) tiles are shown in green, and the Group Sidebar(1) tiles are shown in blue.



Tip: Although the group sidebar will overlap the data tiles when the report is printed, they must not overlap on the form window.

You can add images to a group sidebar using Flash Art. This report is exactly the same as the previous example except for the addition of the Super Flash Art image displaying the state flag (see See “Flash Art™” on page 748).



Here is the printed report.

CALIFORNIA



Cupertino	Christy Alpert, Signal Research
	Arthur Clairmont, South Coast Office Products
	Harold Cobb, Cobb Associates
	Sherry Grossman, Pablo Distribution
	Peter Parks, Hamilton Press
	Michelle Adams, Sceptre
Long Beach	Kathy Schwartz, Wendover Insurance Group
Los Angeles	Charles Arrow, Arrow, Inc.
	Dave Elko, First Row Group
Palo Alto	Cindy Blunden, Hot Lines, Inc.
	Robert Dorn, Valley Services
	Roxie Jacobsen, Alpha Pic
	Donna Brady, Challenger Air Cargo
San Diego	Jan Morgan, McCormick-Ridder
	Frank Pearce, Taylor & Associates
	Ray Cavalier, Stagg Instant Press
San Francisco	Kris Frazee, Mariani Publishing
	Mike Reynolds, Birch Catering
	John Chelsie, British Consolate
	George Kraus, Sand Hill Technology
San Mateo	Roy Jones, Wheeler & Assoc
	Donald Mentzinger, Sequoia Advertising
	Marty Abrams, Minutemen Press
	Karly Gersh, Hamilton Printing
San Rafael	Isabel Alston, Leader Systems
	Lew Farrell, Coast Label & Supply
	Barry Church, Montbuild
	Mark Kochs, Alexander Escrow
Santa Ana	Tim Hill, Alameda Escrow
	Charlene Stein, Borregas-Wilson Inc.
	Jared Alexander, Kinetic Computing

FLORIDA



Naples	Mark Lauing, Sherman-Davis
Tallahassee	Robin Knight, Fico Appliance Service
Tampa	Ralph Webster, Nordhoff Aviation

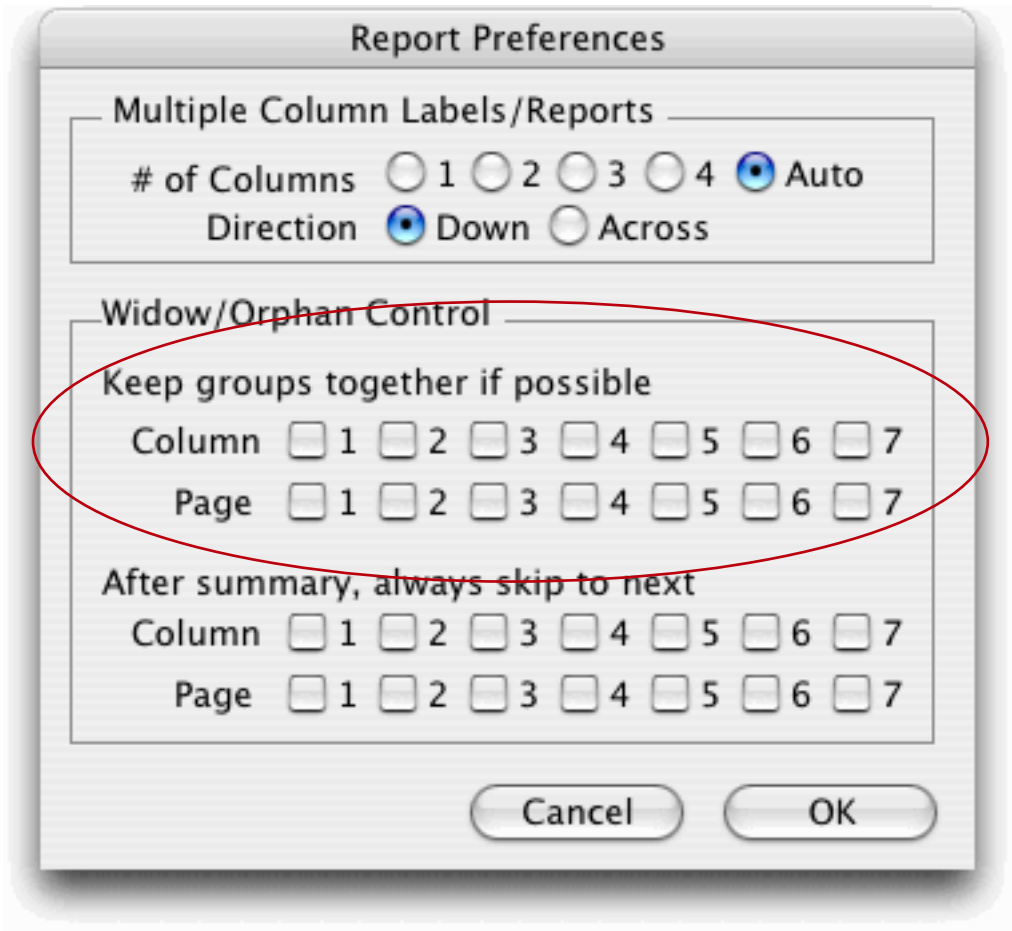
GEORGIA

Marietta	Kay Desia, Industrial Development Board

Notice that the for the state of Florida the sidebar is actually taller than the data records. In that case Panorama will leave a gap between the last data record in a group and the first data record of the next group.

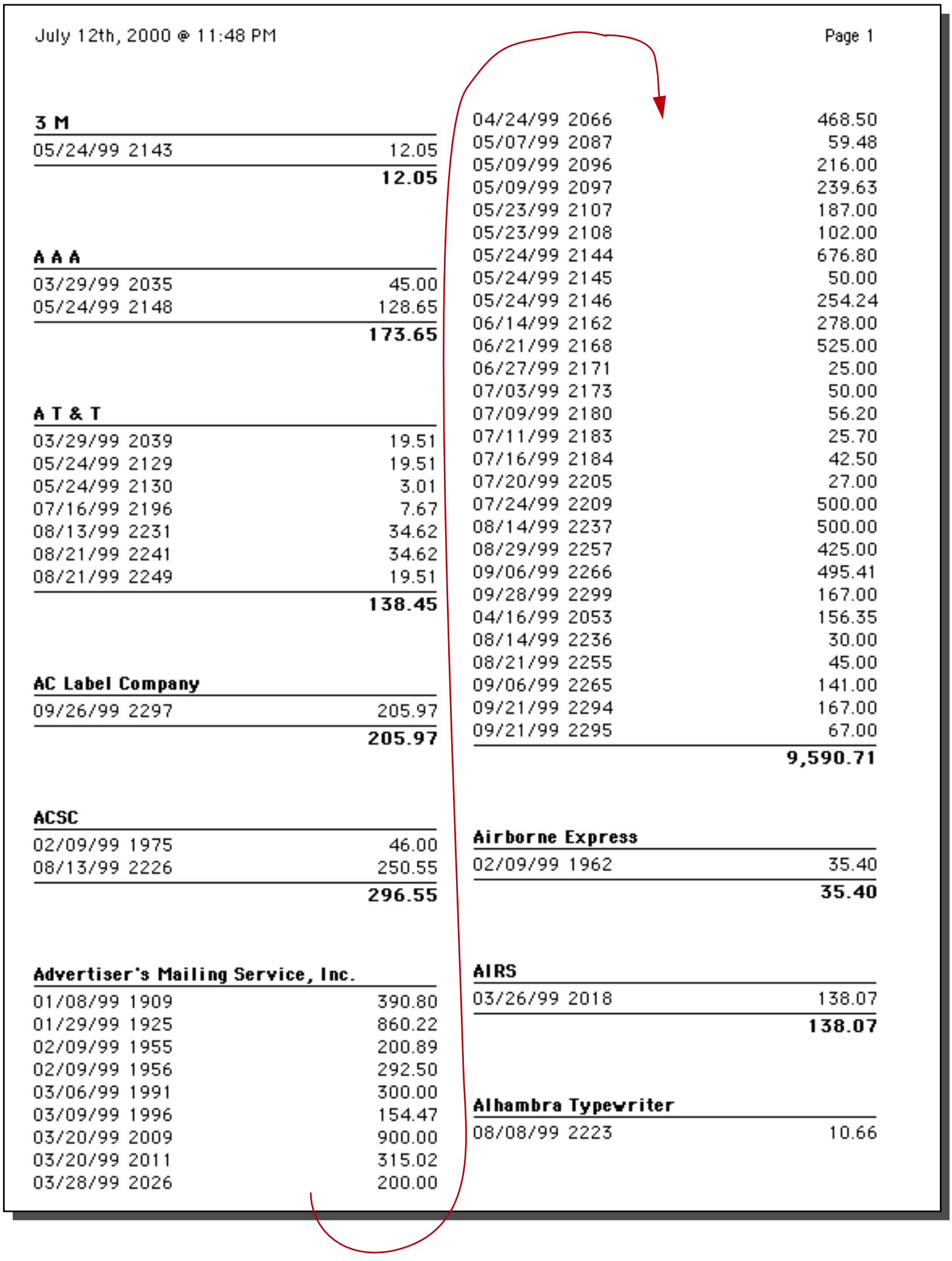
Keeping a Group Together on a Column or Page

If you wish, Panorama can automatically make sure that groups are not divided across column or page boundaries. This can make a report easier to read and much more attractive. (Of course it can only do this for groups that are small enough to fit on a single column or page.) This feature is called widow / orphan control and is available in the **Report Preferences** dialog in the Setup menu.



Select the largest group level you want to keep together on a column or page (or both).

For example, consider the report shown below. The group of records for Advertiser's Mailing Service is split across two columns.



July 12th, 2000 @ 11:48 PM Page 1

3 M

05/24/99 2143	12.05
	12.05

A A A

03/29/99 2035	45.00
05/24/99 2148	128.65
	173.65

A T & T

03/29/99 2039	19.51
05/24/99 2129	19.51
05/24/99 2130	3.01
07/16/99 2196	7.67
08/13/99 2231	34.62
08/21/99 2241	34.62
08/21/99 2249	19.51
	138.45

AC Label Company

09/26/99 2297	205.97
	205.97

ACSC

02/09/99 1975	46.00
08/13/99 2226	250.55
	296.55

Advertiser's Mailing Service, Inc.

01/08/99 1909	390.80
01/29/99 1925	860.22
02/09/99 1955	200.89
02/09/99 1956	292.50
03/06/99 1991	300.00
03/09/99 1996	154.47
03/20/99 2009	900.00
03/20/99 2011	315.02
03/28/99 2026	200.00
04/24/99 2066	468.50
05/07/99 2087	59.48
05/09/99 2096	216.00
05/09/99 2097	239.63
05/23/99 2107	187.00
05/23/99 2108	102.00
05/24/99 2144	676.80
05/24/99 2145	50.00
05/24/99 2146	254.24
06/14/99 2162	278.00
06/21/99 2168	525.00
06/27/99 2171	25.00
07/03/99 2173	50.00
07/09/99 2180	56.20
07/11/99 2183	25.70
07/16/99 2184	42.50
07/20/99 2205	27.00
07/24/99 2209	500.00
08/14/99 2237	500.00
08/29/99 2257	425.00
09/06/99 2266	495.41
09/28/99 2299	167.00
04/16/99 2053	156.35
08/14/99 2236	30.00
08/21/99 2255	45.00
09/06/99 2265	141.00
09/21/99 2294	167.00
09/21/99 2295	67.00
	9,590.71

Airborne Express

02/09/99 1962	35.40
	35.40

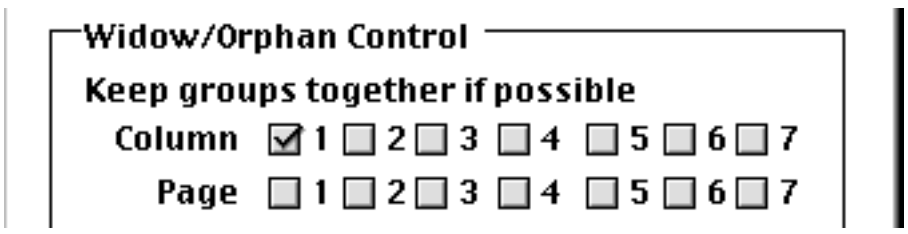
AIRS

03/26/99 2018	138.07
	138.07

Alhambra Typewriter

08/08/99 2223	10.66

To prevent this from happening, open the **Report Preferences** dialog and check the box as shown below.



Panorama will automatically push the group to the next column so that it is no longer split in the middle.

July 12th, 2000 @ 11:56 PM Page 1

3 M	
05/24/99 2143	12.05
	12.05

A A A	
03/29/99 2035	45.00
05/24/99 2148	128.65
	173.65

A T & T	
03/29/99 2039	19.51
05/24/99 2129	19.51
05/24/99 2130	3.01
07/16/99 2196	7.67
08/13/99 2231	34.62
08/21/99 2241	34.62
08/21/99 2249	19.51
	138.45

AC Label Company	
09/26/99 2297	205.97
	205.97

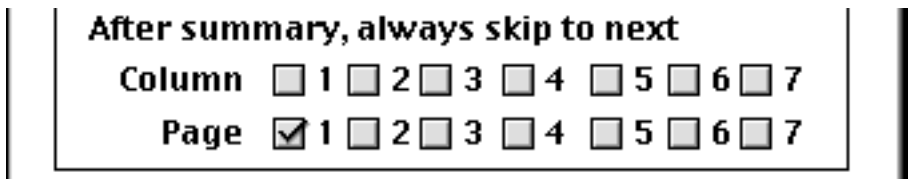
ACSC	
02/09/99 1975	46.00
08/13/99 2226	250.55
	296.55

Advertiser's Mailing Service, Inc.	
01/08/99 1909	390.80
01/29/99 1925	860.22
02/09/99 1955	200.89
02/09/99 1956	292.50
03/06/99 1991	300.00
03/09/99 1996	154.47
03/20/99 2009	900.00
03/20/99 2011	315.02
03/28/99 2026	200.00
04/24/99 2066	468.50
05/07/99 2087	59.48
05/09/99 2096	216.00
05/09/99 2097	239.63
05/23/99 2107	187.00
05/23/99 2108	102.00
05/24/99 2144	676.80
05/24/99 2145	50.00
05/24/99 2146	254.24
06/14/99 2162	278.00
06/21/99 2168	525.00
06/27/99 2171	25.00
07/03/99 2173	50.00
07/09/99 2180	56.20
07/11/99 2183	25.70
07/16/99 2184	42.50
07/20/99 2205	27.00
07/24/99 2209	500.00
08/14/99 2237	500.00
08/29/99 2257	425.00
09/06/99 2266	495.41
09/28/99 2299	167.00
04/16/99 2053	156.35
08/14/99 2236	30.00
08/21/99 2255	45.00
09/06/99 2265	141.00
09/21/99 2294	167.00
09/21/99 2295	67.00
	9,590.71

If the checkbox in the Page row had been checked Panorama would only have made sure that a group wasn't split across a page - it wouldn't worry about group's split across a column like this.

Starting a Group on a New Column or Page

If you wish, Panorama can automatically skip to a new column or a new page at the beginning of each new group. To do this, choose **Report Preferences** from the Setup Menu, then choose the summary levels that triggers skipping to a new column or page.



Here's what the first few pages of this report looks like. Only one group is printed per page.

July 15th, 2000 @ 12:08 PM Page 1

3 M

05/24/99 2143	12.05
	12.05

July 15th, 2000 @ 12:08 PM Page 2

A A A

03/29/99 2035	45.00
05/24/99 2148	128.65
	173.65

July 15th, 2000 @ 12:08 PM Page 3

A T & T

03/29/99 2039	19.51
05/24/99 2129	19.51
05/24/99 2130	3.01
07/16/99 2196	7.67
08/13/99 2231	34.62
08/21/99 2241	34.62
08/21/99 2249	19.51
	138.45

July 15th, 2000 @ 12:08 PM Page 4

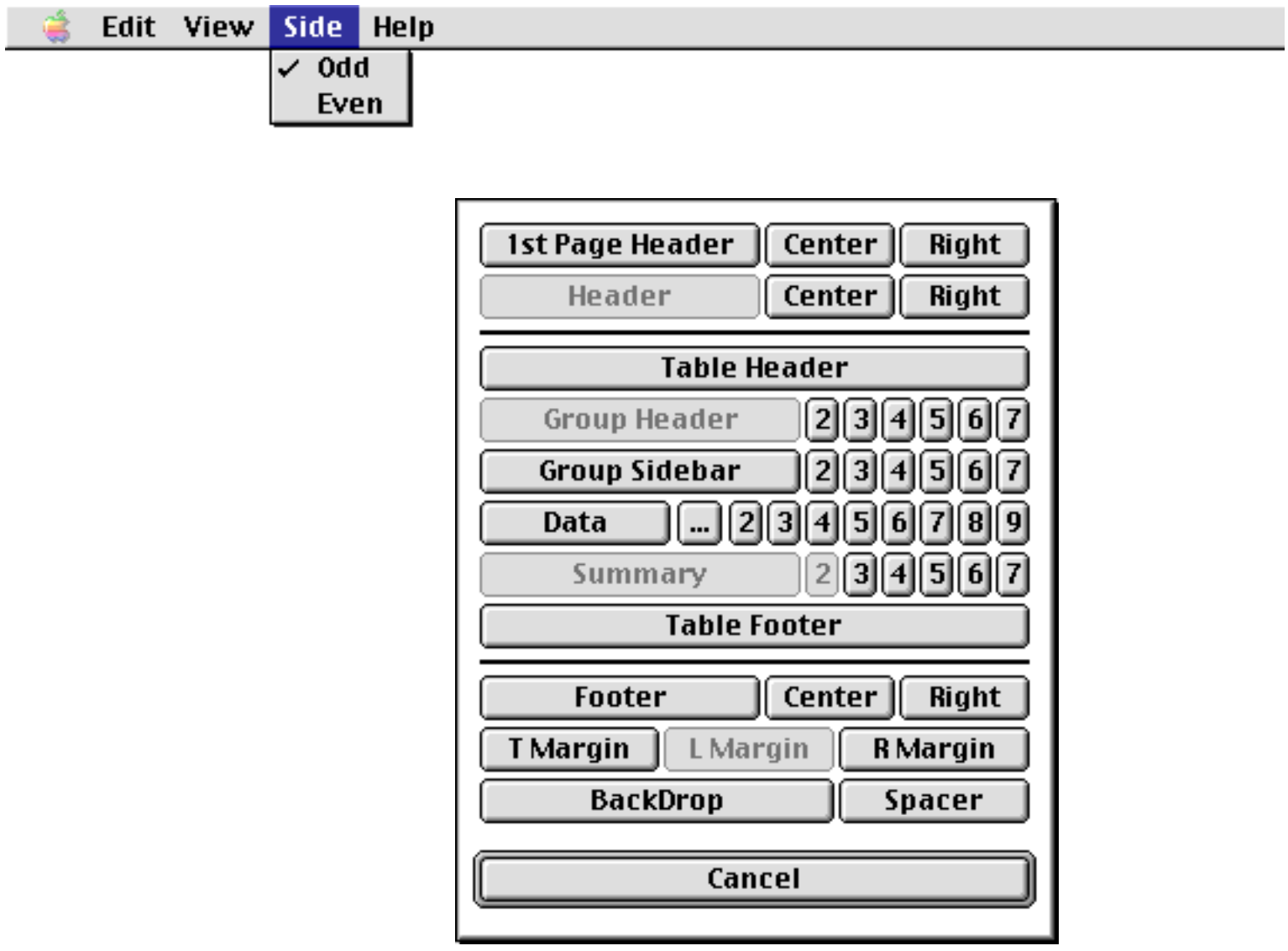
AC Label Company

09/26/99 2297	205.97
	205.97

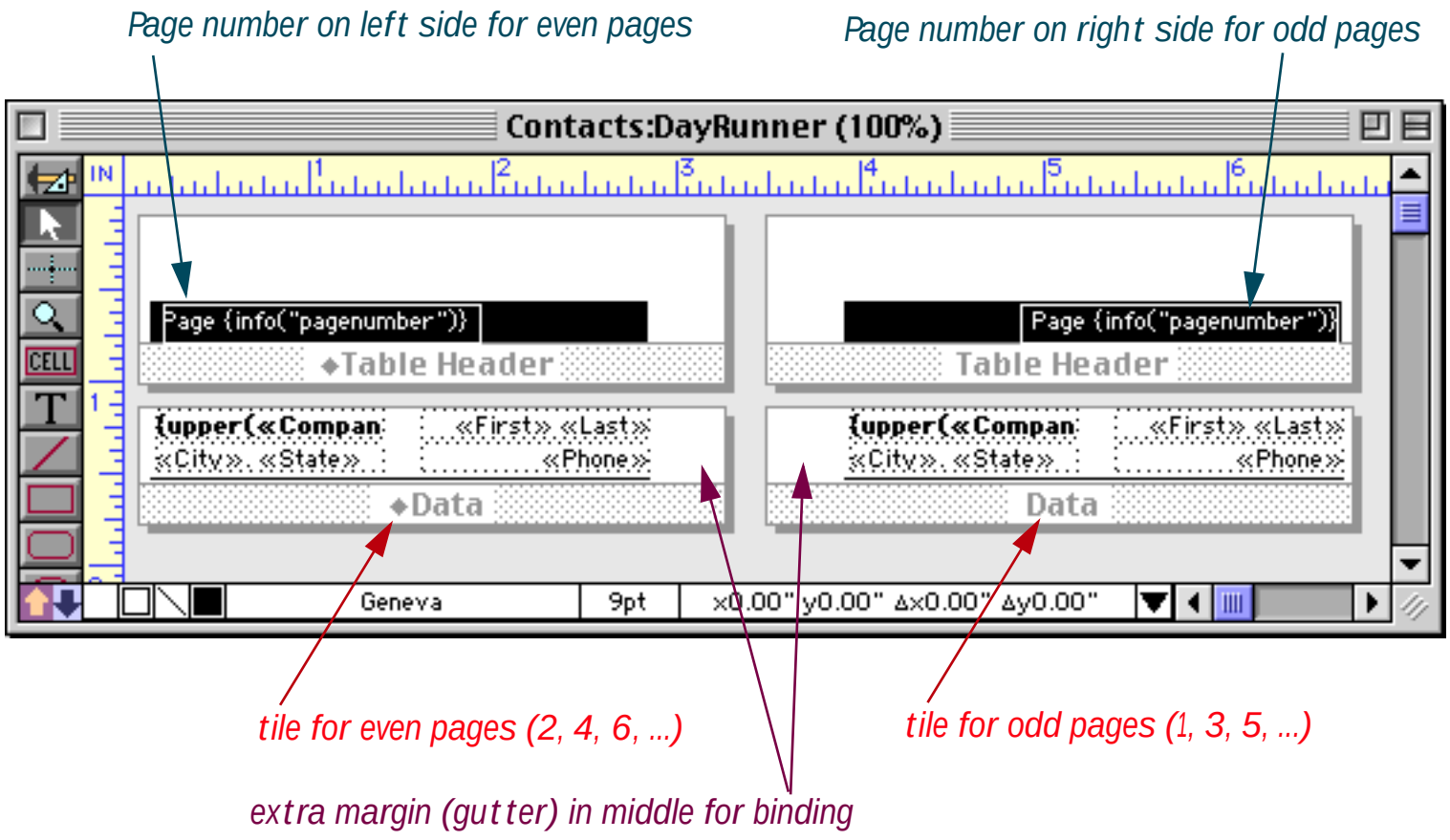
Even and Odd Page Layout

Panorama allows you to format the even and odd pages of a report differently. For example you could change the margins from even to odd pages to create a wide gutter in the middle, or flip the titles so the page number is always on the outside.

To set up these amazing feats, Panorama allows you to set up two sets of tiles: even page tiles and odd page tiles. When you create a tile with the **Specialized Tile** dialog, you can use the **Side** menu to select whether the tile is for even or odd pages. (The **Side** menu only appears when this dialog is open.)



You can recognize an even tile on a form by the ◊ symbol in front of the tile name.

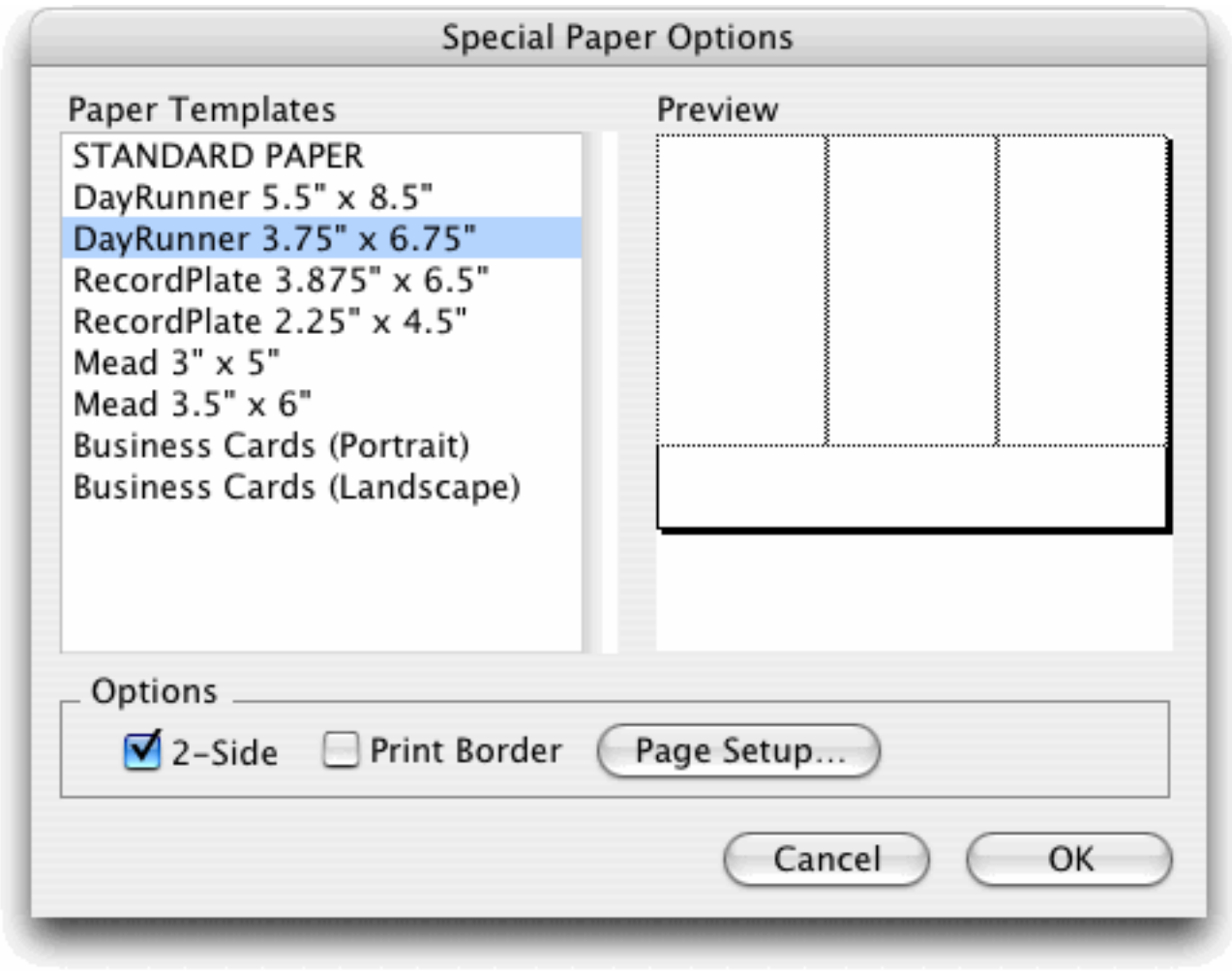


It's not necessary to create a complete duplicate set of both even and odd tiles. You only need to create even tiles for tiles that are actually different from the odd tiles. For example, suppose you want the left margin to be different on even and odd sides, but everything else is the same. In that case you would create an odd data tile, header tiles, etc. along with both an odd and even left margin tile.

Special Paper Options

The **Special Paper Options** dialog (Setup Menu) allows Panorama to print reports with multiple page images on a single physical sheet of paper. After the report is printed you can cut up the individual pages…usually so that they can be inserted in an organizer notebook. It also allows a report to be printed double sided (you must print one side, then turn the paper over and feed it back into the printer to print the second side). Note: You must turn off background printing to use the double sided option.

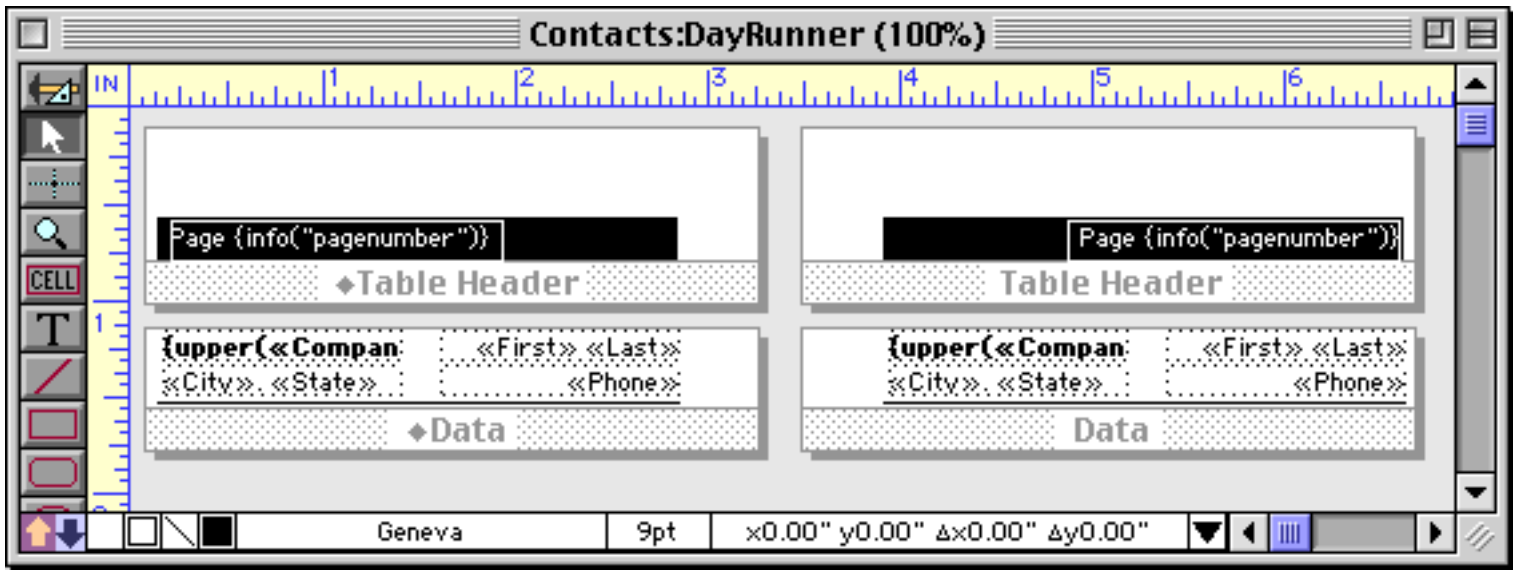
The left side of the **Special Paper Options** dialog displays a list of paper formats.



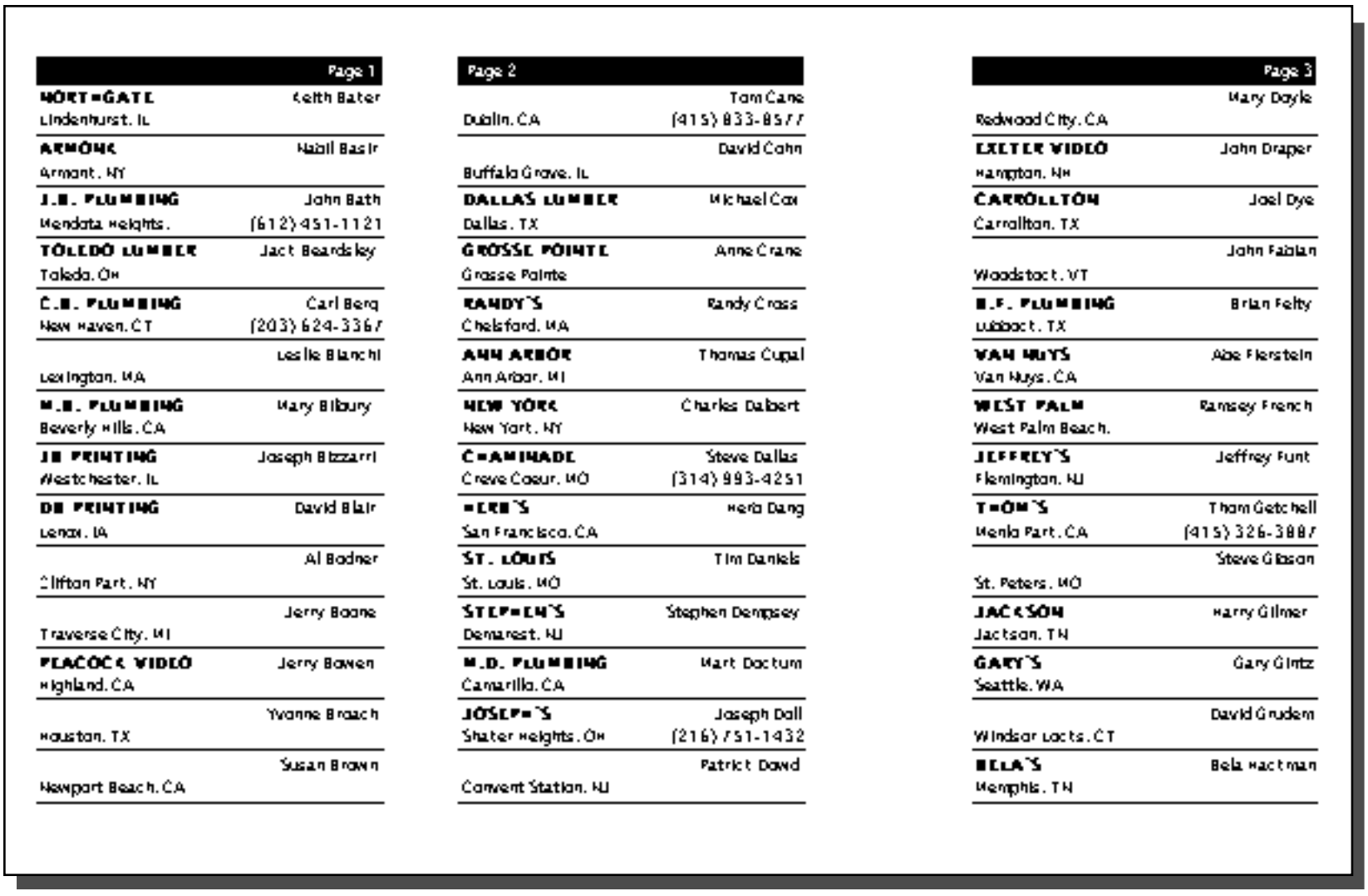
The default format, Standard Paper, prints normally on the entire page. The other options split the physical paper into two or more separate subpages. The area on the right shows a preview of how the paper will be split into separate pages. If necessary, you should use the **Page Setup** dialog to adjust the paper orientation (portrait vs. landscape) to make sure that all of the subpages fit on the paper.

If you select a format with two or more subpages, Panorama will print a separate report page on each subpage. For example, if you use the DayRunner 5.5" x 8.5" format (two subpages per sheet), Panorama will print report pages 1 and 2 on the first sheet, report pages 3 and 4 on the second sheet, etc. You must design your report so that each report page fits on a subpage.

The report below is designed to print using the DayRunner 3.75" x 6.75" option.



This option will cause 3 DayRunner pages to print on one, real page, like this.



Page 1

NORTHGATE	Keith Bater
Lindenhurst, IL	
ARMONK	Nabil Bashir
Armont, NY	
J.B. PLUMBING	John Bath
Mendota Heights,	(612)451-1121
TOLEDO LUMBER	Jack Beardsley
Toledo, OH	
C.B. PLUMBING	Carl Berg
New Haven, CT	(203) 624-3367
	Leslie Bianchi
Lexington, MA	
M.B. PLUMBING	Mary Bilbury
Beverly Hills, CA	
JB PRINTING	Joseph Bizzarri
Westchester, IL	
DB PRINTING	David Blair
Lenox, IA	
	Al Bodner
Clifton Part, NY	
	Jerry Boone
Traverse City, MI	
PEACOCK VIDEO	Jerry Bowen
Highland, CA	
	Yvonne Broach
Houston, TX	
	Susan Brown
Newport Beach, CA	

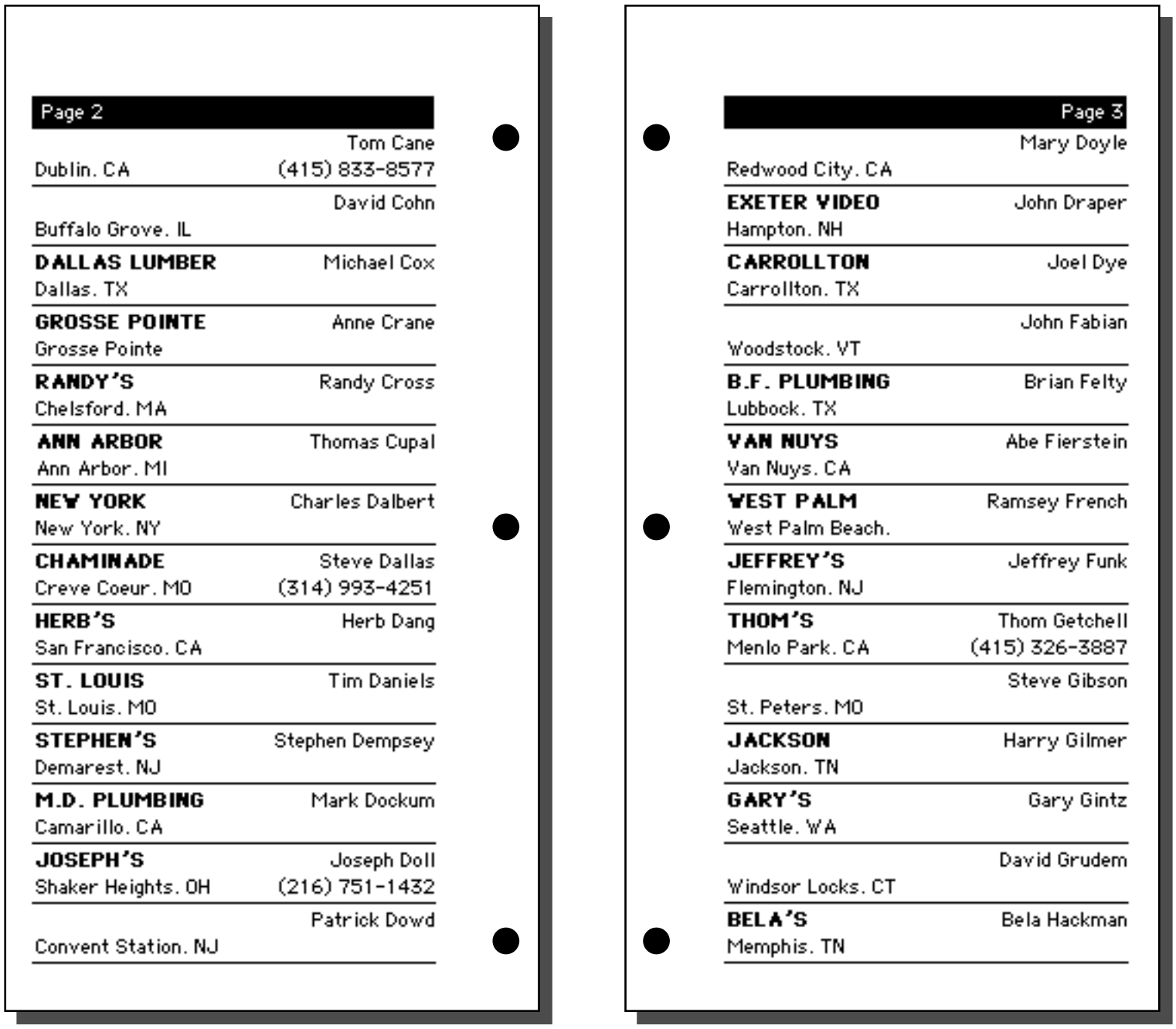
Page 2

	Tom Cane
Dublin, CA	(415) 833-8577
	David Cohn
Buffalo Grove, IL	
DALLAS LUMBER	Michael Cox
Dallas, TX	
GROSSE POINTE	Anne Crane
Grosse Pointe	
RANDY'S	Randy Cross
Chelsford, MA	
ANN ARBOR	Thomas Cupal
Ann Arbor, MI	
NEW YORK	Charles Dalbert
New Yort, NY	
CAMINADE	Steve Dallas
Creve Coeur, MO	(314) 993-4251
HERB'S	Herb Dang
San Francisco, CA	
ST. LOUIS	Tim Daniels
St. Louis, MO	
STEPHEN'S	Stephen Dempsey
Demarest, NJ	
M.D. PLUMBING	Mart Doctum
Camarillo, CA	
JOSEPH'S	Joseph Doll
Shaker Heights, OH	(216) 751-1432
	Patrick David
Convent Station, NJ	

Page 3

	Mary Doyle
Redwood City, CA	
EXETER VIDEO	John Draper
Hampton, NH	
CARROLLTON	Joel Dye
Carrollton, TX	
	John Fabian
Woodstock, VT	
B.F. PLUMBING	Brian Felty
Lubbock, TX	
VAN NUYS	Abe Fierstein
Van Nuys, CA	
WEST PALM	Ramsey French
West Palm Beach,	
JEFFREY'S	Jeffrey Funt
Flemington, NJ	
THOM'S	Thom Getchell
Menlo Part, CA	(415) 326-3887
	Steve Gibson
St. Peters, MO	
JACKSON	Harry Gilmer
Jackson, TN	
GARY'S	Gary Glintz
Seattle, WA	
	David Grudem
Windsor Locks, CT	
BELA'S	Bela Hactman
Memphis, TN	

After the report is printed you can cut apart the individual pages, punch holes, and install them in a DayRunner style binder. (You can also purchase paper that is pre-perfed to be split apart this way.)



Page 2

	Tom Cane
Dublin. CA	(415) 833-8577
	David Cohn
Buffalo Grove. IL	
DALLAS LUMBER	Michael Cox
Dallas. TX	
GROSSE POINTE	Anne Crane
Grosse Pointe	
RANDY'S	Randy Cross
Chelsford. MA	
ANN ARBOR	Thomas Cupal
Ann Arbor. MI	
NEW YORK	Charles Dalbert
New York. NY	
CHAMINADE	Steve Dallas
Creve Coeur. MO	(314) 993-4251
HERB'S	Herb Dang
San Francisco. CA	
ST. LOUIS	Tim Daniels
St. Louis. MO	
STEPHEN'S	Stephen Dempsey
Demarest. NJ	
M.D. PLUMBING	Mark Dockum
Camarillo. CA	
JOSEPH'S	Joseph Doll
Shaker Heights. OH	(216) 751-1432
	Patrick Dowd
Convent Station. NJ	

Page 3

	Mary Doyle
Redwood City. CA	
EXETER VIDEO	John Draper
Hampton. NH	
CARROLLTON	Joel Dye
Carrollton. TX	
	John Fabian
Woodstock. VT	
B.F. PLUMBING	Brian Felty
Lubbock. TX	
VAN NUYS	Abe Fierstein
Van Nuys. CA	
WEST PALM	Ramsey French
West Palm Beach.	
JEFFREY'S	Jeffrey Funk
Flemington. NJ	
THOM'S	Thom Getchell
Menlo Park. CA	(415) 326-3887
	Steve Gibson
St. Peters. MO	
JACKSON	Harry Gilmer
Jackson. TN	
GARY'S	Gary Gintz
Seattle. WA	
	David Grudem
Windsor Locks. CT	
BELA'S	Bela Hackman
Memphis. TN	

The bottom of the **Special Paper Options** dialog has two additional options: 2-sided and Borders. If you select the 2-sided option, Panorama will use a special procedure for printing the report on both sides of the paper. First, it will print all the odd pages (1, 3, 5, 7, etc.). Then it stops printing and displays a dialog requesting that you flip the paper over and re-insert the paper into the printer. When you press the **OK** button, Panorama will print the even pages (2, 4, 6, 8, etc.). If the report is printed using multiple subpages, Panorama will adjust the positions as it prints the even pages to make sure that page 2 is on the back of page 1, page 4 on the back of page 3, etc. (Warning: The 2-sided option does not work properly if background spooling is on. You must turn off background spooling before printing a 2-sided report.)

The Borders option makes Panorama print a border around each subpage. If you are not printing on pre-perfed paper, you can use this border to help you cut apart the individual subpages. The border can also be useful to see where the subpages are when you use **Print Preview**.

Chapter 22: Labels



Two of the most common jobs for a database program are printing mailing labels and form letters. Mailing labels can be tricky because the printing must line up with the pre-cut labels. This chapter describes tips and techniques that can help take the “trial and error” out of printing labels and form letters.

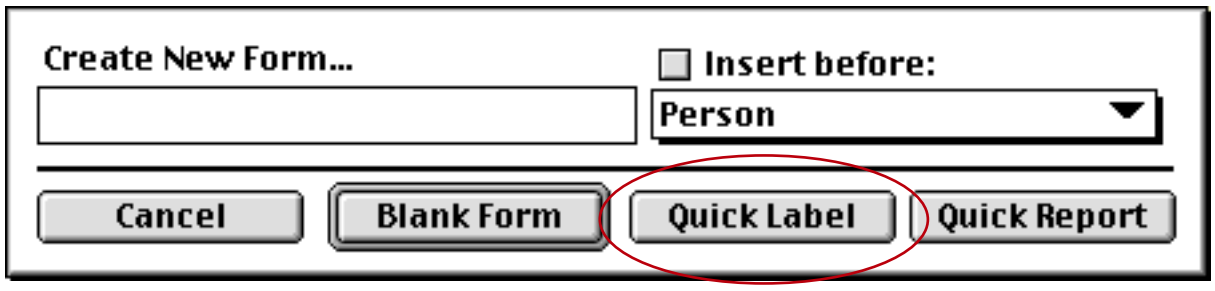
Label Fundamentals

Labels are printed using report tiles in a form (see “Custom Reports” on page 1049 for an introduction to report tiles). Many labels only require a single tile: the data tile (see “Data Tiles” on page 1064). This tile should be the same size as the size of the label plus the gap between labels. In some cases you may also need margin tiles, see “Top Margin Tile” on page 1073 and “Left Margin Tile” on page 1075.

To actually print the names and addresses you will usually use an auto-wrap text object (“Displaying Data in Auto-Wrap Text” on page 595) or a Text Display SuperObject (see “Text Display SuperObjects™” on page 608). Either of these types of objects allow data, text, and punctuation to be mixed in the label.

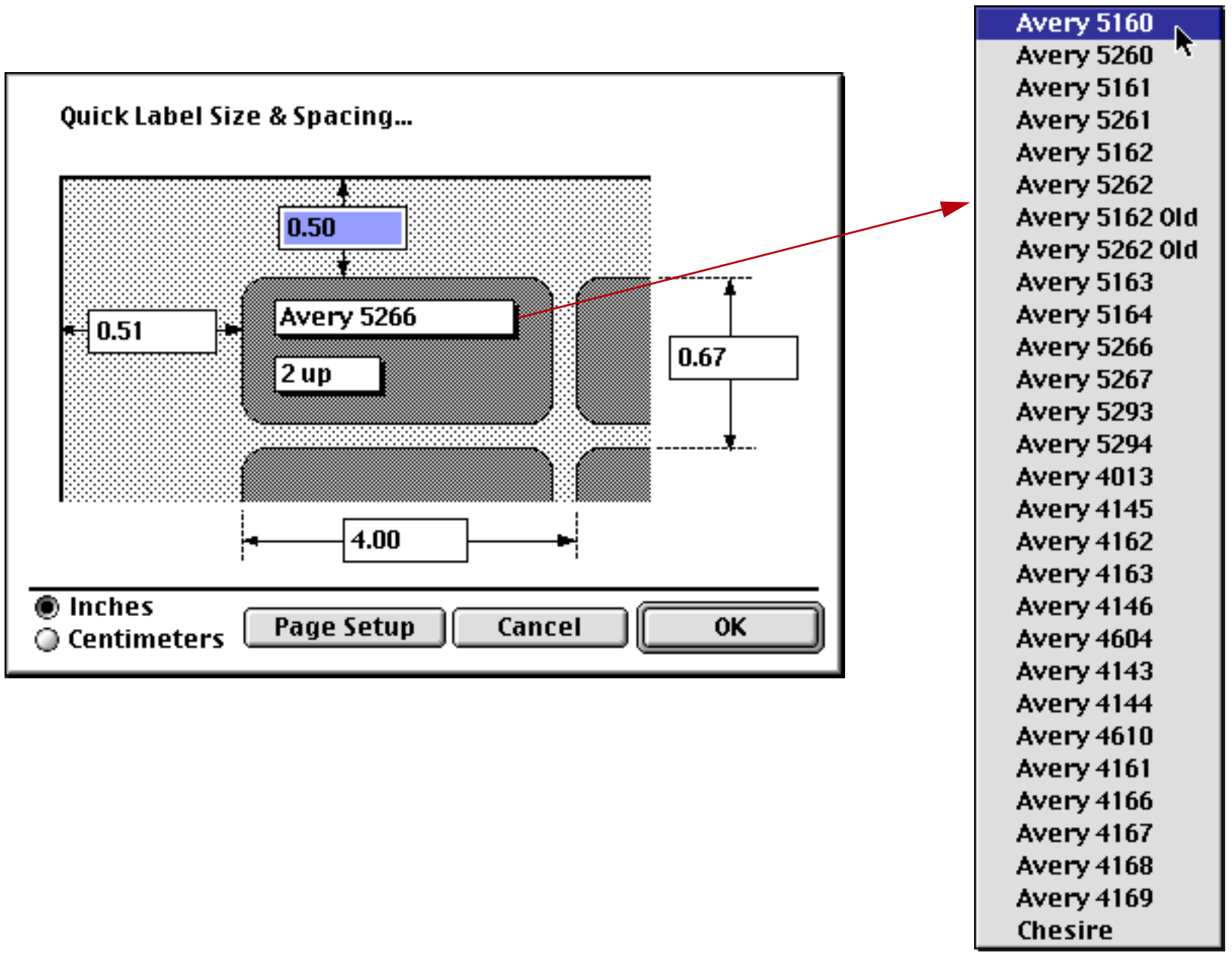
The QuickLabel Dialog

When you create a new form, Panorama gives you the option of creating a blank form or automatically creating a label or report.

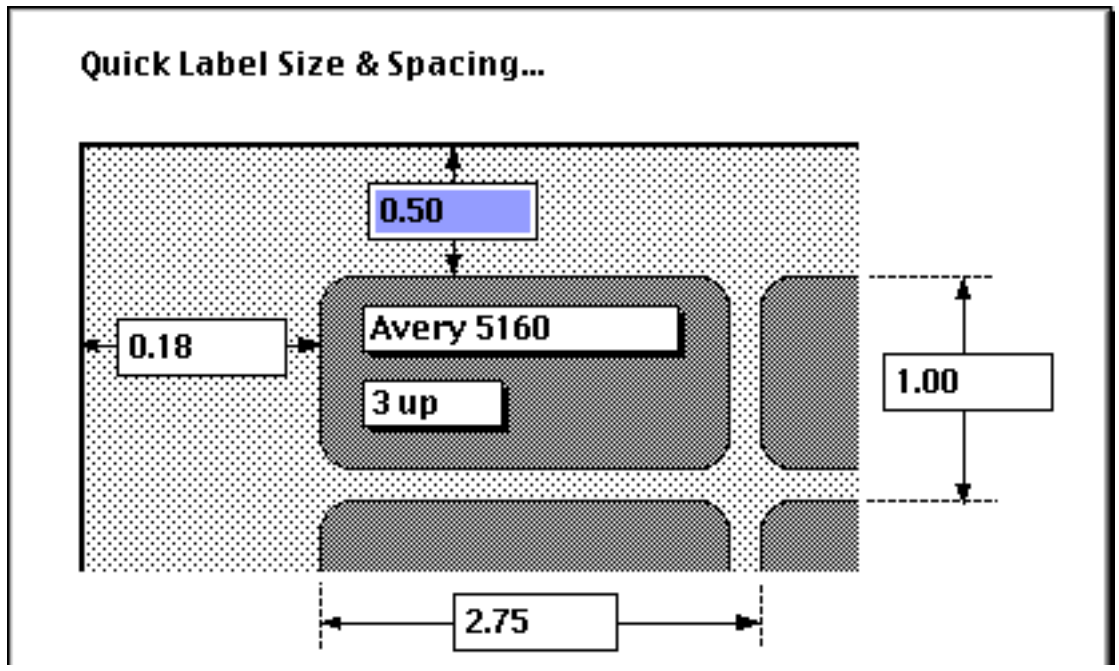


The **QuickLabel** button opens a dialog that can do most or all of the work of setting up a label for you. To use the QuickLabel dialog, select **New Form** from the View Menu (see “Switching Between Views” on page 222), give the new form a name, and then click on the **QuickLabel** button.

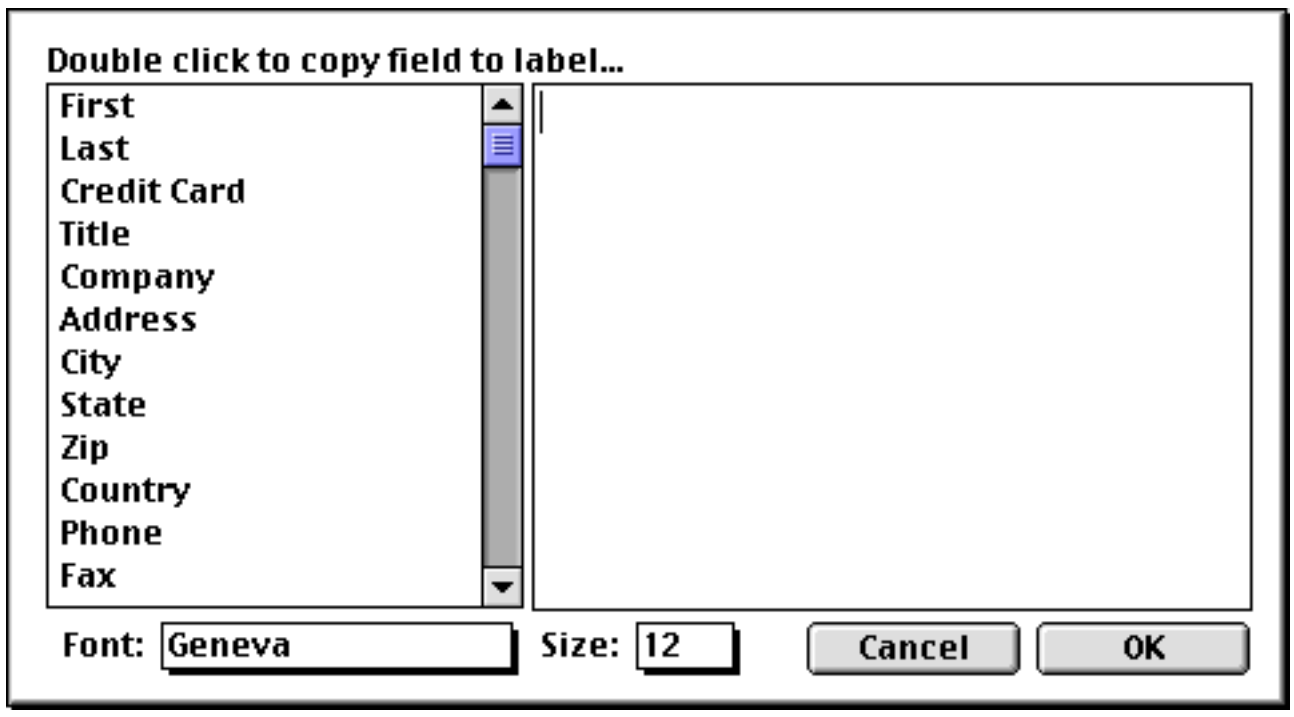
The QuickLabel dialog is really a two part dialog. The first part allows you to define the dimensions of the label. The dialog shows a picture of a label with dimensions around it. You can enter the dimensions manually, or you can pick a pre-defined label size using the pop-up menus inside the label. The top pop-up menu allows you to choose from popular label sizes. The bottom pop-up menu allows you to choose whether the label should be printed 1, 2, 3, or 4 up.



The most popular style of label is Avery 5160, which prints 30 labels on a sheet (10 rows by 3 columns).



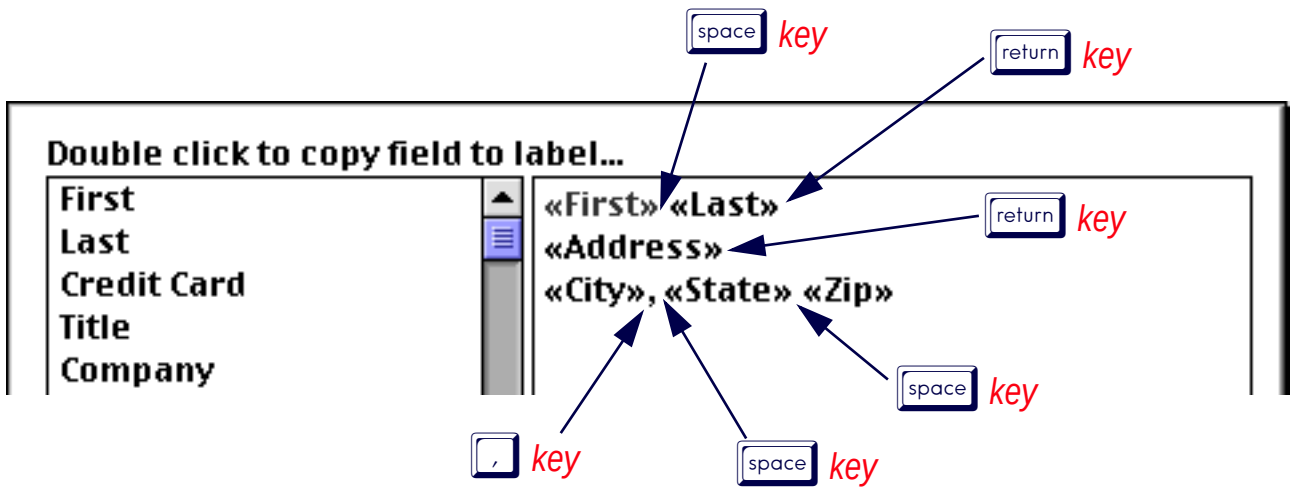
The second part of the dialog allows you to specify the data that will be printed on each label (usually name and address). The left side of the dialog lists all the database fields, while the right hand side contains an image of the actual label.



To copy a field into the label, double click on the field name. Double clicking types the field name into the area on the right, surrounded by « » characters.

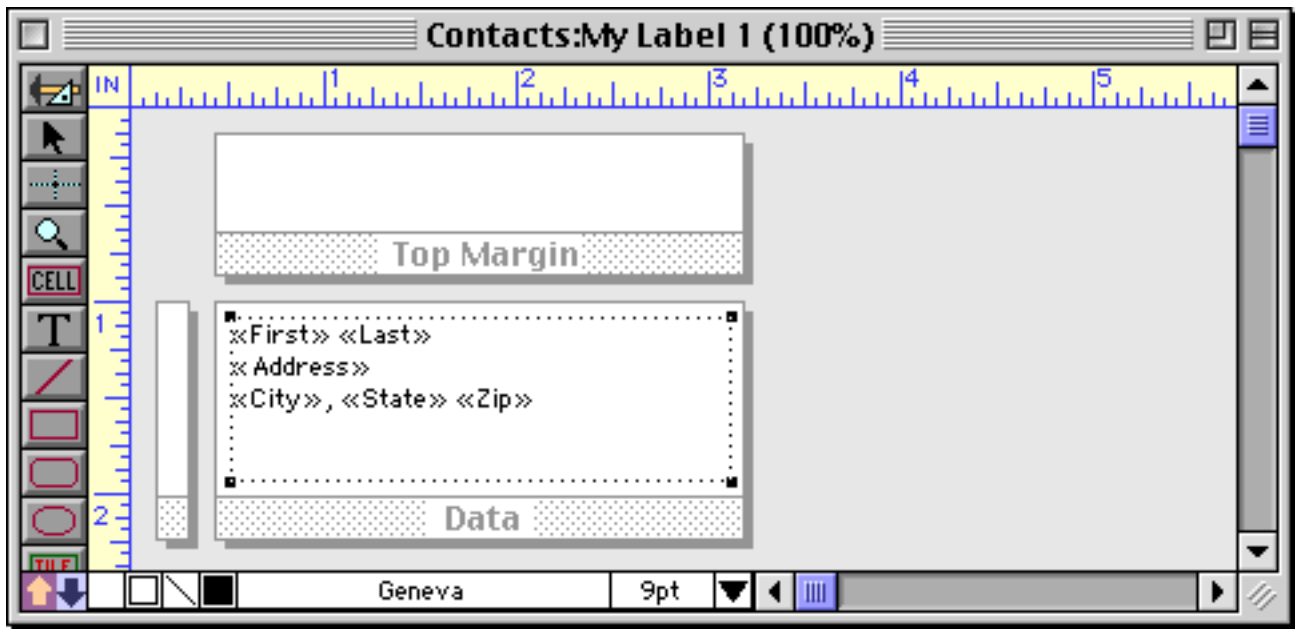


To add a space or punctuation, simply press the appropriate key. To start a new line, press the **Return** key.



You can also edit the text in the label like any other text—click to create an insertion point, drag to select one or more characters, or type to enter new text.

When the content of the label is finished, press the **OK** button. The dialog will automatically create one or more tiles and an auto-wrap text object.



The label is now finished and can be printed or modified further. Here's what the printed labels look like.



Printing Labels on Sheets

Labels come in two styles — sheets and rolls. Most of today's laser and inkjet printers work well with sheets, so we'll discuss those first. To learn about printing roll labels see "Printing Roll Labels" on page 1161.

Some sheets contain labels that go right up to the edge of the sheet. Depending on your printer you may not be able to print all the way to the edge. You may be able to use the **Page Setup** command to make it possible to print closer to the edge of the sheet. Consult the documentation that came with your printer to find out if this is possible.

The height of the data tile should equal the distance from the top of one label to the top of the next label, while the width of the label should equal the distance from the left edge of one label to the left edge of the next label.

Printing 3 by 10 1” Labels (Avery 5160)

The most common type of label sheet contains 30 1” high labels in 3 columns of 10 labels (Avery 5160). The Avery 5160 labels are spaced 1” vertically and 2.75” horizontally. There is a 1⁄2 inch top margin above the labels, and a 0.18 inch left margin.

You can use the QuickLabel dialog to set up the tiles for this type of label (see “The QuickLabel Dialog” on page 1157). If the edges of the labels are cut off when you print, use the Page Setup dialog to adjust your printer settings to allow Panorama to print as close to the edge of the sheet as possible. Consult the documentation that came with your printer to find out how to set the printer margins (if possible).

Aligning Labels on the Sheet

If the names and addresses are not properly aligned on the labels, you can create Left Margin and Top Margin tiles. (The QuickLabel dialog automatically creates these tiles for you.) To shift the text up or down, change the height of the **Top Margin** tile (see “Top Margin Tile” on page 1073). To shift the text left or right change the width of the **Left Margin** tile (see “Left Margin Tile” on page 1075). You can make precision adjustments to the size of a tile with the **Dimensions** dialog (see “Viewing and Setting Exact Object Dimensions” on page 512).

Note: If you do not specify a left or top margin, Panorama will use the default printer margins.

Printer Inaccuracy and Vertical Creep

If you try to print very small labels (less than 1⁄2 inch high), or if you try to print to the very edge of each label (for example a border around the label), you may run into problems due to printer inaccuracy. Due to mechanical tolerances, most printers are only accurate to about 1⁄8 inch over a full 11 inch page. Even if the labels line up properly at the top of the sheet, they may gradually creep out of place towards the bottom of the page. Unfortunately this problem is difficult to correct.

You may be able to reduce or eliminate the creep by adjusting for the inaccuracy in your printer. To try to compensate for this problem you can adjust the height of the data tile in 1/8th pixel increments. To do this, use the **Form Preferences** dialog to reduce the nudge increment to 1⁄8th pixel, then use the up/down arrows to nudge the size of the tile. Remember this adjustment may be different for different printers, or may even change for the same printer at different times.

A simpler solution is to not print anything less than 1⁄8th inch from the edge of a label. These leaves enough tolerance for the inaccuracy of the printer.

Printing Roll Labels

Roll labels can be much more difficult to set up than sheet fed labels, and on some printers you may not be able to get roll labels to print correctly at all.

Before you read this section we have a confession to make. Printers that can print roll labels are very rare these days, and here at ProVUE we have not used such a printer for at least a decade. We also rarely get tech support calls on this topic. The material below was written primarily for the Apple ImageWriter printer, a printer that has not been available for many years, so this material is pretty out of date. However, we decided to include it since we do not have any more recent experience with roll feed labels.

Printing on 1-up 1” Roll Labels

The most common type of roll label contains labels spaced 1 inch from label to label (usually the label itself is 15⁄16 inch, with a 1⁄16 inch gap between the labels). To print 1 inch labels, use the **QuickLabel** dialog to set up the report tiles. The data tile should be exactly 1 inch high, and as wide as the label. Choose 1-up from the pop-up menu. If your printer has an option for turning off gaps between pages use the **Page Setup** dialog to make sure that gaps are disabled. If you printer always prints a gap between labels you won’t be able to print roll labels on that printer.

Printing Non 1” 1-up Labels

There are two basic methods for printing roll labels. The first method is to set up a custom page size and treat each label as a separate page. The second method is to use a normal page size and print several labels per page. For example, you could print eleven 1 inch high labels on a normal 8½ by 11 page. Each of these methods has advantages and disadvantages.

Using Custom Page Size to Print Labels

To print 1-up labels using a custom page size, start by creating a data tile the same width as the label. Make the height of the tile equal to the distance from the top of one label to the top of the next. For instance if the labels are 1 15⁄16 inch high with a 1⁄16 inch gap between labels, make the tile 2 inches high.

Once the data tile is set up choose the **Page Setup** command from the File Menu. Click the Custom Labels button. Custom Labels tells Panorama to automatically set the paper size to the same dimensions as the data tile. Warning: If you ever change the size of the data tile, you must go back to the Page Setup command and re-set the paper size to Custom Labels. Otherwise Panorama will continue to use the old custom label size.

Remember that for most labels you will also have to set the No Gaps Between Pages option to eliminate the top and bottom margins.

You should only use a custom page size if you need to print on an odd size label. Custom page sizes are only available for the Apple ImageWriter printer—they are not available for non-Apple printers.

Using Standard Page Sizes to Print Labels

The most common label size is 1 inch high (1 inch from the top of one label to the top of the next). Since 11 of these labels are an exact fit on a standard 11 inch high paper size, you can avoid the use of custom page size to print these labels. This can help reduce problems with labels peeling off inside the printer that can occur when custom page sizes are used with the No Gaps Between Pages option.

Using a standard page size to print labels is easy—just set up the data tile and print. Remember that the height of the data tile should equal the distance from the top of one label to the top of the next. For 1-up labels the tile should be the same width as the label.

You can print using a standard page size even if the labels don’t fit evenly on the page. For instance, you can get almost nine 1 1⁄4 inch high labels onto a standard page, but not quite. However, if you set the No Gaps Between Pages option Panorama will compensate, automatically printing labels that are split over the page break. The only problem will be that before printing begins, the Macintosh will automatically skip over the first 11 inch page. This wastes labels and throws the labels out of alignment. One solution to this problem is to start with a single blank sheet of paper in the printer. Let the printer skip over this page, then feed in the roll of labels and continue printing. Of course another solution is to use a custom page size. This only wastes one label but runs an increased risk of label jamming.

2, 3, and 4-Up Roll Labels

Panorama can print multiple column labels, but we don’t recommend doing this with adhesive backed labels in an ImageWriter printer. As each label is printed the printer platen will rock back and forth. We’ve found that with multiple column adhesive labels this almost always results in labels peeling off inside the printer. At a minimum you should watch the printer carefully during printing to prevent a minor problem from turning into a catastrophe. You can also reduce the problem quite a bit by using Best or Faster print quality instead of Draft.

You can print multiple column labels simply by using the **Report Preferences** dialog to select the number of columns (from 1 to 4). If you are using the Custom Labels page size you should set up the number of columns before you use Page Setup to set the page size to Custom Labels.

4-Up Cheshire Labels

To print 4-up Cheshire labels, use the QuickLabel dialog and select 4-up Cheshire from the pop-up menu. Once the form is set up, use the **Report Preferences** dialog to set the Across option, so the labels will print across the page instead of down. Don't forget to use the Page Setup dialog to set the No Gaps Between Pages option. When you print the labels, we recommend using Best or Faster print quality instead of Draft.

Selecting Font and Print Quality

When you are printing labels using an ImageWriter, the choice of font and print quality can have a major impact on print speed. For maximum speed, use Draft quality with Monaco 10 point text. It looks funny on the screen, but prints several times faster than any other font because it matches the ImageWriter's built in font.

Of course if the label contains any graphics, it must be printed using Best or Faster quality. Using Best or Faster quality can also reduce the amount of forward/back platen motion when printing multiple column labels.

Chapter 23: Formulas



The result we proceed to divide, as you see,
by Nine Hundred and Ninety Two:
Then subtract seventeen, and the answer must be
Exactly and perfectly true.

- Lewis Carrol, The Hunting of the Snark

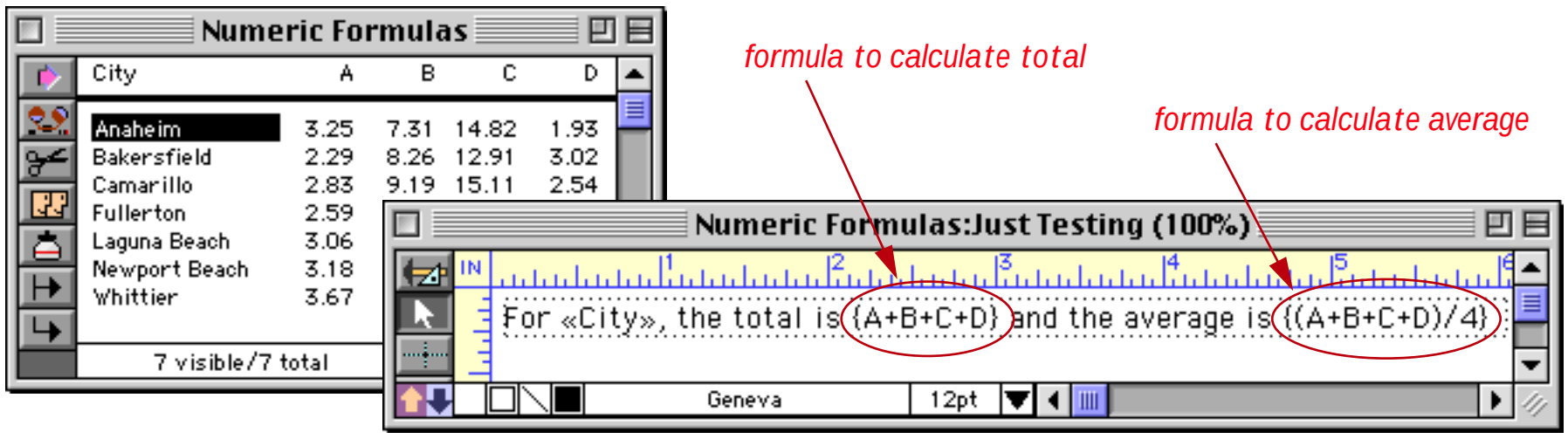
Panorama's primary job is storing and retrieving data. The primary job of formulas is to combine and manipulate data, both numeric and textual. Using formulas Panorama can automatically add up all the items in an invoice and calculate the sales tax. Using formulas Panorama can automatically divide all the names in a database into separate first and last names, or convert all the company names in a database to upper case. Using formulas Panorama can automatically look up the price of an item in inventory, or check the quantity on hand, or look up and display the items on a customers previous order. As you can see, you'll need to learn how to use formulas to get the most from your Panorama investment. Fortunately, formulas are easy to learn and use (especially the most common mathematical formulas like totals, taxes and percentages). (However, we have to admit that sometimes formulas can be frustrating because they are very picky. If you get one little detail wrong, the formula won't work correctly. This isn't just a problem with Panorama, but with virtually any computer program that uses formulas. To help ease the potential frustration factor Panorama has some wizards you'll learn about later in this chapter that can help you build error free formulas.)

Formulas In Action

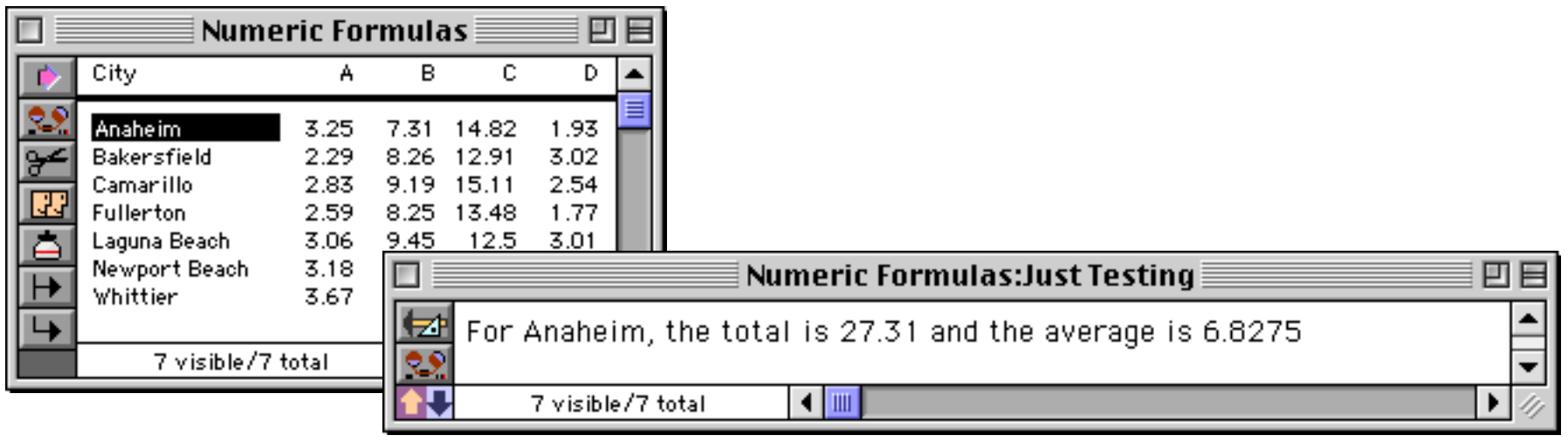
Formulas are a general purpose tool that Panorama can use in a variety of different situations. You can display or print the result of a formula, use a formula to modify the database, or use a formula to help locate information in the database. The next few sections demonstrate each of these techniques.

Displaying/Printing A Formula

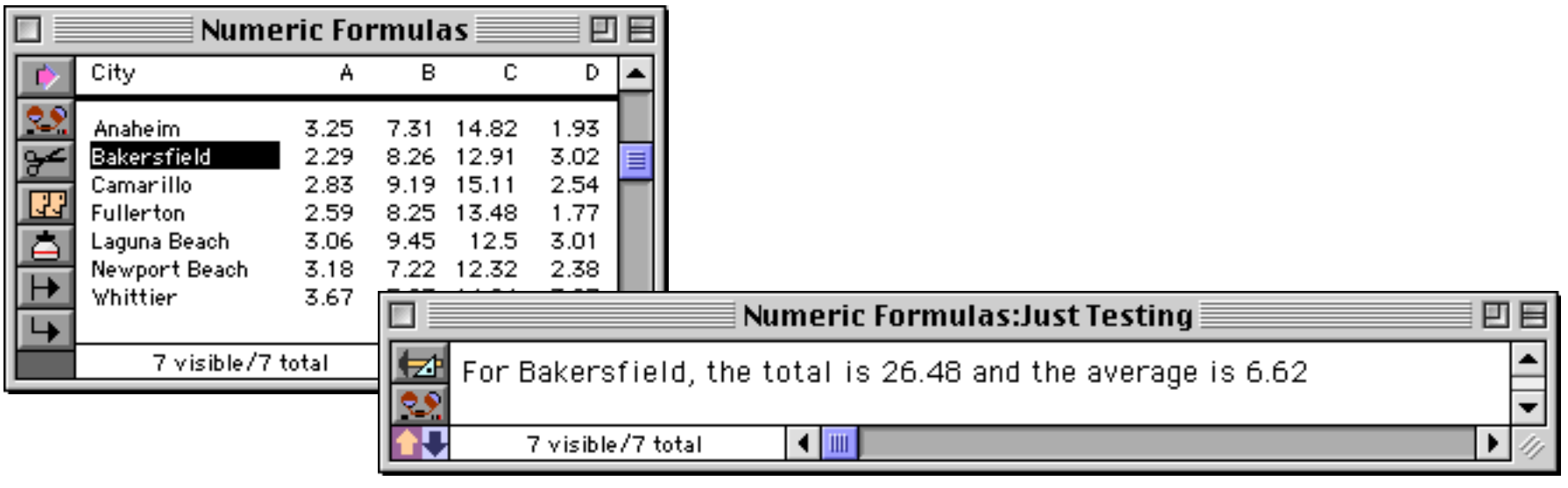
A formula can be displayed or printed anywhere on a form with an auto-wrap text object (see "Displaying Formulas in Auto-Wrap Text" on page 602) or Text Display SuperObject (see "Text Display SuperObjects™" on page 608). For example, consider the database shown below. The auto-wrap text object contains two formulas, one which calculates the total of the four columns (A, B, C and D) and one which calculates the average.



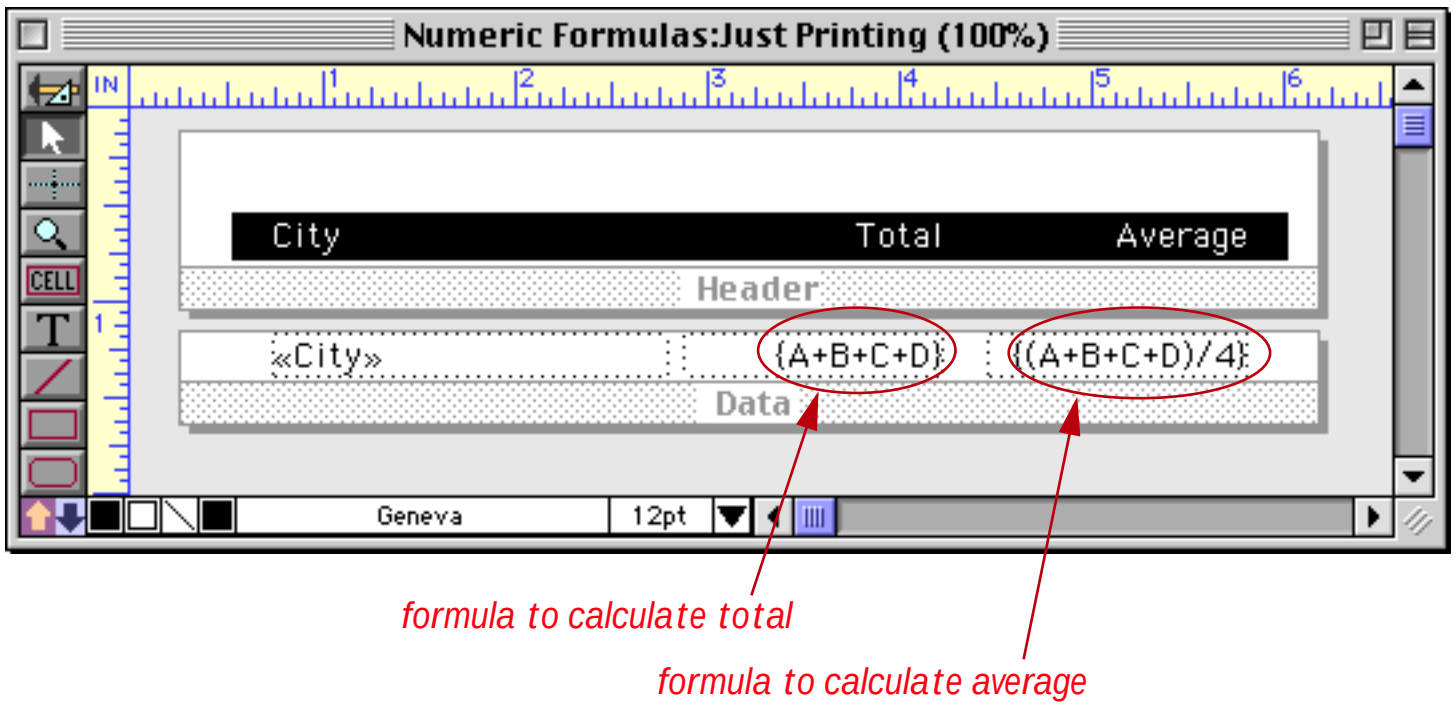
When the form is switched to Data Access Mode, Panorama calculates the formula results and displays them.



When a formula is used this way the results are not stored anywhere in the database, they are simply calculated on the fly and displayed or printed, then thrown away. If you switch to a different record Panorama will calculate and display the new values.



You can even print a report using formulas calculated on the fly. (See "Custom Reports" on page 1049 to learn more about creating a custom report like this.)



Once again, the formula results are calculated on the fly as the report is printed, then discarded. Here is the finished report.

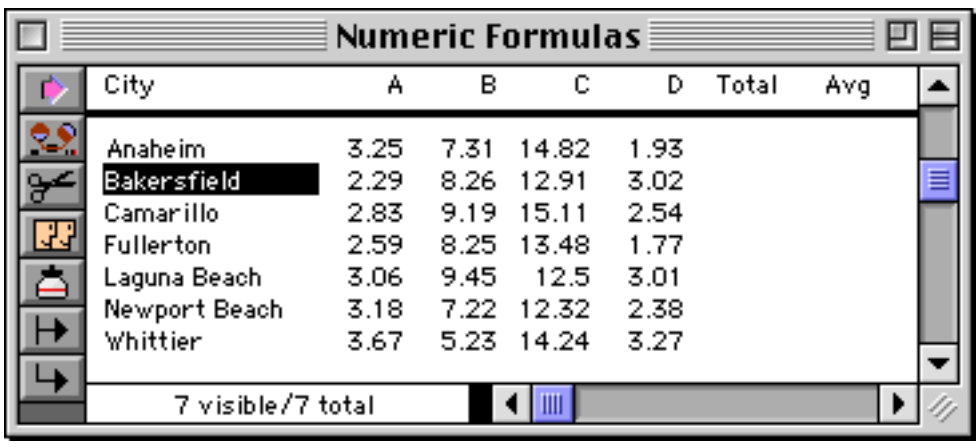
City	Total	Average
Anaheim	27.31	6.8275
Bakersfield	26.48	6.62
Camarillo	29.67	7.4175
Fullerton	26.09	6.5225
Laguna Beach	28.02	7.005
Newport Beach	25.1	6.275
Whittier	26.41	6.6025

(You may notice that the columns in the report above don't line up because they don't all have the same number of places after the decimal point. You can fix this with the `pattern(` function, see "Converting Between Numbers and Strings" on page 1227).

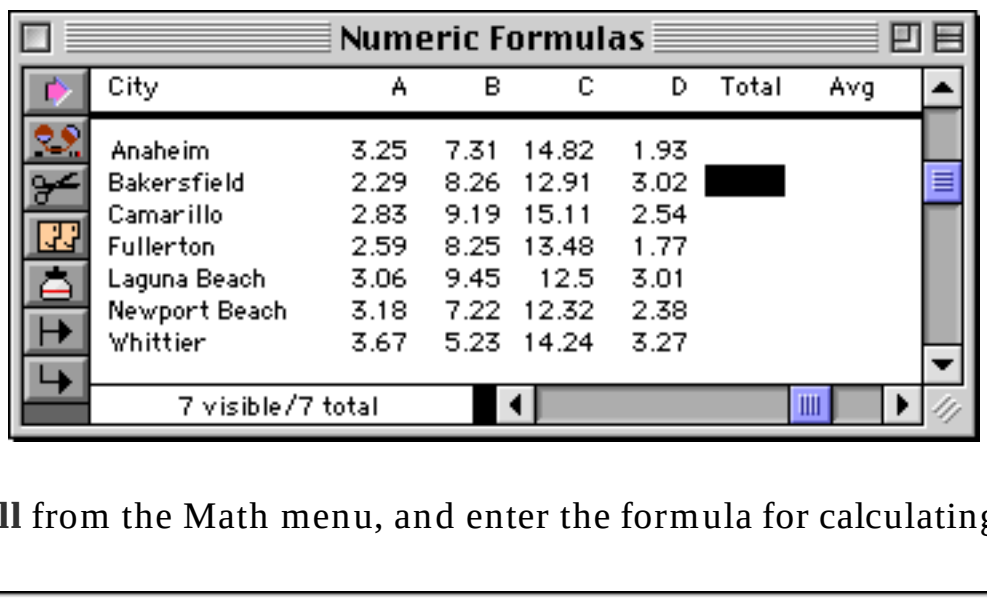
Panorama's Flash Art feature allows a formula result to be displayed visually. See "Flash Art™" on page 748.

Storing Formula Results in the Database

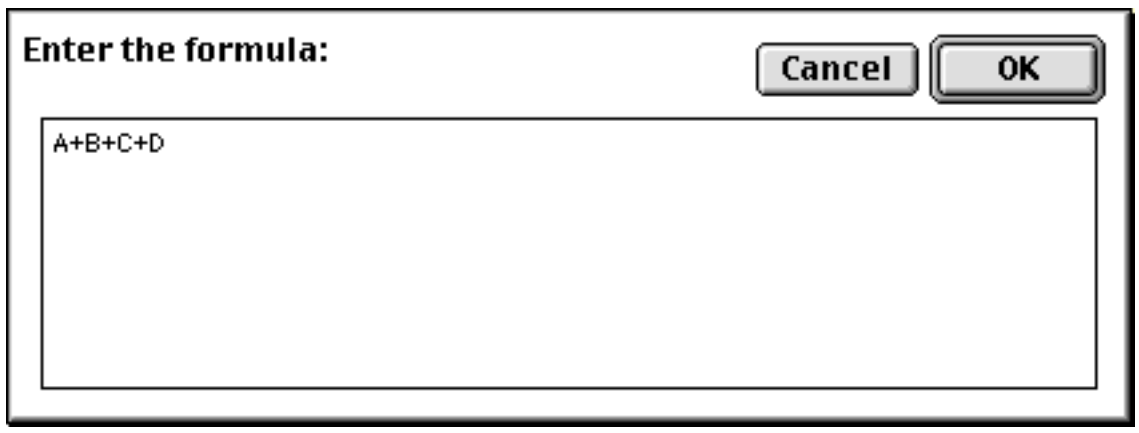
Sometimes you'll want to actually store the result of a formula in the database itself. You can do this manually after data has already been entered, or automatically as data is entered or changed. To illustrate these two techniques we'll add two new columns to the example database used in the last section, Total and Avg.



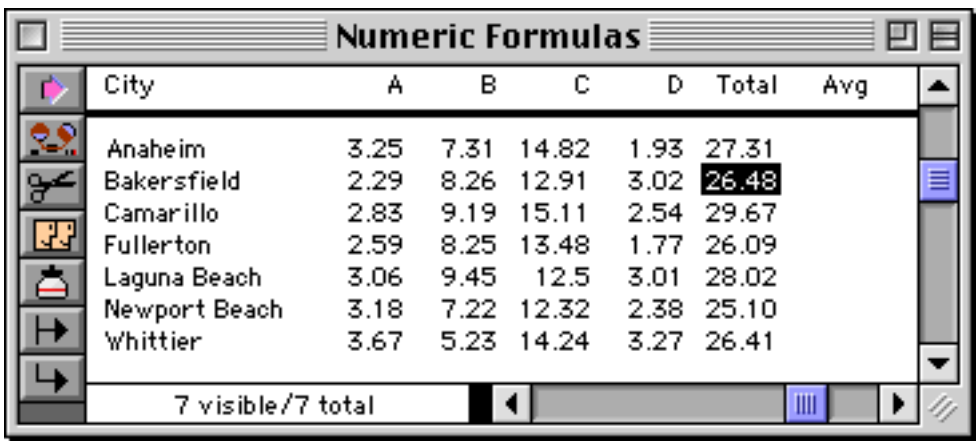
To calculate the values for these new fields we need to use the **Formula Fill** command (see "Filling a Field with a Formula" on page 453). To calculate the total, first click anywhere in the appropriate column.



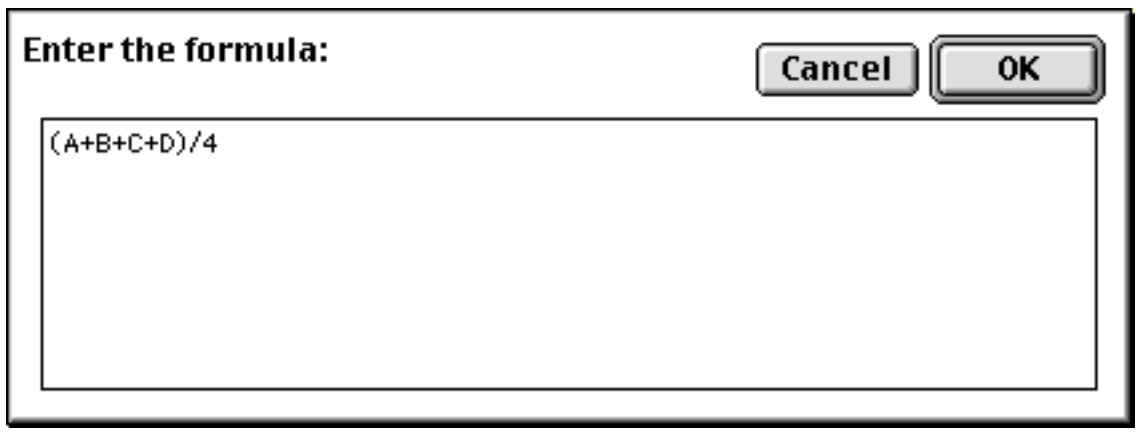
Now choose **Formula Fill** from the Math menu, and enter the formula for calculating the total.



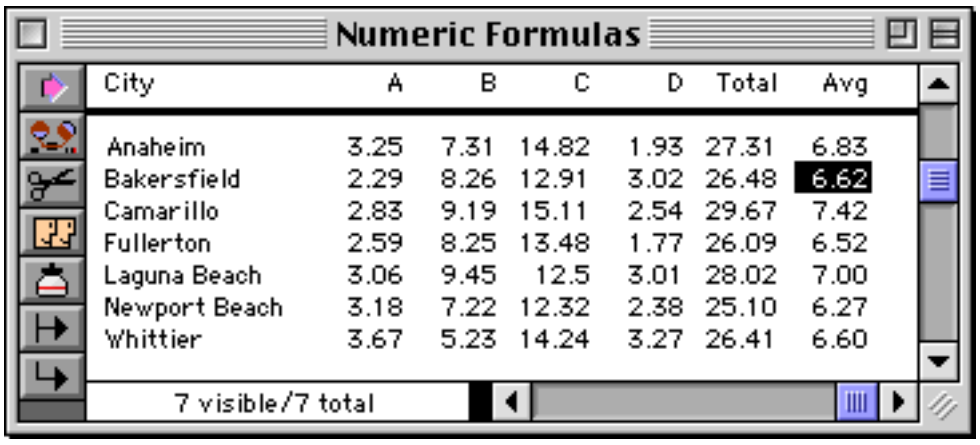
When you press **OK** Panorama will calculate and store the value for every selected record in the database. In this case it performs seven calculations and stores seven values.



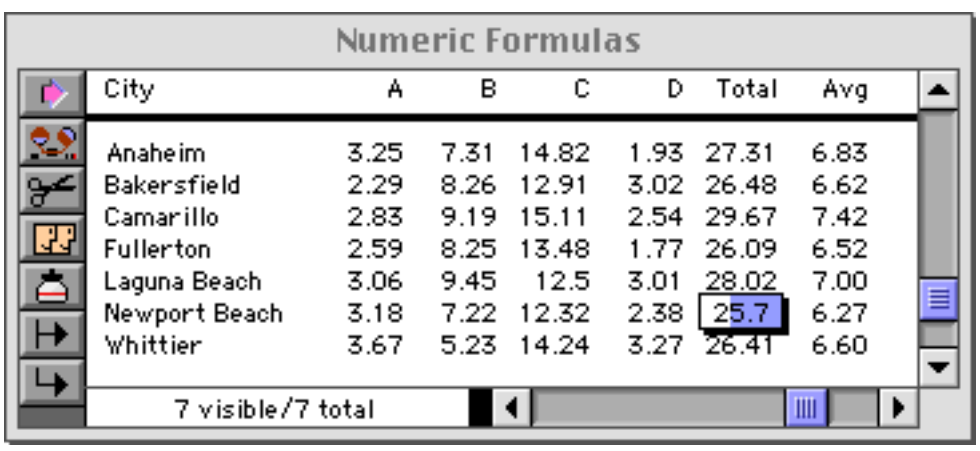
Repeat the same process for the average, but of course with a different formula.



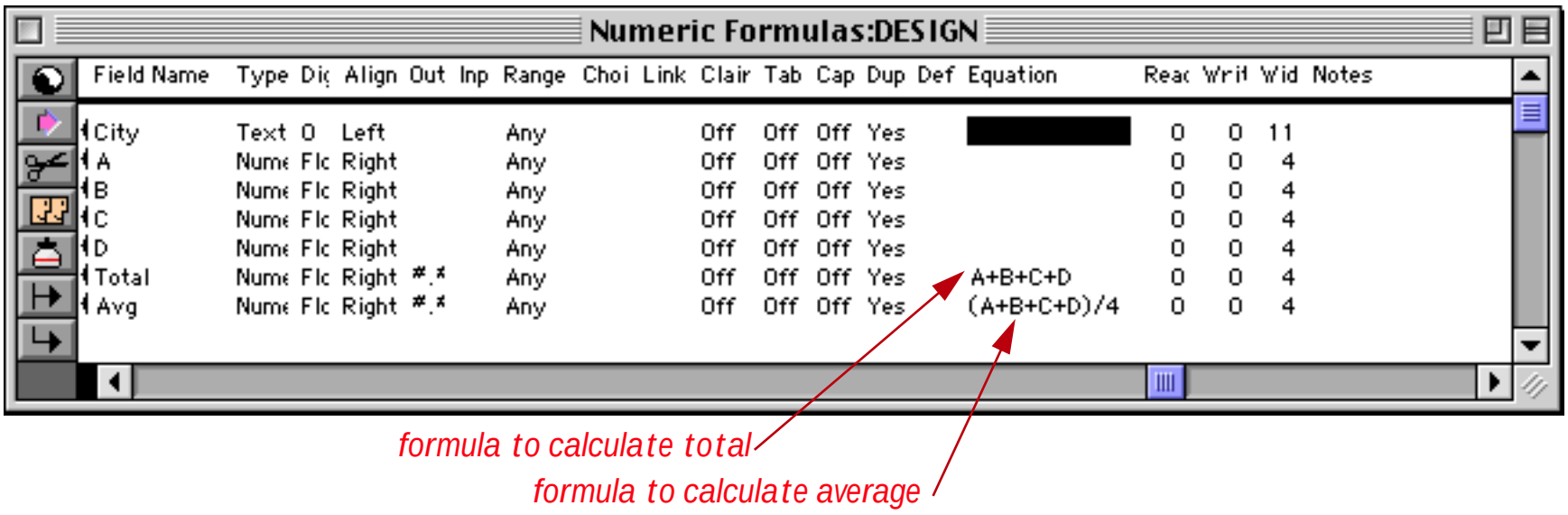
Here's the end result.



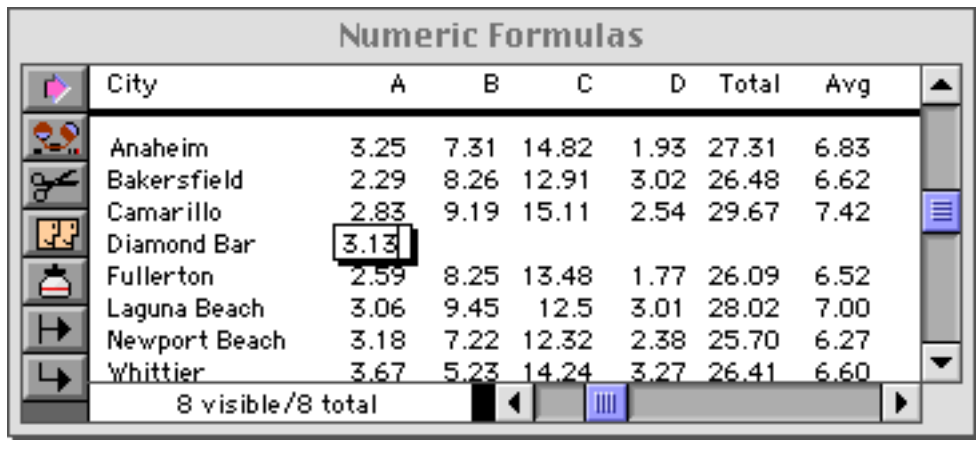
Once the **Formula Fill** calculation is finished the formula is forgotten and the numbers are simply stored in the database just like a number that has been typed in. You can even manually edit a value to override the result of the formula calculation.



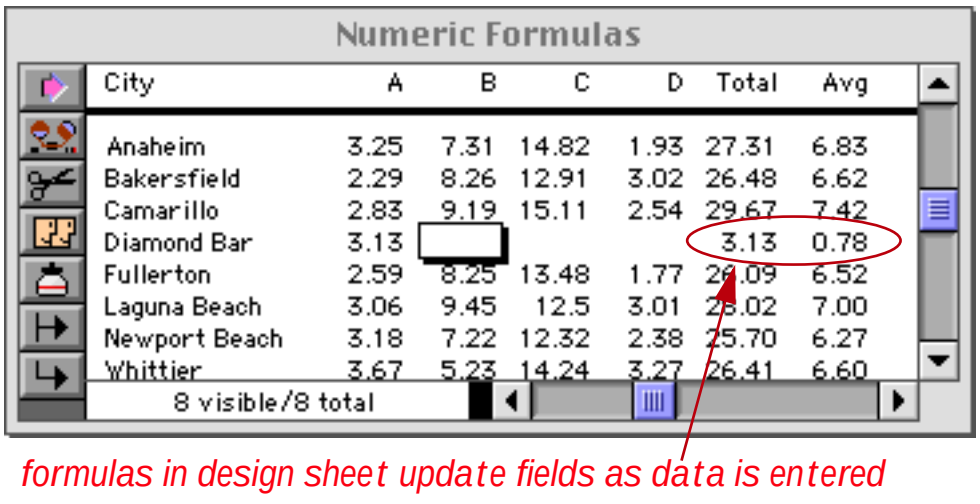
Since the **Formula Fill** formula is forgotten as soon as it is complete, Panorama does not update the values if the original numbers (in this case A, B, C or D) change or if new records are added to the database. If you want values stored in the database to update automatically as the database is updated you must set up automatic calculations in the design sheet (see "Automatic Calculations" on page 340). Here's the design sheet for our example updated to automatically calculate the total and average.



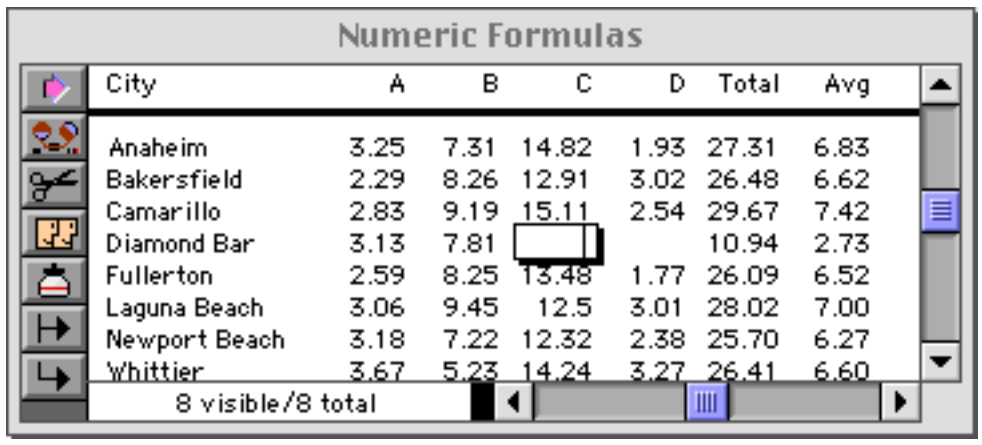
To activate these formulas you need to create a new generation for this database (see "Database "Generations"" on page 252). Once you've done this you can start entering or updating information. In this illustration a new record has been added (Diamond Bar) and the first number typed in (but not entered into the database yet).



As soon as the data is entered by pressing the **Tab** (or **Enter**) keys the formulas update the Total and Avg fields.



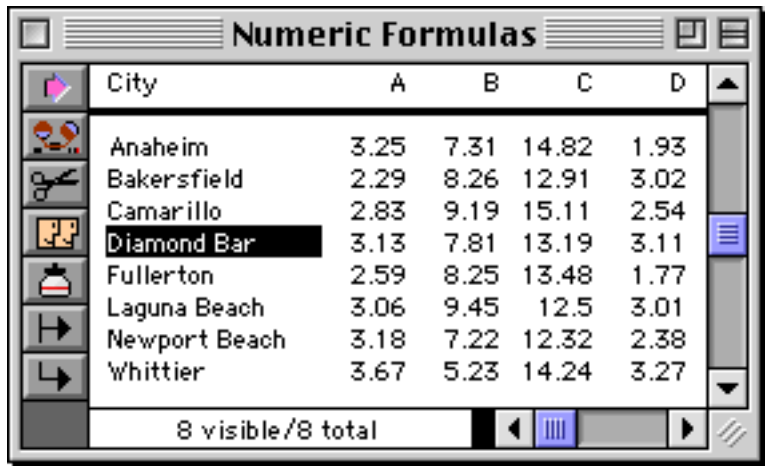
As more data is entered the Total and Avg fields are updated instantaneously.



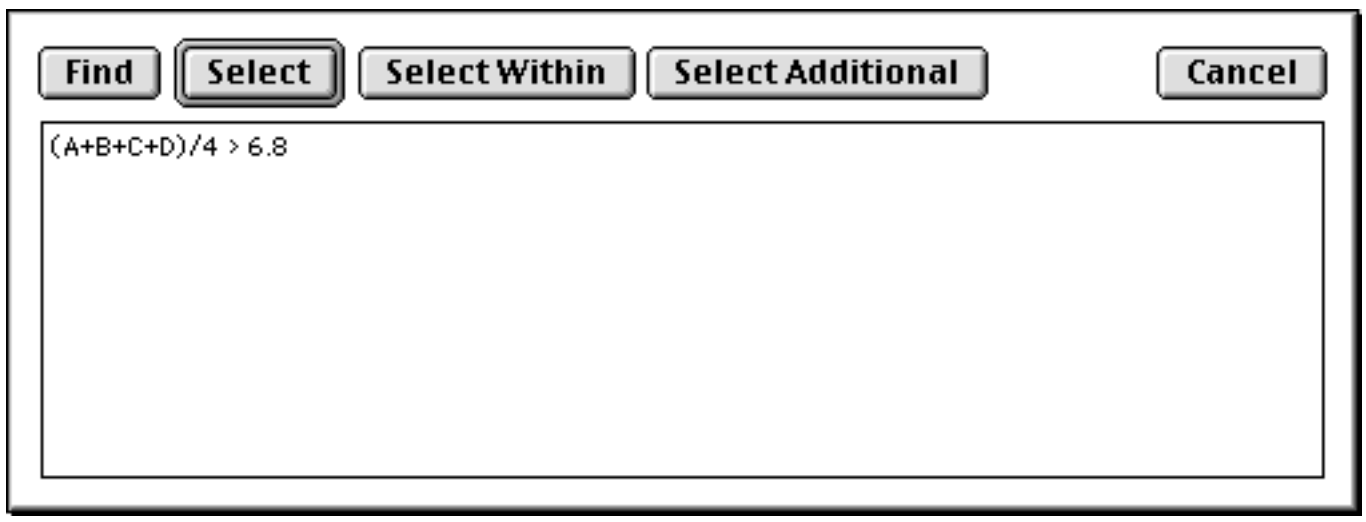
The Total and Avg fields will be updated any time the A, B, C or D fields are modified.

Using a Formula to Locate/Select Information

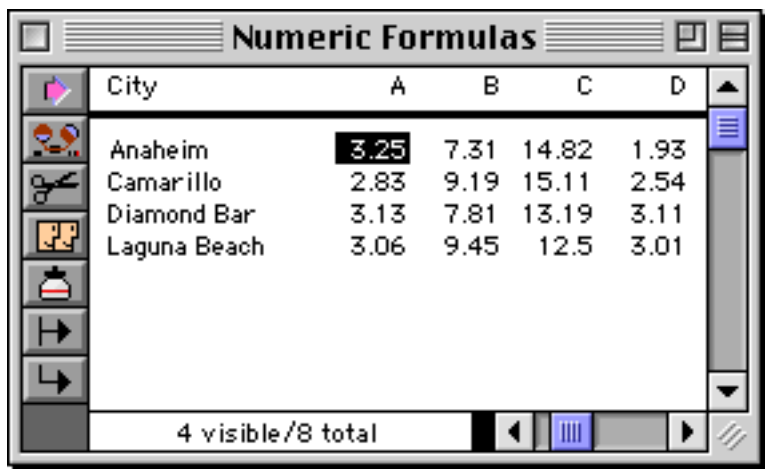
The **Formula Find/Select** command (see "Formula Find / Select" on page 387) allows you to select data based on a formula. This allows you to make selections on data that is not directly stored in the database. For example, suppose you want to select all records with an average greater than 6.8, but without actually storing the average in the database. Here's the database.



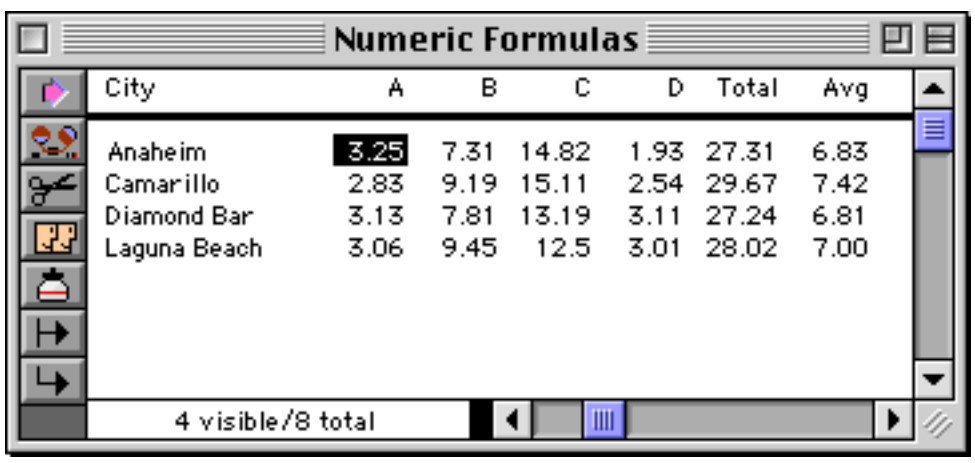
Choose the **Formula Find/Select** command and enter the formula. This formula calculates the average and then compares the average to 6.8.



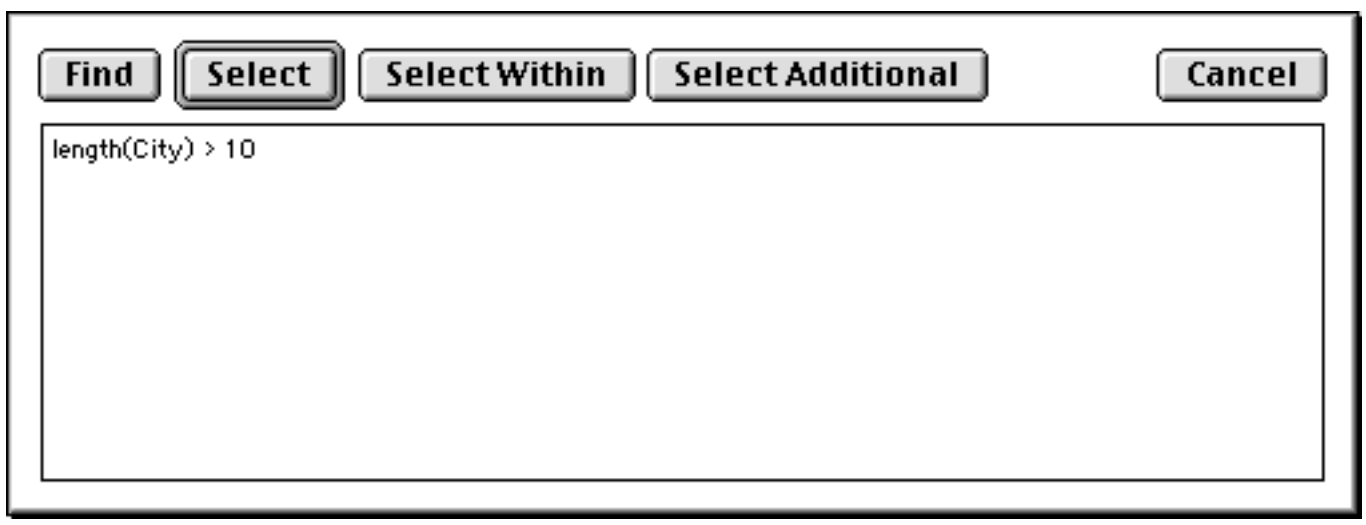
When the **Select** button is pressed records with averages above the threshold are selected.



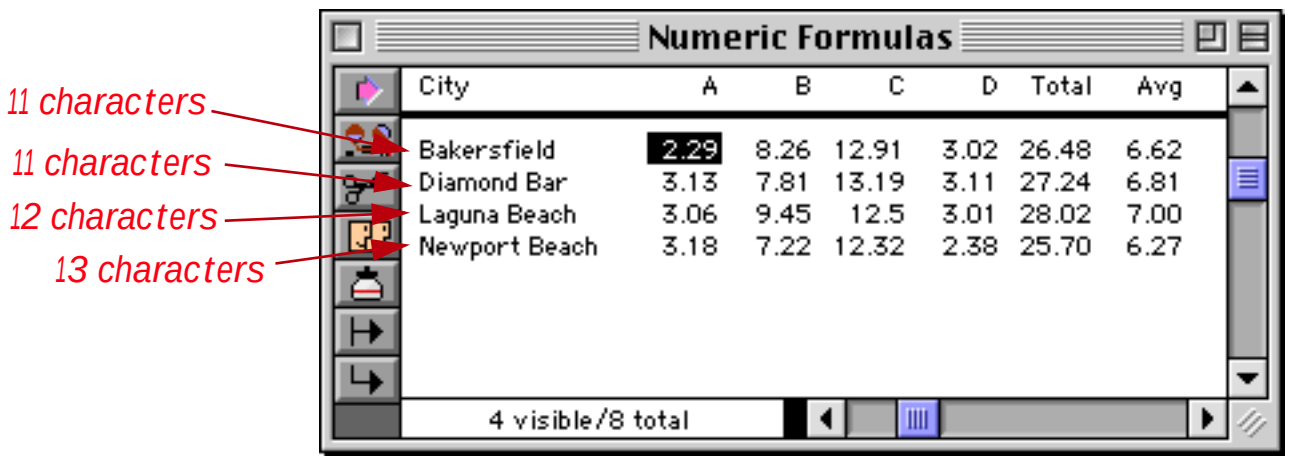
Hey - I cheated (sort of)! This database already has the averages stored in the database. I can expand the window to double check that I really selected only records with averages over 6.8.



Do you forgive me? Anyway, the point is that the selection can be made even if the average is not stored in the database. Here's another example. This formula will select every record where the city name is longer than 10 characters (11, 12, 13, etc.)



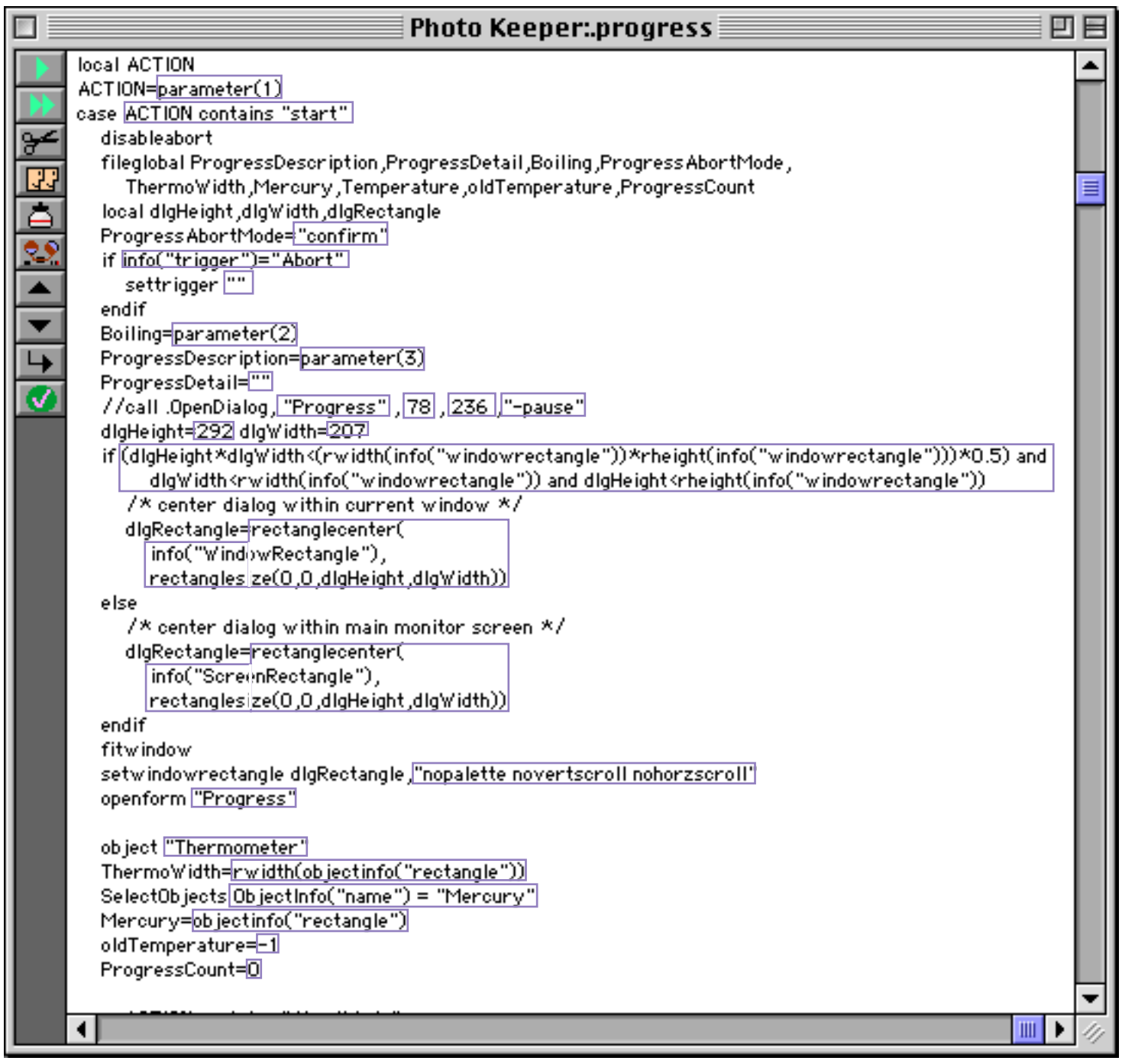
Press **Select** to see only the records with long city names.



With a bit of ingenuity you can almost come up with a formula to locate or select even the most obscure information.

Formulas in Procedures

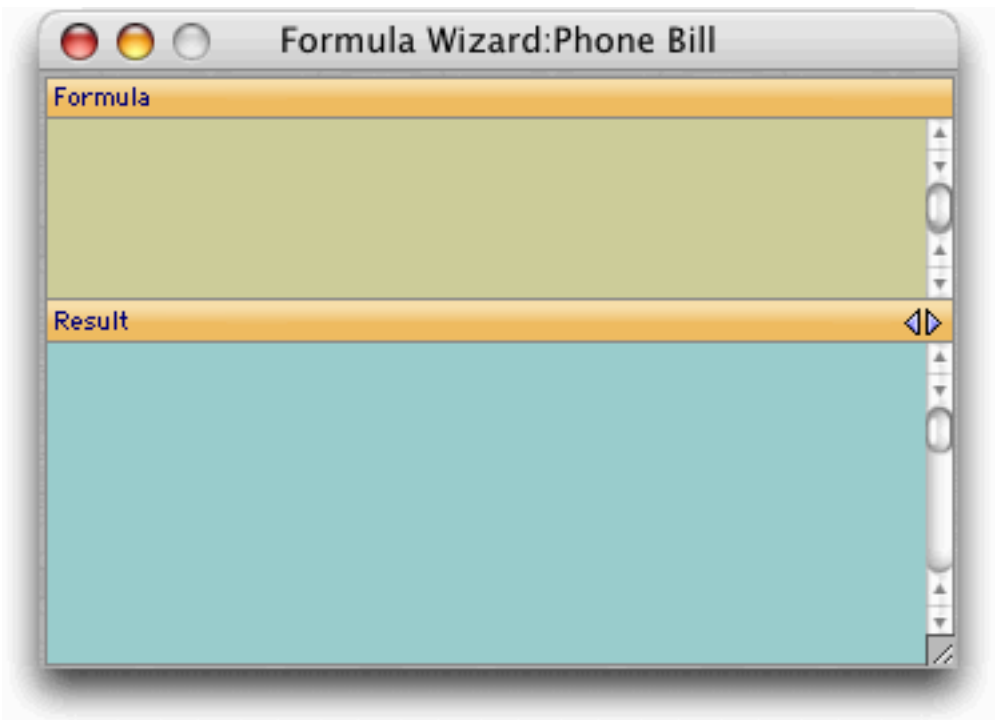
Within a procedure formulas are used to calculate values and control program flow. Most procedures contain many formulas — a typical example is shown below. The formulas in this procedure (there are 25 visible in this window) have been highlighted with a light blue box. Don't worry, I don't expect you to understand this procedure right now — the point is to show how vital formulas are to the operation of almost any procedure, and to show the wide variety of formulas possible, from very simple like a single number or text item to a complicated multi-line formula.



To learn more about creating procedures see "Procedures" on page 1339.

Using the Formula Wizard

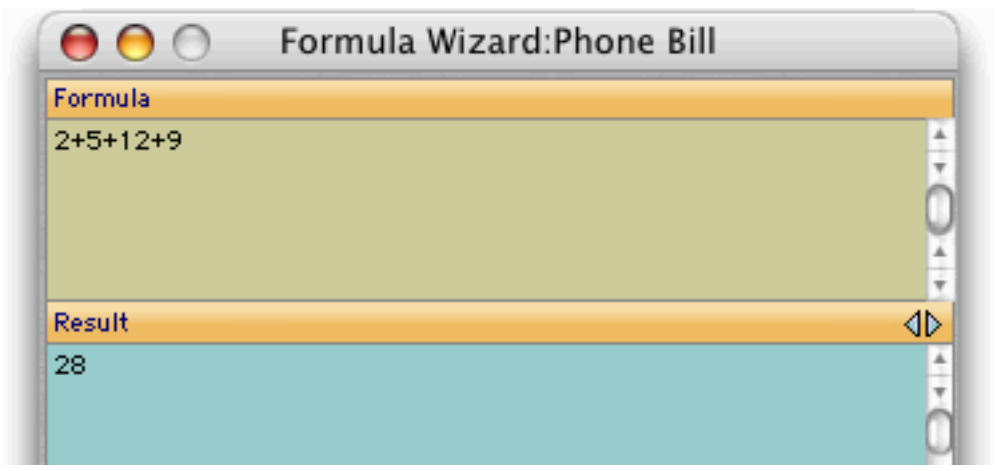
Panorama includes a **Formula Wizard** that you can use as a "sandbox" for playing with formulas. The **Formula Wizard** let's you play around with formulas and see the results immediately. It's great for trying out a new function or technique you are not familiar with or to work the bugs out of a formula before actually incorporating it into your database. To use this wizard start with any database and select Formula Wizard from the Calculation submenu in the **Wizard** menu. (Tip: Unless you've changed this setting with the **Hotkey** wizard, you can also open the Formula Wizard by pressing **Control-1** on Macintosh systems.)



To use the **Formula Wizard** type a formula into the top section.



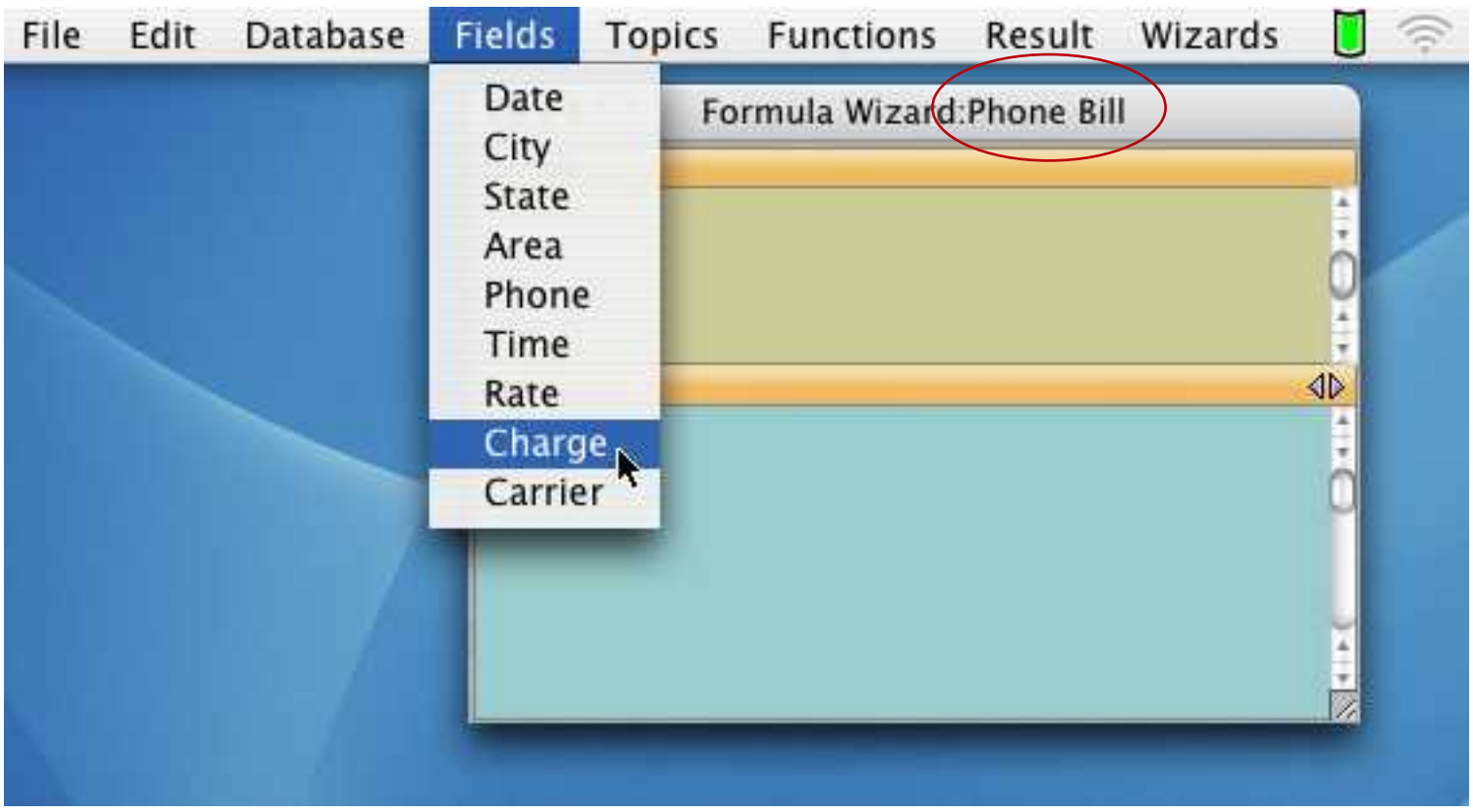
When you press the **Enter** key the result appears in the bottom section (see "Arithmetic Formulas" on page 1205).



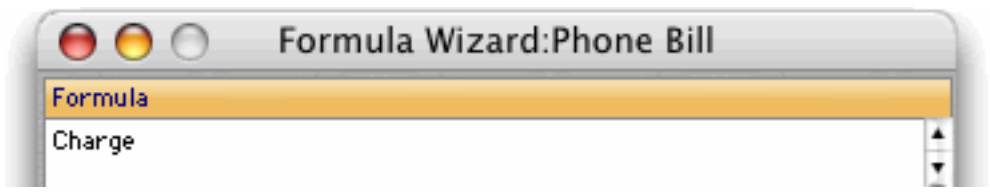
As you can see, the **Formula Wizard** can be used as a handy calculator.

Calculations with Database Fields

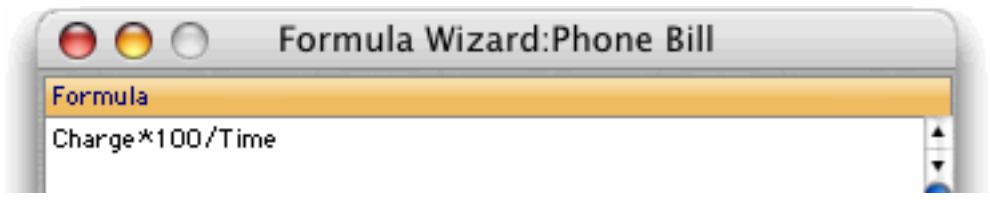
The **Formula Wizard** can include values in database fields as part of the calculation. The name of the database that is currently linked with the wizard is shown in the window's title bar, in this case Phone Bill. If you forget a field name you can see a list of all the fields in the **Fields** menu.



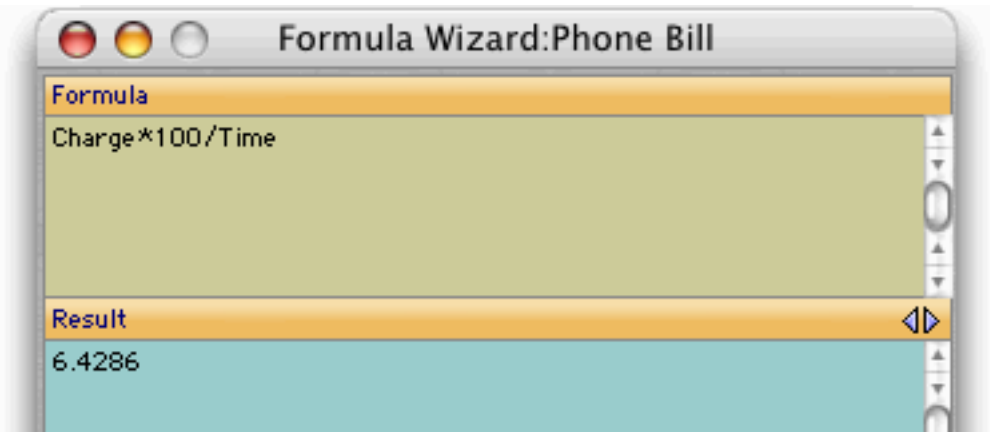
When you select a field from this menu the wizard will automatically type it in for you.



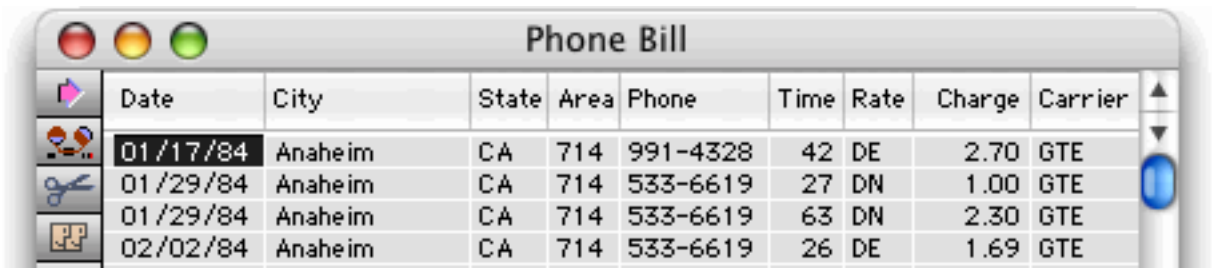
You can included as many fields as you want in the formula. However, they normally must all be from the active database, in this case Phone Bill.



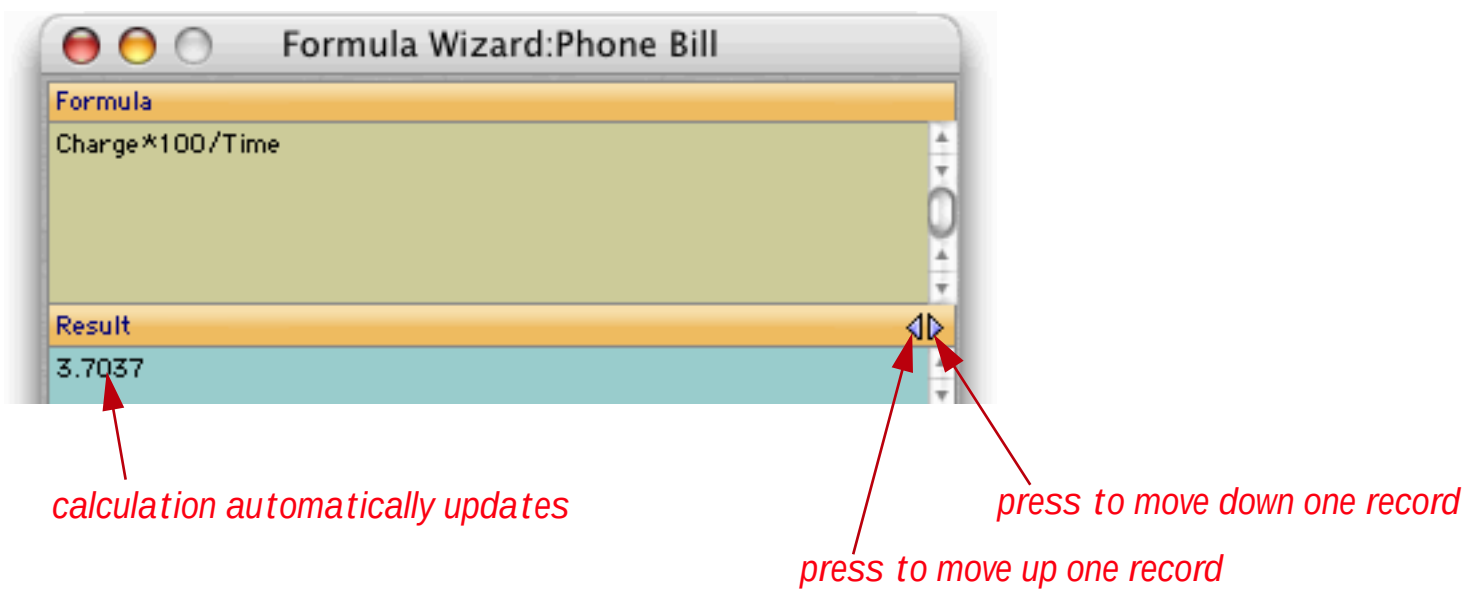
Once again when you press the **Enter** key the wizard will calculate the result, in this case the cost of the phone call in cents per minute.



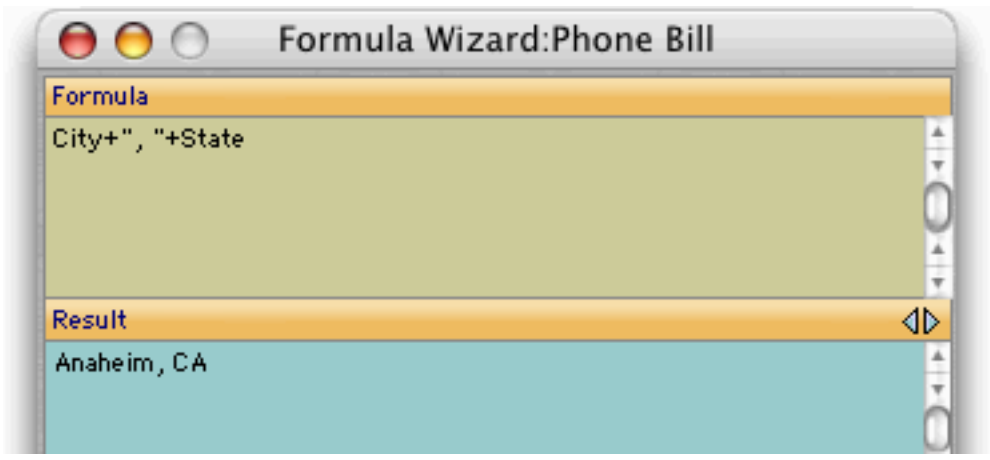
When the wizard performs the calculation it does so based on the currently active record in the active database. In this example the calculation was based on a charge of 2.70 for a 42 minute phone call.



You can use the two small triangles to try out the calculation for other records in the active database. As you press the triangles the active record will move up and down.



A formula can also work with text fields (see “Text Formulas” on page 1212). This formula glues the city and state together with a comma in between (see “Gluing Strings Together” on page 1212).

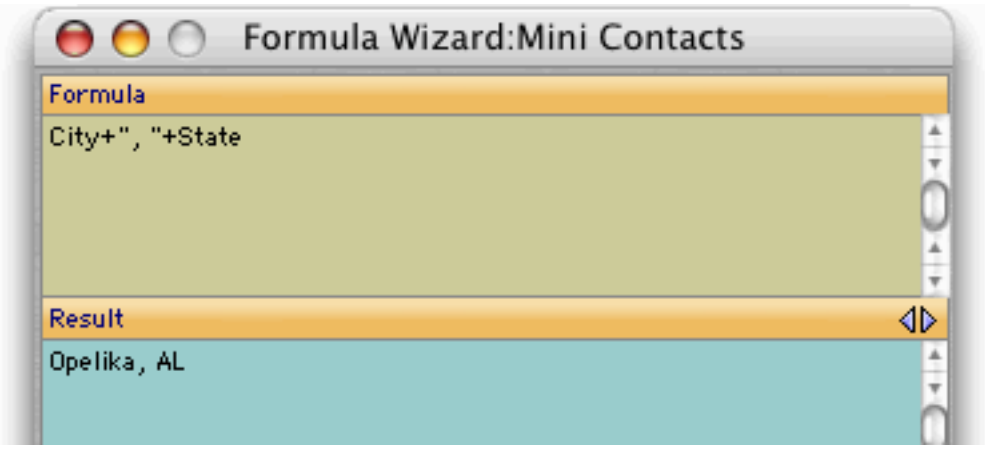


Changing the Active Database

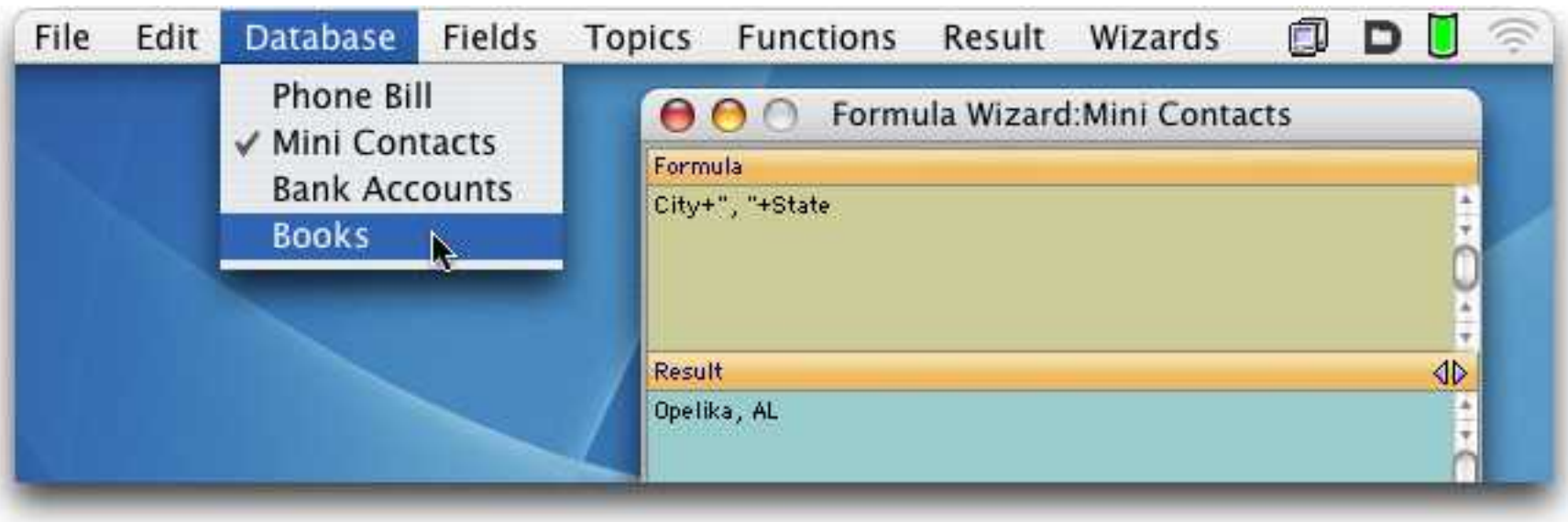
You can change the database that is used by the **Formula Wizard** at any time with the **Database** menu. This menu lists all of the currently open databases.



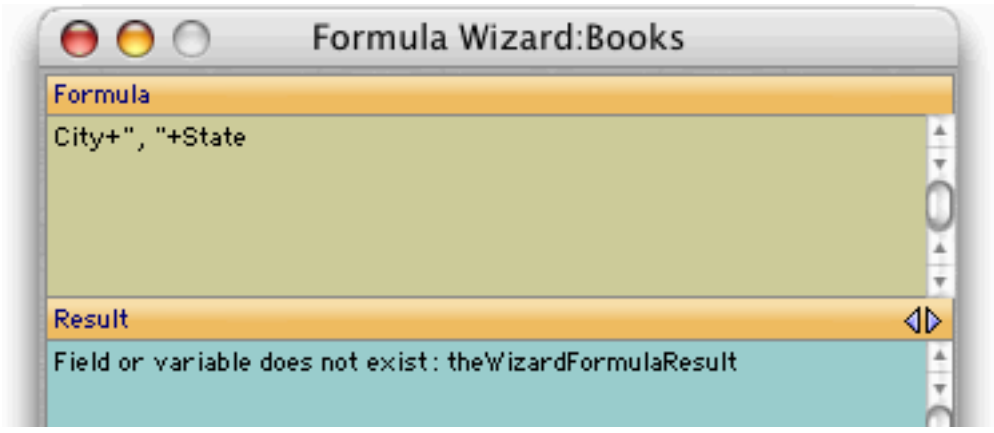
Now when we re-evaluate the same formula it comes up with a different result (Opelika, AL).



If you switch to a database that does not contain one or more of the fields used in the formula…



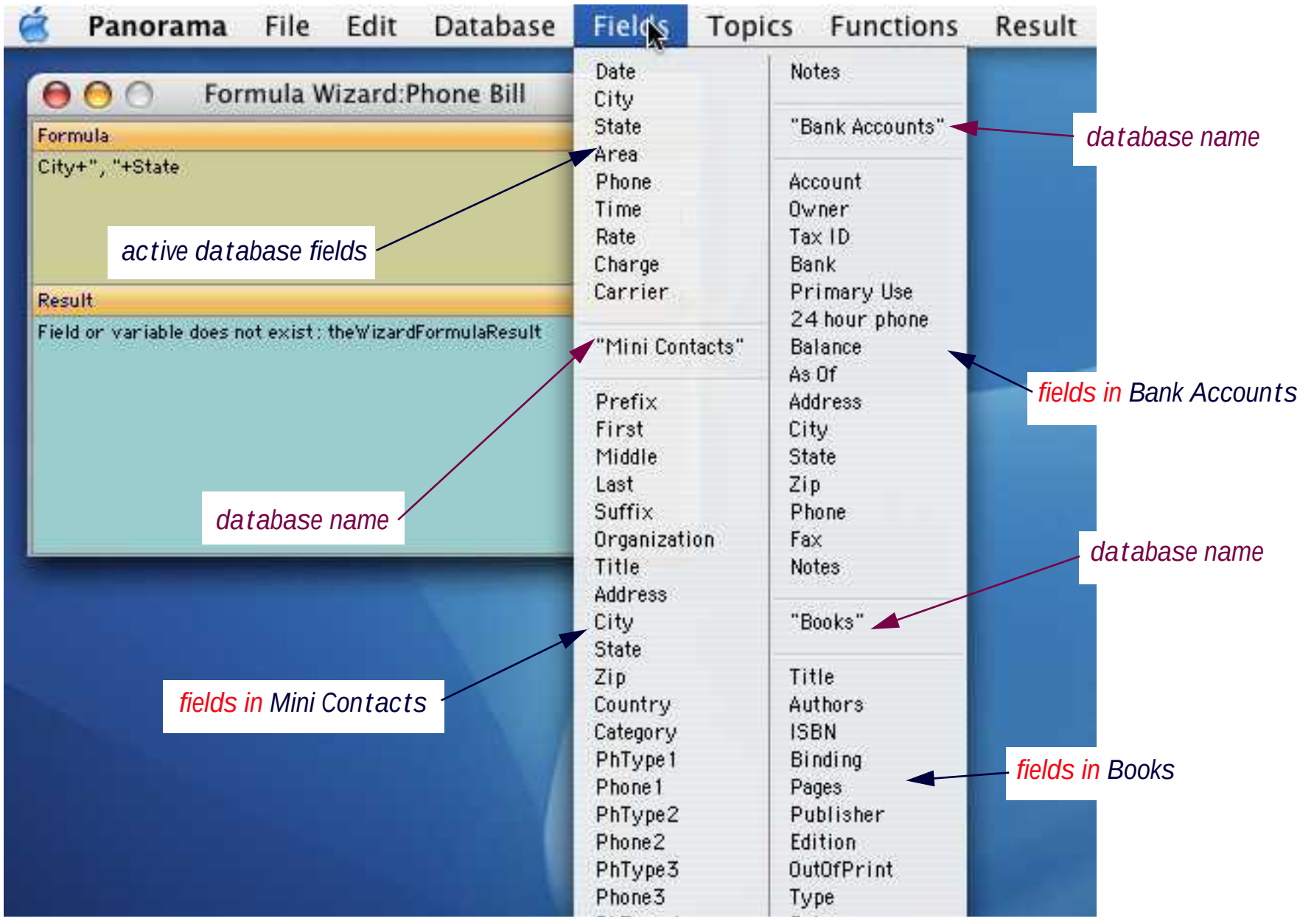
the result will be an error.



Another way to set the active database is to click on one of the database's windows and then choose **Formula Wizard** from the Calculations submenu of the Wizard menu. This brings the wizard back to the front and makes the selected database active.

Using Fields from Other Databases

Some functions (for example `lookup(` and `grabdata(`, see "Linking With Another Database" on page 1272) use fields from other open databases, not just the active database. The **Fields** menu can help you type in these field names. When more than one database is open the **Fields** menu will list the fields in all of them. The fields for the active database (in this case Phone Bill) are listed first, followed by the name of each database (in quotes) and the fields for each database.



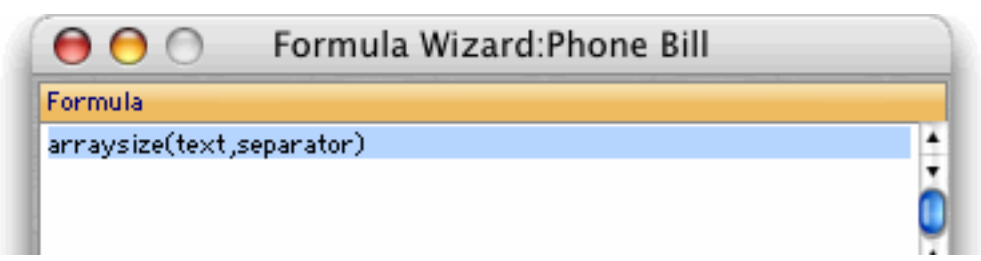
To type in a database name or field simply select it from the menu.

Topic and Functions Help Menus

Panorama contains hundreds of function and operators available for your use. Even here at ProVUE Development we can't remember all of them, so we have several methods for finding a particular function. The **Topics** menu organizes functions into submenus by topic.



When you release the mouse the function or operator will be typed into the formula.

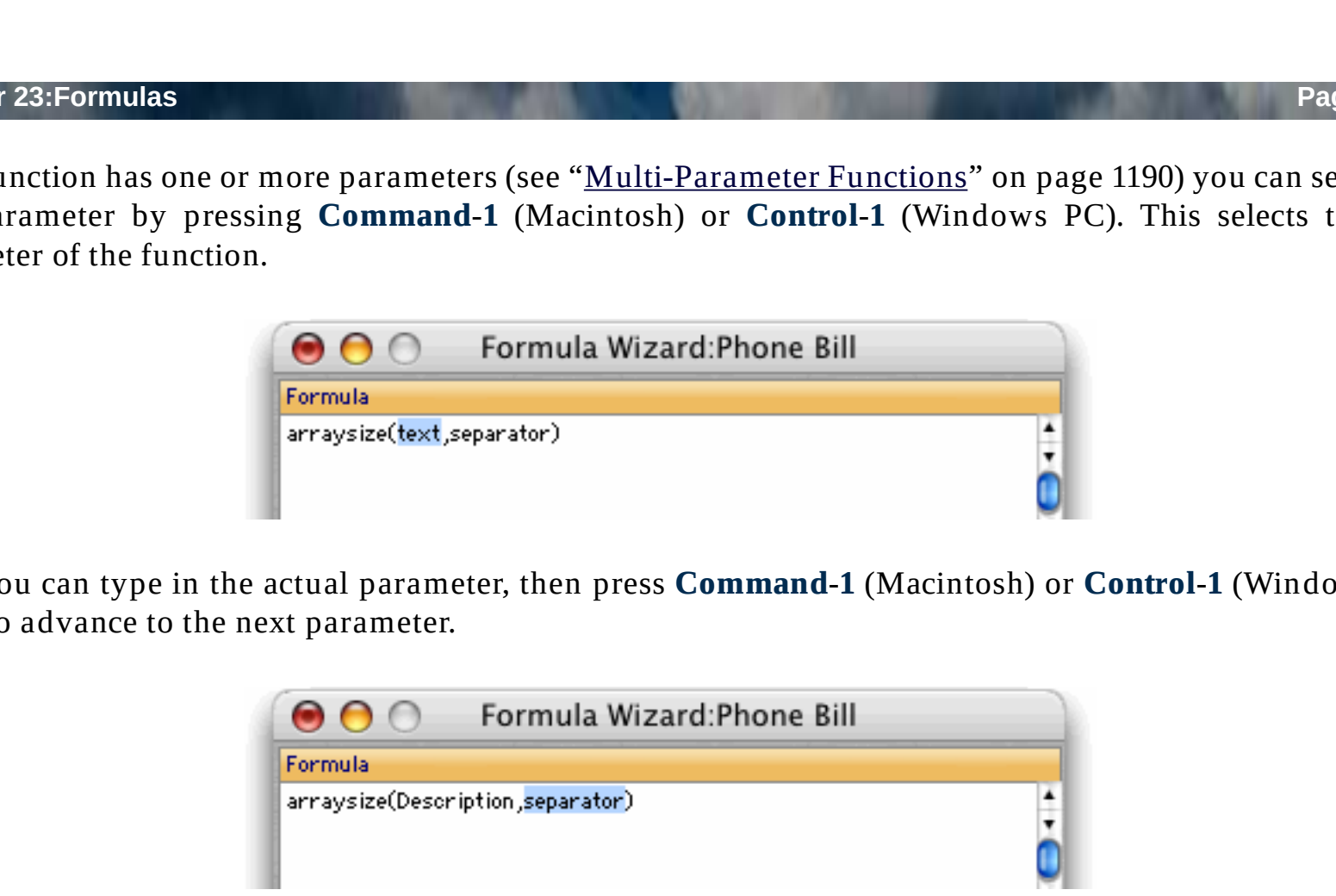


If the function has one or more parameters (see "Multi-Parameter Functions" on page 1190) you can select the first parameter by pressing **Command-1** (Macintosh) or **Control-1** (Windows PC). This selects the first parameter of the function.

Now you can type in the actual parameter, then press **Command-1** (Macintosh) or **Control-1** (Windows PC) again to advance to the next parameter.

If this function had additional parameters you could continue to advance until all of the parameters were complete.

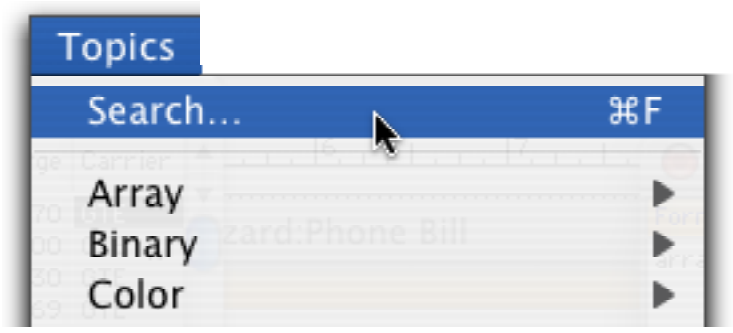
The **Topics** menu lists most, but not all of Panorama's functions. To see an alphabetical list of every available function use the Functions menu. As you can see, this is a giant menu that will fill your entire screen.



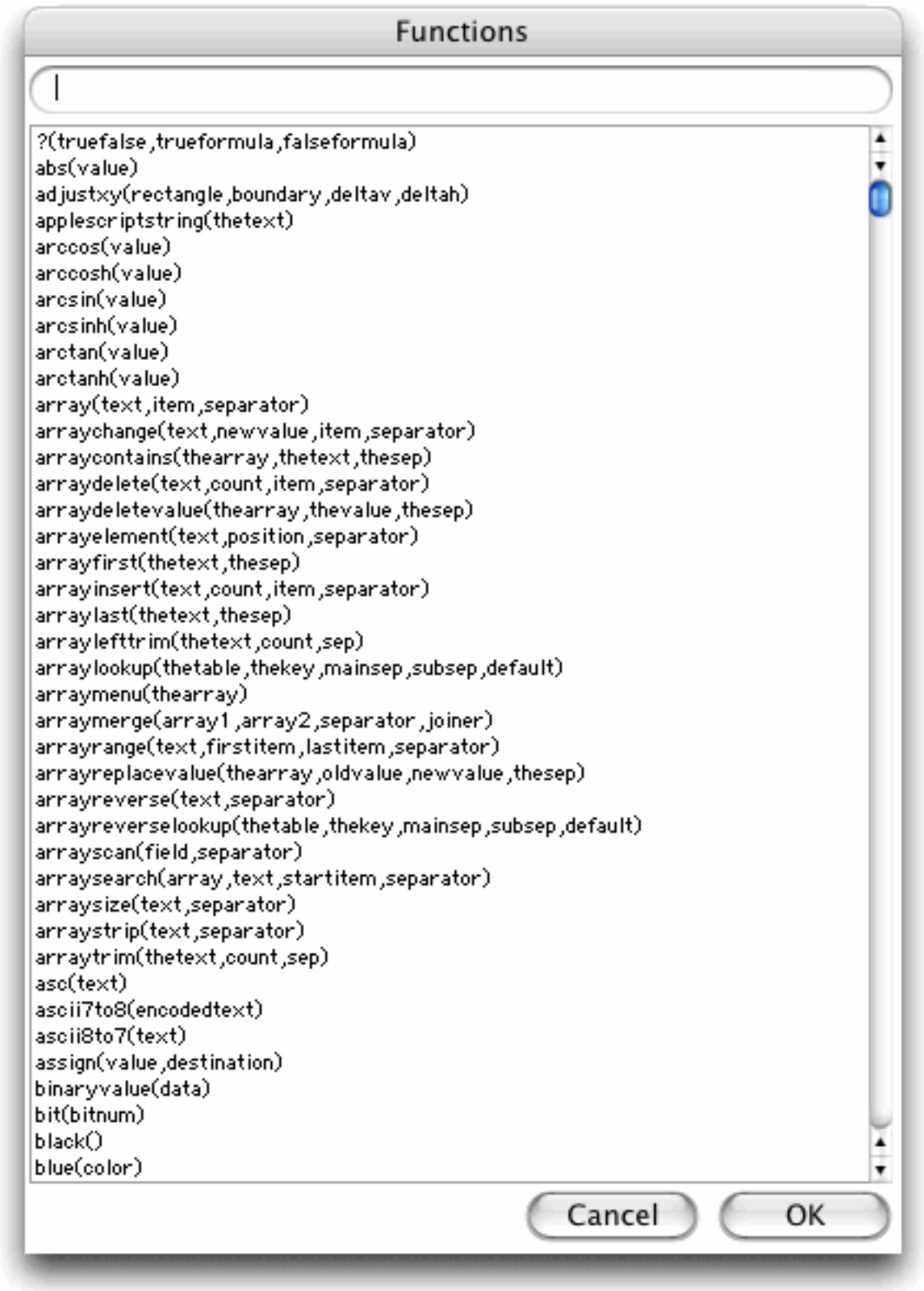
Just like the **Topics** menu, you can select any function in this menu and it will be typed into the Formula Wizard for you.

Function Search

The Search dialog provides an alternate way to help you enter functions and operators. To open this dialog choose **Search** from the Topics menu.



The dialog displays an alphabetized list of functions and operators.

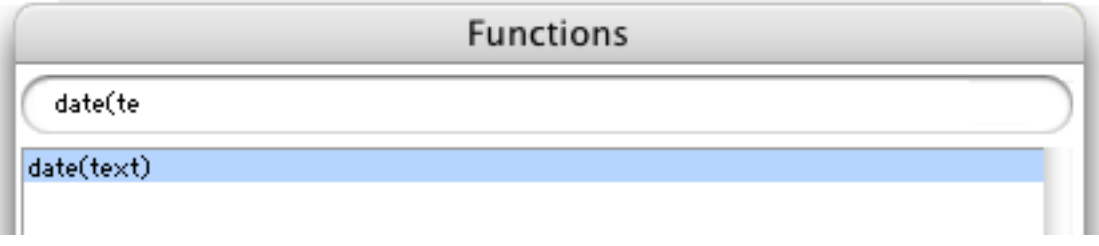


To type a function or operator into the formula you can click on it and press **OK** or the **Enter** key, or you can simply double click on the function or operator you want.

The list initially contains several hundred functions and operators. To cut down this list you can type in a search criteria. As you enter each key the number of items in the list will be reduced to show only functions or operators that contain the text you have typed in. For example the list below shows only functions that contain the word date.



At any time you can click on one of these functions and press **OK** or the **Enter** key, or you can simply double click on the function you want. If the list has been reduced to a single item you can simply press the **Enter** key.



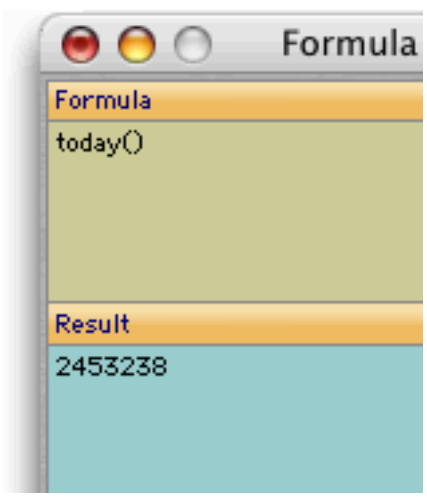
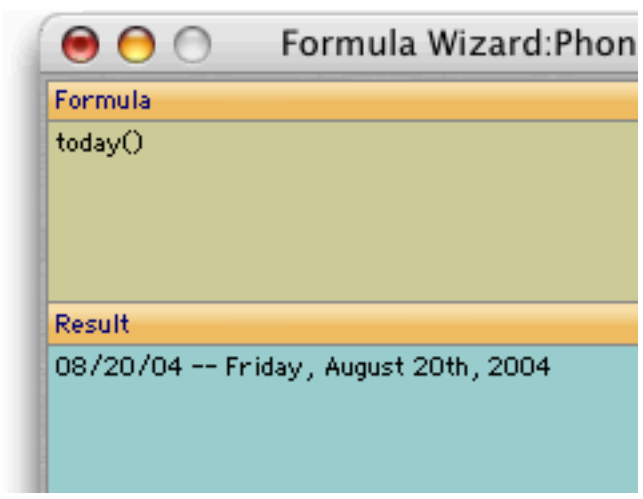
After the function or operator has been typed in you can advance through the parameters as described in the previous section.

Special Formula Result Formats

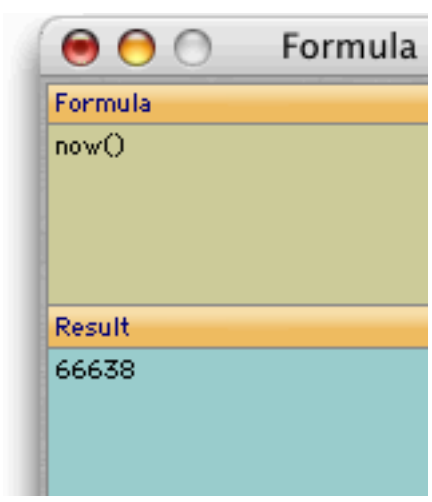
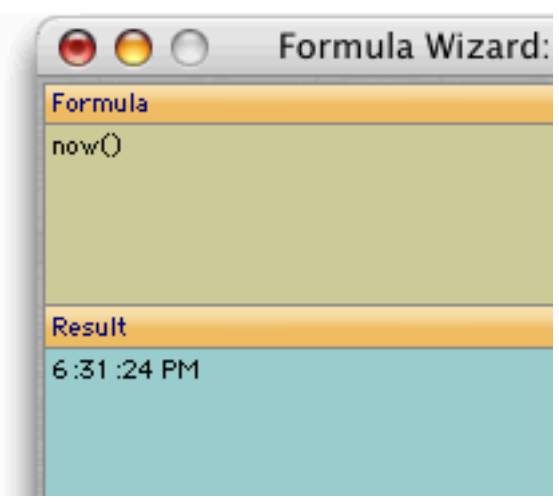
The Formula Wizard normally displays the result of the formula as a number or as text. Using the **Formula** menu you can select other custom formats for displaying the result.



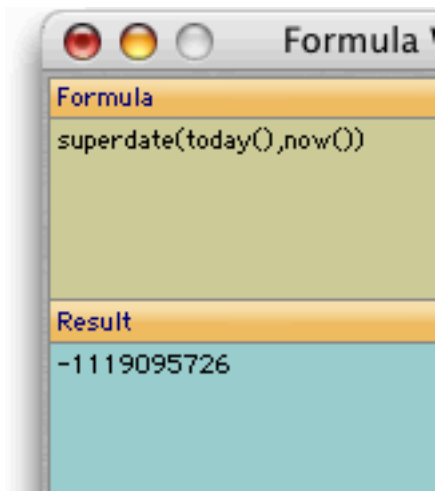
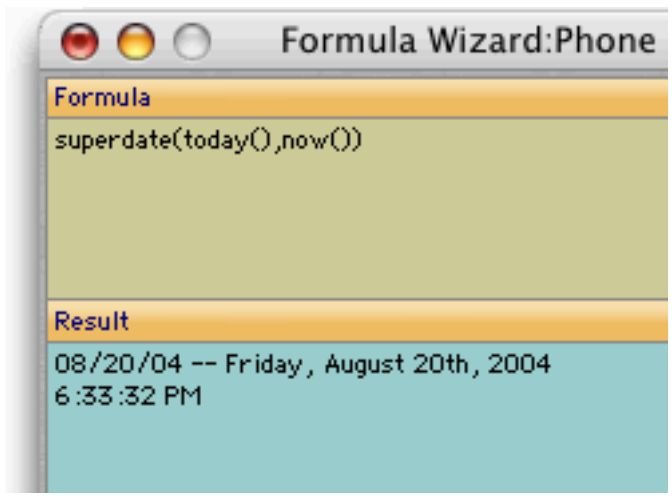
Use the **Date** option when you want to display the numeric result of a formula as a date (see "HTML Generating Functions" on page 1247). The table below illustrates how a date is displayed with both the **Normal** and **Date** options.

Normal	Date
	

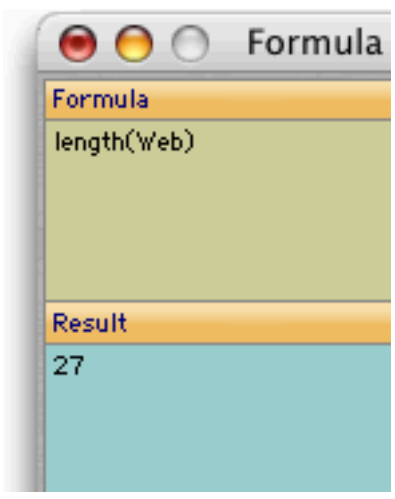
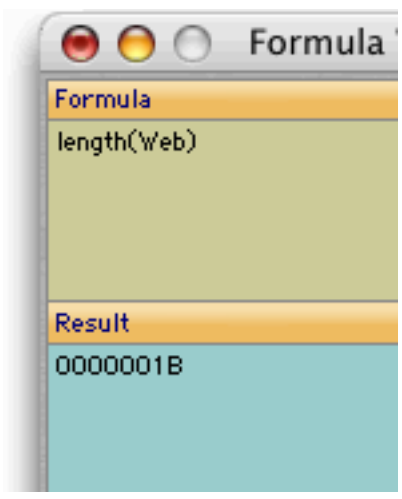
Use the **Time** option when you want to display the numeric result of a formula as a time (see "Time Arithmetic" on page 1255). The table below illustrates how a time is displayed with both the **Normal** and **Time** options.

Normal	Time
	

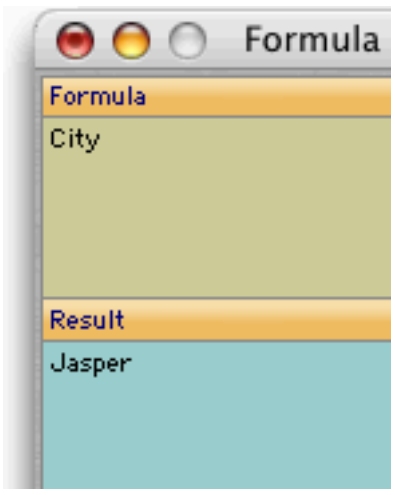
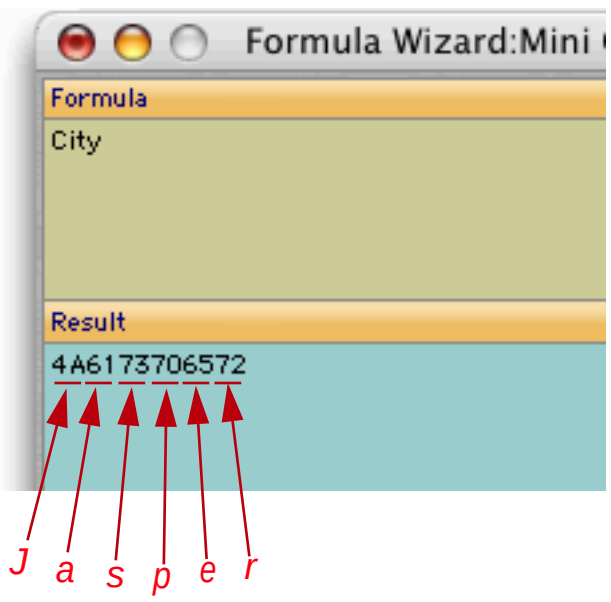
Use the **SuperDate** option when you want to display the numeric result of a formula as a SuperDate (see "SuperDates (combined date and time)" on page 1259). The table below illustrates how a result is displayed with both the **Normal** and **SuperDate** options.

Normal	SuperDate
	

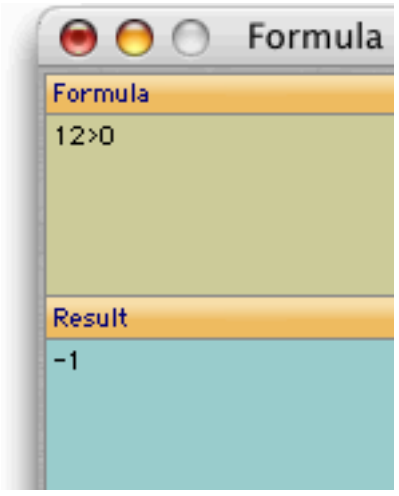
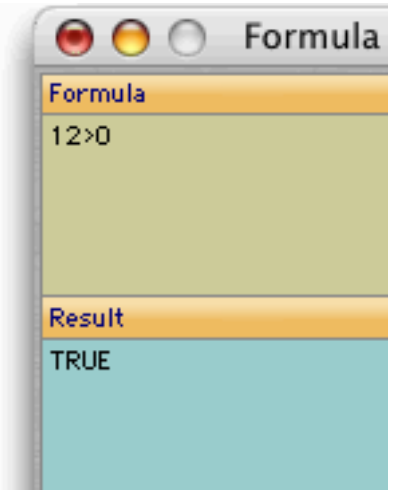
Use the **Hexadecimal** option when you want to display the result of a formula as a hexadecimal number (see "Raw Binary Data" on page 1296). The table below illustrates how a number is displayed with both the **Normal** and **Hexadecimal** options.

Normal	Hexadecimal
	

The **Hexadecimal** option may also be used to display text results. The ASCII value of each character is displayed in hexadecimal.

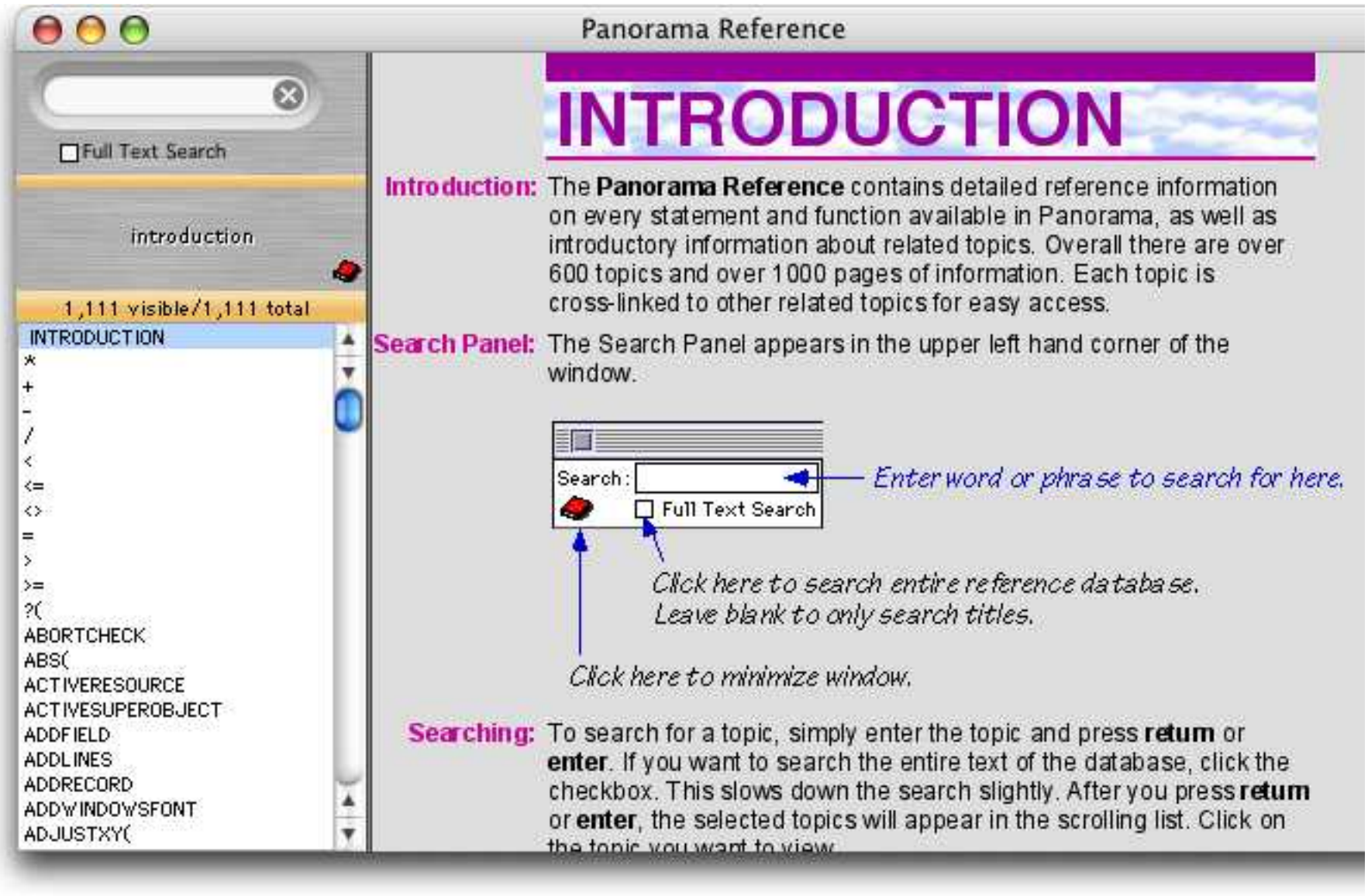
Normal	Hexadecimal
	

Use the **True-False** option when you want to display the result of a formula as true or false (see “True / False Formulas” on page 1265). The table below illustrates how a number is displayed with both the **Normal** and **True-False** options.

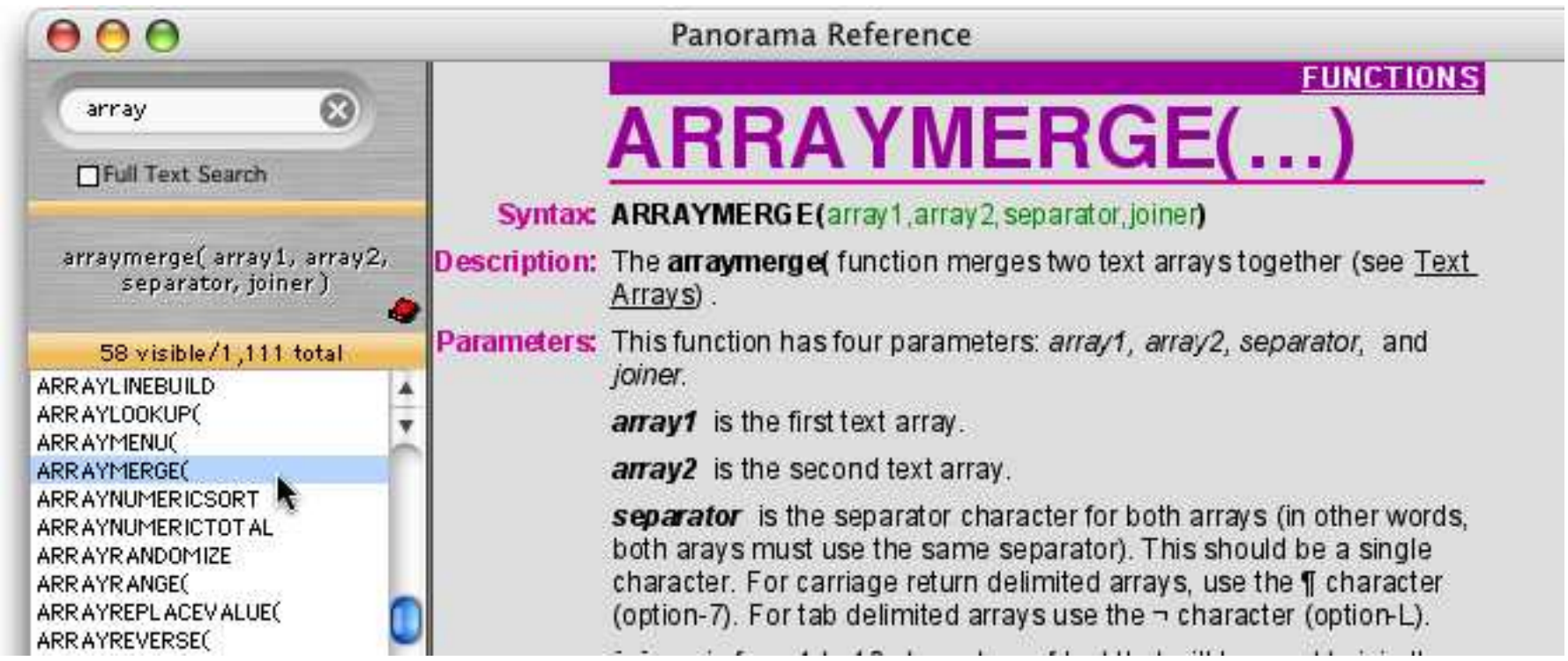
Normal	Hexadecimal
	

The Programming Reference Wizard

The fastest way to find complete information for any function or operator is to use the **Programming Reference** wizard. To open this wizard, select it from the Documentation submenu of the Wizard menu, or simply press **Control-R** if you are using a Macintosh computer.



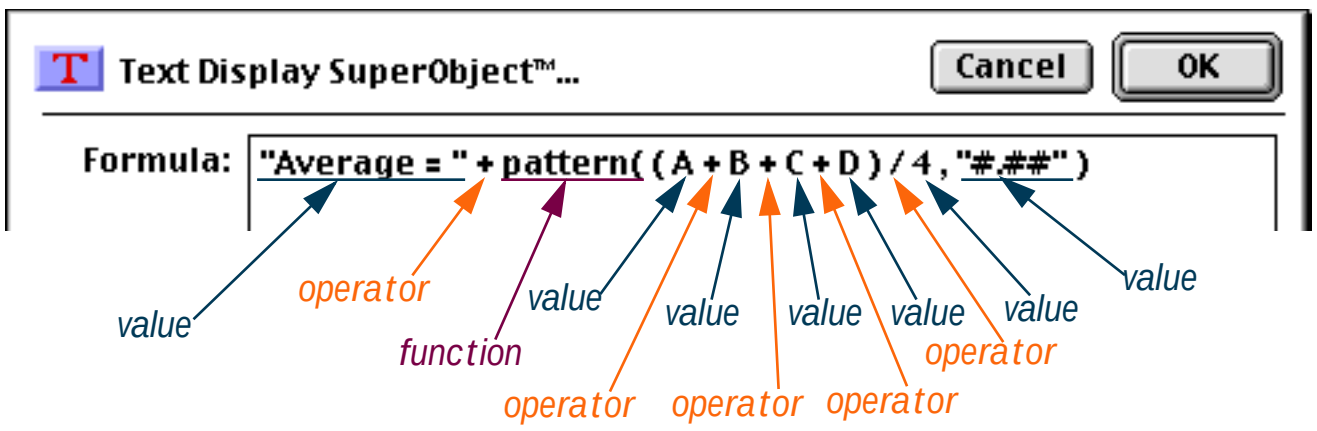
To find complete information for any function or operator simply type in the name of the function or operator in the search box in the upper left, and / or select the function or operator in the scrolling list on the left.



For additional information on this wizard see “Programming Reference Wizard” on page 1361.

Formula Components

Just as a sentence is constructed from basic words, a formula is created by combining simple elements — **values** (also called **operands**), **operators** and **functions**. Values (operands) are roughly equivalent to nouns, while operators and functions act as verbs. This illustration show the components that go into a typical formula.



Formula Grammar

Panorama formulas have grammar rules just as languages like English and Spanish do. These rules tell how values, operands and functions can be combined to make a valid formula.

The simplest formula is a single data value. Here are four examples of such simple formulas.

```
A

47

"Oregami"

ShippingMethod
```

Two values can be combined with an operator in between. The first example below adds two numbers together. The second example multiplies two numbers together. The third example appends two text values together (to produce a value like Mr. Jones).

```
2 + 2

Total * TaxRate

"Mr. " + LastName
```

The values must be the appropriate type for the operator. For example, you can multiply two numbers together like this

```
2 * 2
```

but you cannot multiply two text values together like this (see “Grammar Errors” on page 1193).

```
"Mr. " * LastName
```

You can combine three or more data values with an operator between each pair of values.

```
7 + 3 * 4 / 2

FirstName + " " + MiddleInitial + " " + LastName
```

Calculation Order and Parentheses

When a formula contains more than one operator, the calculations are performed from left to right unless one of the operators has a higher precedence (priority). This is the natural arithmetic order—multiply and division first, then addition and subtraction. This table lists the order of precedence for all operators.

1. Unary minus (example: -12)
2. Raise to power (example: 10^5)
3. Multiply and Divide
4. Integer Divide
5. MOD (remainder)
6. Add and Subtract
7. Comparisons (=, <>, <, >, …)
8. NOT
9. AND
10. OR and XOR

For example, consider the formula below.

```
7 + 3 * 4 / 2
```

Panorama first multiplies 3 * 4 to get 12, then divides this by 2 to get 6. Finally it adds 7 (addition is last because of it's low precedence) to get the final result, 13.

You can override the natural calculation order with parentheses. For example, the parentheses in the formula below force the addition to be calculated first, then the multiplication and division.

```
(7 + 3) * 4 / 2
```

Now the final result is 20 instead of 13. When in doubt you can always add parentheses to force Panorama to calculate the formula in any order you want.

Functions

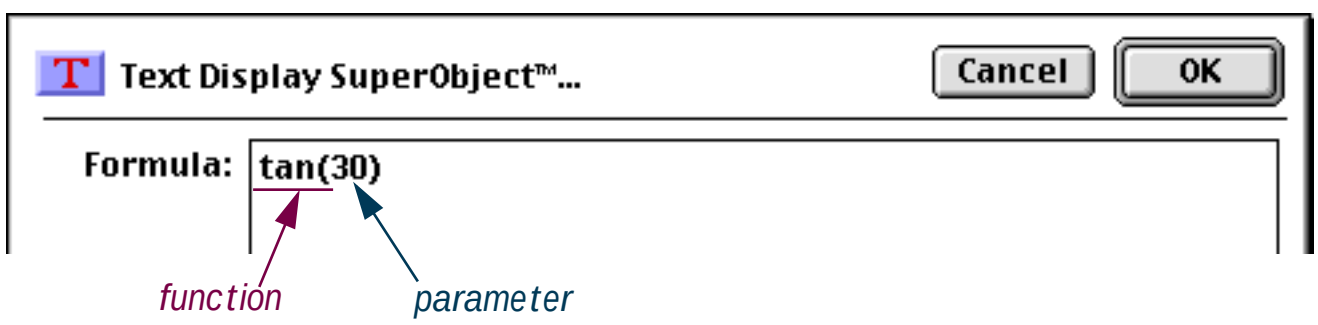
A function is a formula component that calculates a value. It may calculate the value out of thin air (for example, calculating the value of the current date or time) or it can calculate the value from other values (for example trigonometry functions calculate values from angles). Panorama has several hundred functions available. Each function has a name, and is always followed by parentheses. For example, the `tan(` function calculates the tangent (a trigonometry function) of an angle.

```
tan(30)
```

A function can be used in a formula anywhere a regular value can be used. Just as with ordinary values, you can use operators to combine functions with other values (and functions).

```
3 + tan(30)
```

The value operated on by the function is called a **parameter**.



A function takes the parameter value (in this case 30) and transforms it into another value (in this case -6.4053, the tangent of 30). The parameter can be a formula itself, like this.

```
tan( A + B )
```

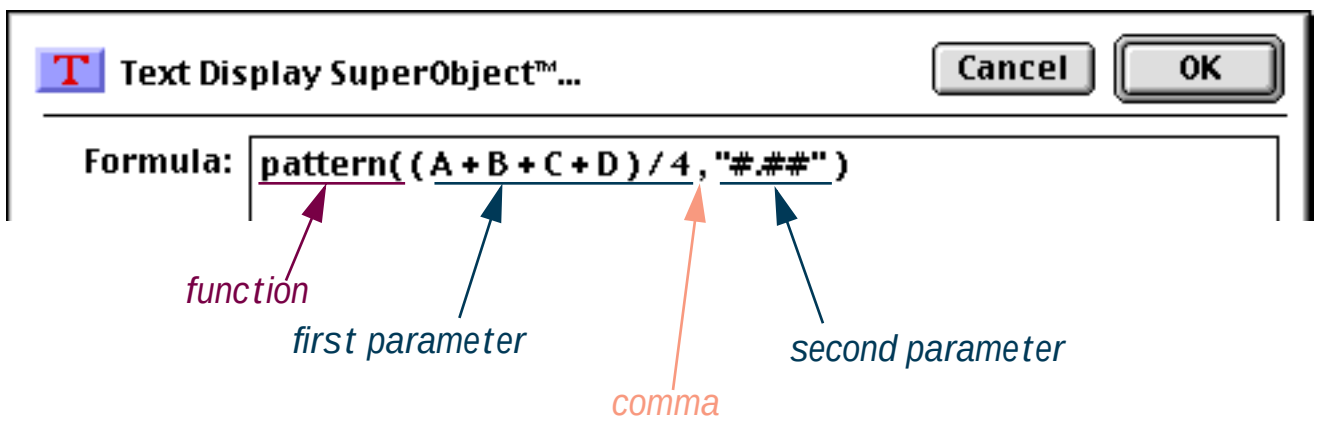
In this case Panorama first calculates the value A+B, then computes the tangent of that sum. A parameter may be as complex a formula as you need, with additional parentheses and even other functions nested inside the first function.

```
tan( sqr( A + B ) + 1 )
```

The parameter to the `sqr(` function is A+B, while the parameter to the `tan(` function is sqr(A+B)+1. (The `sqr(` function, by the way, calculates square roots.) Panorama will always calculate the formula from the inside out until the entire formula has been computed.

Multi-Parameter Functions

Many functions use more than one parameter. When more than one parameter is required each parameter is separated from the next by a comma. All of the parameters are surrounded by parentheses, just as with single parameter functions. For example, the `pattern(` function (shown below) requires two parameters. The first parameter must be a numeric value (in this case a calculated average) and the second parameter must be a text value containing a pattern for formatting the number (see “Numeric Output Patterns” on page 290).



Some functions require as many as six parameters. You must always supply every parameter — you cannot leave one out (see “Grammar Errors” on page 1193).

Zero Parameter Functions

A small handful of functions don’t require any parameters at all. These functions generate a value all by themselves, either by consulting the computer hardware (current date, current time), querying internal Panorama data (line number, imported data) or by generating a completely random number each time the formula is computed.

```
today()       -- current date

now()         -- current time

seq()         -- line number

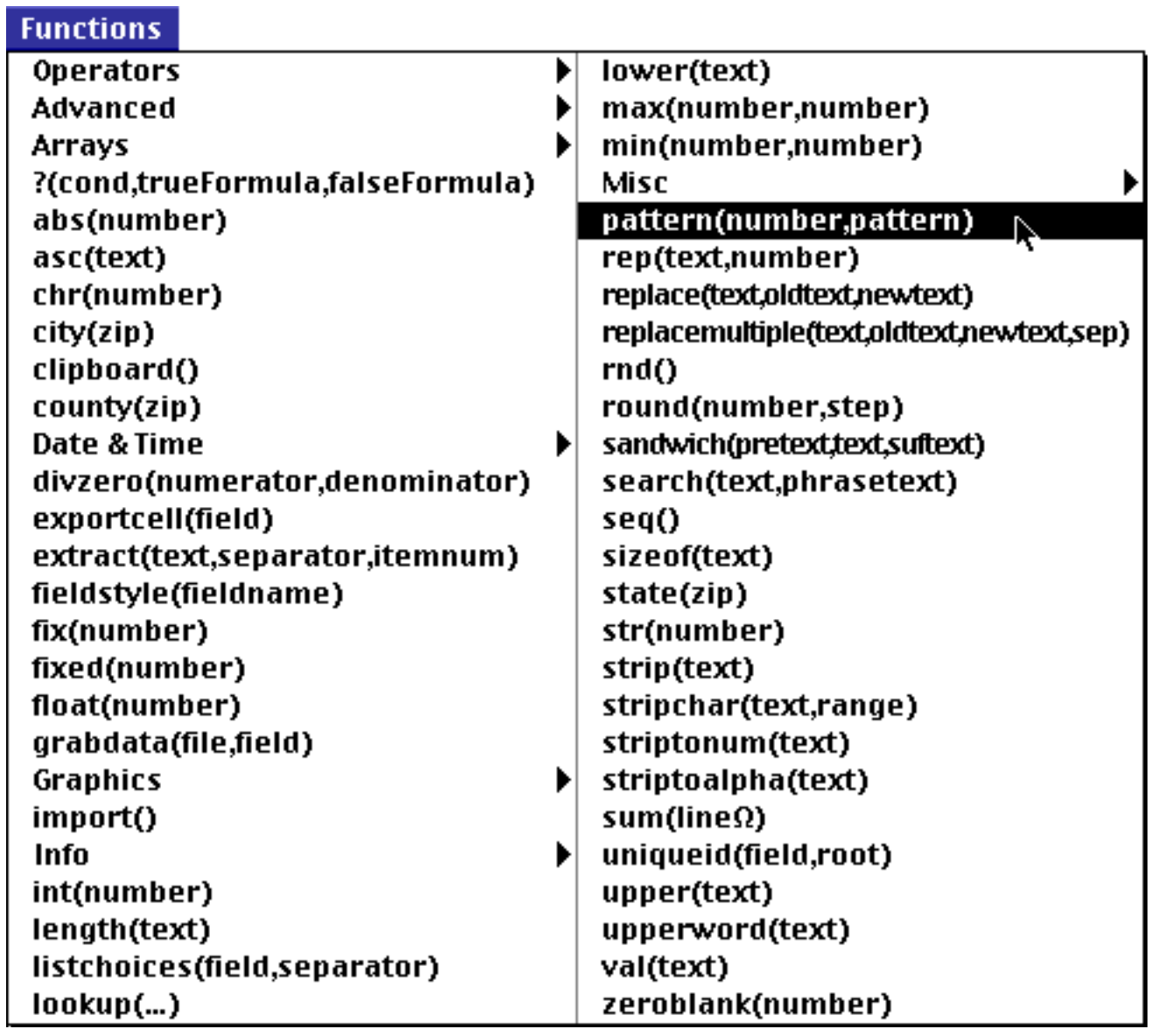
import()      -- line of text from import file

rnd()         -- random number
```

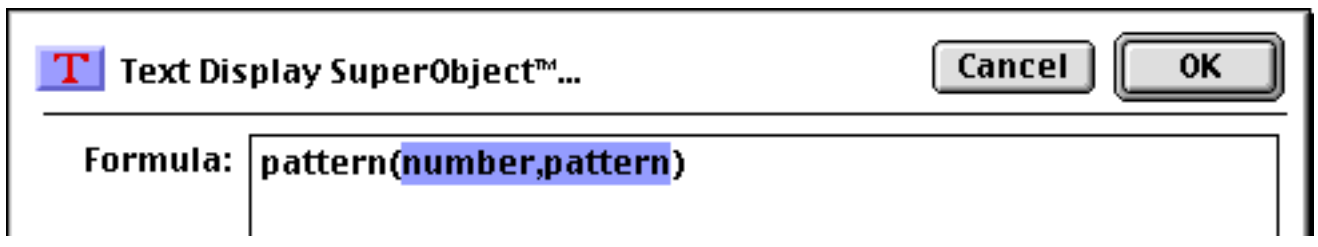
As you can see, these functions simply have both parentheses next to each other, with no parameter in between. You cannot omit the parentheses — you are required to include them as shown in the examples above.

Functions Menu

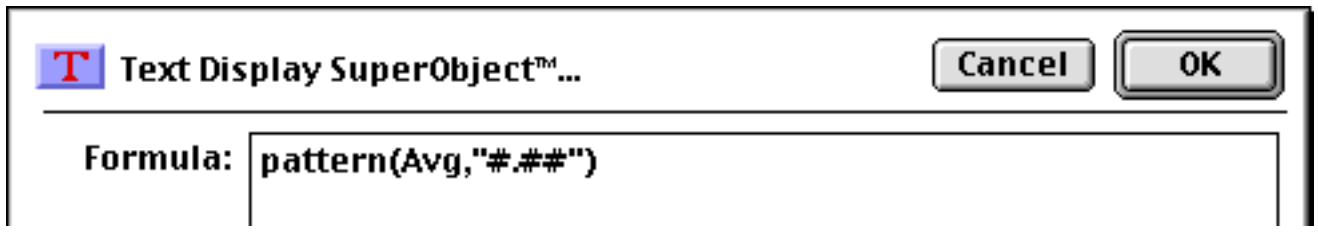
To help you type in a formula without errors Panorama has a special **Functions** menu that will type in a function for you. This menu is available whenever you enter a formula into a dialog, the design sheet or a procedure window. The function has a number of submenus that allow you to select from many of the hundreds of functions available. (Note: The Functions menu does not have ALL of Panorama's functions. For a complete and always up-to-date list see the Formula Wizard (see "Topic and Functions Help Menus" on page 1180) and the Programming Reference Wizard (see "The Programming Reference Wizard" on page 1187).)



When you select the function you want Panorama will automatically type it in for you. If the function requires parameters Panorama will type in a template for each parameter.



Simply type in the actual parameters to complete the function.



Whitespace

Most of the examples you've seen so far have extra spaces between the components, like these.

```
7 + 3 * 4 / 2

FirstName + " " + MiddleInitial + " " + LastName

tan( sqr( A + B ) + 1 )
```

Panorama ignores spaces between components. You can leave out the spaces, like this.

```
7+3*4/2

FirstName+" "+MiddleInitial+" "+LastName

tan(sqr(A+B)+1)
```

Or you can add extra spaces between components, or even carriage returns, like this. (Note: Some dialogs do not allow you to enter carriage returns, because pressing the **Return** key closes the dialog.)

```
7 + 3   *     4 / 2

FirstName + " " +
  MiddleInitial + " " +
  LastName

tan(   sqr( A + B ) + 1    )
```

Spaces are only ignored **between** components, not within components. A common mistake is to place a space in between the function name and the left parenthesis. This is not allowed. The formula below will not work (see "Grammar Errors" on page 1193) because of the spaces after `tan` and `sqr`.

```
tan ( sqr ( A + B ) + 1 )
```

Another common problem is spaces or other punctuation in field names. If your database has fields named First Name, Middle Initial and Last Name you might be tempted to try a formula like this.

```
First Name + " " + Middle Initial + " " + Last Name
```

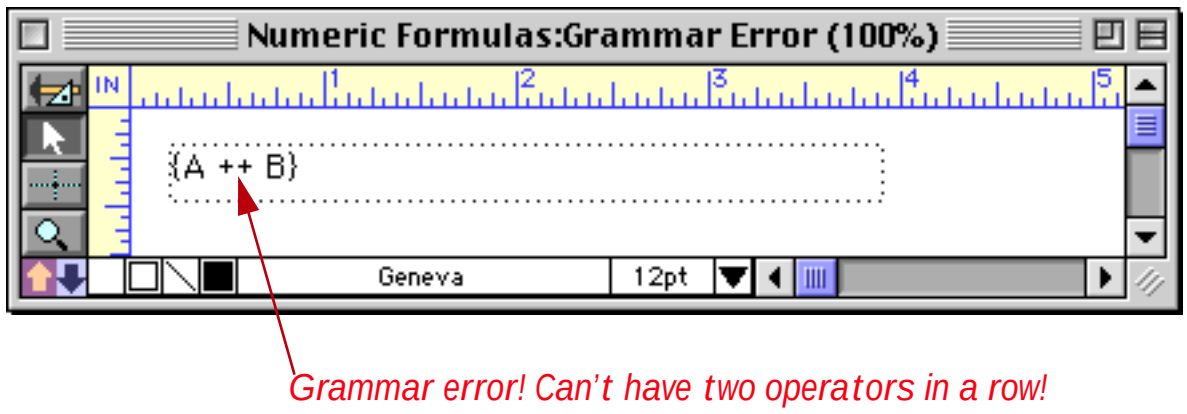
Sorry, but it won't work (see "Grammar Errors" on page 1193). Because of the spaces inside the field names, Panorama will think that First and Name, Middle and Initial and Last and Name are separate components. The solution is to place chevron (« and ») characters around the field names. In many cases you can use the **Field** menu to type in the field name with chevrons for you. Otherwise, on the Macintosh press **Option**-\ to create the « chevron character and **Shift**-**Option**-\ to create the » chevron character. On Windows systems press **Alt**-**0171** to create the « chevron character and **Alt**-**0187** to create the » chevron character. Here's the revised formula, which will work perfectly

```
«First Name» + " " + «Middle Initial» + " " + «Last Name»
```

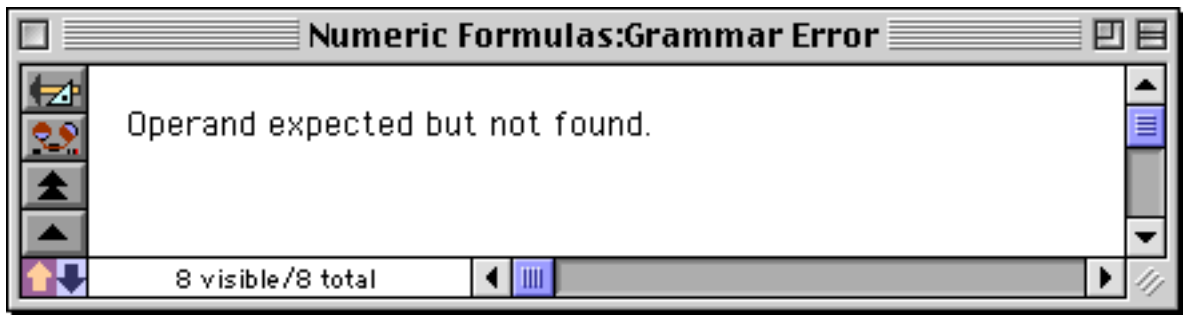
You'll also need to put chevrons around a field or variable name that contains punctuation, for example «P / E Ratio». Without the chevrons Panorama will think that this is four separate components — P, /, E and Ratio.

Grammar Errors

Unlike a human listener, Panorama is not able to tolerate incorrect or sloppy grammar. If you ask Panorama to calculate a formula that has incorrect grammar it will refuse to comply until you correct the mistake. For example, consider the formula shown below in an auto-wrap text object.

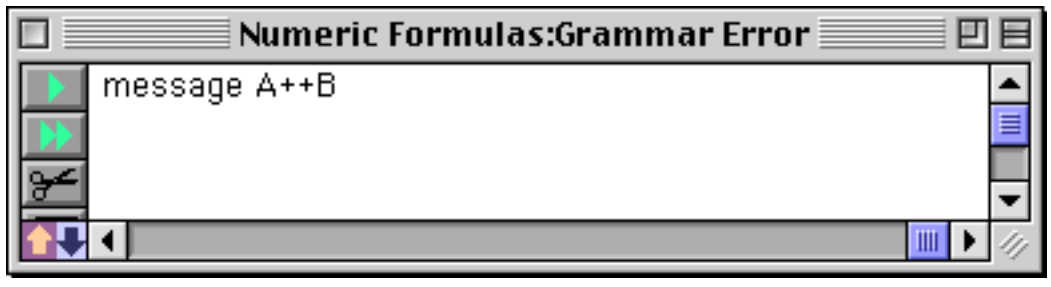


When you switch to Data Access Mode Panorama tells you about the grammar error.

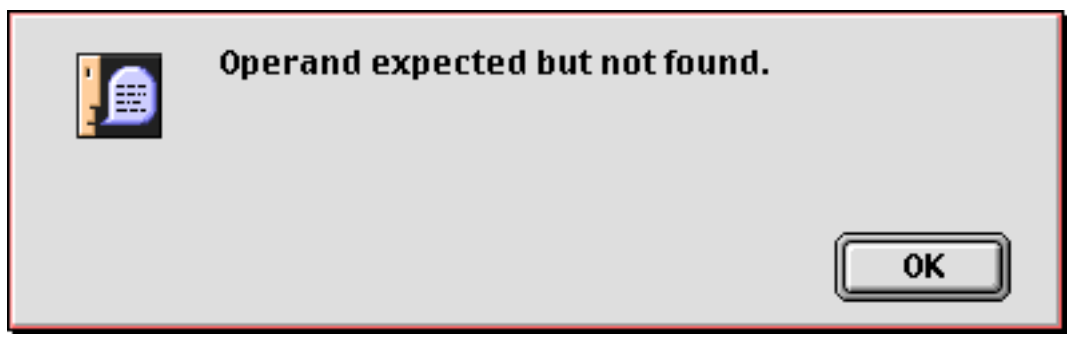


The error message tells you that Panorama expected an operand (value) after the + operator. The solution is either to remove the extra + operator or add another value in between the two + symbols.

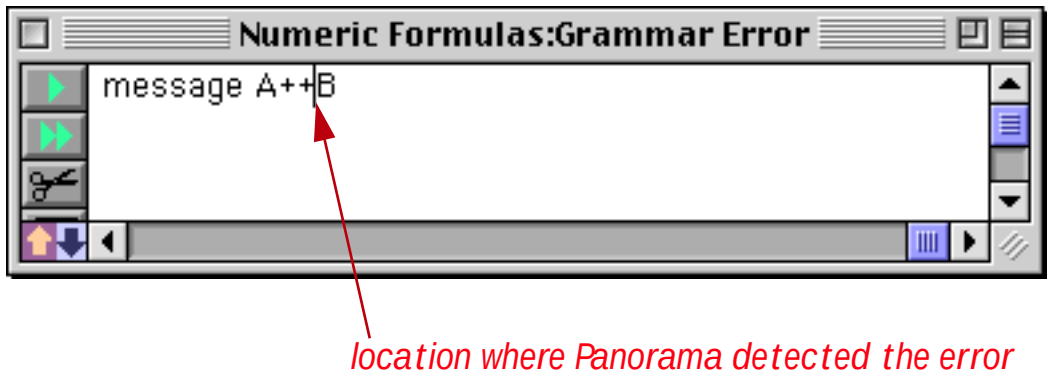
When you are editing a formula within a procedure, Panorama will attempt to point out the location of the grammatical error. For example, here is the same formula with the same error used in a procedure.



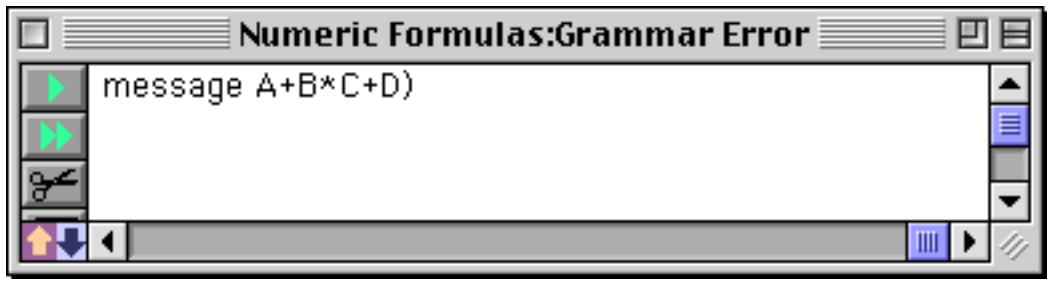
If you click to another window or use the **Check Procedure** command (in the Edit menu) Panorama will display an alert letting you know about the problem.



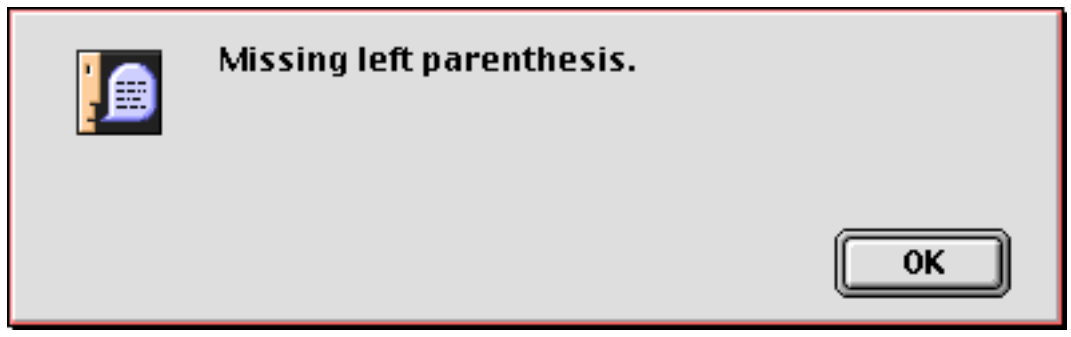
When you close the alert window Panorama will move the insertion point to the location where Panorama detected the error.



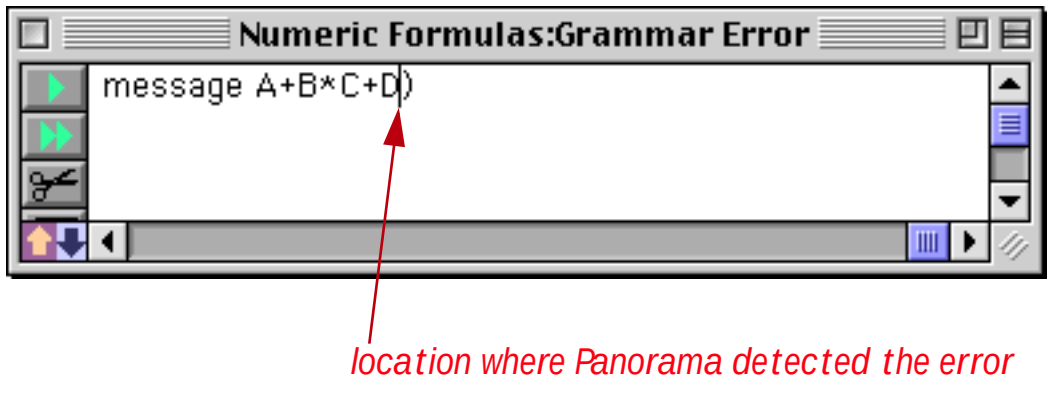
This location is usually fairly close to where the actual error is. However, in some cases Panorama is unable to determine exactly where the problem is. Consider the formula shown below, which has a missing left parenthesis.



When you click to another window or use the **Check Procedure** command (in the Edit menu) Panorama will display an alert letting you know about the problem.



When you close the alert window Panorama will move the insertion point to the location where Panorama detected the error.



But wait — is this really where the error is? No, the error actually is somewhere earlier in the formula. In this case the missing (probably goes in front of the B or the C. Panorama has done the best job it could to locate the error for you. One thing you do know for sure, though, is that the error is always before the insertion point and not after.

Comments

Panorama allows “comments” to be placed inside a formula. A comment is a note within the formula that is ignored when the formula is evaluated. A comment must start with /* and end with */. Anything between these will be ignored. (C and JavaScript programmers will recognize this style.) For example, this formula:

```
«CRatio» /* CRatio must be updated every 24 hours */ * Amount
```

will produce the same value as this one:

```
«CRatio» * Amount
```

In addition to notes to yourself, comments are also useful for temporarily disabling a section of a long formula (for example if you are trying to debug the formula).

Values

Values are the raw material that formulas work with—numbers and text. A value may be embedded in the formula itself, may be stored in a database field or may be stored in a variable (see “Variables” on page 1199 and “Variables” on page 1370).

Constants

When a value is embedded in the database itself it is called a **constant**. A numeric constant may be in fixed point format, like the numbers in this example (the numeric constants are highlighted in purple).

```
x + 2

today() - 90

Total * 0.0625
```

A numeric constant may also be in floating point format, which consists of the mantissa followed by the letter **e** followed by the exponent. The example below is equivalent to the mathematical formula $x \cdot 6.02^{23}$.

```
x * 6.02e23
```

A formula may also contain **text constants**. A text constant is a series of characters surrounded by quotes. When writing a text constant you may choose from five different types of quotes, as shown in this table.

Type	Open	Close	Example
Double Quote	"	"	"January"
Single Quote	'	'	’Tuesday’
Curly Braces	{	}	{San Francisco}
Smart Double Quote	“	”	Gothic
Smart Single Quote	‘	’	Bohemian

The primary reason for different types is to allow quotes themselves to be used in a text constant. Suppose that you needed to use the text The shim was 6" high in a formula. Using double quotes around the constant will cause a grammar error, because Panorama won’t know what to do with the text after 6" (shown in red below).

```
"The shim was 6" high"
```

One possible solution is to use a different quote character around the constant. Any of the examples shown below will work.

```
'The shim was 6" high'

{The shim was 6" high}

“The shim was 6" high”

‘The shim was 6" high’
```

Another solution is put two double quotes in a row (as highlighted dark blue in the example below). Panorama will convert these into a single quote and continue with the text constant.

```
"The shim was 6"" high"
```

Build in Constants: Pi, Carriage Return and Tab

Panorama has one built in numeric constant—**pi**. Use the Greek **π** symbol to access this value. For example the area of a circle can be calculated with this formula.

```
π * radius^2
```

To create the **π** symbol on the Macintosh press **Option-P**. On the PC, type **Alt-0254**.

Panorama has two built in text constants—¶ (Carriage Return) and¬ (Tab). For example three line address can be included in a formula like this.

```
"Suzette Elliot"+¶+892 Melody Lane"+¶+"Fullerton, CA 92831"
```

To create the ¶ symbol on the Macintosh press **Option-7**. On the PC, type **Alt-0182**.

To create the ¬ symbol on the Macintosh press **Option-L**. On the PC, type **Alt-0172**.

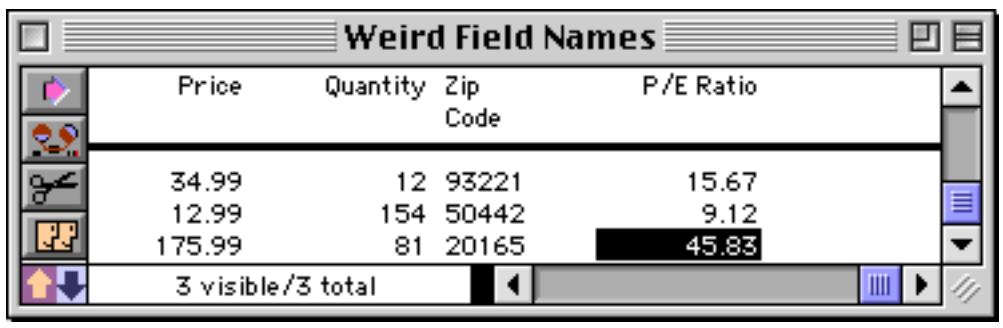
Fields

To use a field within a formula, type the name of the field into the formula. This formula adds up the sum of three fields.

```
SubTotal+Shipping+Tax
```

When a field is used in a formula it always refers to the value of that field in the current record in the current database (the database belonging to the topmost window). As you move from record to record the result of the computation will change depending on the values in that particular record. (The only exception to this rule is the `lookup(` and `grabdata(` functions, which may refer to fields in other records or even other databases.)

If a field name contains spaces, numbers, or punctuation marks in it, you must surround the name with chevron characters (« and »). (On the Macintosh press **Option-** to create the « chevron character and **Shift-Option-** to create the » chevron character. On Windows systems press **Alt-0171** to create the « chevron character and **Alt-0187** to create the » chevron character.) If the field name contains carriage returns, they must be represented with spaces. Here is a database with some unusual field names.



The first two names can be used without chevrons, but the last two require chevrons because of spaces and punctuation in the names.

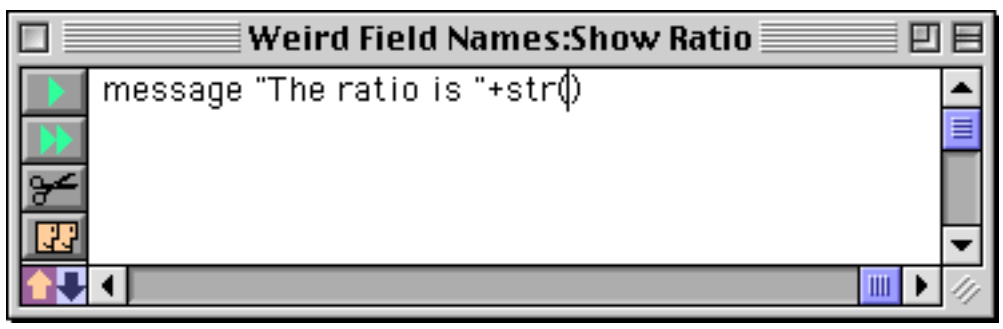
```
Price

Quantity

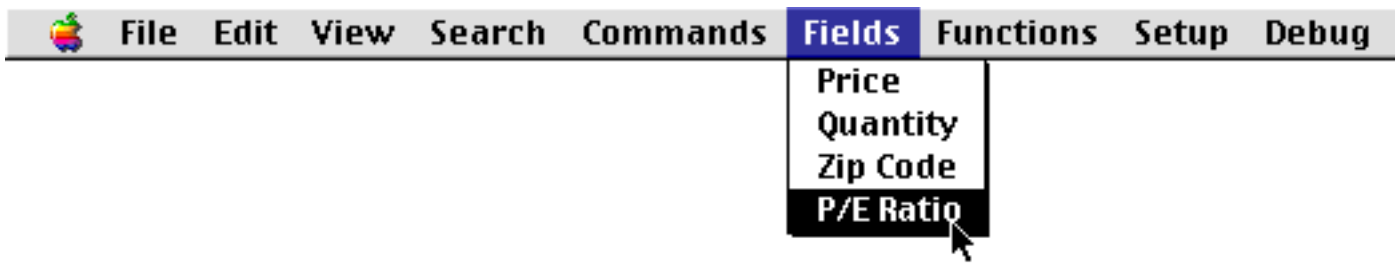
«Zip Code»

«P/E Ratio»
```

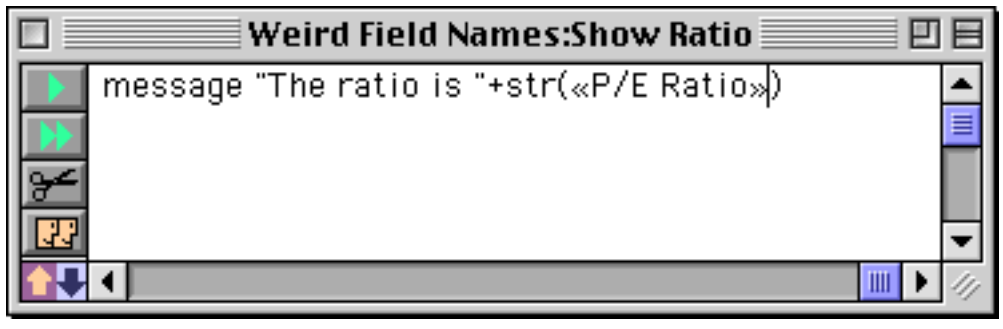
Formulas require field names to be spelled exactly as they appear in the database, with no typos allowed. Fortunately, Panorama can help you out with this. Start by positioning the insertion point where you want the field to appear.



Now pick the field from the **Field** Menu. This menu is available whenever you are editing a formula in a dialog, design sheet or procedure.



Panorama will type in the field name for you, including the chevrons if necessary (as they are in this case).



If the chevrons are not necessary (for example for Price or Quantity) Panorama will not include them.

Using the Current Field

A formula may use «» (see “Special Characters” on page 1202) to refer to the current field without having to know what the current field is. For example, this formula converts the current cell to upper case.

```
upper(«»)
```

If necessary, a formula can find out what the current field name is with the `info("fieldname")` function (see “INFO("FIELDNAME")” on page 5375).

Line Item Fields

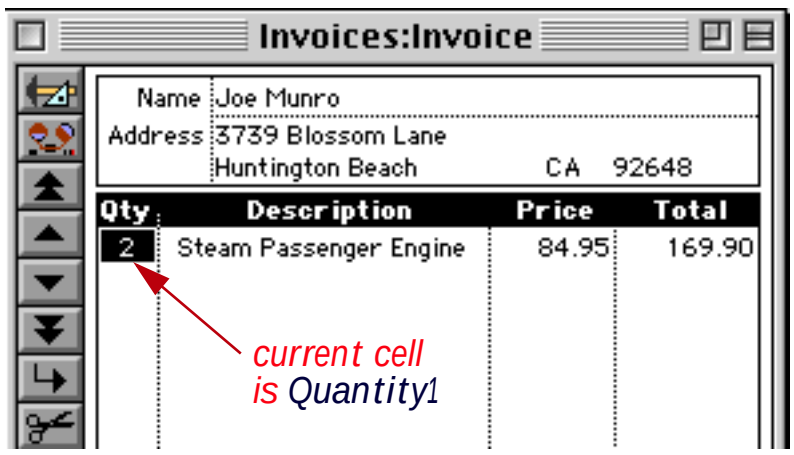
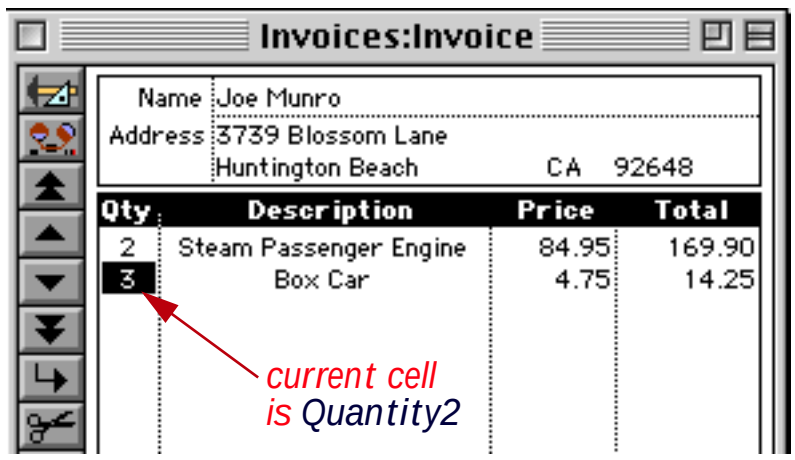
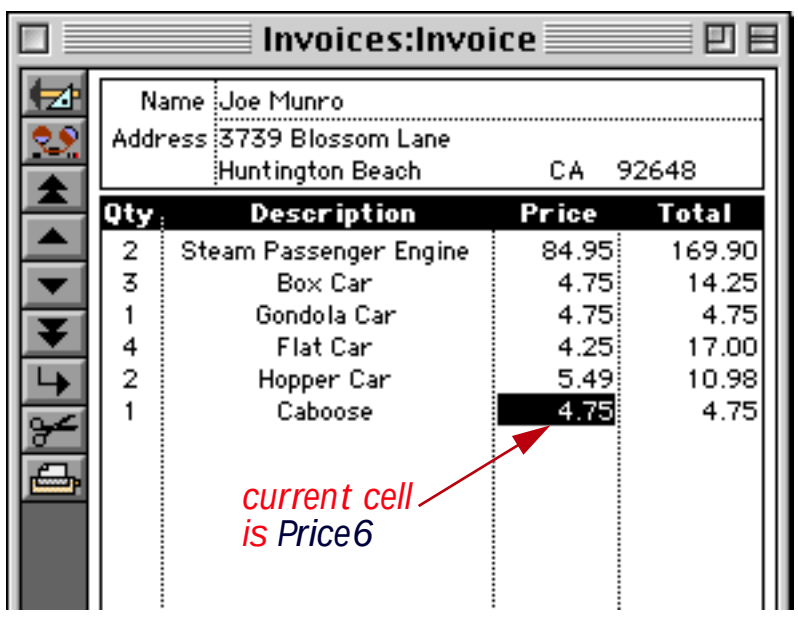
Line items are used for repeating items within a record (see "Repeating Fields (Line Items)" on page 262). Line item fields always end with a numeric suffix, for example Qty1, Qty2, Qty3, etc. Line item fields can be used in formulas just like other fields, for example:

```
Quantity3*Price3
```

When the current cell is a line item field the Ω symbol (see "Special Characters" on page 1202) can be used as an automatic numeric suffix. Panorama automatically adjusts the suffix depending on what cell is currently active. For example, consider this formula using the Ω symbol.

```
QuantityΩ*PriceΩ
```

When this formula is calculated, Panorama will automatically substitute the correct line item number for each Ω symbol as shown in this table.

Current Cell	Adjusted Formula
	`Quantity1*Price1`
	`Quantity2*Price2`
	`Quantity6*Price6`

If a formula containing the Ω character is used when the current cell is not a line item cell an error will occur.

To add up a series of line items you can use the `sum(` function. See “Adding Line Item Fields” on page 1207.

Variables

A variable is a place in the computer where an item of data can be stored, kind of like a storage bin for a value. Variables may be created by procedures or by SuperObjects. Most procedures will use one or more variables to hold and transfer data as the program runs (see “Variables” on page 1370 for more details on how variables can be created and used in procedures). Use a variable whenever you need to store a single data item so that you can use it later. Unlike a field, the value variable doesn’t change as you move from record to record, or, in the case of a global variable, even when you move from database to database.

Variable Names

Just as a house is identified by its address, a variable is identified by its name. A street address tells you exactly how to find a house or business. It doesn’t tell you who or what is inside the house, however. Families may come and go, but the street address remains the same. In a similar way, a variable name identifies a place where data can be stored. The data may change, but the variable name remains the same.

Panorama allows any sequence of characters to be used as a variable name. However, if the variable name contains any punctuation (including spaces) it must be surrounded by the chevron characters « and ». (On the Macintosh press **Option**-`\` to create the « chevron character and **Shift**-**Option**-`\` to create the » chevron character. On Windows systems press **Alt-0171** to create the « chevron character and **Alt-0187** to create the » chevron character.) Here are some examples of typical variable names:

```
X

birthDay

Counter

«Tax Rate»

«PrimeRate%»
```

A variable name must be spelled exactly the same way every time, including upper and lower case. The variable name birthDay is not the same as Birthday or birthday. In fact, you could create three different variables using these three different names (although this is not recommended because it would be very confusing).

By the way, it’s always ok to use chevrons around a variable name, even if the name doesn’t have any punctuation. «Counter» is exactly the same as Counter, and they can be used interchangeably. So if you have any doubts about whether or not chevrons are necessary, go ahead and use them. No harm, no foul.

What’s Inside A Variable?

By itself, a variable has no meaning, no value…until you put some data in it. When you use a variable in a formula or procedure, you are actually telling Panorama to use the contents of the variable.

A variable is sort of like a cup that you can pour anything into. A cup may contain water, soda, tea, or coffee. If you tell a person to drink the blue cup, what you really mean is to drink whatever liquid is in the blue cup. Each time they drink they may get a different liquid, depending on what the blue cup has been filled with.

Using a variable is similar. If you tell Panorama to calculate X+Y (where X and Y are variable names), what you really mean is “take whatever value is in X and whatever value is in Y and add them together.”

It’s important to remember that a variable name simply identifies the variable, but the name is not the variable itself. The name is like a placeholder for the real contents of the variable.

The Life Cycle of a Variable

A variable doesn't just appear by magic. It must be created, just as you have to build a house before you can move in. Once the variable has been created it can be used for storing a data item. However, variables don't last forever. Most variables eventually disappear without a trace. You can also force a variable to disappear at any time — see "Destroying a Variable" on page 1372.

Panorama has five kinds of variables: local, window, fileglobal, global and permanent. The only difference between these three types of variables is how long they last before disappearing and when the variables are available.

Local variables are the most short-lived. A local variable disappears when the procedure that created the variable is finished. In addition, a local variable can only be used by the procedure that created it. If procedure A calls procedure B as a subroutine, procedure B cannot access the local variables created by procedure A. In fact, procedure B could create its own local variables with the same names as the local variables created by procedure A. Panorama keeps the local variables for each procedure completely separate from each other.

Window variables are associated with a particular window. A window variable is only accessible when the window it is associated with is on top, and the variable disappears completely when the window is closed. It is possible for several different windows to have window variables with the same name. In that case, each window variable may have a different value.

FileGlobal variables are associated with a particular database (file). A fileglobal variable is only accessible when the database it is associated with is the current database (on top), and the variable disappears completely when the file is closed. It is possible for several different files to have fileglobal variables with the same name. In that case, each fileglobal variable may have a different value. For many applications fileglobal variables are the best choice because there is no chance of an accidental conflict with a variable of the same name in another database.

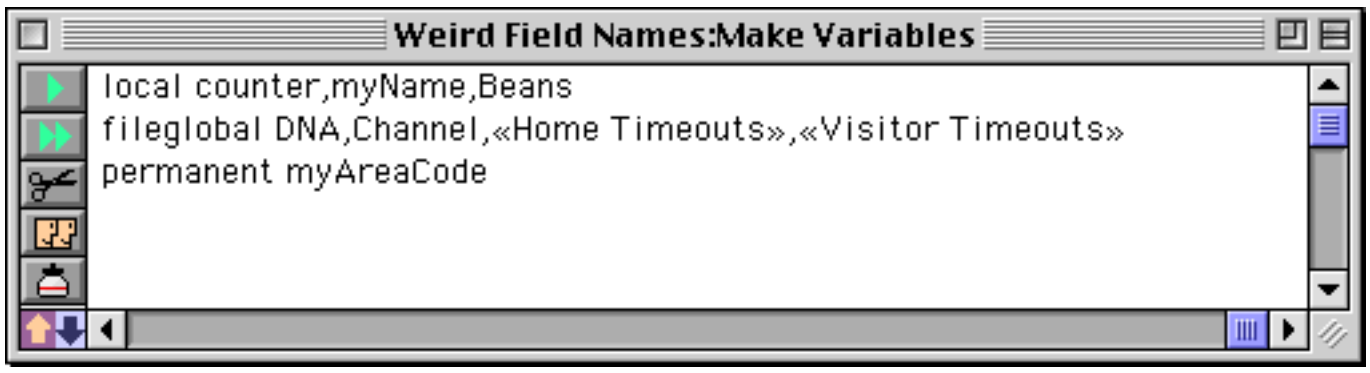
Global variables are relatively long-lived. A global variable doesn't disappear until you quit from Panorama. Even if you close the database, the global variable remains. Once a global variable has been created it can be accessed in any procedure, in any database or window, at any time. You should avoid using global variables unless you absolutely need universal access across databases for the value stored in the variable. If the value is only needed in one database it is much better to use a fileglobal variable to avoid the chance of an accidental conflict with another database using the same global variable name.

Permanent variables are almost immortal. When the database is saved, all permanent variables in that database are also saved. Like a fileglobal variable, a permanent variable is only accessible when the database it is created in is the current database, and a fileglobal variable disappears when you close the database. However, unlike a fileglobal variable, a permanent variable will re-appear like a phoenix from the ashes when you re-open the database. In fact, there are only two ways a permanent variable can permanently disappear. First, you can explicitly kill a permanent variable with the `unpermanent` statement. Secondly, you can create a permanent variable but never save the database.

Creating Variables in a Procedure

Panorama has five statements for creating variables in a procedure: `local`, `windowglobal`, `fileglobal`, `global`, and `permanent`. Each of these statements is exactly the same except for the type of variable created. The statement must be followed by a list of one or more variables to create, with each variable name separated from the next by a comma. (Remember: if a variable name contains punctuation, it must be surrounded by chevrons « and ».)

Here are a few examples of typical statements for creating variables:



There is no limit to the number of local, global, and permanent statements you use in your programs, and no limit to the total number of variables (except for scratch memory, see below).

Initializing Variables

Creating a variable creates a place to store data, but it doesn't actually put any data in the variable. It's kind of like a new house that no one has moved into yet. If you try to access the variable before any data has been put into it, an error occurs.

To put data into a variable, use an assignment statement. Here's the start of a procedure that creates a variable named Count and initializes it to zero. The variable is now ready to use.

```
local Count
Count=0
```

Sometimes you may not be sure if a global variable has been initialized yet. If it has not been, you want to initialize it. But if it has already been initialized, you don't want to disturb the value that is already there. You can get around this problem with the if error statement, as shown in this example.

```
global AreaCode
AreaCode=AreaCode
if error
   AreaCode="714"
endif
```

This procedure starts by creating a global variable named AreaCode. However, it's possible that AreaCode has already been created and initialized by another procedure. To test this, the procedure copies the variable to itself. If the variable is already initialized, there will be no error and the contents of the variable have not been disturbed. If the variable is brand new and has not been initialized, an error occurs. This error is trapped by the if error statement and the variable is initialized. If you have a number of variables that are always initialized as a group, you don't need to test each one. Just test one, and if an error occurs initialize the entire group of variables (see "The Define Statement" on page 1369).

Variables and Data Types

A variable can hold any kind of data: text, numbers, and secondary data types like dates, times, points, rectangles, etc. In addition, you can change the type of data in a variable at any time. One minute the AreaCode variable can contain text, moments later it can contain a number. The variable takes on the data type of whatever data you copy into it.

SuperObject Variables

A number of Panorama SuperObjects™ have the option of linking to a variable or a field. These SuperObjects™ include the Text Editor, Data Button, Pop-up Menu, List, Sticky Button and Scroll Bar. If one of these objects is linked to a variable and the variable does not exist, Panorama will automatically create a global variable when it opens the form, and initialize the variable to empty text. Except for how it was created, this global variable is just like any other global variable and can be used freely in procedures and formulas.

Variable Name Conflicts

If two database files define a global variable with the same name, you've got a conflict. It's kind of like two families trying to share the same house. This can work if the two families have an arrangement, but if they don't the result is chaos.

The best solution to this problem is to avoid it. If you can, use a fileglobal variable instead of a global variable. If this is not possible, stay away from simple global variable names like X, Payment, Count, etc. If possible, choose names that incorporate the database name (or an abbreviation of the name), for example InvoiceTaxRate, ReceivablesTotal, or APLastReconcileDate.

Variable names (even for local variables) can also conflict with fields in a database. In this battle, the variable always wins. Panorama will use the data in the variable instead of the data in the field. Avoid variable names that are the same as field names.

Permanent Variable Tips

When the `permanent` statement creates a permanent variable, it really creates two variables: one in memory and one in the current database. The one in memory is an ordinary fileglobal variable. Whenever the database is saved, Panorama copies the contents of the fileglobal variable into the copy of the variable in the database itself, then saves the database. Just like any other data, the contents of the permanent variable are not saved unless the database itself is saved. However, if you have not made any other changes to the database, Panorama will not warn you if you attempt to close a database without saving changes to the permanent variable.

Whenever a database is opened, Panorama automatically creates fileglobal variables for any permanent variables associated with that database. Next it copies the values from the database into the fileglobal variables. The variables are now ready to use.

If you ever want to make a permanent variable un-permanent, use the `unpermanent` statement, which is followed by a list of variables you want to make unpermanent. This statement doesn't make the variables go away, but they will no longer be permanent. The `unpermanent` statement only affects variables that are permanent in the current database. The example below changes two permanent variables back into regular (non-permanent) fileglobal variables.

```
unpermanent myAreaCode,myZipCode
```

Special Characters

Formulas are very picky about special characters. You've got to use the right special character in the right spot—no substitutes are allowed.

For example, some people mistake the bracket [] characters for the parentheses (). On your keyboard, the parentheses are created by pressing **Shift** and the **9** or **0** keys. Another common mistake is using the **** (backslash) instead of the **/** (slash) for divide. The table below lists all the special characters used by formulas and shows how to type them.

Character	Name	Mac	PC
(	left parenthesis	Shift-9	Shift-9
)	right parenthesis	Shift-0	Shift-0
[	left bracket	[	[
]	right bracket	]	]
{	left curly brace	Shift-[	Shift-]
}	right curly brace	Shift-]	Shift-]
«	left chevron	Alt-0171 Option-\	

Character	Name	Mac	PC
»	right chevron	Shift-Option-\	Alt-0187
^	caret (raise to power)	Shift-6	Shift-6
*	asterisk (multiply)	Shift-8	Shift-8
÷	divide	Option-/	not available, use /
=	equal	=	=
≠	not equal	Option-=	not available, use <>
<	less than	<	<
>	greater than	>	>
≤	less than or equal	Option-<	not available, use <=
≥	greater than or equal	Option->	not available, use >=
¶	paragraph	Option-7	Alt-0182
¬	export tab	Option-L	Alt-0172
§	section mark	Option-6	Alt-0167
¢	cents	Option-4	Alt-0162
‘	left smart quote	Option-]	Alt-0145
’	right smart quote	Shift-Option-]	Alt-0146
“	left smart double quote	Option-[	Alt-0147
”	right smart double quote	Shift-Option-]	Alt-0148
Ω	omega (line items)	Option-Z	Alt-0166
π	pi	Option-P	Alt-0254

To use the **Alt** key on the PC you must hold down the **Alt** key, then press the numeric digits (for example **0182**) then release the **Alt** key. When you release the **Alt** key the special symbol will appear.

Working With Extremely Complex Formulas

Panorama has an internal 2000 byte buffer it uses for processing formulas. This allows very complicated formulas to be processed (since each function is represented in the buffer by a single byte, the actual formula can contain even more than 2,000 characters). However, it is possible to create a formula too large to fit into the buffer. When this happens Panorama generates an Expression too complicated error message.

To avoid this error Panorama allows you to create a larger expression buffer, letting you work with formula as complex as you want. The only downside is that increasing the size of the expression buffer consumes scratch memory, so you may want to increase the scratch memory setting (see "Changing Scratch Memory Size (Macintosh OS 9)" on page 192).

To increase the size of the expression buffer, use the `formulabuffer` statement. This statement has one parameter, the number of bytes to make the formula buffer. For example, to make the formula buffer 12,000 bytes long, insert the following line into your procedure (you probably want to put this line into your **.Initialize** procedure, see ".Initialize" on page 1511):

```
formulabuffer 12000
```

This would allow formulas up to six times as complicated as would be allowed normally.

The `formulabuffer` statement is semi-permanent: it applies to all formulas in all databases until you quit Panorama or change the setting again. If you want to cancel expanded buffer and go back to the internal buffer, use `formulabuffer` statement with a size of 0:

```
formulabuffer 0
```

The expanded formula buffer is not created until the procedure is run. That means that if the complex formula is in the same procedure as the `formulabuffer` statement, you won't be able to compile the procedure because the buffer hasn't been expanded yet. The best way to eliminate this problem is to put the `formulabuffer` statement into your **.Initialize** procedure.

How Large Should the Buffer Be?

Most users have never encountered the Expression too complicated error message and have no need to expand the buffer. If you do encounter this error, you should probably start by modestly expanding the buffer, perhaps to 3000 to 4000 bytes. If you still have a problem you can expand it further until the problem disappears.

However, if your database allows users to enter formulas that are out of your control (for example a formula that is automatically generated by selecting options on a form or a web page), you may wish expand the buffer in advance to a very large size, perhaps 32000 bytes.

Arithmetic Formulas

Panorama formulas are very adept at performing arithmetic—from simple addition to complex financial calculations. Arithmetic formulas usually work just like the ones you learned about in high school. Panorama has seven arithmetic operators, as shown in this table.

symbol	operator
+	add
-	subtract
*	multiply
/ or ÷	divide
^	raise to power
integer divide \	
mod	modulo (remainder)

The ^ operator (press **Shift-6**) raises the operand on the left to the power specified on the right. For example the formula

```
2^3
```

means raise 2 to the third power (equivalent to the mathematical formula 2^3).

The \ operator converts both operands into integers and then divides them. The result is also an integer. For example,

```
19/5
```

is 3.8 (a normal division), but

```
19\5
```

is 3. Notice that because this is an integer operation, the result is not rounded.

The **mod** operator computes the remainder after an integer division. For example the result of the formula

```
19 mod 5
```

is 4, but

```
20 mod 5
```

is zero. The result of the **mod** operator will always be an integer between zero and the value of the operand on the right (in this case 0, 1, 2, 3, or 4).

Dividing by Zero

Dividing by zero is, of course, a no-no. If you do attempt to divide by zero, Panorama will display an alert reminding you of this arithmetical impossibility. Sometimes, however, you may want to defy mathematical reality and divide by zero without getting slapped on the wrist. For example, since formulas treat empty data cells as zeros, attempting to divide by a cell that hasn't been entered yet will result in a divide by zero error. To bypass the error message, use the `divzero(` function instead of the / operator. The `divzero(` function returns zero if you attempt to divide by zero. For example, using the formula

```
Price/Qty
```

can result in a divide by zero error if Qty field is empty, but

```
divzero(Price,Qty)
```

will not.

Overflow/Underflow Problems

A number is a number, right? Well, not quite. You may remember that Panorama actually stores two different kinds of numbers—fixed digit and floating point, with fixed digit numbers being further divided into 0, 1, 2, 3, and 4 digit precision (see "Numeric Data" on page 289). In a formula these differences may be important, since some numbers are too big or too small to be represented in some of the fixed point formats.

Formulas try to perform arithmetic using the final numeric type required for the answer. For example, if the result of a formula will be placed in a fixed 2 digit field, calculations will be performed in a fixed 2 digit format unless you force the formula to use another format. If the final destination is not a numeric field, arithmetic will be performed using floating point. Floating point is also used when the answer is not going to be stored in a field—for example formulas that are merged into auto-wrap text object (see "Displaying Formulas in Auto-Wrap Text" on page 602) or Text Display SuperObject (see "Text Display SuperObjects™" on page 608).

Since the internal format used for arithmetic can vary depending on the final destination of the answer, the same formula can give different results depending on where it is used. For example, the formula

```
1/4
```

gives the result 0.25 if the result is a floating point field, but 0 if the result is a fixed 0 digit field.

A more subtle problem can occur if an intermediate calculation causes an overflow, underflow, or loss of precision. Often this can be fixed by re-arranging the formula. For example, this formula for computing sales tax can have problems if the result will be stored in a 2-digit fixed field.

```
total*taxrate/100
```

If the tax rate is 6.5%, the intermediate result of the division is 0.065. But since 2-digit fixed point arithmetic is being used, this intermediate result will be rounded to 0.07, resulting in an incorrect calculation. You can fix this formula by doing the multiplication first.

```
(total*taxrate)/100
```

You can also fix this formula by forcing all the numbers to floating point using the float(function.

```
float(total)*float(taxrate)/float(100)
```

If all the operands are in the same numeric format, the formula will calculate the result using that format, in this case floating point.

If you don't want to worry about overflow/underflow problems one solution is simply to make all numeric fields floating point. Floating point fields take up slightly more RAM than fixed point fields, but for most databases the difference isn't critical.

Adding Line Item Fields

Line items are used for repeating items within a record (see "Repeating Fields (Line Items)" on page 262). Line item fields always end with a numeric suffix, for example Qty1, Qty2, Qty3, etc. Line items can be added up just like ordinary fields:

```
Qty1+Qty2+Qty3+Qty4+Qty5
```

You can also use the sum(function to add up line item fields:

```
sum("QtyΩ")
```

Using the sum(function is easier to type, and it is slightly faster than regular addition when used in the design sheet or a procedure. (Ordinary addition is faster than the sum(function when used in a **Formula Fill**.)

When you use the sum function, don't forget to include the quotes around the field name as shown above, and don't forget the Ω symbol (see "Special Characters" on page 1202). To learn how to perform calculations within line item fields see "Line Item Fields" on page 1198.

Warning: The sum(function is not compatible with the Design Sheet's **Spreadsheet Mode** (see "Spreadsheet Mode Calculations" on page 340). If you are using Spreadsheet Mode you must add up the items field by field (i.e Qty1+Qty2+Qty3…).

Basic Numeric Functions

These functions perform various mathematical operations. Each of these functions takes one or more numeric parameters and returns a numeric result.

Function	Reference Page	Description
abs(number)	Page 5010	This function returns the absolute (positive) value of the numeric parameter. In other words, negative numbers are converted to positive numbers while positive numbers remain positive.
divzero(numerator,denominator)	Page 5178	This function divides two numbers. However, unlike the / operator, the divzero(function does not care if you attempt to divide by zero. If you attempt to divide by zero, this function simply returns zero.
fix(number)	Page 5254	This function truncates a number to an integer. It always truncates towards zero. For example fix(-4.6) is -4, while int(-4.6) is -5. For positive numbers the int(and fix(functions are identical. Don't confuse this function with the fixed(function, which converts numbers from floating to fixed point format.
fixed(number)	Page 5255	This function forces a number to fixed point format, using the least number of digits possible. Since formulas usually perform this conversion automatically, you probably won't ever need this function. Don't confuse this function with the fix(function, which truncates a number to an integer but does not change the type of the data.
float(number)	Page 5256	This function forces a number to a floating point format. You may need to use floating point to get around overflow, underflow, and accuracy problems that can occur when using fixed point arithmetic.
int(number)	Page 5458	This function truncates a number to an integer. It always truncates towards negative infinity. For example int(-4.6) is -5, while fix(-4.6) is -4. For positive numbers the int(and fix(functions are identical.

Function	Reference Page	Description
max(number,number)	Page 5523	This function compares two numbers and returns the larger value. If you need to compare more than two numbers, you can nest this function within itself, for example max(a,max(b,c)).
min(number,number)	Page 5530	This function compares two numbers and returns the smaller value. If you need to compare more than two numbers, you can nest this function within itself, for example min(a,min(b,c)).
numsandwich(value,extra)		This function is similar to the sandwich function, but for numbers. If the value is zero the result is zero, but if the value is not zero the result is the value plus the extra. For example, this could be useful for calculating the size of an object plus a border. If the object size is zero, the border is omitted also, for example numsandwich(20,7) will be equal to 27, but numsandwich(0,7) will be equal to zero.
randominteger(startnum,endnum)		Returns a random integer value greater than or equal to the startnumber and less than or equal to the end number.
round(number,step)	Page 5682	This function rounds a number to the nearest step. You can use any value you want for the step: 1, 10, 0.5, whatever. For example, you could use the formula round(Quantity,12) to round the quantity to the nearest dozen. The quantity 16 will be rounded to 12; the quantity 20 will be rounded to 24. Since dates are treated as numbers (see "HTML Generating Functions" on page 1247) you can use this function round to the nearest week. Use a step value of 7 (7 days per week), for example round(Date,7).
rnd()	Page 5681	This function returns a random number between 0 and 1. Each time you use this function it will return a different number. If you need a random number in a different range just adjust the output of this function. For example, to get a random number between 1 and 10, use the formula int(1+10*rnd()). Notice that even though this function has no parameters, you must still include the empty parentheses after the function name.
sum("lineitemΩ")	Page 5816	This function adds up all the instances of a line item field within the current record. You must specify the name of the line item field followed by the Ω character (see "Special Characters" on page 1202). The whole thing must be surrounded by quotes, for example sum("QtyΩ"). This example is the same as the formula Qty1+Qty2+Qty3… but much easier to type! Warning: The sum(function is not compatible with the Design Sheet's Spreadsheet Mode (see "Spreadsheet Mode Calculations" on page 340). If you are using Spreadsheet mode you must add up the items field by field (i.e Qty1+Qty2+Qty3…).
zeroblank(number)	Page 5915	This function tells Panorama to store zero as an empty space. If the final formula result is not zero, this function has no effect. The zeroblank(function is handy when you want to leave the result of a calculation blank if one of the operands are blank. For example, if you use the formula zeroblank(Qty*Price), the result will be empty if either the quantity or price is empty.

Scientific Functions

These functions perform various log, trig, and exponential calculations. Each of these functions takes one or more numeric parameters and returns a numeric result.

The trig functions listed in this table normally use radians to measure angles (1 radian = 180/π degrees). In a procedure the `degree` statement may be used to temporarily switch Panorama's trig functions to use degrees instead of radians (see "DEGREE" on page 5155). The `radians` statement switches the mode back to radians (Panorama also switches back automatically when the procedure is finished). For example, the procedure below calculates the tangent of 30 degrees, not 30 radians.

```
degree
height=tan(30)
```

Calculations performed outside of a procedure always use radians (for example in a Text Display SuperObject). If you need to convert degrees into radians you can simply multiple the number of degrees by 180/π (see "Special Characters" on page 1202), for example **`tan(30*180/π)`**.

Function	Reference Page	Description
arccos(number)	Page 5030	This function calculates the inverse cosine of a number. The number must be between -1 and +1. The result is normally in radians, but may be in degrees if the degree statement has been used (see "DEGREE" on page 5155).
arccosh(number)	Page 5031	This function calculates the inverse hyperbolic cosine of a number. The number must be between 1 and ∞.
arcsin(number)	Page 5032	This function calculates the inverse sine of a number. The number must be between -1 and +1.
arcsinh(number)	Page 5033	This function calculates the inverse hyperbolic sine of a number.
arctan(number)	Page 5034	This function calculates the inverse tangent of a number. The result is normally in radians, but may be in degrees if the degree statement has been used (see "DEGREE" on page 5155).
arctanh(number)	Page 5035	This function calculates the inverse hyperbolic tangent of a number. The number must be between -1 and +1.
cos(number)	Page 5128	This function calculates the cosine of an angle. The angle is normally specified in radians, not degrees. To convert degrees to radians, divide by 180/π, which is 57.2958. For example cos(A*180/π) calculates the cosine of A, where A is in degrees. It is also possible to modify the action of Panorama to use degrees instead of radians for all trig functions, see "DEGREE" on page 5155.
cosh(number)	Page 5130	This function calculates the hyperbolic cosine of a number. The result will be a value between 1 and ∞.
exp(number)	Page 5202	This function raises **e** to a number. For example, the formula exp(10.2) is equivalent to $e^{10.2}$. Incidentally, e is a constant that is used in many mathematical formulas. It's approximate value is 2.71828.
fact(number)	Page 5214	This function calculates the factorial of a number. For example, the formula fact(4) is equivalent to 4! or 4*3*2*1. You can calculate the factorial of any integer from 0 to 170.
log(number)	Page 5492	This function calculates the natural logarithm (base e) of a number.
log10(number)	Page 5493	This function calculates the common logarithm (base 10) of a number.

Function	Reference Page	Description
sin(angle)	Page 5772	This function calculates the sine of an angle. The angle is normally specified in radians, not degrees. To convert degrees to radians, divide by 180/π, which is 57.2958. For example sin(A*180/π) calculates the sine of A, where A is in degrees. It is also possible to modify the action of Panorama to use degrees instead of radians for all trig functions, see "DEGREE" on page 5155.
sinh(angle)	Page 5774	This function calculates the hyperbolic sine of a number.
sqr(angle)	Page 5791	This function returns the square root of the number.
tan(angle)	Page 5841	This function calculates the tangent of an angle. The angle is normally specified in radians, not degrees. To convert degrees to radians, divide by 180/π, which is 57.2958. For example tan(A*180/π) calculates the tangent of A, where A is in degrees. (Note: The tangent of π/2 (90°) is ∞, which results in an overflow error.) It is also possible to modify the action of Panorama to use degrees instead of radians for all trig functions, see "DEGREE" on page 5155.
tanh(number)	Page 5843	This function calculates the hyperbolic tangent of a number. The result will be a value between -1 and +1.

Financial Functions

These functions calculate financial data, including loan payments, future value, and present value. They are designed to be compatible with the same functions in Microsoft Excel®. The financial functions are based on the following formula.

```
pv(1+rate)periods+payment(1+rate ×begin)×((1+rate)periods-1)/rate+fv=0
```

Function	Reference Page	Description
pmt(rate,periods,amount,fv,begin)	Page 5604	This function calculates the periodic payment required to pay off a loan. The rate is the interest rate of the loan per period. Periods is the term of the loan expressed in payment periods, for example 36 months for a three year loan that is paid monthly. Amount is the amount being borrowed. The fv (future value) and begin values are optional, and should usually be set to zero. For example, suppose you are taking out a 36 month loan of $20,000 to buy a car. If the annual interest rate is 13.5% (1.125% compounded monthly), what would the monthly payment be? `pmt( 0.135/12 , 36 , 20000 , 0 , 0 )` The monthly payment is $678.71.

fv(rate,periods,payment,pv,begin)	Page 5283	This function calculates the future value of an investment. Rate is the interest rate per period. Periods is the term of the investment, for example ten years or 48 months. The pv is the present value of the investment, for example the starting balance in a savings account. Begin should be either 1 or 0; 1 if the payments occur at the beginning of the period, 0 if the payments occur at the end of the period.
		For example, to calculate the final balance in a savings plan when you invest $500 per year for 10 years at 9% annual interest use the formula—
		`fv( 0.09 , 10 , -500 , 0 , 1 )`
		At the end of ten years you would have $8280.15. What if this savings plan already has $2000 in it at the time you start this 10 year savings program? The new formula would be—
		`fv( 0.09 , 10 , -500 , -2000 , 1 )`
		At the end of 10 years you would have $13,014.87.

pv(rate,periods,payment,fv,begin)	Page 5621	This function calculates the present value of an investment. Rate is the discount rate, periods is the periodic investment, and payment is the periodic payment. The fv is an optional lump sum at the end of the final period; use zero if there is no lump sum. Begin specifies whether payments are received at the beginning or end of each period—1 for beginning or 0 for end. Present value is a variation of the old theme that a bird in the hand is worth two…well, you know. It's better to get $1000 now instead of $1000 next year, but how much better? The present value computation puts a numeric value on time and money. For example, suppose you find an investment opportunity that promises to pay you $1,000 per year for the next 3 years. Assuming the current interest rate is 10% per year, how much are these payments worth right now? `pv( 0.1 , 3 , 1000 , 0 , 0 )` The computation shows that $3000 paid over 3 years is worth $2486 right now (assuming 10% interest).

Text Formulas

Formulas can work on text as well as numbers. Formulas can combine two or more pieces of text, extract a portion of a piece of text (for example the area code or last name), or even re-arrange the text. Formulas can also convert numbers into text and back again.

Programmers call a piece of text a **string**, referring to the fact that the text is made up of a string of characters. Since this is such a handy term we'll use it ourselves. So whenever you see the word **string** think "piece of text."

Where do strings come from? Most strings come from the database itself. Any text or choice field can be used as a string. You can also store strings in a variable (see "Variables" on page 1199), or put a string right into the formula itself (see "Constants" on page 1195).

Gluing Strings Together

The simplest operation that can be performed on two strings is sticking them together, also called **concatenation**. To glue strings together use the + operator. This operator attaches the string on the right to the end of the string on the left. For example the formula

```
"abc"+"def"
```

produces the result abcdef. To attach the word Mr. to the beginning of a last name field use the formula

```
"Mr. "+«Last Name»
```

(Of course, you better be sure everyone in the database is a man!).

You can use more than one + operator to stick several strings together at once. For example to combine separate first and last names into a single string using the format Last, First use this formula:

```
«Last Name»+", "+«First Name»
```

Another way to glue strings together is with the `sandwich(` function (reference page 5689). This function combines up to three items of text: a prefix, a suffix, and the root text. The prefix and suffix are slapped on the ends of the root, just like a sandwich. However, if the root is empty (sort of like a sandwich with no meat!) the prefix and suffix are also left off, just as you wouldn't bother to make a sandwich without any meat.

Let's revisit our previous example with the `sandwich(` function. The previous formula will work fine as long as there is a first name. But if the first name is empty, the formula will produce an extra comma, for example Jones, . The sandwich function can solve this problem:

```
«Last Name»+sandwich(", ",«First Name»,"")
```

If the First Name field contains a name, the `sandwich(` function will slap the prefix in front of the name (in this case the prefix is a comma and a space). But if the First Name field is empty, the sandwich(function will also leave off the prefix. All the formula will produce is the Last Name, with no extra comma and space.

The `rep(` function (reference page 5662) repeats an item of text by concatenating it to itself over and over. The number of times the item is repeated is specified by the second parameter, which must be an integer. For example, this formula will create twenty asterisks in a row:

```
rep("*",20)
```

This is exactly the same as the formula:

```
"********************"
```

The `rep(` function, however, is less prone to error, and the count can be changed easily or even vary dynamically. Here is a function which adds leading asterisks to a number so that there are always 15 characters.

```
rep("*",15-length(pattern(Amount,"$#,.##")))+pattern(Amount," $#,.##")
```

This formula is perfect for displaying numbers with an auto-wrap text object or Text Display SuperObject. The numbers will be padded with asterisks, for example ***** $4,983.45.

Functions for Taking Strings Apart

These functions return portions of a string. See also "Taking Strings Apart (Text Funnels)" on page 1214, "String Modification Functions" on page 1224, "Text Arrays" on page 1236 and "HTML Tag and Tag Parsing Functions" on page 1243.

Function	Reference Page	Description
firstline(string)		This function extracts the first line from the text.
firstword(string)		This function extracts the first word from the text (the text up to the first space).
lastline(string)		This function extracts the last line from the text.
lastword(string)		This function extracts the last word from the text (the text from the last space to the end).
left(string,len)		Extracts characters from the left edge of the text. For example left(text,2) extracts the leftmost two characters.
mid(string,len)		Extracts characters from the middle of the text. For example mid(text,6,4) extracts four characters starting with the sixth character.
nthline(string,num)		This function extracts the nth line from the text. For example nthline(text,4) extracts fourth line.
nthword(string,num)		This function extracts the nth word from the text. For example nthword(text,7) extracts seventh word.
right(string,len)		Extracts characters from the right edge of the text. For example right(text,7) extracts the rightmost seven characters from the text.
snip(string,startposition,count)		This function removes (snips!) one or more characters from the middle of an item of text. The startposition specifies the first character removed, the count is the number of characters to remove. (Note: This function requires the startposition to be a positive number.) If count is -1 then all the text from the start position to the end of the text is snipped, otherwise the count must be a positive number.
textafter(string,tag)		This function extracts the text after the tag. The tag many be one or more characters long. If the tag doesn't occur in the text then the entire original string is returned. For example textafter("someone@isp.net","@") will return isp.net.
textbefore(string,tag)		This function extracts the text before the tag. The tag many be one or more characters long. For example textbefore("someone@isp.net","@") will return someone. If the tag doesn't occur in the text then the entire original string is returned.
trim(string,len)		This function removes characters from the right edge of the text. For example trim(text,4) removes the last four characters from the text.
trimleft(string,len)		This function removes characters from the left edge of the text. For example trimleft(text,2) removes the first two characters from the text.

Taking Strings Apart (Text Funnels)

Sometimes you may have an item of text where you only need a portion of the text and want to strip off the beginning and or the end of the text. In addition to the functions in the previous section Panorama has a special tool for stripping off the ends of a text item. This tool is called a **text funnel**. Text funnels are powerful tools, however, many users find them a bit difficult to figure out. In recent years we've added many functions that can perform most of the operations that a text tool can perform. Before deciding to use a text funnel you may want to check out "Functions for Taking Strings Apart" on page 1213, "String Modification Functions" on page 1224, "Text Arrays" on page 1236 and "HTML Tag and Tag Parsing Functions" on page 1243.

A text funnel is used a bit differently than other Panorama functions and operators. The text funnel always follows the text item that is being "stripped." In a sense a text funnel has three parameters, the text item, start, and end. But as you can see below, these parameters are arranged quite differently than they are for other functions:

```
<text item>[<start>,<end>]
```

The first parameter, text item, is the item of text which will be stripped to get the final result. This may be a field, a variable, or an entire formula (as long as it produces a text item as its final result). If you use an entire formula you should put parentheses around the formula.

The second parameter, start, specifies the first character you want to include in the final output. For example if you want to strip off the first three characters the start should be 4 (because the 4th character is the first one we want to keep). If the starting position is past the end of the text all the text will be stripped out and the formula is left with an empty text item.

The third parameter, end, specifies the last character you want to include. For example, if you want to strip off everything after the 12th character, the end should be 12. If the starting position is after the ending position, all the text will be stripped and the formula is left with an empty text item.

The real trick in setting up text funnels is deciding what the start and end parameters should be. The following sections will describe several techniques for setting up these parameters.

Numeric Start and End Positions

The simplest way to specify starting and ending positions is with a number. Positive numbers are counted from the beginning of the original text item (1 is the first character in the original text item). Negative numbers are counted from the last character of the original text item (-1 is the last character).

Our first example removes the first character from the Notes field.

```
Notes[2,-1]
```

The next example does the exact opposite—it removes the last character from the Notes field.

```
Notes[1,-2]
```

By using the same number for the start and end a text funnel can strip out a single character. The procedure below uses the text funnel `[1,1]` to check to see if the first character of the phone number is a (. If so, it uses another text funnel to strip out the area code.

```
if Phone[1,1]="("
   AreaCode=Phone[2,4]
endif
```

A procedure can use a variable to pre-load the start and end positions. The procedure below will strip out everything starting with the phrase Private Notes Below ---.

```
local X
X=search(Notes,"Private Notes Below ---")
if X≠0
   PublicNotes=Notes[1,X-1]
else
   PublicNotes=Notes
endif
```

Specifying Numeric Length Instead of Position

An alternate form of text funnel allows you to specify the length of the text to be stripped out, instead of the ending position. This alternate form simply uses a semicolon instead of a comma:

```
<text item>[<start>;<length>]
```

The length specifies the number of characters from the starting position. A positive length means that the stripped text begins at the starting position and extends to the right. A negative length means that the stripped text begins at the starting position and extends to the left. The character at the starting position is always included (unless the length is zero).

Let's look at two examples of this technique. The first extracts the area code from a long distance phone number.

```
Phone[2;3]
```

The next example strips out the local phone number (the last 8 characters).

```
Phone[-1;-8]
```

If the original text item is too short to fulfill the request the text funnel will take whatever it can get. For example, if the phone number is only 3 characters long, the value in LocalNumber will be 3 characters long.

Start/End Positions by Character Matching

The previous section described how to strip out text by absolute numeric position within the original text (for example from character 3 to character 8). Another technique is to specify not the absolute position, but the character value where stripping should begin and/or end. For example instead of telling the text funnel to strip off everything before position 5, you tell the funnel to strip off everything before the first $ character, or everything after the last % character. The text funnel scans the original text looking for a matching character, and then strips the text accordingly.

To specify a starting or ending position by character matching, simply supply a character instead of a number. For example, suppose you had a field named Line that contained data like this:

```
X2245A Tape Cartridge $22.95
```

To extract just the price from Line you could use this text funnel.

```
Line["$",-1]
```

This formula will take Line and strip off everything in front of the first dollar sign. In our example this will be the value $22.95. If there is no dollar sign in Line then the result will be empty text ("").

Notice that the output of a text funnel is always a text item, not an actual number. If you wanted to convert this to a number you would have to remove the dollar sign with an additional text funnel and use the `val(` function (reference page 5886).

A text funnel can use a character value for either the starting or ending position, or both. Here is an example that extracts the hour from the time by stripping off everything after the first colon:

```
Time[1,":"]
```

Both of these examples are developed further in the next section.

Cascading Text Funnels

The examples in the previous section both have a problem: they don't strip off enough text. The first example strips off the price but leaves the dollar sign (for example $45.67). The second example strips the hour from the time but leaves the colon (for example 9:). These problems can be solved by using two text funnels in a row.

Adding a second text funnel is easy—just enter it after the first funnel. This example strips off the $ symbol from the beginning of the price using a regular numeric position text funnel.

```
Line["$",-1][2,-1]
```

The table below shows how some typical data would be processed by this formula.

Original Data	After `["$",1]`	After `[2,-1]`
X2245A Tape Cartridge $22.95	$22.95	22.95
AF8899 Data Casette $7.80	$7.80	7.80
XB3 Head Cleaner $19.50	$9.50	9.50

This example strips off the hour from the time—including the pesky extra colon.

```
Time[1,":"][1,-2]
```

Once again, the table shows how the data is processed by each text funnel.

Original Data	After `[1,":"]`	After `[1,-2]`
9:42 AM	9:	9
3:07:12 PM	3:	3
11:23 AM	11:	11

These examples show two text funnels cascaded together, but there is no limit to the number of text funnels you can use in a row. Each funnel chops away at the text until you have just the text you want. Usually the best approach to developing a series of cascaded funnels is to develop one funnel at a time. Make sure one funnel really does what you want and expect it to before adding the next one.

Character Matching in Reverse Gear

If the character to be matched is preceded by a minus sign the text funnel will match the last instance of the value in the original text instead of matching the first.

The example below strips out the year from an appointment. The formula assumes that there is a date in the format mm/dd/yy somewhere in the text item. The funnel will attempt to match up the last / symbol in the original text.

```
Year="19"+Appointment["-/",-1][2;2]
```

Here is how the data is processed.

Original Data	After `["-/",-1]`	After `[2;2]`
Lunch with Bob 3/4/01	/01	01
call Joan 4/2/99 3PM	/99 2PM	99
10/7 L's Birthday	/7 L's Birthday	7
call Ted	call Ted	al

The last two lines above shows the hazards of making faulty assumptions. Neither line contains a valid `mm/dd/yy` date. The result in this case is a bogus year. Unfortunately there is no magic pill fix for this kind of problem. As a programmer you must think of, check for and process every possible option. If you absolutely know that there will be a date in the text item, fine. If not, you'll have to write a more complicated procedure to check for a properly formatted date before you strip out the year. Here's an example of a more robust procedure.

```
if Appointment notmatch "*/*/*"
message "Sorry, the appointment has no year!"
   stop
endif
Year="19"+Appointment["-/",-1][2;2]
```

This procedure could still be fooled—for example data containing two dates would trip it up. Designing a completely foolproof procedure is left as an exercise to the reader.

Stripping Out Individual Words

One of the most common needs is to strip out a single word at the beginning, middle or end of a text item. This is easily done by using a space as the matching character value. You'll need to look at the formulas in this section very carefully. Don't confuse a space (" ") with an empty text item (""). They're not the same thing. (For clarity, the samples below showing how the data is processed use space to show a space whenever it is at the beginning or end of a text item. For example, spacenow means space followed by now.)

Here's a formula that extracts the first word from an item of text by stripping off all the rest of the words.

```
Original[1," "][1,-2]
```

Here's how this formula would process several sample text strings.

Original Data	After `[1," "]`	After `[1,-2]`
Now is the time	Now space	Now
Boston reports 23 degrees	Boston space	Boston
Apple stock up 5 points	Apple space	Apple

This next formula does the exact opposite: it strips off the first word, leaving the rest of the words.

```
Original[" ",-1][2,-1]
```

Here is how this formula would process the same sample strings as before.

Original Data	After `[1," "]`	After `[1,-2]`
Now is the time	space is the time	is the time
Boston reports 23 degrees	space reports 23 degrees	reports 23 degrees
Apple stock up 5 points	space stock up 5 points	stock up 5 points

You can cascade these two text funnels to produce a formula that extracts the 2nd word from the original text, stripping off the rest.

```
Original[" ",-1][2,-1][1," "][1,-2]
```

Here is how this formula would process the same sample strings as before.

Original Data	After `[" ",-1]`	After `[2,-1]`	After `[1," "]`	After `[1,-2]`
Now is the time	space is the time	is the time	is space	is
Boston reports 23 degrees	space reports 23 degrees	reports 23 degrees	reports space	reports
Apple stock up 5 points	space stock up 5 points	stock up 5 points	up space	up

This process can be repeated indefinitely. However, a better approach is probably to use the `array(` function with space as a separator character. See reference page 5036 to learn more about this function.

Here's a simple formula that extracts the last word from a text item, stripping off the earlier words (if any).

```
Original["- ",-1][2,-1]
```

Here's how this formula would process several sample text strings.

Original Data	After `["- ",-1]`	After `[2,-1]`
Now is the time	space time	time
Boston reports 23 degrees	space degrees	degrees
Apple stock up 5 points	space points	points

A close examination will show that this is exactly the same as the first example but with an extra minus sign to specify the last space instead of the first space.

Multiple Matching Characters for Start/End Position

Sometimes you may need to use multiple character values to specify the starting or ending position of a text funnel. Any one of these character values will match up with the original text. For example, a sentence may end with a period, a question mark, or an exclamation mark. To use more than one matching character simply list each character separated by commas. Here is an example that extracts the first sentence from a letter. All the text after the first sentence is stripped off.

```
Letter[1,".,?,!"]
```

You can include a comma as one of the character values. This example extracts everything up to the first semicolon, comma, or colon. All the text after that point is stripped off.

```
Description[1,";,,,:"]
```

If you use alphabetic values, don't forget that upper and lower case are separate values, even for the same letter. This example extracts `am` or `pm` from a text item.

```
Appointment["a,p,A,P";2]
```

It's also possible to specify a range of matching characters, for example `0` through `9` or `A` through `Z`. To specify a range the starting and ending characters must be separated by a dash, for example `"0-9"`. The range will include all characters between the two characters on the ASCII table (see "Characters and ASCII Values" on page 1230.)

Here is an example that extracts the frequency from a radio station. The call letters are stripped off.

```
Station["0-9",-1]
```

Here's how this formula would process several sample text strings.

Original Data	After `["0-9",-1]`
KFI 640 AM	640 AM
KLSX 97.1 FM	97.1 FM
KFAC 105.1 FM	105.1 FM
KROQ 106.7 FM	106.7 FM

A text funnel can combine multiple character ranges, or combine a range with one or more separate character values. The next example strips off everything before the first number, or before the first dollar sign (whichever comes first).

```
Line["0-9,$",-1]
```

Here's how this formula would process several sample text strings.

Original Data	After `["0-9,$",-1]`
Tape Cartridge $22.95	$22.95
Data Cassette 7.80	7.80
XB3 Cleaner $19.50	3 Cleaner $19.50

The last line shows a possible pitfall of this text funnel. Text funnels rely on consistent patterns in the data. If there isn't a pattern you can identify accurately, you won't be able to design a funnel to strip the text apart reliably. In this case a more reliable pattern would be to notice that the price is always the last word of Line, so the text funnel below will strip off the price reliably.

```
Line["- ",-1][2,-1]
```

Be sure to test your text funnels with a wide variety of sample data to make sure you have identified a consistent pattern.

As mentioned in the previous section, putting a minus sign in front of the character value tells the text funnel to find the last matching character, instead of the first. This works for character ranges too. This example extracts the item name from Line by stripping off everything after the last letter.

```
Line[1,"-A-Z,a-z"]
```

Here's how this formula would process our sample text strings.

Original Data	After `[1,"-A-Z,a-z"]`
Tape Cartridge $22.95	Tape Cartridge
Data Cassette 7.80	Data Casette
XB3 Cleaner $19.50	XB3 Cleaner

Although this example has two ranges (A-Z or a-z) only one minus sign is needed (at the beginning). If the first character is a minus sign, the text funnel will always look for the last matching character in the original text.

Non-Matching Character for Start/End Position

The previous examples have all used one or more characters that must match a character in the original text item. By using the ≠ symbol (see "Special Characters" on page 1202) you can specify that the text funnel should begin (or end) with the first character (or characters) that does not match. For example, you might want to match with the first character that is not a number, or the last character that is not a space.

An example should make this clearer. Suppose you have imported some numbers that have one or more asterisks in front of them, and you want to strip off the asterisks. The text funnel in this formula will set the starting position to the first character in the original text that is not an asterisk.

```
Imported["≠*",-1]
```

Here's how this formula would process some sample text strings.

Original Data	After `[">*",-1]`
****23.67	23.67
***782.12	782.12
*****2.98	2.98

You can use this feature to strip off leading spaces.

```
Name["≠ ",-1]
```

Here's how this formula would process some sample text strings (leading spaces are shown as space for clarity).

Original Data	After `["› ",-1]`
space space Jeff Nance	Jeff Nance
space Williams	Williams

(An easier way to strip leading and trailing spaces is to use the `strip(` function, which is designed for that purpose. See reference page 5801 for more information about this function.)

The example below specifies that the starting position should be the first character that is not a letter, not a comma, and not a space. It extracts the zip code or Canadian postal code from an address.

```
CityStateZip["≠A-Z,a-z,,, ",-1]
```

Here's how this formula would process some typical addresses.

Original Data	After `[">A-Z,a-z,,, ",-1]`
Fullerton, CA 92831	92831
Kamloops, BC 3J2 X7G	3J2 X7G

The astute reader may have realized that a simpler text funnel can do the same job, `["0-9",-1]`. Of course it would not have illustrated the ≠ feature. The moral of the story is: watch out for college solutions when a grade school solution may work just as well!

A text funnel that uses the ≠ symbol can also work in reverse gear, so that it specifies the last character that does not match, instead of the first. The ≠ symbol must be first, and then the - symbol. For example, here is yet another formula for extracting the price from Line.

```
Line["≠-0-9,.",-1][2,-1]
```

Here's how this formula would process some of our favorite sample text strings.

Original Data	After `["›-0-9,.",-1][2,-1]`
Tape Cartridge $22.95	22.95
Data Cassette 7.80	7.80
XB3 Cleaner $19.50	19.50

Unlike some of our previous examples, this formula does not rely on a $ symbol or a space in front of the price, and it does not choke if there is a number in the item description.

Limitations of Text Funnels

Unlike Humpty Dumpty, text items are easy to put together but hard to take apart intelligently. Text funnels are a powerful tool, but they do have limitations. One limitation is that, by themselves, they can only work with one character at a time. If you want to start stripping text with the word fax or P.O. Box a text funnel can't do it on its own. You'll have to combine the funnel with the `search(` function for jobs like this (see reference page 5707).

The most important limitation of text funnels is that they cannot work reliably if there is not a single consistent pattern in the data. If the data has no pattern at all, you're out of luck (short of re-keying data). If the data has two or more patterns you'll need to isolate each pattern and process each one with a separate text funnel. One way to do this is with the `?(` function (see reference page 5008). This formula extracts the local phone number from a complete phone number. If the complete phone number starts with (, the formula uses a text funnel that strips out the area code, otherwise the local number starts with the first character.

```
?( Phone[1,1]="(" , Phone[7;8] , Phone[1;8] )
```

Here's how this formula would process some of our favorite sample text strings.

Original Data	After `?(Phone[1,1]="(",Phone[7;8],Phone[1;8])`
(714) 555-1212	555-1212
852-9632	852-9632
(562) 492-1438 ext 23	492=1438

Don't be afraid to combine text funnels with other functions and statements. Some functions that are often useful with text funnels include `?(` (see reference page 5008), `length(` (see reference page 5467), `strip(` (see reference page 5802), `stripchar(` (see reference page 5802), `search(` (see reference page 5707), `replace(` (see reference page 5665), and `array(` (see reference page 5036).

String Testing Functions

These functions return information about the content of a string.

Function	Reference Page	Description
length(string)	Page 5467	This function counts the number of characters in a string. The result is an integer. If the string is empty, the result will be zero.
linecount(string)		This function counts the number of lines in the text.
rangematch(string,range)		This function checks text to see if the text matches the specified range. The range must be a series of character pairs, for example AZ for upper case alphabetic characters, AZaz for upper and lower case, 09 for numeric digits, etc. If it matches the function returns true, if it doesn't match, it returns false. For example, rangematch(Address,"AZaz09 ") will return true if the address contains only letters, numbers and spaces, false if it contains any other characters.
search(string,phrase)	Page 5707	This function searches through a string looking for a word or phrase. If the search is successful, the function returns the position of the phrase within the string, otherwise the function returns zero. For example, the formula search(Name,"Dr.") will return a non-zero value (usually 1) if the name contains Dr., or zero if it does not.

sizeof(name)	Page 5777	This function calculates the amount of memory used by a field cell or a variable. Name is the name of the field or variable that you want to calculate the size of. The function returns the number of bytes of memory used by the variable or field cell.
		The sizeof(function can be used to decide if a numeric or date field is empty or not. The example procedure shown below selects all the records with no price (not the same as records with a price of zero).
		`select sizeof(Price)=0`
		Another use for the sizeof(function is to check if a variable is taking up too much scratch memory (see "Changing Scratch Memory Size (Macintosh OS 9)" on page 192). This example checks to see if the variable importLetter is more than 500 bytes long. If it is, the procedure clears the variable.
		`if sizeof(importLetter)>500`
		`  importLetter=""`
		`endif`

wordcount(string)		This function counts the number of words in the text.

String Modification Functions

These functions modify the contents of a string. Usually the string is actually a database field. Remember, to use a database field as a string parameter simply use the name of the field, for example upper(Name).You'll often want to use these functions to modify the existing data in a field. For example, you might want to convert all company names to upper case. To convert existing data use the **Formula Fill** command in the Math Menu (see "Filling a Field with a Formula" on page 453). This command calculates the formula over and over again—once for each selected record.. Note: In addition to the functions listed here you will also find methods for modifying strings in "Functions for Taking Strings Apart" on page 1213, "Taking Strings Apart (Text Funnels)" on page 1214, "Text Arrays" on page 1236 and "HTML Tag and Tag Parsing Functions" on page 1243.

Function	Reference Page	Description
applescriptstring(string)		This function converts the string into an AppleScript string literal. It surrounds the text with double quote characters, and escapes any double quote and/or backslash characters within the text. This function is designed to be used with the executeapplescript statement.
connect(prefix,connector,suffix)		This function appends a prefix and suffix together with a connector in between. If either the prefix or the suffix is missing then the connector will also be left out. For example, connect(City,", ",State) combines the city and state with a comma and space in between, but if either the city or state is missing then the comma and space will also be left out. See also the sandwich(and yoke(functions in this table.
crtovtab(string)		This function converts carriage returns (ASCII 0x0D) into vertical tabs (ASCII 0x0B). Some programs (including Panorama) will convert vertical tabs into carriage returns when importing, allowing individual data cells to contain carriage returns.
extract(text,separator,item)	Page 5211	This function extracts a single data item from a text array. This function is almost identical to the array(function. The extract(function is excellent for extracting a word, line or phrase from a larger text item. It can also be used to count the number of items in the array. There are three parameters: text, separator and item. Text is the item of text that contains the data you want to extract. Separator is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character. Item is the number of the data item you want to extract. The first item is item 1, the second is item 2, etc. Using an item number of -1 tells the extract(function to count the number of data items in the array. This is similar to the arraysize(function. In this case the extract(function will return a number, not text. If the item parameter is 1 or greater, this function returns an item of text from the array. Only the item itself is returned, the separator characters on each end are not included. If the item does not exist (for example if you ask for item 12 from a 7 item array) the function will return empty text ("").
fixedwidth(string,width)		This function makes the text a fixed width. If the text is shorter than the specified width, it is padded with spaces. If it is longer than the specified width, it is cut off.
fixedwidthright(string,width)		This function makes the text a fixed width. If the text is shorter than the specified width, it is padded with spaces on the left (i.e. the text is right justified). If it is longer than the specified width, it is cut off on the left.
lower(string)	Page 5516	This function converts all of the letters in the string to lower case. For example, the formula lower(Terms) will convert NET 30 to net 30, or C.O.D. to c.o.d. See also the upper and upperword functions.
onespace(string)		This function removes any extra spaces between words, so that there is exactly one and only one space between each word.

Function	Reference Page	Description
onewhitespace(string)		This function removes any extra whitespace between words, making sure that there is one and only one space between each word. Other whitespace characters (carriage returns, tabs) are converted to spaces and removed if there is more than one between words.
quoted(string)		This function surrounds the supplied text with double quote characters. If the text contains any double quotes they will be doubled, making this a legal string constant. This function is typically used in conjunction with the execute and executeapplescript statement.
randomletter(option)		This function returns a random letter. If the option is "U" then the letter will be between A and Z. If the option is "L" then the letter will be between a and z. If the option is any other value then the result may be either A-Z or a-z.
rep(string,count)	Page 5662	This function replicates a string over and over. The number of replications is specified by the count (a number). This function is handy for creating a long repeating string. For example to create a string containing twenty asterisks in a row, use the formula rep("*",20). The count does not have to be a constant, but it must be an integer.
replace(string,search,replace)	Page 5665	This function searches for a word or phrase within a string and if found, replaces it with a new word or phrase. The first parameter is the string that may contain the word or phrase. Usually this parameter is a database field. The second parameter is the word or phrase to search for. The third parameter is the new word or phrase. For example, to replace Corporation with Corp. in the Client field, use the formula replace(Client,"Corporation","Corp."). To use this formula to replace the data in the database, use the **Formula Fill** command. (For a simple replace case like this, however, it is easier to use the **Change** command. The replace(function is useful when you want to perform other transformations in addition to the replace.)

replacemultiple(string,search,replace,sep)	Page 5667	This function searches for a set of words or phrases and replaces them with another set of words or phrases. It is similar to the replace(function, but can replace a whole set of items at once. The string parameter must contain the text that contains the words or phrases you want to replace. The search parameter contains a list of words or phrases to search for. The items in this list must be separated by the sep character. Here's an example that uses comma as the separator.
		
		`"Drive,Lane,Avenue,Boulevard"`
		
		The replace parameter contains another list of words or phrases. These must use the same separator character and be in the same order as the search parameter. Here's another example.
		
		`"Dr,Ln,Ave,Blvd"`
		
		Putting it all together, here's an example that inserts abbreviations in an address.
		
		`replacemultiple(`
		`  Address,`
		`  "Drive,Lane,Avenue,Boulevard",`
		`  "Dr,Ln,Ave,Blvd",`
		`  ",")`
		
		In this example we've separated each parameter onto a separate line, but this is not necessary. Also, keep in mind that you can use any character as a separator, not just a comma.

Function	Reference Page	Description
sandwich(prefix,root,suffix)	Page 5689	The sandwich(function assembles a text item from three smaller text items. The prefix and suffix are slapped on the ends of the root, just like a sandwich. However, if the root is empty, the prefix and suffix are also left off (the result is an empty text item), just as you wouldn't make a sandwich without any meat. Suppose you have a database with names and titles, and you want to display this information in a report with the titles surrounded by parentheses. The formula below could be used with an auto-wrap text object or Text Display SuperObject. `Name+sandwich(" (",Title,")")` If the person has a title it will appear in parentheses like this: Steve Johnson (Sales Mgr). If they don't have a title then no parentheses will appear. The sandwich(function is useful any time you have optional data items combined together with punctuation in between. See also the connect(and yoke(functions in this table.
strip(text)	Page 5801	This function strips off leading and trailing blanks and other whitespace (carriage returns, tabs, etc.) This function has one parameter, the item of text that you want to strip. The function removes blanks at the beginning or end of the text, but does not affect blanks in the middle of the text. It also removes carriage returns, tabs, or any character with an ASCII value less than 32.
stripchar(text,range)	Page 5802	This function removes characters you don't want from a text item. You specify exactly what kinds of characters you want and don't want included in the final output. Text is the item of text that you want to strip. Range specifies what kinds of characters you want to keep and what kinds of characters you want to strip away. The range consists of one or more pairs of characters. Each pair specifies a set of characters you want to keep. For example, the pair AZ means that you want to keep the characters from A to Z. For alphanumeric characters the set is pretty obvious. For other types of characters you should check an ASCII chart (see "Characters and ASCII Values" on page 1230). For example the pair #& specifies a set of four characters: #, $, % and &. You can use the **ASCII Chart** wizard to try out your character ranges, see "Showing Character Ranges with the ASCII Wizard" on page 1234. If a pair consists of the same character repeated twice in a row, the set is just that single character. For instance the pair ## means you want to keep one character: #. The range may consist of several pairs put together. For example the range AZaz09.. consists of four pairs, and specifies that all letters, numbers, and periods will be kept, with all other characters stripped away. One handy use for this function is to quickly check if a field or variable contains any inappropriate characters. If a field or variable changes when you run it through the stripchar(function it must contain characters that are not part of the specified range.
striphtmltags(text)		This function removes all HTML tags from the text.
stripprintable(text)		This function removes any non-displayable characters from the text.
striptoalpha(text)	Page 5805	This function removes everything but alphabetic letters from a text item. Everything else (numbers, spaces, punctuation, non-English letters, etc.) will by removed from the text. One handy use for this function is to quickly check if a field or variable contains all alphabetic characters. If a field or variable changes when you run it through the striptoalpha(function it must contain non-alphabetic characters.

Function	Reference Page	Description
striptonum(text)	Page 5807	This function removes everything but numeric digits from a text item. Everything else (letters, spaces, punctuation, non-English letters, etc.) will by removed from the text. One handy use for this function is to quickly check if a field or variable contains all numeric digits. If a field or variable changes when you run it through the striptonum(function it must contain non-numeric characters.
upper(string)	Page 5880	This function converts all of the letters in the string to upper case. For example, the formula upper(Terms) will convert net 30 to NET 30, or c.o.d. to C.O.D. See also the lower(and upperword(functions.
upperword(string)	Page 5881	The upperword(function converts the first letter of each word in the string to upper case, and all other letters to lower case. For example the formula upperword(State) will convert new york to New York, or will convert VERMONT to Vermont. See also the lower and upper functions.
vtabtocr(string)		This function converts vertical tabs (ASCII 0x0B) into carriage returns (ASCII 0x0D). Some programs (including Panorama) will convert vertical tabs into carriage returns when importing, allowing individual data cells to contain carriage returns. After the import you can use this function to turn the vertical tabs back into carriage returns.
yoke(prefix,joiner,suffix)		This function appends two text items (prefix and suffix) together. If both are non-blank, a joiner is placed in between. If either (or both) is blank, the joiner is not used. In some ways this is the reverse of the sandwich(function.

Converting Between Numbers and Strings

These functions convert numbers into strings and strings into numbers.

Function	Reference Page	Description
asc(string)	Page 5060	This function converts the first character of the string into a number based on the ASCII value of the character. For example the formula asc("Y") returns the value 89, while asc("Z") returns the value 90. See also the chr(function.
chr(number)	Page 5099	This function converts a number into a single character of text based on the ASCII value of the number. The number should be an integer between 0 and 255. For example, the letter A has an ASCII value of 65, while the letter B is 66. You can create special characters with this function; TAB is 9 and RETURN is 13. See also the asc(function.
dollarsandcents(number)		This function converts a number to text formatted as dollars and cents (for example 98123.45 becomes Ninety eight thousand one hundred twenty three dollars and 45 cents).
exportcell(value)	Page 5209	This function converts a value into text without any special formatting. For numeric values this function is the same as the str(function (see below). The advantage of this function is that it works with any kind of value - text, numeric or date. Use this function when for some reason you don’t know what kind of data you need to convert.
hex(string)		Converts text with hex characters into a number. For example the value of hex("0C2") is 194.
hexbyte(number)		Converts a number to text formatted as a two digit hexadecimal number. For example the result of hexbyte(68) is 44.
hexlong(number)		Converts a number to text formatted as a eight digit hexadecimal number. For example the result of hexlong(68) is 00000044.

Function	Reference Page	Description
hexstr(number)		Converts a number to text formatted as a hexadecimal number. For example the result of hexstr(68) is 00000044. This function can also accept binary values with more than 4 bytes (see "Raw Binary Data" on page 1296).
hexword(number)		Converts a number to text formatted as a four digit hexadecimal number. For example the result of hexword(68) is 0044.
money(number)		Converts a number to text, formatted with commas every three digits and two digits after the decimal point (for example 98,123.45).
pattern(number,string)	Page 5599	This function converts a number into text, using the string as an output pattern. For example the formula pattern(Price,"$#.,##") will convert the price 3458.23 into the string $3,458.23. The pattern adds the $ and the comma. For more information on numeric output patterns see "Numeric Output Patterns" on page 290.
radix(radix,text)	Page 5626	This function converts a text item containing a hex, octal, or binary number into a standard Panorama number (decimal). See "NON DECIMAL NUMBERS" on page 5543 for background information on hex, octal and binary numbers. Radix is the base for the numbering system you are converting from. Legal radix values are 2, 4, 8, 16 or 32. Or you can specify the radix as "binary" (same as 2), "octal" (same as 8) or "hex" (short for hexadecimal, same as 16). Text is a text item that contains the non-decimal number you want to convert. This function normally returns an integer that contains the decimal (base 10) number corresponding to the hex, octal, or binary number input to the function. If the radix is hex and there are more than 8 digits in the input text, or if the radix is binary and there are more than 32 digits, this function will return a raw binary value instead of a number. This binary value may be of unlimited length. Like all binary values, it cannot be calculated with, but should be handled as a text item.

radixstr(radix,number)	Page 5628	This function converts a number into a text item containing the equivalent hex, octal, or binary number. See "NON DECIMAL NUMBERS" on page 5543 for background information on hex, octal and binary numbers. Radix is the base for the numbering system you are converting from. Legal radix values are 2, 4, 8, 16 or 32. Or you can specify the radix as "binary" (same as 2), "octal" (same as 8) or "hex" (short for hexadecimal, same as 16). Number is the number you want to convert to hex, octal, or binary. If the radix is 2, 16, "binary", or "hex" the number can be a raw binary data (text) value. This function returns a text item that contains the hex, octal, or binary number equivalent to the number (or binary data) passed to the function. The first example converts the decimal value 256 to hexadecimal.
		
		`radixstr(16,256)`
		
		This function will calculate that 256_{10} is 100 hex.
		
		Here is another example:
		
		`radixstr("binary",5)`
		
		This will calculate that 5_{10} is 00000000000000000000000000000101 binary.

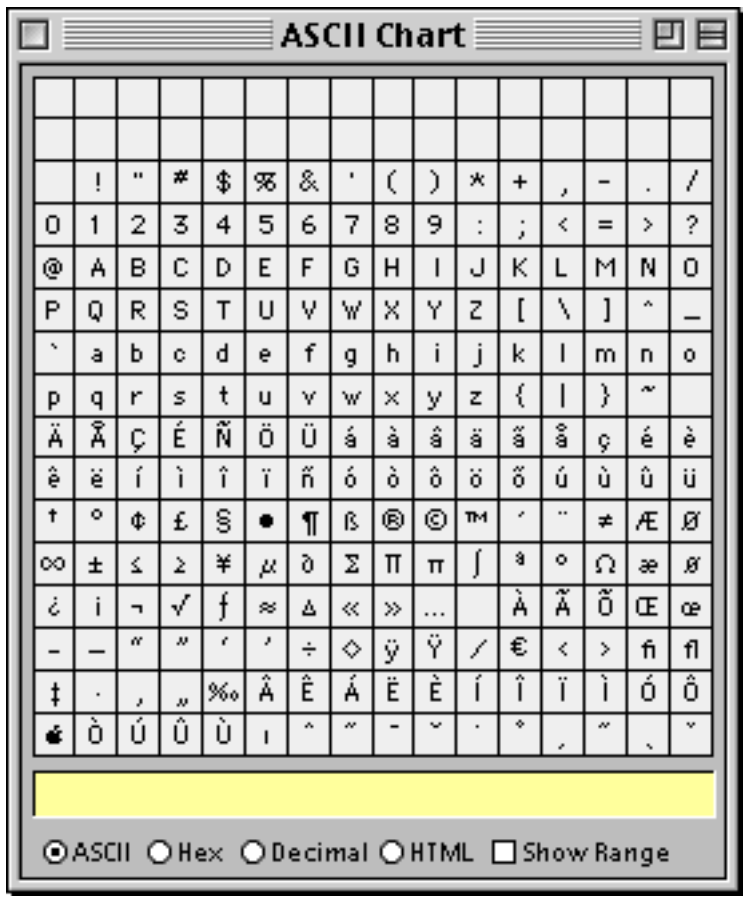
scientificnotation(number)		Converts a number to text, formatted in scientific notation with three places after the decimal point (for example 9.812e+4).
str(number)	Page 5799	This function converts a number into text without any special formatting. If you want to format the number (add commas, set # of digits, etc.) use the pattern(function.

Function	Reference Page	Description
val(string)	Page 5886	This function converts a string into a number. The string must start with one or more numeric digits. Everything after the first non-numeric character will be ignored. For example, the formula val(Address) will return the number 731 if the address is 731 N. Miller St.
zbpattern(number,pattern)		This function displays a number using a pattern. Unlike the normal pattern(function, the zbpattern(function will output "" if the number is zero. (Note: zb is short for zeroblank.)
exportcell(field)	Page 5209	This function takes any database field and converts it to text, using the appropriate pattern if one has been defined in the design sheet. Field is the name of the field to be converted to text. The function always returns a text type data item. The power of the exportcell(function is that it does not require you to know what type of data you are exporting. It simply takes whatever kind of data is in the field (text, number, date, whatever) and converts it into text.

Characters and ASCII Values

Just as molecules are built from atoms, text is built from characters. And like an atom which can be divided into electrons, protons, and neutrons (among others), characters also have an internal structure. Just as with atoms, the internal structure of characters can usually be ignored, and you may want to skip the following section if you are a beginner. Sometimes however, knowledge of the internal structure of characters can be very helpful.

On most computer systems there are 256 possible characters. (Some Japanese and Chinese systems allow thousands of characters, however Panorama does not currently support this.) Each character has a number from 0 to 255. Of these 256 characters, about 200 are associated with symbols (letters, digits, punctuation, etc.). For example, the symbol for the letter A is represented by character number 65. You can use the **ASCII Chart** wizard (in the Developer Tools subfolder of the Wizard menu) to see a complete list of all 256 characters and their symbols.



The numbers have not been assigned to symbols arbitrarily, but have been assigned using a system called ASCII. The number associated with a character is called the ASCII value of the character. (For you techno-weenies, ASCII stands for American Standard Computer Interchange Interface.) If you look at the ASCII table on the next page you'll notice that the characters with ASCII values from 0-31 have no symbols. These characters are used for special keys like return, tab, and enter. ASCII value 32 is the space character, then we have some punctuation. ASCII values 48 through 57 are the numeric digits 0 through 9, in order. ASCII values 65-90 are the upper case letters A through Z, in alphabetical order. ASCII values 97-122 are the lower case letters a through z, again in alphabetical order.

Panorama uses the ASCII values of characters when it compares two text items to see which is larger or smaller. Since the ASCII value of B (66) is greater than the ASCII value of A (65), the text item B is "larger" than A. However, the ASCII value of a (97) is greater than B (66), so the text item a is "larger" than B. You have to watch out for this problem whenever you compare text that is a mixture of upper and lower case.

Working with Character Values

Usually it's not necessary to worry about the numeric value of a particular character—you can just think of it as a character. However, if you want to perform any kind of math on the character itself it is necessary to convert the character in to a number. For example you can add one to a character value to get the next character value (A → B → C etc.). Or you can calculate the number of characters between two characters.

Panorama has two special functions that allow you to work with character values directly. The `asc(` function converts a character to its ASCII value. The `chr(` function converts an ASCII value to the corresponding character.

The following example procedure asks the user to enter a range of characters, for example A-F. It uses the asc(function to convert the characters into the corresponding ASCII numeric values, then calculates the number of characters in the range.

```
local LetterRange,StartLetter,EndLetter,LetterCount
LetterRange=""
gettext "Enter character range:",LetterRange
StartLetter=LetterRange[1,1]
EndLetter=LetterRange[-1,-1]
LetterCount=abs(asc(EndLetter)-asc(StartLetter))
message LetterRange+": "+pattern(LetterCount+1,"# character~")
```

If the person enters A-F the procedure will display A-F: 6 characters.

The next example procedure is similar but actually displays a list of the characters in the range. It uses the chr(function to convert the numbers back into characters.

```
local LetterRange,StartLetter,EndLetter
local LetterCount,LetterBump,Letters
LetterRange=""
gettext "Enter character range:",LetterRange
StartLetter=asc(LetterRange[1,1])
EndLetter=asc(LetterRange[-1,-1])
LetterCount=EndLetter-StartLetter
LetterBump=LetterCount/abs(LetterCount)
Letters=""
loop
   Letters=Letters+chr(StartLetter)
   StartLetter=StartLetter+LetterBump
while StartLetter≠EndLetter
message LetterRange+": "+Letters
```

If the person enters A-F the procedure will display A-F: ABCDEF. If the person enters F-A the procedure will display F-A: FEDCBA.

Warning: Don't confuse the `asc(` and `chr(` functions with the `val(` and `str(` functions. The `asc(` and `chr(` functions convert single characters based on their ASCII values. The `val(` and `str(` functions convert entire text items based on the number the characters spell out. For example `asc("4")` is 52, because 52 is the ASCII value of the character "4." On the other hand, `val("4")` is 4. Confused? You almost certainly want to use `val(` and str(unless you are sure you know what you are doing.

Invisible Characters

The ASCII system contains a number of characters that are normally invisible. In fact, every ASCII character with a value of 32 or lower is invisible. Normally you will not be concerned with invisible characters. However, there are three special invisible characters that do get a lot of use: **space**, **carriage return**, and **tab**.

The space character (ASCII value 32) is not quite invisible, because it does take up space. You can easily enter this value by pressing the **Space Bar**. In a formula you can enter a space directly [`" "`] or using the `chr(` function [`chr(32)`].

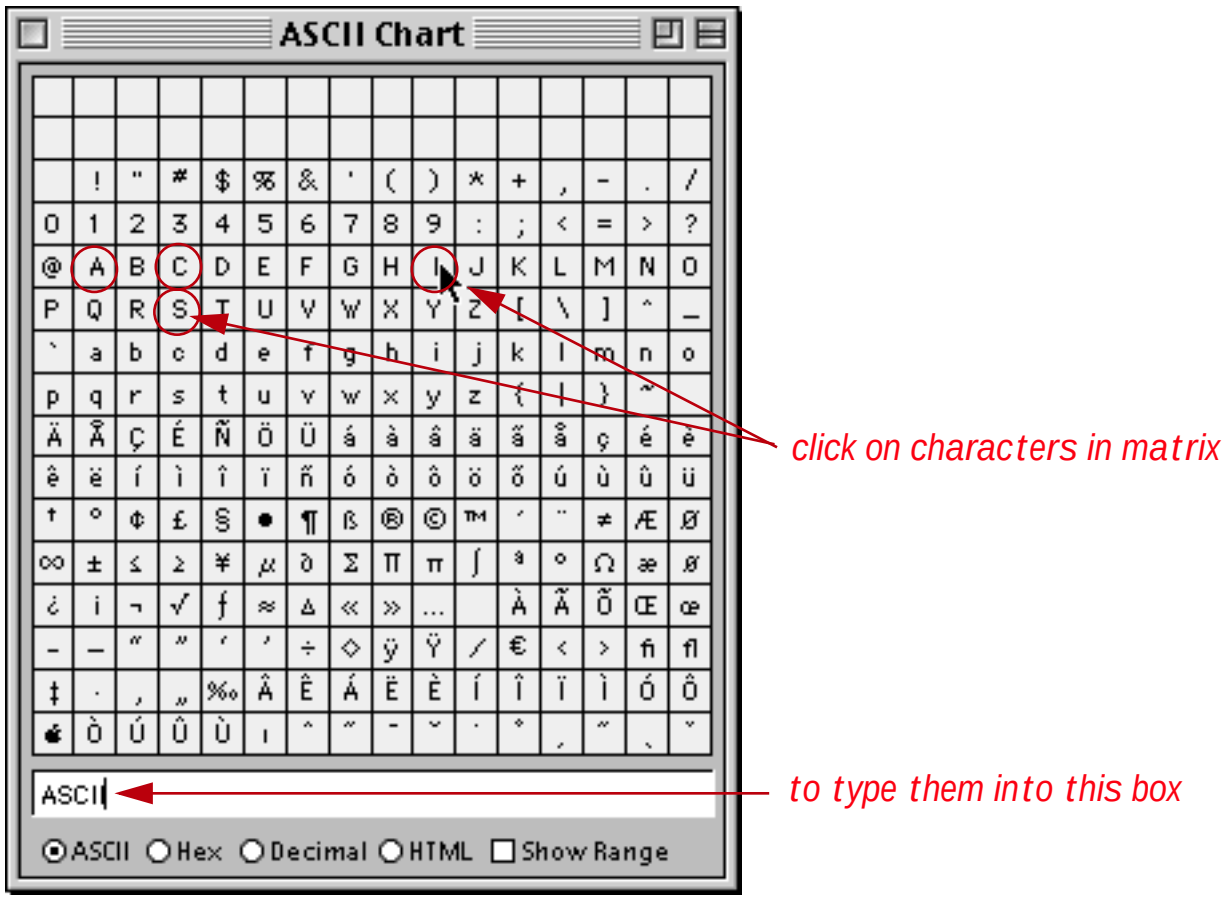
The carriage return character is used to start a new line of text. This character has an ASCII value of 13. You can enter this value into a formula using the *ƒ* symbol (see "Special Characters" on page 1202) or as `chr(13)`.

(Trivia question: why is this character called carriage return? In a few years probably no one will remember. In case you are already too young to remember, typewriters (and teletypes) used to place the paper on a carriage that moved back and forth as you typed. When you pressed the **Return** key the carriage would "return" back to the beginning of the line and also advance down to the next line, hence carriage return. In fact, on old manual typewriters this was accomplished with a lever, not a key.)

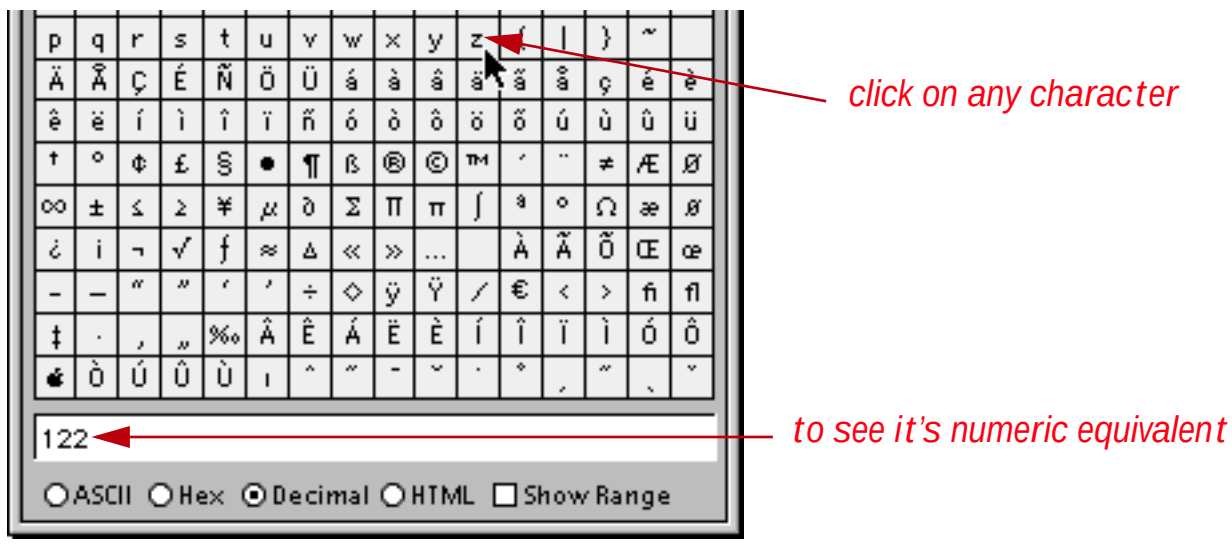
The **tab** character is usually not found inside data, but is often found in text files created by editors or word processors (including the Panorama word processor). The tab character has an ASCII value of 9. You can enter this value into a formula using the ´ symbol (see "Special Characters" on page 1202) or as `chr(9)`.

The ASCII Chart Wizard

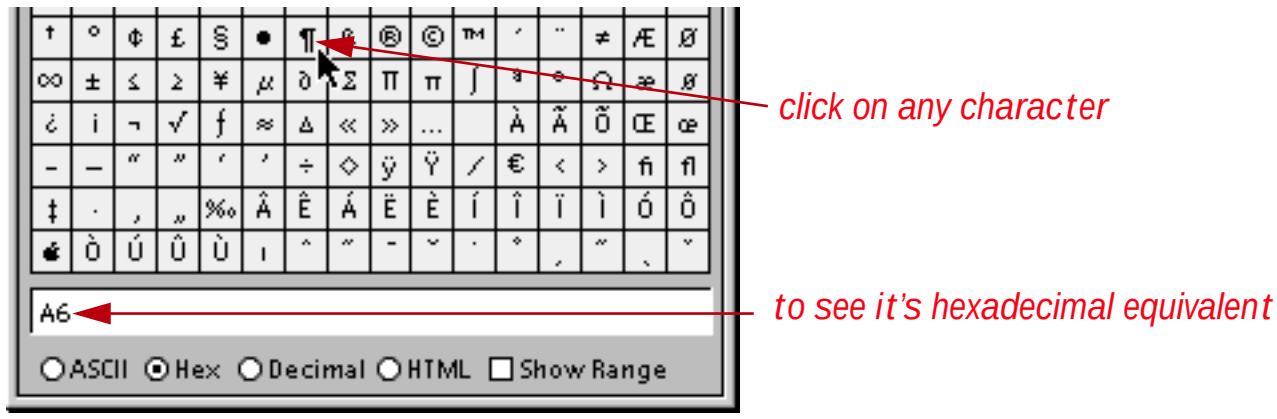
The **ASCII Chart** wizard allows you to displays a matrix showing all 256 ASCII characters. When you click on a character it types that character into the box at the bottom.



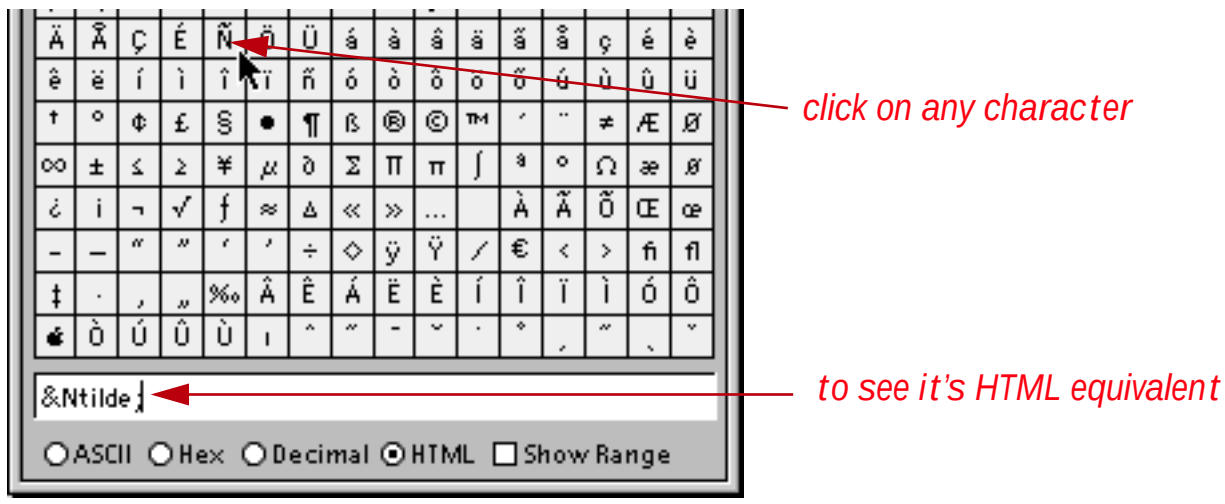
If you select the Decimal option then clicking on a character enters the corresponding numeric value of the character into the box.



Use the Hex option to see the numerical value of the character in hexadecimal (see "Raw Binary Data" on page 1296).

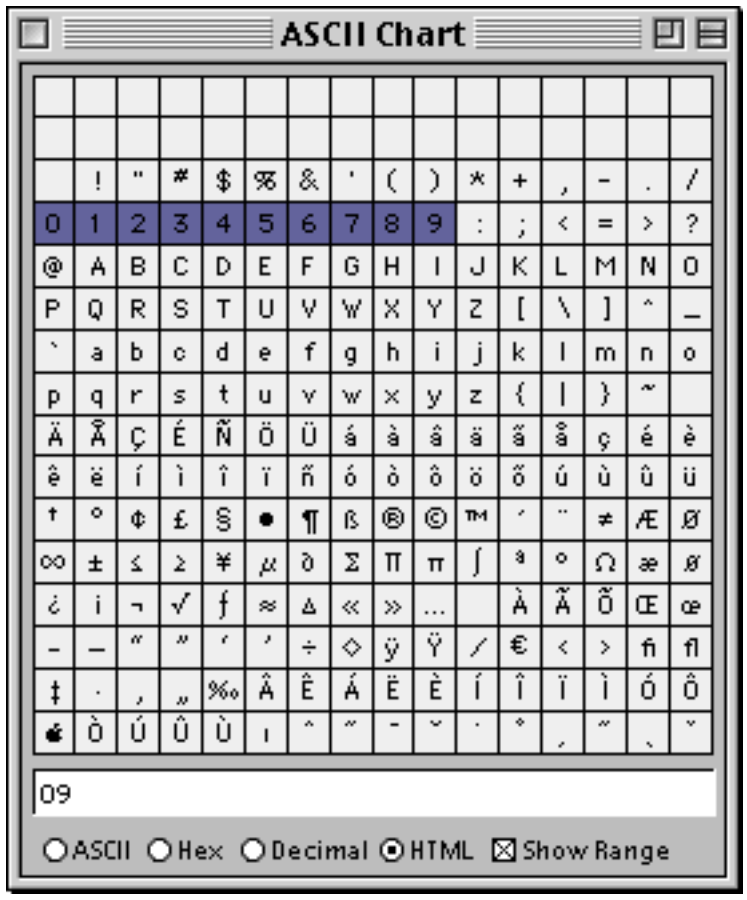


HTML has special codes for many characters. Use the HTML option to see the equivalent code (if any) for a special character.

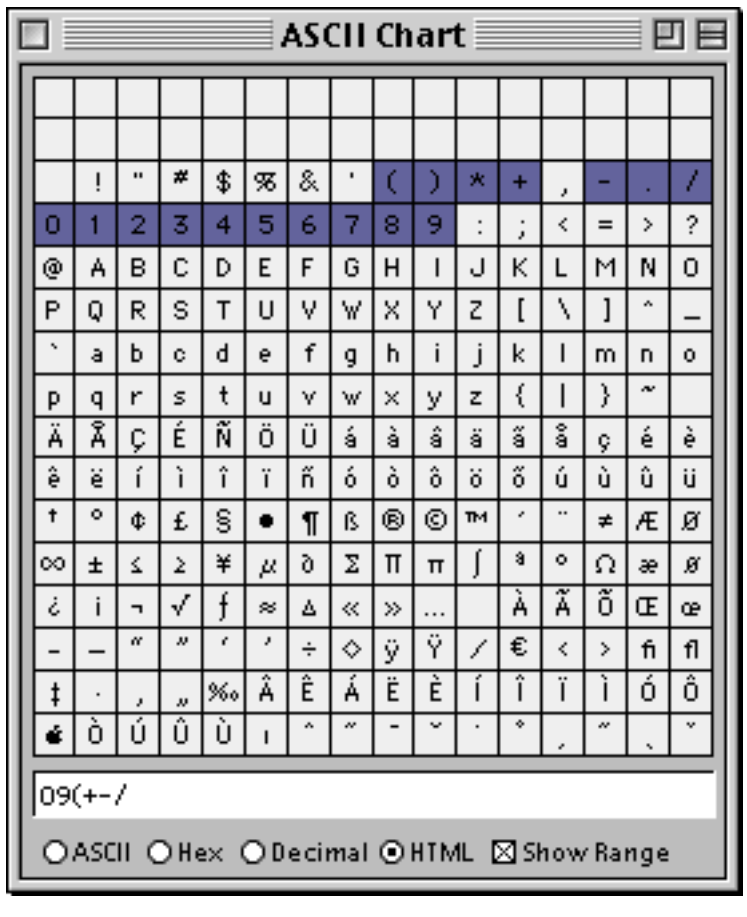


Showing Character Ranges with the ASCII Wizard

Several Panorama features use character ranges, including field properties (see "Restricting Character Types" on page 330), text funnels (see "Taking Strings Apart (Text Funnels)" on page 1214) and the `stripchar(` function (see "String Modification Functions" on page 1224). The **ASCII Chart** wizard allows you to preview a character range by selecting the Show Range option and typing the range into the box. All of the characters in the range will be highlighted in blue. For example, the illustration below shows the range 09, which includes all numeric characters.



Here is a more complex range that includes all the characters used in basic mathematical formulas.



You can use the **ASCII Chart** wizard to try out your ranges before you actually use them in a database.

ASCII Character Constant Functions

These functions return common ASCII characters.

Function	Description
info("lineseparator")	This function returns the line separator character on the current platform. On Macintosh systems this is a carriage return. On Windows PC systems this is a carriage return followed by a linefeed (CR-LF).
cr()	This function generates a carriage return. This is equivalent to chr(13) and is also the same as ¶.
crlf()	This function generates a carriage return line feed. This is equivalent to chr(13)+chr(10).
lf()	This function generates a line feed. This is equivalent to chr(10).
tab()	This function generates a tab character. This is equivalent to chr(9) and is also the same as ¬.
vtab()	This function generates a vertical tab character. This is equivalent to chr(11).

Text Arrays

An array is a numbered collection of data items. Panorama includes a number of functions and statements that treat a single text data item as if it were a numbered collection of smaller items. The smaller text data items must be separated from each other by a delimiter, for instance a comma or carriage return.

Consider the text data item shown below. Panorama would normally treat this as a single item with a length of 40 characters. The functions described in this section, however, can treat this text as a collection of 7 elements separated by semicolons.

```
white;red;orange;yellow;green;blue;black
```

In this example, the ; is the separator character. You can use any character you want for a separator character, in fact, you can use different separator characters at different times. You could even build a multi-level array by using two different separator characters.

Using the array functions and statements provided by Panorama you can extract elements from an array, change array elements, even sort an array. Since arrays are really text, they can be stored in any variable or any text field, and they can be edited with the data sheet, a data cell, or a Text Editor SuperObject.

There are many statements and user interface elements that work with text arrays, including lists and pop-up menus. There are also a number of functions that generate text arrays, including functions for building lists of files, windows, fields, choices, and data. Most of these statements, user interface elements, and functions require that carriage returns be used as separators, so that each array element is on a separate line.

It is up to you to keep track of the fact that you are using an array and what the separator character is. Panorama won't stop you from trying to access the array of colors above as if it were delimited with commas instead of semicolons, but you probably won't get the results you wanted unless you use the correct separator character.

(If you are familiar with the arrays in C or Pascal, Panorama text arrays are quite a bit different, although both are a numbered collection of items. As with anything unfamiliar, Panorama text arrays probably won't look as good as the ones you are used to at first. Panorama arrays do have some significant advantages though: they don't have to be declared in advance, each array element can be of unlimited length without wasting space, and Panorama arrays can be directly edited. It's also very easy to "pre-fill" a Panorama array with a list of values.)

Picking a Separator Character

Any ASCII character can be used as a separator character, so you have 256 possible choices. Common separators include comas, semicolons, slashes, carriage returns, spaces and tabs.

It's important to pick a separator character that will not occur in the data elements of your array. If your data may include commas, don't use the comma as a separator character. If the data might include carriage returns, don't use a carriage return. If you want to be extra sure to avoid conflicts, pick a non-printing character. You can use the `chr(` function (reference page 5099) to generate non-printing characters, for example `chr(1)`, `chr(2)`, `chr(3)`. Most `chr(` values below 32 are non-printing except for `chr(9)` and `chr(13)`, which correspond to tab and carriage return.

Some Panorama user interface elements and functions use text arrays as parameters or to hold a list of values. For these applications the separator character is usually required to be a carriage return. For example, the Pop-Up Menu SuperObject uses a carriage return delimited array to define the list of pop-up menu choices (see "The Pop-Up Menu Formula" on page 853). The `lookupall(` function (reference page 5502) extracts information from another database and places it into an array with whatever separator you specify. Consult the documentation for each individual statement, function or SuperObject to see the exact specifications for any arrays they may use.

Working With Arrays

Panorama has about a dozen functions and procedure statements for working with arrays. These functions are described in this table.

Function	Reference Page	Description

array(text,item,sep)	Page 5036	This function extracts a single data item from a text array. Text is the item of text that contains the data you want to extract. Item is the number of the data item you want to extract. The first item is item 1, the second is item 2, the third item is 3, etc. Separator is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character.
		
		The array(function returns a single item of text from the array. Only the item itself is returned, the separator characters on each end are not included. If the item does not exist (for example if you ask for item 12 from a 7 item array) the function will return empty text ("").
		
		There are 7 VHF television stations in Los Angeles. The example procedure below will convert channel numbers into the names of the stations. For example, the procedure converts Channel 7 into KABC.
		
		`Stations=",KCBS,,KNBC,KTLA,,KABC,,KCAL,,KTTV,,KCOP"`
		`«Channel Name»=array(Stations,7,",")`
		
		The example uses an array called Stations. This array uses commas as a separator character.

arraychange(text,value,item,sep)	Page 5040	This function changes a single value inside a text array. Only the one item is changed, all the other items in the array remain the same. Text is the text array that contains the data you want to change. Value is the new value of the data item. Item is the number of the data item you want to change. Items are numbered starting from 1 (1,2, 3,…). This item must already exist in the array. The arraychange(function will not add the item if it does not exist. Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character. This function returns a copy of the text array, with the data item changed. If you want to change the original array you should use an assignment statement (see below). The example procedure below will change the 5th item of the array to Navajo White. `Colors=arraychange(Colors,"Navajo White",5,";")` This example assumes that a field or variable named Colors already exists.
arraycontains(text,item,sep)		This procedure checks to see if any element of an array matches the specified text. For the result to be true, the array element must match the specified text exactly, including upper and lower case. Otherwise the function will return false. So checking for "Green" will only match that exact array element, not "green" or "Olive Green". (Note that this is quite different from the contains operator, which ignores upper and lower case and allows a submatch.)

Function	Reference Page	Description
arraydelete(text,item,count,sep)	Page 5043	This function deletes one or more elements from the middle of a text array. Text is the text array that you want to insert elements into. Item is the spot where you want the elements to be deleted. Count is the number of elements you want to delete from the array. Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character. This function returns a copy of the original text array, with the specified elements deleted from the middle. The example procedure below will delete the 3rd item from the SpeedDial array: `SpeedDial=arraydelete(SpeedDial),3,1,¶)`
arraydeletevalue(text,value,sep)		This function deletes any array elements that match the value parameter. This mush be an exact match, including upper and lower case. If the value occurs multiple times in the array, every occurrence of the value will be removed, with one exception: If the value occurs in two consecutive array elements, only the first occurrence will be deleted.

arrayelement(text,position,sep)	Page 5044	This function converts between character positions and array element numbers in a text array. Given a character position within the overall text, the arrayelement(function tells what array element the character is in. For example, in the array red;blue;green the 7th character (u) is in the 2nd array element.
		
		This function has three parameters: text, position and sep. Text is the text array that you are working with. Position is the position of the character within the overall text (starting with 1 for the first character). Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character.
		
		This function returns a number. This is the number of data element in the array corresponding to the character position parameter. If the position corresponds to a separator character, the function will return the element number of the data element to the right of the separator.
		
		The example procedure below adds a new color to the RecentColors array. It then arbitrarily cuts off the array so that it is less than 200 characters long. The arrayelement(function makes it possible to write this procedure so that the array can be cut off without cutting an array element in the middle.
		
		`local lastElement`
		`RecentColors= parameter(1)+`
		`sandwich( ¶,RecentColors,"")`
		`lastElement=arrayelement(RecentColors,200,¶)`
		`RecentColors=`
		`arrayrange(RecentColors,1,lastElement,¶)`
		
		This procedure could be useful for maintaining a pop-up menu of recently used colors. The procedure automatically keeps the menu to a reasonable size by lopping off old colors from the bottom if the array gets over 200 characters long.

arrayfirst(text,sep)		This function extracts the first element of an array.

Function	Reference Page	Description
arrayinsert(text,item,count,sep)	Page 5047	This function inserts one or more elements into the middle of a text array. Text is the text array that you want to insert elements into. Item is the spot where you want the new elements to be inserted. Count is the number of blank elements you want to insert into the array. Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character. This function returns a copy of the original text array, with the new blank array elements inserted into the middle. The example procedure below will add 5 new array items to the SpeedDial array between the 2nd and 3rd array items: `SpeedDial=arrayinsert(SpeedDial),¶,3,5)` The new array items created by arrayinsert(are blank (empty). You can fill them in with the arraychange(function.
arraylast(text,sep)		This function extracts the last element of an array.
arraylefttrim(text,count,sep)		Removes the first elements of an array. For example arraylefttrim(text,",",2) removes the first two elements from a comma separated array.
arraylookup(text,key,mainsep, subsep,default)		This function looks up a value in a double column table, similar to a lookup. The first column in the table is the key value, the second column is the data value. Each line in the table is separated by the mainsep character, while the two columns in each line are separated by the subsep character. If no match is found the default value is returned. For example this function: arraylookup("AL.ALABAMA;AK.ALASKA; ... WY.WYOMING",State,":",".","") can be used to look up a long state name given the two letter abbreviation.

arrayrange(text,start,end,sep)	Page 5050	This function extracts a series of data item from a text array. Text is the item of text that contains the data you want to extract. Start is the number of the first data item you want to extract. Items are numbered starting from 1 (1, 2, 3,…). End is the number of the last data item you want to extract. Items are numbered starting from 1 (1, 2, 3,…). Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character.
		
		This function returns a series of items from the array. It returns the first item, the last item, and everything in between (including any separators that are in between). If the last item does not exist (for example if you ask for item 12 from a 7 item array) the function will return up to the actual last item in the array. If both requested items do not exist, the function will return empty text ("").
		
		This example procedure will fill the variable WeekDays with the text Mon,Tue,Wed,Thu,Fri.
		
		`Days="Sun,Mon,Tue,Wed,Thu,Fri,Sat"`
		`WeekDays=arrayrange(Days,2,6,",")`

Function	Reference Page	Description

arraymerge(array1,array2, separator,joiner)		This function merges two text arrays together. This function has four parameters: array1, array2, separator and joiner . Array1 is the first text array you want to merge, array2 is the second array. Separator is the separator character for both arrays (in other words, both arrays must use the same separator). This should be a single character. For carriage return delimited arrays, use the ¶ character (option-7). For tab delimited arrays use the ¬ character (option-L). Joiner is from 1 to 10 characters of text that will be used to join the individual elements of the two arrays.
		
		The result is a new array with the elements of the original arrays joined together. This means that the first element of the first array will be joined with the first element of the second array, then the second element of the second array will be joined with the second element of the second array, and so on until both arrays are completely merged.
		
		The example could be used to display names and phone numbers from a contact database in a Text Display SuperObject.
		
		`arraymerge(`
		`lookupall( "Contacts","Company","Name",¶),`
		`lookupall( "Contacts","Company","Phone",¶),`
		`¶,", ")`
		
		The display will look something like the text shown below.
		
		John Smith , (510) 323-4905
		Susan Wilson , (510) 590-1341
		Bill Franklin , (510) 323-6781

arrayreplacevalue(text, oldvalue,newvalue,sep)		This function replaces any array elements that match the value parameter. This must be an exact match, including upper and lower case. If the value occurs multiple times in the array, every occurrence of the value will be replaced, with one exception. If the value occurs in two consecutive array elements, only the first occurrence will be replaced.
arrayreverse(text,sep)	Page 5051	This function reverses the order of the elements in a text array. In other words, the first element becomes the last element, the second element becomes the second to last, etc. Text is the text array that you want to modify. Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see “Special Characters” on page 1202). For tab delimited arrays use the ¬ character. The arrayreverse(function reverses the order of the elements of an array. For example, the formula: `arrayreverse("1;2;3;4",";")` will produce the array 4;3;2;1.
arrayreverselookup(text,key, mainsep,subsep,default)		This function looks up a value in a double column table, similar to a lookup. The first column in the table is the key value, the second column is the data value. However, this function reverses the function of these two columns. The key parameter is looked up in the second column, then the associated value is returned from the first column. This is the reverse of the arraylookup(function. Each line in the table is separated by the mainsep character, while the two columns in each line are separated by the subsep character. If no match is found the default value is returned. For example this function: arrayreverselookup("AL.ALABAMA;AK.ALASKA; ... WY.WYOMING",State,":",".","") can be used to look up a two letter state abbreviation given the full name of the state.

Function	Reference Page	Description
arrayscan(field,sep)	Page 5052	This function allows the individual elements of a text array in a database field to be exported on separate lines. Field is the name of the field that contains the array you want to export. (You can also use a variable, but this usually doesn't make sense). Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character. This function returns one element from the array. However, unlike the array(function, the arrayscan(modifies the way the `export` and `arraybuild` statements work. These statements will repeat the formula containing arrayscan(over and over again for each record. Each time, the function will return the next element in the array, until there are no more items.
arraysearch(array,text,start,sep)	Page 5054	This function searches a text array to see if it contains a specific value. Array is the text array that you want to search. Text is the text that you want to search for. This parameter may contain the wildcard characters ? and * . For example, to search for array items that start with John use John* . To search for any array item containing Pacific use *Pacific*. The array item must match the text exactly, including upper/lower case. For more information on wildcard characters, see "A match B" on page 1267. Start is the spot in the array where you want the search to begin from. If you want to search the entire array, this parameter should be one. Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character. If the arraysearch(function finds an array element that matches what you are searching for it returns the number of that array element (1, 2, 3, etc.). If there is no matching element, the function returns 0.

arraysize(text,sep)	Page 5057	This function counts the number of items in a text array. Text is the text array that you want to count. Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character.
		
		This function returns a number. This is the number of elements in the array. If there is no text in the array, the function will return one. If you need a function that returns zero if there is no text you can use the extract function with the last parameter set to -1 (see "String Modification Functions" on page 1224).
		
		This example uses the arraysize(function to display the number of forms in the current database. (The dbinfo("forms,"") function creates an array listing all the forms in the current database, separated by carriage returns.)
		
		`message "This database contains "+`
		`  str(arraysize(dbinfo("forms",""),¶))+" forms"`

arraystrip(text,sep)	Page 5059	This function removes any blank elements from a text array. Text is the text array that you want to strip the blank elements from. Sep is the separator character for this array. This should be a single character. For carriage return delimited arrays, use the ¶ character (see "Special Characters" on page 1202). For tab delimited arrays use the ¬ character. This function returns a copy of the original text array, with any blank array elements removed from the array.

Function	Reference Page	Description
arraytrim(text,count,sep)		This function removes the last elements of an array. For example arraytrim(text,",",2) removes the last two elements from a comma separated array.
lineitemarray(field,separator)	Page 5468	This function converts the data in a set of line item fields into a text array (see "Text Arrays" on page 1236). Field is the line item field that contains the data. You should put the line item field name in quotes, and it should end with the Ω symbol (see "Special Characters" on page 1202). Separator is the separator character for this array. This should be a single character. This function returns a copy of the line item data packed into an array. If the line items contain numbers or dates they are converted to text before being added to the array.

HTML Tag and Tag Parsing Functions

Panorama has several functions for working with text that contains data delimited by tags. These functions are not actually specific to HTML, and you may find other uses for them.

These functions treat a tag as three components: header, body and trailer. In this example the tag header is <, the tag trailer is >, and the tag body is IMG SRC="happy.gif".

```
<IMG SRC="happy.gif">
```

The tag header and trailer may be more than one character long. Here is the same tag but with only the picture name as the body. In this example the tag header is <IMG SRC=", the tag trailer is ">, and the tag body is happy.gif.

```
<IMG SRC="happy.gif">
```

The tag functions don't care about upper or lower case, so this tag will work fine if it is <img src="happy.gif">.

Here are descriptions of all of the tag parsing functions.

Function	Reference Page	Description

tagarray(text,header,trailer,sep)	Page 5830	This function builds an array containing the body of all the tags in the text. The header is the character or sequence of characters that appears at the start of each tag. To extract all HTML tags the header would be <. To extract all image tags the header would be <img or <IMG (either upper or lower case will work). The trailer is the character or sequence of characters that appears at the end of each tag. To extract all HTML tags the trailer would be >. Each element in the array is separated from the next with the sep character. This character is often a carriage return (¶) or comma, see "Text Arrays" on page 1236.
		
		This example displays a list of all HTML tags in the variable Page. The list will be separated by commas, for example H1,/H1,B,/B,IMG SRC="my picture.jpeg".
		
		`message tagarray(Page,"<",">",",")`
		
		This example lists all of the pictures in the variable Page.
		
		`message tagarray(Page,"<IMG ",">",¶)`
		
		This will list each image on a separate line, like this:
		
		`src="happy.gif"`
		`src="rocket.jpeg"`
		`SRC="My Picture.jpeg"`
		`src="logo.gif"`

tagcount(text,header,trailer)	Page 5832	This function returns the number of tags in the text. The header is the character or sequence of characters that appears at the start of each tag. To count all HTML tags the header would be <. To count all image tags the header would be <img or <IMG (either upper or lower case will work). The trailer is the character or sequence of characters that appears at the end of each tag. To count all HTML tags the trailer would be >. Here is an example that uses this function to count the number of links in an HTML document.
		
		`message "This page contains "+`
		`  str(tagcount(Page,"<A HREF",">"))+" links."`

Function	Reference Page	Description
tagdata(text,header,trailer,number)	Page 5833	This function returns the body of the specified tag. The header is the character or sequence of characters that appears at the start of each tag. To extract an HTML tag the header would be <. To extract an image tags the header would be <img or <IMG (either upper or lower case will work). The trailer is the character or sequence of characters that appears at the end of each tag. To extract an HTML tag the trailer would be >. The number parameter specifies which tag you want: 1, 2, 3, etc. for the first, second, third tag etc.
tagstart(text,header,trailer,number)	Page 5840	These functions return the starting and ending position of the body of a tag within a text. The header is the character or sequence of characters that appears at the start of each tag. The trailer is the character or sequence of characters that appears at the end of each tag. The number parameter specifies which tag you want: 1, 2, 3, etc. for the first, second third tag etc. This example displays 20 characters or so around the first image tag. `message Page[tagstart(Page,"<IMG",">",1)-20, tagend(Page,"<IMG",">",1)+20]` If the requested tag does not exist, the tagstart(and tagend(function will return 0.
tagend(text,header,trailer,number)	Page 5834	
tagnumber(text,header,trailer,pos)	Page 5835	This function checks to see if position is in a tag within text, and if so, returns the number of that tag within the document (1, 2, 3). If the position is not inside of a tag the function will return zero. The header is the character or sequence of characters that appears at the start of each tag. The trailer is the character or sequence of characters that appears at the end of each tag.
striphtmltags(text)		This function removes all HTML tags from the text.

Tag Parameter Functions

Many HTML tags contain parameters. For example, this tag has three parameters, src, align and border.

```
<IMG SRC="mylogo.gif" align=left border=0>
```

Panorama has built in functions that can help you extract a series of parameters like this. Although these functions were designed with parsing HTML tags in mind you may find other uses for them as well.

Function	Reference Page	Description
tagparameter(text,name,num)	Page 5837	This function returns the value of a specified parameter in the tag. Text is the list of parameters. If you are parsing HTML this should be the body of the tag. Name is the name of the parameter you want to extract, including any trailing punctuation (= for HTML tags). Either upper or lower case is ok. For example to extract the name of the image itself from an image tag (see example above) the name would be src= or SRC=. To extract the alignment the name would be align= or ALIGN=. The num is in case there is more than one parameter with the specified name, it tells Panorama which one to extract (1, 2, 3, etc.) If the parameter value is quoted (for example src="my logo.jpg" Panorama will remove the quotes as it extracts the value. If the parameter value is not quoted it will extract up to the first non-alphanumeric value. For example, this formula will return the image file name of the first IMG tag in a field named HTML. `tagparameter(tag(HTML,"<img",">",1),"src=",1)` If the first image tag in this text is <IMG SRC="mylogo.gif" align=left border=0> this function will return mylogo.gif.

tagparameterarray(text,name,sep)	Page 5838	This function returns an array of the specified parameters in a tag. This is useful if the same parameter may occur multiple times within the tag. Text is the list of parameters. If you are parsing HTML this should be the body of the tag. Name is the name of the parameter you want to extract, including any trailing punctuation (= for HTML tags). Either upper or lower case is ok. For example to extract the name of the image itself from an image tag (see example above) you the name would be src= or SRC=. To extract the alignment the name would be align= or ALIGN=. Sep is the separator character that will be placed in between each value in the output array (see "Text Arrays" on page 1236).
		To illustrate this function, suppose that you have a field named HTML that somewhere within it contains text that looks like this:
		`<MERGE field="Name" field="Address" field="City" field="State">`
		Using this formula we can extract an array of all the field names.
		`tagparameterarray(tag(HTML,"<merge",">","field=",";")`
		With the sample data listed above this formula will return the value Name;Address;City;State.

HTML Table Parsing Functions

These functions are specifically designed for extracting data from an HTML table.

Function	Reference Page	Description
htmltablecell(table,row,cell)		This function extracts the data from a cell in an HTML table. Any HTML tags in the cell are removed, leaving only the actual text. The thetable parameter must contain the body of an html table with <tr> and <td> tags (however, the actual <table> and </table> tags themselves are not required.)
htmltablecellexists(table,row,cell)		This function checks to see whether a cell in an HTML table exists or not. The result will be true if the cell exists, or false if it doesn't. The thetable parameter must contain the body of an html table with <tr> and <td> tags (however, the actual <table> and </table> tags themselves are not required.)
htmltablecellraw(table,row,cell)		This function extracts the data from a cell in an HTML table. Unlike the htmltablecell(function, any HTML tags in the cell are retained. The thetable parameter must contain the body of an html table with <tr> and <td> tags (however, the actual <table> and </table> tags themselves are not required.)
htmltableheight(table)		This function calculates the height (number of rows) in an HTML table. It assumes that the table is a regular matrix (no rowspan tags).
htmltablerowraw(table,row,cell)		This function extracts the data from a row in an HTML table. Any HTML tags in the row are retained. The the table parameter must contain the body of an html table with <tr> and <td> tags (however, the actual <table> and </table> tags themselves are not required.)
htmltablewidth(table)		This function calculates the height (number of rows) in an HTML table. It assumes that the table is a regular matrix (no colspan tags) and that all of the rows have the same number of columns as the top row in the table.

HTML/URL Conversion Functions

The HTML and URL standards used on the Internet do not use standard ASCII text. Panorama includes conversion functions for converting between standard ASCII and HTML and URL's. These functions are very convenient for generating HTML from a database, for example in CGI code for a web server.

Function	Reference Page	Description
htmlencode(text)	Page 5346	This function converts from standard ASCII to HTML. Wherever possible, special characters are converted to their HTML equivalents (for example © is converted to ©, & is converted to &. Special characters that do not have HTML equivalents are removed. (However, the smart quote characters, “,”, ‘ and ’ are converted to regular quote characters " and '.) To allow you to convert HTML text that contains tags, the htmlencode(function does not convert the < and > characters. If you want to convert these characters into their HTML equivalents use this formula: `replacemultiple( htmlencode(text), "<.>", "<.&gt", ".")`
htmldecode(text)	Page 5343	This function converts from HTML to standard ASCII, the exact opposite of the htmlencode(function. HTML characters like © and & are converted into the normal ASCII characters © and &.

Function	Reference Page	Description
urlencode(text)	Page 5884	The urlencode(function converts standard ASCII to URL format. For example, the text My URL would be converted to My%20URL.
urldecode(text)	Page 5883	The urldecode(function converts a URL to standard ASCII format. For example the text My%20URL would be converted to My URL.
urlfilename(url)		This function extracts the filename from a complete url.
urlpath(url)		This function extracts the path from a url.

HTML Generating Functions

These functions help with generating HTML.

Function	Reference Page	Description
htmlbold(string)		This function takes the text and adds and tags to it.
htmlitalic(string)		This function takes the text and adds and tags to it.

Date Arithmetic

Formulas can perform several useful calculations on dates. For example, you can calculate the number of days between two dates, or you can add or subtract a certain number of days to a date. You can also convert a date to text using a wide variety of formats.

Usually we think of a date in terms of years, months, and days. Formulas, however, treat dates as a certain number of days—specifically, the number of days between that date and January 1, 4713 B.C., adjusted for the Gregorian calendar correction in October 1582. (The date 4713 B.C. is chosen for obscure astronomical reasons). For example, to a formula the date August 7, 1991 is day number 2,448,476.

Fortunately you should never have to worry about numbers like 2,448,476. The formula will automatically convert a date field into the number of days, perform the calculation, and then convert back into a regular date again.

Since formulas handle dates as numbers, you can use any numeric operator or function to manipulate dates. However it doesn't make much sense to take the square root of a date (although Panorama will let you). There are really only two numeric operations that make sense on dates—subtracting two dates to find the number of days in between and adding or subtracting a number of days to a date.

To calculate the number of days between two dates, just subtract one from the other. For example, the formula

```
«Ship Date»-«Order Date»
```

will calculate the number of days required to process an order.

To calculate an offset from a given date, just add the number to the date. For example the formula

```
«Ship Date»+30
```

calculates the normal due date 30 days after the ship date.

Today's Date

The `today()` function (reference page 5862) returns the number corresponding to today's date, allowing you to use today's date in a formula. For example, to calculate the age of an invoice use a formula like this.

```
today()-«Ship Date»
```

To calculate the due date for a library book, use the formula like this.

```
today()+14
```

This formula assumes that books are checked out for two weeks.

Converting Between Dates and Text

These functions allow you to convert a date into text, or text into a date. You should only use these functions if you want to store the result of a date calculation in a text field instead of a date field, or if you want to access a date that has been stored as text.

Note: Remember, formulas handle dates as numbers, so these functions actually convert numbers into text and vice versa. It's up to you to make sure that these numbers actually represent the correct dates.

Function	Reference Page	Description
completedatestr(number)		Convert a date to text, including the day of the week (for example Sunday, April 20th, 2003).
date(text)	Page 5146	This function converts a text string in a date format into the number representing that date. Use this function to include a constant date in your formula, for example date("12/9/1979"). You should also use this function to access dates that have been stored in text fields (but why are you doing that in the first place?). Several formats are supported, including mm/dd/yy, mm/dd/yyyy, Month dd, yyyy, and Mon dd, yyyy. Dates in the current week can be represented by the name of the day, for instance Tuesday or Fri. Dates in the previous or upcoming week can be represented by adding the words last or next, for example last friday or next wed.
datepattern(number,pattern)	Page 5148	This function converts a number representing a date into a formatted text string. The pattern parameter is an output pattern telling the function how to format the date. For more information on date output patterns, see "Date Output Patterns" on page 295. Use the datepattern(function to store a date in a text field, or to display a formatted date in an auto-wrap text object or Text Display SuperObject. For example, the formula: `datepattern(«Ship Date»,"Month ddnth, yyyy")` can be used to display the date an order was shipped in the format May 12th, 2003.
datestr(number)		Convert a date to text using format mm/dd/yy (for example 4/20/03).
daystr(number)		Convert a date to the day of the week (for example Sunday).
eurodatestr(number)		Convert a date to text in European format (for example 20-APR-2003).
exportcell(field)	Page 5209	This function takes any database field and converts it to text, using the appropriate pattern if one has been defined in the design sheet. Field is the name of the field to be converted to text. The function always returns a text type data item. The power of the exportcell(function is that it does not require you to know what type of data you are exporting. It simply takes whatever kind of data is in the field (text, number, date, whatever) and converts it into text.
longdatestr(number)		Convert a date to text with format Month ddnth, yyyy (for example April 20th, 2003).

Date Functions

These functions perform various calculations and conversions on date values. Unless specified otherwise the date is always processed as a numeric value (see “HTML Generating Functions” on page 1247).

Function	Reference Page	Description

dayofweek(date)	Page 5149	This function computes the day of the week (0-6) of a date, with Sunday being 0, Monday 1, etc. The function returns a number from 0 to 6. The days of the week are:
		
		`0 Sunday`
		`1 Monday`
		`2 Tuesday`
		`3 Wednesday`
		`4 Thursday`
		`5 Friday`
		`6 Saturday`
		
		The procedure below uses the dayofweek(function to select all weekday records (monday through friday).
		
		`select dayofweek(Date)≥1 and dayofweek(Date)≤5`

month1st(date)	Page 5532	This function computes the first day of a month. For example, if the date passed to this function is October 18, 1997, this function will return the date October 1, 1997. The date is returned as a number.
		
		The example procedure below uses this function to select the orders placed this month, then displays the count.
		
		`select OrderDate≥month1st( today()) and`
		`  OrderDate<month1st(today())+monthlength(today())`
		`message str(info("records"))+" orders this month"`

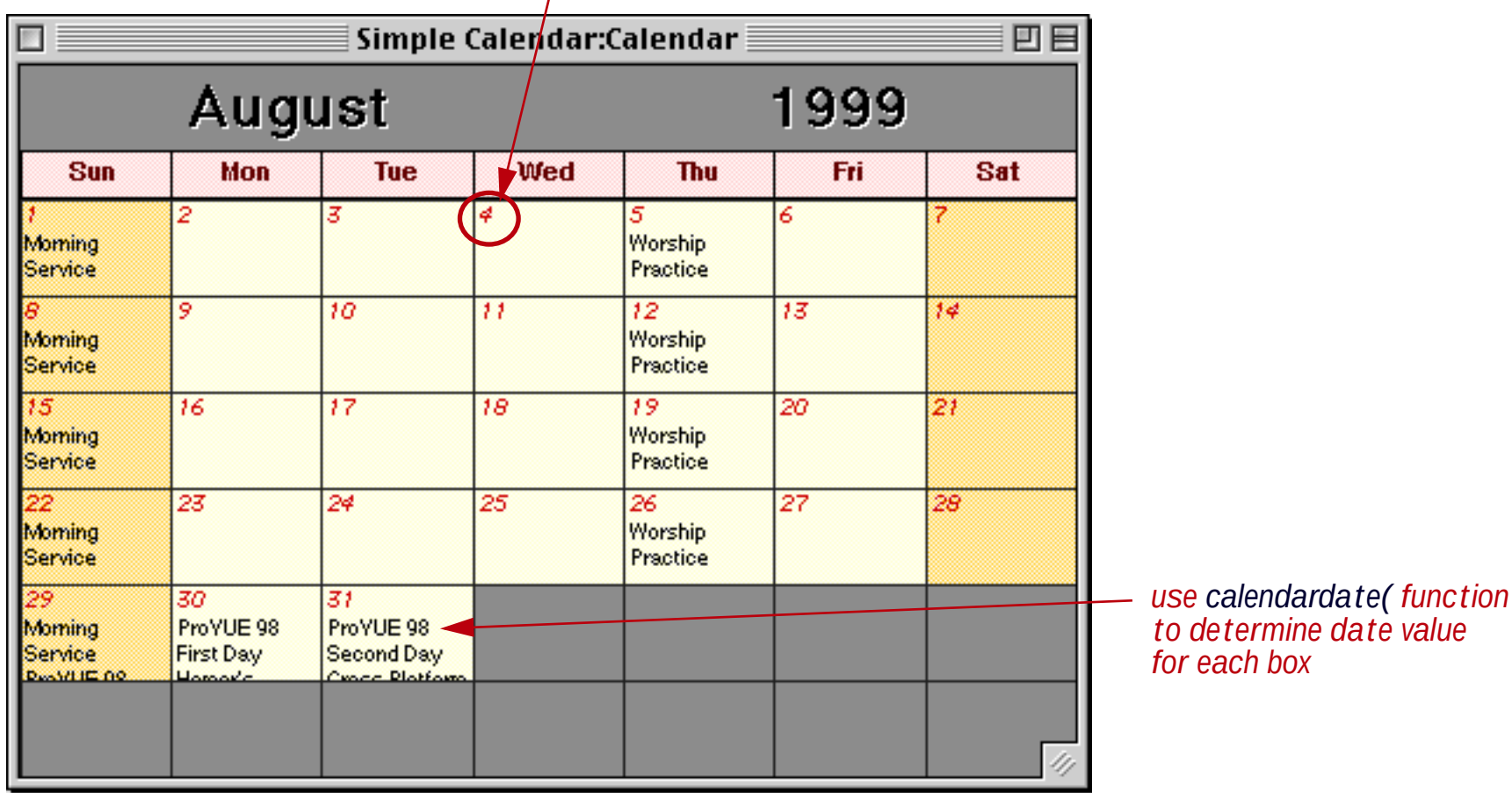
monthlength(date)	Page 5533	This function computes the length (number of days) of a month. For example, if the date passed to this function is October 18, 1997, this function will return 31, the number of days in October. This function knows about leap years and adjusts the length of February accordingly.
		
		The example procedure below uses this function to select the orders placed this month, then displays the count.
		
		`select OrderDate≥month1st( today()) and`
		`  OrderDate<month1st(today())+monthlength(today())`
		`message str(info("records"))+" orders this month"`

monthmath(date,offset)	Page 5534	This function takes a date and computes another date that is one or months before or after the original date. Date is a number representing the original date. Offset is the number of months that you want to add or subtract to the original date. Use a positive number to move forward in time, a negative number to go backwards. For example, if you offset the date May 12, 1997 by 2 (two months forward) the result is July 12, 1997. If you offset the same original date by -2 (two months backward) the result is March 12, 1997. If the new date does not exist because a month does not have enough days in it, the monthmath(function will pick the last day of the month. For example, if you offset March 31 by 1 month the result is April 30. If the new month lands in February the function knows about leap years and adjusts accordingly. This example calculates a renewal date exactly one year from today. `monthmath(today(),12)`

Function	Reference Page	Description
quarter1st(date)	Page 5623	This function computes the first day of a quarter. For example, if the date passed to this function is August 18, 1997, this function will return the date July 1, 1997. The date is returned as a number.
today()	Page 5862	This function returns today's date (assuming, of course, that your computer clock has been set correctly).
week1st(date)	Page 5890	This function computes the first day of a week (Sunday). For example, if the date passed to this function is July 12, 1995 (a Wednesday), this function will return the date July 9, 1997 (a Sunday). The date is returned as a number.
year1st(date)	Page 5912	This function computes the first day of a year. For example, if the date passed to this function is July 12, 1995, this function will return the date January 1, 1995. The date is returned as a number. The example below calculates the number of days remaining in the current year. `yearfirst(year1st( today( ))+366)-today()`

Calendar Functions

The functions described in this section facilitate the creation of monthly calendars like this.



See “Building a Calendar” on page 960 for step-by-step instructions on building a calendar like this.

Function	Reference Page	Description
calendarday(date,boxnumber)	Page 5084	This function is designed to help in creating monthly calendars. A standard monthly calendar has 6 rows and 7 columns (Sunday through Saturday) for a total of 42 boxes. For any given month from 28 to 31 of these boxes will be valid dates. The calendarday(function calculates what number from 1 to 31 (if any) should be displayed in one of these 42 boxes. This function has two parameters: date and boxnumber. Date is any date in the month being displayed. Boxnumber is the box within the monthly calendar being displayed. The boxes are numbered from 1 to 42, starting with the upper left hand corner. The table below shows the position of all 42 monthly calendar boxes. S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 This function returns a number from 1 to 31, or zero if the specified calendar box does not contain a day in this month.

Function	Reference Page	Description
calendardate(date,boxnumber)	Page 5082	This function is designed to help in creating monthly calendars. A standard monthly calendar has 6 rows and 7 columns (Sunday through Saturday) for a total of 42 boxes. For any given month from 28 to 31 of these boxes will be valid dates. The calendarday(function calculates what date corresponds to one of these 42 boxes. This function has two parameters: date and boxnumber. Date is any date in the month being displayed. Boxnumber is the box within the monthly calendar being displayed. The boxes are numbered from 1 to 42, starting with the upper left hand corner. The table below shows the position of all 42 monthly calendar boxes. S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 This function returns a date value (a number), or zero if the specified calendar box does not contain a day in this month. The output of the calendardate(function is usually fed into a lookupall(, lookupcalendar(, or lookuprtime(function. The last two functions can be used to lookup the events (appointments, to-do's, etc.) that occur on a particular day.
lookupcalendar(file, reminderField, date, dataField, separator)	Page 5504	This function builds a text array containing one item for every record in the target database where the date in the reminderField matches the date. Each item in the text array contains the value extracted from the dataField for that record. This function has five parameters: file, reminderField, date, dataField and separator. File is the name of the database that you want to search and grab data from. The database must be open. If you want to search and grab from the current database, use `info("databasename")`. ReminderField is the name of the field that you want to search in. This field must contain valid reminders (see "Reminders" on page 1260). The field must be in the database specified by the first parameter. Date is the actual date that you want to match. For example if you want to look up all appointments on july 23rd, this should be date("July 23"). This parameter is often a field in the current database. DataField is the name of the field that you want to retrieve data from. For example if you want to retrieve appointment information, this should be the name of the field that contains that information. This must be a field in the database specified by the first parameter. Separator is the separator character for the text array you are building (see "Text Arrays" on page 1236). The function returns a text array from all the records in the specified database where the reminderField and date match. This example returns a list of today's reminders. `lookupcalendar("Reminders",When,today(),Message,¶)`

Function	Reference Page	Description
lookuprtime(file, reminderField, date, pattern, separator)	Page 5509	This function builds a text array containing one item for every record in the target database where the date in the reminderField matches the date. Each item in the text array contains the time of the corresponding reminder. This function has five parameters: file, reminderField, date, pattern and separator. File is the name of the database that you want to search and grab data from. The database must be open. If you want to search and grab from the current database, use `info("databasename")`. ReminderField is the name of the field that you want to search in. This field must contain valid reminders (see “Reminders” on page 1260). The field must be in the database specified by the first parameter. Date is the actual date that you want to match. For example if you want to look up all appointments on july 23rd, this should be date("July 23"). This parameter is often a field in the current database. Pattern is the pattern you want to use to format the time. See the timepattern(function for details (reference page 5861). Separator is the separator character for the text array you are building (see “Text Arrays” on page 1236). The function returns a text array from all the records where the reminderField and date match. The text array contains the times for the reminders that match.
lookuprtypes(file, reminderField, date, pattern, separator)	Page 5511	The lookuprtypes(function builds a text array containing one item for every record in the target database where the date in the reminderField matches the date. Each item in the text array contains the type of the corresponding reminder, either "a" (appointment) or "t" (to-do). This function has five parameters: file, reminderField, date, pattern and separator. File is the name of the database that you want to search and grab data from. The database must be open. If you want to search and grab from the current database, use `info("databasename")`. ReminderField is the name of the field that you want to search in. This field must contain valid reminders (see “Reminders” on page 1260). The field must be in the database specified by the first parameter. Date is the actual date that you want to match. For example if you want to look up all appointments on july 23rd, this should be date("July 23"). This parameter is often a field in the current database. Pattern is not used, and should be "". Separator is the separator character for the text array you are building (see “Text Arrays” on page 1236). The function returns a text array from all the records where the reminderField and date match. Each element of the text array contains either "a" or "t" for the reminders that match.

Time Arithmetic

To Panorama, time is not hours, minutes, and seconds, but simply seconds. To be precise, a time is the number of seconds since midnight. For example, the time 4:32 AM is 16,320 seconds after midnight. As you can see, a Panorama time is really a number in disguise. Since times are numbers, it's easy to compare them, sort them, or find the difference between them (number of seconds).

Converting Between Times and Text

Unlike dates, Panorama does not automatically provide a time data type that automatically converts a date in text format into a number. You must use a function to convert time in text format into seconds before you can do math calculations with the time, and use another function to convert back.

Function	Reference Page	Description
now()	Page 5548	This function returns the current time (number of seconds since midnight). Of course the clock on your computer must be set correctly!

seconds(text)	Page 5709	This function converts text into a number representing a time. The function has one parameter — the text that you want to convert to a number representing a time. If the text includes and AM or PM suffix, the number of seconds is calculated from midnight (12 A.M.), otherwise it is calculated from 0:00:00 (elapsed time). The text must contain a valid time. Here are some examples of valid times:
		
		`4:13 PM`
		`11:00 AM`
		`2:30`
		`18:45`
		
		This function returns a number representing the time. The number is the number of seconds since midnight. For example, if the time is 10:23 AM this function will return the number 37,380.

timepattern(number,pattern)	Page 5861	This function converts a number representing a time into text. The function uses a pattern to control how the date is formatted.
		
		The function has two parameters: number and pattern. Number is the number that you want to convert to text. This number must be the number of seconds since midnight. Pattern is text that contains a pattern for formatting the date. The pattern is assembled from four components: hh (hours), mm (minutes) ss (seconds), and am/pm. Some of the more common time patterns are listed here:
		
		`Pattern` `Converted Text`
		`"hh:mm:ss am/pm"` `4:32:17 pm`
		`"hh:mm am/pm"` `4:32 pm`
		`"hh:mm:ss"` `16:32:17`
		
		If am/pm is left off the pattern the time will be formatted in 24 hour format, as shown on the last line of the table above. You should also leave off am/pm for converting elapsed times.

Function	Reference Page	Description

time(text)	Page 5855	This function converts text into a number representing a time. The function has one parameter — the text that you want to convert to a number representing a time. The time function allows you to leave out the colons in the time, and also allows you to leave off the am/pm. Here are some examples of valid times:
		
		`4:13 PM`
		`11:00 AM`
		`2:30`
		`18:45`
		`230`
		`4p`
		`midnight`
		`noon`
		`afternoon`
		`evening`
		`night`
		`nite`
		
		The time(function is very lenient about the format you use to enter the time. It will accept a time without colons, for example 425 pm instead of 4:25 pm. If there is no am or pm the time function will try to make an intelligent guess. For example, 230 is almost certainly 2:30 pm, not 2:30 am. By default, the time(function assumes that any time from 6:00 to 11:59 is AM, and any time from 12:00 to 5:59 is PM, but you can change these assumptions with the timedefaults statement (reference page 5858).
		
		The time(function will also convert “named” times: noon, midnight, morning, afternoon, evening, and night. This function assumes that morning is 9:00 am, afternoon is 1:00 pm, evening is 6:00 pm, and night is 10:00 pm. These assumptions can be changed with the timedefaults statement (reference page 5858).

timestr(number)		Convert a number to text in am/pm time format (for example 9:34 AM).

Time Calculations

Once time has been converted into seconds you can perform arithmetic on it. For example, to calculate the number of hours worked from a time card use a formula like this (this formula assumes that In and Out are text fields containing times).

```
(seconds(Out)-seconds(In))/3600
```

(The division by 3600 converts the result into hours.)

To find out when a task will be finished that takes 2 1⁄2 hours to complete, use the formula

```
seconds(«Start Time»)+seconds("2:30")
```

Simple addition and subtraction does not compensate for time wrapping around midnight. For example, if you want to calculate the length of a shift that begins at 11 P.M. and ends at 7 A.M., you must add 24 hours to 7AM before subtracting the times. To solve this problem you can use one of the functions described below, or you can use a SuperDate, which combines time and date into a single number (see "SuperDates (combined date and time)" on page 1259).

Function	Reference Page	Description

time24(time)	Page 5857	This function takes a time and makes sure it falls within a 24 hour period. If the time is less than 24 hours, it is unchanged. If the time is greater than 24 hours, it is converted to the equivalent time in a 24 hour period (for example 30:00:00 is converted to 6:00:00).
		The time24(function can help with calculations of an ending time from a start time and duration. The basic formula for such a calculation is shown here.
		`EndTime=StartTime+Duration`
		This formula works fine unless the interval extends over midnight. The time24(function adjusts the result to make sure it starts over at zero as it crosses midnight.
		`EndTime=time24(StartTime+Duration)`
		This formula will correctly calculate that 10:30 PM + 4 hours is 2:30 AM.

timedifference(start,end)	Page 5859	This function calculates the difference between two times. It works correctly even if the interval between the two times crosses over midnight. This function returns a time interval between -12 and +12 hours. See also the timeinterval(function, which returns a time interval between 0 and 24 hours. There are two parameters, start and end. Start is a number (number of seconds) representing the starting point of the time interval. End is a number (number of seconds) representing the ending point of the time interval. This function returns the number of seconds between the two times. For example, if the start time is 9:30 PM and the end time is 2:05 AM, the difference would be 4:35. But if the parameters are reversed and the start is 2:05 AM and the end is 9:30 PM, the difference is -4:35. If the result is positive, the end is after the start. But if the result is negative, the start is after the end.
timeinterval(start,end)	Page 5860	This function calculates the time interval between two times. It works correctly even if the interval between the two times crosses over midnight. This function returns a time interval between 0 and 24 hours. See also the timedifference(function, which returns a time interval between -12 and +12 hours. There are two parameters, start and end. Start is a number (number of seconds) representing the starting point of the time interval. End is a number (number of seconds) representing the ending point of the time interval. This function returns the number of seconds between the two times. For example, if the start tine is 9:30 PM and the end time is 2:05 AM, the interval would be 4:35. But if the parameters are reversed and the start time is 2:05 AM and the end time is 9:30 PM, the interval is 19:25.

Time Calculations with Text

Unlike the functions in the previous sections, these functions operate with time values in strings. There aren't as many functions available as for times expressed as numbers, but if your input and output values will be in strings using these function saves the intermediate conversion steps.

Function	Reference Page	Description
texttimedifference(start,end)		This function calculates the difference between two times. Instead of being expressed as numbers, the input output times are expressed as text (for example 12:45 pm). This function works correctly even if the interval between the two times crosses over midnight. This function returns a time interval between -12 and +12 hours. See also the timeinterval(function, which returns a time interval between 0 and 24 hours. There are two parameters, start and end. Start is a string representing the starting point of the time interval. End is a string representing the ending point of the time interval. This function returns the time difference between the start and end. For example, if the start time is 9:30 PM and the end time is 2:05 AM, the difference would be 4:35. But if the parameters are reversed and the start is 2:05 AM and the end is 9:30 PM, the difference is -4:35. If the result is positive, the end is after the start. But if the result is negative, the start is after the end.
texttimeinterval(start,end)		This function calculates the time interval between two times. It works correctly even if the interval between the two times crosses over midnight. This function returns a time interval between 0 and 24 hours. See also the timedifference(function, which returns a time interval between -12 and +12 hours. There are two parameters, start and end. Start is a string representing the starting point of the time interval. End is a string representing the ending point of the time interval. This function returns the time between the start and end. For example, if the start tine is 9:30 PM and the end time is 2:05 AM, the interval would be 4:35. But if the parameters are reversed and the start time is 2:05 AM and the end time is 9:30 PM, the interval is 19:25.

Calculating Time Intervals Smaller Than One Second

The `info("tickcount")` function can be used to calculate time intervals down to 1/60th of a second (0.0165 second). This function returns the number of 1/60th second intervals since the computer was turned on. Here is an example that uses this function to delay for 1/4 second.

```
local beginDelay
beginDelay=info("tickcount")
loop
   nop
while info("tickcount")<beginDelay+15
```

This loop will delay for 1/4 second (plus or minus 1/60th second) on any computer, no matter what the processor speed.

SuperDates (combined date and time)

SuperDates combine the date and time into a single number…the number of seconds since January 1, 1904. SuperDates make it easy to calculate time intervals across multiple days. However, SuperDates take up more storage than regular dates, and are not as easy to work with. In addition, SuperDates are limited to dates between 1904 and 2040. Panorama has several functions for working with SuperDates.

Function	Reference Page	Description
superdate(date,time)	Page 5822	This function converts a regular date and a regular time into a superdate. The date parameter is a regular Panorama date (see "Converting Between Dates and Text" on page 1249). This date must be between 1904 and 2040 A.D. The time parameter is the number of seconds since midnight, usually computed with the seconds(or time(functions (see "Converting Between Times and Text" on page 1255). The result is a single value that combines both the date and time into a single number that can be used for multi-day calculations.
regulardate(number)	Page 5643	This function extracts a regular date (number of days from January 1, 4713 B.C.) from a superdate. You can then format this date with the datepattern(function (see "Converting Between Dates and Text" on page 1249), as is shown in this example (which assumes that the field or variable Arrival contains a SuperDate value). `datepattern(regulardate(Arrival),"Month dd, yyyy")`
regulartime(number)	Page 5644	This function extracts a regular Panorama time (seconds since midnight) from a SuperDate. You can then format this date with the timepattern(function (see "Converting Between Times and Text" on page 1255), as is shown in this example (which assumes that the field or variable Arrival contains a SuperDate value). `timepattern(regulartime(Arrival),"hh:mm am/pm")`
superdatestr(number)		Converts a number containing a superdate to text in standard format (for example 4/20/03 9:56 AM).
supernow()		This function returns the number representing the current date and time as a superdate.

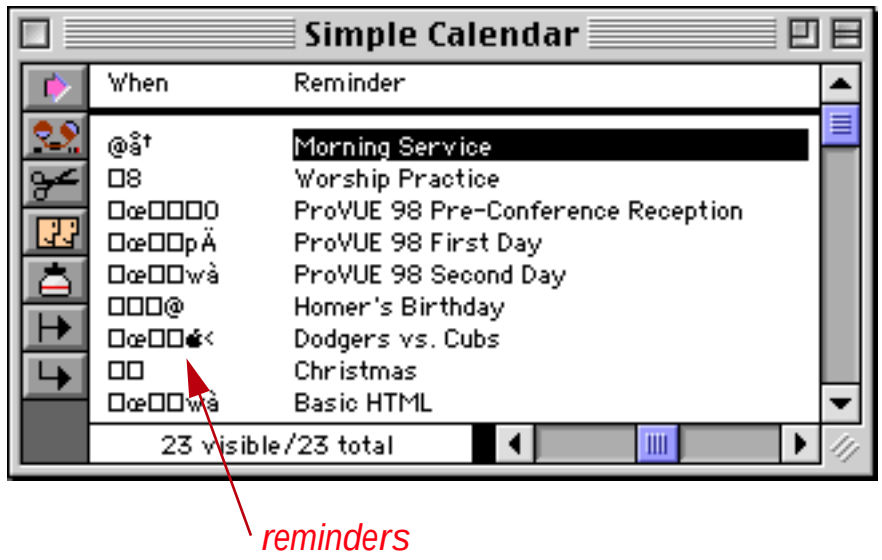
Reminders

A reminder is a special data type that holds scheduling information. Reminders are usually used in calendar database applications. A reminder is a raw binary data item 30 bytes long (stored in a text field or variable) and contains the following information:

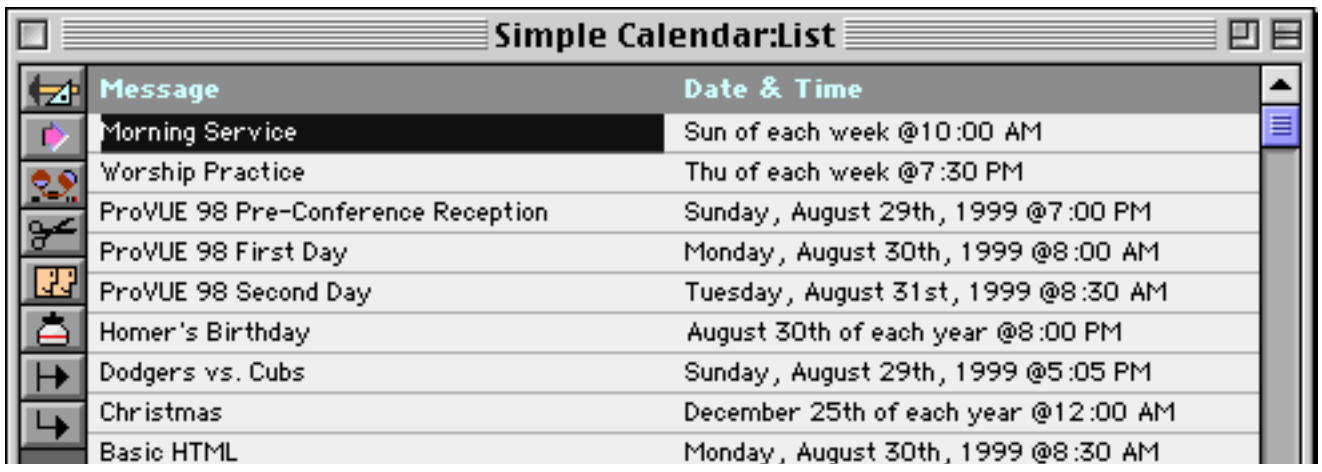
- The reminder type (either appointment or to-do)
- The reminder date, or recurring date information (july 12, every tuesday, etc.)
- The reminder time (3:30pm, 7:20am, etc.)
- Alarm status
- Completion status (to-do only)
- Priority (to-do only)

Notice that the reminder only contains scheduling information. It does not contain any message or other information about the event. If there is a message associated with a reminder (for example, Lunch with Bob) it should be stored in a separate field or variable.

Although reminders can be kept in a variable, they are usually kept in a database field. Here's what reminder data looks like in the data sheet.



As you can see, reminders don't look like much in the data sheet. They are usually displayed in a form using the `remindercaption(` function (see "Reminder Functions" on page 1263).



Reminders can also be displayed in a monthly calendar format. See "Building a Calendar" on page 960 for step by step instructions on setting up this format using a Super Matrix object.

Appointments vs. To-Do's

There are two different types of reminders: Appointments and To-do's. **Appointment** reminders are used for anything that has a definite, fixed, time: appointments, birthdays, meetings, etc. Once the time has passed the appointment is no longer relevant. For example, it won't do much good to be reminded that your spouse's birthday was yesterday!

To-do reminders have a completion status as well as a time and date. For example, suppose you set up a to-do reminder to order parts on Monday. If you don't get around to it, you'll still want it on your to-do list on Tuesday, and again on Wednesday etc. until you actually do order the parts. To-do reminders remain active until the task is completed (or at least it is marked as completed!)

Creating and Modifying a Reminder

Although reminders can be in a variable, they are usually kept in a database field. For the following example, we are going to assume that we have a database that contains fields named Reminder and Message. Both of these must be text fields.

To create a new reminder, you'll need to add a new record to the database, then use the `buildreminder` statement to allow the user to edit the reminder (see "BUILDREMINDER" on page 5078). Here's a procedure to do that (see "Writing a Procedure from Scratch" on page 1352).

```
addrecord
buildreminder today(),now(),0,Reminder
reminder Reminder,Message
```

The `buildreminder` statement creates a new reminder. This statement has four parameters:

```
buildreminder date,time,type,field
```

The first parameter, date, is the date for this reminder. Since usually you are going to have the user edit the reminder right away, this is just a default date to get them started. In the example above we used today's date.

The second parameter, time, is the time for this reminder. Again, this is usually just a default until the user edits the time. In the example above we used the current time.

The third parameter, type, specifies whether this is an appointment (0) or a to-do reminder (1).

The fourth parameter, field, specifies what field the new reminder should be placed into.

Another way to create a new reminder is with the `reminder(` function (see "REMINDER(" on page 5648). This is similar to the `buildreminder` statement, but with an important difference. The function has three parameters:

```
reminder(date,time,type)
```

The first parameter specifies the date for the new reminder. However, do not use the date data type here. This function wants to see text that describes the date, for example "2/7/96", "july 3", or "last tuesday".

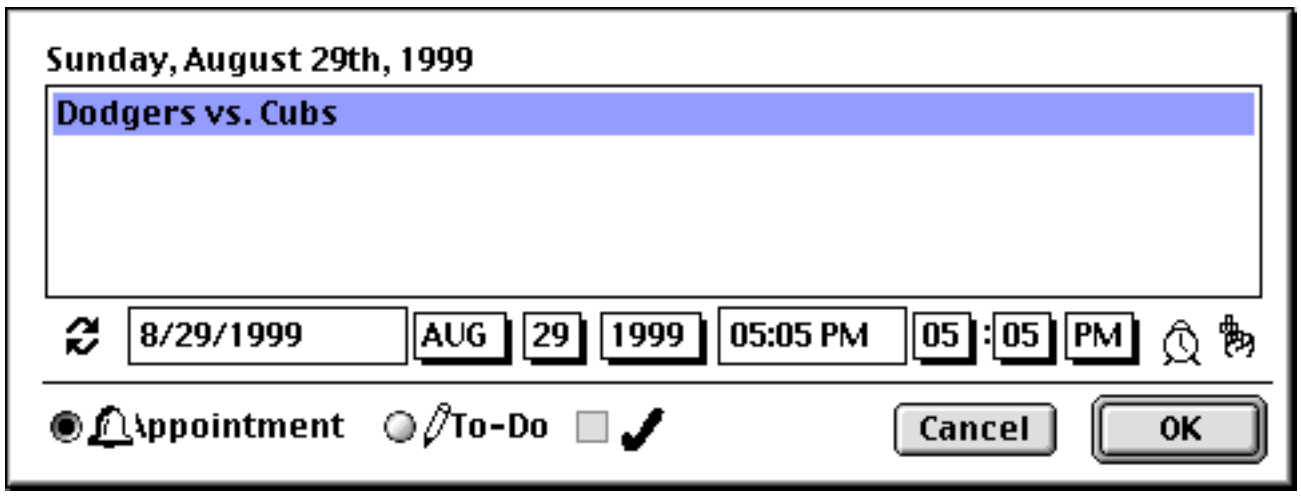
The second parameter specifies the time for the new reminder. However, do not use a number here. This function wants to see text that describes the time, for example "5:20 pm".

The third parameter specifies whether this new reminder should be an appointment ("a") or a to-do reminder ("t").

Here is our example rewritten to use the `reminder(` function:

```
addrecord
Reminder=reminder("today","12:00 pm","a")
reminder Reminder,Message
```

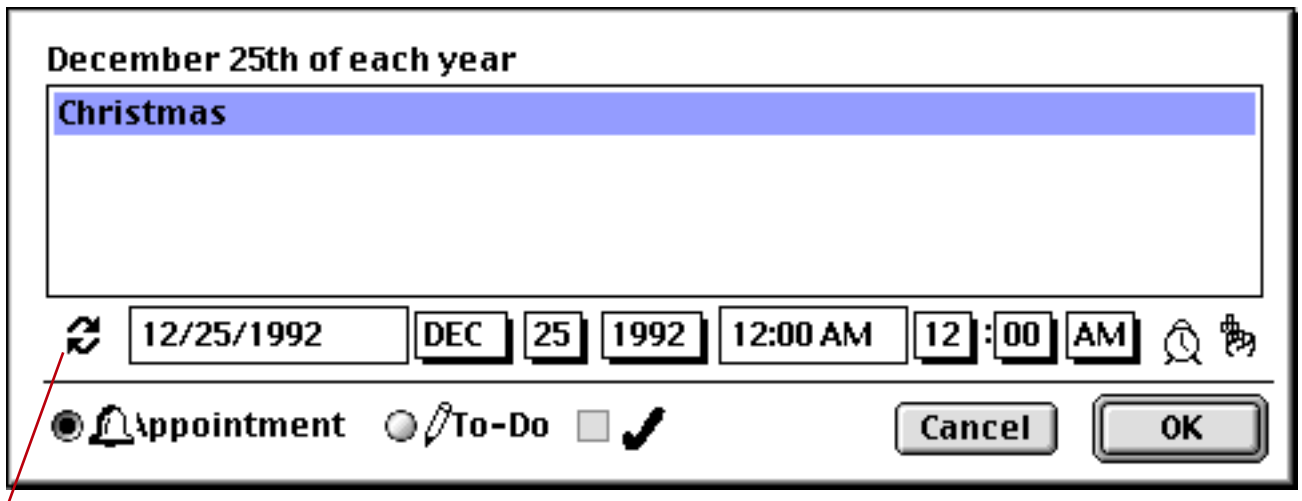
The `reminder` statement is used to edit reminders. This statement displays a dialog that allows the user to set up all the reminder information: date, time, message, alarm status, etc. The reminder statement has two parameters: the field containing the reminder, and the field containing the message.



Any time the user wants to edit a reminder the reminder statement should be used. You may want to set up a procedure that gets triggered whenever the user double clicks on the reminder display. This procedure only needs a single line:

```
reminder Reminder,Message
```

The reminder dialog has a subdialog for setting up repeating reminders.





Using this subdialog you can set up reminders that repeat every day, every week, every month, every quarter or once per year.

Reminder Functions

The functions listed below can build and work with reminders.

Function	Reference Page	Description

reminder(date,time,type)	Page 5645	This function builds a new reminder. There are three parameters: date, time and type. Date is the date for the new reminder (the function cannot create recurring reminders). However, you should not use a number here as you do with most date functions. You should use text that describes the date, for example "5/25/03" or "next tuesday". Time is the time for the new reminder. However, you should not use a number here as you do with most time functions. You should use text that describes the time, for example "5:22 pm".Type is the type of the new reminder: "a" for appointments or "t" for to-do's.
		This example adds a new reminder next tuesday at 2pm. The procedure stores this reminder in a field called Schedule:
		`addrecord`
		`Schedule=reminder("next tue","2:00 pm","a")`
		Another way to build a reminder is with the `buildreminder` statement, see reference page 5078. You can also edit a reminder with a user friendly dialog using the `reminder` statement (reference page 5645). This dialog allows you to set up recurring reminders (every tuesday, 15th of each month, etc.) and to configure alarms.

remindercaption(reminder)	Page 5649	This function extracts the date from a reminder as formatted text that describes when the reminder will occur. The table below shows typical examples of how different reminder frequencies will be formatted by this function:
		`Once only reminder:  Tuesday, May 16th, 2004`
		`Annual reminder:  August 8th of each year`
		`Quarterly reminder:  First day of each quarter`
		`Monthly reminder:  5th day of each month`
		`Last day of each month`
		`2nd Wed of each month`
		`Weekly reminder:  Tue of each week`
		`Daily reminder:  Every day`
		The formula below displays the reminder date in a field called Schedule. This formula could be used in an auto-wrap text object or a Text Display SuperObject™.
		`remindercaption(Schedule)`

remindercompare(reminder,date)	Page 5650	This function checks to see if a reminder occurs on a specified date. Reminder is the reminder you want to compare. Date is the date you want to compare with the reminder. This function returns either true or false. It will return true if the reminder will occurs on the specified date, including a repeating reminder that falls on that date. The example procedure below uses this function to select all reminders that will occur on next tuesday. `select remindercompare(Schedule,date("next tue"))`

Function	Reference Page	Description

reminderdate(reminder)	Page 5651	This function extracts the date from a reminder. If a reminder repeats, the function will try to come up with the most appropriate single date. The table below shows how different reminder frequencies will be handled by this function:
		
		`Once only reminder: Actual date`
		`Annual reminder: Next occurence of this reminder`
		`Quarterly reminder: 0`
		`Monthly reminder: 0`
		`Weekly reminder: 0`
		`Daily reminder: 0`

reminderpriority(reminder)	Page 5652	This function extracts the priority of a to-do reminder (completed / not completed) from a reminder. The function returns a number from 0 to 3: 0 for the lowest priority to 3 for the highest priority.
remindertime(reminder)	Page 5653	This function extracts the time (number of seconds since midnight) from a reminder. The result of this function can be used with functions like timepattern((see “Converting Between Times and Text” on page 1255).
remindertodo(reminder)	Page 5654	This function extracts the status of a to-do reminder (completed / not completed) from a reminder. The function returns a number: 0 if the to-do has not been completed (or the reminder is an appointment type), or 1 if the to-do has been completed.
remindertype(reminder)	Page 5655	This function extracts the type (to-do or appointment) from a reminder. This function returns a number: 0 if the reminder is an appointment type, or 1 if it is a to-do.

Alarms

If you have the optional Team Alarm extension installed (Mac OS 9 only), you can be notified of your reminders even when Panorama is not currently running. To do this, the Team Alarm extension keeps a separate private list of pending alarms. This list is in a special format that cannot be accessed by Panorama. However, this extra alarms database is updated automatically when a reminder is updated with the `reminder` statement (see “REMINDER” on page 5645). However, if you modify a reminder yourself without using the `reminder` statement, you’ll need to make sure the Team Alarm list is also updated. There are three statements for doing this: `alarmedit` (see “ALARMEDIT” on page 5023), `alarmdelete` (see “ALARMDELETE” on page 5022), and `alarmreset`(see “ALARMRESET” on page 5024).

True/False Formulas

In Panorama as in most programming languages, control flow decisions are made on the basis of formulas that are either true or false. The most basic true/false formula compares two values to see if they are equal.

```
PaymentMethod="C.O.D."
```

This formula will compare the value in the field PaymentMethod with C.O.D. The result will be true if PaymentMethod is C.O.D., and false if it contains anything else (for example Check, Cash, Visa, etc.).

Comparison Operators

Panorama has about a dozen different operators that can compare two values and produce a true false result. You can type these operators in yourself (see "Special Characters" on page 1202), or you can use the **Operator** sub-menu in the Function menu to type in the symbols for you. The table below lists the universal comparison operators. These comparison operators will work with any type of data: text, numeric, or date.

Operator	Example	True/False Meaning	Notes
=	A=B	is A equal to B?	
≠	A≠B	is A not equal to B?	Not available on PC
	A<>B	is A not equal to B?	
>	A>B	is A greater than B?	
≥	A≥B	is A greater than or equal to B?	Not available on PC
>=	A>=B	is A greater than or equal to B?	
<	A<B	is A less than B?	
≤	A≤B	is A less than or equal to B?	Not available on PC
<=	A<=B	is A less than or equal to B?	

All of the above operators require that A and B be the same data type. In other words, you cannot directly compare numbers to text, or text to dates. If A and B are different types you must convert them to the same type before comparing them, using the str(, val(, pattern(, date(or datepattern(functions. See "Converting Between Numbers and Strings" on page 1227 and "Converting Between Dates and Text" on page 1249 for more information on these functions.

Panorama also has a number of specialized comparison operators that work only with the text data type.

Operator	Example	True/False Meaning
beginswith	A beginswith B	does A begins with B?
endswith	A endswith B	does A end with B?
contains	A contains B	does A contain B?
notcontains	A notcontains B	does A not contain B?
soundslike	A soundslike B	does A sound like B (phonetically)?
match	A match B	does A match the wildcard pattern in B (disregarding upper/lower case)?
matchexact	A matchexact B	does A exactly match the wildcard pattern in B?
notmatch	A notmatch B	does A not match the wildcard pattern in B (disregarding upper/lower case)?
notmatchexact	A notmatch B	does A not exactly match the wildcard pattern in B?

Each of these operators deserves a more complete explanation, so here they are.

A beginswith B

This operator checks to see if the text in A begins with the characters in B. For example, the formula below will determine if the Name begins with the letters Dr. .

```
Name beginswith "Dr."
```

This formula will be true if the name is Dr. Robert Johnson, and false if the name is Mark Reynolds.

Note: The beginswith operator does not worry about upper or lower case, so DR. ROBERT JOHNSON or dr. robert johnson will also produce true results. If upper and lower case are important to you use the matchexact operator.

A endswith B

This operator checks to see if the text in A ends with the characters in B. For example, the formula below will determine if the Name ends with the letters D.D.S. .

```
Name endswith "D.D.S."
```

This formula will be true if the name is Ronald Nelson, D.D.S, and false if the name is Mark Reynolds.

Note: The endswith operator does not worry about upper or lower case, so ronald nelson, d.d.s. would also produce a true result. If upper and lower case are important to you use the matchexact operator.

A contains B

This operator checks to see if the text in A contains the characters in B. For example, the formula below will determine if the Address contains the letters box.

```
Address contains "box"
```

This formula will be true if the address is P.O. Box 5328, and false if the address is 6389 E. Wilson Blvd.

Note: The contains operator does not worry about upper or lower case, so P.O. BOX 5328 and p.o. box 5328 would also produce true results. If upper and lower case are important to you use the matchexact operator.

A notcontains B

This operator checks to see if the text in A does not contain the characters in B. This is the exact opposite of the contains operator. For example, the formula below will determine if the Address contains the letters box.

```
Address notcontains "box"
```

This formula will be true if the address is 6389 E. Wilson Blvd, and false if the address is P.O. Box 5328.

Note: This same function could also be performed by combining the not operator with the contains operator in the formula: `not (Address contains "box")`.

A soundslike B

This operator checks to see if the text in A “sounds like” the text in B. For example, the formula below will determine if the LastName sounds like the name Smith.

```
LastName soundslike "Smith"
```

This formula will be true if the name is Smith, Smyth or Smythe, and false if the name is Jones or Williams.

The method Panorama uses to determine whether two values sound alike is called “soundex.” This technique is not very exact, and often will produce extra matches that you might not think really sound similar. However, it almost never fails to match on names that do sound similar, so it is a good starting point when you are not sure of an exact spelling.

The soundex technique does require that the first letter of the two values match. For example even though we think they sound alike, Christy and Kristy will not match because the first letter is different.

A match B

This operator checks to see if the text in A matches a pattern you specify in B. The pattern allows you to set up very flexible "wildcard" matches where some characters must match and some don't have to.

The pattern should combine normal characters, which must match the text in A, and wildcard characters: **?** and *****. The **?** wildcard character will match any character. The ***** wildcard character (asterisk) will match a variable number of characters. The best way to understand wildcard matches is probably to look at a few examples.

Our first example uses the pattern j*johnson. With this pattern the name must begin with j (or J) and end with johnson (or Johnson, etc.) The characters in between don't matter.

```
Name match "j*johnson"
```

This formula will produce a true result for names like Jim Johnson, Jack Johnson, Joe Johnson, etc. The formula will also be true for names like J346 Ujohnson or J@#opcjohnson.

The second example uses the pattern 926??. With this pattern the zip code must begin with 926 and must be 5 digits long. (Our example assumes that ZipCode is a text field, not a numeric field.)

```
ZipCode match "926??"
```

This formula will produce a true result for zip codes like 92631 or 92685 but a false result for zip codes like 89324 or 92685-0301. Here's a variation that will work with 5 or 9 digit zip codes. The ?? characters mean that there must be at least five digits, while the * means that any extra characters are ok.

```
ZipCode match "926??*"
```

This formula will produce a true result for zip codes like 92631, 92685 or 92685-0301, but a false result for 926 or 9262.

Don't forget that a space is a normal character. The example below checks for people with a middle initial. The pattern looks for any number of characters followed by a space, followed by a single character, followed by a period, followed by another space, followed by any number of characters.

```
Name match "* ?. *"
```

This formula will produce a true result for Robert E. Lee or Winston O. Link, but a false result for Frank Tesh, Billy Martin, or Sara Jessica Parkman.

The match operator can be used to simulate the beginswith, endswith and contains operators. The table below shows the equivalent match formulas for each of these operators.

These formulas…	are the same as these.
`A match B+"*"`	`A beginswith B`
`A match "*"+B`	`A endswith B`
`A match "*"+B+"*"`	`A contains B`

Note: The match operator does not worry about upper or lower case. If upper and lower case are important to you, use the matchexact operator.

A matchexact B

This operator checks to see if the text in A matches a pattern you specify in B. This operator works exactly the same as the match operator, except that the normal characters must match exactly, including upper and lower case. For example, the formula below

```
Name matchexact "J*Johnson"
```

will produce a true result for Jeff Johnson, but a false result for JEFF JOHNSON. (However, JEFF Johnson would produce a true result.)

You can use the matchexact operator instead of beginswith, endswith, or contains if you need an exact upper and lower case match.

A notmatch B

A notmatchexact B

These operators are the exact opposite of match and matchexact.

A like B

This operator checks to see if the text in A matches a pattern you specify in B. This operator is similar to the matchexact operator, but it uses different wildcard characters: **%** and **_** instead of * and ? Here are some examples showing both formats:

```
Name matchexact "J*Johnson"
Name like "J%Johnson"

Zip matchexact "926??*"
Zip like "926__%"
```

The like operator is included for compatibility with SQL servers. The like operator can be used for selecting a subset from a SQL master file, the match operator cannot.

Combining Comparisons

The basic comparisons described in the previous section can be combined together for more complicated decisions. There are four basic operators that can combine or modify decisions: **and**, **or**, **xor**, and **not**.

Operator	Sample	Description
and	A and B	true if both A and B are true
or	A or B	true if either A or B are true
xor	A xor B	true if A and B are different
not	not A	true if A is false

A and B

The **and** operator combines two true/false formulas together so that the result is only true if both formulas are true. The example procedure below determines if a person is a teenager.

```
if Age≥13 and Age<20
   Status="Teenager"
endif
```

The result of the formula is only true if the person is 13 or older and less than 20.

A or B

The **or** operator combines two true/false formulas together so that the result is true if either one of the two formulas are true. The example below determines if a transaction is being paid with a credit card.

```
if PaymentMethod="Visa" or PaymentMethod="MasterCard"
   Terms="Credit Card"
endif
```

The result of the formula is only true if the payment method is Visa or MasterCard.

Notice that each side of the **or** operator must contain a complete formula. The formula below looks right in English, but will not work in Panorama. The example below is WRONG:

```
if PaymentMethod="Visa" or "MasterCard"      /* WILL NOT WORK !! */
```

There must be a comparison on both sides of the **or**, as shown in the first example.

A xor B

The **xor** (short for exclusive-or) operator is a bit tricky. **Xor** combines two true/false formulas together so that the result is true if one of the two formulas is true, but false if both are true or both are false. Another way to put it is that the result will be true if A and B are different, but false if they are the same. The example below determines if two shoes are a pair.

```
if Shoe1="Left" xor Shoe2="Left"
   message "These shoes are a pair"
endif
```

The result of the formula is only true if one shoe is Left and the other shoe is Right (or to be more precise, not Left).

not A

The not operator reverses a true-false formula. If the result was true, now it will be false. If it was false, now it will be true.

```
if (not (Shoe1="Left" xor Shoe2="Left"))
   message "These shoes are not a pair!"
endif
```

Note: This example shows that if **not** is used as the very first operator in a formula in a procedure, you must surround the entire formula with an extra pair of parentheses. If `not` is in the middle of the formula the extra parentheses are not necessary. The parentheses are also not necessary if the formula is not in a procedure (in the Design Sheet or a **Formula Fill** dialog, for example).

Equals Comparison vs. Assignment

If you have skipped ahead to read about procedures you know that the equals sign is used to assign a value to a field or variable. The example formula we used earlier to compare two values:

```
PaymentMethod="C.O.D."
```

would also be the same formula used to assign the value C.O.D. to the field or variable PaymentMethod. At first glance this may appear ambiguous…the same formula is used to compare two values and to assign a value. How do we know when we are assigning and when we are comparing? The answer lies in the context in which the formula is found.

In a procedure, an assignment is always by itself, not part of a larger statement. A true-false formula is always part of another statement, for example `if`, `case`, `until`, `while`, `stoploopif`, `repeatloopif`, `find`, `select`. Here's an example that shows two formulas that look almost the same, but one is a true-false formula and one is an assignment.

```
if PaymentMethod="C.O.D."
   ShippingMethod="UPS"
endif
```

The first formula, `PaymentMethod="C.O.D."`, is part of the `if` statement. This formula means: Is the field (or variable) PaymentMethod equal to C.O.D. (true / false)?

The second formula, `ShippingMethod="UPS"`, is not part of any statement, but stands alone, so this is an assignment. The statement means: Take the value UPS and copy it into the field or variable named ShippingMethod.

If an assignment has more than one equals sign, the first equals sign is for the assignment and the rest are for comparisons. The example assignment below compares B and C. If they are equal (true) the value -1 will be copied into A. If they are not equal (false) the value 0 will be copied into A.

```
A=B=C
```

In other words, `A` becomes the result of the comparison between `B=C`, or `A = (B=C)`.

True/False Values

For purposes of calculation, Panorama treats true and false as numbers: true is -1 and false is zero. Panorama also has two functions that directly generate these values.

Function	Reference Page	Description
true()		This function always returns true (-1).
false()		This function always returns false (0).

Like any other number, you can store a true / false value in a field or variable and then use it later. The example below calculates whether a person is a teenager, then uses that information later.

```
local Teenager
Teenager=Age≥13 and Age<20
...
if Teenager
   Price=4.50
else
   Price=6.00
endif
```

Notice that the `if` statement doesn't need to compare, it simply uses the result of the comparison that was calculated earlier. In fact, the `if` statement (and all other statements that use true / false logic) can use any formula that produces a numeric integer result. The value 0 will be regarded as false, and any non-zero value will be regarded as true. The example below will be true if the length of the name is non-zero.

```
if length(Name)
   yesno "Is this a home address?"
   ...
endif
```

The first line of this example could also have been written `if length(Name)›0`. The result is the same either way.

The ? Function

The ?(function allows a formula to make a decision. Will it be door number 1 or door number 2? The function uses a true-false value to pick from one of two values. The syntax for this function is like this.

```
?(decision-value,true-value,false-value)
```

The first parameter, decision-value, is used to pick which of the two choices will be returned as the final value, the true-value or the false-value.

For example, the formula below can be used to calculate a 10% discount if the quantity is 100 or more—

```
?( Qty<100 , Price , Price*0.9 )
```

The decision is based on the comparison `Qty<100`. If Qty is less than 100, the ? function picks the second parameter, `Price`. But if the quantity is 100 or more, the ? function will pick the third parameter, `Price*0.9`, for a 10% discount.

If you need to pick from three or more choices you can nest several ? functions together. For example, this formula shows how you can add a third discount level (20% for quantities of 500 or more)—

```
?( Qty<100 , Price , ?(Qty<500 , Price*0.9 , Price*0.8 ))
```

Although these examples have used numeric data, text can also be used for either the true-false logic or the choices. The formula below, for example, could be used by a movie theater to check if a person is a child or an adult.

```
?( Age≤12 , "Child" , "Adult" )
```

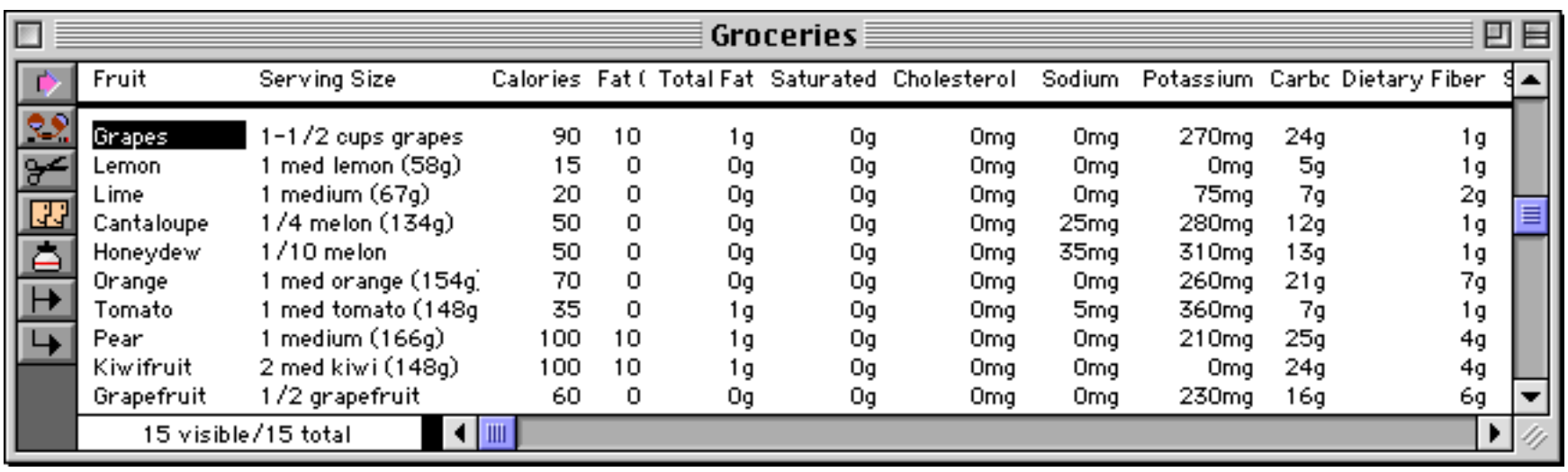
Note: The ? function always evaluates all three parameters you give it, even though it really uses only two of the parameters. This means that you cannot use the ? function to avoid errors (for example divide by zero errors) because the error will happen before the ? function decides which parameter to use (use the divzero(function to avoid divide by zero problems).

Linking With Another Database

Many database applications require multiple database files working together. For example, organizing a company's order entry operations usually requires an invoice file, an inventory / price list file, and possibly a customer file. The primary method for accessing information in other databases is the **lookup(** function (and other related functions). This function can search for and retrieve information from any open database. Need to look up a price or a customer's credit limit? Chances are the lookup(function is the tool for the job.

When you look up information manually (for example, looking up someone's number in the phone book), you are actually performing a multi-step process. You start with one piece of information—a person's name, for example. The first step is to locate the correct phone book. Once you've located the correct book, you must search through it to find the name of the person you are looking for. When you find the name, the final step is to copy down the person's phone number.

Panorama's lookup(function follows a similar process when it looks up data. For example, suppose you want to find out the number of calories in an orange using the database shown here.

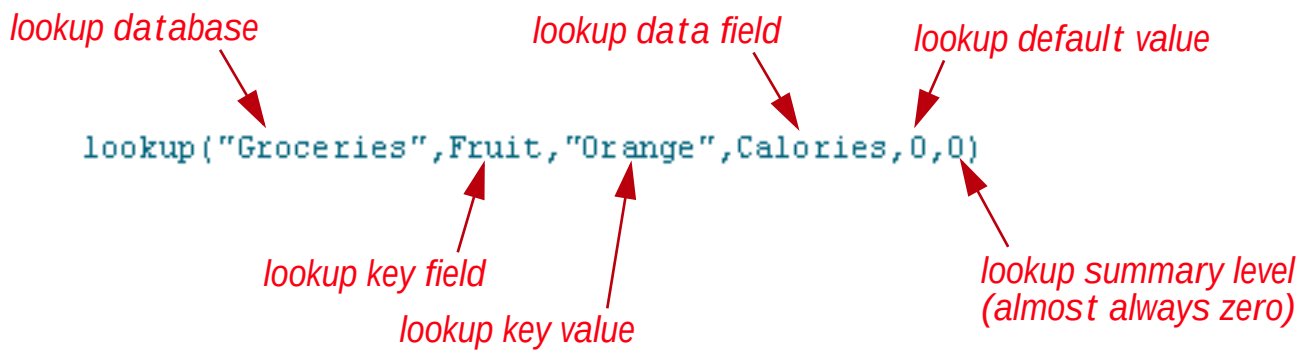


Groceries

Fruit	Serving Size	Calories	Fat (	Total Fat	Saturated	Cholesterol	Sodium	Potassium	Carbc	Dietary Fiber
Grapes	1-1/2 cups grapes	90	10	1g	0g	0mg	0mg	270mg	24g	1g
Lemon	1 med lemon (58g)	15	0	0g	0g	0mg	0mg	0mg	5g	1g
Lime	1 medium (67g)	20	0	0g	0g	0mg	0mg	75mg	7g	2g
Cantaloupe	1/4 melon (134g)	50	0	0g	0g	0mg	25mg	280mg	12g	1g
Honeydew	1/10 melon	50	0	0g	0g	0mg	35mg	310mg	13g	1g
Orange	1 med orange (154g	70	0	0g	0g	0mg	0mg	260mg	21g	7g
Tomato	1 med tomato (148g	35	0	1g	0g	0mg	5mg	360mg	7g	1g
Pear	1 medium (166g)	100	10	1g	0g	0mg	0mg	210mg	25g	4g
Kiwifruit	2 med kiwi (148g)	100	10	1g	0g	0mg	0mg	0mg	24g	4g
Grapefruit	1/2 grapefruit	60	0	0g	0g	0mg	0mg	230mg	16g	6g

15 visible/15 total

Here is the formula for looking up the number of calories in an orange. The parameters to the lookup contain all the information necessary to locate the information.



The first parameter is called the **lookup database**. It tells Panorama what database to look in for the information, in this case Groceries.

The second and third database tell Panorama how to search for the data you want. In this case Panorama is being told to "search through the Fruit column until you find Orange." The field to look in (in this case Fruit) is called the **lookup key field**. The data to look for (in this case Orange) is called the **lookup data value**. By the way, Panorama is very picky about the lookup data value. It must exactly match the value in the database, or Panorama won't find a match. In this case only Orange will work — not orange or ORANGE or even oRaNGe!

At this point we come to a fork in the road. Perhaps Panorama found Orange in the database, perhaps not. If it did the fourth parameter tells Panorama what to do next. This fourth parameter is called the **lookup data field**, and it may be any field in the lookup database. In this case it is Calories, so Panorama will lookup the value in the Calories field (70) and return it as the result of the function.

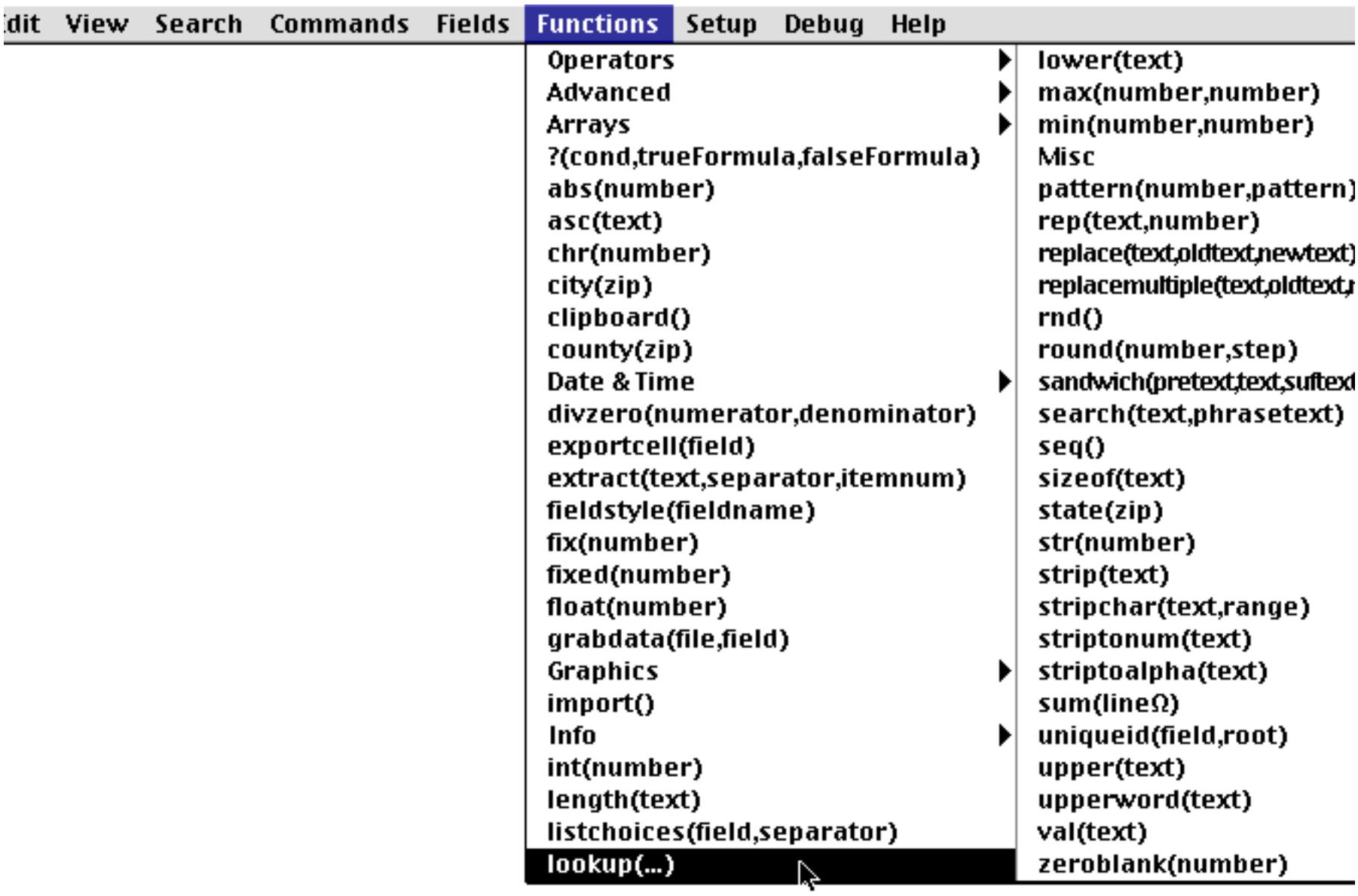
What if Panorama didn't find Orange in the database? In that case Panorama simply returns the value of the fifth parameter, the **lookup default value**. In this case the default value is 0. The default value should match the data type of the lookup data field. Since Calories is a numeric field, the default is also numeric. If the lookup data field had been a text field (for instance Serving Size) the default would need to be text (for example "").

The sixth and final parameter to the lookup function is the **lookup summary level**. This is the minimum summary level to be searched within the lookup database. Usually the lookup summary level is zero so that the entire lookup database will be searched. If the level is set to 1 through 7, only summary records will be searched. This is useful if you want to look up summary information (see "3-Step Summarizing" on page 393) while ignoring the raw data.

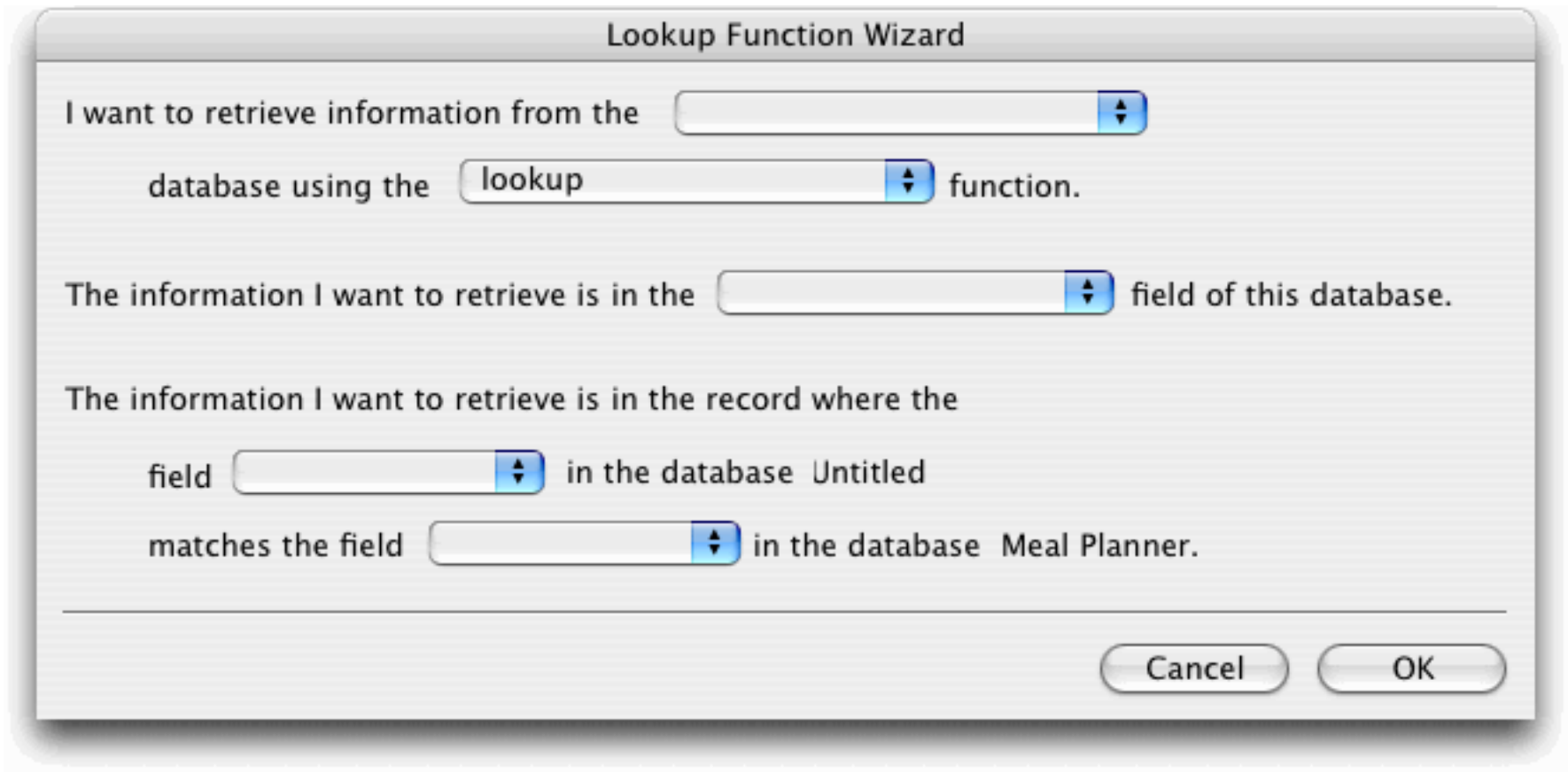
In this example the end result of the lookup is the value 70. The lookup(function is often used by itself, but a more complicated formula can take this value and perform additional computations. If the result of the lookup is a text value then all of the text functions described earlier in this chapter can be used to modify the result.

The Lookup Wizard

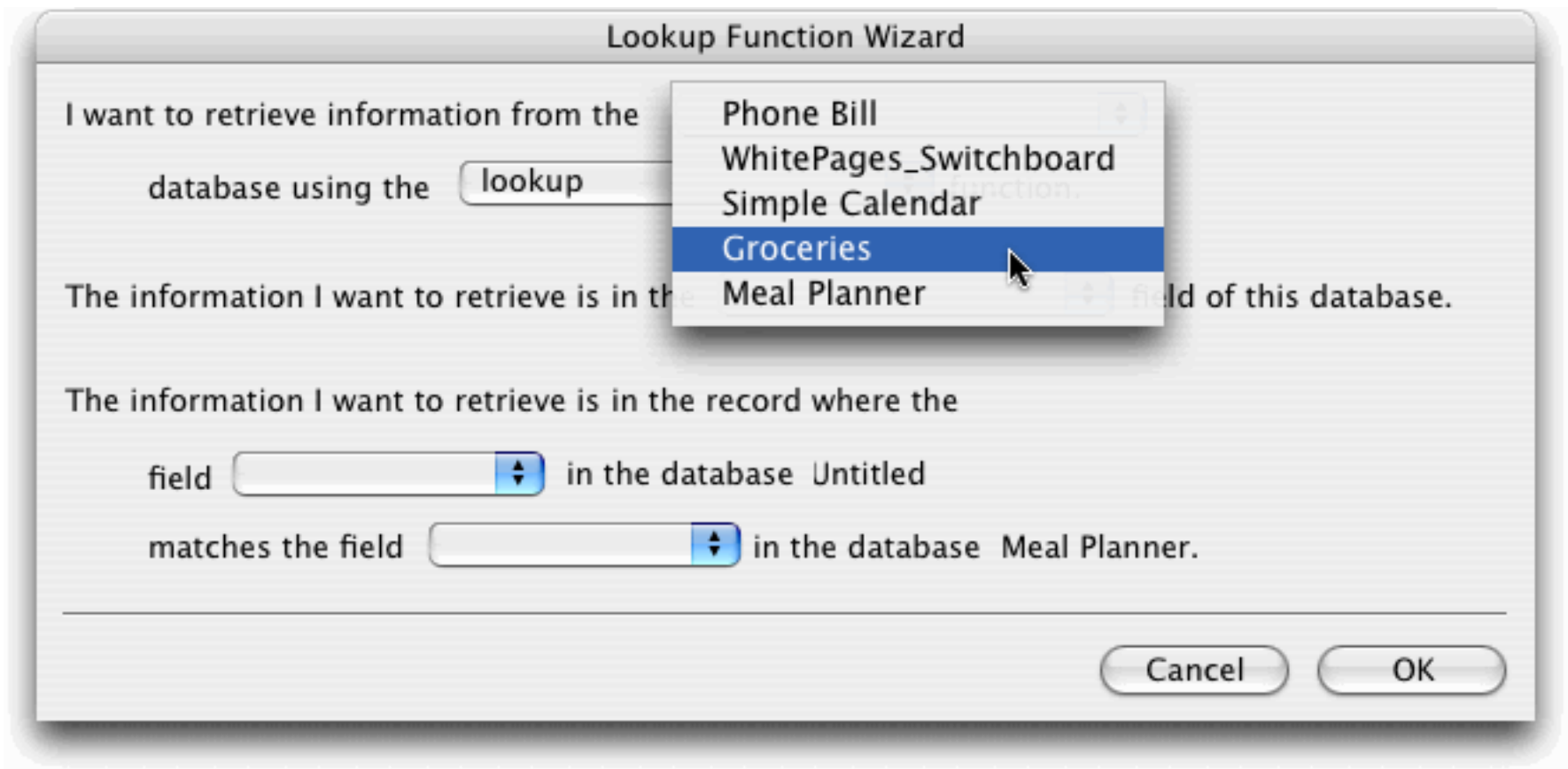
Since the lookup(function is kind of picky about all of it's parameters we've provided a "fill-in-the-blanks" dialog to help build the function. To open this dialog simply pull down the **Functions** Menu and choose **lookup(...)**.



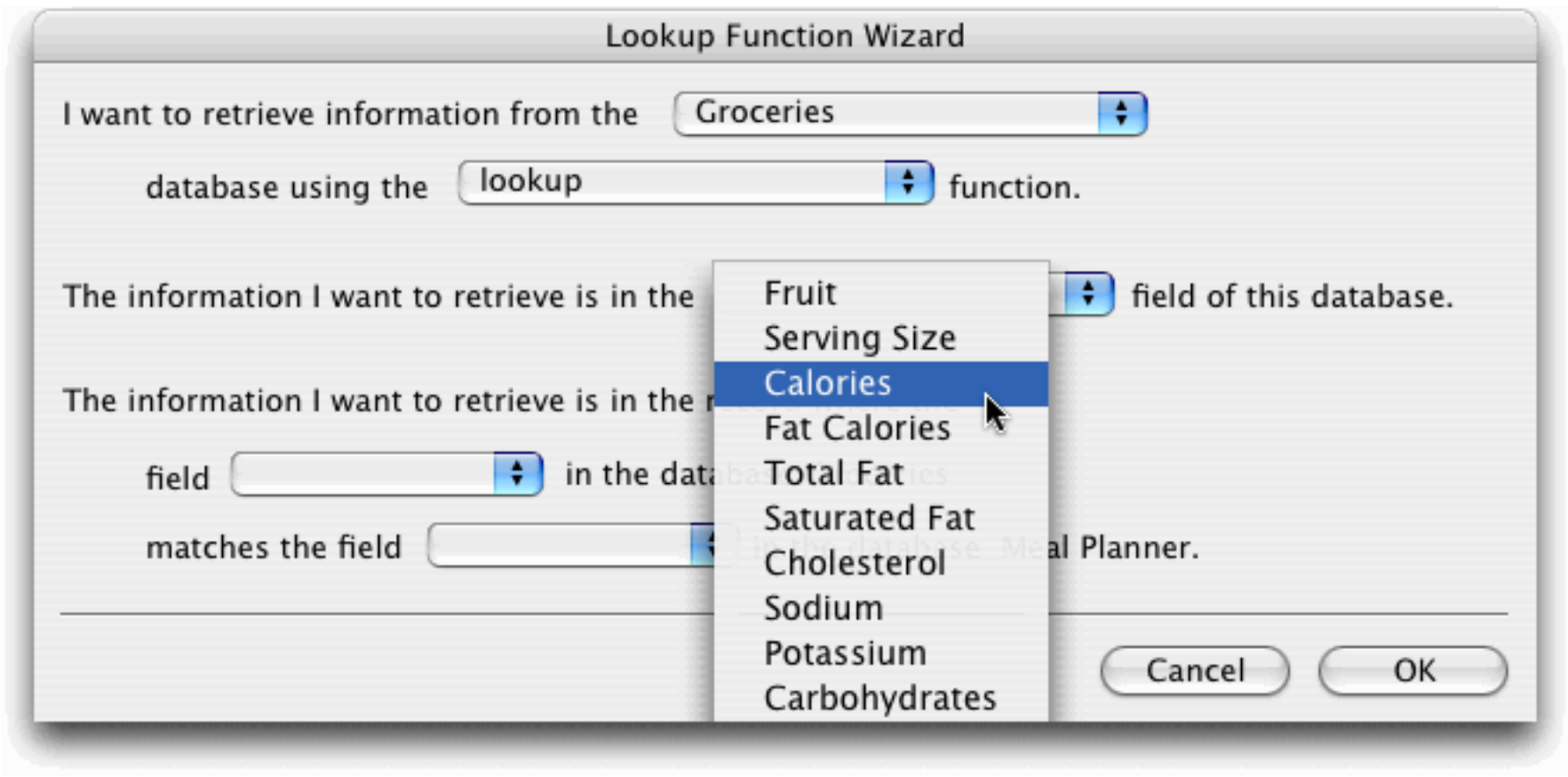
Now the lookup wizard dialog appears.



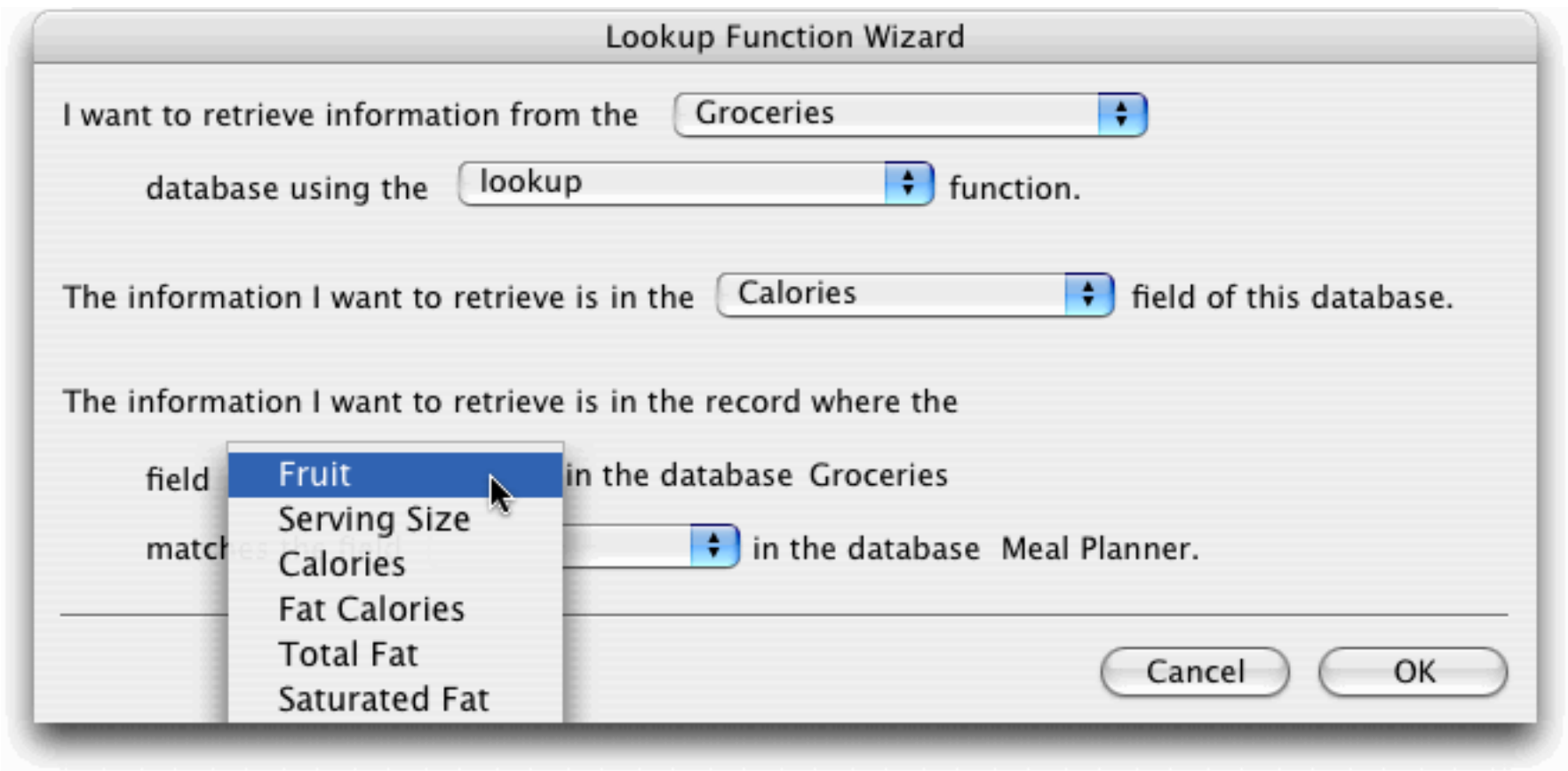
To create the lookup function, start at the top of the dialog and work your way down. Start by selecting the database to lookup from (in this case Groceries).



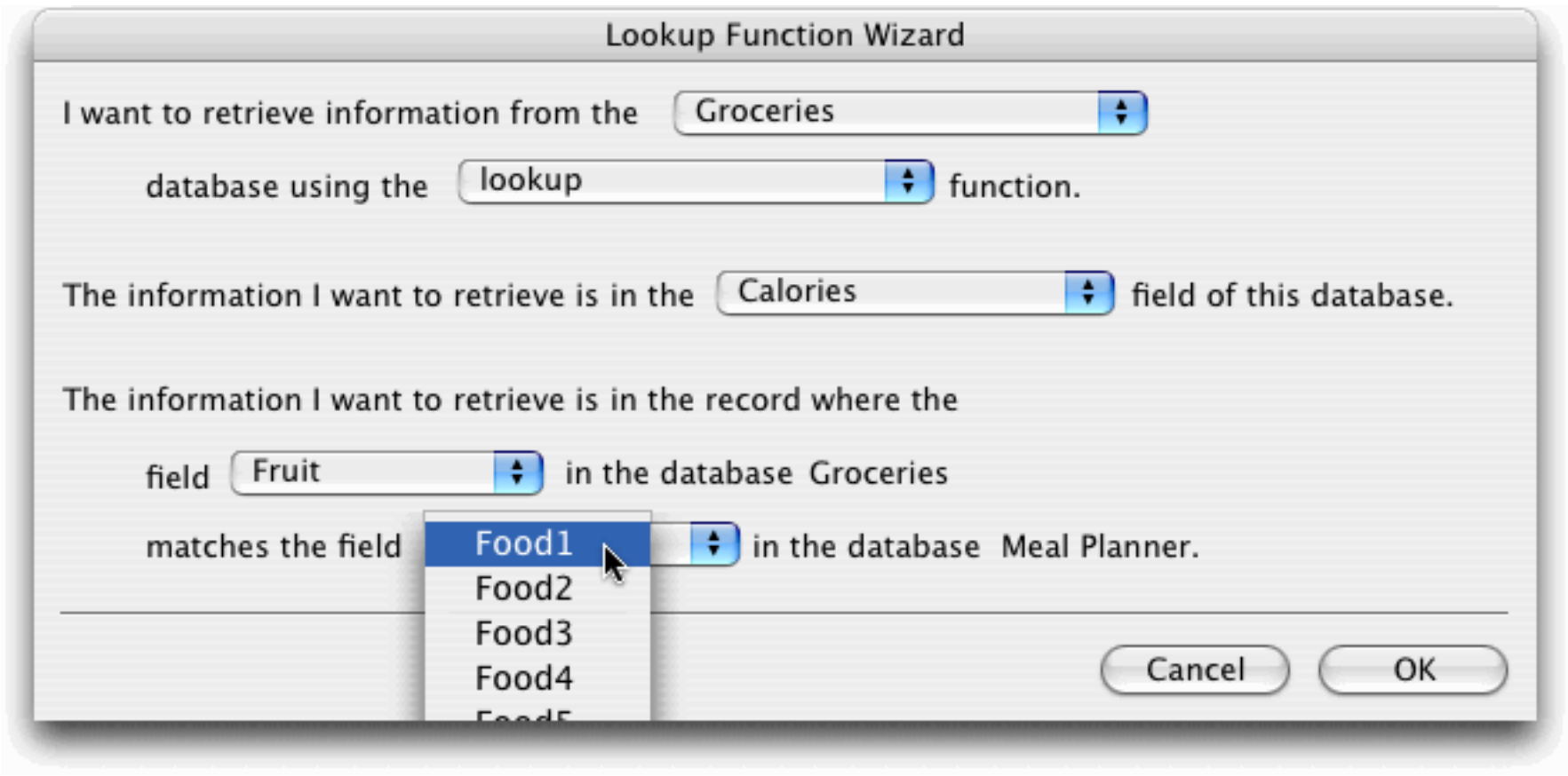
Next, choose the data you want to retrieve (the lookup data field, which will become the fourth parameter to the lookup function). In this case we want to retrieve the number of Calories.



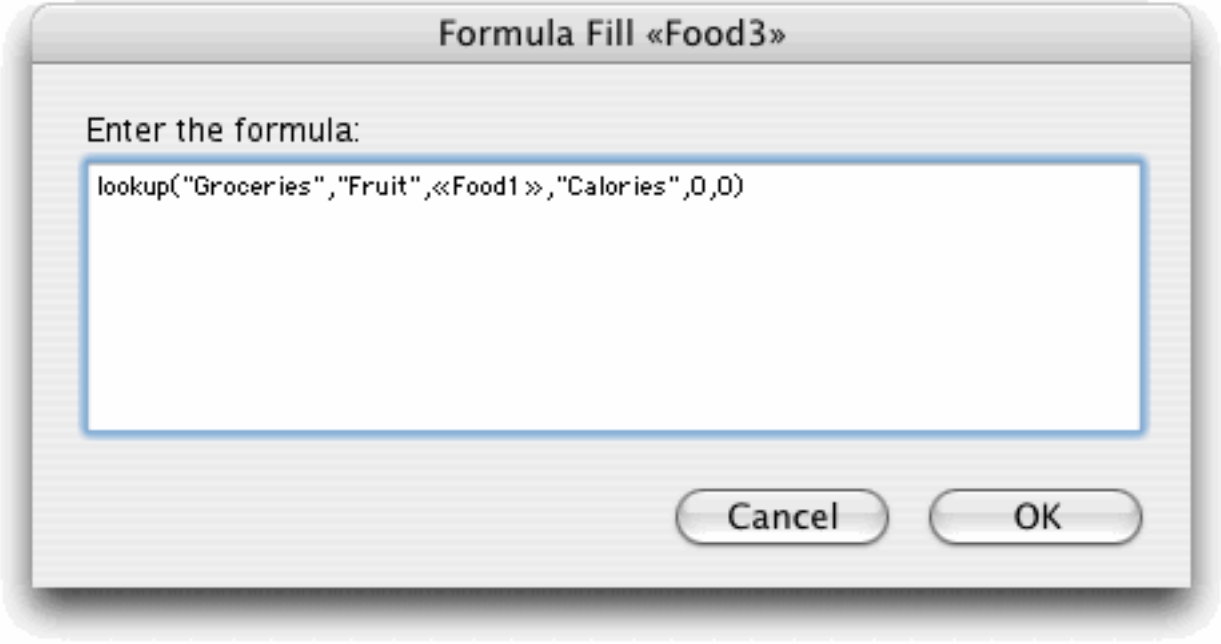
The next step is to choose the lookup key field, which in this case is Fruit.



Finally, choose the field containing the lookup key value. If there is no such field (perhaps the value is in a variable) then just choose any field and adjust the formula once the wizard is finished.



Once you've made all of the selections press **OK** to generate the finished formula. For example, if you were using the Formula Fill command the formula would look like this:



Type Mismatch Problems

One of the most common problems when setting up a lookup function is type mismatches. With some careful thought, however, you can avoid these problems.

The first source of type mismatch problems is the lookup key field and the lookup key value. The field and value must be the same type of data. In other words, if the lookup key field is numeric, the lookup key value must be numeric also. If necessary, you can convert a text key value into numeric with the `val(` function, or you can convert a numeric key value into text with the `str(` function (see "Converting Between Numbers and Strings" on page 1227 for details on both of these functions).

```
lookup("Catalog","Part#",val(Item),"Price",0,0)
```

Another source of type mismatch problems is the lookup data field. This field must have the same type of data as the field you want to store the result in. For example if you look up a price, the result must be stored in a numeric field.

If you need to store a numeric value in a text field, use the str(function to convert the value. The str(function should go outside the entire lookup function, for example

```
str(lookup("Catalog","Item",Desc,"Price",0,0))
```

Another source for type mismatch problems is the lookup default value. The default value should be the same type as the lookup data field. If the lookup data field is numeric, the default should be numeric (for example 0 or 100). If the lookup data field is text, the default should also be text (for example "" or "n/a").

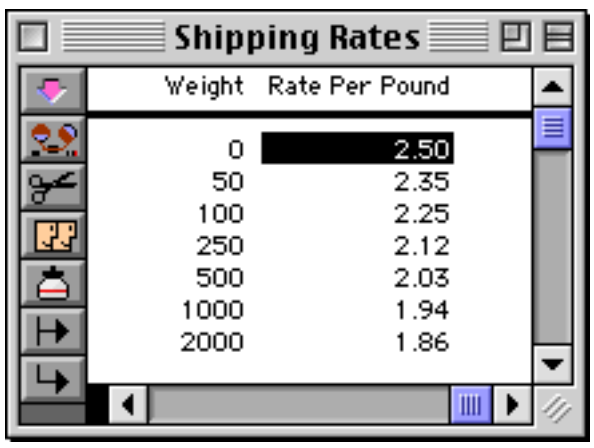
Lookup Variations

There are actually several different variations of the lookup function. All of the variations have the same six parameters. The standard lookup function locates the first occurrence of the key (nearest to the beginning of the file).

Function	Reference Page	Description
lookup(	Page 5500	This function searches for the first occurrence of the value within the lookup database. If there is more than one copy of the value in the database this function will find the one closest to the top.
lookuplast(	Page 5505	This function searches for the last occurrence of the value within the lookup database. If there is more than one copy of the value in the database this function will find the one closest to the bottom. However, there is one exception. If you are looking up within the current database Panorama will skip the current record. If the current record matches the key value then Panorama will skip backwards to the next matching record.
lookupselected(	Page 5513	This function searches for the first occurrence of the value within the selected records in the lookup database. Unselected (invisible) records are ignored. If there is more than one copy of the value within the selected records this function will find the one closest to the top.
lookuplastselected(	Page 5507	This function searches for the last occurrence of the value within the selected records in the lookup database. Unselected (invisible) records are ignored. If there is more than one copy of the value within the selected records this function will find the one closest to the bottom.
table(	Page 5827	The table function allows you to lookup data by an approximate match instead of exact match. If the table function does not find an exact match, it uses the next lower value. A common example is a shipping rate table. Rate tables do not have an entry for every possible weight. Instead, the table only lists weights where the shipping rate changes. For example, suppose a rate table contains entries for 100 pounds and 250 pounds, and you have a 158 pound package. The table function will return the rate for the next lower value, in this case the 100 pound rate.

Looking Up Rates in a Rate Table

The table(function is designed for looking up rates from a table. For example, this function can be used to look up shipping rates, tax rates, discount rates, or any kind of stepped rate where the rate changes according to a sliding scale. To illustrate this function, consider this shipping rate database.



For packages from 0 to 49.99 pounds the rate is 2.50 per pound. For packages from 50 to 99.99 pounds the rate is 2.35 per pound, from 100 to 249.99 the rate is 2.25 etc. Suppose we use a regular lookup function to look up the weight, like this.

```
lookup("Shipping Rates",Weight,PackageWeight,«Rate Per Pound»,0,0)
```

This formula will work fine for weights that appear in the table like 50, 100 and 250. But for other weights like 47 or 182 the formula will return the default value, zero. To fix this, use the table function instead of the lookup function.

```
table("Shipping Rates",Weight,PackageWeight,«Rate Per Pound»,0,0)
```

The table function will return the closest lower match. This means that if the PackageWeight is 3, 17 or 42 the formula will return 2.50. If the PackageWeight is 110 or 246 the formula will return 2.25, etc. Here is a complete formula that calculates the shipping cost for any package.

```
PackageWeight*table("Shipping Rates",Weight,PackageWeight,«Rate Per Pound»,0,0)
```

The formula looks up the rate per pound and then multiplies that rate by the package weight.

Looking Up Multiple Fields From One Record

Sometimes you may need to lookup several fields in the same record. For example, when you lookup someone's address you may also want to lookup their city, state, zip code, phone number and recent purchasing history. In a procedure one way to do this is with multiple `lookup(` functions, like this.

```
Address=lookup("Customers",Company,Company,Address,"",0)
City=lookup("Customers",Company,Company,City,"",0)
State=lookup("Customers",Company,Company,State,"",0)
Zip=lookup("Customers",Company,Company,ZipCode,"",0)
Phone=lookup("Customers",Company,Company,"Phone#","",0)
```

When a procedure contains several `lookup(` in a row for the same thing like this Panorama doesn't actually search the database over and over again. Instead it notices that it is searching for the same item and simply grabs the data from the record it has already found.

To make multiple field lookups even faster you can use the speedcopy statement (see "SPEEDCOPY" on page 5784). (Remember, since this is a statement it can only be used within a procedure, see "Procedures" on page 1339). The speedcopy statement can transfer many fields at once, but only if the fields to be copied in the two databases match exactly. The fields to be copied must appear in exactly the same order in both databases, and the fields must have the same data types. With all these restrictions, you may be surprised to find out that the fields do not have to have the same names!

Here's how speedcopy works. Before you use speedcopy, you must perform an assignment with a lookup(function (or a variation of the lookup(function: lookuplast(, lookupselected(, etc., see "Lookup Variations" on page 1277). The lookup(function locates the record containing the information to be copied. Once the record has been located the speedcopy statement can be used to copy the additional data.

The speedcopy statement has three parameters.

```
speedcopy FirstAssignField,LastAssignField,FirstTargetField
```

The first two parameters are fields in the current database. The last parameter is a field in the target database. All of these field names should be surrounded by quotes (for example "Name", not Name). Speedcopy starts by converting these field names into field numbers. For example, if a field would be the third column in the data sheet, it is field #3.

Once speedcopy has converted the field names into numbers, it starts copying data. Suppose the FirstAssignField was field number 3, and the FirstTargetField was field number 8. Speedcopy will start by copying field #8 in the target database into field #3 in the current database. Then it will copy field #9 in the target database into field #4 in the current database. It will continue copying fields until it has copied something into the LastAssignField.

To show a specific example, suppose we have two databases, Organizer and Customers, with the fields listed below:

	Organizer	Customers
1	Name	Company
2	Title	Address
3	Company	City
4	Address	State
5	City	ZipCode
6	State	Phone#
7	Zip	Fax#
8	Phone	Cust#

The procedure below will quickly copy the Address, City, State, Zip and Phone fields from the Customers database to the Organizer database.

```
Address=lookup("Customers",Company,Company,Address,"",0)
if Address<>""
   speedcopy "City","Phone","City"
endif
```

Let's take a close look at how this procedure works. The first line attempts to lookup the Address from the Customer database. If this lookup fails, the procedure is finished. However, if the lookup succeeds the procedure continues with the speedcopy statement.

The first parameter of the speedcopy statement is City, which is field #5 in the current database (Organizer). The second parameter is Phone, which is field #8 in the current database. The final parameter is City, which is field #3 in the target database (Customers).

In this example speedcopy will copy 4 fields from Customers into Organizer, as shown by the blue arrows in this table. The green arrow represents the original lookup(.

	Organizer	Customers
1	Name	Company
2	Title	Address
3	Company	City
4	Address	State
5	City	ZipCode
6	State	Phone#
7	Zip	Fax#
8	Phone	Cust#

As speedcopy moves data from one database to another, it doesn't make any kind of checks on the data. If the fields aren't really in the same order, speedcopy will cheerfully copy them in the wrong order. Even worse, if you try to copy a numeric field into a text field or a text field into a numeric field, speedcopy will not object, but will speedily turn your current database into swiss cheese. The moral of the story is to use the speedcopy statement very carefully. Like any sharp instrument you want to make sure it is pointed in the right direction before you use it.

The GrabData Function

The grabdata(function (reference page 5327) grabs the contents of a field in the current record of any open database. You can grab data from the current database, or from another database. The function has two parameters — the name of the database to grab from and the name of the field within that database. For example here is the formula to look up the number of calories of the currently selected fruit.

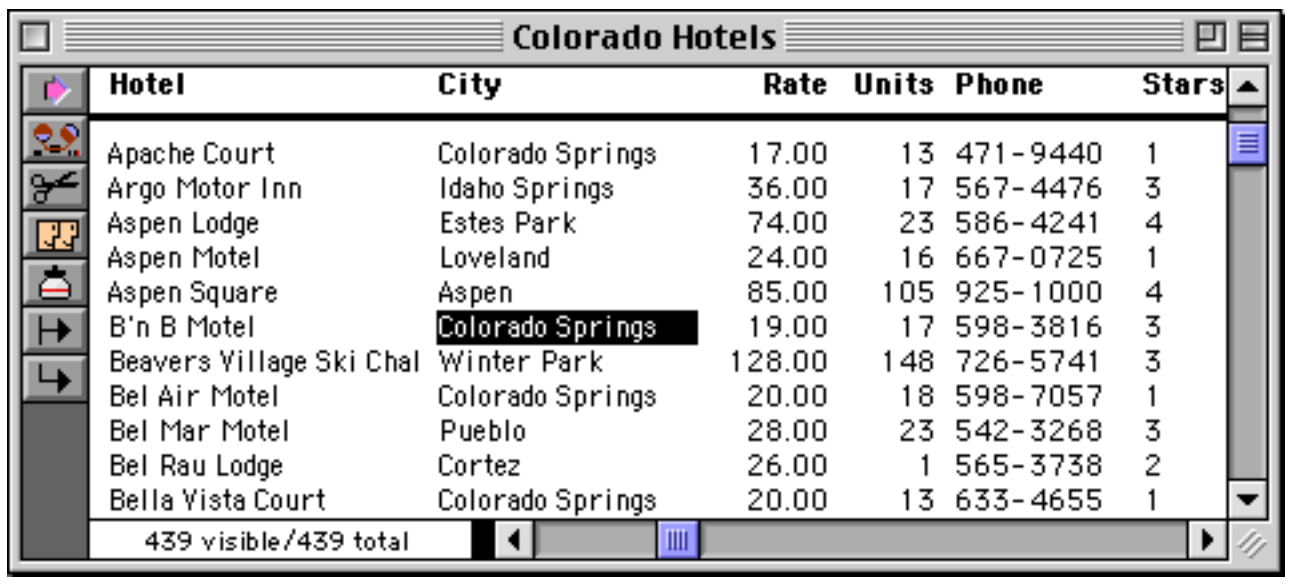
```
grabdata("Groceries",Calories)
```

The value returned by this function will change depending on what record is active in the Groceries database.

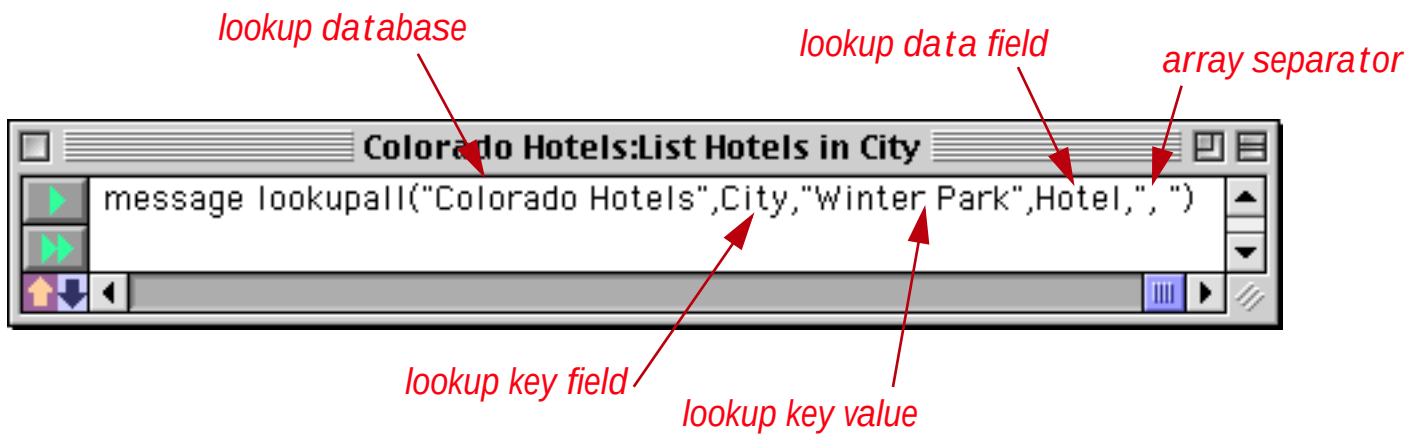
Looking Up Multiple Values at Once

The normal lookup functions return only a single value even if many records in the lookup database match the key value. The **lookupall(** function (reference page 5502) builds a text array containing one item for every record in the target database that matches. The function has five parameters. The first four parameters are the same as the other lookups: lookup database, lookup key field, lookup key value and data field (see "Linking

With Another Database" on page 1272). The fifth parameter is the separator character for the array that is constructed (see "Text Arrays" on page 1236). Unlike most text array functions, the `lookupall(` function allows the separator to have more than one character. For example, you could use ", " to place a comma and space between each item. To illustrate this function, consider this database of hotels.



Using the `lookupall(` function we can create a procedure that lists all of the hotels in Winter Park.



When you run this procedure it displays a list of all of the hotels in Winter Park, like this.



Of course you might want to list cities other than Winter Park. Here's a modified version of this formula that lists the hotels in whatever the current city is.

```
lookupall("Colorado Hotels",City,City,Hotel,", ")
```

Of course you could also display the list of hotels in an auto-wrap text object or Text Display SuperObject. The list would update automatically as you moved from record to record.

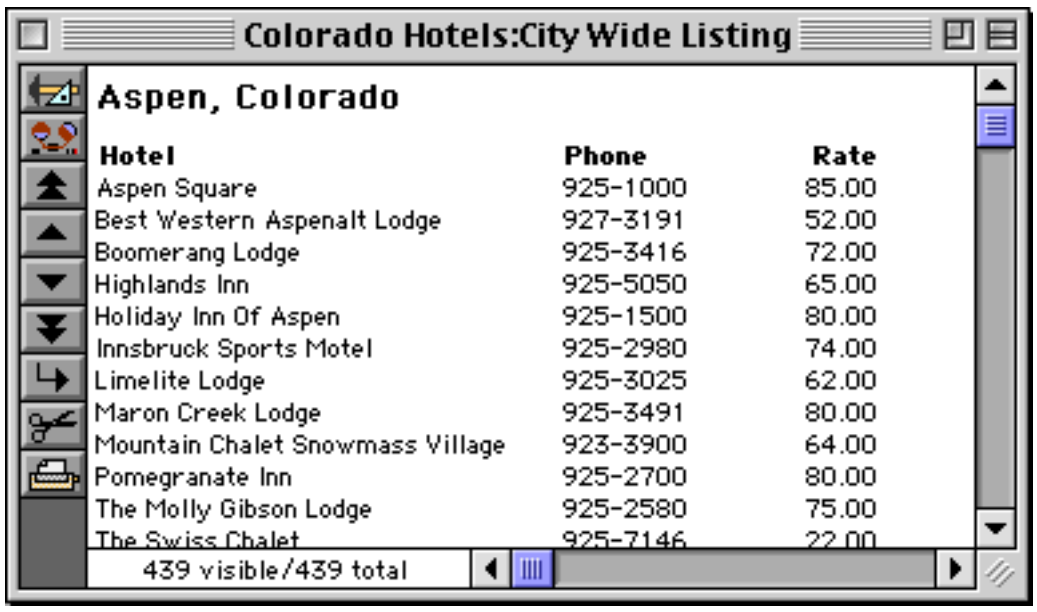
In our hotel example the lookup data field is a text field (Hotel). However, the lookupall(function will also work with numeric and date data fields. If the field is numeric or date it will be converted to text before it is added to the array. In this illustration three different lookupall(functions are used to display the hotel name, phone number, and rate.



When switched to Data Access mode you can see that all of the rates have been converted to text and appended together, one per line.



As you switch to a different record the form updates automatically.



In this case the lookup is from the current database, but any open database can be used.

Panorama also includes seven additional variations on the lookupall(function that lookup from two to eight fields from the target database, instead of just one. Each of these functions includes a parameter for controlling the separator between records and the separator between each fields.

Function	Reference Page	Description
lookupalldouble(thedb, keyfield,keyvalue,datafield1, datafield2,mainsep,subsep)		This function does a lookupall for two fields (double). The lookup is of all records in thedb where the keyfield matches the supplied keyvalue. For each record datafield1 and datafield2 are joined together with the subsep separator (which may be more than one character) and each record is joined together with the mainsep characters.
lookupalltriple(thedb, keyfield,keyvalue,datafield1, datafield2,datafield3, mainsep,subsep)		This function does a lookupall for three fields (triple). The lookup is of all records in thedb where the keyfield matches the supplied keyvalue. For each record datafield1 through datafield3 are joined together with the subsep separator (which may be more than one character) and each record is joined together with the mainsep characters.
lookupallquadruple(thedb, keyfield,keyvalue,datafield1, datafield2,datafield3,datafield4, mainsep,subsep)		This function does a lookupall for four fields (quadruple). The lookup is of all records in thedb where the keyfield matches the supplied keyvalue. For each record datafield1 through datafield4 are joined together with the subsep separator (which may be more than one character) and each record is joined together with the mainsep characters.
lookupallquintuplet(thedb, keyfield,keyvalue,datafield1, datafield2,datafield3,datafield4, datafield5,mainsep,subsep)		This function does a lookupall for five fields (quintuplet). The lookup is of all records in thedb where the keyfield matches the supplied keyvalue. For each record datafield1 through datafield5 are joined together with the subsep separator (which may be more than one character) and each record is joined together with the mainsep characters.
lookupallsextet(thedb,keyfield, keyvalue,datafield1, datafield2,datafield3,datafield4, datafield5,datafield6, mainsep,subsep)		This function does a lookupall for six fields (sextet). The lookup is of all records in thedb where the keyfield matches the supplied keyvalue. For each record datafield1 through datafield6 are joined together with the subsep separator (which may be more than one character) and each record is joined together with the mainsep characters.
lookupallseptuplet(thedb,keyfield, keyvalue,datafield1, datafield2,datafield3,datafield4, datafield5,datafield6,datafield7, mainsep,subsep)		This function does a lookupall for seven fields (septuplet). The lookup is of all records in thedb where the keyfield matches the supplied keyvalue. For each record datafield1 through datafield7 are joined together with the subsep separator (which may be more than one character) and each record is joined together with the mainsep characters.
lookupalloctet(thedb,keyfield, keyvalue,datafield1, datafield2,datafield3,datafield4, datafield5,datafield6,datafield7, datafield8,mainsep,subsep)		This function does a lookupall for eight fields (octet). The lookup is of all records in thedb where the keyfield matches the supplied keyvalue. For each record datafield1 through datafield8 are joined together with the subsep separator (which may be more than one character) and each record is joined together with the mainsep characters.

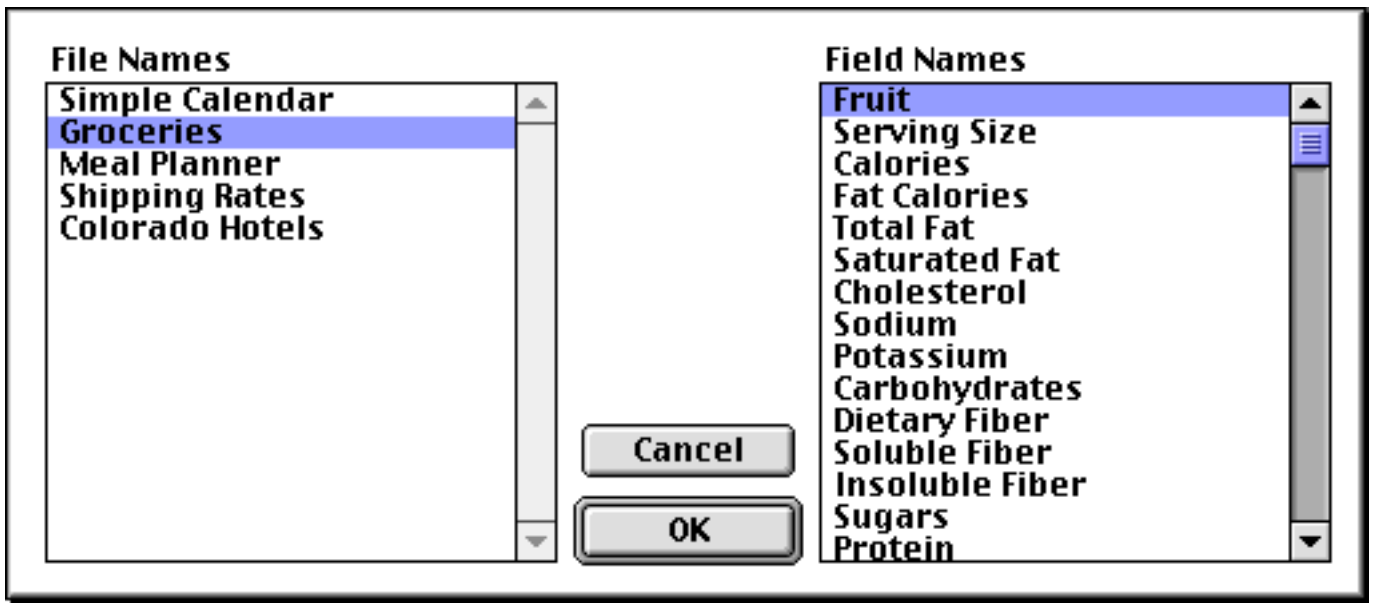
Linking Clairvoyance to the Lookup Key Field

Panorama's Clairvoyance feature anticipates what you are about to type by scanning the entries you have already made in the same database. When you are working with multiple files, you can configure Clairvoyance so that it scans the entries in another database instead (see "Clairvoyance® Across Multiple Files" on page 323). This is called linking clairvoyance to another field.

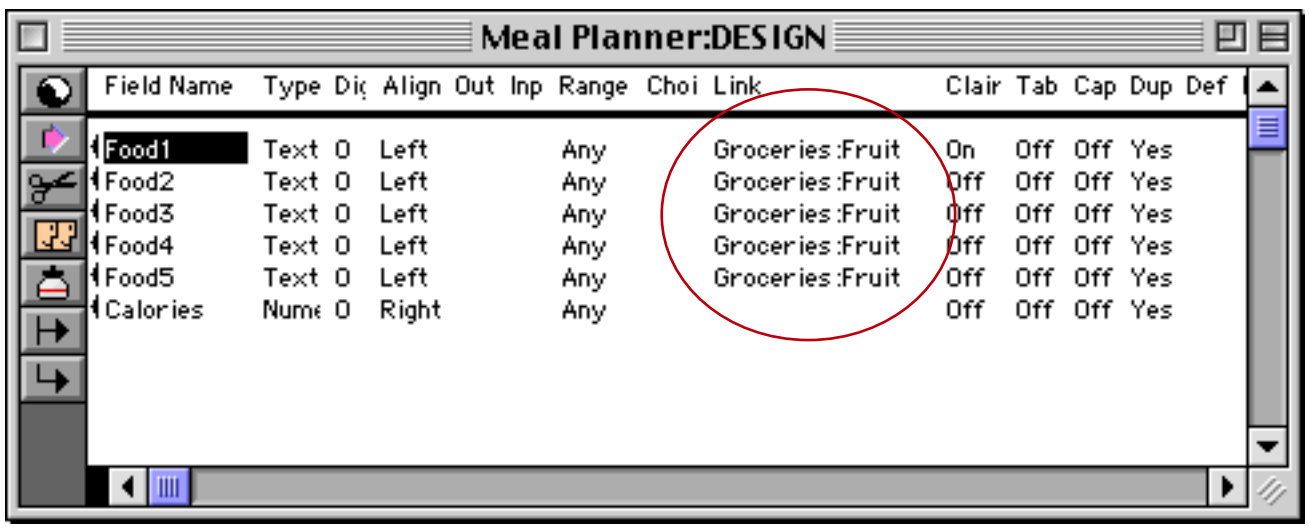
There are two reasons for linking clairvoyance to another field. Clairvoyance cannot anticipate values until they have been typed in at least once. If all the possible values have already been entered into another database, Clairvoyance can start working immediately by looking into the other database.

Another advantage is speed. If your price list contains 200 records and your invoice database contains 2000 records, Clairvoyance can scan the price list 10 times faster. As your database gets larger, this speed difference may become noticeable.

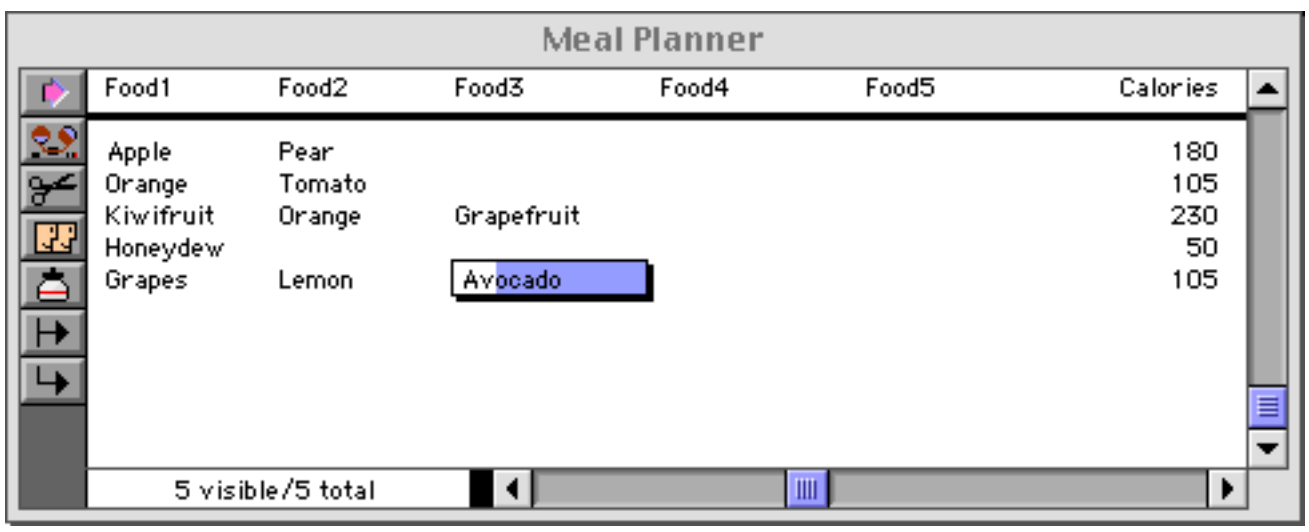
To set up a clairvoyance link to another field, use the design sheet. Click on the name of the field you want to set up. Then choose **Set Up Link** from the Special Menu. Choose the database you want to link to, and the field within that database.



Press **OK** to enter the link into the design sheet. Like all other design sheet options, the link does not actually take effect until you tell Panorama to create a new generation. In the design sheet shown below five fields have been linked to the Fruit field in the Groceries database.



When you are editing data within a field that has a clairvoyance link set up, clairvoyance checks the characters you type against the data in the second database. When it finds a possible match, it enters the rest of the value for you.



Looking Up Data in the Current File

You can use the lookuplast(function to look up the previous entry, with the same value, in the same database. For example, in a checkbook database you can automate repetitive payments by looking up the previous payment to the same company. By using the info("database") function to look up the database name you can make sure that the formula will continue to work even if the database is renamed.

```
lookuplast(info("database"),PayTo,PayTo,Amount,0,0)
```

Suppose that your last check to Pacific Mutual was $178.34. Using the formula above you could automatically enter this value the next time you write a check to this company.

Another application for looking up data in the current file is locating summary information further down in the database. To do this, set the lookup summary level to a non-zero value so that only summary records will be located.

The Assign Function

The `assign(` function is so different from every other Panorama V function that we gave it it's own section, all by itself. Instead of calculating a new value based on its parameters, the assign(function allows you to assign the result of a partial formula to a field or variable. This is called a "side effect" because this assignment is in addition to the normal operation of the formula that contains the assign(function (or multiple assign(functions).

The assign(function has two parameters: value and destination. The value parameter is simply any formula that calculates a value. The calculated value may be either numeric or a text value.

The destination parameter is the name of the field or variable to store the value into. If the destination is a field, the field must have the same data type as the calculated value (in other words, numeric values must be stored in numeric fields, text values in text fields. If the destination is a variable, the variable must already have been created with a local, global, fileglobal, permanent or windowglobal statement.

Let's look at some examples to illustrate the operation of this function. For example, consider the equation below, which might be part of a procedure. In this example, both A and B will be assigned the value X+Y.

```
B=assign(X+Y,"A")
```

When using the assign(function it's important to keep in mind that this function also produces a value - the value calculated by the value parameter. For example, in the example above B is also assigned the value X+Y. If you forget that the assign(function produces a value you may encounter unexpected error messages or even accidentally overwrite data.

The previous example illustrated the operation of the assign(function, but it wasn't really very useful. Here is a more interesting example. This example quickly scans a field and finds the largest value. The output of the arraybuild (the variable x), is simply ignored. The real result is the variable biggest, which is calculated as a side effect. At the end of the procedure this variable will contain the largest value in the current field.

```
local biggest,x
biggest=«»
arraybuild x,¶,"",assign(?(biggest>«»,biggest,«»),"biggest")
message biggest
```

The assign(function is very useful for formulas in forms, for example TextDisplay or auto-wrap text objects. In these objects the assign(function can be used for intermediate values that need to be used over and over. Of course when you use the assign(function this way you have to keep in mind the order of expression evaluation. You must make sure that the assign(function is evaluated before the destination is used. Here is an example where the customer name is assigned to a variable named tempCustomerName (which must be created with a global or fileglobal statement before this formula is displayed), then used in several locations in the text.

```
"Dear "+
assign(lookup("CustomerFile","CustomerNumber",custID,"Name","",0),"tempCustomerName")+","+
¶+tempCustomerName+", you are invited to particpate blah blah blah. "+tempCustomerName",
you won't want to miss out on this blah blah blah"
```

If you use the ?(function in combination with the assign(function be careful because the assign(function will always be evaluated. It doesn't matter whether the assign(function is in the true or false part of the ?(function.

When the assign(function is used with a field, the field display will not be updated automatically. You'll need to do that separately with `showfields`, `showcolums`, etc. (depending on what operation you have done).

Note: Assigning a value to a field in a `formulafill` (or `select`) in a multi-user Partner/Server database doesn't work properly. The field on the local client will change, but the server will not be updated. There's no warning or error message, so you should avoid using this function with fields in a Partner/Server database.

Zip Code Lookup

If you have purchased Panorama's optional zip code dictionary you can lookup the city, county and state of a zip code using the functions listed in the table below.

Function	Reference Page	Description
city(zip)	Page 5101	This function looks up a zip code and returns the name of the city for that zip code. The zip code may be either a number or text. For example the formula `city(92831)` will return the city name Fullerton, while the formula `city("92648")` will return Huntington Beach. If there is more than one possible name, the function returns the primary zip code name as defined by the US Post Office. If the zip code is not a valid zip code the function will return an empty string (""). If the zip code dictionary has not been installed the function will return --.
county(zip)	Page 5132	This function looks up a zip code and returns the name of the county for that zip code. The zip code may be either a number or text. For example the formula `city(92831)` will return the county name Orange, while the formula `county("95234")` will return San Joaquin. If the zip code is not a valid zip code the function will return an empty string (""). If the zip code dictionary has not been installed the function will return --.
state(zip)	Page 5793	This function looks up a zip code and returns the two letter abbreviation for the name of the state the zip code is in. The zip code may be either a number or text. For example the formula `state(92831)` will return the state abbreviation CA, while the formula `state("15234")` will return PA. If the zip code dictionary has not been installed the function will return --.

For more information about purchasing this optional package please visit our web site, **www.provue.com**.

Note: If you have purchased Panorama's optional spelling dictionary, you can lookup a list of words using the `wordlist` statement in a procedure. See "WORDLIST" on page 5907 for details.

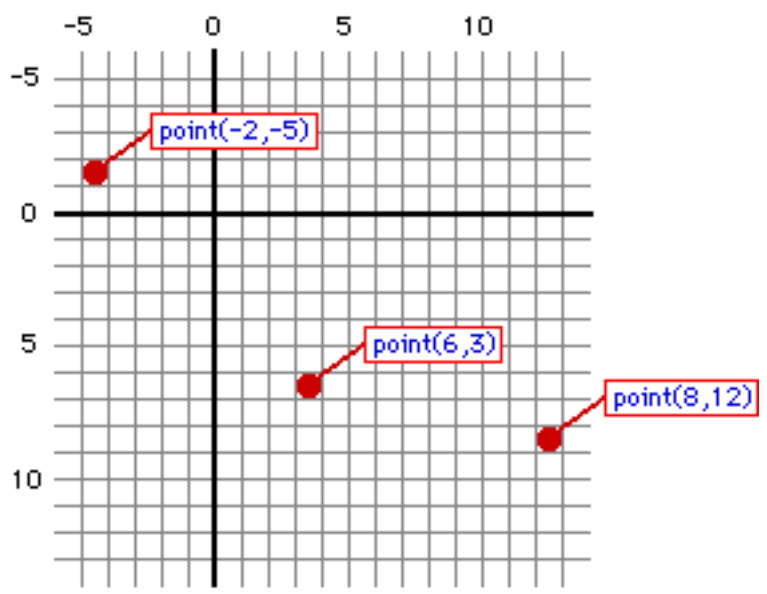
US Post Office Abbreviation Functions

These functions return text arrays that contain lists of official US Post Office abbreviations. These functions are designed to be used with the `arraylookup(` and `arrayreverselookup(` functions.

Function	Reference Page	Description
stateabbreviations()		This function returns a list of state abbreviations in this format: AL:ALABAMA;AK:ALASKA; ... This table is designed to be used with the arraylookup(and arraylookupreverse(functions.
statelookup(state)		This function looks up the name of a state from the state abbreviation (for example CALIFORNIA from CA). If the parameter does not match any state then the original value is returned.
uspssecondaryunits()		This function returns a list of USPS secondary suffix designation abbreviations in this format: APT:APARTMENT;RM:ROOM; ... This table is designed to be used with the arraylookup(and arraylookupreverse(functions.
uspsstreetsuffixes()		This function returns a list of USPS street suffix abbreviations in this format: ALY:ALLEY;AVE:AVENUE; ... This table is designed to be used with the arraylookup(and arraylookupreverse(functions.

Graphic Co-Ordinates

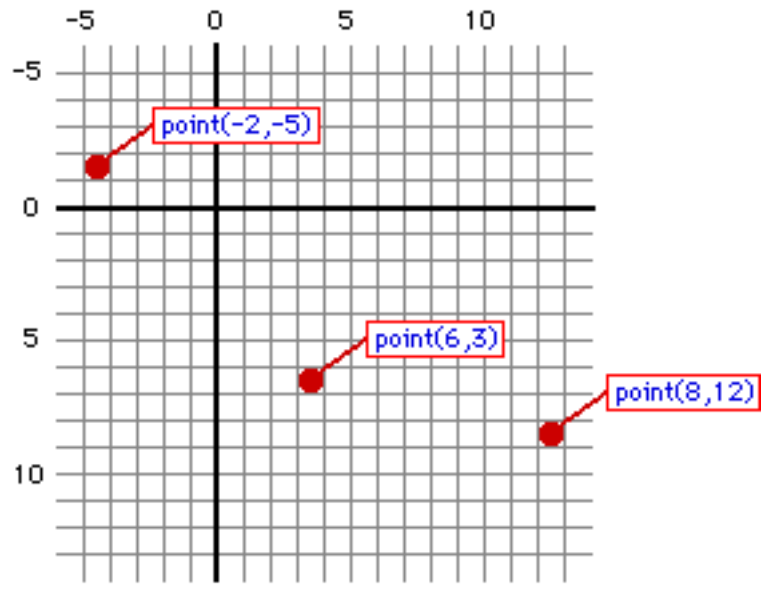
Many Panorama operations refer to locations on the screen, within a window, or within a form. The Macintosh uses an X-Y co-ordinate system to define locations. This X-Y system divides any area into an invisible grid of criss-crossing lines. There are 72 lines per inch. Each point where the lines intersect is identified by two numbers, the vertical and the horizontal position. The numbers increase as you move down and to the right. The illustration below shows a greatly expanded view of the X-Y coordinate system with several sample points.



See “GRAPHIC COORDINATES” on page 5330 for more information about co-ordinates.

Points

A point is a spot on the X-Y grid. A point has two elements, the vertical position and the horizontal position. Each position is a number (integer) between -32,767 and +32,767. A point combines these two numbers into a single number. Functions that work with points are listed in the table below.

Function	Reference Page	Description
point(v,h)	Page 5606	This function combines vertical and horizontal co-ordinates into a single number that describes the position of a point. V is the vertical position of the point. This must be a number between -32,768 and +32,767. (Unlike standard cartesian co-ordinates, positive is down and negative is up.) H is the horizontal position of the point. This must be a number between -32,768 and +32,767. (Like standard cartesian co-ordinates, positive is right and negative is left.) All dimensions are in pixels (1 pixel=1/72 inch). The function returns a number (an integer) that describes the location of the point. You can use this number in any function or statement that accepts a point as a parameter. The greatly magnified illustration below shows several sample points and the functions used to create them. Note that the actual point “hangs” down and to the right of the co-ordinate grid lines. 
h(point)	Page 5338	This function extracts the horizontal position from a point. The result is a number (an integer) that describes the horizontal position of the point. This number will be between -32,768 and +32,767. (Like standard cartesian co-ordinates, positive is right and negative is left.)

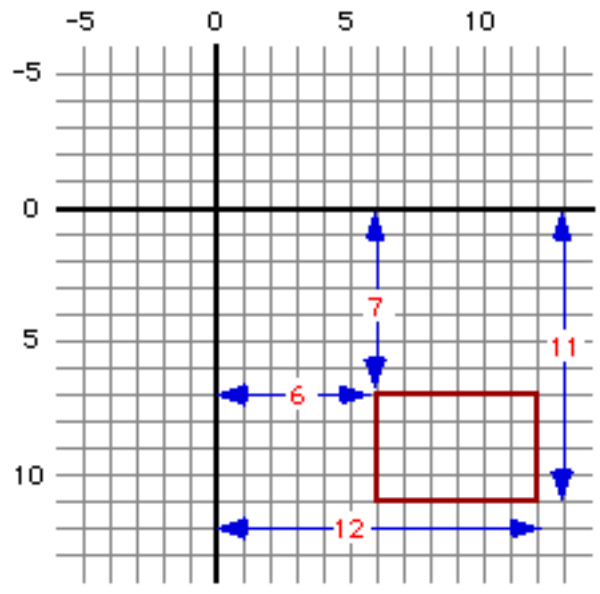
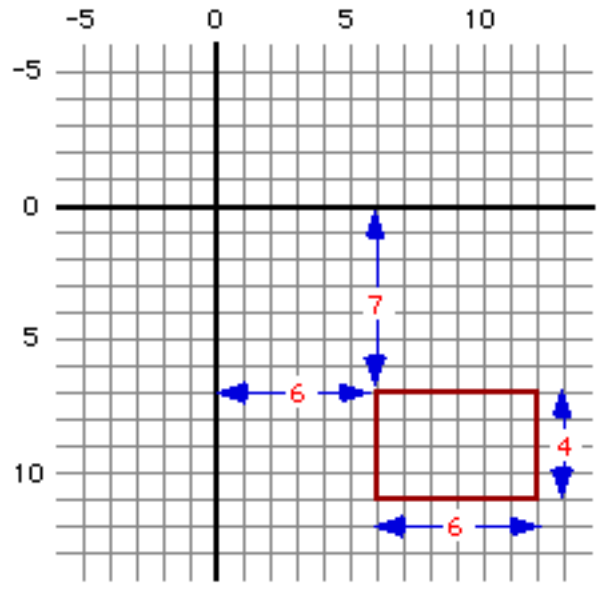
Function	Reference Page	Description
v(point)	Page 5885	This function extracts the vertical position from a point. The result is a number (an integer) that describes the vertical position of the point. This number will be between -32,768 and +32,767. (Unlike standard cartesian co-ordinates, positive is down and negative is up.)
info("click")	Page 5364	This function returns the location of the last mouse click in screen relative co-ordinates.
info("mouse")	Page 5398	This function returns the current location of the mouse in screen relative co-ordinates.
pointstr(point)		This function converts a point value into text in the format V,H (for example 34,56).

xytoxy(point,from,to)	Page 5910	This function converts a point or rectangle from one co-ordinate system to another. There are three possible co-ordinate systems: Screen Relative, Window Relative, and Form Relative (see "GRAPHIC COORDINATES" on page 5330 for illustrations of these three systems).
		
		The function has three parameters: point, from and to. Point is the point or rectangle that you want to convert to another co-ordinate system. From is the current co-ordinate system for the point, while to is the new co-ordinate system. The three options for these parameters are:
		
		`"Screen"  (may be abbreviated "S" or "s" )`
		`"Window"  (may be abbreviated "W" or "w" )`
		`"Form"    (may be abbreviated "F" or "f" )`
		
		This function returns a new, translated point that has been converted to a different co-ordinate system (for example screen relative to window relative). (Note: This function can work with rectangles as well as points — see the next section.)

Rectangles

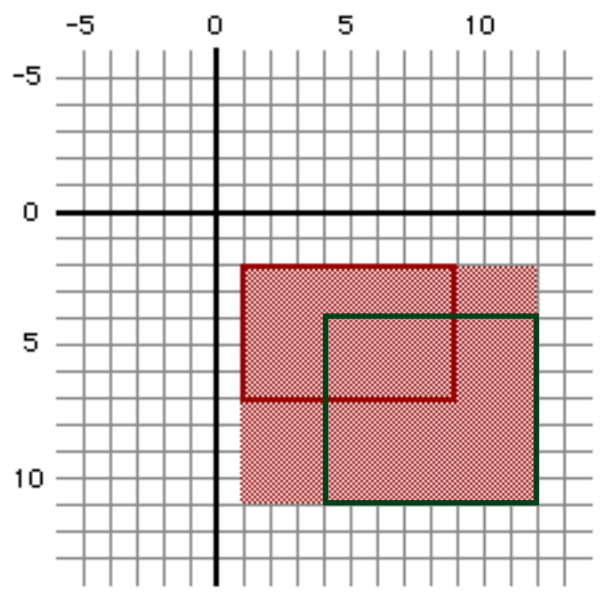
A rectangle is a simply box laid out on the X-Y grid. A rectangle is defined by four co-ordinates: top, left, bottom, right. The rectangle data type stores these four co-ordinates as raw binary data in an 8 byte text data item (see “Raw Binary Data” on page 1296).

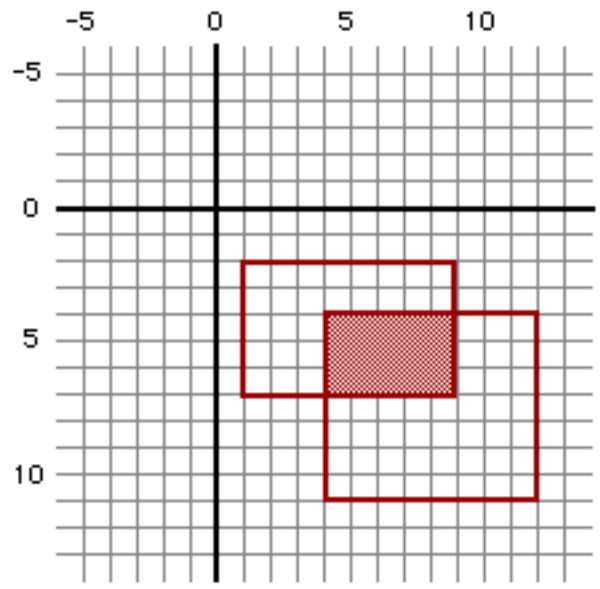
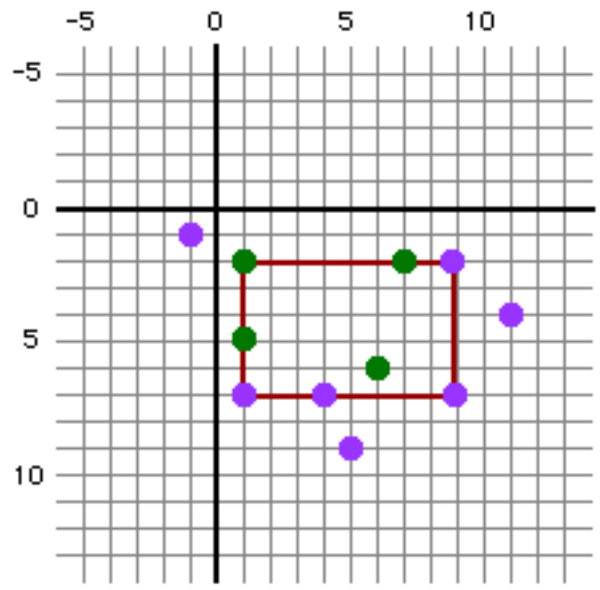
Don’t confuse the rectangles described in this section with rectangles that are graphic objects in a form. The rectangle data type merely describes an imaginary rectangle on the X-Y grid. This may correspond to a rectangle that is actually visible, but not necessarily.

Function	Reference Page	Description
rectangle(top,left,bottom,right)	Page 5631	This function defines a rectangle from four dimensions: top, left, bottom and right. These parameters specify the location of the edges of the rectangle. All measurements are in pixels (1 pixel = 1/72 inch). The formula below creates a rectangle that is 4 pixels high and 6 pixels wide. `rectangle(7,6,11,12)` Here is a magnified view of this rectangle. 
rectanglesize(top,left,height,width)	Page 5636	This function defines a rectangle from four dimensions: top, left, height and width. All measurements are in pixels (1 pixel = 1/72 inch). The height and width must be numbers between 0 and +32,767. The formula below creates a rectangle that is 4 pixels high and 6 pixels wide. `rectanglesize(7,6,4,6)` Here is a magnified view of this rectangle. 

Function	Reference Page	Description
rtop(rectangle)	Page 5685	This function extracts the position of the top edge of a rectangle. The function returns a number between -32,768 and 32768. This is the position of the top edge of the rectangle (in pixels).
rleft(rectangle)	Page 5680	This function extracts the position of the left edge of a rectangle. The function returns a number between -32,768 and 32768. This is the position of the left edge of the rectangle (in pixels).
rbottom(rectangle)	Page 5630	This function extracts the position of the bottom edge of a rectangle. The function returns a number between -32,768 and 32768. This is the position of the bottom edge of the rectangle (in pixels).
rright(rectangle)	Page 5683	This function extracts the position of the right edge of a rectangle. The function returns a number between -32,768 and 32768. This is the position of the right edge of the rectangle (in pixels).
rheight(rectangle)	Page 5678	This function computes the height of a rectangle. The function returns a number between 0 and 65535. This value is the height of the rectangle (in pixels).
rwidth(rectangle)	Page 5688	This function computes the width of a rectangle. The function returns a number between 0 and 65535. This value is the width of the rectangle (in pixels).

xytoxy(rect,from,to)	Page 5910	This function converts a point or rectangle from one co-ordinate system to another. There are three possible co-ordinate systems: Screen Relative, Window Relative, and Form Relative (see "GRAPHIC COORDINATES" on page 5330 for illustrations of these three systems).
		
		The function has three parameters: rect, from and to. Rect is the point or rectangle that you want to convert to another co-ordinate system. From is the current co-ordinate system for the point, while to is the new co-ordinate system. The three options for these parameters are:
		
		`"Screen"  (may be abbreviated "S" or "s" )`
		`"Window"  (may be abbreviated "W" or "w" )`
		`"Form"    (may be abbreviated "F" or "f" )`
		
		This function returns a new, translated rectangle that has been converted to a different co-ordinate system (for example screen relative to window relative). (Note: This function can work with points as well as rectangles — see the previous section.)

unionrectangle(rect1,rect2)	Page 5868	This function defines a rectangle by combining two rectangles. The new rectangle is large enough to exactly cover both of the input rectangles. The illustration below shows how this function combines two rectangles, defining a third rectangle that covers the original two rectangles. 

Function	Reference Page	Description
intersectionrectangle(rect1,rect2)	Page 5459	This function defines a rectangle by combining two rectangles. The new rectangle is the area where the two rectangles overlap (if any). If the two rectangles do not touch each other the function will return an empty rectangle (same as `rectangle(0,0,0,0)`). The illustration below shows how this function combines two rectangles, creating a third rectangle where the original two rectangles overlap: 
inrectangle(point,rectangle)	Page 5453	This function checks to see if a point is inside a rectangle. There are two parameters: point and rectangle. The result is true or false. If the point is inside the rectangle, the function returns true (-1). If the point is not inside the rectangle, the function returns false (0). You can use this function with the `if` statement (see “IF Statements” on page 1380) and the `?(` function (see “The ? Function” on page 1271). The illustration below shows a rectangle and several points. Green points are inside the rectangle, purple points are not. Notice that points on the top and left edge of the rectangle are considered inside. Points on the bottom and right edge are considered outside. 

Function	Reference Page	Description

rectanglecenter(largrect,smallrect)	Page 5635	This function adjusts a small rectangle so that it is centered inside of a larger rectangle. Largerect is a large rectangle. How large is large? Well, it should at least be larger than the smallrect rectangle. Smallrect is a small rectangle. How small is small? Well, it should at least be small enough to fit inside the largerect rectangle, although the function will do its best to center it even if it does not fit.
		The formula below creates a 1 inch square rectangle that is centered within the current screen dimensions.
		`rectanglecenter(`
		`info("screenrectangle"),`
		`rectanglesize(0,0,72,72))`

rectangleadjust(
rect,
Δtop,
Δleft,
Δbottom,
Δright)
	Page 5633	This function adjusts all four edges of a rectangle independently. There are five parameters: rect,Δtop, Δleft, Δbottom and Δright. Rect is the original rectangle. The other four parameters are the distance each edge should be moved, which should be a number between -32,768 and +32,767.
		The formula below creates a rectangle that is inset 20 pixels from all four edges of the screen.
		`rectangleadjust(`
		`info("screenrectangle"),20,20,-20,-20)`
		The formula below creates a rectangle that is the same size as the button rectangle but shifted 1 inch (72 pixels) to the right.
		`rectangleadjust(`
		`info("buttonrectangle") ,0,72,0,72)`

adjustxy(rectangle, boundary, deltav, deltah)	Page 5020	This function adjusts the four corners of a rectangle. However, only corners that are inside a boundary are adjusted. Corners outside the boundary are left alone. There are four parameters: rectangle, boundary, deltav and deltah. Rectangle is the rectangle that is being adjusted. Boundary is a rectangle describing the area to be adjusted. Only points inside this rectangle will be adjusted. Deltav is the vertical distance each corner inside the boundary should be adjusted. Deltah is the horizontal distance each corner inside the boundary should be adjusted.
rectanglesizestr(rect)		This function converts a rectangle into text in the format TOP,LEFT,HEIGHT,WIDTH (for example 100,20,80,30).
rectanglestr(rect)		Converts a rectangle into text in the format TOP,LEFT,BOTTOM,RIGHT (for example 100,20,180,50).
info("buttonrectangle")	Page 5361	This function returns a rectangle defining the edges of the button that was clicked on (needless to say, this function should be used in a procedure that is triggered by a button). The rectangle is in screen relative coordinates (use the xytoxy(function to convert to window or form relative co-ordinates).
info("cursorrectangle")	Page 5365	This function returns a rectangle defining the edges of the current data cell (if any). The rectangle is in screen relative coordinates (use the xytoxy(function to convert to window or form relative co-ordinates).
info("screenrectangle")	Page 5414	This function returns a rectangle defining the edges of the main screen (the screen that contains the menu bar). The rectangle is in screen relative coordinates.
info("windowrectangle")	Page 5446	This function returns a rectangle defining the edges of the current window. The rectangle is in screen relative coordinates.

Colors

We think of colors as the spectrum of the rainbow, but the computer builds up all colors from just three: red, green, and blue. By varying the relative intensity of these three colors the computer can generate all the colors of the rainbow. A Panorama color data item combines red, green and blue intensity values into a single data item. Color intensity is measured on a scale from 0 (completely dark) to 65,535 (full brightness). Values in between denote intermediate intensity. The table below shows a small sample of the colors that are possible.

RED	GREEN	BLUE	COLOR	SAMPLE
0	0	0	Black	
65535	65535	65535	White	
15000	15000	15000	Dark Gray	
45000	45000	45000	Light Gray	
65535	0	0	Pure Red	
0	65535	0	Pure Green	
0	0	65535	Pure Blue	
65535	0	65535	Purple	
65535	65535	0	Yellow	
0	65535	65535	Cyan	
3441	4276	32336	Dark Blue	
39235	30211	30211	Brown	
24367	23356	31931	Light Green	
65535	23356	2936	Orange	

Another way to specify a color is the **HSB**, or **Hue**, **Saturation**, **Brightness** system. Like the RGB system, the HSB system uses three numbers from 0 to 65,535 to describe a color. However, the three components have different meanings in this system.

The **Hue** component specifies where this color falls in the spectrum. If you are familiar with the standard Apple color picker, the Hue would specify the angle of the color from the center of the wheel.

The **Saturation** component refers to how intense this color is. Is it a very intense deep color, or is it a soft pastel color, or somewhere in between? Again using the standard Apple color picker, the Saturation would specify the distance of the color from the center of the wheel.

The **Brightness** component refers to how light or dark the color is. Is the color very bright, or is it almost black? This sounds similar to Saturation, but it isn't. Imagine a blue ball under a white light. As the light gets dimmer, the Hue and Saturation of the color don't change, but the Brightness does.

A color in a field or variable is just a piece of data that describes a color…you can't actually see the color. However, some SuperObjects allow you to control their color using a color data item, and you can look at or modify the color of any graphic object in a form using the functions and statements listed below.

Function	Reference Page	Description
rgb(red,blue,green)	Page 5677	This function creates a color by combining red, green, and blue primary colors. Red is the intensity of the red component of this color. This must be a number from 0 (black) to 65535 (full intensity). Green is the intensity of the green component of this color. This must be a number from 0 (black) to 65535 (full intensity). Blue is the intensity of the blue component of this color. This must be a number from 0 (black) to 65535 (full intensity).
hsb(hue,saturation,brightness)	Page 5347	This function creates a color by combining hue, saturation, and brightness components. Hue specifies where this color falls in the spectrum. This must be a number from 0 to 65535. Saturation specifies how intense this color is. Is it a very intense deep color, or is it a soft pastel color, or somewhere in between? This must be a number from 0 (black) to 65535 (full intensity). Brightness specifies how light or dark the color is. Is the color very bright, or is it almost black? This sounds similar to Saturation, but it isn't. Imagine a blue ball under a white light. As the light gets dimmer, the Hue and Saturation of the color don't change, but the Brightness does. This must be a number from 0 (black) to 65535 (full intensity).
red(color)	Page 5638	This function extracts the red intensity from a color. This intensity is a number between 0 (black) and 65535 (full intensity). The example below calculates the red intensity of the color (in percent, from 0 to 100%). `red(HighlightColor)*100/65535`
green(color)	Page 5333	This function extracts the green intensity from a color. This intensity is a number between 0 (black) and 65535 (full intensity). The example below calculates the green intensity of the color (in percent, from 0 to 100%). `green(HighlightColor)*100/65535`
blue(color)	Page 5076	This function extracts the blue intensity from a color. This intensity is a number between 0 (black) and 65535 (full intensity). The example below calculates the blue intensity of the color (in percent, from 0 to 100%). `blue(HighlightColor)*100/65535`
hue(color)	Page 5347	This function extracts the hue value from a color. Hue specifies where this color falls in the spectrum. This is a number from 0 to 65535.
saturation(color)	Page 5691	This function extracts the saturation intensity from a color. Saturation specifies how intense this color is. Is it a very intense deep color, or is it a soft pastel color, or somewhere in between? This is a number from 0 (black) to 65535 (full saturation).
brightness(color)	Page 5077	This function extracts the brightness of a color. Brightness specifies how light or dark the color is. Is the color very bright, or is it almost black? This sounds similar to Saturation, but it isn't. Imagine a blue ball under a white light. As the light gets dimmer, the Hue and Saturation of the color don't change, but the Brightness does. The brightness value is a number from 0 (black) to 65535 (full brightness).

Function	Reference Page	Description
info("formcolor")	Page 5378	This function returns the background color of the current form. If the current window does not contain a form, the function will return empty text ("").
black()		This function returns black (it is equivalent to rgb(0,0,0)).
gray(saturation)		This function returns a gray color. The saturation value is between 0 (white) and 100 (black).
htmlrgb(string)		This function converts text formatted as an HTML color (for example FFFFFF for white or FF0000 for red) into a Panorama color value.
htmlrgbstr(color)		This function converts a Panorama color value into text formatted as an HTML color (for example BB4DFD).
white()		This function returns white (it is equivalent to rgb(65535,65535,65535)).

If you are writing a procedure there are also two procedure statements that deal with color. The `colorwheel` statement (see "COLORWHEEL" on page 5119) opens a dialog for picking a color. The `formcolor` statement (see "FORMCOLOR" on page 5262) changes the background color of the current form.

Raw Binary Data

At the core, computers work with 1's and 0's, on and off, true and false. This is called binary data, because there are only two options. Fortunately, Panorama users don't ever have to deal with raw binary data. The programmers take the 1's and 0's and give them structure to create text, numbers, pictures, and other complex elements.

It's not much fun, and it's rarely necessary, but Panorama does allow a programmer to work with raw, unstructured, binary data: 1's and 0's. When you work with raw binary data it will always be in a text field or variable. Panorama normally interprets text as a series of characters. The functions listed in this section, however, do not interpret the binary data as characters. Instead, they allow you to directly access and manipulate the 1's and 0's. Panorama uses the text data type to hold raw binary data because text may be of any length and places no restrictions on the binary information that is placed in it. (However, the text may look very strange if you display it in the data sheet or on a form.)

Note: Panorama never requires you to use raw binary data. However, raw binary data may be useful for working with data from the operating system or with data generated by another program (or to generate data for another program). If you don't already know you need to use raw binary data, you can probably skip this section (unless you are curious). If you are a C, Java or Pascal programmer, these functions let you work with virtually any data structure you can cook up.

The functions that work with binary data are listed in the table below.

Function	Reference Page	Description
byte(number)	Page 5079	This function converts a number into a single byte of binary data. (Note: the byte(function is basically the same as the chr(function.) The number parameter must be between 0 and 255. This function converts the number into a single byte of binary data (8 bits).
word(number)	Page 5906	This function converts a number into a single word (2 bytes) of binary data. Number is the value that you want to convert into a binary number. This value must be between 0 and 65,535. This function converts the number into a two bytes of binary data (16 bits).
longword(number)	Page 5499	This function converts a number into a single longword (4 bytes) of binary data. Number is the value that you want to convert into a binary number. This value must be an integer. This function converts the number into a four bytes of binary data (32 bits).

Function	Reference Page	Description
binaryvalue(data)	Page 5075	This function converts binary data (a byte, word, or longword) into a number. Data is the binary value that you want to convert into a number. This value must be a byte, a word (2 bytes) or a longword (4 bytes). The result is an integer.

textstuff(text,new,position)	Page 5853	This function replaces one or more characters in the middle of a piece of text. Text is the original text data item that contains one or more characters you want to replace. New is the new text that you want to use to replace characters in the original text data item. Position is the location within the original text where you want to replace text. The position is a number starting with zero. The function returns a copy of the original text item with one or more characters replaced.
		This example replaces two characters in a 24 character text item.
		`textstuff("Temperature:  87 degrees"," 92",13)`
		The operation of this formula is shown in the table below.
		`Original Text: Temperature:  87 degrees`
		`New Text: 92`
		`Position: 13`
		`----------`
		`Result (TEMP): Temperature:  92  degrees`
		If the new text is positioned beyond the end of the original text, the characters in between are undefined.
		`textstuff("Temp:"," 92",13)`
		The operation of this formula is shown in the table below.
		`Original Text: Temp:`
		`New Text:  92`
		`Position: 13`
		`----------x--------`
		`Result (TEMP): Temp: \#ø•º¢∏ƒ92`
		The characters in between (in orange) may be anything. (Of course you could use another textstuff(function to fill them in, or you could add characters to the original text before using textstuff(in the first place.)

string255(text,space)	Page 5800	This function converts text into a Pascal string. A Pascal String is a special text format that is sometimes used by the Macintosh ROM's (also called a String255 or Str255 because the text is limited to a maximum length of 255 characters). (Pascal is the name of a computer language, which in turn is named after a famous mathematician.) The function has two parameters: text and space. Text is the text that you want to convert into a Pascal string. This text should be less than 255 characters long. Space is a number defining the amount of space taken up by the Pascal string. If space is zero, the string may be up to 255 characters, and is not padded. If space is from 1 to 255, the string255(function makes sure that the string takes up exactly this amount of space. If the string is too long, it will be cut off. If the string is too short, it will be padded with nulls (bytes containing zeroes). The function returns a text data item containing a Pascal string.

Function	Reference Page	Description
text255(data)	Page 5851	This function converts binary data containing a Pascal String into regular text. A Pascal String is a special text format that is sometimes used by the Macintosh ROM's (also called a String255 or Str255 because the text is limited to a maximum length of 255 characters). (Pascal is the name of a computer language, which in turn is named after a famous mathematician.) The function returns the text equivalent of the Pascal String passed to it.
radix(radix,text)	Page 5626	This function converts a text item containing a hex, octal, or binary number into a standard Panorama number (decimal). See "NON DECIMAL NUMBERS" on page 5543 for background information on hex, octal and binary numbers. Radix is the base for the numbering system you are converting from. Legal radix values are 2, 4, 8, 16 or 32. Or you can specify the radix as "binary" (same as 2), "octal" (same as 8) or "hex" (short for hexadecimal, same as 16). Text is a text item that contains the non-decimal number you want to convert. This function normally returns an integer that contains the decimal (base 10) number corresponding to the hex, octal, or binary number input to the function. If the radix is hex and there are more than 8 digits in the input text, or if the radix is binary and there are more than 32 digits, this function will return a raw binary value instead of a number. This binary value may be of unlimited length. Like all binary values, it cannot be calculated with, but should be handled as a text item.

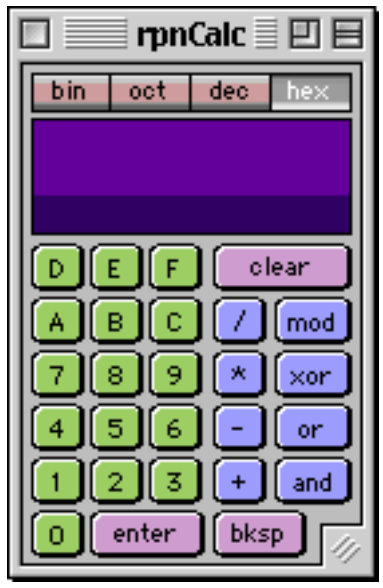
radixstr(radix,number)	Page 5628	This function converts a number into a text item containing the equivalent hex, octal, or binary number. See "NON DECIMAL NUMBERS" on page 5543 for background information on hex, octal and binary numbers. Radix is the base for the numbering system you are converting from. Legal radix values are 2, 4, 8, 16 or 32. Or you can specify the radix as "binary" (same as 2), "octal" (same as 8) or "hex" (short for hexadecimal, same as 16). Number is the number you want to convert to hex, octal, or binary. If the radix is 2, 16, "binary", or "hex" the number can be a raw binary data (text) value. This function returns a text item that contains the hex, octal, or binary number equivalent to the number (or binary data) passed to the function. The first example converts the decimal value 256 to hexadecimal.
		
		`radixstr(16,256)`
		
		This function will calculate that 256_{10} is 100 hex.
		
		Here is another example:
		
		`radixstr("binary",5)`
		
		This will calculate that 5_{10} is 00000000000000000000000000000101 binary.

bit(number)		This function converts a bit number (1 to 32) into a number (1, 2, 4, 8, 16, etc.)
bytearray(data,index)		This function extracts a value from an array of bytes. This is not a Panorama style delimited array but a C style array of 8 bit values. The result is an integer.
chararray(data,index)		This function extracts a characters from an array of characters. This is not a Panorama style delimited array but a C style array of ASCII characters. The result is an single characters.
chunkarray(data,index, chunklength)		This function extracts a binary chunk from an array of chunks. This is not a Panorama style delimited array but a C style array of binary chunks. The result is a binary value (text).

Function	Reference Page	Description
getbit(number,bitnumber)		This function returns a true or false value by testing a bit. The bit number may be from 1 to 32. If the bit is set, the value will be true, if it is not set, the value will be false.
hex(string)		Converts text with hex characters into a number. For example the value of hex("0C2") is 194.
longwordarray(data,index)		This function extracts a value from an array of longwords. This is not a Panorama style delimited array but a C style array of 32 bit values. The result is an integer.
onescomplement(number)		This function returns the one's complement of a 32 bit number (all bits are reversed)
setbit(number,bitnumber,truefalse)		This function sets one bit within a number, without disturbing any of the other bits. The bit number may be from 1 to 32. The bit will be set based on the true / false parameter - set if true, cleared if false.
wordarray(data,index)		This function extracts a value from an array of words. This is not a Panorama style delimited array but a C style array of 16 bit values. The result is an integer.

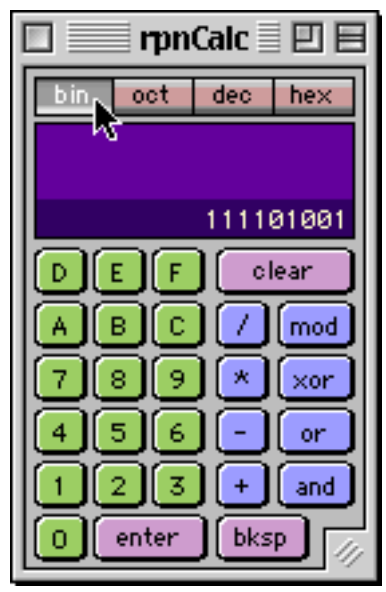
The RPN Programmer's Calculator

Panorama includes a built in calculator that can perform calculations in decimal (base 10), hexadecimal (base 16), octal (base 8) and binary (base 2). To open this calculator choose the **RPN Programmer's Calculator** from the Wizard menu.



Converting Between Different Bases

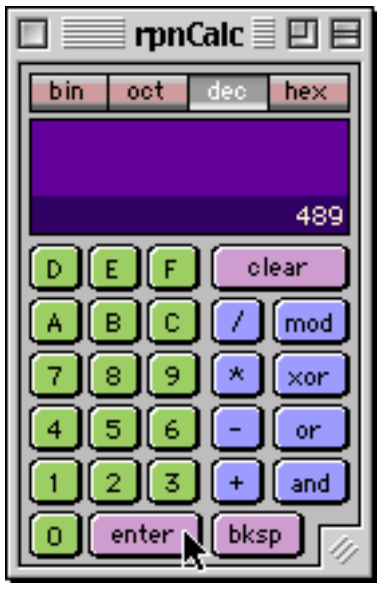
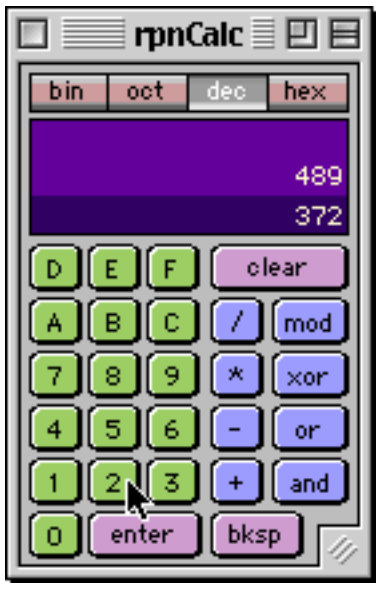
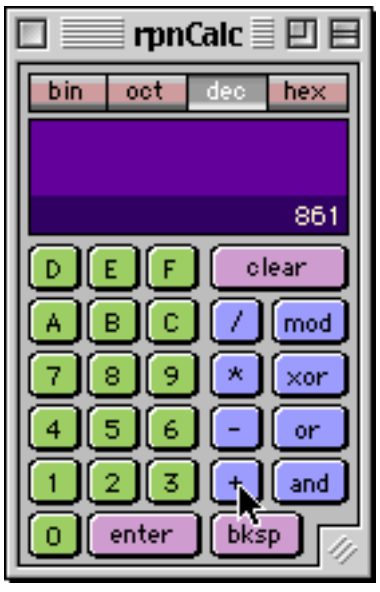
The calculator can be used to convert numbers from decimal to hex, hex to decimal, or in fact from any of the four supported bases into any other. For example, suppose you want to convert the number 489 (decimal) into hex. Start by pressing the **dec** (for decimal) button, then enter **489**, then press the **hex** button. The result is 1E9 (remember, hexadecimal numbers may include the digits A-F in addition to 0-9). If you wanted to see the same number in binary you would press the **bin** button You can change the number base at any time.

Press the **dec** button	Enter **489**	Press the **hex** button	Press the **bin** button
			

Calculations with Reverse Polish Notation

RPN is a variation on a parentheses-free mathematical logic known as "Polish Notation," developed by the Polish logician Jan Lukasiewicz (1878-1956). Over the years the most popular calculators in business and science have been HP calculators. These feature Reverse Polish Notation, especially the HP-12C business calculators and the HP-41 series. Texas Instruments was the other big player in calculators, and their machines used algebraic notation. You were either an RPN supporter, walking around with the motto ENTER > = (RPN is greater than Algebraic) emblazoned on T-shirts, or you were a TI algebraic "heretic." Here at ProVUE we were weaned on the original HP-35 scientific calculator in the late 70's, so our programmer's calculator uses Reverse Polish Notation.

With RPN, numbers are entered or "stacked" in the register. RPN is implemented by means of a numeric stack, the enter key, and the convention of "postfix operators." Postfix operators simply means that the user specifies the operation to be performed after the entry of numbers, instead of in the middle. For example, suppose you wanted to add 489+372. Here's how you do this with the RPN calculator.

Enter **489**	Press the **enter** button	Enter **372**	Press the **+** button
			

According to HP, "RPN is an effective way to deal with arithmetic expressions in programming. RPN makes it possible to perform compound calculations with a minimum of special symbols and no punctuation. Numbers are stored in the register. With the elimination of parentheses and the consistency of the entry method, the calculator accepts more of the problem-solving burden, reducing the user's time and effort."

The RPN system eliminates the need for parentheses. Here's how you would calculate (6+3)*(5+2).

Enter **6** and press **enter**	Enter **3**	Press the **+** button	Enter **5** and press **enter**
			

Enter **2**	Press the **+** button	Press the ***** button	
			

Performing this calculation in RPN requires only 9 keystrokes, vs. 11 using conventional algebraic notation. Ok, so it's a religious issue. So sue me (and all my RPN loving cronies)!

The purple area shows the numeric stack, where intermediate results are stored. You can expand the calculator window to see more entries in this numeric stack.

Boolean Operators

The four buttons on the right (**mod**, **xor**, **or** and **and**) are the boolean functions. These combine two numbers on a bit by bit basis. The table below explains each of these functions. The example numbers (A, B and Result) are in binary, but these operators will work in any number base..

Operator	Description	A	B	Result
and	For each bit in A and B, the result will be 1 if both A and B are 1. It will be 0 if either A or B are zero.	10110	00101	00100
or	For each bit in A and B, the result will be 1 if either A or B is 1. It will be 0 if both A and B are zero.	10110	00101	10111
xor	For each bit in A and B, the result will be 1 if A and B are different. It will be 0 if both A and B are the same.	10110	00101	10011
mod	This operator computes the remainder when dividing A by B	10110	00101	10

This example shows how to calculate the remainder when dividing 63 by 8.

Enter **63** and press **enter**	Enter **8**	Press the **mod** button
		

The remainder is 7.

Disk Files and Folders

Panorama formulas can directly access files and directories on the disk. The functions below read information from the disk. For a more detailed discussion of file i/o, including writing to files, see “Directly Reading and Writing Disk Files” on page 1549.

Function	Reference Page	Description

folder(path)	Page 5259	This function creates a binary data item that unambiguously describes the location of a folder on the hard disk. This **pathid** can be used in other functions and statements. Path is a complete description of the path to this folder. On the Macintosh a path looks like this:
		`My Disk:System Folder:Extensions:`
		On a Windows computer a path looks like this:
		`C:\Windows\Temporary`
		This function returns a 6 byte binary data item that unambiguously describes the location of the folder. However, if the folder does not exist the function returns an empty binary data item ("").

folderpath(pathid)	Page 5260	This function takes a six byte **pathid** (see the folder(function above) and converts it to a textual description of the path to that folder. A **pathid** is a binary data item that unambiguously describes the location of a folder on the hard disk. **Pathid’s** are created by the folder(and dbinfo(functions, and the openfiledialog and savefiledialog statements.
		The function returns complete a description of the path to this folder. On the Macintosh a path looks like this:
		`My Disk:System Folder:Extensions:`
		On a Windows computer a path looks like this:
		`C:\Windows\Temporary`

Function	Reference Page	Description

fileload(folder,file)	Page 5231	This function reads the entire contents of any file on disk. It is especially useful for reading text files. Folder is a **pathid** that unambiguously describes the location of the folder. A **pathid** is a binary data item that unambiguously describes the location of a folder on the hard disk. **pathid's** are created by the folder(and dbinfo(functions, and the openfiledialog and savefiledialog statements. If this parameter is empty text ("") the folder containing the current database is assumed. File is the name of the file that is to be read.
		
		This function returns the entire contents of the file as an item of text. (Technical note: Macintosh files may be split up into two components, called the "data fork" and the "resource fork." The fileload(function reads the data fork, but not the resource fork. You can read the resource fork by using a special statement, see "Reading and Writing Resource Forks" on page 1555.)
		
		This example reads a file named Report.txt. This file must be in the same folder as the current database.
		
		`fileload("","Report.txt")`
		
		The example below reads the contents of the Macintosh notebook file.
		
		`fileload(info("systemfolder"),"Note Pad File")`
		
		On Macintosh systems you must be careful when reading large files — the file must fit in Scratch Memory (see "Changing Scratch Memory Size (Macintosh OS 9)" on page 192). For very large files you may need to read in only a portion at a time using the fileloadpartial(function (see below).

fileloadpartial(folder,file,start,len)	Page 5233	This function reads a portion of the contents of any file on disk. It is especially useful for reading text files. Folder is a **pathid** that unambiguously describes the location of the folder. A **pathid** is a binary data item that unambiguously describes the location of a folder on the hard disk. **pathid's** are created by the folder(and dbinfo(functions, and the openfiledialog and savefiledialog statements. If this parameter is empty text ("") the folder containing the current database is assumed. File is the name of the file that is to be read. Start is the first byte (character) of the file that should be read. This function assumes bytes in the file are numbered starting from 0. Len is the number of bytes that should be read. The function returns a portion of the contents of the file as an item of text. (Technical note: Macintosh files may be split up into two components, called the "data fork" and the "resource fork." The fileloadpartial(function reads the data fork, but not the resource fork. You can read the resource fork by using a special statement, see "Reading and Writing Resource Forks" on page 1555.)
filesize(folder,file)	Page 5238	This function determines the size of any file on disk. Folder is a **pathid** that unambiguously describes the location of the folder. A **pathid** is a binary data item that unambiguously describes the location of a folder on the hard disk. **pathid's** are created by the folder(and dbinfo(functions, and the openfiledialog and savefiledialog statements. If this parameter is empty text ("") the folder containing the current database is assumed. File is the name of the file that is to be measured. This function returns a number—the size of the entire contents of the file. (Technical note: Macintosh files may be split up into two components, called the "data fork" and the "resource fork." The filesize(function reads the size of the data fork, but not the resource fork.) The example below determines the size of the Macintosh notebook file. `filesize(info("systemfolder"),"Note Pad File")`

Function	Reference Page	Description

fileinfo(folder,file)	Page 5229	This function gets information about a file (or folder) on the disk, including the size, creation and modification date and time, type, creator and lock status. Folder is a **pathid** that unambiguously describes the location of the folder. A **pathid** is a binary data item that unambiguously describes the location of a folder on the hard disk. **pathid's** are created by the folder(and dbinfo(functions, and the openfiledialog and savefiledialog statements. If this parameter is empty text ("") the folder containing the current database is assumed. File is the name of the file that you are requesting information about.
		
		This function returns a text array with 8 elements separated by carriage returns. (However, if the specified file does not exist it returns empty text ("")). The eight elements are:
		
		1) Type of item. This is either "File" or "Folder"
		
		2) Type (4 bytes) and Creator (4 bytes). Here are some typical Type/Creator values.
		
		`ZEPDKASX  Panorama database`
		`KSETKASX  Panorama file set`
		`APPLKASX  Panorama application`
		`CWWPBOBO  ClarisWorks word processing file`
		`STAKWILD  Hypercard Stack`
		`TIFF8BIM  TIFF file (Photoshop)`
		`EPSFART3  Adobe Illustrator (version 3)`
		
		This is only a small sample of the types and creators you will find on your hard disk.
		
		3) Creation Date in internal Panorama format. Although this is a number, it has been converted to text. If you convert the number back to text you can format the date with datepattern(.
		
		4) Creation Time in seconds since midnight. Although this is a number, it has been converted to text. If you convert the number back to text you can format the time with timepattern(.
		
		5) Modification Date in internal Panorama format. Although this is a number, it has been converted to text. If you convert the number back to text you can format the date with datepattern(.
		
		6) Modification Time in seconds since midnight. Although this is a number, it has been converted to text. If you convert the number back to text you can format the time with timepattern(.
		
		7) File size in bytes. (Or if the specified file is actually a directory, this is the number of files in directory
		
		8) File status: This is either "Locked" or "Unlocked".

Function	Reference Page	Description

listfiles(folder,filter)	Page 5476	This function builds a text array listing the files in a folder. Folder is a **pathid** that unambiguously describes the location of the folder. A **pathid** is a binary data item that unambiguously describes the location of a folder on the hard disk. **pathid's** are created by the folder(and dbinfo(functions, and the openfiledialog and savefiledialog statements. If this parameter is empty text ("") the folder containing the current database is assumed. Filter is a text item that specifies what type (or types) of files (and folders) to list. If this is an empty text item ("") all files will be listed. Otherwise the type parameter should be a series of one or more 8 character sections. The first four characters are the file type, the second four are the file creator. You can also use the ? character if you do not care what a character is. Here are some useful file types:
		
		`TEXT????  list all text files`
		`APPL????  list all applications`
		`????KASX  list all Panorama database files`
		
		You can combine more than one specification into a filter, for example TEXT????????KASX to list all text files and Panorama database files.
		
		The listfiles(function normally does not list folders. However, if you precede the filter specification with the *ƒ* (option-F) character the function will list folders as well as files. For example:
		
		`ƒTEXT????`
		`  list all text files and folders`
		`ƒ????KASXTEXT????`
		`  list databases, text files, and folders`
		
		If the filter is empty ("") then ALL files and folders will be included.
		
		The function returns a carriage return separated text array. Each item contains a single file name.

info("foldersepchar")		Returns the folder separator character for the current platform (: or \).
info("lineseparator")		This function returns the line separator character on the current platform. On Macintosh systems this is a carriage return. On Windows PC systems this is a carriage return followed by a linefeed (CR-LF).
info("systemfolder")	Page 5429	This function returns a **pathid** that unambiguously describes the location of the system folder. This **pathid** can be used in other functions and statements.
info("panoramafolder")	Page 5405	This function returns a **pathid** that unambiguously describes the location of the folder containing the Panorama application. This **pathid** can be used in other functions and statements.
filedate(folder,file)		Returns the creation date of the specified file.
fileexists(folder,file)		Checks to see if a file exists. Returns true or false.
fileextension(text)		This function extracts the extension (if any) from a complete path and filename. For example the function fileextension("Alaska:Denali:Image45.jpg") would return .jpg.
filename(text)		This function extracts the filename from a complete path and filename. For example the function filename("Alaska:Denali:Image45.jpg") would return Image45.jpg.
filepath(text)		This function extracts the path from a combined path and filename. For example the function filepath("Alaska:Denali:Image45.jpg") would return the text "Alaska:Denali:".
filetime(folder,file)		Returns the creation time of the current file.
filetypecreator(folder,file)		Returns 8 characters - 4 characters with the file type and 4 characters with the creator code (for example TEXTR*ch for BBEdit text files).

Function	Reference Page	Description
folderexists(folder,foldnername)		Checks to see if a folder exists. Returns true or false.
foldersepchar()		Returns the folder separator character for the current platform (: or \).
fullpath(path)		This function converts a filename or relative path (starting with the : or / symbol) into a full path, including the disk name.
listpatnoramafiles(folder)		Returns a list of Panorama database files in the specified folder (carriage return separated).
listtextfiles(folder)		Returns a list of text files in the specified folder (carriage return separated).
panoramafolder()		This function returns a **folderid** that unambiguously describes the location of the folder containing the Panorama application. This **folderid** can be used in other functions and statements.
posixpath(path)		This function converts a MacOS format path and filename into a POSIX path that can be used as a parameter to a shell command. The input parameter must be a FULL path, including the disk name. Relative pathnames are not allowed.
panoramasubfolder(subpath)		This function makes it easy to reference any subfolder of the Panorama folder. Or you can leave it blank ("") to reference the main Panorama folder.
pathexists(path)		Checks to see if a path exists. Returns true or false.
posixpath(path)		This function converts a MacOS format path and filename into a POSIX path that can be used as a parameter to a shell command. The input parameter must be a FULL path, including the disk name. Relative pathnames are not allowed.
subfolder(folder,subfolder)		This function returns the folder ID of a subfolder of a specified folder. For example the function subfolder(info("systemfolder"),"Extensions") returns the folder ID of the Extensions folder within the System folder.
urlfilename(url)		This function extracts the filename from a complete url.
urlpath(url)		This function extracts the path from a url.

Resource Files

The Macintosh has a special kind of file, called a resource file, for storing multiple chunks of information in a single file. Each chunk of information is called a resource. Each resource may be anything from a single character to tens of thousands of bytes of information. (By the way, Panorama has a special facility to allow resource files to be used even on Windows systems. On Windows these files have the extension .RSR).

Each resource is identified by its **type** and **ID**. The **type** is a four letter designation that identifies what type of data is stored in that resource. There are hundreds of different types of resources, with more new types being created all the time. However, the most common types were defined by Apple in 1984 and are still in use today. This table describes some of the most common types.

Type	Description
MENU	List of the items in a menu
DLOG	Description of a dialog
DITL	Description of the items within a dialog
STR	Single item of text

Type	Description
STR#	Multiple items of text
PICT	Picture
ICON	Icon
ICN#	Multiple icons
cicn	Color icon
CURS	Cursor (mouse pointer)

The resource ID is simply a number between 0 and 65535.

Just as a file is identified by its folder and file name, a resource is identified by its type and ID. For example, you may refer to a resource as MENU 97 or ICON 2544.

In addition to a type and ID, a resource may also have a name. However, the name is completely optional. If a resource does have a name, you can identify the resource by its type and name as well as by its type and ID. For example you may refer to a resource as ICON 2544 or as ICON Empty Trash Can.

Before the data in a resource file can be accessed the file must be opened with the `openresource` statement (see “OPENRESOURCE” on page 5580). To learn more about how to create, modify and use resources see “Working with Resources” on page 1560. The functions that allow you to work with resources are described in the table below.

Function	Reference Page	Description

resourcetypes()	Page 5670	This function creates a text array containing a list of the resource types in all currently open resource files. The result of this function is a carriage return delimited text array. Each element in the array contains a resource type. Each resource type is a four letter text item, for example "STR " (Pascal String), "STR#" (multiple strings), "DLOG" (dialog), "DITL" (dialog items), "MENU" (menu).
		
		You can use this function to check if a particular resource type exists, or you can use the function with a pop-up menu or List SuperObject™ to allow the user to select a type of resource for any reason. The formula below will create a text array with resource types.
		
		`resourcetypes()`
		
		The result of this formula will be a list of resource types something like this:
		
		`CNTL`
		`CURS`
		`INIT`
		`KCHR`
		`LDEF`
		`MACA`
		`TPLT`
		`SIZE`
		`dctb`
		`TEXT`
		`STR#`
		`PICT`
		`PAT#`
		`MENU`
		
		As you can see, the resource types are not listed in any particular order.

Function	Reference Page	Description

resources(type)	Page 5669	This function creates a text array containing a list of resources of a particular type. Type is the resource type. This must be a four letter text item. Standard resource types include "STR " (Pascal String), "STR#" (multiple strings), "DLOG" (dialog), "DITL" (dialog items), "MENU" (menu).
		
		This function returns a text array containing a carriage return delimited list of all the resources of the specified type. Each element of this list is itself a tab delimited array. The first item is the resource item number. The second item is the resource name (if any).
		
		This example builds a list of the TEXT resources in the currently open resource files. (The currently open resource files include Panorama itself and the Macintosh system file, as well as any resource files you have opened with the openresource statement.)
		
		`resources("TEXT")`
		
		The result will be an array like this.
		
		`2001 Error Messages`
		`2002 Command List`
		`2003 Conversion Options`

getresource(type,id)	Page 5311	This function gets a resource from an open resource file and copies it into a variable. Type is the resource type. This must be a four letter text item. Standard resource types include "STR " (Pascal String), "STR#" (multiple strings), "DLOG" (dialog), "DITL" (dialog items), "MENU" (menu). ID is the identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item).
		
		The function returns whatever binary data is in the specified resource.
		
		This example procedure loads the contents of TEXT resource # 415 into the field LetterBody.
		
		`openresource "Letter Templates"`
		`LetterBody=getresource("TEXT",415)`
		
		All resource have numbers, but they do not all have names. If the resource does have a name, you can use the name for the ID. This example loads the contents of the TEXT resource named Thank You #2 into the field LetterBody.
		
		`openresource "Letter Templates"`
		`LetterBody=getresource("TEXT","Thank You #2")`

Function	Reference Page	Description

getstring(type,id)	Page 5314	This function gets a text resource from an open resource file and copies it into a variable. Type is the resource type. This must be a four letter text item. You can specify any resource type you like here, but strings are usually stored in resources of type "STR " (Pascal String). (If you specify "" for the type, Panorama will assume "STR ".) ID is the identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item).
		
		The function returns whatever text is in the specified resource.
		
		This example procedure displays the contents of STR resource # 1296.
		
		`openresource "Accounting Messages"`
		`message getstring("",1296)`
		
		All resource have numbers, but they do not all have names. If the resource does have a name, you can use the name for the ID. This example displays the text in the Overflow Error resource.
		
		`openresource "Accounting Messages"`
		`message getstring("","Overflow Error")`

getnstring(type,id,number)	Page 5307	This function gets a text resource from an open resource file and copies it into a variable. The string is extracted from a STR# resource, which holds a collection of multiple strings in each resource. Type is the resource type. This must be a four letter text item. You can specify any resource type you like here, but strings are usually stored in resources of type "STR#" (multiple Pascal Strings). (If you specify "" for the type, Panorama will assume "STR#".) ID is the identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item). Number is the number of the string item within the collection. For example, if the collection contains 6 strings they will be numbered 0, 1, 2, 3, 4, and 5.
		
		This function returns whatever text is in the specified item within the specified resource collection.
		
		This example procedure displays the contents of STR# resource # 693 item 12.
		
		`openresource "Accounting Messages"`
		`message  getnstring("",1296,11)`
		
		All resource have numbers, but they do not all have names. If the resource does have a name, you can use the name for the ID. This example displays the 12th item in the Errors collection.
		
		`openresource  "Accounting Messages"`
		`message  getnstring("","Errors",11)`

Function	Reference Page	Description
getstringmatch(type,id,text)	Page 5315	This function searches through a collection of multiple strings in a STR# resource. If it finds a match with the text you supply, it returns the number of the text item within the collection. Type is the resource type. This must be a four letter text item. You can specify any resource type you like here, but strings are usually stored in resources of type "STR#" (multiple Pascal Strings). (If you specify "" for the type, Panorama will assume "STR#".) ID is the identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item). Text is the text you want to search for. For a match, this text must be exactly the same as one of the text items in the STR# collection. This function returns a number. If the text does not match any of the text items in the STR# collection, the function will return 0. If there is a match, the function will return the number of the item that matched, starting with 1 for the first item. (Notice that this numbering system is different than the getnstring(function, which starts with 0 for the first item.)

Import/Export Functions

These functions may be used to customize the import and export of data.

Function	Reference Page	Description
import()	Page 5352	This function returns a line of imported data. This function only works in conjunction with the `importusing` (see "IMPORTUSING" on page 5355) and `arrayfilter` statements (see "ARRAYFILTER" on page 5045). The import(function returns a line of text. When it is used with the `importusing` statement, the import() function returns the contents of the line that is currently being imported. Using this function you can process and re-arrange the data as it is being imported. When it is used with the `arrayfilter` statement, the import() function returns the individual array element currently being processed. Using this function you can process the data in each array element. When it is used at any other time, the import() function returns empty text.
importcell(colNumber)	Page 5353	This function returns one cell of imported data. This function only works in conjunction with the `importusing` statement (see "IMPORTUSING" on page 5355). ColNumber is the column of data from the imported text that you want to return. The text being imported is separated into columns by either tabs or commas. The first column is column 0, the next is column 1, etc. The importcell(function always returns text. When it is used with the `importusing` statement, the importcell(function returns the contents of the specified column from the line that is currently being imported. If the text being imported is comma delimited, the importcell(function will strip off any quotes around the data before returning it. Using this function you can process and re-arrange the data as it is being imported. When it is used at any other time, the importcell(function returns empty text. It will also return empty text if you specify a column number that does not exist in the text being imported.

Function	Reference Page	Description
exportline()	Page 5210	This function generates a tab delimited line of data containing all the fields in the current record. This function is designed to be used with the `export` (see "EXPORT" on page 5207) and `arraybuild` (see "ARRAY-BUILD" on page 5038) statements, but may actually be used anywhere. The function returns a a tab delimited line of data containing all the fields in the current record. Any non-text fields (numeric, date) will be converted to text as they are placed into the tab delimited line. The tab delimited line does NOT include a carriage return on the end.
exportcell(field)	Page 5209	This function takes any database field and converts it to text, using the appropriate pattern if one has been defined in the design sheet. Field is the name of the field to be converted to text. The function always returns a text type data item. The power of the exportcell(function is that it does not require you to know what type of data you are exporting. It simply takes whatever kind of data is in the field (text, number, date, whatever) and converts it into text.

System and Database Information Functions

The functions described in this section allow a formula to access information about the computer system and about the current database.

System Information

These functions return information about the computer system — the keyboard, mouse, memory, clipboard and Panorama itself.

Function	Reference Page	Description
clipboard()	Page 5107	This function returns whatever text or value is currently on the clipboard.
info("abort")	Page 5357	The **info("abort")** function returns true if the user has pressed **Command-Period** (Macintosh) or **Control-Period** (Windows). You should use this procedure if you have used the disableabort statement and want to check for Command / Control-Period yourself.
info("availablescreenrectangle")		This function returns a rectangle defining the edges of the main screen (the screen that contains the menu bar). Any area of the screen that is reserved for the operating system (for example the dock in OS X) is excluded from the rectangle. The rectangle is in screen relative coordinates
info("click")	Page 5364	This function returns the location of the last mouse click in screen relative co-ordinates.
info("computername")		This function returns the short name of the computer (OS X only). This is the computer name that needs to be used in terminal sessions.
info("dialogtrigger")	Page 5370	This function returns the name of the last button pressed in a dialog. (Note: This function does not work with standard system dialogs like the **Open** and **Save As** dialogs.)
info("error")	Page 5373	This function can be used after an `if error` statement. It returns the error message that would have been displayed if the error had not been trapped by if error.
info("files")	Page 5376	This function builds a carriage return separated text array containing a list of all the currently open database files.
info("fonts")		This function returns a carriage return delimited array of all available fonts. This list is created when Panorama launches, and is not updated to include any new fonts that may have been added to the system since then.
info("freememory")	Page 5382	This function calculates how much free database memory is available (this does not include scratch memory).
info("keyboard")	Page 5385	This function returns the last key that was pressed.
info("keycode")	Page 5386	This function returns a special numeric code that represents last key that was pressed. This code is unique for every key in the keyboard. For example, the info("keycode") will return a different value for the 1 key on the numeric keypad and the 1 key above the Q key. See the reference page for a complete list of the numeric codes.

Function	Reference Page	Description

info("modifiers")	Page 5397	This function returns the status of the modifier keys. The five different possible modifier keys are.
		
		`"shift"`
		`"capslock"`
		`"option"  (returned when ALT key pressed on PC)`
		`"command" (returned when Control key pressed on PC)`
		`"control" (returned when Right Mouse button clicked on PC)`
		
		The info("modifiers) function also returns the status of the last mouse click. If the last mouse click was a double click, info("modifiers") will return "doubleclick". If the last mouse click was a triple click, info("modifiers") will return "tripleclick".
		
		If more than one modifier key is active the function will return all of them strung together like this:
		
		`"shift option"`
		
		You should check for a specific modifier with the `contains` operator.
		
		This example opens the Status form if the user double clicks on a button.
		
		`if info("modifiers") contains "double"`
		`  openform  "Status"`
		`endif`

info("mouse")	Page 5398	This function returns the current location of the mouse in screen relative co-ordinates.
info("mousedown")	Page 5399	This function returns true or false depending on whether or not the mouse is currently down.
info("mousestilldown")	Page 5400	This function returns true or false depending on whether or not the mouse is currently down and has not been let up since the button was pressed.
info("panoramafolder")	Page 5405	This function returns the folder id (see “Disk Files and Folders” on page 1304) of the folder containing the currently running copy of Panorama. This folder id can be used in other functions and statements.
info("panoramatoolsfolder")	Page 5407	This function returns the folder id (see “Disk Files and Folders” on page 1304) of the System:Prefs:Panorama Tools folder, if any. If there is no such folder (for example on PC systems) it returns the folder that contains Panorama itself.
info("panoramaname")	Page 5406	This function returns the name of the currently running copy of Panorama. In other words, if you have renamed your copy of Panorama this function will tell what the name is.

info("scratchmemory")	Page 5413	This function returns the amount of memory allocated for scratch memory (“Changing Scratch Memory Size (Macintosh OS 9)” on page 192). On PC systems it always returns 1000000 (one million).
		
		The procedure below checks to make sure that at least 350K of scratch memory is allocated.
		
		`if info("scratchmemory")<350000`
		`  scratchmemory 350000`
		`endif`

info("screenrectangle")	Page 5414	This function returns a rectangle defining the edges of the main screen (the screen that contains the menu bar). The rectangle is in screen relative coordinates.

Function	Reference Page	Description
info("systemfolder")	Page 5429	This function returns folder id (see "Disk Files and Folders" on page 1304) of the system folder (Mac OS). This folder id can be used in other functions and statements.

info("trigger")	Page 5433	This function returns information about how the current procedure was triggered. If the procedure was triggered by data entry this function will return the word Key followed by a period and then the key that actually triggered the procedure:
		`Key.return`
		`Key.enter`
		`Key.tab`
		If the procedure was triggered by a button, the function will return the word Button followed by a period and then the name of the button, for example:
		`Button.Save`
		`Button.Calculate Tax`
		`Button.Show Chart`
		If the procedure was triggered by a custom menu, the function will return the word Menu followed by a period, the name of the menu, another period, and then the menu item. Here are some examples:
		`Menu.Accounting.Aging`
		`Menu.Letter.New`

info("version")	Page 5440	This function returns the version of the currently running copy of Panorama. The version number is returned as text, for example 3.5.1. If you want to extract only a portion of the version number you can use the array or array functions. This example will extract the first two numbers of the version. The result will be something like 3.5 or 4.0. `arrayrange(info("version"),1,2,".")`
info("volumes")	Page 5441	This function returns a carriage return delimited array containing the name of each volume (disk) currently mounted on this computer.
info("unixusername")		This function returns the short user name the user has logged in under (Mac OS X only). This is the user name that needs to be used in terminal sessions.
os9()		This function returns true if running on Mac OS 9, otherwise false.
oswindows()		This function returns true if running on Microsoft Windows, otherwise false.
osx()		This function returns true if running on Mac OS X, otherwise false.
parameter(number)	Page 5592	This function is used to transfer data between a main procedure and a subroutine. The main procedure can set up one or more data item parameters as part of the call statement (see "CALL" on page 5086). The subroutine can retrieve and use these data items using the parameter(function. Number is a number specifying what parameter you want to retrieve. All parameters are numbered, starting with 1 (1, 2,3, 4, etc.). The function returns a data item. This data item may be text or numeric, depending on what kind of data is passed to the subroutine.

User Information

These functions return information about the person that is currently using the computer. The last three functions can only return useful information if the person has logged in. To learn more about setting up users and database security see the Panorama Database Security Supplement, which is available separately from ProVUE for a small charge.

Function	Reference Page	Description
info("user")	Page 5436	This function returns the name of the user of this computer. On Mac OS 9 computers this is the Owner Name which is set with the **File Sharing** control panel. On OS X this is the name of the logged in user.  On Windows computers this function always returns an empty string ("").

info("userid")	Page 5437	This function works with the Panorama security system. It returns the id (usually initials or the first name) the user has logged in under. If the user has not logged in, this function will return empty text (""). For example, the formula below could be used in a report header (in an auto-wrap text object or Text Display SuperObject™ to show who printed the report and when.
		`"Printed by: "+info("userid")+`
		`" @"+timepattern(now(),"hh:mm am/pm")`

info("userlevel")	Page 5438	This function works with the Panorama security system. It returns the current user level for this user, a number from 0 to 255. If the user has not logged in, this function will return 0. The example procedure below only allows users with access levels of 25 or higher to use the rest of the procedure.
		`if info("userlevel")<25`
		`message "Sorry, your access level "+`
		`"does not allow this operation"`
		`stop`
		`endif`
		`…`
		`… (rest of procedure)`
		`…`

info("username")	Page 5439	This function works with the Panorama security system. It returns the name the user has logged in under. If the user has not logged in, this function will return empty text (""). The formula below could be used in a report header (in an auto-wrap text object or Text Display SuperObject™ to show who printed the report and when.
		`"Printed by: "+info("username")+`
		`" @"+timepattern(now(),"hh:mm am/pm")`

info("unixusername")		This function returns the short user name the user has logged in under (Mac OS X only). This is the user name that needs to be used in terminal sessions.

Variable Information

These functions allow a formula to determine what variables are currently available and to access variables in other databases.

Function	Reference Page	Description
grabfilevariable(file,variable)	Page 5328	This function makes it possible to access a `fileglobal` (see "FILEGLOBAL" on page 5227) or `permanent` (see "PERMANENT" on page 5602) variable from other databases. (Usually these variables can only be accessed from the database in which they were created.) File is the name of the database that contains the `fileglobal` or `permanent` variable. Note: This database must currently be open! Variable is the name of the variable you want to access. In general, this variable name must be enclosed in quotes (unless you are using a formula to calculate the name). The result of this function is whatever value that is contained in the specified variable. This may be text or numeric.
grabwindowvariable(window,variable)	Page 5329	This function makes it possible to access a `windoglobal` variable (see "WINDOWGLOBAL" on page 5895) in a different window from the one in which it was created. Window is the name of the window in which the variable was created. (Of course the window must be open!) Variable is the name of the variable you want to access. In general, this variable name must be enclosed in quotes (unless you are using a formula to calculate the name). The result of this function is whatever value that is contained in the specified variable. This may be text or numeric.
info("filevariables")	Page 5377	This function builds a carriage return separated text array containing a list of the currently allocated fileglobal variables in the current database.
info("globalvariables")	Page 5383	This function builds a carriage return separated text array containing a list of the currently allocated global variables.
info("localvariables")	Page 5389	This function builds a carriage return separated text array containing a list of the currently allocated local variables.
info("windowvariables")	Page 5449	This function builds a carriage return separated text array containing a list of the currently allocated window variables for the current window.

Database Information

These functions return information about the currently active database.

Function	Reference Page	Description
datatype(fieldvariablename)	Page 5142	This function determines what kind of data is in a field or variable: text, number, etc. Fieldvariablename is the name of the field or variable that you want to get information about. To get information about a variable the variable name must be enclosed in quotes. The quotes are optional when accessing information about a field. The function returns the type of data from the list below: Text Compressed (Choice) Picture Date Floating Point Integer Fixed 1 Digit (#.#) Fixed 2 Digits (#.##) Fixed 3 Digits (#.###) Fixed 4 Digits (#.####) Note: The Compressed, Picture, and Date types can only occur if the datatype(function is used with a field as the parameter. Variables cannot contain data of these types (for a date, the data type is Integer).
dbfolder()		This function returns the folder ID of the folder the current database is located in.

Function	Reference Page	Description
dbinfo(option,database,)	Page 5150	This function gets information about a database: what forms it contains, what fields, what flash art pictures, etc. There are two parameters: option and database. Database is the name of the database you want to get information about. This must be a database that is currently open. If you want to get information about the current database you can use the info("databasename") function or simply use empty text (""). Option controls what kind of information this function will retrieve. There are about a half dozen possible options: "fields", "forms", "procedures", "crosstabs", "flash art", "folder", "level" and "autosave". The "fields" option produces a text array (with carriage return separators) containing a list of the fields in the database. (If a field name contains carriage returns, they are converted to spaces before being placed into the array.) The "forms" option produces a text array (with carriage return separators) containing a list of the forms in the database. The "procedures" option produces a text array (with carriage return separators) containing a list of the procedures in the database. The "crosstabs" option produces a text array (with carriage return separators) containing a list of the crosstabs in the database. The "flash art" option produces a text array (with carriage return separators) containing a list of the flash art in the database's Flash Art™ gallery. The "folder" option produces a folder id for the folder containing the database. (See "Disk Files and Folders" on page 1304.) The "level" option returns a number that indicates the privilege level for this database: 0 = author mode, 1 = user mode, or 2 = custom mode. The "autosave" option returns the number of minutes between automatic saves, or zero if the auto-save option is turned off. (See also "SETAUTOSAVE" on page 5739.) The "fileglobals" option lists all fileglobal variables associated with the database specified by the second parameter, as does the "filevariables" option. The "fieldtypes" option returns binary data with one byte per field. Each byte indicates the field type of the corresponding field, and will be one of the following values: 0=text, 4=date, 5=float, 6=integer, 7-10=fixed point. The "records" option returns the number of records in the database. The "selected" option returns the number of records in the database. The "changes" option returns the number of changes since last save. This example uses dbinfo(to calculate the number of forms in the current database. `arraysize(dbinfo("forms",""),¶))`
dbmetatag(database,attributes)		This function returns a piece of metadata information for the specified database (if any). This second parameter specifies the metadata item you want to retrieve - legal items are: Version, Title, Project, Authors, Keywords, Description. These are all case sensitive. For example, you can retrieve the document keywords with the formula dbmetatag(myDatabase,"Keywords").
dbmetatagclose()		This function returns the tag that appears after the metadata database information. This tag helps the Tiger (Mac OS X 10.4) metadata import function to locate the metadata.
dbmetatagdictionary(database)		This function returns the metadata database information for the specified database (if any). This metadata contains information like the database author, keyword and description.
dbmetatagopen()		This function returns the tag that appears in front of the metadata database information. This tag helps the Tiger (Mac OS X 10.4) metadata import function to locate the metadata.
dbname()		This function returns the name of the current database.

Function	Reference Page	Description
dbpath()		This function returns the path of the folder the current database is located in.
dbsubfolder(subpath)		This function returns the folder ID of a subfolder within the same folder as the current database. For example if the current database is in the folder "MyDrive:My Stuff", the function dbsubfolder("Images") returns the folder ID of the "MyDrive:My Stuff:Images" folder.
fieldmax(fieldname)	Page 5219	This function returns the maximum number of characters that can be stored in a field. If this is not an SQL client database, this number is always 65535. If this is an SQL client, this function returns the length of the corresponding SQL field in the server database.

fieldstyle(fieldname)	Page 5221	This function determines the style and color of a field (see "Data Style and Color" on page 474). Fieldname is the name of the field that you want to determine the style of. The function returns a text data item listing all the styles that apply to this field in the current record. The possible styles are:
		
		bold
		italic
		underline
		shadow
		black
		red
		green
		blue
		cyan
		magenta
		yellow
		
		The example below selects all the records where the name is bold.
		
		`select fieldstyle(Name)="bold"`
		
		It there is more than one style for a cell, this function will list each one. The example below will select all records where the name is italic, even if other styles also apply (for example bold italic or underline italic).
		
		`select fieldstyle(Name) contains "italic"`
		
		This final example selects all the records where the name is plain (no styles at all).
		
		`select fieldstyle(Name)=""`

listchoices(field,separator)	Page 5475	This function builds a text array containing a list of all the values stored in a specified field. (Note: this function is not related to the choices data type.) There are two parameters: field and separator. Field is the name of the field that contains the values you want to build a list of. Separator is the separator character for the text array you are building (see "Text Arrays" on page 1236). The lischoices(function scans the specified field and builds a list of all the values stored in that field. The list is returned in the format of a text array. Here is a formula that builds a list of the states in the current database. `listchoices(State,¶)`

Function	Reference Page	Description
seq()	Page 5727	This function returns a sequential numbers (1, 2, 3, etc.). This function only works in conjunction with the `formulafill`, `select`, `find` and `arrayfilter` statements. When it is used with the `formulafill`, `find` or `select` statements (see "FORMULAFILL" on page 5274, "FIND" on page 5245, and "SELECT" on page 5710), the seq() function return a sequential number for each record (the first selected record is 1, the second is 2, etc.). When it is used with the `arrayfilter` statement (see "ARRAYFILTER" on page 5045), the seq() function returns a sequential number for each element in the array being processed (the first array element is 1, the second is 2, the third is 3, etc.). When it is used at any other time, the seq() function returns the number 1. This procedure uses the seq() function to select the first 10 records in the database: `select seq()≤10`

uniqueid(field,root)	Page 5870	This function is designed for generating unique ID codes for each record in a database. The function generates ID codes with a text root and a numeric suffix (for example Jeff261). By using the machine name as the text root you can guarantee that the ID will be unique even for multiple copies of the database on different machines.
		
		The function has one parameters: field and root. Field is the name of the field that will contain the ID code. The function needs to know the name of this field so that it can scan the field to find an ID code that has not been used yet. The field name should be surrounded by quotes. For example, if the name of the field is ID, you should use "ID" as the parameter. Root is the text root that the ID code will be based on. This root may contain any kind of character, but it should not end with a numeric digit. To get a root that will be unique for each different computer you have, use the `info("user")` function for the root. This function returns the user name specified in the Sharing Setup control panel.
		
		Although you may find other uses for it, the uniqueid(function was designed specifically for creating unique Smart Merge serial numbers. Whenever a new record is added to a database that supports Smart Merge you must make sure that the ID and Modified fields are filled in. The best way to do this is to add a **.AddRecord** automatic procedure to your database. The two lines shown below will fill in the proper values.
		
		`Modified=superdate(today(), now())`
		`ID=uniqueid("ID",info("user"))`
		
		The uniqueid(function will scan the ID field to find the next serial number available. For example, if you are using a computer with a user name of Sam and the highest Sam serial number is 296, the uniqueid(function will return the value Sam297.

info("bof")	Page 5360	This function returns true if the database is currently on the first visible record. (Note: "bof" stands for "beginning of file".)
info("changes")	Page 5363	This function returns the number of changes that have been made to the current database since the last time it was saved.
info("cursorrectangle")	Page 5365	This function returns a rectangle defining the edges of the current data cell (if any). The rectangle is in screen relative coordinates (use the xytoxy((see "Rectangles" on page 1290) function to convert to window or form relative co-ordinates).
info("databasename")	Page 5367	This function returns the name of the current database. If the database name has a .pan suffix, that suffix is not included.

Function	Reference Page	Description
info("datatype")	Page 5368	This function returns the data type of the current field. The function returns a number from 0 to 10: 0 Text 1 Choice 2 Choice 3 Picture 4 Date 5 Floating Point 6 Integer 7 Fixed 1 Digit (#.#) 8 Fixed 2 Digits (#.##) 9 Fixed 3 Digits (#.###) 10 Fixed 4 Digits (#.####)

info("empty")	Page 5371	This function returns true or false depending on the result of the last select operation. If no records were selected the function will return true, otherwise it will return false. The procedure below selects all records that are "Ready", whatever that means. If there are any ready records, the procedure prints them.
		
		`select Status="Ready"`
		`if info("empty")`
		`  message  "Nothing ready today!"`
		`  stop`
		`endif`
		`print dialog`
		`field Status`
		`formulafill "Printed"`

info("eof")	Page 5372	This function returns true if the database is currently on the last visible record. (Note: "eof" stands for "end of file".)
info("expandable")	Page 5374	This function checks to see if the current record is a collapsed summary record. It returns true if the record is a collapsed summary record, false if it is a data record or an already expanded summary record.
info("fieldname")	Page 5375	This function returns the name of the current field.
info("found")	Page 5381	This function returns true or false depending on the result of the last `find` or `next` statement (see “FIND” on page 5245). If the last find or next found something, this function will return true. Otherwise it will return false.

info("proceduredatabase")		This function returns the name of the database that contains the procedure itself, even if another database is currently active. It's primarily useful when the farcall statement is used to call the procedure (or when the procedure is being used as a custom statement). Here's an example that opens the Help Window form in the database that contains the procedure, even if a window from another database is currently on top,
		
		`window info("proceduredatabase")+":SECRET"`
		`openform "Help Window"`

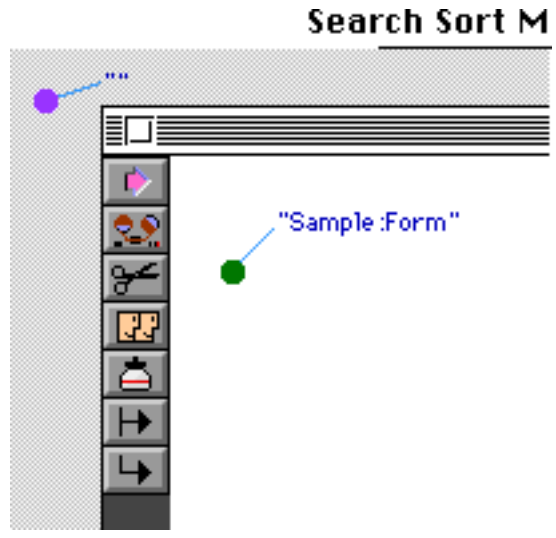
Function	Reference Page	Description

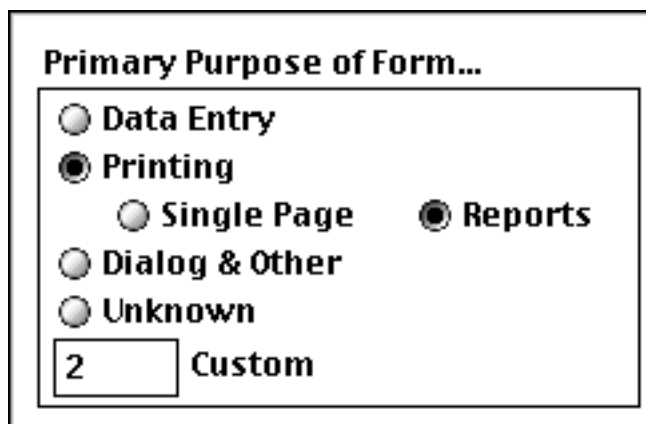
info("records")	Page 5410	This function returns the total number of records in the current database. To find out the number of selected records, use `info("selected")` (see below).
		This example checks to see if all records are selected. If some records are not selected, the procedure does a selectall statement.
		`if info("selected") <info("records")`
		`  selectall`
		`endif`

info("selected")	Page 5415	This function returns the number of selected records in the current database. To find out the total number of records, use `info("records")` (see above).
info("stopped")	Page 5426	This function returns true or false depending on the result of the last `uprecord`, `downrecord`, `left` or `right` statement. If the statement could not move the active cell because the active cell was already as far as it could go, the function will return true. Otherwise it will return false.
info("summary")	Page 5428	This function returns the summary level of the current record, from 0 (data record) to 7 (see "3-Step Summarizing" on page 393).
info("tabdown")	Page 5430	This function returns true if the tab down option is on, false if the tab down option is off (see "Tab Down" on page 315).
info("visible")		This function returns true/false result based on whether or not the current record is visible. Useful for situations where Panorama may scan invisible records, for example the arraybuild and select statements, also the Scrolling List SuperObject.

Window, Form and Report Information

These functions return information about the current database and its windows, forms reports and objects.

Function	Reference Page	Description
extrapages(pagelist)	Page 5213	This function is used to control the printing of extra pages This function must be used in an auto-wrap text object, it has no effect in any other situation. Pagelist is a text item listing the extra pages that should be printed. For example, if you want to print data tiles 3 and 5 the page list should be "35". See “Selectively Printing Multiple Pages per Record” on page 1103.
findwindow(point)	Page 5250	This function checks to see if a point (in screen relative co-ordinates) is inside any Panorama window. If it is inside a window, this function returns the name of the window. Point is a point, which must be in screen relative co-ordinates. All measurements are in pixels (1 pixel = 1/72 inch). The function returns a text item. If the point is inside a Panorama window, the function returns the name of the window. You can use the window statement to bring this window to the top. If the point is not inside any Panorama window the function returns empty text (""). This illustration shows a window and two points. The green point is inside the window, so the findwindow(function will return the window name, in this case Sample:Form. The purple point is not inside the window, so the findwindow(function will return "". 

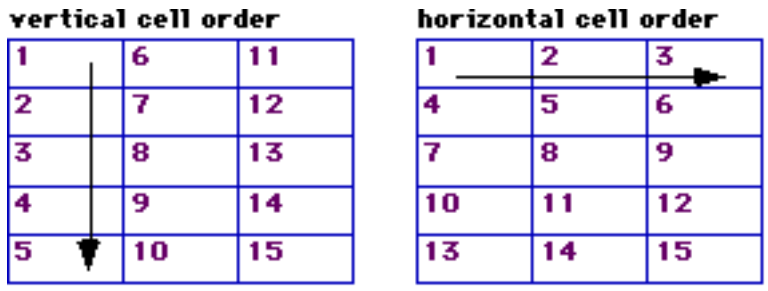
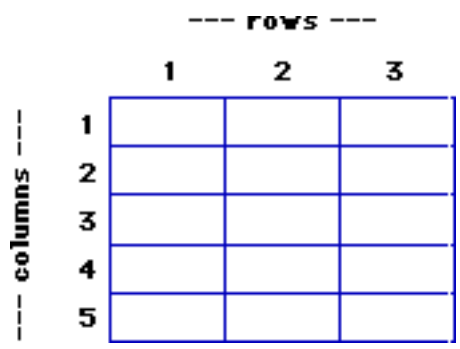
Function	Reference Page	Description
formtype(database,form)	Page 5270	This function returns the form type (a number) for any form in any open database. The form type is a number that you can set up using the **Form Comment** dialog in the Setup menu (see "Form Comments" on page 1845). Database is the name of the database that contains the form. The database must be currently open. If this parameter is empty text ("") the current database is assumed. Form is the name of the form. This function returns a number (integer) from 0 to 255. The value of this number depends on the Primary Purpose of Form area of the **Form Comment** dialog (in the graphics mode Setup Menu.) There are predefined radio buttons for 1) Data Entry, 2) Printing, and 3) Dialog and other. Or you may enter any value from 0 to 255 in the Custom area. The default value of a form is 0 (unknown). 
listwindows(file)	Page 5478	This function builds a text array containing a list of all the open windows associated with a particular file. File is the name of the database file that you want to list the windows of. This should be the name of an open database. If the file parameter is empty ("") the listwindows(function will list all open windows, no matter what database they are in. The function scans the windows and builds a text array using carriage returns (¶) as separators (see "Text Arrays" on page 1236). The windows are listed in order from front to back.

Function	Reference Page	Description

objectinfo(option)	Page 5557	This function returns information about a graphic object: its location, size, color, font, etc. This function must be used in combination with either the `object` (see "OBJECT" on page 5555), `selectobjects` (see "SELECTOBJECTS" on page 5721) or `changeobjects` (see "CHANGEOBJECTS" on page 5097) statement.
		
		Option is the type of information you want to retreive about an object. You must pick the option from the list below:
		
		`objectinfo("rectangle")`
		`objectinfo("ID")`
		`objectinfo("name")`
		`objectinfo("fieldname")`
		`objectinfo("type")`
		`objectinfo("custom")`
		`objectinfo("font")`
		`objectinfo("textsize")`
		`objectinfo("textstyle")`
		`objectinfo("alignment")`
		`objectinfo("color")`
		`objectinfo("selected")`
		`objectinfo("locked")`
		`objectinfo("expandable")`
		`objectinfo("expandshrink")`
		`objectinfo("tile")`
		`objectinfo("text")`
		`objectinfo("fillpattern")`
		`objectinfo("linepattern")`
		`objectinfo("linewidth")`
		`objectinfo("count")`
		`objectinfo("boundary")`
		
		See the reference page for details on each of these options.

overflow()	Page 5587	This function is used with auto-wrap text objects and an overflow report tile to print text or graphics that won't fit on a single page. For example, you can use this function to help print multiple page letters. See "Printing Data that Overflows a Page" on page 1104 for more information on using this function.

Function	Reference Page	Description
textdisplay(color,style)	Page 5852	This function works with Text Display SuperObjects™. By using this function as the first part of the formula in a Text Display SuperObject™ you can control the color and style of the text on the fly (see "Controlling Text Display Color and Style on the Fly" on page 619). For example, you can automatically display all negative numbers in red. (Advanced note: The textdisplay(function actually generates a special header that is intercepted and removed by the Text Display SuperObject™. The header contains information the Text Display SuperObject™ uses to select the style and color.) This function has two parameters: color and style. Color is the color that should be used to display the text. See the rgb(function. If you pass "" for this parameter the text will be displayed in the normal color for this object. Style is the style or combination styles that should be used to display the text. For a single style by itself simply use the name of the style: `"Plain", "Bold", "Italic", "Underline", "Outline"` or `"Shadow"`. If you want to combine multiple styles together you must specify the style numerically. Add up the numbers for the styles you want from the table listed below. For example, for bold italic text the style should be 3. 0 Plain 1 **Bold** 2 *Italic* 4 Underline 8 Outline 16 Shadow
info("activesuperobject")	Page 5358	This function returns the name of the currently active text editor or word processor SuperObject, if any. If no such object is currently being edited, the function returns empty text ("").
info("buttonrectangle")	Page 5361	This function returns a rectangle defining the edges of the button that was clicked on (needless to say, this function should be used in a procedure that is triggered by a button). The rectangle is in screen relative coordinates (use the xytoxy(function to convert to window or form relative co-ordinates).
info("cursorrectangle")	Page 5365	This function returns a rectangle defining the edges of the current data cell (if any). The rectangle is in screen relative coordinates (use the xytoxy((see "Rectangles" on page 1290) function to convert to window or form relative co-ordinates).
info("formcolor")	Page 5378	This function returns the background color of the current form (see "Colors" on page 1294). If the current window does not contain a form, the function will return empty text ("").
info("formcomment")	Page 5379	This function returns the form comment that has been set up for the currently open form (if any). See "Form Comments" on page 1845.
info("formname")	Page 5380	This function returns the name of the current form. If the current window does not contain a form, the function will return empty text ("").

Function	Reference Page	Description
info("matrixcell")	Page 5391	This function is designed to be used with Matrix SuperObjects™ (see "Matrix Co-Ordinates (What cell is this?)" on page 943). The function returns the cell number within the matrix, starting with 1 in the upper left hand corner. If the matrix order is horizontal, then the cell numbers will be consecutively numbered from left to right in each row. If the matrix order is vertical, then the cell numbers will be consecutively numbered from top to bottom in each column. The illustration shows two matrixes with the cell number displayed in the upper right hand corner. The matrix on the left has vertical cell order, the matrix on the right has horizontal cell order. 
info("matrixcelldata")		This function returns the data associated with the current matrix cell. This function is only valid if a formula has been defined for this Super Matrix Object. Note: It's up to you what to do with this data. You can display it using a Text or Flash Art object, or simply ignore it if you want. The Matrix object doesn't display this data automatically. Example: The formula below could be used in a Text Display SuperObject to display the cell number and data. `str(info("matrixcell"))+" "+info("matrixcelldata")`
info("matrixcolumn")	Page 5392	This function is designed to be used with matrix SuperObjects™ (see "Super Matrix Objects" on page 929). The function returns the column number, starting with 1 for the left hand column and increasing by one for each column to the right. The illustration shows the columns and rows for a Matrix SuperObject. 

info("matrixdata")		This function returns the all of the data associated with the current matrix. This function is only valid if a formula has been defined for this Super Matrix Object. Note: It's up to you what to do with this data. You can display it using a Text or Flash Art object, or simply ignore it if you want. The Matrix object doesn't display this data automatically.
		Example: The formula below could be used in a Text Display SuperObject to display the data and the cell number and position, for example Orange [4 of 17] .
		`info("matrixcelldata")+" ["`
		`+str(info("matrixcell") )+" of "+`
		`str(arraysize(info("matrixdata")+`
		`, info("matrixseparator") ) +"]"`

Function	Reference Page	Description
info("matrixrow")	Page 5394	This function is designed to be used with matrix SuperObjects™. The function returns the row number, starting with 1 for the top and increasing by one for each row as you go down. The illustration above the columns and rows for a matrix SuperObject.
info("matrixseparator")		This function returns the separator associated with the current matrix. This function is only valid if a formula has been defined for this Super Matrix Object. Example: The formula below could be used in a Text Display SuperObject to display the data and the cell number and position, for example Orange [4 of 17] . `info("matrixcelldata")+ " ["+str(info("matrixcell"))+ " of "+str(arraysize(info("matrixdata")+ , info("matrixseparator"))+"]"`
info("maximumwindow")	Page 5395	This function returns the largest possible rectangle for this window. This can be controlled by setting up an Auto Grow SuperObject for this window (see “Maximum Window Size” on page 919).
info("minimumwindow")	Page 5396	This function returns the smallest possible rectangle for this window. This can be controlled by setting up an Auto Grow SuperObject for this window (see “Maximum Window Size” on page 919).
info("pagecount")		The new info("pagecount") function calculates the total number of pages that will be printed. For example, if you wanted to print Page 1 of 4, Page 2 of 4, etc. on the top of each page you would use a formula of `"Page "+str(info("pagenumber"))+" of " +str(info("pagecount"))` This function is only valid when used in an object in a form (Text Display SuperObject, auto-wrap text object, etc.) that is being printed.
info("pagenumber")	Page 5403	When printing, this function returns the current report page number. This function is designed to be used as part of an auto-wrap text object or Text Display SuperObject™ in a report form. See “Page Numbers” on page 1088.
info("reportcolumns")	Page 5411	This function returns the number and direction of report columns that have been set up for the currently open form (if any). The function returns a text string that contains the number of columns followed by the direction, for example "3 Down" or "2 Across". See “Controlling the Number of Columns” on page 1127.
info("rulers")	Page 5412	This function returns the current measurement units for the ruler in this form. The function may return five possible values: Inches, Centimeters, Pixels, Deca-Pica, or Deca-Elite. See “Rulers” on page 506.
info("screenrectangle")	Page 5414	This function returns a rectangle defining the edges of the main screen (the screen that contains the menu bar). The rectangle is in screen relative coordinates.

Function	Reference Page	Description

info("typeofwindow")	Page 5435	This function determines what type of window the current window is. The window may be one of the types listed below:
		
		`Data Sheet`
		`Form (Data Mode)`
		`Draw (Graphics Mode)`
		`View As List`
		`Design Sheet`
		`Cross Tab Sheet`
		`Floating Input Window`
		`Procedure`
		`Flash Art Gallery`
		`Clipboard`
		`Memory Usage`
		`Print Preview`

info("windowbox")	Page 5442	This function returns the dimensions of the current window in screen relative co-ordinates. All four dimensions are returned in a text string, for example "34 123 490 630". This is a “classic” Panorama function that is retained for compatibility with older databases. For new applications we recommend using the `info("windowrectangle")` function (see below).

info("windowdepth")	Page 5443	This function returns the pixel depth of the current window. This table shows the possible values:
		
		`1 Black and White`
		`2 4 color`
		`4 16 color`
		`8 256 color`
		`16 Thousands of colors`
		`32 Millions of colors`
		
		If the window crosses over two monitors with different pixel depths, the info("windowdepth") function will return the lower value.
		
		The formula below could be used in a Flash Art object. If this is a black and white monitor it displays the picture bwSky, otherwise it displays the picture Sky.
		
		`?(info("windowdepth)=1,"bwSky","Sky")`

info("windowname")	Page 5445	This function returns the name of the current window.
info("windowrectangle")	Page 5446	This function returns a rectangle (see “Rectangles” on page 1290) defining the edges of the current window. The rectangle is in screen relative coordinates.
info("windows")	Page 5447	This function builds a carriage return separated text array (see “Text Arrays” on page 1236) containing a list of all the currently open windows. The windows are listed in order from front to back.

Function	Reference Page	Description
info("windowtype")	Page 5448	This function determines what number type the current window is. The window number may be one of the listed below: 2 = Data Sheet 5 = Form (Data Mode) 6 = Draw (Graphics Mode) 15 = View As List 7 = Design Sheet 10 = Cross Tab Sheet 1 = Desk Accessory 3 = Floating Input Window 8 = Procedure 11 = Flash Art Gallery 13 = Clipboard 12 = Memory Usage 14 = Print Preview

SQL Database Information

These functions provide information about the SQL database associated with the current Panorama database (if any).

Function	Reference Page	Description
info("plugandrun")	Page 5408	This function tells how Panorama will resolve conflicts between the client and server when the client database is reconnected to the server after being used off line. There are four possible modes: **off** This is the value that will be returned if this is a single user Panorama database that is not linked to an SQL server database. **client** This means that if a record has been modified by both the client and the server, the clients changes will be kept and the servers changes will be discarded. **server** This means that if a record has been modified by both the client and the server, the servers changes will be kept and the clients changes will be discarded. **manual** This means that if a record has been modified by both the client and the server, the user will be presented with a list of the changed record and allowed to "cherry pick" which records to keep.
info("serverfile")	Page 5420	This function returns the name of the SQL database linked to this Panorama database, if any.
info("serverrecordid")	Page 5421	This function returns the internal serial number on the server of the current Panorama record. This number is guaranteed to be unique in this database if this is a SQL connected database. If this is a standalone database, this function will return zero for all records. This function will also return zero for new records created while disconnected to the server. These new records are not assigned an internal serial number until the next time the database is synchronized with the server.

Function	Reference Page	Description
info("serverrecordts")	Page 5422	This function returns the internal "time stamp" number Panorama uses to determine which records need to be synchronized. By itself, this number is basically meaningless. However, if the time stamp for record A is higher than record B, then record A was edited later than record B. If this is a standalone database, this function will return zero for all records. This function will also return zero for new records created while disconnected to the server. These new records are not assigned an internal time stamp until the next time the database is synchronized with the server. This function is intended for debugging purposes only. It is included here for completeness. However, it could possibly have useful non-debugging purposes.
info("serverstatus")	Page 5423	This function returns the connection status of the SQL database linked to this Panorama database, if any. There are four possible values: **Standalone (Read/Write)** This is the value that will be returned if this is a single user Panorama database that is not linked to an SQL server database. **Connected (Read/Write)** This is the value that will be returned if this is a multi user Panorama database that is linked to an SQL server database, and the link is currently open with full record locking. **No Connection (Read/Only)** This is the value that will be returned if this is a multi user Panorama database but there is currently no network connection to the SQL server database. For example the user might be using this database on a laptop computer with no connection to the server. This database does not allow off-line editing, so the database cannot be edited. **Standalone (Read/Write)** This is the value that will be returned if this is a multi user Panorama database but there is currently no network connection to the SQL server database. For example the user might be using this database on a laptop computer with no connection to the server. This database does allow off-line editing, so the database may be edited. The changes made off-line will be saved and synchronized the next time the server is available.
info("servertimeout")	Page 5424	This function returns the maximum time Panorama will keep a record locked with no keyboard or mouse activity. This timeout can help prevent a user from starting to edit a record and then walking away from the computer and leaving the record locked and unavailable to other users indefinitely. The time interval is specified in seconds. A timeout value of zero indicates no timeout (infinite time).
info("subsetformula")	Page 5427	The **info("subsetformula")** function returns the formula used to extract the current local subset from the server database. If the local database contains a copy of the entire server database (select all) the result will be an empty string ("").

Custom Functions

Panorama doesn't limit you to the built-in functions that are supplied with a Panorama. In fact, you can actually create your own user defined functions that can be used in any formula. To build a custom function you assemble it from the functions and operators that already exist. For example, you could define a new `money(` function with one parameter:

```
money(number)
```

This custom function can be defined using Panorama's built in `pattern(` function, like this.

```
pattern(number,"#,.##")
```

Once the function has been defined you can use it in any formula. For example, the formula

```
money(54321.5678)
```

would result in the text value 54,321.56.

The Custom Functions Wizard

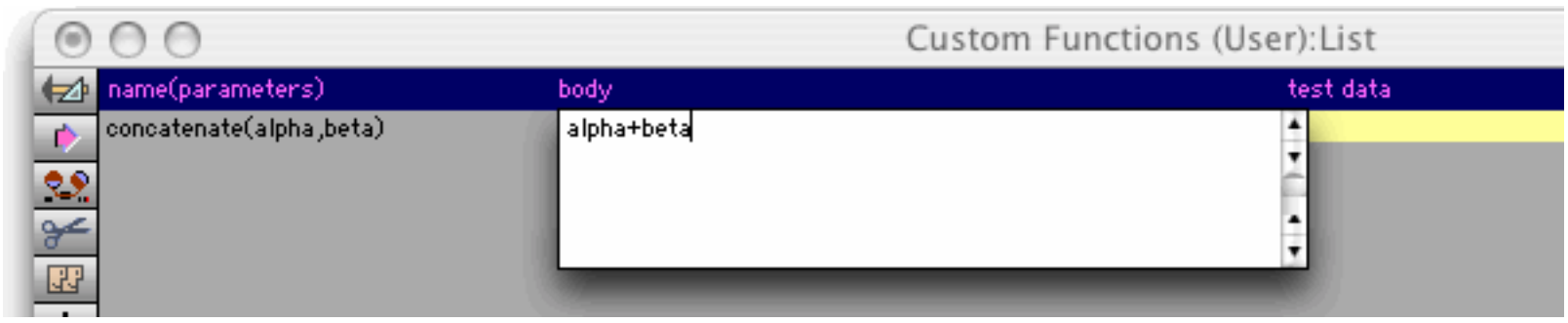
To create your own custom functions you'll use the **Custom Functions (User)** wizard. You'll find this wizard in the Developer Tools submenu of the Wizard menu. This wizard has a view as list form with four columns (three of which are shown in the illustration below. Each line is a slot for a custom function.



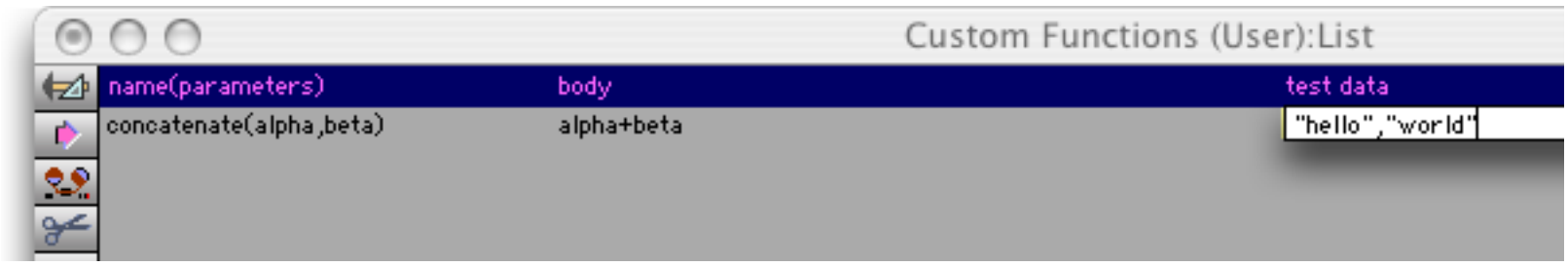
To create a new custom function, use the **Add New Record** tool to add a line to the database. In the leftmost column, type in the name of the function and parameters, like this:



Now tab over to the next column (labeled "body"). Enter the formula that should be used to calculate the custom function's value. For example, the body for the concatenate function could be:



Now you can try out your new function. Type in sample parameters into the test data column, for example:



When you press the **Enter** or **Return** key the results column will show the result of your function (or you'll see an error message if you made a mistake). In this case the result is helloworld.



That's all there is too it! Your custom function is now ready to use in any formula.

Function Names

Function names may consist only of alphabetic characters - no numbers, punctuation or other characters. Upper or lower case is ok, but it is ignored, so you can't create separate functions called `alpha(` and `ALPHA(`. If you rename a custom function the old function will continue to work until you restart Panorama. For compatibility with future versions of Panorama we recommend that you avoid generic names that might be used by a built in Panorama function in the future. If a future version of Panorama uses the name you have chosen, your function will no longer work.

Parameter Names

The name of each parameter must be unique within the body of the function definition. This means you should not use generic names like text or sum. You especially should avoid single letter names like x, y, z, a, b, etc. If you want to make absolutely sure to avoid problems you can put chevrons around the parameters, for example `«number»` or `«text»`. You'll need to use these chevrons in both the name and body fields.

Advanced Topic: The FDF File

When Panorama starts, it looks inside the Panorama folder for a Functions folder inside the Extensions folder. This folder may contain one or more files with names ending in .fdf. These .fdf files contain the actual custom function definitions. Panorama will automatically open and process these files each time it launches. You can't look at these files directly, but they are generated automatically by the Custom Function wizards.

Advanced Topic : Creating Custom Functions In A Procedure

If you are distributing an application to other users you may want to define functions in the .Initialize procedure of your database, instead of using the wizard. This is possible with the RegisterFunction procedure statement. This statement has four parameters: folder, name, parametercount and body.

folder - This parameter is currently unused. For now, simply supply an empty string ("").

name - This is the name of the function. It must be all uppercase letters, terminated by a (. Here are some valid examples:

```
CONCATENATE(
MYCOOLFUNCTION(
```

parametercount - This is the number of function parameters.

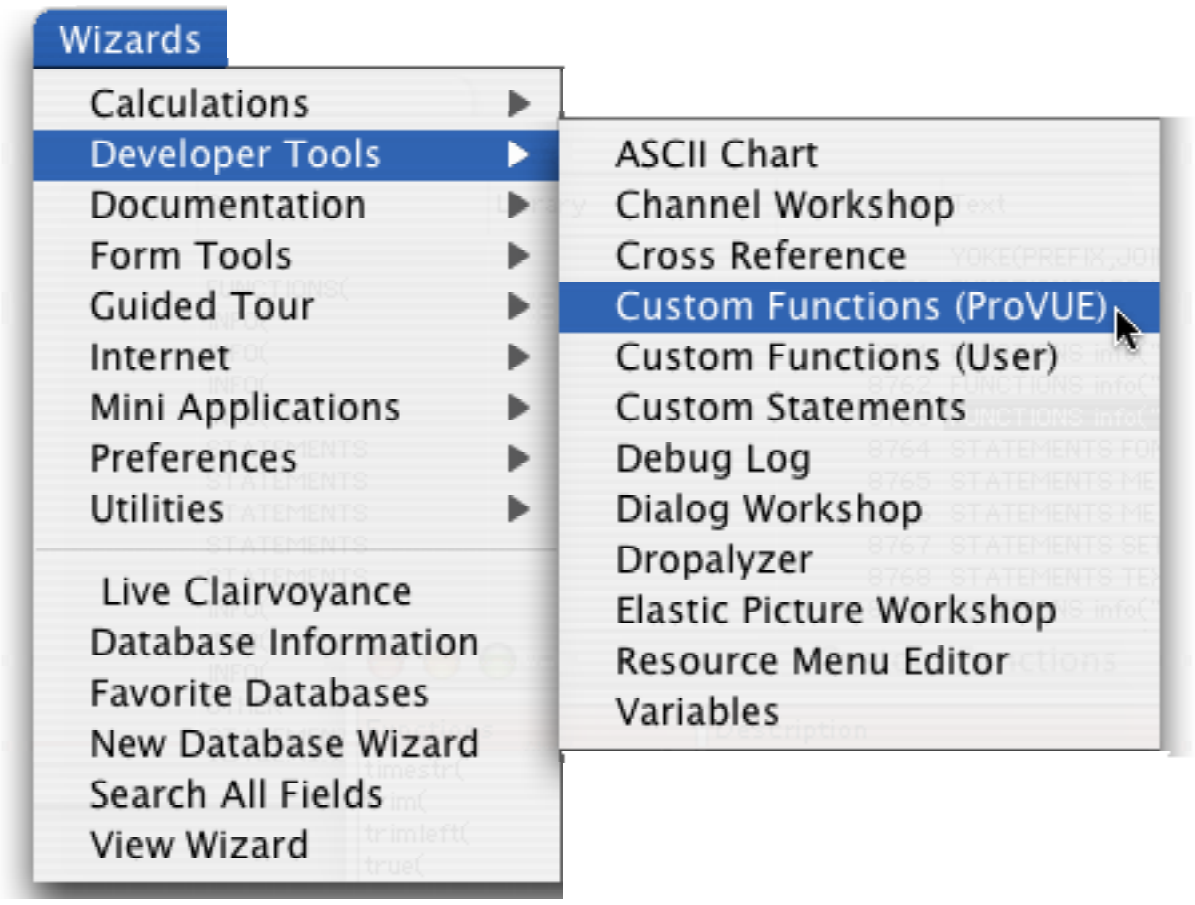
body - This is the the body of the function. Parameters should be represented as «1, «2, «3, etc.

This example shows how to add a custom `concatenate(` function:

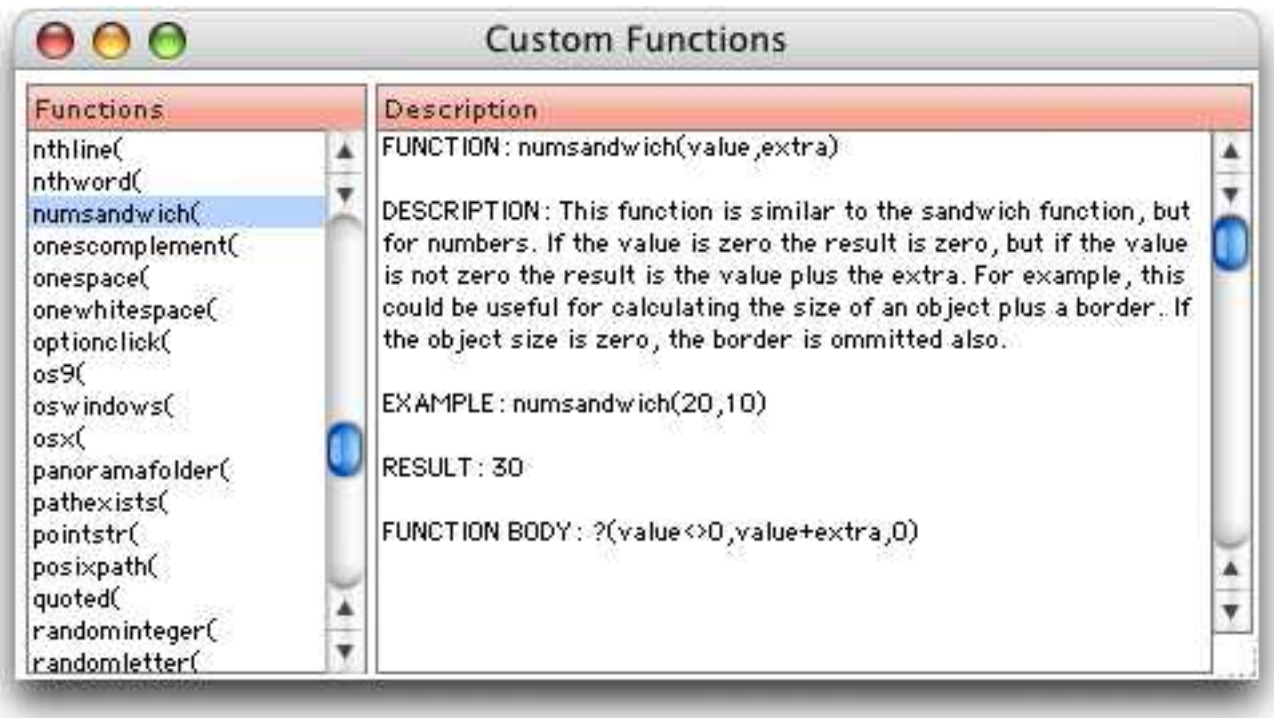
```
registerfunction "","CONCATENATE(",2,{«1+«2}
```

The Custom Functions (ProVUE)

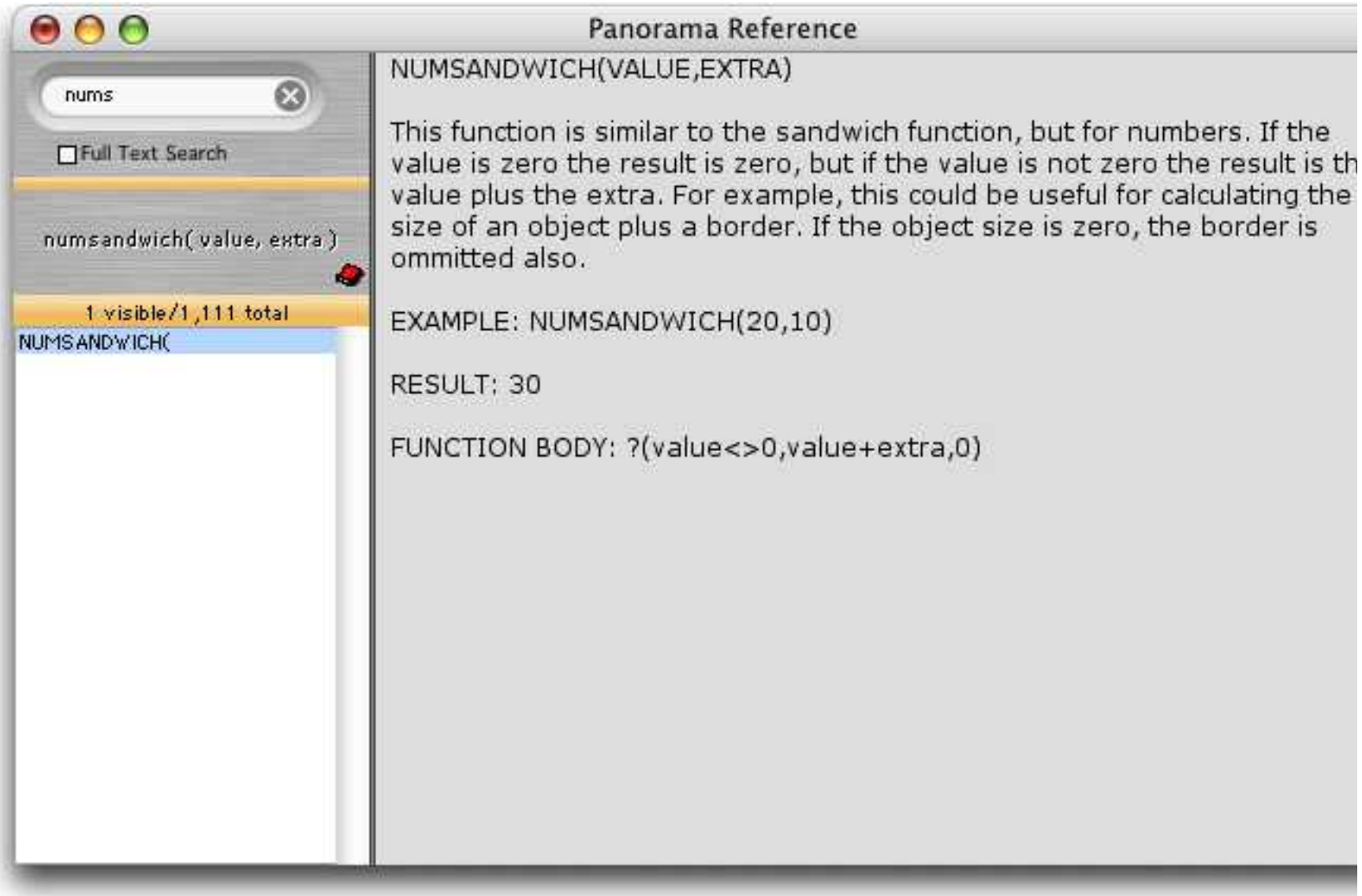
Panorama includes a number of custom functions that have already been defined for you. To see a list of these functions open the **Custom Functions (ProVUE)** wizard.



All of the functions displayed in this wizard are ready to use (and most are already described earlier in this chapter).



You can also find these functions in the Programming Reference Wizard. It's easy to identify custom functions in this wizard because the information is all displayed in a single, plain font.



If you have any suggestions for additional functions you would like to see appear in this library, please let us know.

Chapter 24: Procedures



Right out of the box, Panorama is a very flexible program. Its built in menus and tools bring incredible power to your fingertips. In spite of this power, however, Panorama is a general purpose tool. To get the most out of Panorama you'll want to customize it to meet the specific needs of your business or industry. Doing this takes an up front investment of time and / or money. But if done properly, the payoff can be huge—a tool optimized specifically for running and organizing your business or life, not someone else's idea of how things should be done. You'll save time, reduce errors, and look more professional to your customers, vendors, employees and / or supervisors.

Programming Isn't Magic!

If you've never programmed before, the idea of programming may seem like magic. But really there's nothing magic about it at all. Programming doesn't really add any new features or capabilities to Panorama. Anything that can be done with programming can also be done manually with Panorama's standard menus and tools.

If programming doesn't add any new features, what is it for? Many database tasks take more than one step to complete. To set up a report, for example, you may need to select certain information, sort the database, and perform calculations. A program allows you to define such a sequence of steps in advance. Once the sequence of steps is defined, you don't have to perform that sequence of steps manually any more. Simply ask the computer and it will perform the steps for you, flawlessly and at the highest possible speed. This lets the computer do what it does best, remember things accurately, and frees your mind for more important tasks. Eventually you can teach Panorama all the everyday tasks you need for running your organization. Of course every journey begins with a single step, so let's get started!

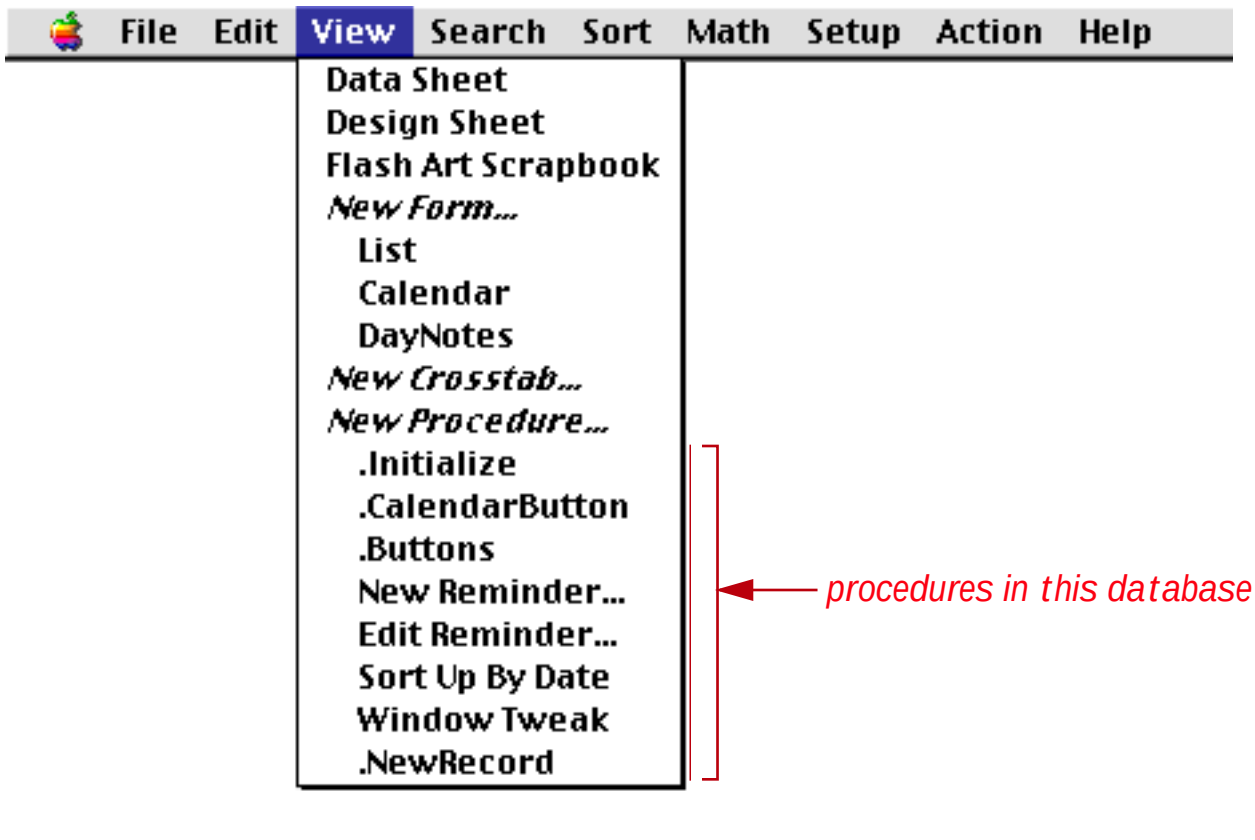
Introduction to (Panorama) Programming

The next few pages introduce the fundamentals of programming with Panorama. You'll learn how complex programs are assembled from a series of small steps, and you'll learn the nuts and bolts of actually creating and using programs. If you are experienced with other programming languages like C or Basic, much of this material will be familiar to you already. If not, welcome to the exciting world of computer programming!

Procedures

A complete program is assembled by combining a series of steps together so that they perform a complete task. In Panorama this complete series of steps is called a procedure. Each procedure performs a complete task from start to finish.

Each database can contain many procedures—one for each task that needs to get done in that database. To help keep all these procedures straight, Panorama requires you to give each procedure a unique name. The procedure name is used whenever you need to refer to the procedure—in a menu, a button, etc. All of the procedures in a database are listed in the **View** menu.



You can view a procedure by selecting it from this menu (“Switching Between Views” on page 222). You can also open the procedure in a new window (see “Opening More Than One Window Per Database” on page 223) by holding down either the **Control** key (Macintosh) or the **Alt** key (Windows) while you select the procedure from the **View** menu.

Statements

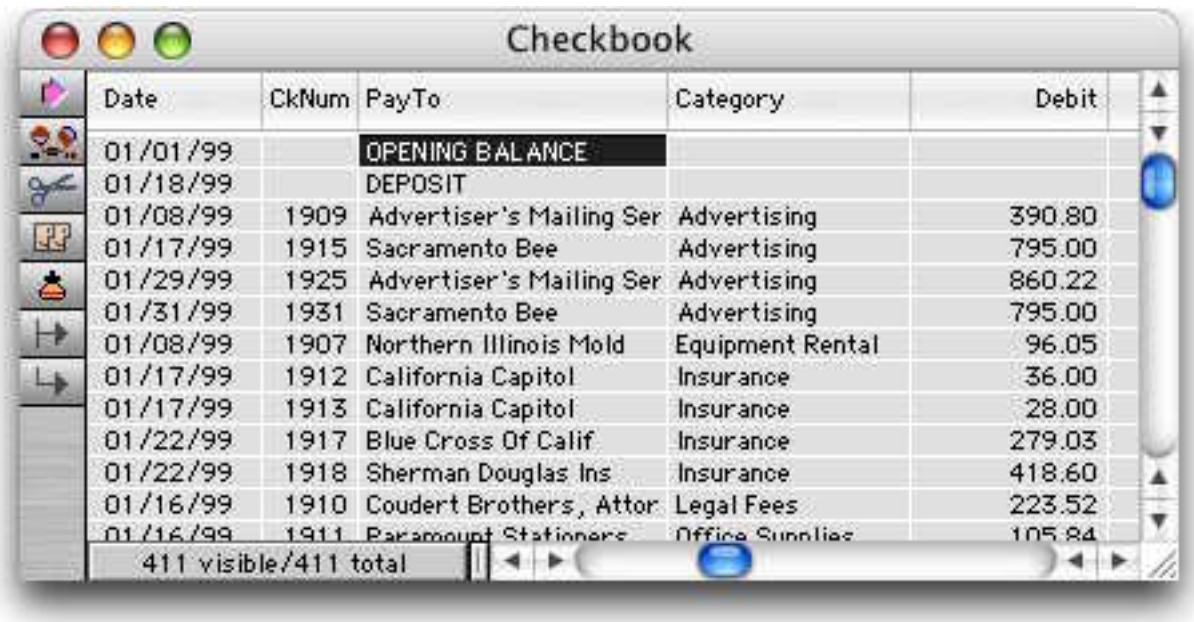
As mentioned in the previous section, a procedure is simply a series of steps. Programmers have a special name for these steps—they call them statements. Each statement is simply a single step to be performed by the computer. Most statements start with a special word (sometimes called a command or keyword) that tells Panorama what the statement should do, for example `SortUp`, `Select`, `Print`, `Open`. Panorama understands several hundred different keywords that perform a wide variety of operations.

A statement may consist of a keyword all by itself, for example `SortUp` or `CloseWindow`. However, many keywords also require additional options, for example `Open "MyDatabase"` or `Select Price>200`. These additional options are called parameters. If a keyword uses parameters, they must follow the keyword in the program.

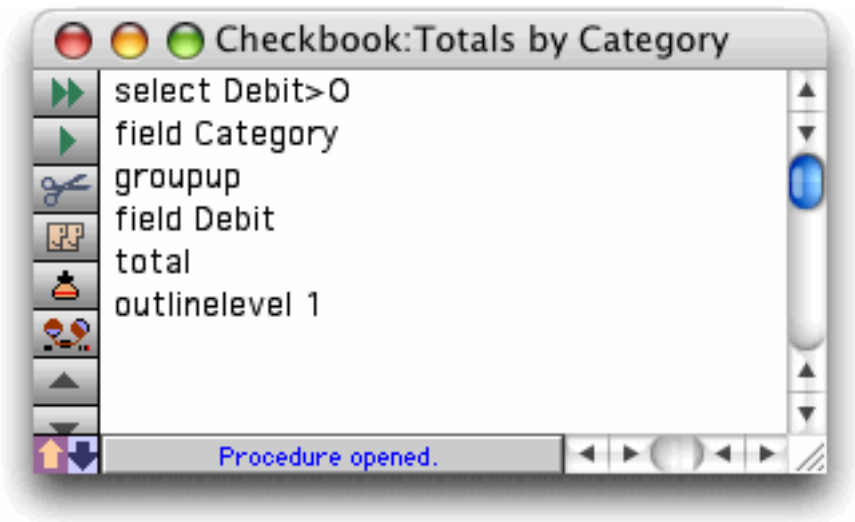
(Note to experienced programmers: Unlike many programming languages, Panorama’s keywords or commands are not reserved words. This means, for example, that you can use a database column named Print or create a variable named Open. These names are perfectly OK in Panorama’s programming language. However, using keywords in this way could easily result in programs that are very confusing to read, so we recommend that you avoid keywords when you are defining fields and variables.)

A Simple Procedure in Action

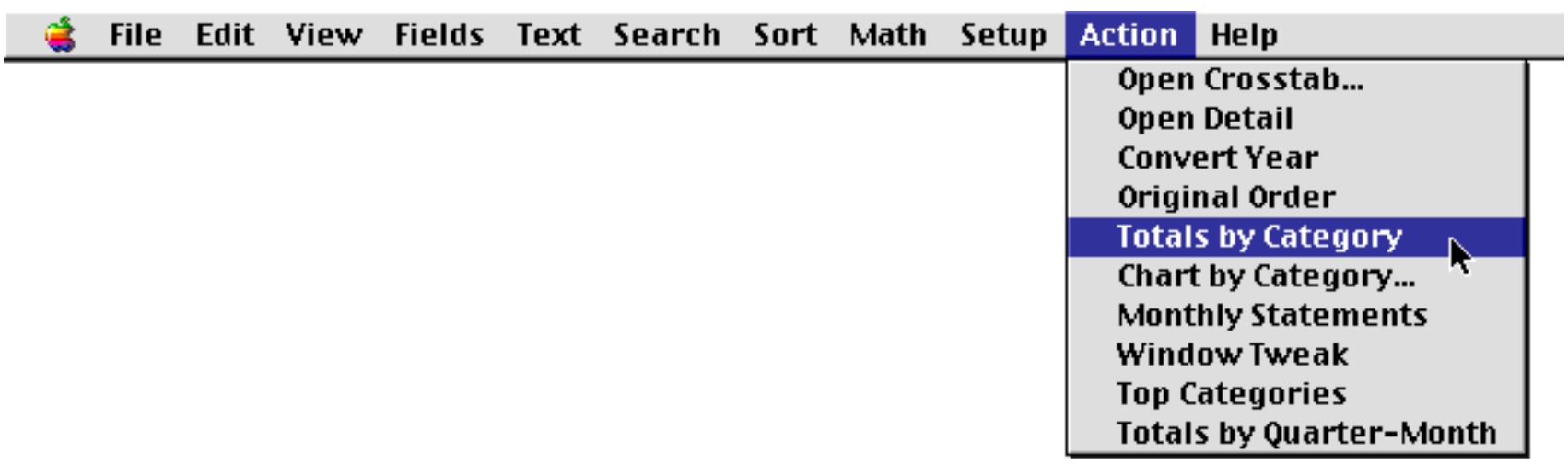
Let's take a look at a simple procedure and see how it works. This procedure is part of a Checkbook database that looks like this:



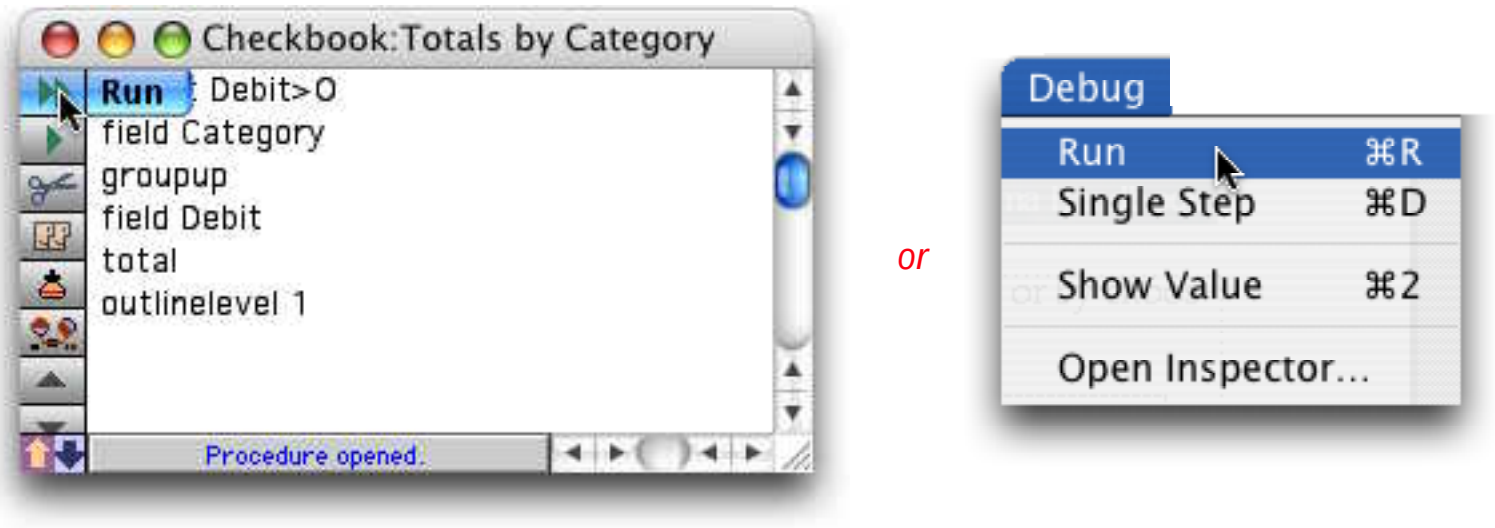
The procedure itself contains six statements. In this procedure each statement is on its own line, but as you'll see later this is not necessary.



To use a procedure you need to start it somehow. (This is called "triggering" the procedure.) There are two easy ways to do this. If the data sheet is currently the top window you can trigger the procedure by choosing it from the **Action** menu.



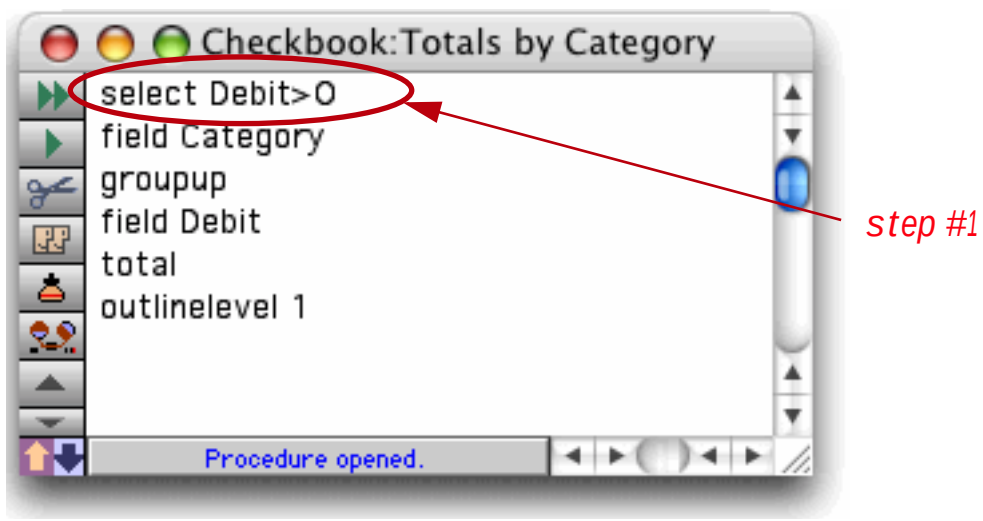
If the procedure itself is currently the top window you can trigger it by clicking on the **Run** tool, or by choosing **Run** from the Debug menu.



Once the procedure is triggered Panorama begins performing the steps in the procedure. You can watch as Panorama rapidly performs each step, kind of like a fast action “Keystone Kops” movie.

Let’s take a look at how Panorama performs each of the steps in our sample procedure, starting with step number 1.

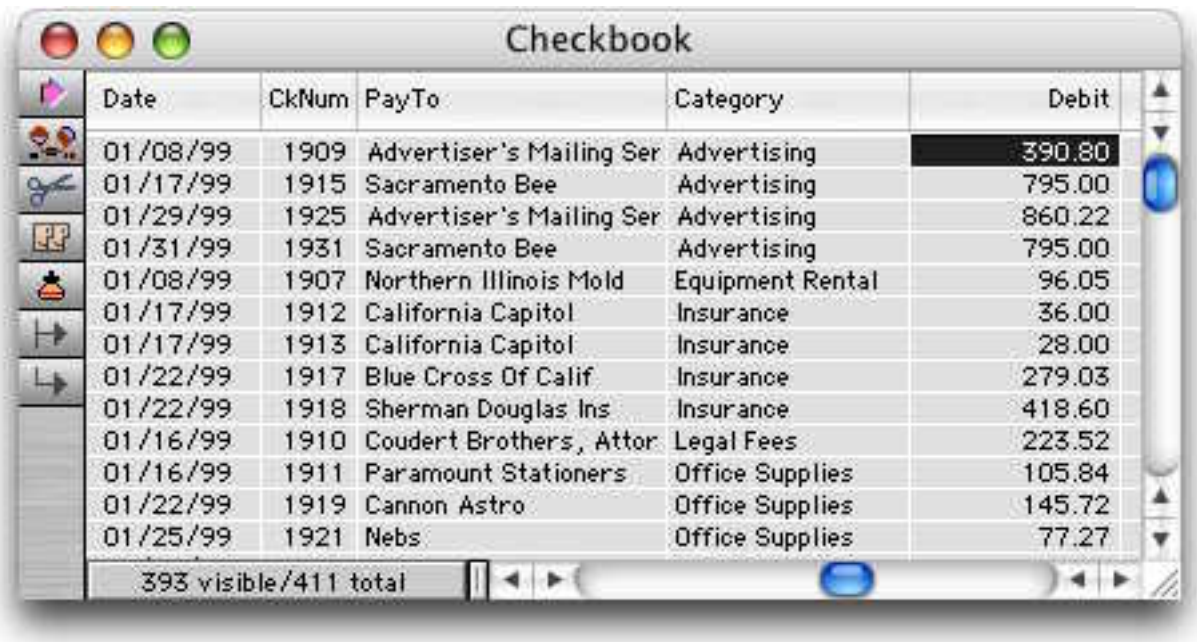
```
select Debit>0
```



This first statement selects a subset of the data. Panorama performs this step exactly as if you had chosen the **Find/Select** command (see “The Find / Select Dialog” on page 369) and filled in the dialog like this. (However since Panorama already knows what to do it doesn’t actually display this dialog.)

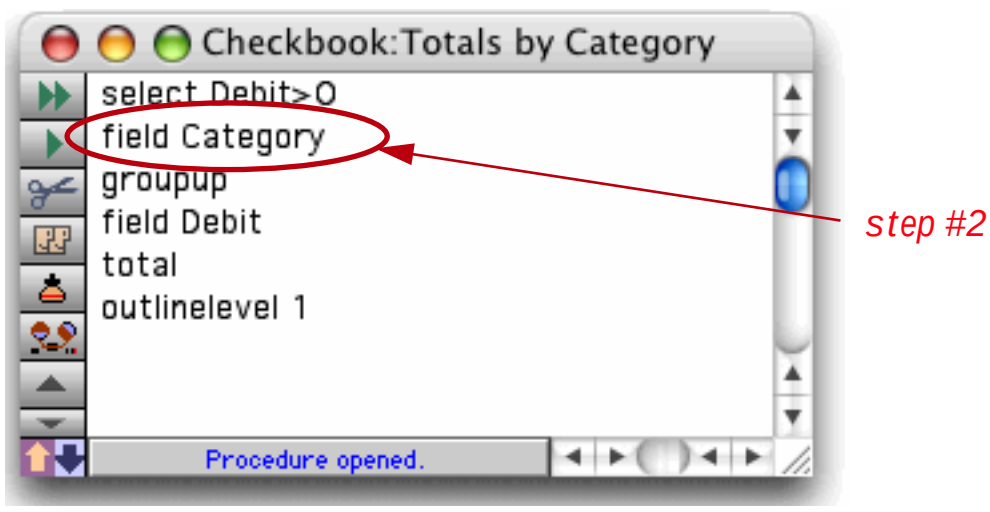


The result of this step is that a subset of the database is now selected.

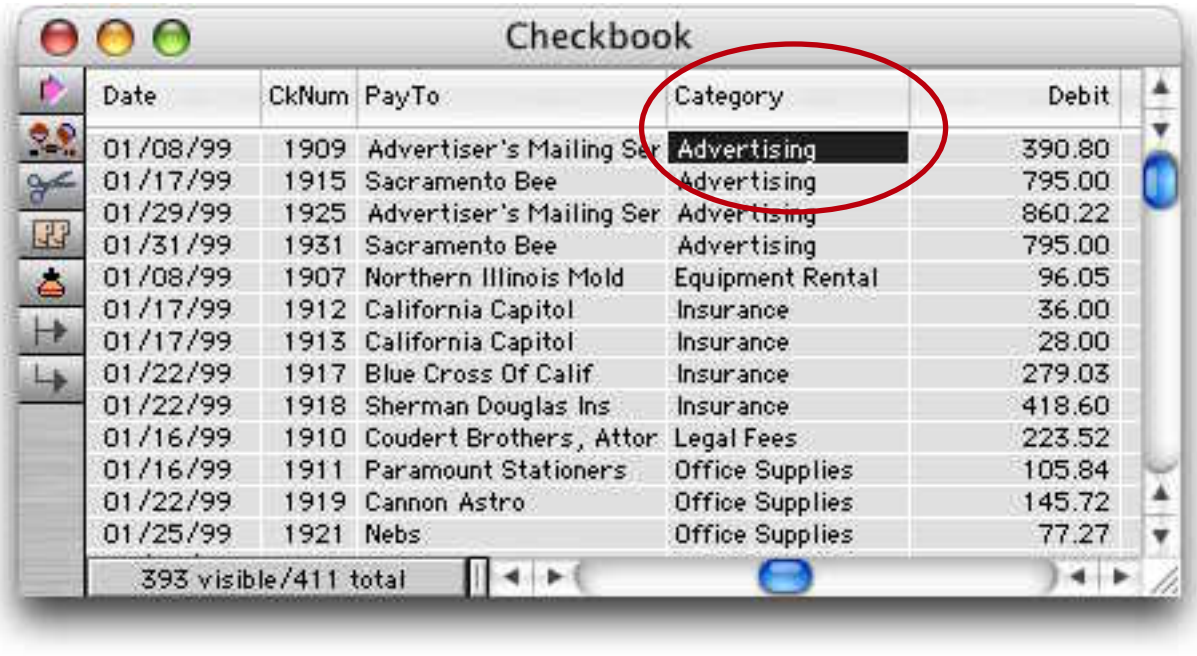


On to step number 2.

```
field Category
```

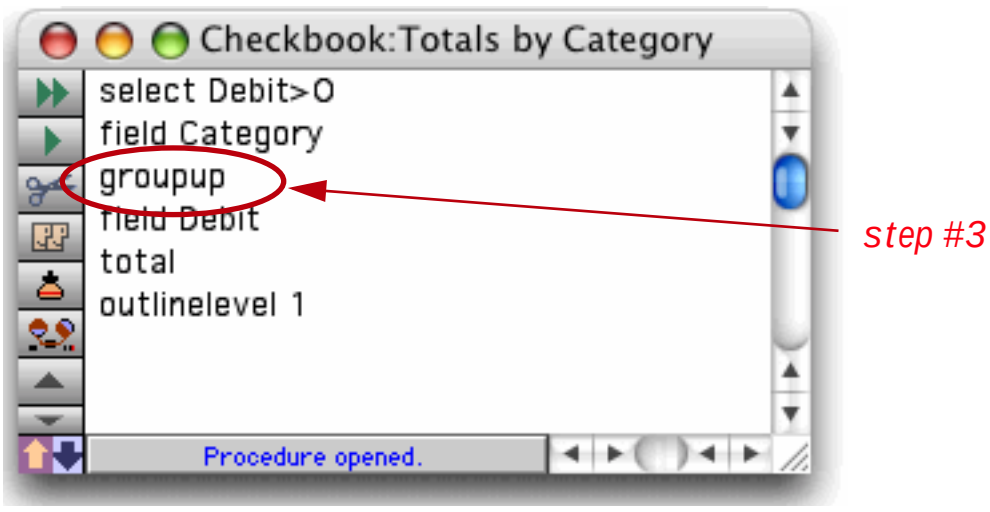


Many Panorama operations require you to click on a field before you perform an operation. For example, to sort or group by a particular column you would first click anywhere in the column. In a procedure this is accomplished with the `field` statement. If you watch quickly you'll see the cursor jump over to the Category column as Panorama performs this step.

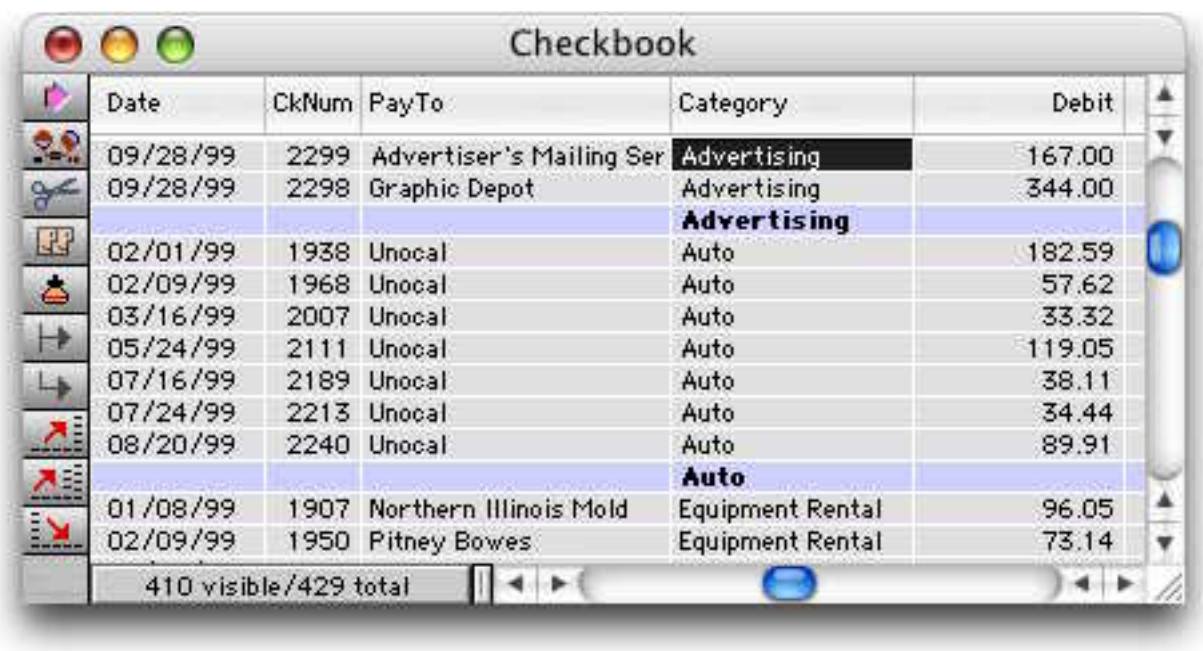


Now Panorama is ready for step number 3.

```
groupup
```

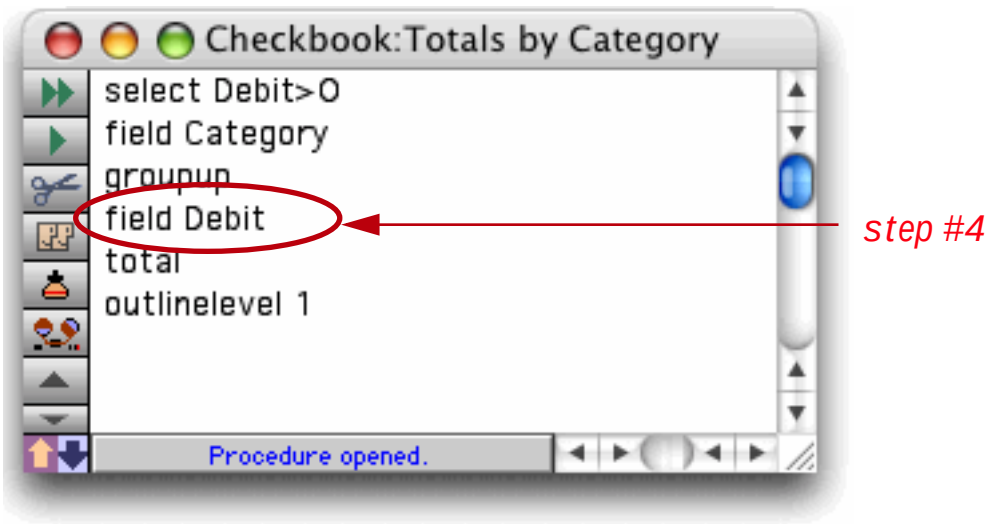


This statement tells Panorama to group the database, just as if you had selected Group Up from the Math menu (see “STEP 1 - GROUP” on page 399).

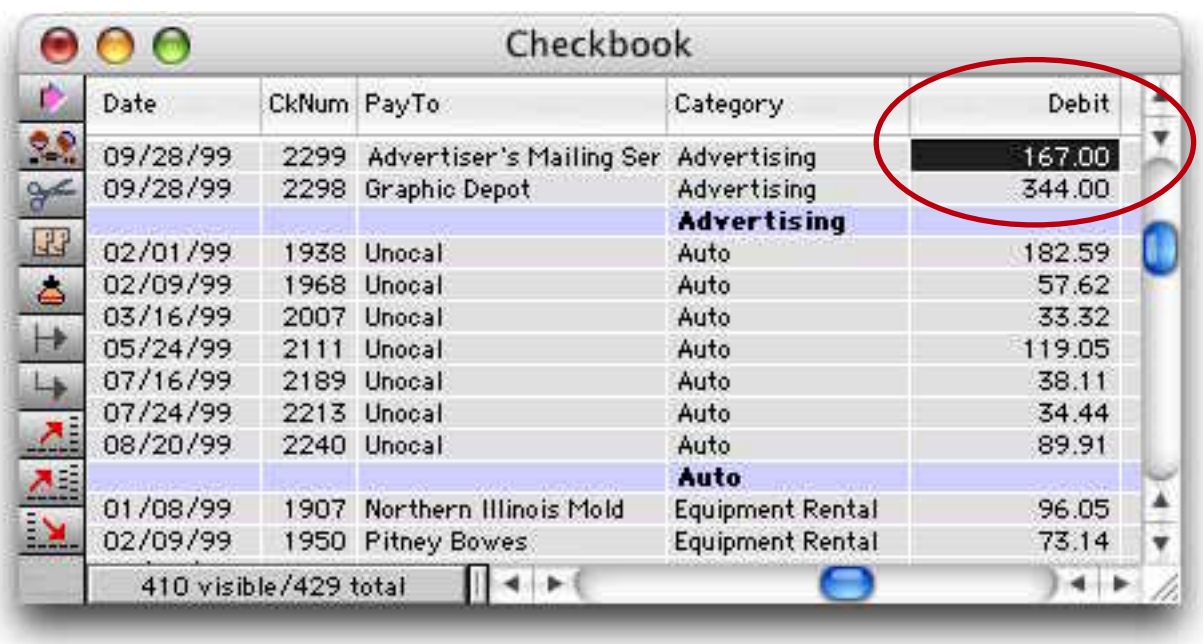


On to step number 4.

```
field Debit
```

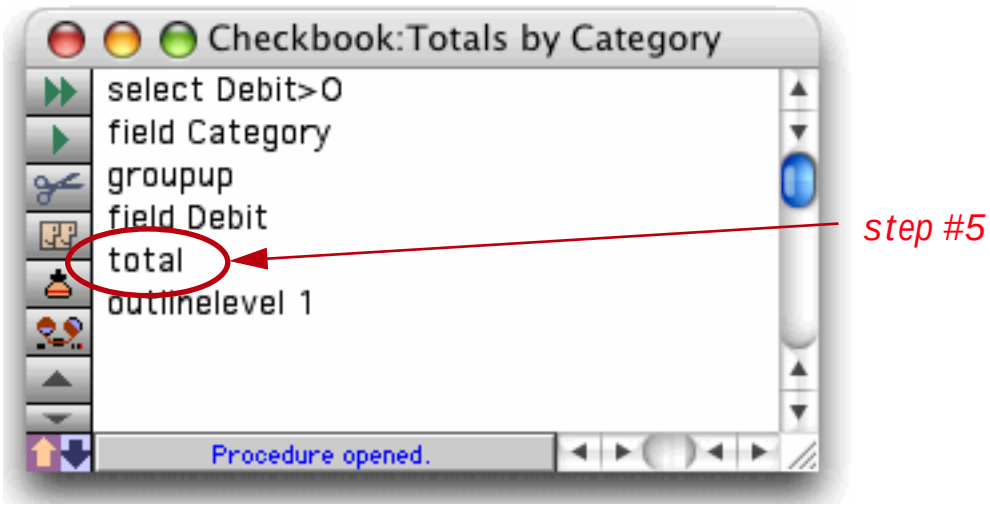


If we were performing this sequence of steps manually we would now click on the Debit column.

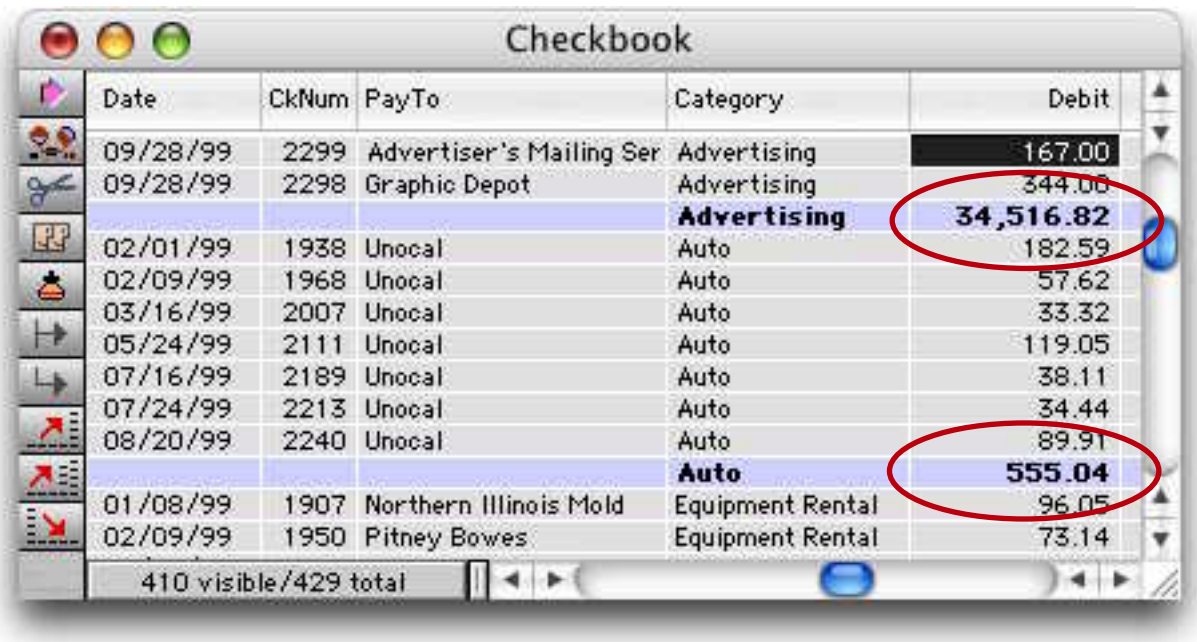


Next is step number 5.

```
total
```

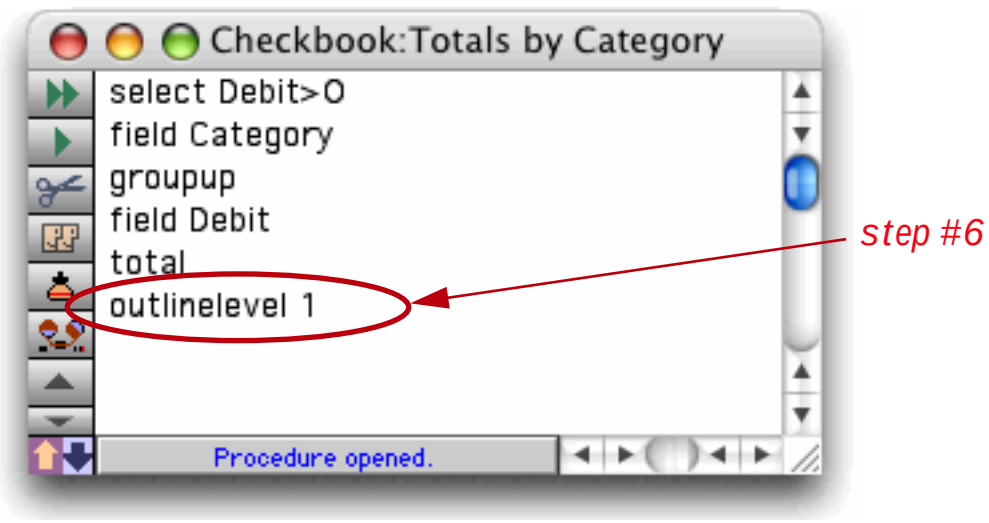


This statement is the same as choosing the **Total** command from the Math menu (see "Total" on page 403).

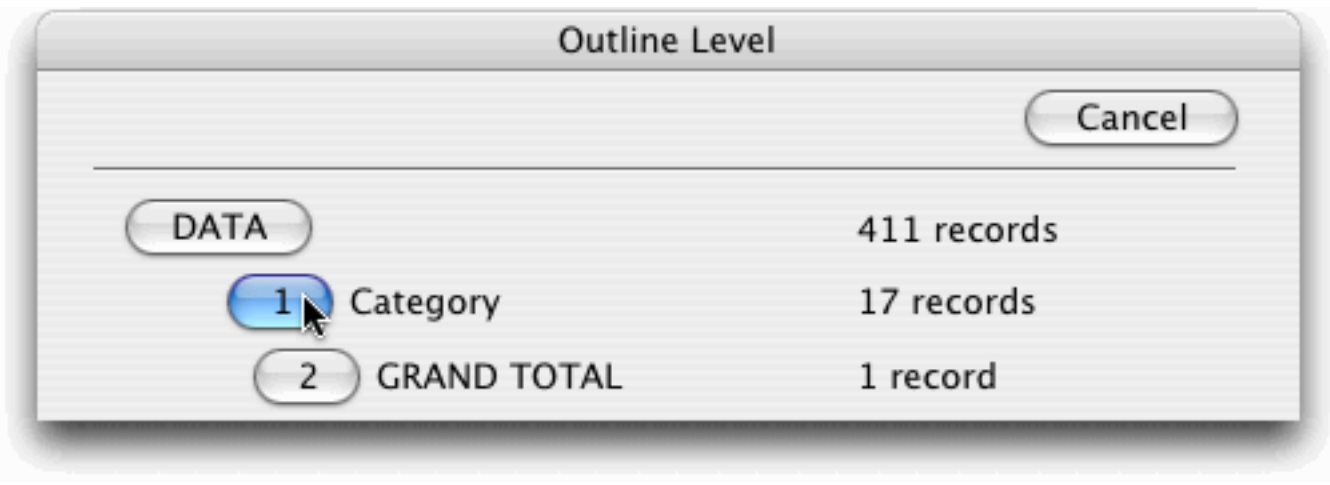


And now for the final step, number 6.

```
outlinelevel 1
```



This statement is the same as choosing the **Outline Level** command from the Sort menu (see "The Outline Level" on page 409). The parameter on this statement, 1, tells Panorama to simulate pressing the **1** button in the dialog. (Once again, since Panorama already knows what to do it doesn't actually display the dialog.)



Here's the final result after all six statements have completed.

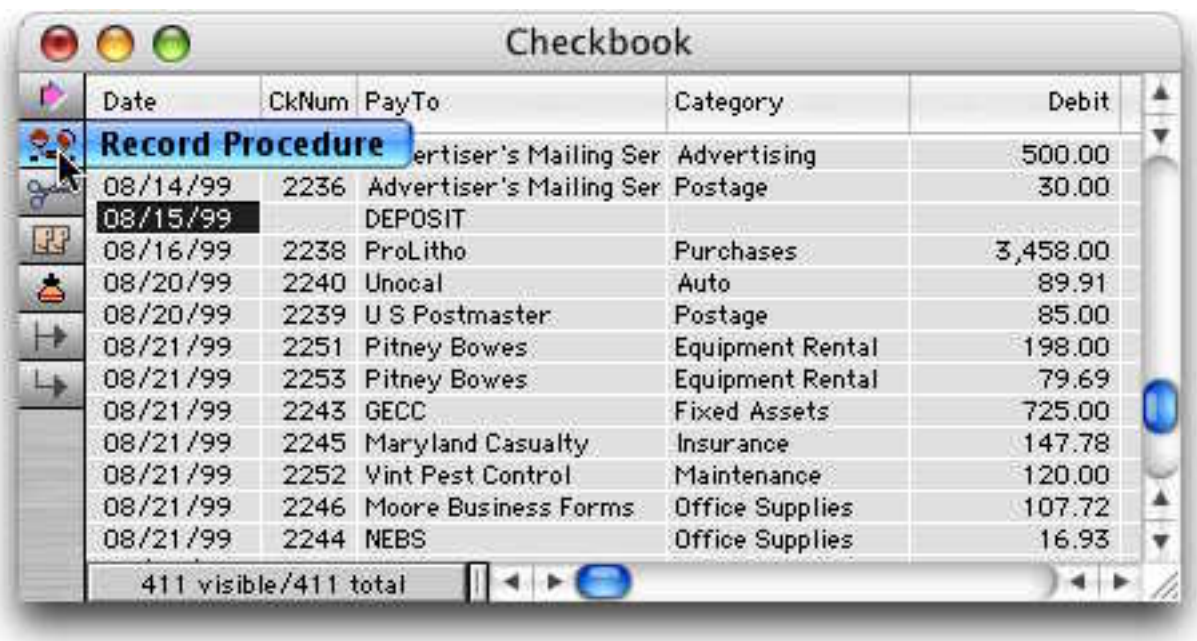


Pretty simple, is it not? This is the basic operation of any procedure — first the procedure is triggered, then Panorama performs each statement from top to bottom. (Later you'll learn several ways to change the top to bottom order when it is necessary, see "Control Flow" on page 1378.)

Creating a Procedure with the Recorder

A basic procedure like the one in the last section is very easy to create with Panorama's built in procedure recorder. The procedure recorder is like a tape recorder that records your actions as you work. Recording a procedure is a four step process— 1) start the recorder, 2) perform the steps while Panorama records, 3) stop the recorder, and 4) give the new procedure a name.

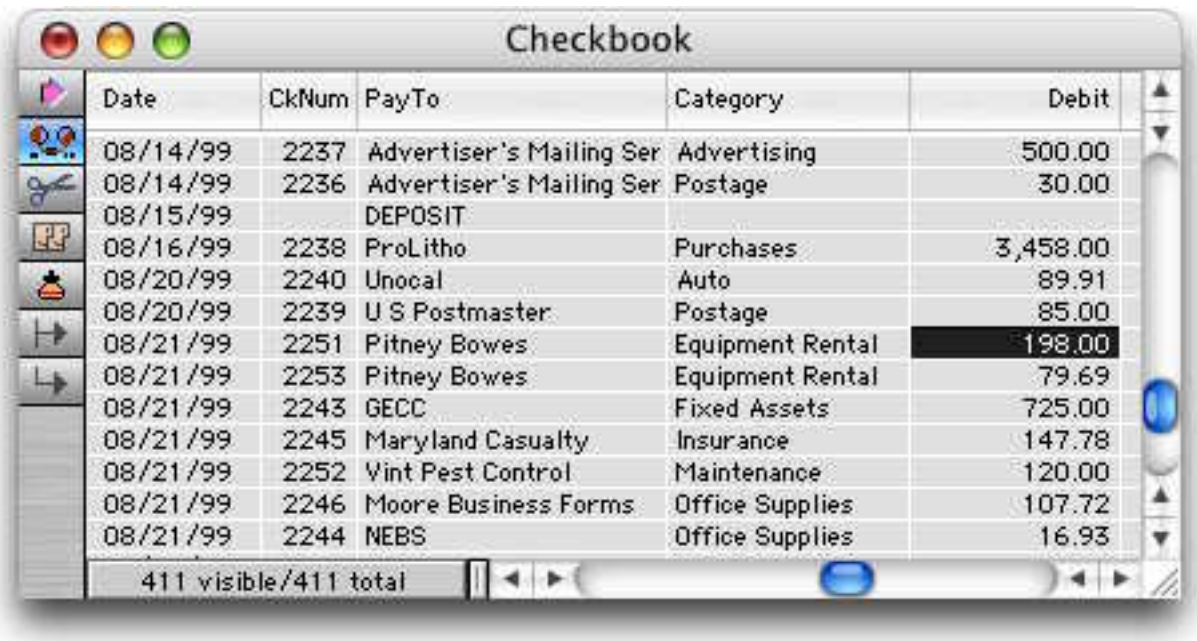
To start the procedure recorder, click the **Record Procedure** tool (available in the Data Sheet and Form tool palettes (unless you are in Graphics Mode)).



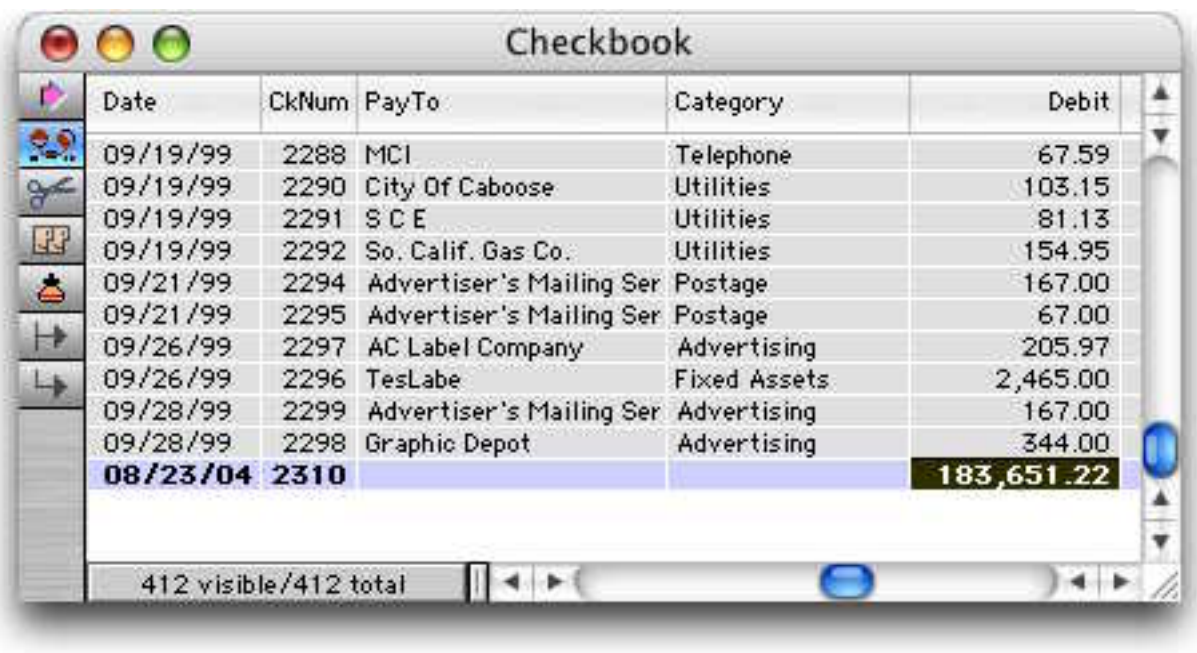
The "reels" on the recorder tool will begin to spin. This lets you know that Panorama is recording your actions.



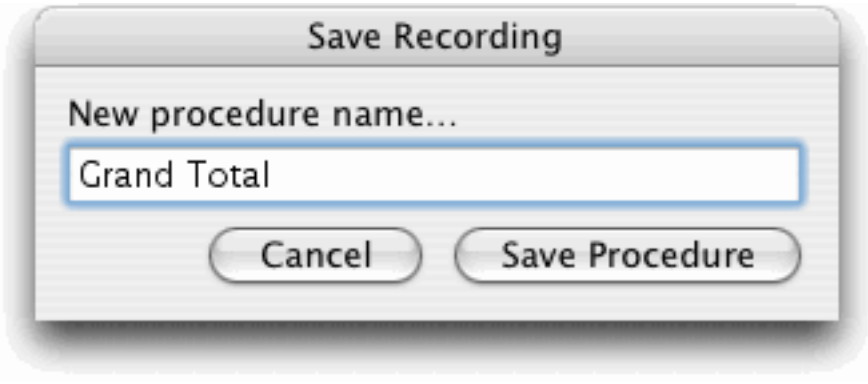
Once the recorder is running just continue to use Panorama normally. Panorama will record every menu command or tool that you use. To demonstrate the recorder we'll create a short procedure that calculates the grand total. Start by clicking anywhere in the Debit field.



Next choose **Total** from the Math menu. Panorama calculates the total.



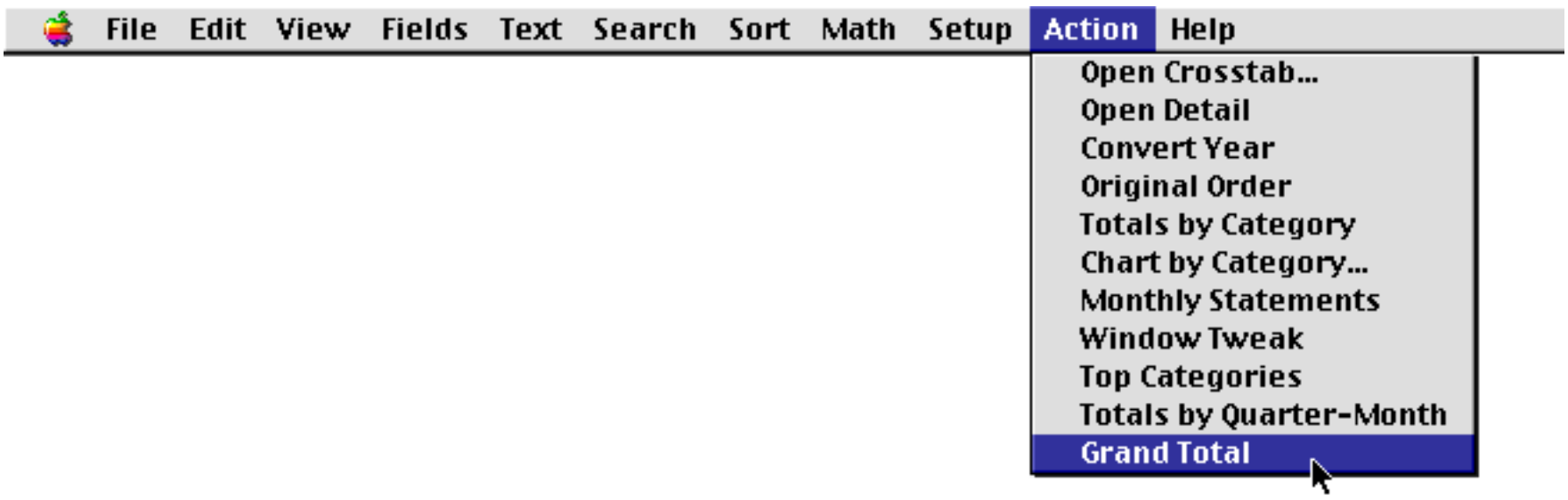
Our simple procedure is complete. To stop the recorder, click on the **Record Procedure** tool again. The "reels" will rewind and stop and a dialog box appears. This dialog box allows you to give your new procedure a name, up to 25 characters. Pick a name that will help you remember what the procedure does, for example Print Invoices or Balance Checkbook.



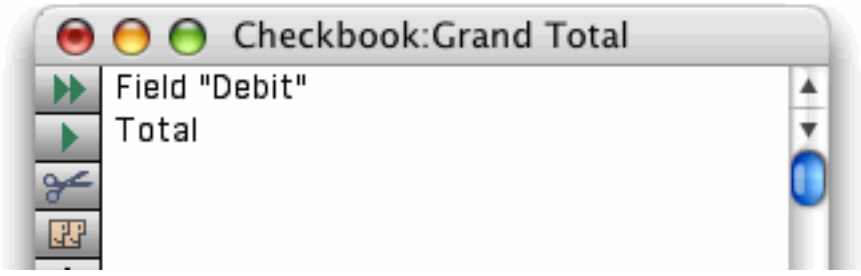
Once you have entered the name, press **Save Procedure**.

Tip: There are five characters you should not use in a procedure name unless you know what you are doing. The five characters are: ^ ; < (/. Later in this manual you'll learn how these characters can be used to create special effects in the **Action** Menu (see "Action Menu Options" on page 1459).

Once you have given the recording a name, the new procedure is added to the end of the **Action** Menu. (If the database doesn't already have a **Action** Menu, it will be created.)



You can play back your new procedure at any time by selecting it from the **Action** Menu. To view the statements in the new procedure you can open it with the View menu (hold down the **Control** key (Mac) or **Alt** key (PC) to open the procedure in it's own new window.)



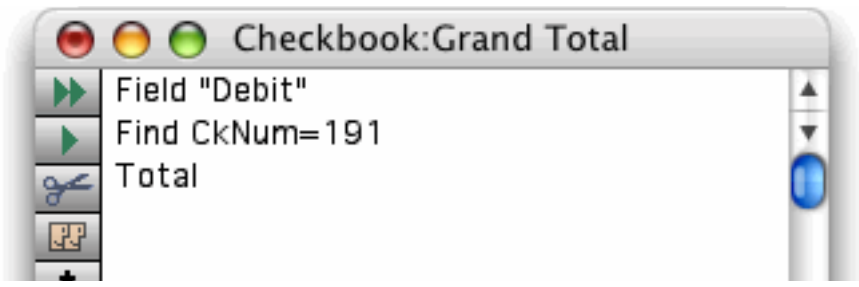
Our simple recording contains two steps. You can use the procedure "as is" or you can customize it further.

Recording Mouse Clicks

Panorama does not normally consider clicking the mouse to be a step—clicking the mouse is not a menu command or tool. However, if you click on a different window or a different field within the current window, Panorama will record that step. The recorder will ignore all other mouse clicks (for example, clicking on the scroll bar, dragging a window to a new position or changing the size of a window, etc.).

Panorama never records the row position of a click. If you look back at the previous section you'll notice that I clicked on check number 1915 when I recorded the procedure. However, this information was not recorded. When the procedure is played back Panorama will not move to check number 1915, but will stay on whatever check it is already on. The `total` statement doesn't care as long as the column (field) is correct.

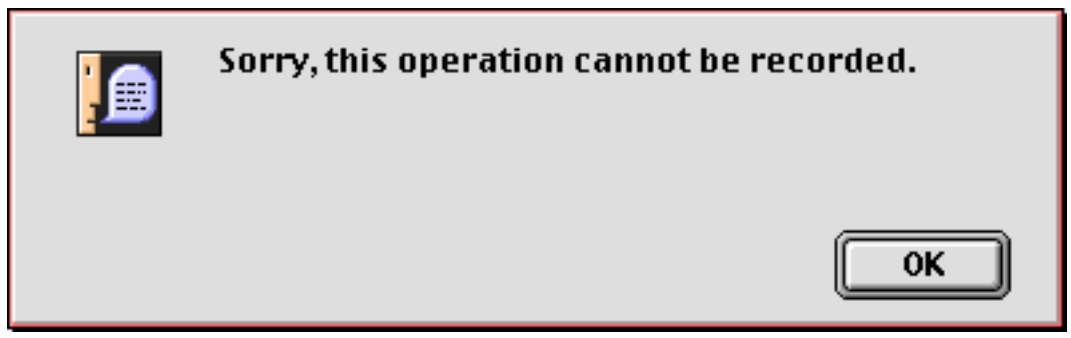
If you do want Panorama to move to a specific record you'll need to use the `find` statement (see "FIND" on page 5245). You can either create this during recording using the Find / Select dialog (see "The Find / Select Dialog" on page 369) or simply by typing it into the procedure window. Here's a modified version of the procedure that moves to check number 1915.



Of course this example doesn't make much sense because Panorama will only be on check number 1915 for a fraction of a second before the `total` statement makes Panorama jump to the end of the database.

Non Recordable Menus and Tools

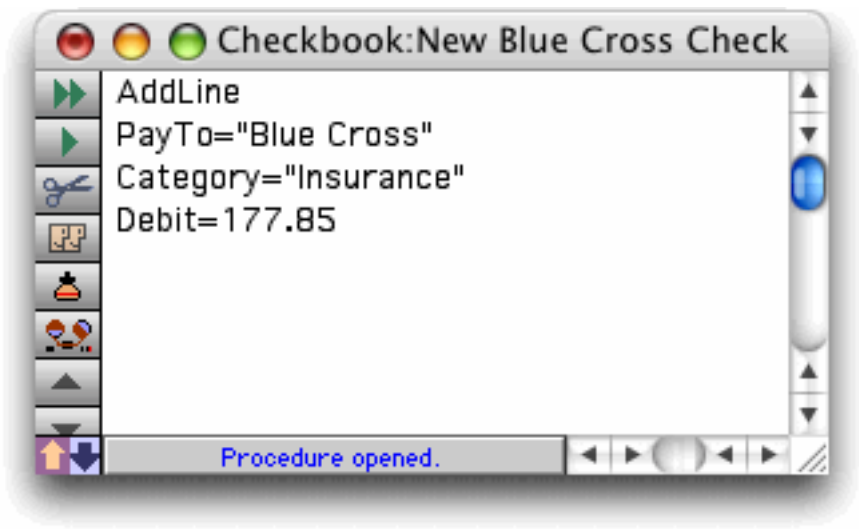
A couple of pages ago we told you that Panorama records every menu and tool when the recorder is on. Sorry, but that was a lie. Some menus and tools are not recordable. Most commands and tools that work with graphics cannot be used. For example, **Sort Up**, **Insert Record**, and **Print** can be used as steps in a procedure, but **Bring to Front**, **Oval**, and **Align** cannot. In general, only actions that affect data can be included in a procedure. If the recorder is on and you attempt to use a menu command or tool that cannot be used as a step in a procedure, Panorama will alert you.



As long as you stick with the Data Sheet and Data Access Mode Forms you'll usually be fine.

Recording Data Entry

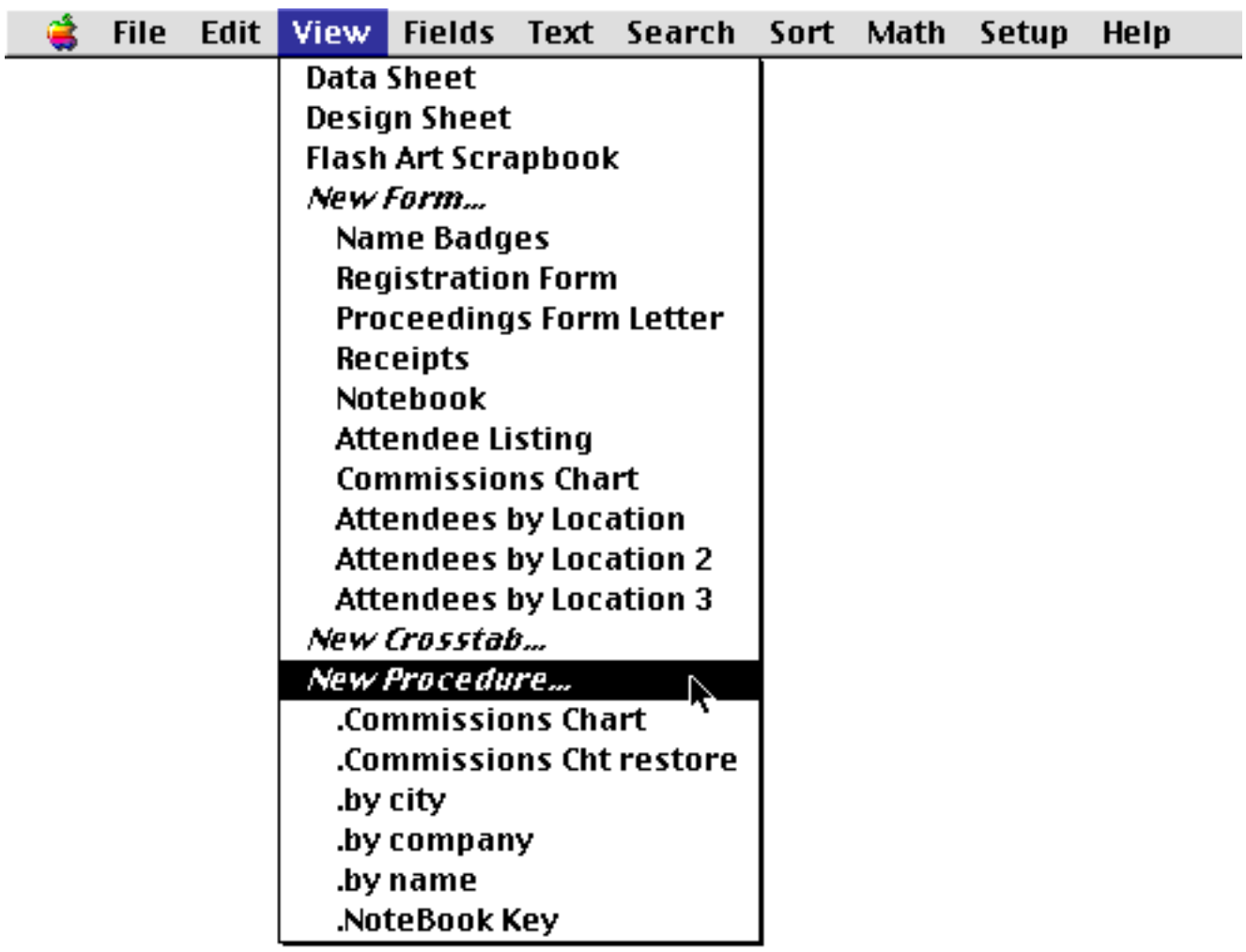
Panorama doesn't do a very good job of recording data entry. Instead of using the recorder we recommend that you use assignment statements to perform data entry, as shown in this example.



You cannot record these assignment statements, you have to type them in manually. See "Assignment Statements" on page 1368 for more information on assignment statements.

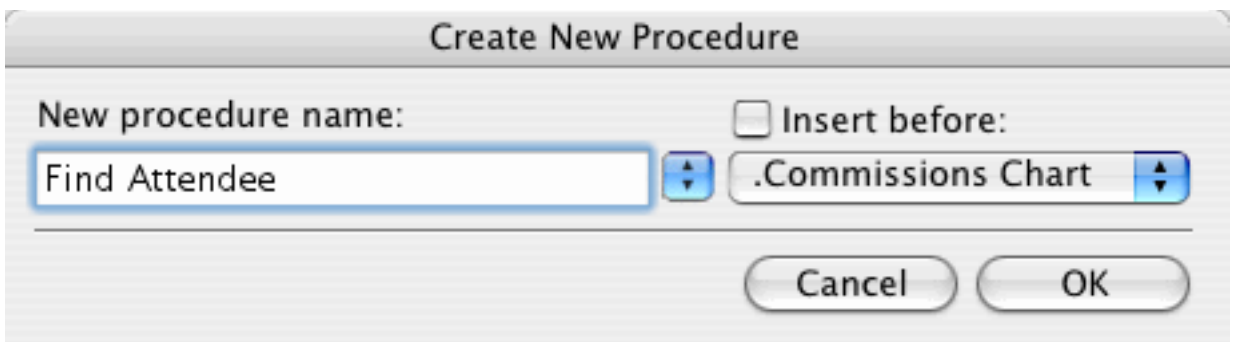
Writing a Procedure from Scratch

If you want to write a procedure from scratch (instead of using the recorder), the first step is to create a new, empty procedure. To do this select **New Procedure** from the View menu.



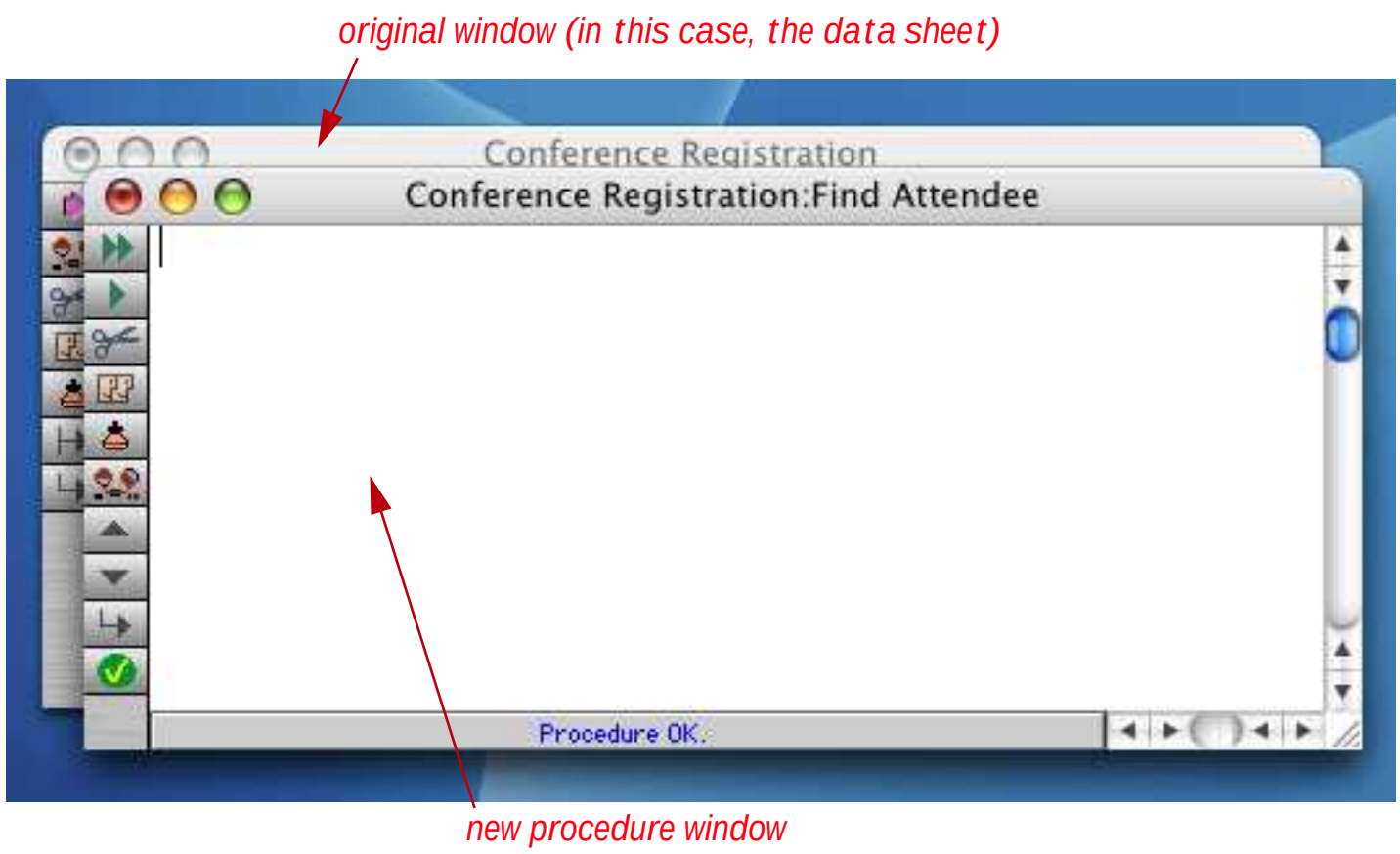
Selecting **New Procedure** normally creates the new procedure in the same window you are currently in. It's often convenient to create the new procedure in a new, separate window, leaving the original window open. That way you can easily flip back and forth between your database window (data sheet or form) and the procedure window. To open the new procedure in a separate window hold down the **Control** key (Macintosh) or **Alt** key (Windows) as you click on the **View** menu.

After you select **New Procedure** this dialog appears.



Type in the name of the new procedure, then press **OK**. The name may be up to 25 characters long, and must be unique within this database. Tip: There are five characters you should not use in a procedure name unless you know what you are doing. The five characters are: ^ ; < (/. Later in this manual you'll learn how these characters can be used to create special effects in the **Action** Menu (see "Action Menu Options" on page 1459).

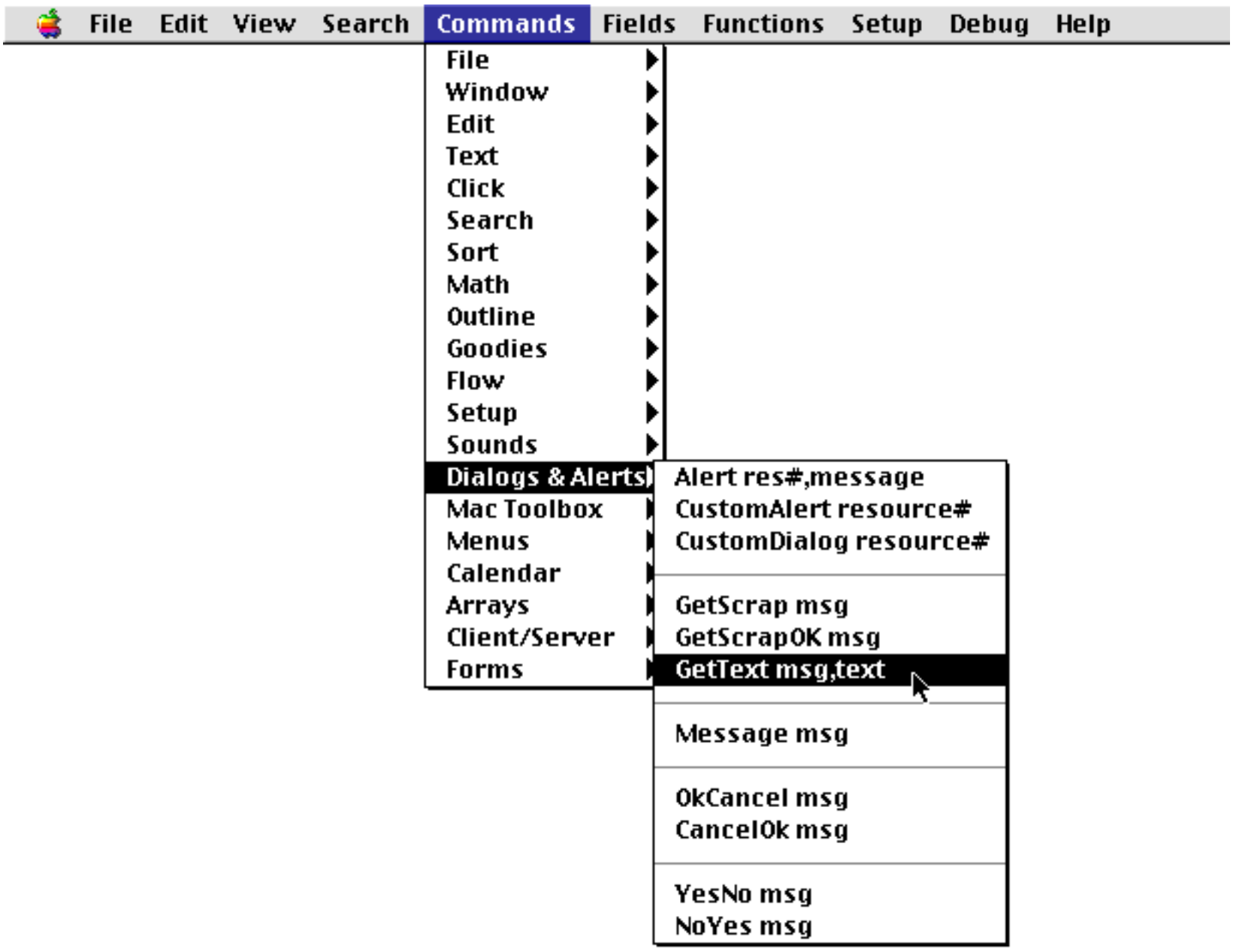
After you press the **OK** button a new, empty procedure is created, and the window switches to show you this new procedure. If you held down the **Control** key (Mac) or **Alt** key (Windows) the new procedure will open in a new window just below and to the right of the original window, like this.



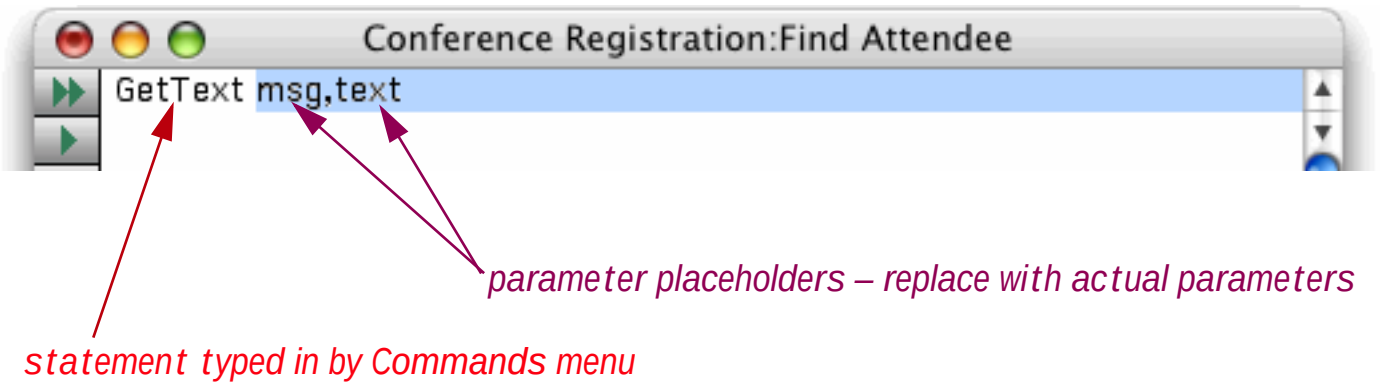
Writing Statements

Once you have created an empty procedure you can start adding statements to the procedure. If you know the keywords for the statements you want to use, just type them in. For example, if you want to sort the database in ascending order just type in the statement `SortUp`. By the way, the capitalization of statements doesn't matter, so you could also type `sortup`, `SORTUP`, or even `SoRTuP`. However, keywords are always a single word with no blanks, so `sort up` will not work.

If you don't know the exact keywords for the operations you want the procedure to perform, you have a several choices. You could look up the keyword in this manual, then type it in from the keyboard. Or you could locate the keyword in the **Commands** menu and let Panorama type in the keyword for you. (A third choice is to use the **Programming Reference** wizard, see "Programming Reference Wizard" on page 1361.)



You'll still have to type in any options and parameters yourself, but the **Commands** menu will type in placeholders for these values to help remind you that they are necessary (we'll talk more about options and parameters later).



Here is the finished statement with the actual parameters typed in.



Here is the completed procedure.



This procedure has three statements. The `local` statement creates a local variable named `whatname`. The `gettext` statement displays a dialog asking to enter a name. Whatever is typed in is placed in the `whatname` variable. The `find` statement searches the database to locate the requested name.

Trying Out a Procedure

The easiest way to try out a procedure you are working on is to press the **Run** button (you can also press **Command-R** on a Mac or **Control-R** on a PC).



When you run a procedure this way Panorama will automatically switch to a window that contains data. Any window that allows you to display and edit data will do (as long as it is in the same database), so Panorama will pick the topmost one that will work. If there is no window that will work (for example if the procedure is in the only window for this database), Panorama will display an error message and the procedure will not run.

In this case Panorama will switch to the data sheet window and begin performing the steps in the procedure, starting with the topmost statement.

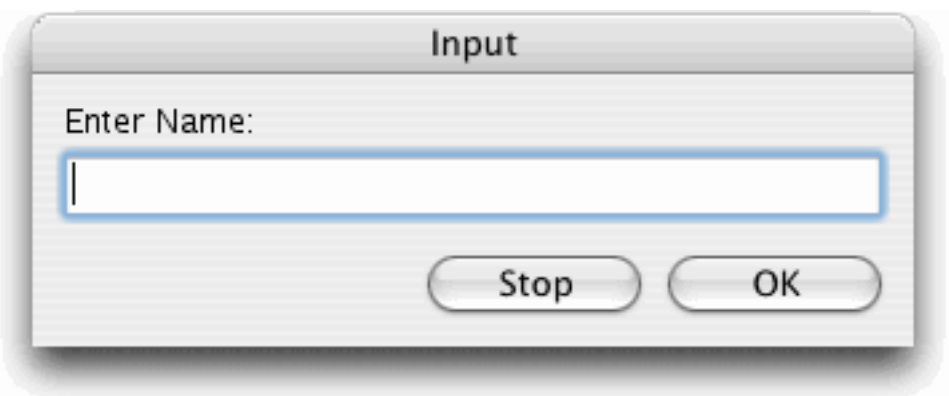
```
local whatname
```

This statement allocates a local variable named whatname for temporary storage (see “Variables” on page 1199). This operation is completely invisible.

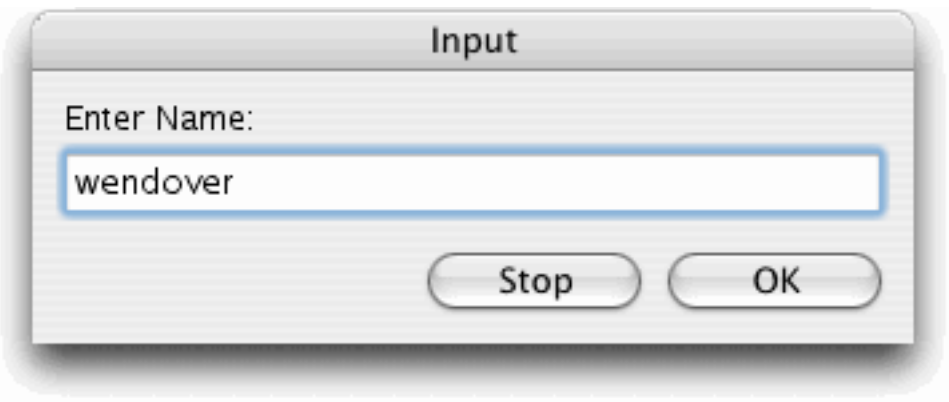
On to step 2 —

```
GetText "Enter Name:",whatname
```

This statement displays a dialog asking the user to type in some text.



Enter the name you want to search for.



When you press **OK** the dialog disappears and the procedure continues on with the next statement. Before it does, however, it copies the text that was typed in into the variable named whatname.

```
find «First Name»+" "+«Last Name»+" "+«Company Name» contains whatname
```

This final statement searches three fields in the database to see if they contain whatever text was typed in (see "Comparison Operators" on page 1265). This database does contain the name wendover, so Panorama jumps to that record. The name is circled in the illustration below to make it more clear.



Now you have an easy way to locate any person in this database without having to bother with the **Find/Select** dialog.

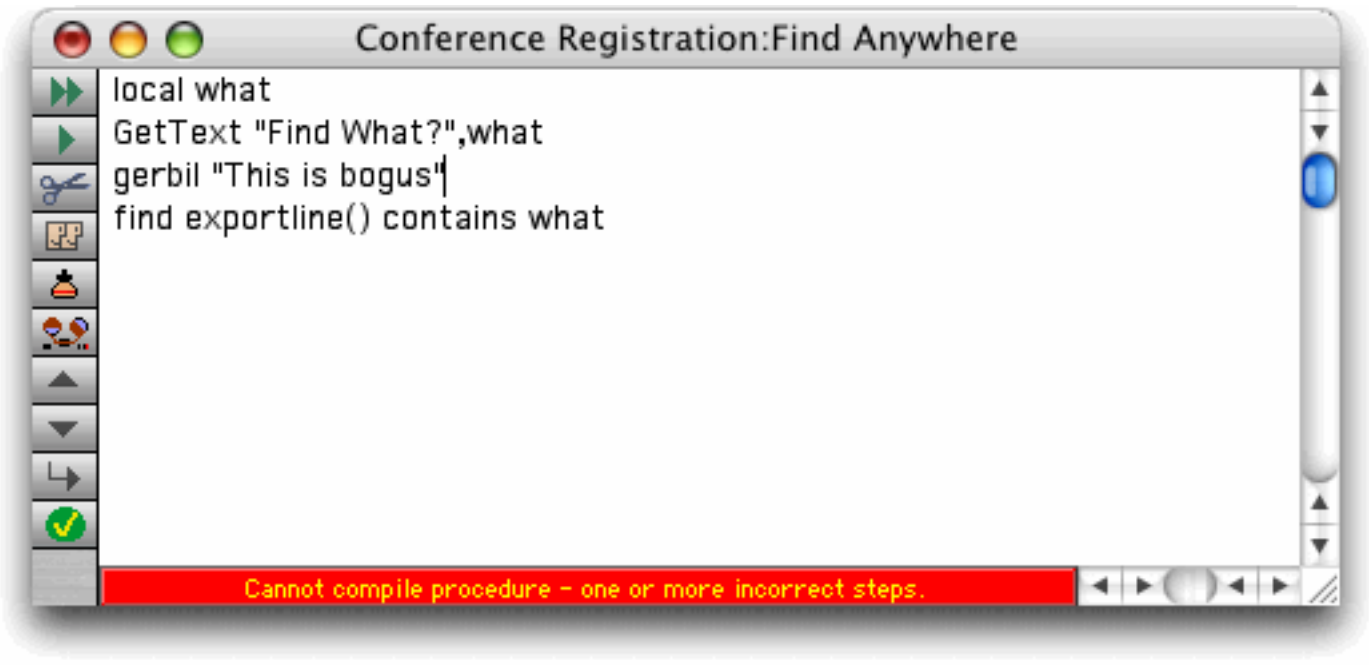
Checking for Mistakes

If you write a procedure yourself without using the recorder, you may make a mistake. You might misspell a keyword, forget to include a parameter, leave off a closing parenthesis, etc. Panorama will not let you use a procedure until you find and fix all of these mistakes.

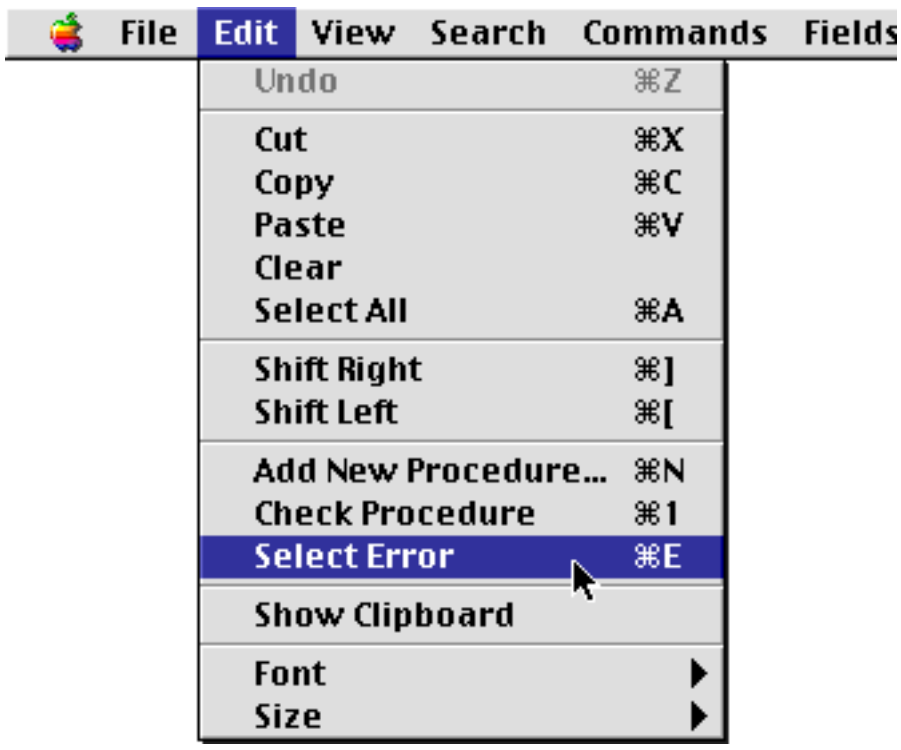
To help you find these mistakes Panorama provides the **Check Procedure** tool or menu command (Edit menu).



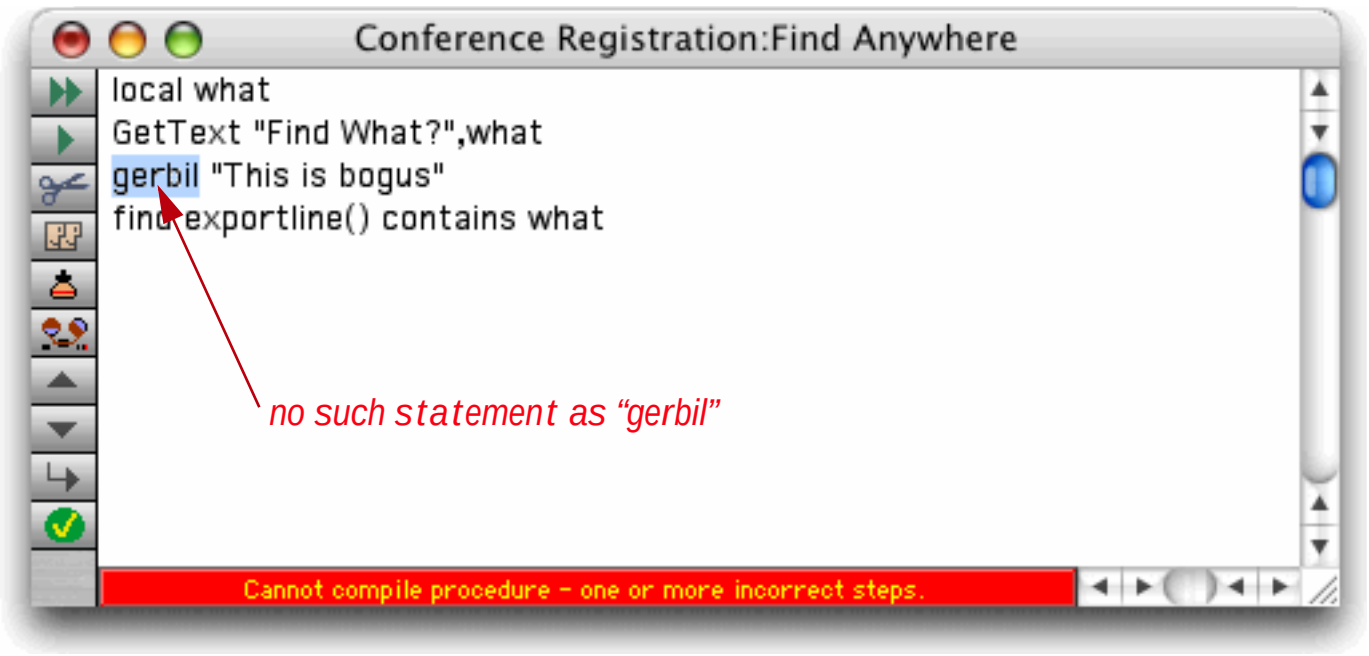
This tool scans the procedure looking for misspelled keywords and other mistakes. If it finds an error the status bar turns red and contains a description of the error.



Choose **Select Error** from the Edit menu if you would like Panorama to attempt to highlight the location of the error for you.



Panorama will identify the spot where it thinks the error occurred (see “Mysterious Errors” on page 1359).



Correct the error, then use the **Check Procedure** tool again to see if there are any more mistakes. Repeat this process until the **Check Procedure** tool no longer finds any mistakes in your procedure. (Note: Panorama also automatically checks your procedure whenever you click on another window, save the database, switch to another view, or close the window containing the procedure.)

Mysterious Errors

Usually the **Check Procedure** tool in combination with **Select Error** is able to pinpoint the exact spot where the mistake in your procedure is located. Some types of mistakes, however, are not detectable right away, so Panorama actually will highlight the wrong spot in the procedure. Usually this is caused by a missing `endif` statement, mismatched parentheses, or mismatched quotes. If Panorama tells you that there is an error but the spot highlighted looks ok to you, check carefully above the spot where the error was flagged. The actual error may be many lines above the spot Panorama has flagged. If you still can’t find the error, try splitting the procedure into several smaller procedures and checking each piece separately until you find the section containing the error.

Closing the Window When a Procedure is Finished

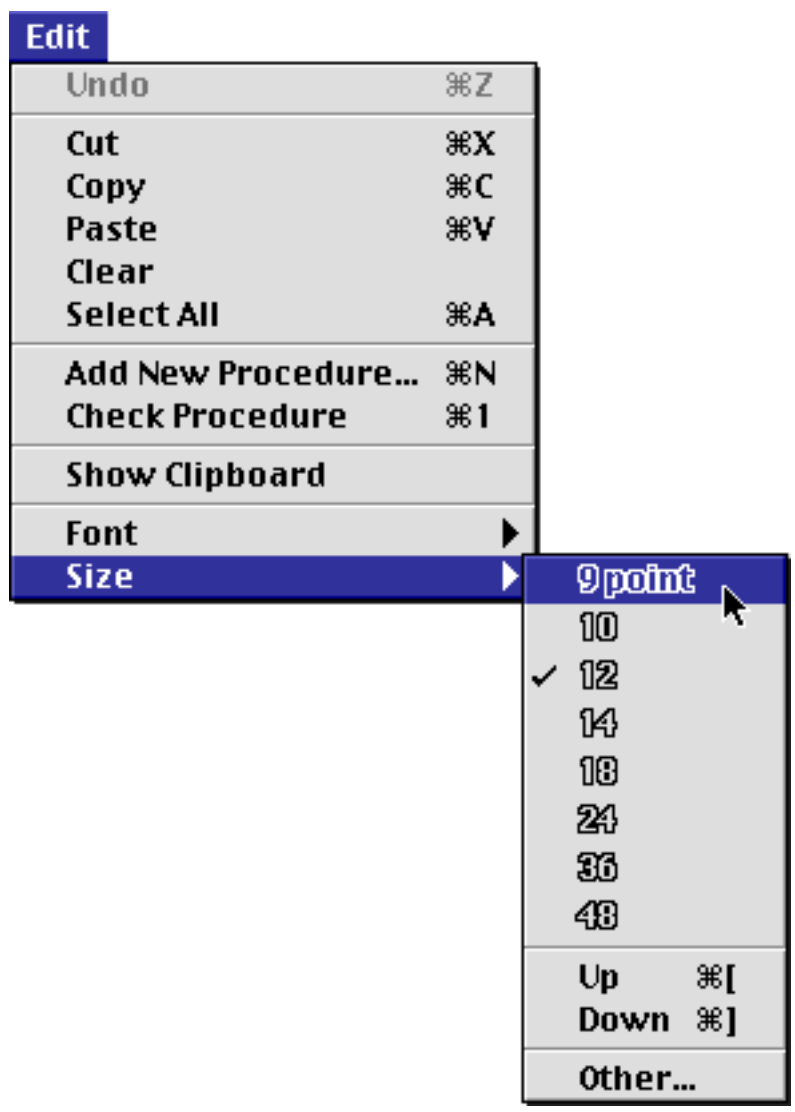
When you have finished writing a procedure you’ll probably want to close the window for that procedure. If it is in a separate window, just click on the close box. If it is the only window for the database use the **View** Menu to flip to another view. You don’t have to close the procedure window to use the procedure, but when you are sure it is working properly you will probably want to close it just to cut down on window clutter.

Re-Opening a Procedure

You can change any procedure at any time. Simply open the procedure with the **View** Menu, then make the changes. Don’t forget that you can hold down the **Control** key (Macintosh) or **Alt** key (Windows) to make the procedure open into it’s own separate window.

Font and Size

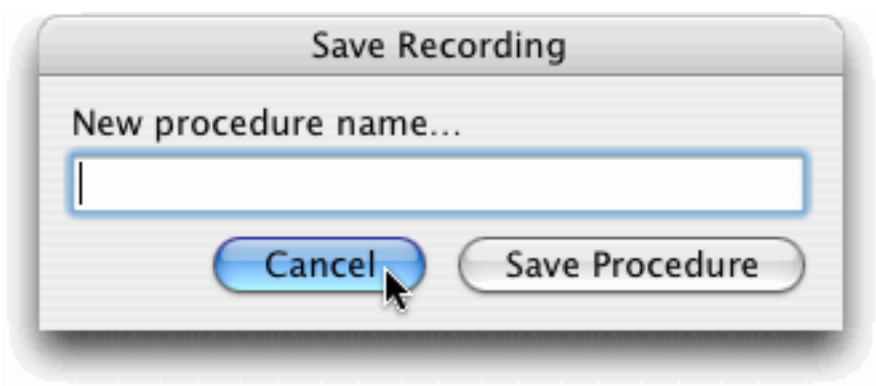
The **Font** and **Size** submenus (Edit Menu) allow you to change font and size of the procedures in this database. The font and size are always the same for every procedure in the database—you cannot set different procedures in the same database to different fonts or sizes.)



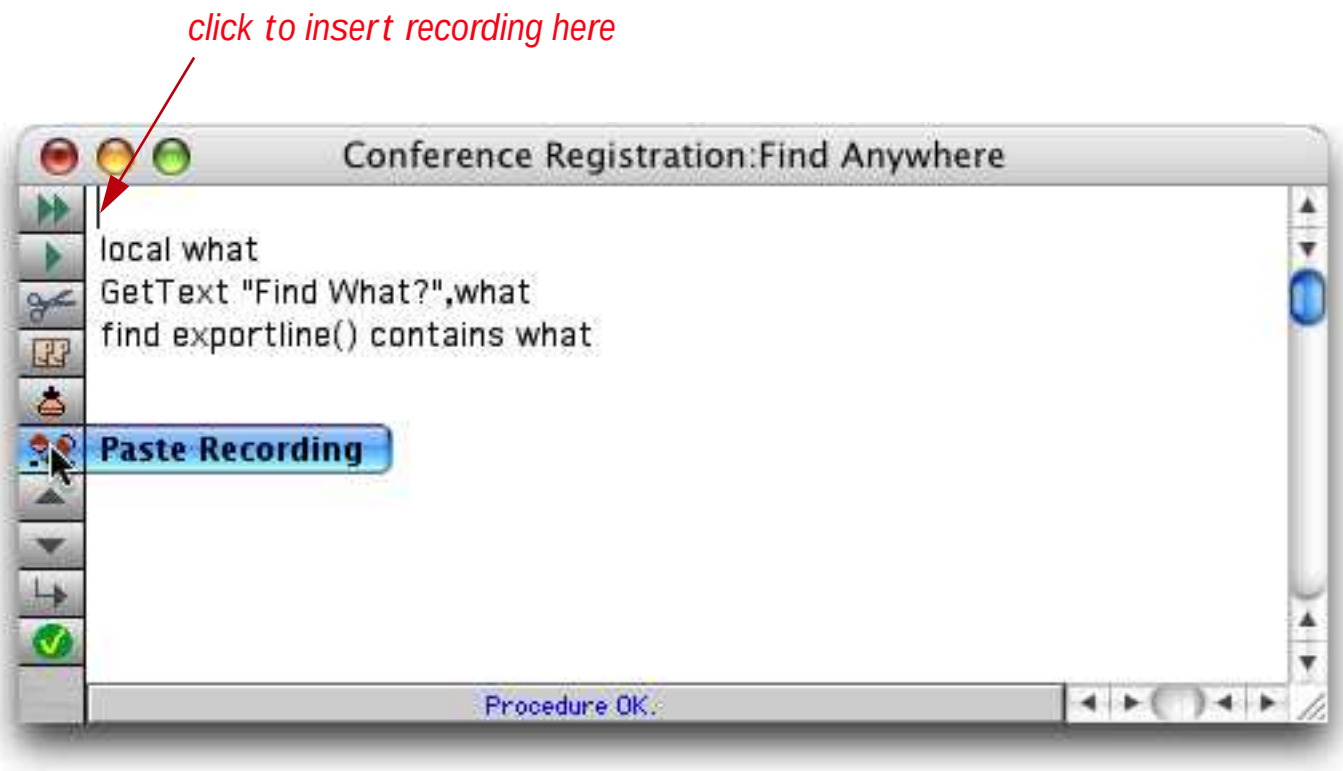
If you are using a Windows system we recommend that you stick with one of the four fonts installed with Panorama: Alpine, Block, City or Yankee (the default is Alpine 12). These fonts are designed to be able to display some of the special characters used by Panorama that are not normally available on Windows systems (≠, ≤, ≥ etc.)

Adding a Recording to an Existing Procedure

Earlier you learned how to create a new procedure by recording (see "Creating a Procedure with the Recorder" on page 1348). It's also possible to use the recorder to add statements to an existing procedure. To do this, start the recorder normally, and record the steps you want to include. When the steps have been completed click on the recorder again to stop the recording. When Panorama asks you what name you want to give the new procedure, click the **Cancel** button.



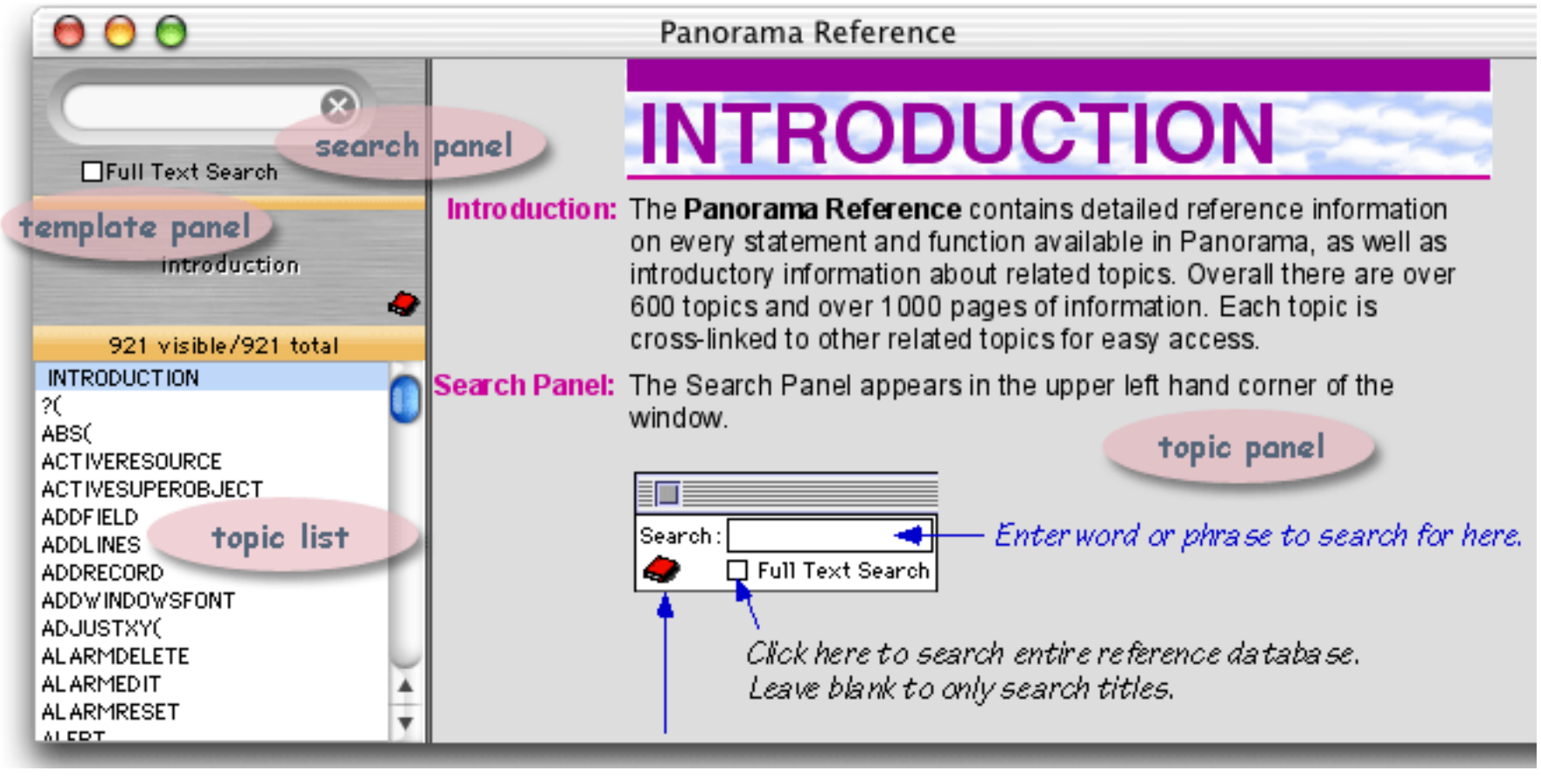
Now go to your procedure window and click on the spot where you want the recording to be inserted. Then choose the **Paste Recording** tool.



Panorama will insert the recording into the procedure. If necessary you can edit the recorded statements or use them “as is.”

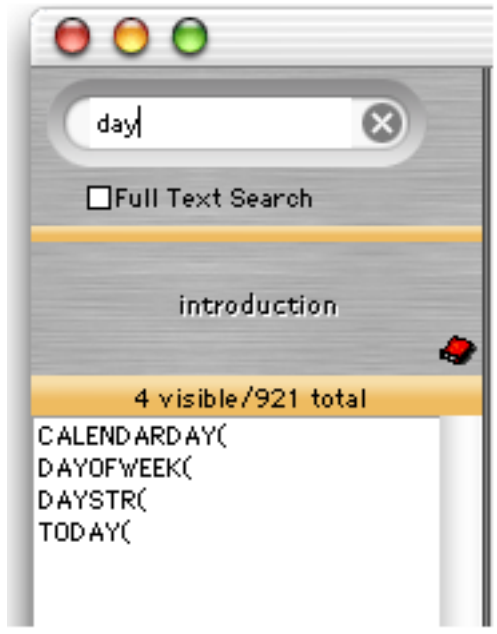
Programming Reference Wizard

Panorama has over one thousand functions and statements that you can use in formulas and procedures. Most of these have one or more parameters that must be used correctly. Even here at ProVUE Development we can't keep all of this memorized. To help, we created the **Programming Reference** wizard, an instant, online reference to what's what in Panorama formulas and programs. You can open this wizard from the Documentation submenu of the Wizard menu, or by pressing **Control-R** (Macintosh only). Once the wizard opens the reference window is divided into four sections: search panel, template panel, topic list and topic panel.

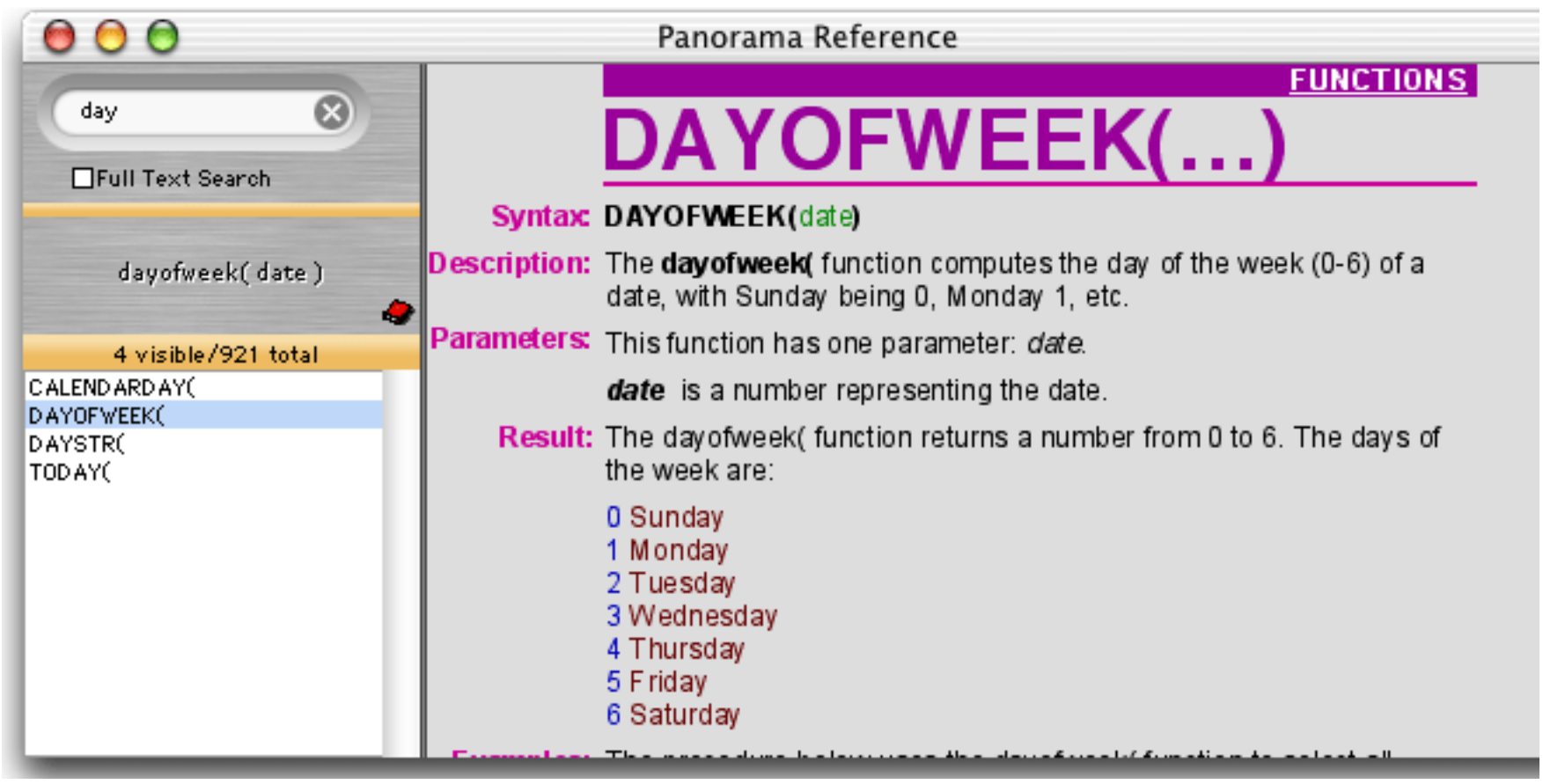


Navigation Using the Search Panel and Topic List

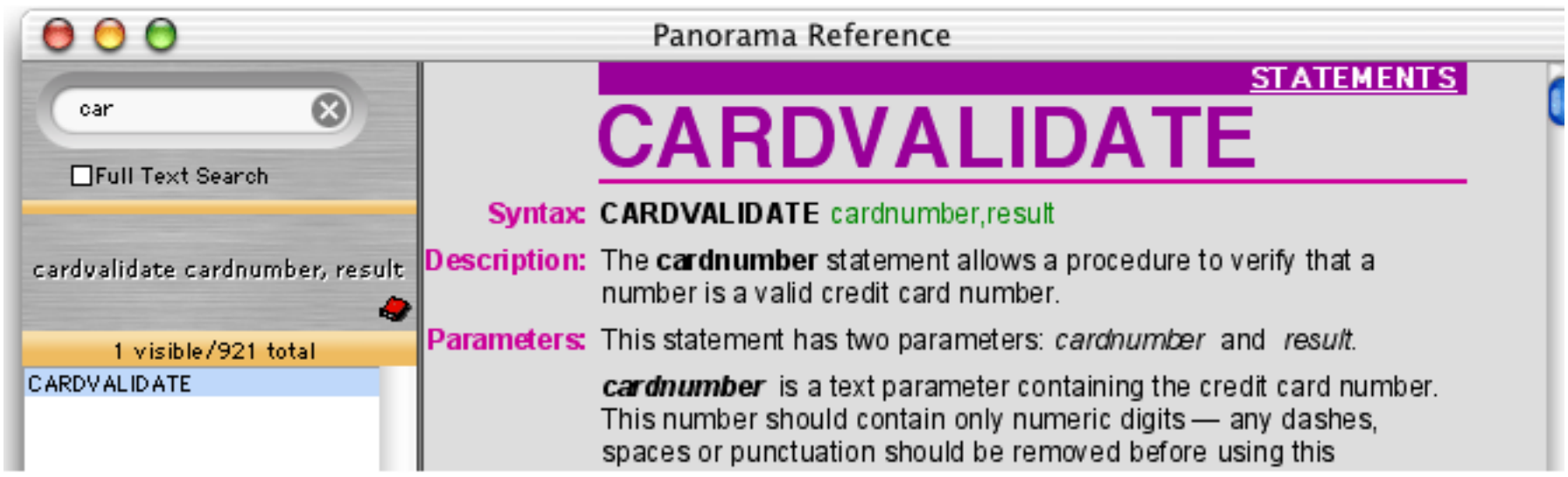
The search panel and topic list work together to help you locate a specific topic. As you type into the search panel, the topic list updates to show topics that match.



When you see the topic you want, click on it to display the topic in the topic panel.



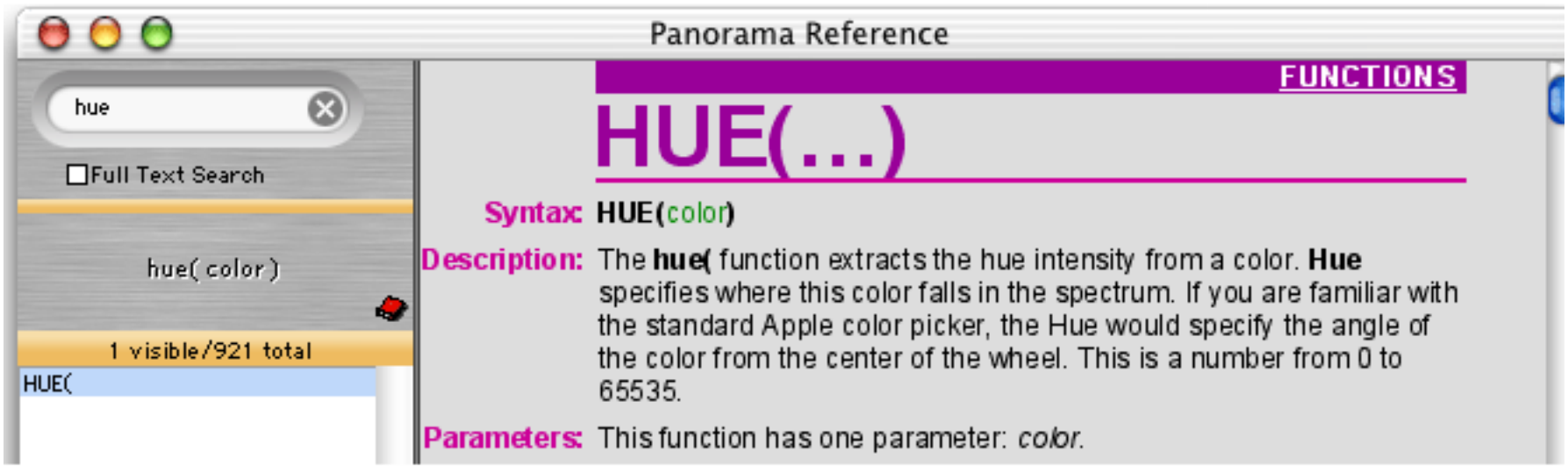
If there is only one topic in the topic list, the wizard will display the topic automatically (without having to click on the list).



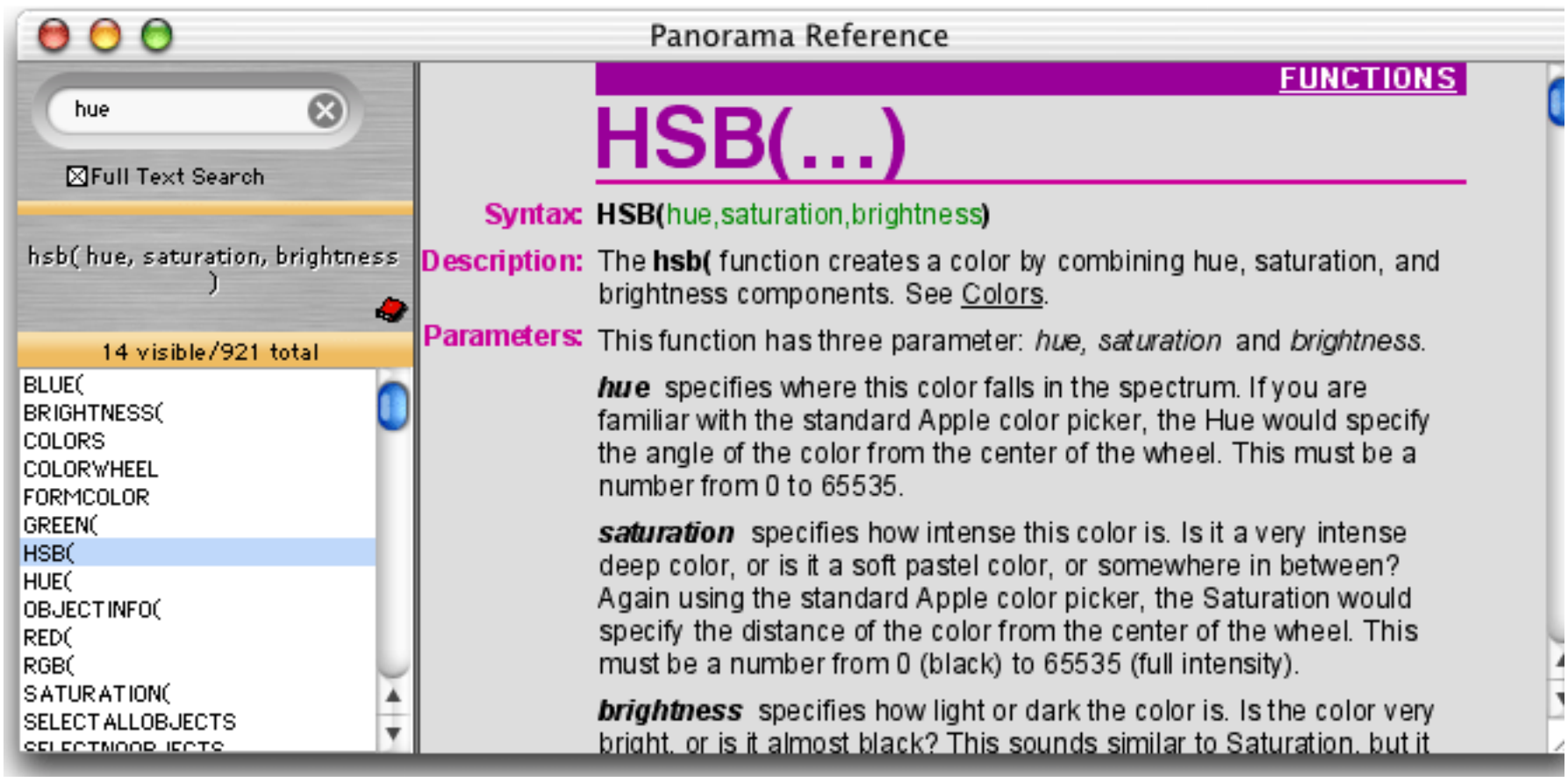
To quickly erase the query in the search panel, click on the button.

The Full Text Search Option

The search panel normally searches only the name and category of each topic. When the Full Text Search option is checked the wizard will also search the complete text of each topic. This makes it possible to quickly find every topic that references a particular function or statement, as well as the function or statement itself. For example, a normal search for the word hue will turn up only one match.

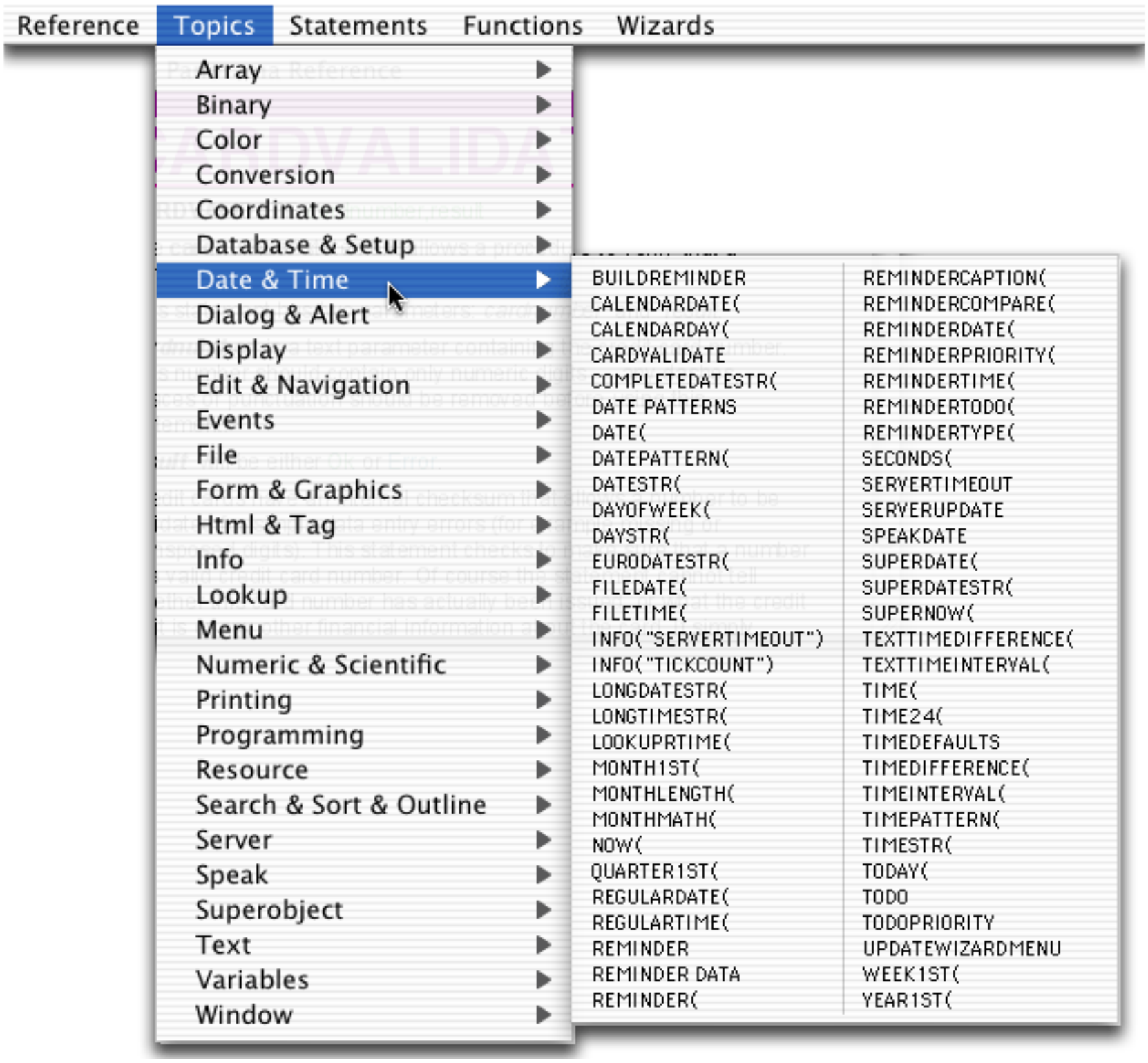


Repeating the search with the Full Text Search option turned on yields 14 matches. You can click on the match you are interested in.



Navigation Using the Topic, Statement and Function Menus

To jump directly to any topic use the **Topic**, **Statement** or **Function** menus. The Topic menu divides topics into about two dozen submenus. (Some topics may be display under more than one submenu, and some topics may not be listed under any topics.)



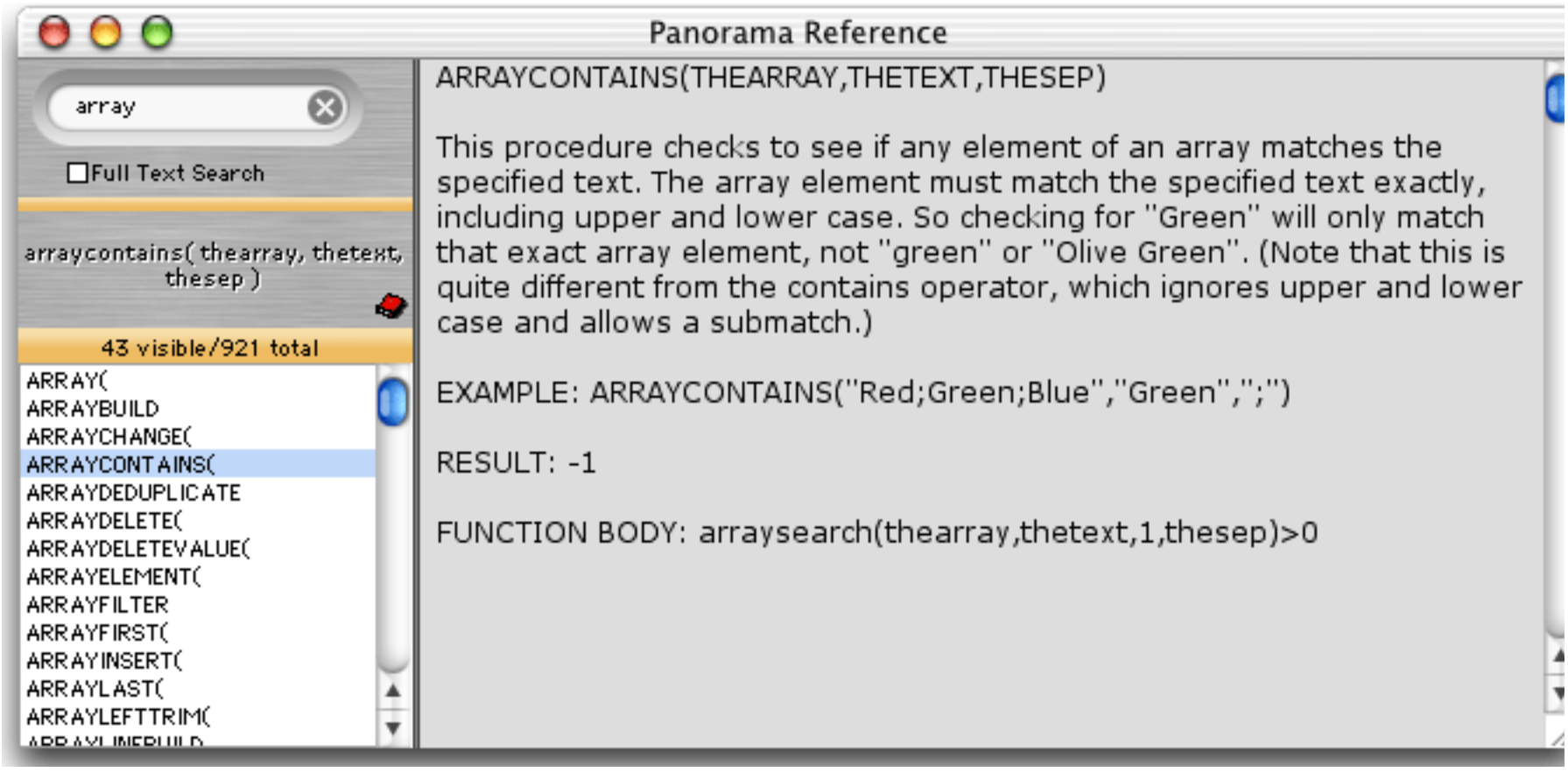
The **Statements** menu lists every statement in alphabetical order. The **Functions** menu lists every function in alphabetical order. Simply select a statement or function from one of these menus to jump to see the description of that topic in the topic panel.

Navigation Using HyperLinks

Like a web browser, the **Programming Reference** contains links from one page to other related topics. These links are underlined in the text. To jump any linked topic simply click on the underlined text.

Built In vs. Custom Statements and Functions

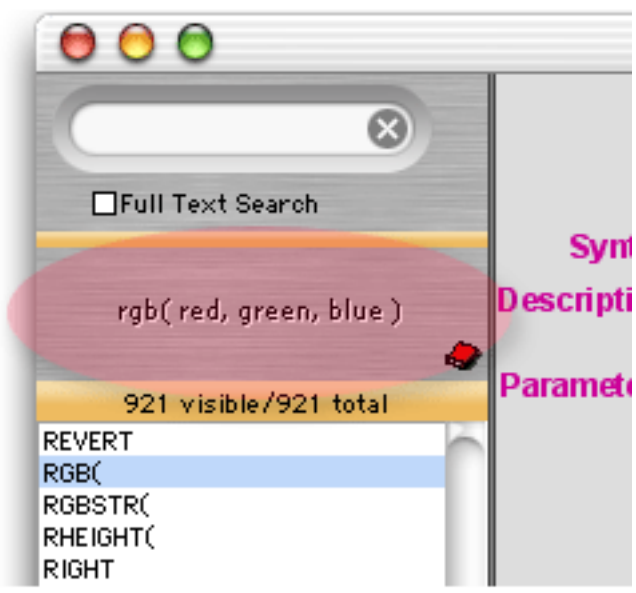
Panorama supports both built in and custom statements and functions. Several hundred custom statements and functions are included with Panorama, and these are also included as topics in the **Programming Reference** wizard. (You can also create your own custom statements and functions, but these are not included in the **Programming Reference** wizard.) For most custom statements and functions the topic panel uses a basic "plain text" format instead of the more graphical format used for built-in statements and functions.



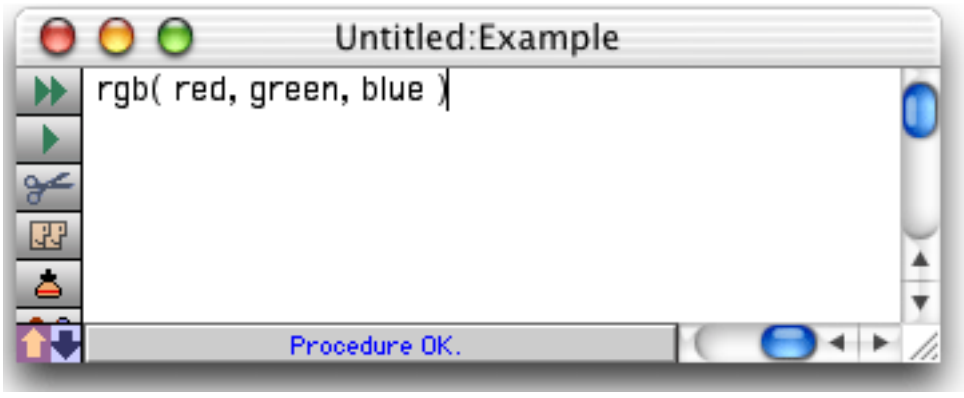
Don't adjust your set -- this plain text view is normal for custom statements and functions.

Using the Template Panel

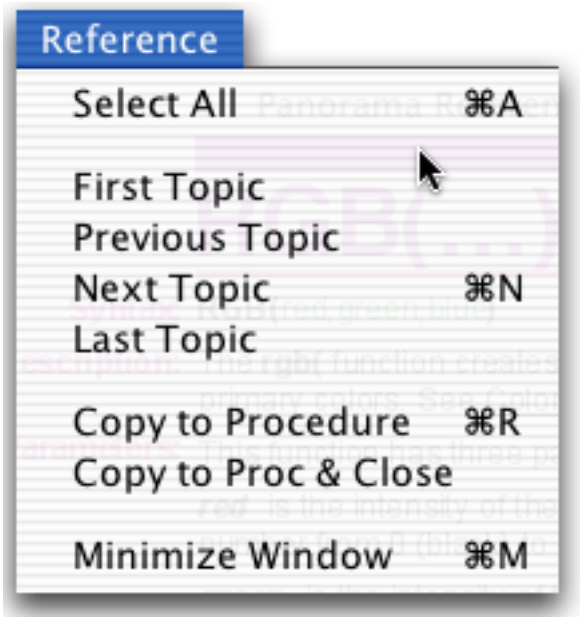
The template panel displays a sample that illustrates how this statement or function would be used in a formula or procedure. In this case the panel shows an example of the `rgb(` function, which has three parameters.



To copy the template into the topmost procedure window, hold down the **Control** key and click on the template panel. (If you are using a PC system you should right-click on the template panel.) The template will be pasted into the procedure at the current insertion point, and the procedure window will be brought to the front so that you can edit it further (for example, filling in the actual parameters).



You can also copy the template into the procedure window using the **Reference** menu.



The **Copy to Procedure** command copies the template into the procedure and brings the procedure window forward (exactly like control-clicking on the template panel). The **Copy to Proc & Close** command does the same, but also closes the **Programming Reference** wizard.

Minimizing the Programming Reference Wizard

In addition to its normal "wide-screen" view, the wizard can also be used in a minimized view. To minimize the wizard, either choose **Minimize Window** from the Reference menu or click on the button in the template panel. As shown here the minimized wizard hides the search panel, topic list, and topic display panel.



Although you can't search for topics when the window is minimized, you can still use the **Reference**, **Topics**, **Statements** and **Function** menus. When you want to maximize the window again choose either the **Maximize Window** from the Reference menu or click on the button.

Data Flow

The purpose of almost any program is to organize and channel data. Since Panorama is a database, this is even more true for programs written in Panorama. This section discusses the techniques for storing and manipulating data within a procedure.

Assignment Statements

An assignment statement computes a value (text or numeric) and stores that value somewhere. Unlike every other statement, an assignment statement has no specific keyword that identifies the statement. Assignment statements always have the format shown below:

```
<data storage location> = <formula>
```

The first part of the assignment statement is the **data storage location**. This is the final destination for the data that is being moved. In fact, sometimes the data storage location is simply called the **destination** of the assignment. The data storage location may be a variable, a field in the currently active record, or the clipboard.

The next part of the assignment statement is the equals symbol. This identifies this statement as an assignment statement.

After the equals symbol is the formula. The formula produces the data that will be stored in the data storage location. The formula may simply take a variable or field and pass it along, or it may process, calculate or filter the data before it passes it along to be stored in the data storage location.

Here‘s a simple assignment statement that takes the contents of B and moves it into A. After this statement is finished both A and B will contain the same value.

```
A=B
```

More complicated assignment statements may combine multiple fields or variables, and they may process the data in some way. An assignment statement may also take a constant value and store it. Here are some examples:

```
A=B*C

Name=upper(myName)

City="San Francisco"
```

In each case, the process is the same. First Panorama calculates the formula to produce a data value. Then it stores the data value in a data storage location.

Triggering Automatic Calculations

A database can be set up so that when a field is modified by the user, one or more formulas are automatically calculated (see “Automatic Calculations” on page 340). When an assignment statement modifies a field, however, these formulas are not automatically calculated. This is to give the procedure programmer the ultimate control over all calculations that occur during the procedure.

If you as the programmer would like the automatic calculations to be performed during an assignment, add an extra equal symbol to the assignment. The two equal symbols must be adjacent with no spaces between them, like this:

```
PriceΩ==19.95
```

In this example, storing the value 19.95 will most likely trigger several additional calculations to compute the total for this line item and the total for the entire invoice.

The Define Statement

The `define` statement is a special kind of assignment statement. This statement defines a value for a variable, but only if that variable doesn't already have a value. In other words, this statement will initialize the variable if the variable's value has not been defined yet, but if the variable already has a value it will not touch the value.

The define statement has two parameters: the name of the variable and the value for the variable.

```
define <variable>,<value>
```

The example shown below will initialize the variables DefaultAreaCode and TaxRate unless they have already been initialized.

```
global DefaultAreaCode,TaxRate
define DefaultAreaCode,"714"
define TaxRate,4.25
```

Variables

A variable is a place in the computer where an item of data can be stored, kind of like a storage bin for a value. Variables may be created by procedures or by SuperObjects. Most procedures will use one or more variables to hold and transfer data as the program runs. Use a variable whenever you need to store a single data item so that you can use it later. Unlike a field, the value variable doesn't change as you move from record to record, or, in the case of a global variable, even when you move from database to database.

Creating a Variable

Panorama has five different statements for creating variables within a procedure. The most common one is the `local` statement (see "LOCAL" on page 5489), which generates temporary variables that only last until the procedure is finished. This statement should be followed by a list of the variables to be created, with each name separated from the next by a comma. This example creates four variables. Because these variables were created with the `local` statement they are called **local variables**.

```
local alpha,gamma,delta,sigma
```

Creating a variable is kind of like surveying a lot on empty land. Once the land is surveyed you know where it is, but the land is still empty until something is built on it. A new variable is like an empty plot of land that has just been surveyed — it has an address (the name) but it doesn't have any data yet.

By the way, Panorama allows any sequence of characters to be used as a variable name. However, if the variable name contains any punctuation (including spaces) it must be surrounded by the chevron characters « and ». (On the Macintosh press **Option**-`\` to create the « chevron character and **Shift**-**Option**-`\` to create the » chevron character. On Windows systems press **Alt**-**0171** to create the « chevron character and **Alt**-**0187** to create the » chevron character.) Here are some examples of typical variable names:

```
x

birthDay

Counter

«Tax Rate»

«PrimeRate%»
```

A variable name must be spelled exactly the same way every time, including upper and lower case. The variable name birthDay is not the same as Birthday or birthday. In fact, you could create three different variables using these three different names (although this is not recommended because it would be very confusing).

By the way, it's always ok to use chevrons around a variable name, even if the name doesn't have any punctuation. «Counter» is exactly the same as Counter, and they can be used interchangeably. So if you have any doubts about whether or not chevrons are necessary, go ahead and use them. No harm, no foul.

Note: Some programming languages require you to create all variables first, before any other statements. Panorama isn't that picky. You can create new variables anywhere in the program. Here is a procedure that creates four variables, does some work, then creates two more variables.

```
local alpha,gamma,delta,sigma
   alpha=4
   gamma="blue"
   delta=alpha*3
   sigma=delta/alpha
local epsilon,omega
   epsilon=0.01
   omega="z"
```

By the way, the indentation is not necessary, it's simply to make the local statements easier to see.

Assigning a Value to a Variable

Once a variable has been created you can assign a value to it. This is done by putting the variable on the left side of an assignment statement (see “Assignment Statements” on page 1368) like this.

```
alpha=4

gamma="blue"

delta=alpha*3

sigma=delta/alpha
```

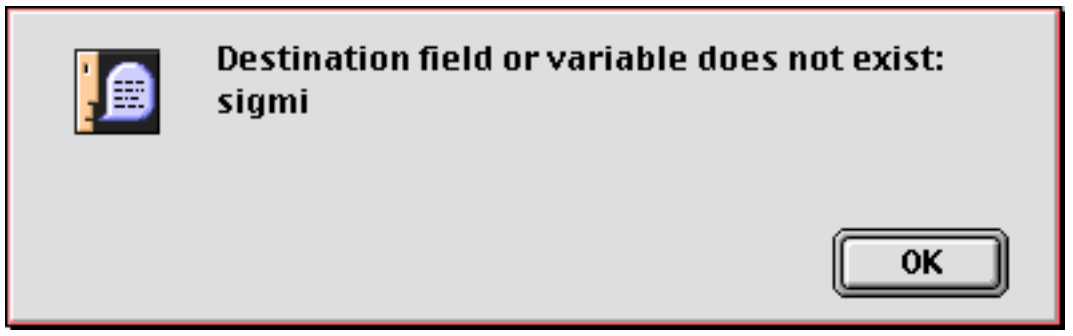
A new value can be assigned to a variable at any time. You can assign a value to a variable once or over and over again a million times. The new value does not have to have anything to do with the old value — you can store a number in a variable that originally held text or vice versa — Panorama doesn’t care. This line of text assigns the number 400 to the gamma variable, which originally had a text value stored in it.

```
gamma=400
```

Some programming languages allow you to assign a value to a variable without having to create the variable first. Panorama’s programming language does not allow this. For example, this procedure creates four variables, then attempts to store a value into a fifth variable, sigmi.

```
local alpha,gamma,delta,sigma
sigmi=200
```

When you run this procedure Panorama will complain. Picky picky picky!



In this case the problem is probably a typo, and the variable name in the assignment needs to be corrected. If this really is a separate variable you must create it first.

Using a Variable in a Formula

Once a variable has been assigned a value you can use it in a formula. A variable may be used anywhere a field or constant may be used. Here are some typical examples.

```
alpha*sigma

gamma+" action"

4*alpha/(delta+20)
```

See “Formula Grammar” on page 1188 to learn more about using variables in a formula.

The Birth and Death of a Local Variable

When a lot is surveyed, that lot usually exists more or less forever (barring wars or natural disasters). A local variable, however, is not nearly that permanent. In fact, local variables only exist until the end of the procedure that they are created in. When the procedure finishes, Panorama checks to see if it created any local variables. If it did, the values in these variables are dumped out and all record of the variables are destroyed, just as if they had never existed in the first place. It's kind of as if you went down to the county recorder's office and burned all the survey records for a tract of land.

Why are the local variables destroyed? Two reasons. First, they take up memory that is no longer needed and can be used for other things. Secondly, this allows different procedures to use the same variable names without having to worry about conflicting with each other. Suppose procedure A and procedure B both have a local variable named gamma. Since the variable is destroyed when each procedure finishes, neither procedure needs to worry about the other. Each happily creates and uses it's own copy of gamma, which is then destroyed before it can interfere with any other procedure.

Note; If you are an experienced programmer you may be wondering about recursion at this point. If you can't resist, you can jump ahead to "Recursive Subroutines" on page 1396.

Long Life Variables

Sometimes, of course, you won't want a variable to be destroyed when the procedure is finished. Panorama actually has five different kinds of variables, each with different life cycles (local, window, fileglobal, global and permanent). You've already learned about local variables. The next most common type of variable is a **fileglobal variable**, which I'm sure you'll be surprised to learn is created with the `fileglobal` statement (see "FILEGLOBAL" on page 5227). Just as with the local statement, the fileglobal statement is followed by a list of variables to create. This statement creates two variables.

```
fileglobal Speed,Direction
```

Unlike a local variable, a fileglobal variable isn't destroyed when the procedure is finished. It hangs around and can be used over and over again. However, fileglobal variables don't last forever. When the database is closed, the values in these variables are dumped and the variables themselves are destroyed.

As you might guess, a **permanent variable** has a very long life. However, the life of a permanent variable is not continuous but interrupted. When the database is closed any permanent variables associated with that database are destroyed. However, before the variables are destroyed the values are stored in the database itself. When the database is re-opened later Panorama automatically re-creates the permanent variables again. (Note: You must save the database. Just as with data in database fields, Panorama only saves permanent variables when the database is saved.)

Another type of long life variable is a **global variable**. A global variable is not destroyed even when the database that created it is closed. It's almost immortal. However, when you **Quit** from Panorama, that's the end of the road for global variables. They are not re-created automatically the next time Panorama opens.

A specialized kind of variable is a **window variable**. This kind of variable is attached to whatever window was open and on top when it was created. When that window is closed, the variable is dumped. Poof! Window variables are usually used with clonable forms (see "Window Clones" on page 1584).

Destroying a Variable

If necessary you can use the `undefine` statement to destroy any variable at any time (see "UNDEFINE" on page 5866). When you use this statement the variable (or variables) is completely destroyed as if it had never been created in the first place. This example destroys the variables Speed and Direction.

```
undefine Speed,Direction
```

You can destroy any kind of variable. If a variable can be accessed, it can be destroyed. Before you destroy a permanent variable, however, you should first make in un-permanent, like this:

```
unpermanent timeStamp
undefine timeStamp
```

It's rarely necessary to destroy a variable yourself. Panorama automatically destroys local variables when the procedure open, destroys fileglobal variables when the file is closed, and destroys windowglobal variables when the window is closed.

Variable Accessibility

Just because a variable exists doesn't mean you can access it. Many types of variables are "attached" to a file or a window and are only available when that file or window is active. When the file or window isn't active, these variables are "dormant." They still exist (and take up memory), but you cannot access or modify them until the file or window they are attached to becomes active again.

`Fileglobal` and `permanent` variables are attached to the file that was active when they were created. When this file is open and on top, the variables are accessible and can be used in a formula or modified with an assignment statement. When some other file is on top these variables are dormant and cannot be used.

The beauty of this system is that it allows you to create variables without worrying about conflicting with other databases. Consider the two fileglobal variables created in the previous section, Speed and Direction. What if some other open database also has variables with these names? As long as both databases use fileglobal variables instead of global variables (see below) they'll both be all right. Each will have their own separate Speed and Direction variables. When database A is active it's variables will also be active while B's are dormant. When database B is active A's variables become dormant. Essentially Panorama will keep two completely separate sets of Speed and Direction variables, each with their own values.

`Windowglobal` variables are attached to the window that was active when they were created. You can have multiple cloned windows that use the same variable name but actually each has it's own separate variable that is only active when the window is on top (see "Window Clones" on page 1584).

`Global` variables are always active, no matter what file or window is on top. This is great if you need to have data that is accessible anywhere at any time. But be careful! If two different databases use the same global variable they had better be co-operating with each other! Remember thay you may open databases that were created by other people. Who knows what global variable names they will use? If you do need to use global variables we recommend that you use very long descriptive names like ProTechSpeed or AcmeSalesTaxRate. Names like this are more likely to be unique and not conflict with anyone elses global variable names.

Accessing "Dormant" Variables

Fileglobal, permanent and window variables are normally "dormant" when the file or window they are associated with is not open and on top (see "Variable Accessibility" on page 1373). However, it is possible to access the values in these dormant variables with special functions. The `grabfilevariable(` function (see "GRABFILEVARIABLE(" on page 5328) can grab the value of a value even if it is dormant. The function below will grab the value of the Speed fileglobal variable even if the Transpac Race database is not on top (it must be open, however). Notice that the variable name (`"Speed"`) must be in quotes .

```
grabfilevariable("Transpac Race","Speed")
```

The `grabwindowvariable(` function is similar (see "GRABWINDOWVARIABLE(" on page 5329) except that it grabs the value of windowglobal variables for windows that are not on top.

"Hidden" Variables and Fields

You may wonder what happens if two or more variables are both accessible and have the same name. For example, what if the current database has a fileglobal variable named Grok and there is also a global variable named Grok. In this case the global variable is "hidden" behind the fileglobal variable and cannot be accessed. The global variable will remain hidden as long as this database is active. The table below shows how different types of variables can hide other types of variables.

Type of Variable	Hides these types (if the name is the same)
local	all other types of variables and fields
windowglobal	fileglobal variables, permanent variables, global variables and fields
fileglobal or permanent	global variables and fields
global	database fields
field	nada!

As the table shows, variables can also hide database fields if they have the same name. This can especially be a problem with global variables, which can interfere with any field in any database. (By interfere we mean that the field will not be accessible in a formula.) Be sure to avoid global variable names like Name, Address, State, Amount, or any name that might be likely to be used as a database field.

Accessing Variables In Form Objects (Text or Images)

Several different form objects can display the results of a formula.

Graphic Objects That Can Display Variables Using a Formula
Auto-Wrap Text (see "Displaying Formulas in Auto-Wrap Text" on page 602)
Text Display SuperObjects (see "Text Display SuperObjects™" on page 608)
Text Editor SuperObjects (see "Text Editor SuperObject" on page 639)
Word Processor SuperObject (see "Merging Data into Word Processing Documents" on page 707)
Flash Art and Super Flash Art (see "Flash Art™" on page 748)
Data Button (see "Data Buttons" on page 827)
Pop-Up Menu Buttons (see "Pop-Up Menus" on page 850)
List SuperObjects (see "List SuperObjects" on page 869)

As long as it is accessible (see "Variable Accessibility" on page 1373) a variable can be used in the formulas for any of these objects. For example, a Text Display SuperObject may display any global variable, any fileglobal or permanent variable created in the same file, or any windowglobal variable created in the same window.

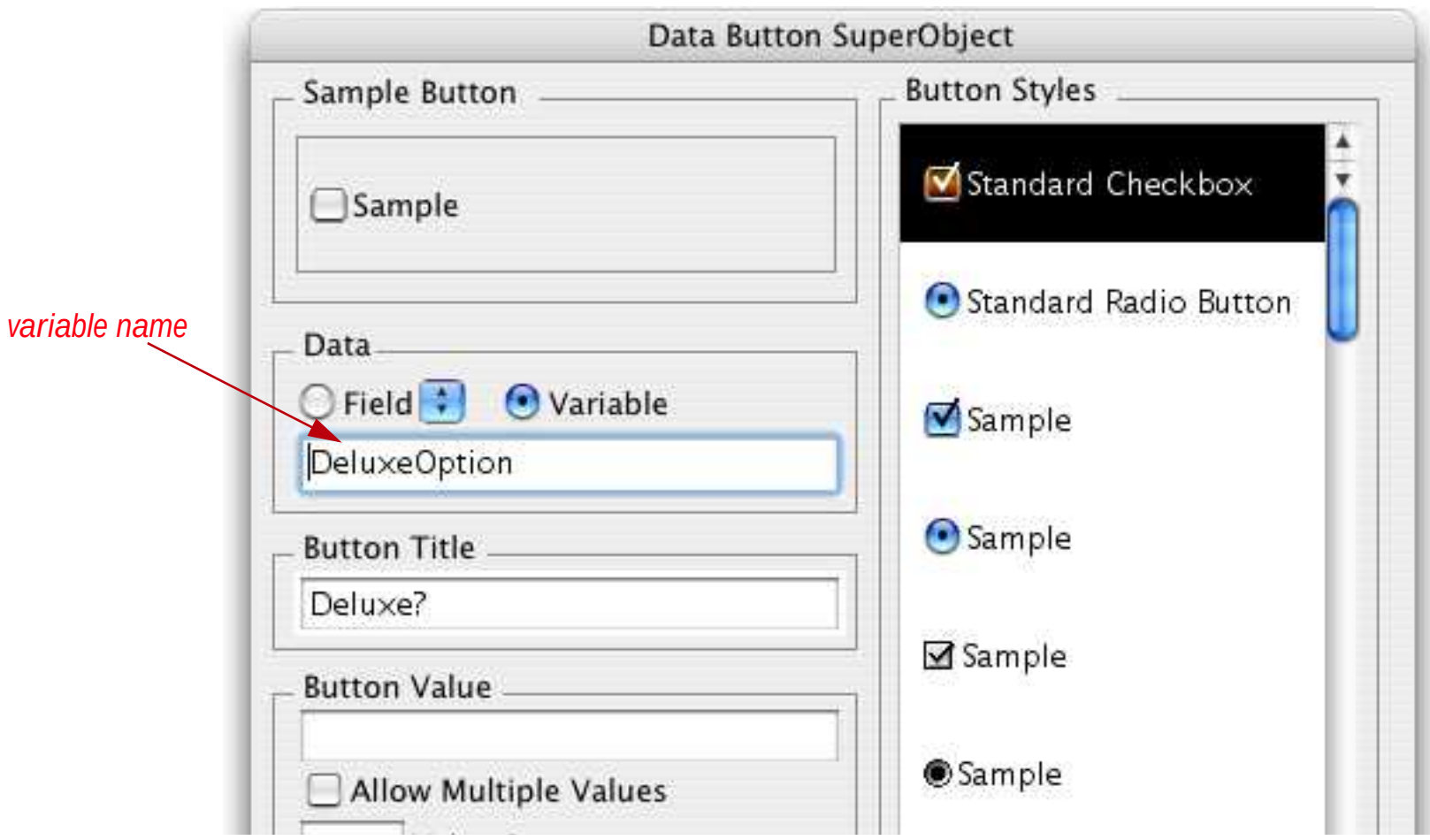
Since a local variable is created and destroyed within each procedure local variables are not accessible to formulas in form objects. If you want to display a variable in a form it must be a windowglobal, fileglobal, permanent or global variable.

Creating Variables with a SuperObject

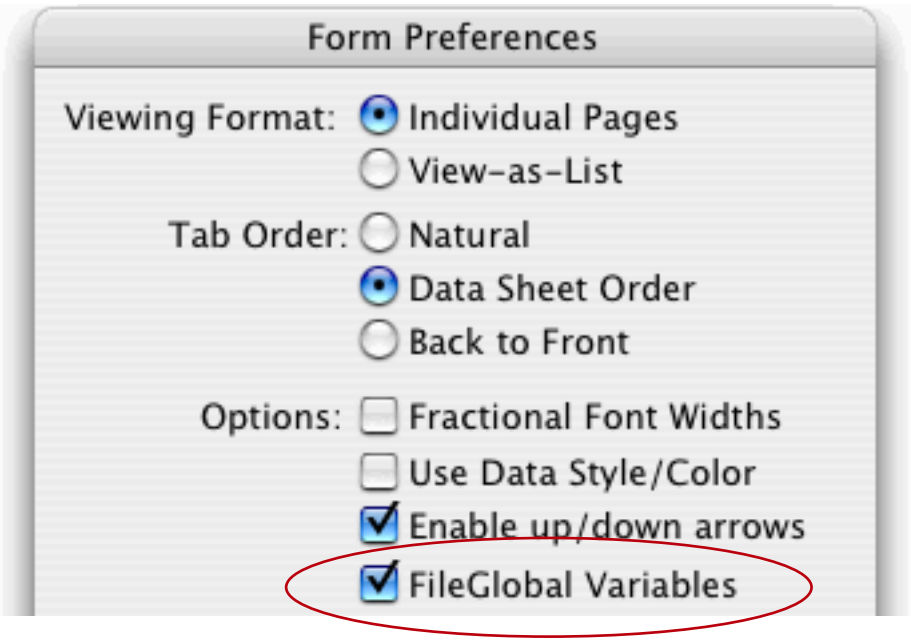
Variables are usually created in a procedure (see "Creating a Variable" on page 1370). However, several different types of SuperObjects have the option of linking to a variable or a field, and these objects will automatically create the variable if it does not exist. When a variable is created by a SuperObject it is always a global or fileglobal variable and is initialized to empty text. SuperObjects that can create variables include the Text

Editor (see "Text Editor SuperObject" on page 639), Word Processor (see "Word Processor SuperObject" on page 673), Data Button (see "Data Button SuperObjects™" on page 828), Pop-up Menu (see "Pop-Up Menu SuperObjects™" on page 850), List (see "List SuperObjects" on page 869), Sticky Button (see "Sticky Push Button SuperObjects™" on page 845) and Scroll Bar (see "Scroll Bars" on page 968).

For example, suppose you are working on a form and create a checkbox using the **Data Button** tool (see "Data Buttons" on page 827). You select the Variable option and type in the variable name DeluxeOption, as shown in this illustration.



When the **OK** button is pressed, Panorama checks to see if you have already created a variable named DeluxeOption. This may be a global variable, a fileglobal or permanent variable (in this database) or a windowglobal variable (in this window). If the variable has already been created, Panorama will simply use it. But if there is no such variable, Panorama will create it as a global or fileglobal variable. The default is a global variable unless the FileGlobal Variables option is set in the **Form Preferences** dialog (Setup menu). Except for how it was created, this variable is just like any other variable and can be used freely in procedures and formulas.



Panorama actually creates the variable the first time it displays this object. If you shut down Panorama and then later re-open it, the variable will be created the first time the form is displayed. If database has been saved with the Save Window Position option turned on (see “Saving Window Positions” on page 132) so that this form opens automatically the variable will be created immediately when the file is first opened.

Permanent Variable Tips

When the `permanent` statement creates a permanent variable, it really creates two variables: one in memory and one in the current database. The one in memory is an ordinary fileglobal variable. Whenever the database is saved, Panorama copies the contents of the fileglobal variable into the copy of the variable in the database itself, then saves the database. Just like any other data, the contents of the permanent variable are not saved unless the database itself is saved. However, if you have not made any other changes to the database, Panorama will not warn you if you attempt to close a database without saving changes to the permanent variable.

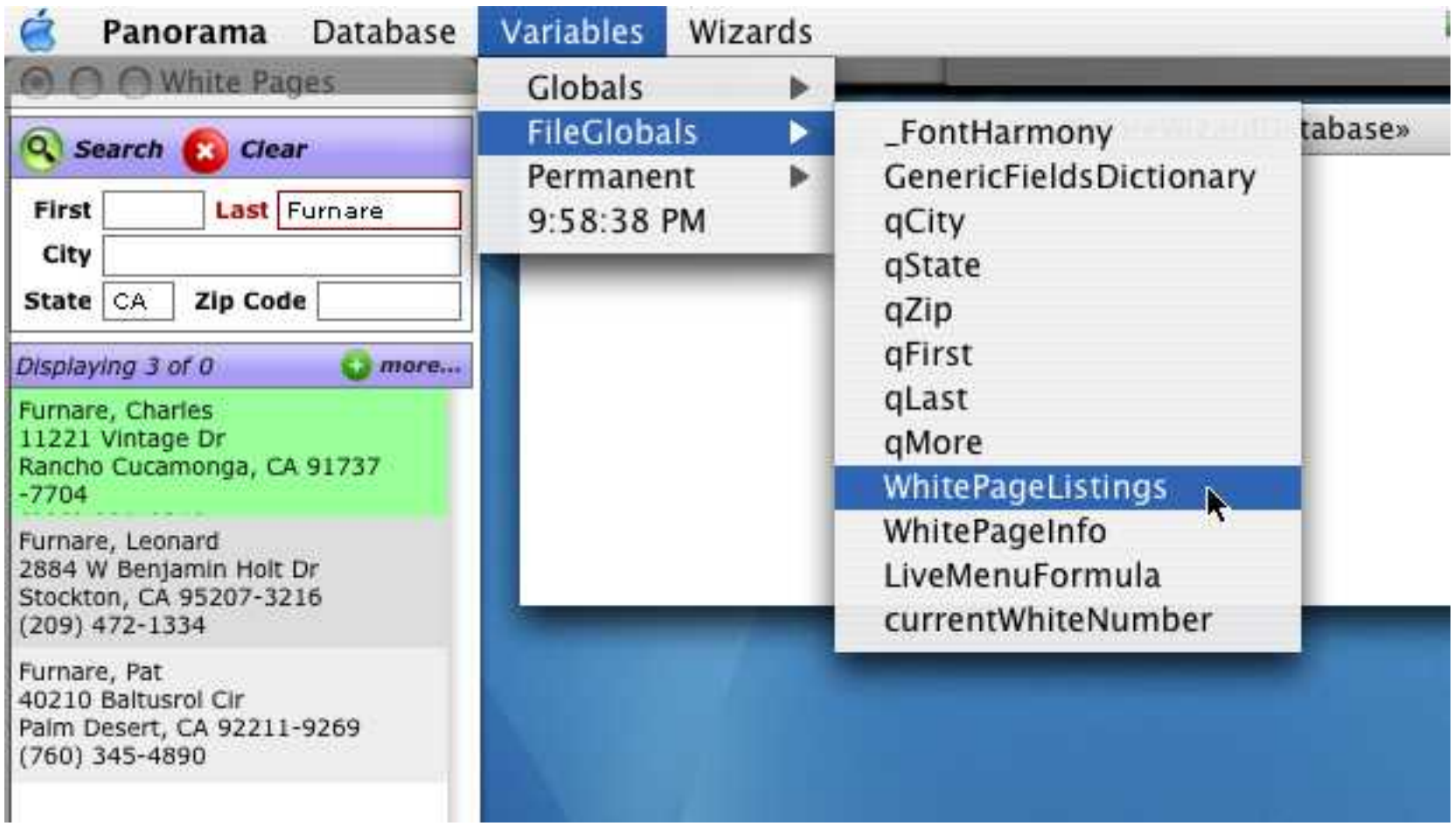
Whenever a database is opened, Panorama automatically creates fileglobal variables for any permanent variables associated with that database. Next it copies the values from the database into the fileglobal variables. The variables are now ready to use.

If you ever want to make a permanent variable un-permanent, use the `unpermanent` statement, which is followed by a list of variables you want to make unpermanent. This statement doesn’t make the variables go away, but they will no longer be permanent. The `unpermanent` statement only affects variables that are permanent in the current database. The example below changes two permanent variables back into regular (non-permanent) global variables.

```
unpermanent myAreaCode,myZipCode
```

Displaying and Changing Variables

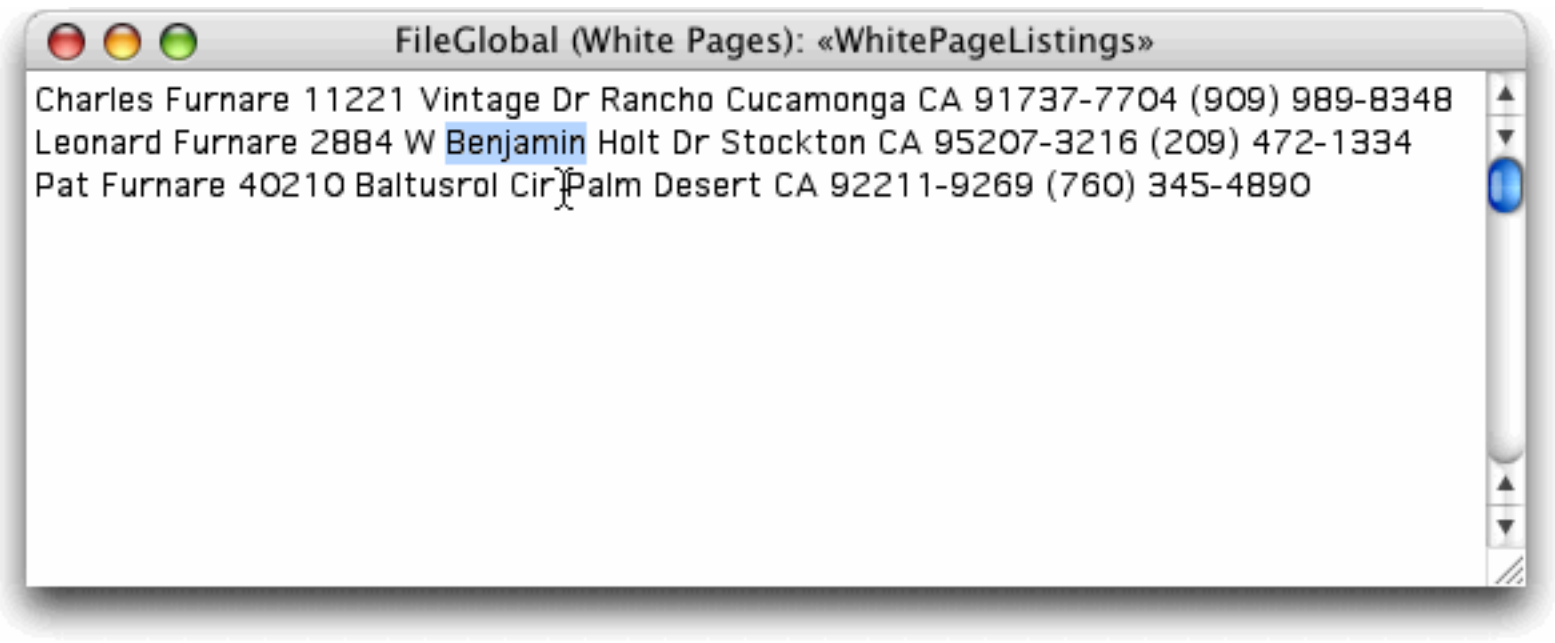
`Global`, `fileglobal` and `permanent` variables can be displayed and modified with the **Variables** wizard (in the Developer Tools submenu of the Wizard menu). To display a variable open the wizard and then pick from one of the three variable submenus.



The wizard window will show the value of the variable. (The window title will show the type of variable, the database it is associated with and the actual name of the variable.



If you are absolutely sure you know what you are doing, you can click on the variable to edit the value.



When you press the **Enter** key the variable will be changed. Press **Command-Period** if you decide not to change the value (**Control-Period** on PC systems).

If you want to view variables in another database use the **Database** menu to select the database that contains the variables you want to examine.

Control Flow

As it runs a procedure, Panorama usually starts with the first statement in a procedure and works its way down. However, Panorama is not limited to this kind of linear approach. The procedure can be designed to make comparisons or decisions and take different steps depending on the result. For example, the procedure might decide to skip one or more steps, or it might decide to repeat a sequence of steps more than once. The procedure may even decide to trigger another procedure to help it complete its job. Programmers call this decision making process control flow, because it controls and possibly alters the flow of statements dynamically as the program is running.

The ability to change the flow of steps "on-the-fly" is the key to programming. A simple example may help make the utility and power of this concept more clear. Suppose you want to create a procedure that adds fifty new records to the end of the database. You could simply create a procedure with the `AddRecord` statement repeated fifty times. Of course this is inconvenient, and what if you wanted to create 2500 new records? Instead of repeating the `AddRecord` statement over and over, you can write a program that repeats a single `AddRecord` statement over and over until the proper number of records have been added.

```
loop
   addrecord
until 50
```

The program is much shorter than if you had literally repeated the `AddRecord` statement 50 times, and can be changed easily if you want to add a different number of new records.

Now suppose you want to change this program so that as it adds new records, it alternately puts the word Black or Gold into the Color field of each record. This requires the program to make a decision for each record—is this record even or odd? There are several ways this could be programmed in Panorama, The example below shows just one of them. In this example you'll notice that all seven steps between loop and until are being repeated 50 times.

```
local nextColor
nextColor="Black"
loop
   addrecord
   Color=nextColor
   if nextColor="Black"
      nextColor="Gold"
   else
      nextColor="Black"
   endif
until 50
```

Although this program is quite simple, it illustrates the basic elements of control flow.

True/False Formulas

In Panorama as in most programming languages, control flow decisions are made on the basis of formulas that are either true or false. The most basic true/false formula compares two values to see if they are equal.

```
PaymentMethod="C.O.D."
```

This formula will compare the value in the field PaymentMethod with C.O.D. The result will be true if PaymentMethod is C.O.D., and false if it contains anything else (for example Check, Cash, Visa, etc.). To learn more about true/false formulas and how to create them see "True/False Formulas" on page 1265.

Equals Comparison vs. Assignment

If you have been paying attention you undoubtedly noticed that the formula in the previous section looks exactly like an assignment. Why doesn't this formula

```
PaymentMethod="C.O.D."
```

assign the value C.O.D. to the field PaymentMethod? At first glance this may appear ambiguous…the same formula is used to compare two values and to assign a value. How do we know when we are assigning and when we are comparing? The answer lies in the context in which the formula is found.

In a procedure, an assignment is always by itself, not part of a larger statement. A true-false formula is always part of another statement, for example `if`, `case`, `until`, `while`, `stoploopif`, `repeatloopif`, `find`, `select`. Here's an example that shows two formulas that look almost the same, but one is a true-false formula and one is an assignment.

```
if PaymentMethod="C.O.D."
   ShippingMethod="UPS"
endif
```

The first formula, `PaymentMethod="C.O.D."`, is part of the `if` statement. Because it is part of the `if` statement this formula means: Is the field (or variable) PaymentMethod equal to C.O.D. (true / false)?

The second formula, `ShippingMethod="UPS"`, is not part of any statement, but stands alone, so this is an assignment. The statement means: Take the value UPS and copy it into the field or variable named ShippingMethod.

If an assignment has more than one equals sign, the first equals sign is for the assignment and the rest are for comparisons. The example assignment below compares B and C. If they are equal (true) the value -1 will be copied into A. If they are not equal (false) the value 0 will be copied into A.

```
A=B=C
```

In other words, A becomes the result of the comparison between `B=C`.

True/False Values

For purposes of calculation, Panorama treats true and false as numbers: true is -1 and false is zero. Like any other number, you can store a true / false value in a field or variable and then use it later. The example below calculates whether a person is a teenager, then uses that information later.

```
local Teenager
Teenager=Age≥13 and Age<20
...
if Teenager
   Price=4.50
else
   Price=6.00
endif
```

Notice that the `if` statement doesn't need to compare, it simply uses the result of the comparison that was calculated earlier. In fact, the `if` statement (and all other statements that use true / false logic) can use any formula that produces a numeric integer result. The value 0 will be regarded as false, and any non-zero value will be regarded as true. The example below will be true if the length of the name is non-zero.

```
if length(Name)
   yesno "Is this a home address?"
   ...
endif
```

The first line of this example could also have been written `if length(Name)<>0`. The result is the same either way.

IF Statements

The basic building block for making decisions in a Panorama program is the **if statement**. The `if` statement will skip over the next few statements (up to the next `endif` statement) if the true/false formula is false. Here's a simple example.

```
if City=""
   City="Pismo Beach"
   State="CA"
endif
message City+", "+State
```

Depending on what's in the City field, this procedure can work one of two ways. If the City field is empty, the true/false formula `City=""` will be true, so the procedure will perform the assignments `City="Pismo Beach"` and `State="CA"`. But if the City field is not empty, the true/false formula will be false, and Panorama will skip past the `endif` to the message statement.

In this example there are two statements between the `if` and `endif` statements. These two statements will be skipped if the formula is false. However, there is no limit to the number of statements that may be between the `if` and `endif`. Just make sure that there is always a matching `endif` for every `if`. Although it is not required, indenting the statements between the `if` and the `endif` usually makes the procedure easier to read and understand.

ELSE Statements

The **else statement** turns the `if` statement into a two way operation: if true, do this, otherwise, do that. You could do this with two `if` statements in a row, but the `else` is simpler.

To use the `else` statement, place it between the `if` and `endif` statements. If the true/false formula is true, Panorama will perform the statements from the `if` up to the `else` and skip the statements from the `else` up to the `endif`. If the true/false formula is false, Panorama will skip the statements from the `if` up to the `else` and perform the statements from the `else` up to the `endif`.

The example below calculates sales tax and shipping for both in-state and out-of-state purchases.

```
if State="CA"
   SalesTax=0.08
   Shipping=2.50
else
   SalesTax=0
   Shipping=5.00
endif
```

If the state is California, the sales tax is 8% and shipping is $2.50. But if the purchase is from any other state, the sales tax is zero and shipping is $5.00. In this example more or less the same statements are used in both halves of the `if`/`else`, but this is not necessary. The two sections could be completely different.

Nested if Statements

Panorama is not limited to one `if` at a time. Panorama can make a decision, execute some more statements, and then make a subdecision. Since the inner `if endif` pair is completely surrounded by the outer pair, this is called nesting.

```
local CardLength
if PaymentMethod="Credit Card"
   CardLength=length(CardNumber)
   if CardLength<13 or CardLength>16
      message "Sorry, invalid credit card number."
   endif
endif
```

If the PaymentMethod is not Credit Card, the procedure will skip all the following statements and do nothing. But if the PaymentMethod is Credit Card, the procedure will continue and calculate the CardLength variable. The second `if` statement checks the card length. The `message` statement will only be performed if both `if` statements are true.

Error Handling with if error

There are literally hundreds of different errors that can occur while a procedure is running. Of course you'll want to eliminate all of the errors in the procedure itself, but many errors are the result of circumstances beyond the programmers control. A file can fail to open because it was placed into the wrong folder, the user can enter the wrong data type into a formula, the list is endless. When such an error occurs, Panorama's normal response is to display an error message and stop the procedure immediately. (In addition, if the procedure window is open, Panorama will attempt to highlight the location of the error.)

If you want to create a database that operates professionally, simply stopping the procedure half finished if there is an error may not be acceptable. Instead, you may want your procedure itself to trap the error and try to correct it, if possible. At a minimum, you may be able to display an error message that is more relevant to an untrained operator than Panorama's general purpose error messages.

To trap errors, use the `if error` statement (two words - there **must** be a space). This statement must be placed immediately after the statement that you are worried might cause an error. For example, suppose you have a procedure that appends a file with the `openfile` statement. If the file is missing or has been moved an error will occur. This example checks for that error, and if the error occurs, asks the user to enter a new file name. The procedure will keep trying until the file is opened successfully or the user gives up and enters an empty name.

```
local txFileName
txFileName="New Transactions"
loop
   openfile "+"+txFileName
   if error
      gettext "Enter the file name",txFileName
      reloopif txFileName≠""
   endif
while 1≠1
if txFileName="" stop endif
/* further processing of the new transactions, below */
...
```

A very useful trick for `if error` is checking to see if a global variable has been initialized with a value (see "Assigning a Value to a Variable" on page 1371). If the variable has already been initialized with a value, you don't want to change that value, but if it has not been initialized, you do want to set the value. The example below checks the AreaCode global variable to see if it has already been set by another procedure. If it has, the statement `xTest=AreaCode` will work perfectly. But if AreaCode doesn't have a value yet, this statement will produce an error. The `if error` statement traps the error and sets the AreaCode variable to 714.

```
fileglobal AreaCode
local xTest
xTest=AreaCode
if error
   AreaCode="714"
endif
```

If you have a lot of variables it may not be necessary to test each one, as long as they are initialized as a group by any procedure that sets them up. If they are initialized as a group you can just test one variable, then if it has not been initialized you can initialize the entire group.

`If error` must be used by itself, you cannot combine other conditions. For example, the statement:

```
if error and info("modifiers") contains "shift"    /* WILL NOT WORK !! */
```

will NOT work. To get this effect you must nest a second if statement inside the if error, like this.

```
if error
   if info("modifiers") contains "shift"
      ...
   endif
endif
```

Another way to handle errors is with the `onerror` statement, which allows you to change Panorama's default behavior for handling an error. See "Catching Program Errors (Especially for Web and other Server Applications)" on page 1407 for details on this statement.

CASE Statements

If a program needs to select one (and only one) option out of many, the **case statement** is the way to go. Like the `if` statement, the `case` statement uses a true-false formula to decide whether or not to perform the following statements. But unlike the `if` statement, which is used alone, the `case` statement is always used in groups. Panorama checks the true-false formula for each `case` statement. If it is false, it skips to the next `case` statement. If it is true, it performs the statements until the next `case` statement. Then it skips past all the rest of the `case` statement to the `endcase` statement.

After all the `case` statements, you may optionally add a `defaultcase` statement. This will pick up any leftovers that weren't included in any of the other cases.

The example below shows how the case statement can be used to divide people up into five age groups.

```
case Age<5
   AgeGroup="Pre-School"
case Age<13
   AgeGroup="Youth"
case Age<20
   AgeGroup="Teen"
case Age≥65
   AgeGroup="Senior"
defaultcase
   AgeGroup="Adult"
endcase
```

This example has included one statement for each `case` statement, but there is no limit to the number of statements that may be included in each section. However, there is a maximum limit of 75 `case` statements per `endcase` statement.

LOOP Statements

A **loop** allows Panorama to repeat a sequence of statements over and over again. The loop can be repeated a fixed number of times, or until a special condition is fulfilled.

All loops begin with the `loop` statement, and end with either `until` or `while`. The statements in between these two statements are said to be "inside the loop." These are the statements that will be repeated over and over again. Although it is not required, your procedures will usually be easier to read and understand if the statements inside the loop are indented.

To repeat the statements inside the `loop` a fixed number of times, use the `until` statement with a number after it. This number may be a fixed number, or a variable or formula that calculates a number. For example, this procedure will add a dozen shiny new records to the database:

```
loop
   addrecord
until 12
```

To repeat the statements inside a loop until a specific condition is met, put a true/false formula after the `until` statement. Like the previous example, this example adds new records to the database. In this case, however, the number of new records is determined by asking the user (with the `gettext` statement, see "GETTEXT" on page 5317).

```
local NewCount
NewCount="1"
gettext "How many new records?",NewCount
NewCount=val(NewCount)
loop
   NewCount=NewCount-1
   AddRecord
until NewCount=0
```

The `while` statement is the exact opposite of the `until` statement; it repeats the loop as long as the formula remains true. Here is the previous example rewritten to use the `while` statement.

```
local NewCount
NewCount="1"
gettext "How many new records?",NewCount
NewCount=val(NewCount)
loop
   NewCount=NewCount-1
   AddRecord
while NewCount>0
```

These two examples are exactly the same except for the last line.

Note: The `while` statement can also be followed by the special word `forever`, which tells Panorama to repeat the loop forever. Usually this is used with a `stoploopif` statement to break the loop, otherwise your procedure won't ever stop!

Stopping a Loop in the Middle

The `stoploopif` statement allows Panorama to break out of the loop in the middle (or even at the top), instead of at the bottom. Panorama will break out of the loop if the true-false formula is true.

The example below finds every record where the field PrintDuplicate contains Yes. Each of these records is duplicated. But what if there were no such record? The `stoploopif` statement will stop the loop before it ever begins. The `stoploopif` statement also checks each time the loop is repeated to see if the next statement has found another record to duplicate, or if the loop is done.

```
toprecord
find PrintDuplicate="Yes"
loop
   stoploopif info("notfound")
   copyrecord
   pasterecord
   downrecord
   next
while forever
```

Notice that this sample uses `while forever`. This means that the `while` statement will never stop the loop.

Restarting a Loop in the Middle

The `repeatloopif` statement tells Panorama to restart the loop from the top. The example procedure below tries to extract a phone number from the clipboard.

```
local X,theChar,aPhone
X=1
aPhone=""
loop
   theChar=clipboard()[X;1]
   X=X+1
   stoploopif theChar=""
   repeatloopif theChar≠"(" and aPhone=""
   aPhone=aPhone+theChar
until aPhone match "(???) ???-????"
```

Each time the loop goes around it copies the next character from the clipboard into the variable theChar. If there are no more characters, the loop stops. Each character is checked to see if it is a left parenthesis. Until a (is found, the `repeatloopif` statement stops the loop short, repeating only the top portion of the loop. Once the (is found the loop starts collecting the following data into aPhone. The loop finally stops when the entire phone number is collected or the clipboard runs out of data.

Subroutines

Sometimes you may need to use the exact same series of steps in several places in your program. Wouldn't it be nice if Panorama had a special statement that performed this series of steps for you, so you wouldn't have to type those same steps over and over again? Your programs would be smaller, easier to create, and easier to modify. You can't create your own statements, but a subroutine is the next best thing.

A subroutine is used by "calling" it. It's sort of like calling someone to dinner. When a subroutine is called, Panorama temporarily stops performing the steps in the current procedure. It marks its place in the current procedure, and then starts performing the steps in the subroutine. When it has completed all the steps in the subroutine Panorama goes back to the original procedure and starts off right where it left off. The net effect is as if the statements from the subroutine were copied into the middle of the original procedure.

When the same steps are used in different places, a subroutine has many advantages. First of all, using a subroutine makes the database smaller, because these statements appear only once. An even bigger advantage is that if the statements in the subroutine ever need to be changed, they will only have to be changed in one place, instead of over and over again.

It's possible for the main procedure and the subroutine to pass values back and forth between them. These are called parameters. Parameters allow very general subroutines to be written that can handle a wide variety of situations.

CALL Statement

The call statement allows any procedure in the current database to be called as a subroutine. The basic format is simple:

```
call <procedure name>
```

For example, suppose you have created a procedure called DuplicateRecord. The procedure looks like this:



We can use this procedure in another procedure by calling it.

```
toprecord
find PrintDuplicate="Yes"
loop
   stoploopif info("notfound")
   call DuplicateRecord
   next
while forever
```

Each time Panorama repeats the loop it will call the DuplicateRecord procedure. The three steps in that procedure will be performed, then it will return to the loop and perform the next statement. As far as Panorama is concerned, this is exactly the same as if you had written the procedure this way.

```
toprecord
find PrintDuplicate="Yes"
loop
   stoploopif info("notfound")
   copyrecord
   pasterecord
   downrecord
   next
while forever
```

Although there is no difference as far as running the procedure is concerned, there is a big difference for writing procedures. Suppose you have many procedures that need to duplicate a record. If you create a subroutine to duplicate the record, you can save two lines of typing each time you need to duplicate a record. Most subroutines have more than three lines, so the savings are even more substantial.

An even more important advantage is that using subroutines allows you to “modularize” your code. You’ll probably never have to modify the simple code needed to duplicate a record, but more complicated subroutines often need to be adjusted from time to time. If you had simply typed the statements of the subroutine into each location where they were needed (as shown in orange above) then making the adjustment would be very time consuming because you would have to locate and modify each copy of the statements. By collecting these statements together in a subroutine you can make any adjustments necessary to the code in a single location. Every procedure that calls the subroutine will automatically get the benefit of the adjustments.

Calling Procedures With Unusual Names

If a procedure has a space or other punctuation inside the procedure name, you must enclose the procedure name in quotes, like this:

```
call "Calculate P/E Ratio"
```

Quotes are not necessary for a procedure name that contains a period, even if the period is the first character of the name.

```
call .DialNumber
```

It is even possible to calculate the procedure name with a formula. The formula must be surrounded with parentheses. The example below assumes that there is a dialing procedure for several different fields in the database, Dial Name, Dial Company, etc. If there is such a procedure for the current field, this procedure will call it.

```
call ("Dial "+info("fieldname"))
if error
   message "Sorry, can't dial  the "+info("fieldname")
endif
```

If there is no such procedure, the error message will appear.

Passing Values to a Subroutine (Parameters)

There are a couple of ways to communicate values between the original procedure and the subroutine. One is to simply put the values in one or more fileglobal or global variables.

A more flexible method is to use **procedure parameters**. A subroutine may have one or more procedure parameters. Each procedure parameter is numbered, starting from 1. When you call the subroutine, you must pass the procedure parameters after the subroutine name. Each parameter must be separated from the next with a comma, like this:

```
call <procedure>,<parameter 1>,<parameter 2>, …
```

Each procedure parameter may be a field, a variable, a text or numeric constant, or a complete formula. However, if you are going to change a parameter with the `setparameter` statement, that parameter must be a field or a variable.

Inside the procedure, the programmer can use the `parameter(` function to retrieve the parameter values (see “PARAMETER(” on page 5592). This function itself has one parameter: the procedure parameter number, for example `parameter(1)`, `parameter(2)`, etc.

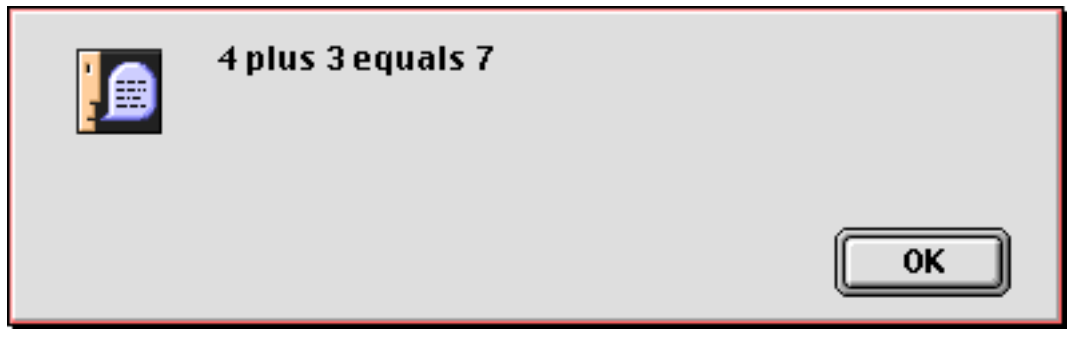
Here is a silly little procedure named Addition that displays the result of an addition problem.

```
message str(parameter(1))+" plus "+str(parameter(2))+" equals "+
   str(parameter(1)+parameter(2))
```

Any other procedure in the same database can call this procedure with two numeric parameters, like this.

```
call Addition,4,3
```

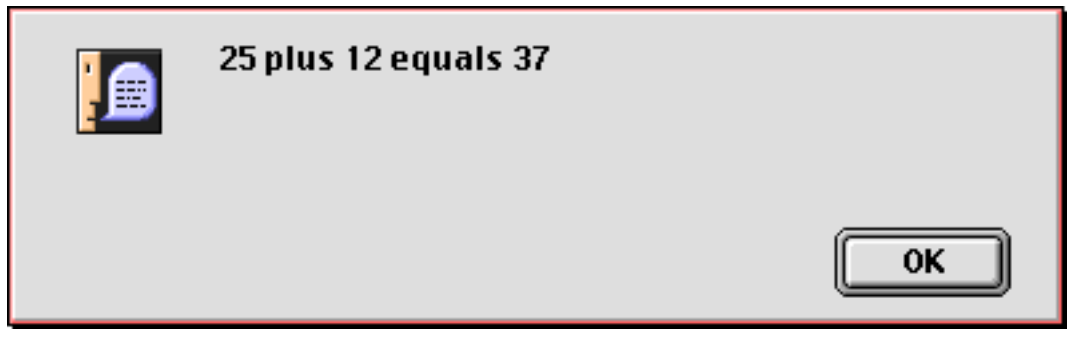
When you run this procedure it calls the subroutine and displays this alert.



By changing the parameters you can change the result.

```
call Addition,35,12
```

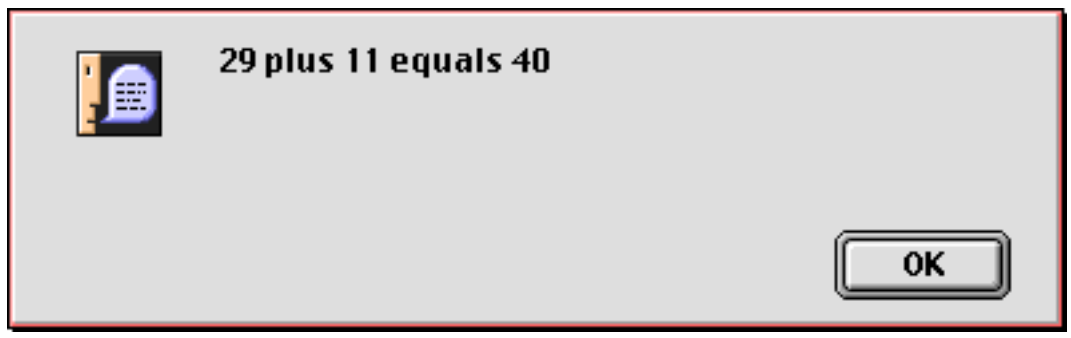
The subroutine grabs the parameters and puts up the result.



Each parameter may be a complete formula containing variables, constants, operators and functions.

```
call Addition,4*2+3*7,sqr(121)
```

Panorama will compute each parameter and pass it to the subroutine.



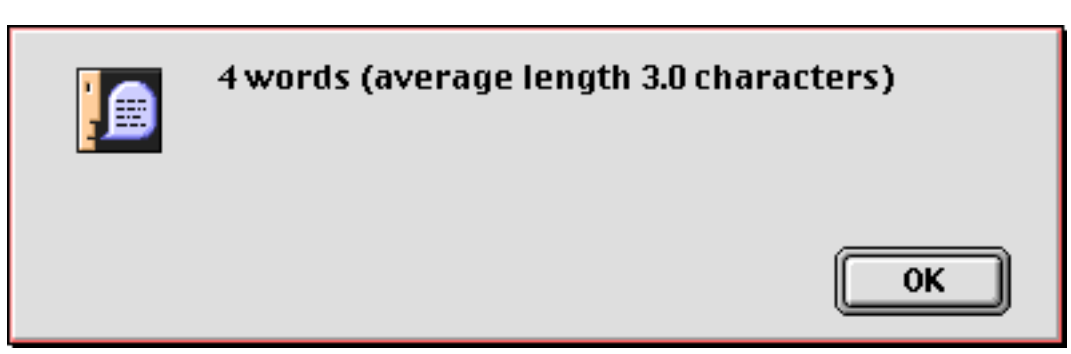
Subroutine parameters can be numbers or text. Here is a procedure named WordStats that takes a single text parameter.

```
local words,wordcount,letters
words=parameter(1)
wordcount=arraysize(words," ")
letters=stripchar(words,"AZaz")
message str(wordcount)+" words (average length "+
   pattern(length(letters)/wordcount,"#.#")+" characters)"
```

This procedure can be called as a subroutine like this.

```
call WordStats,"Now is the time"
```

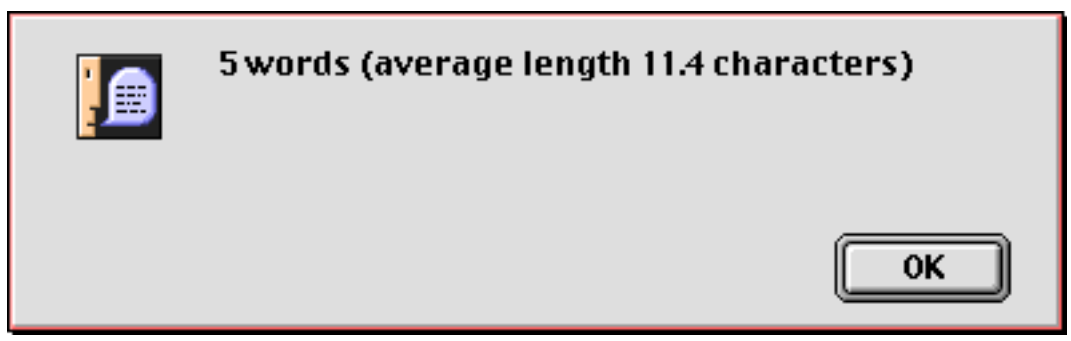
Here is the result.



Just as with the previous example you can pass any data you want to this subroutine.

```
call WordStats,"Dysfunctional institutions instantiate excessive gobbledigook"
```

The subroutine calculates the new statistics.



Passing Values Back From a Procedure

The subroutine can also change a parameter value using the `setparameter` statement (see “SETPARAMETER” on page 5747). This statement itself has two parameters, the procedure parameter number you want to change, and new value.

```
setparameter <number>,<value>
```

Here's a procedure named Weekend that decides whether the current day is during the week or on a weekend.

```
if datapattern(today(),"DayOfWeek") beginswith "S"
   setparameter 1,"Weekend"
else
   setparameter 1,"Weekday"
endif
```

Any other procedure in this database can call this procedure to find out if today is a weekday or a weekend. This procedure adds a new record to the database on weekdays but not on weekends. The parameter is the local variable TypeOfDay.

```
local TypeOfDay
call Weekend,TypeOfDay
if TypeOfDay="WeekDay"
   addrecord
endif
```

A parameter can be passed to a procedure and then back again. Here's a modified version of the Weekend procedure that works for any day, not just today.

```
if datapattern(parameter(1),"DayOfWeek") beginswith "S"
   setparameter 1,"Weekend"
else
   setparameter 1,"Weekday"
endif
```

Here is a procedure that uses this revised subroutine. Notice that the DayInfo variable is assigned a value (December 7, 1941) before being passed to the procedure. The procedure gives the DayInfo variable a new value (Weekend).

```
local DayInfo
DayInfo=date("December 7, 1941")
call Weekend,DayInfo
message "Pearl Harbor was bombed on a "+DayInfo
```

An important point to understand is that the subroutine does not know the name of the field or variable it is modifying. It could be DayInfo, TypeOfDay, or ZippityDoo — it's up to the procedure that calls the subroutine.

In the previous section you learned that a parameter can be any formula, for example `3*4` or `array(Address,2,f)`. However, this is not true for parameters that are modified by the subroutine. A parameter that is going to be modified must be a field or variable. For example, you cannot call the Weekend subroutine like this.

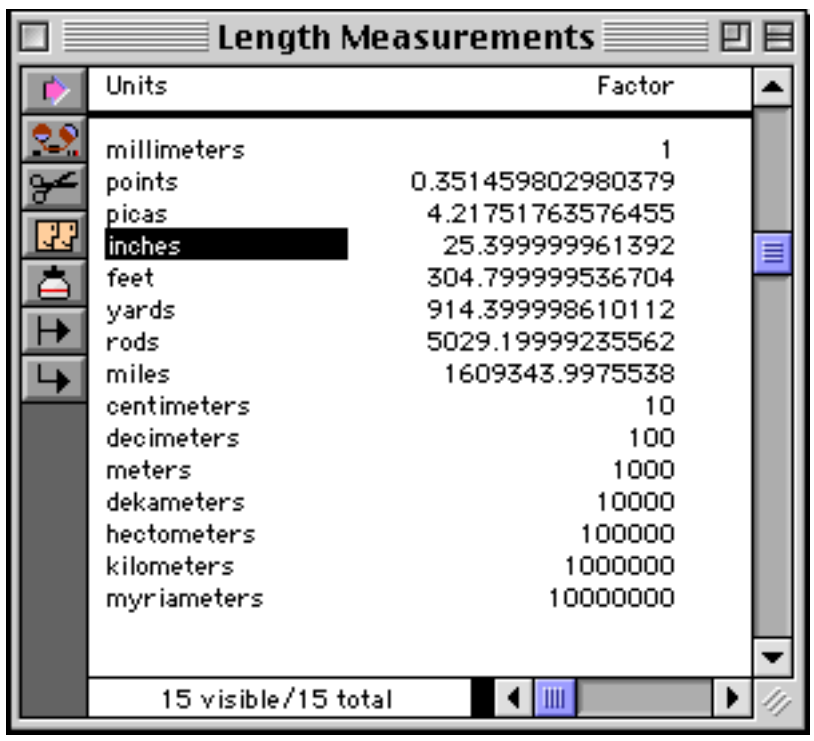
```
local DayInfo
call Weekend,date("December 7, 1941")  <--- WRONG
message "Pearl Harbor was bombed on a "+DayInfo
```

The problem with this procedure is that the subroutine has no idea where to put the result.

Here is a more useful subroutine that uses parameters. This subroutine (named ConvertLength) can convert one measurement system into another (for example feet into meters).

```
/* call ConvertLength,dimension,from,to */
local from,to,length
from=lookup(info("databasename"),Units,lower(parameter(2)),Factor,0,0)
to=lookup(info("databasename"),Units,lower(parameter(3)),Factor,0,0)
if from=0 or to=0 rtn endif
length=(parameter(1)*from)/to
setparameter 1,length
```

The ConvertLength subroutine is designed to be part of this database, which contains the measurement factors used by the `lookup(` functions in the procedure (see above).



Here is a procedure that uses the subroutine to convert 12 inches into feet.

```
local original,converted
original=12
converted=original
call ConvertLength,converted,"inches","feet"
message str(original)+"="+pattern(val(converted),"#.###")
```

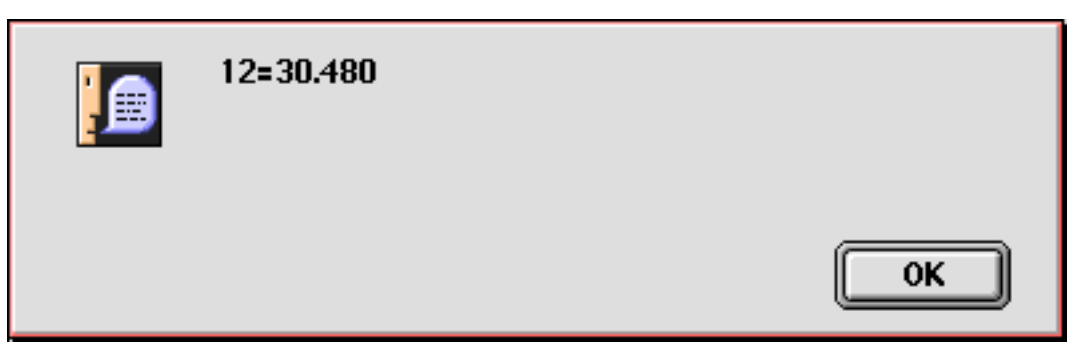
The result is this alert.



By making a slight adjustment we can convert 12 inches into centimeters.

```
local original,converted
original=12
converted=original
call ConvertLength,converted,"inches","centimeters"
message str(original)+"="+pattern(val(converted),"#.###")
```

The result is this alert. News flash — 12 inches equals 1 foot.



This procedure/subroutine combination illustrates a quirk in the way Panorama handles numeric parameters. If you look at the original procedure you will notice that it is setting the parameter to a numeric value.

```
/* call ConvertLength,dimension,from,to */
local from,to,length
from=lookup(info("databasename"),Units,lower(parameter(2)),Factor,0,0)
to=lookup(info("databasename"),Units,lower(parameter(3)),Factor,0,0)
if from=0 or to=0 rtn endif
length=(parameter(1)*from)/to
setparameter 1,length
```

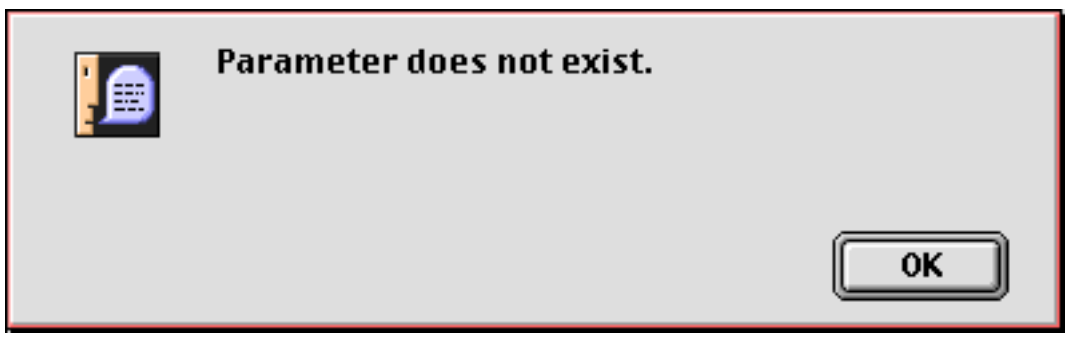
However, the `setparameter` statement always converts numbers to text when it stores the result in a variable. Because of this the calling procedure must use the `val(` function to convert the number back into a number again.

```
local original,converted
original=12
converted=original
call ConvertLength,converted,"inches","centimeters"
message str(original)+"="+pattern(val(converted),"#.###")
```

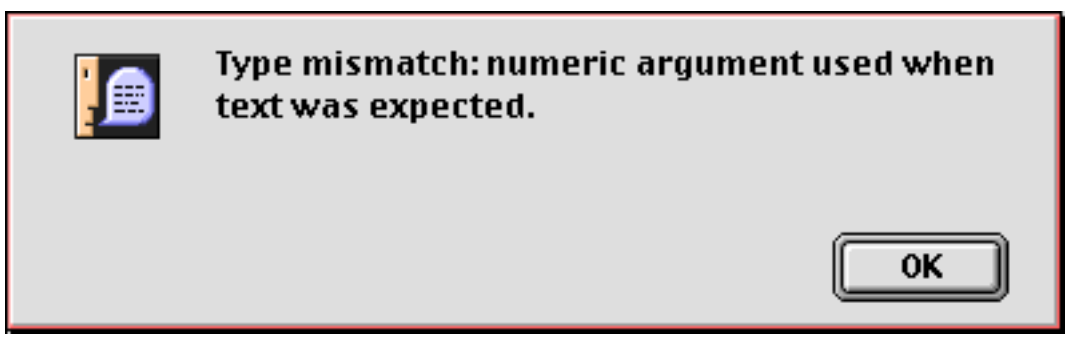
You'll need to keep this in mind if your subroutine passes back numeric values. (Note: This quirk could probably be considered a bug. However, if we fixed it then all of the databases that currently work (by using the `val(` function) would suddenly become broken. Therefore we intend to keep it this way.)

What if the parameters don't match the procedure?

Like a hand in a glove, the procedure parameters supplied as part of the `call` statement must exactly match the parameters used by the procedure being called. For example, consider the ConvertLength procedure in the last example. If you call this procedure, you must supply at least three parameters. (It's ok to supply more than three…the extra parameters will be ignored.) If you supply less than three parameters, the ConvertLength procedure will stop and an error message will be displayed when it tries to access a missing parameter.



In addition to having the correct number of parameters, the parameters must also have the correct data type. In our `ConvertLength` example, the first parameter supplied must be a number, while the second and third parameters must be text. If the wrong type of data is passed in the parameter, the procedure will stop and display an error message when you try to use the value. Here's the message that appears if a numeric parameter is passed when a text parameter is required, a similar message appears for the opposite case.



It's possible to check for missing parameters in a procedure using the `if error` statement. This allows you to perform your own action instead of displaying the default error message. Here's a revised version of the ConvertLength procedure that displays custom error messages and then stops if a parameter is missing.

```
/* call ConvertLength,dimension,from,to */
local from,to,length,fromUnits,toUnits
fromUnits=parameter(2)
if error
   messaage "From units must be inches, feet, yards, centimeters, etc."
   stop
endif
toUnits=parameter(3)
if error
   messaage "To units must be inches, feet, yards, centimeters, etc."
   stop
endif
from=lookup(info("databasename"),Units,lower(fromUnits),Factor,0,0)
to=lookup(info("databasename"),Units,lower(toUnits),Factor,0,0)
if from=0 or to=0 rtn endif
length=(parameter(1)*from)/to
setparameter 1,length
```

Although this example stops if there is a parameter error, that is not absolutely necessary. If you can determine a reasonable default value for a missing parameter the procedure can simply substitute that value and continue on its way. Here is another variation of the ConvertLength procedure that defaults to inches if the a parameter is missing.

```
/* call ConvertLength,dimension,from,to */
local from,to,length,fromUnits,toUnits
fromUnits=parameter(2)
if error
   fromUnits="inches"
endif
toUnits=parameter(3)
if error
   toUnits="inches"
endif
from=lookup(info("databasename"),Units,lower(fromUnits),Factor,0,0)
to=lookup(info("databasename"),Units,lower(toUnits),Factor,0,0)
if from=0 or to=0 rtn endif
length=(parameter(1)*from)/to
setparameter 1,length
```

Here is a procedure that uses the revised subroutine to converts 12 centimeters into inches.

```
local original,converted
original=12
converted=original
call ConvertLength,converted,"centimeters"   <-- missing parameter defaults to inches
message str(original)+"="+pattern(val(converted),"#.###")
```

Or the missing parameter can be in the middle of the list like this. In this case 12 inches will be converted into centimeters.

```
local original,converted
original=12
converted=original
call ConvertLength,converted,,"centimeters"   <-- missing parameter defaults to inches
message str(original)+"="+pattern(val(converted),"#.###")
```

Calling a Subroutine in Another Database

The `call` statement calls another procedure in the current database as a subroutine. (The current database is the database associated with the topmost window — not necessarily the database the current procedure belongs to.) The `farcall` statement can call any procedure in any open database, not just the current one. The format of this statement is almost identical to the `call` statement, but you must specify the database name.

```
farcall <database>,<procedure>,<parameter 1>,<parameter 2>, …
```

The database name should usually be in quotes, like this.

```
farcall "Length Measurments",ConvertLength,converted,"feet","miles"
```

It's also possible to use a formula to calculate the database name. The example below searches for any open database with the word Phone in the name, then attempts to call the .Dial procedure in that database.

```
local X,dbList
dbList=info("files")
X=search(dbList,"Phone")
if X=0
   message "No phone database open!"
   stop
endif
X=arrayelement(dbList,X,¶)
farcall (array(dbList,X,¶)),.Dial,Name
```

When the database name is calculated as in this example, the formula must be surrounded by parentheses ( ).

Terminating a Subroutine in the Middle

The `rtn` statement (short for return) allows a subroutine to stop short in the middle and return to the original procedure. Usually when a subroutine is called, all the statements in the subroutine are performed from top to bottom. But if the `rtn` statement is encountered, the subroutine stops and immediately goes back to the original procedure. The `rtn` statement is almost always used in combination with the `if` or `case` statement.

The simple example below dials a local phone number, which is passed to the subroutine in parameter 1. If no phone number is passed, or if a long distance phone number is passed, the subroutine returns without doing anything.

```
local DialNumber
DialNumber=parameter(1)
if error
   rtn
endif
if DialNumber="" or length(DialNumber)>8
   rtn
endif
dial DialNumber
```

If the `rtn` statement is encountered in a procedure that has not been called as a subroutine (i.e. an original procedure) the procedure will simply stop.

Panorama also has another statement that terminates a subroutine in the middle: `rtnerror`. This statement allows a subroutine to generate an error (complete with error message). If the statement following the original call statement is `if error`, the subroutine will return and the error will be processed by the `if error` statement (see “Error Handling with if error” on page 1381). If there is no `if error` statement, the program will simply stop immediately and display an error message.

Mini Subroutines within a Procedure

Sometimes you may want to use a short subroutine, perhaps two or three lines. It just seems like too much hassle to create a separate procedure. For these situations, Panorama allows you to create a subroutine right inside the current procedure. This special subroutine within a procedure is called a **short subroutine**.

A short subroutine always begins with a **label**. A label is a unique series of letters and numbers that identifies a location within the procedure. The label may not contain any spaces or punctuation except for . and $, and must always end with a colon. The colon is not actually part of the label, it simply identifies the series of letters and numbers as a label, as opposed to a field or variable. Here are some examples of labels:

```
diamond:

blue:

mailAction7:

Dispatch.Route:
```

A short subroutine ends with the end of the procedure, or with a `rtn` statement. A single procedure may contain many short subroutines, each starting with a label and ending with a `rtn` statement.

Short subroutines are called with the `shortcall` statement. This statement is always followed by the name of the short subroutine (the label). Don’t include the colon here. You must type the label exactly as it appears at the top of the short subroutine (except for the colon), no quotes, and unlike a regular call statement the subroutine name cannot be calculated. The `shortcall` statement also does not allow parameter passing. Here are examples of how to call short subroutines.

```
call diamond

call blue

call mailAction7

call Dispatch.Route
```

The example below contains a short subroutine called GroupTotal. The short subroutine starts on line 7, with the label `GroupTotal:`. This short subroutine performs three steps and then returns to the main section of the program. The main section of the program calls the subroutine twice, then stops.

```
field City
shortcall GroupTotal
field State
shortcall GroupTotal
stop

GroupTotal:
   groupup
   field "Amount"
   total
   rtn
```

If the `stop` statement was not included, the program would continue down and perform the steps in the short subroutine a third time. In fact, we do this on purpose to produce a shorter version of this program.

```
field City
shortcall GroupTotal
field State

GroupTotal:
   groupup
   field "Amount"
   total
```

We've also removed the `rtn` statement at the end of the short subroutine. It's redundant in this case because the short subroutine ends at the end of the entire procedure. However, if there were additional short subroutines after this one, all but the last one would require a `rtn` statement at the end.

Subroutines and Local Variables

Earlier in this chapter you learned that local variables are destroyed at the end of the procedure that created them (see "The Birth and Death of a Local Variable" on page 1372). Local variables also become dormant when a subroutine is called (except for short subroutines, see "Mini Subroutines within a Procedure" on page 1393). At the end of the subroutine the local variables come out of hibernation. Because of this, local variables created in one procedure cannot be accessed in another procedure. Local variables are always completely separate.

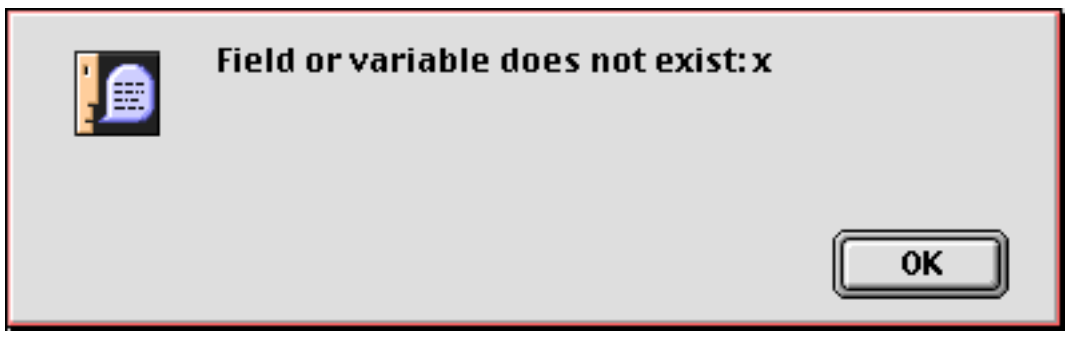
To illustrate this, consider this procedure which creates a local variable named x.

```
local x
x=2
call test
message x
```

The procedure test contains only one line which displays the x variable.

```
message x
```

However, this subroutine does not work! The x variable is now dormant and is not accessible to the test procedure. Instead of displaying the value 2, this error message appears.



As a matter of fact, Panorama allows the test subroutine to have it's own separate x variable, like this.

```
local x
x=9
message x
```

Now when the original procedure (see above) is run two alerts appear. The first displays the value 9 (the value of x set in the test subroutine). The second displays the value 2 (the original value of x which was dormant but re-appeared when the test subroutine was finished).

The UseCallersLocalVariables and UseMyLocalVariables Statements

As described in the previous section, local variables created in a procedure can normally only be accessed in that procedure and not any other. In some special cases, however, it can be useful for a subroutine to access the local variables of the subroutine that called it. This is especially true when you are creating procedures for use as custom statements, and when using the execute statement.

Panorama has two statements that allow a procedure to access the local variables of the procedure that called the current procedure:`UseCallersLocalVariables` and `UseMyLocalVariables`. These statements **must** be used as a pair. (Be careful not to use the `Rtn` statement without first using the `UseMyLocalVariables` statement.) The `UseCallersLocalVariables` statement temporarily swaps out a procedures current local variables with the local variables of the procedure that called this procedure. The `UseMyLocalVariables` statement swaps the variables back again. While the variables are swapped you have full access to the local variables of the caller. You can even create new local variables, which become local variables belonging to the calling procedure. Here is an example that creates a local variable named alphabet and initializes it with 26 letters.

```
usecallerslocalvariables
local alphabet
alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
usemylocalvariables
```

A procedure could call this subroutine like this. The message statement will display ABCDEFGHIJKLMNOPQRSTUVWXYZ even though this local variable wasn't created in this procedure.

```
call MakeAlphabet
message alphabet
```

Recursive Subroutines

A **recursive subroutine** is a subroutine that calls itself. Some programming languages don't allow recursion, but Panorama does allow recursion up to 8 levels deep. Here is an example of a recursive procedure called AddAddAdd. Notice that it calls itself on line 4.

```
local x,y
x=parameter(1)
if x>1
    call AddAddAdd,x-1,y
    y=val(y)+x
else
    y=x
endif
setparameter 2,y
```

Given an integer, this procedure will calculate the sum of that integer plus all lower integers. For example, if you start with 4 the procedure will compute 4+3+2+1 = 10. The procedure calls itself for each addition. Here is a procedure that calls AddAddAdd to start the computation. In this case the computation starts with 6, so the result will be 6+5+4+3+2+1 = 21.

```
local answer
call AddAddAdd,6,answer
message answer
```

Although Panorama allows recursion it is very limited. The AddAddAdd procedure won't work with any number larger than 8. Usually it's best to try to write a procedure without recursion. Here's an example of a procedure that solves the same problem without recursion. Now we can calculate that for a larger number like 68 our cumulative sum is 2346.

```
local x,sum
x=68
sum=0
loop
    sum=sum+x
    x=x-1
while x>0
message sum
```

As you can see, the non-recursive solution is actually quite a bit simpler than the recursive solution. However, there are some cases where recursion can greatly simplify the solution of a problem, and Panorama does have a limited capability to allow recursion.

Other Control Flow Statements

There are a few other control flow statements that don't fit into any neat categories. These statements are described in the following sections.

Jumping to an Another Location in the Program

The `goto` statement tells Panorama to jump immediately to another spot in the program (see "GOTO" on page 5326). The format of this command is simple:

```
goto <label>
```

A **label** is a unique series of letters and numbers that identifies a location within the procedure. The label may not contain any spaces or punctuation except for . and $, and must always end with a colon. The colon is not actually part of the label, it simply identifies the series of letters and numbers as a label, as opposed to a field or variable. Here are some examples of labels:

```
diamond:

tryAgain:

accessRoute3:

Start.Over:
```

The example subroutine below asks the user to enter an angle. If the angle is not between 0 and 360, Panorama will jump back to the label TryAgain.

```
fileglobal GoAngle
GoAngle="0"

TryAgain:
   gettext "Enter direction (0-360)",GoAngle
   if GoAngle>360 or GoAngle<0
      goto TryAgain
   endif
   setparameter 1,val(GoAngle)
```

You may wonder why the `goto` statement is buried here at the end of the chapter. It's simple: we don't want you to use it! At least, not much. For years professional programmers have known that too many goto's quickly create "spaghetti" code that is usually confusing. In fact, it has been mathematically proven that any program can be written with `if` and `loop` statements, without any `goto`'s at all. Here's how our example could be rewritten without a `goto`.

```
global GoAngle
GoAngle="0"
loop
   gettext "Enter direction (0-360)",GoAngle
while GoAngle>360 or GoAngle<0
setparameter 1,val(GoAngle)
```

As you can see, this example is actually simpler without the `goto` statement. Occasionally the `goto` statement does make it easier to write a program. Frankly we couldn't come up with an example of this, but if you do, feel free to use the `goto` statement. That's what it's there for.

Stopping the Program

The `stop` statement tells Panorama to stop the procedure immediately (see "STOP" on page 5796). If the current procedure is a subroutine, the original procedure is also stopped. If you want the current subroutine to stop but the original procedure to continue, use the `rtn` statement (see "Terminating a Subroutine in the Middle" on page 1392). The `rtn` statement also acts to stop the procedure if it is not being used as a subroutine.

Aborting a Program

Sometimes you may need to stop a program in the middle, before it has finished running. For example, suppose you make a mistake and create a loop that never stops. If that happens you need to abort the program. On the Macintosh you can do this by pressing **Command-Period**, on the PC by pressing **Control-Period**.

The ability to abort any program is normally an important safety valve, but you may have a procedure that should not be stopped in the middle with a job halfway done. For these types of cases Panorama allows you to disable the ability to abort during some or all of a procedure.

To disable the abort feature use the `disableabort` statement. To re-enable the ability to abort use the `enableabort` statement. (If you don't include an `enableabort` statement Panorama automatically re-enables aborting at the end of the procedure.)

The example below shows how these statements can be used. In this case there is no way that the new record can be added without being filled in by the lookup formulas. You either get all or nothing, but not a halfway done job.

```
disableabort
addrecord
Name=dialogName
Address=lookup("Contacts","Name",Name,Address,"",0)
City=lookup("Contacts","Name",Name,City,"",0)
State=lookup("Contacts","Name",Name,State,"",0)
Zip=lookup("Contacts","Name",Name,Zip,"",0)
enableabort
```

When using the `disableabort` statement you must be careful, especially when using loops. The procedure below will hang Panorama. The only way to stop the loop is to reboot the computer or do a force quit on Panorama (**Command-Shift-Option-Escape** on the Macintosh, **Control-Alt-Delete** on the PC).

```
disableabort
local i,tag
i=1
loop
   tag="<"+array(Text,i,¶)+">"
   stoploopif tag=""
   i=i+1
while forever
enableabort
```

While this example may look silly, it is easy to create an endless loop without realizing it.

Controlling the Abort Process

Sometimes you may want to allow a procedure to be aborted before it is finished, but in a controlled way. For example, suppose a procedure opens a progress window then loops over and over again to perform some operation (perhaps copying files or some other slow function). You might want to allow the procedure to be cancelled before it finishes, but you want to make sure that it closes the progress window even if the procedure is aborted. This can be done with the `info("abort")` function. This function returns a true or false value depending on whether the **Command-Period** (Mac), or **Control-Period** (PC) key has been pressed. Here is an example of a procedure that stops the loop if these keys are pressed.

```
disableabort
loop
   if info("abort")
      alert 1014,"Abort?"
      if info("dialogtrigger") contains "yes"
         stoploopif 1=1
      endif
   endif
   ...
   ... body of loop
   ...
while forever
...
... clean up after loop (close temporary windows, etc.)
...
enableabort
```

Since the procedure itself is testing to see if it should abort it is able to abort cleanly, finishing up any necessary tasks like closing progress windows, etc. This is much better than simply stopping the procedure at a semi-random spot.

Important note: The `info("abort")` function will only return true ONCE for each time the **Command-Period** (Mac), or **Control-Period** (PC) key combination is pressed. If you need to test it more than once (for example to cancel two nested loops you must copy the result into a variable and then test the variable. In other words, you generally don't want to have the `info("abort")` function in more than one spot within a loop.

Doing Nothing for a While

The `nop` statement (short for no operation) tells Panorama to do absolutely nothing! You can use the `nop` statement as a placeholder, or to delay for a short time. Here's an example of a procedure that will delay for a short time (probably less than a second).

```
loop
   nop
until 20
```

The exact amount of time delay depends on the speed of your computer. Here's an example that will delay exactly 10 seconds.

```
local startTime
startTime=now()
loop
   nop
until now()>startTime+10
```

Another use for the `nop` statement is to fool Panorama into not displaying a warning dialog. When used as the last statement in a procedure, or just before a `stop` statement, statements like `quit` and `close` will ask the user if they want to save changes. By adding a `nop` statement you can prevent this dialog from appearing.

```
if info("trigger") contains "Close w/o Save"
   close
   nop
   stop
endif
```

See "NOP" on page 5545 to learn more about doing nothing with the `nop` statement.

Building Subroutines On The Fly (The Execute Statement)

Subroutines are usually written in advance using the procedure editing window (see "Writing a Procedure from Scratch" on page 1352). However it is possible for a procedure to construct a subroutine "on-the-fly" and then immediately call (execute) that subroutine. This magical trick is performed by a special statement called **execute**. The `execute` statement is followed by a formula that calculates the text of the subroutine to be called.

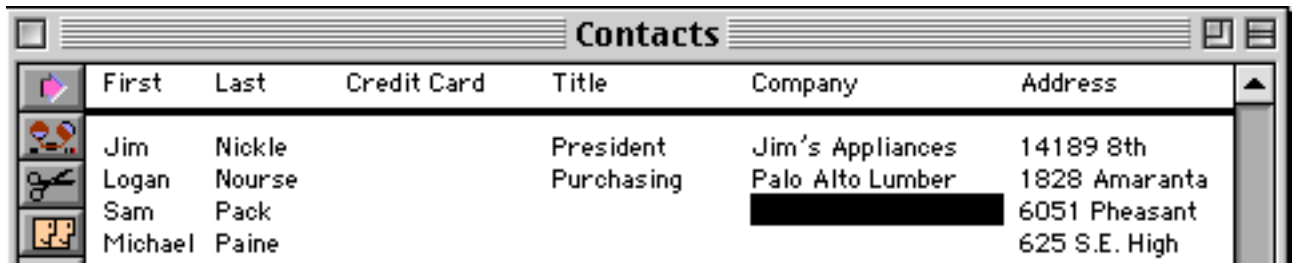
```
execute <formula>
```

Here is an example procedure that uses the execute statement to “ditto” the value in the cell above the current cell. Basically, this procedure moves up a line, grabs the cell, then moves down a line and assigns the new value to the current cell.

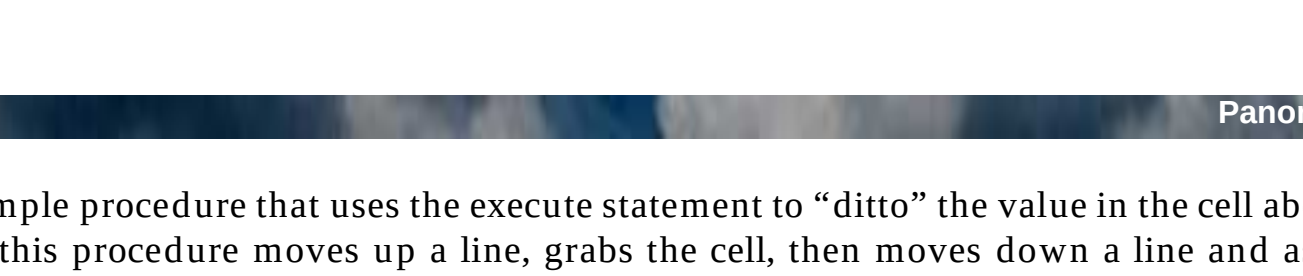
```
local dittoValue,dittoField
dittoField=info("FieldName")
uprecord
if stopped rtn endif
dittoValue=«»
downrecord
execute dittoField+"={"+dittoValue+"}"
```

The first line (`local dittoValue,dittoField`) simply sets up the two local variables we will need (see “Variables” on page 1370).

The second line (`dittoField=info("FieldName")`) gets the name of the current field and places it in the dittoField variable. In the example above this value will be Company.



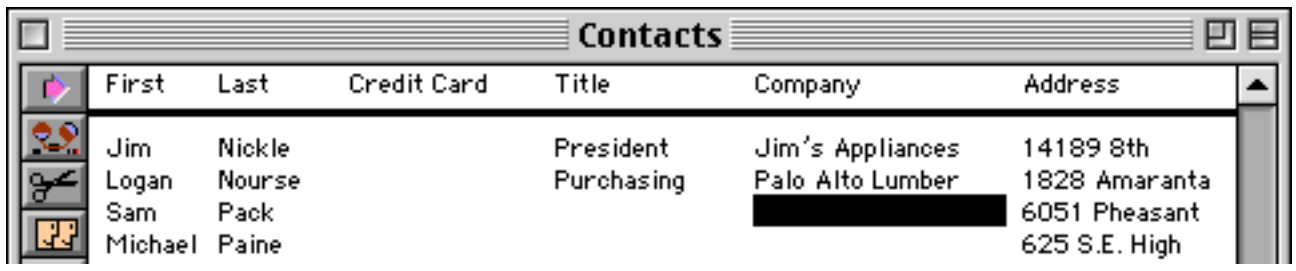
The third line (`uprecord`) moves up one record, like this.



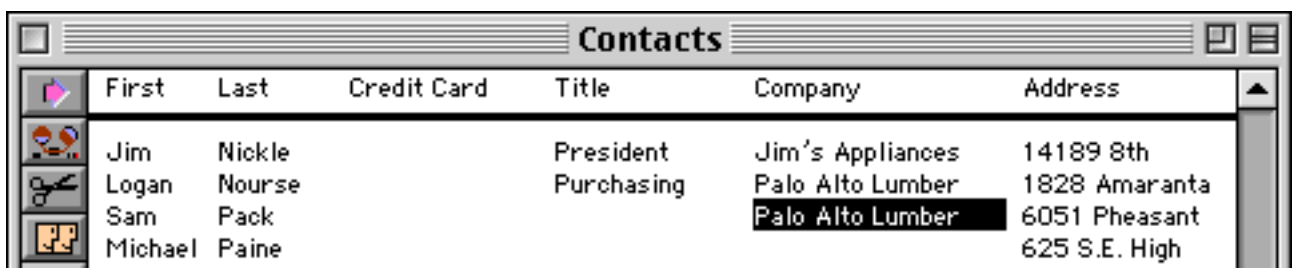
The fourth line (`if stopped rtn endif`) checks to see if we were already on the top line of the database, and if so, stops the procedure.

The fifth line (`dittoValue=˙¨`) grabs the value in the current cell, in this case Palo Alto Lumber (see “Using the Current Field” on page 1197).

The sixth line (`downrecord`) moves back down one record to the original position.



The seventh line (`execute dittoField+"={"+dittoValue+"}"`) assigns the value in dittoValue to the cell.



Let's take a close look at how the `execute` statement did its job. Here's the formula it used.

```
dittoField+"={"+dittoValue+"}"
```

Now we know that dittoField is Company, and dittoValue is Palo Alto Lumber. So the result of this formula is this —

```
Company={Palo Alto Lumber}
```

Now this is a valid assignment statement (see "Assignment Statements" on page 1368) that assigns a value into the Company field! (In case you have forgotten, `{` and `}` are alternate quote characters that can be used instead of `"`. See "Constants" on page 1195 for a complete list of quote characters.) This line is a completely valid subroutine all by itself. So Panorama goes ahead and executes this custom subroutine which causes the company name to be filled in.

If we move one column to the left, the formula will generate a different custom subroutine.

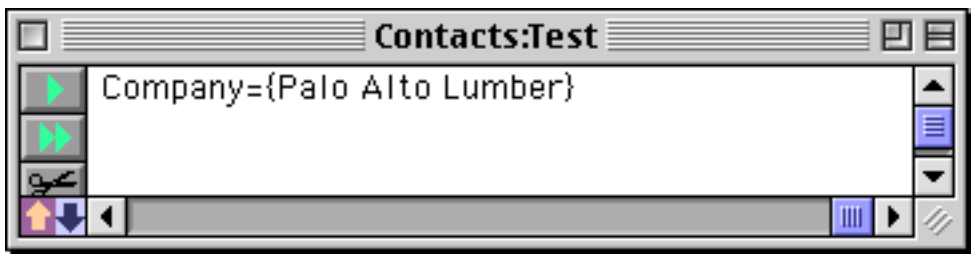
```
Title={Purchasing}
```

The ditto procedure will work for any text field. (It could be revised to work with numeric and date fields as well with some extra work with the `str(` and `datepattern(` functions.)

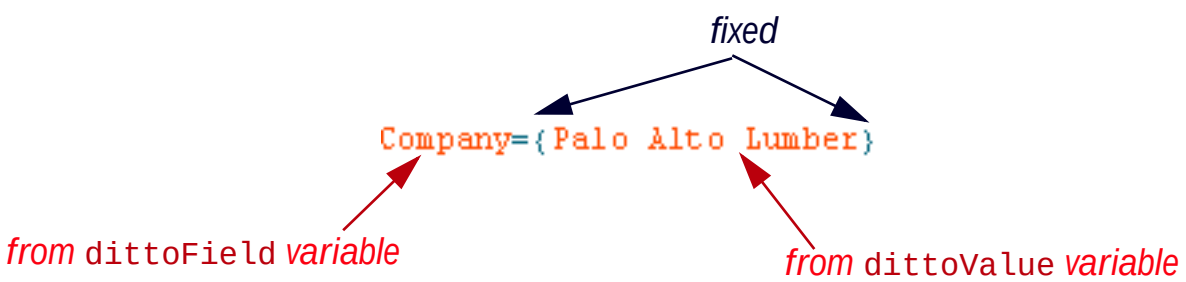
Tips for On-The-Fly Program Writing

Writing a regular program can be tricky enough. Attempting to create a procedure on the fly can turn into a quagmire if you don't take a systematic approach. Careful planning will insure success.

Before you start attempting to write a formula make sure you have a good idea of what the final subroutine will look like. If the subroutine is more than one or two lines it's a good idea to create a "mock up" using the procedure editor. This should be a fully running procedure hard coded for specific data. Here's an example mock up for the ditto cell procedure described in the previous section.



Once the mock up is debugged you can start converting it into a formula. Start by figuring out which sections of the procedure are fixed and which will change, and where the data for the changeable parts will come from.



Now we'll start to assemble the formula. The first part is changeable, and comes from the dittoField variable. So the first part of the formula is simply

```
dittoField
```

Next is a fixed component ={. We want this to appear exactly like this in the final subroutine, so we quote it and concatenate it to the first section. Since we're using { and } for quotes in the final procedure we'll need to use a different kind of quote here. We're using "" here but we also could have used smart quotes (see "Constants" on page 1195 for a list of different types of quotes).

```
dittoField+"={"
```

The next component is changeable, and comes from the dittoValue variable, so we'll add that on next.

```
dittoField+"={"+dittoValue
```

Finally we'll add on the closing } quote. Again this must be enclosed in a different type of quote.

```
dittoField+"={"+dittoValue+"}"
```

The trickiest part is often the nested quotes. Each type of quote must be nested in another type. An alternative technique would be to use the chr(function (see "CHR(" on page 5099) to generate the { and } quotes, like this.

```
dittoField+"="+chr(123)+dittoValue+chr(125)
```

The values 123 and 125 must be looked up in an ASCII chart. A partial ASCII chart is shown below. (Use the **ASCII Wizard** to see complete ASCII chart — see "The ASCII Chart Wizard" on page 1232).

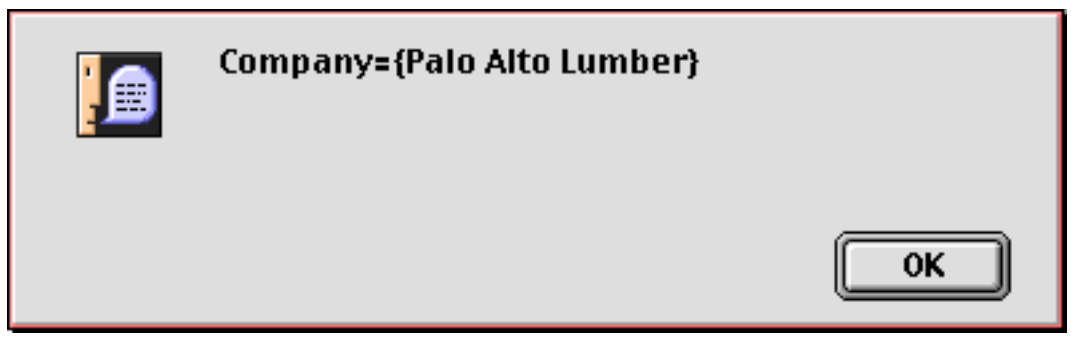
0	1	2	3 enter	4	5	6	7	8	9 tab	10	11	12	13 return	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33 !	34 "	35 #	36 $	37 %	38 &	39 '	40 (	41)	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 [	92 \	93]	94 ^	95 _
96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 ⌂

As you can see, a regular " quote can be generated with chr(34).

If you are having a difficult time getting your execute formula to work properly a good tip is to temporarily replace the execute statement with a message statement, like this.

```
local dittoValue,dittoField
dittoField=info("FieldName")
uprecord
if stopped rtn endif
dittoValue=«»
downrecord
message dittoField+"={"+dittoValue+"}"
```

When the procedure runs it displays the actual subroutine that it was about to execute.



Often at this point the error is obvious and you can easily go back and check it. If the subroutine is too long to fit in the dialog you can copy it into the clipboard instead.

```
local dittoValue,dittoField
dittoField=info("FieldName")
uprecord
if stopped rtn endif
dittoValue=«»
downrecord
clipboard=dittoField+"={"+dittoValue+"}"
```

After the program runs you can paste the generated subroutine into a text editor and carefully examine it.

If the execute was not the last statement in the procedure you will probably want to place a `stop` statement (see “Stopping the Program” on page 1397) after the `message` or `clipboard=` statement.

Execute and Local Variables

The subroutine generated “on-the-fly” by the execute statement is not part of the current procedure but is a separate procedure on it’s own (although it doesn’t have a name and disappears as soon as it is finished). This means that the local variables in the original procedure become dormant while the “on-the-fly” subroutine is running, and that the “on-the-fly” subroutine cannot use or modify any of those local variables (see “Variable Accessibility” on page 1373). (It can have its own local variables, however.)

To get around this limitation a new statement was added to Panorama in 2004 — the `ExecuteLocal` statement. This statement is identical to the `execute` statement except for the fact that it uses the local variables of the enclosing procedure instead of its own separate local variables. Here is an example:

```
local x
x="Hello"
executelocal {message x}
```

This procedure will display the word “hello”, even though the x variable is not defined within the executelocal part of the procedure.

Using Execute to Process Arrays

The execute statement can be very handy for working with arrays. For example, suppose you have an comma separated text array (see “Text Arrays” on page 1236) named Numbers that contains a series of numbers like this —

```
78,173,9,32,201,12,82,376,249
```

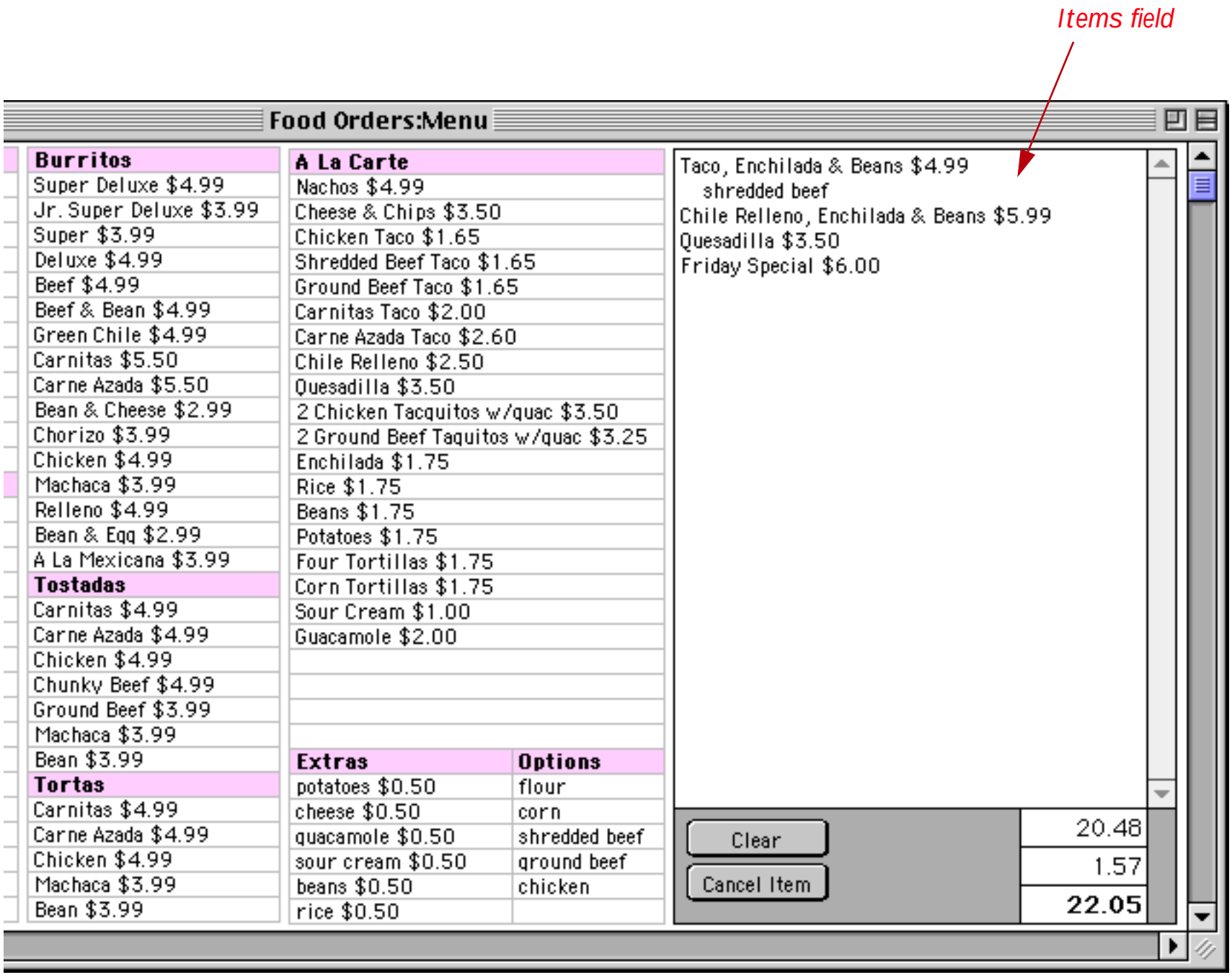
and you’d like to add up the numbers and place the sum in a field named Total. With the `execute` statement this can be done with a single line of code!

```
execute "Total="+replace(Numbers,",","+")
```

The `replace(` function (see "REPLACE(" on page 5665) converts the commas into plus symbols. The generated subroutine is —

```
Total=78+173+9+32+201+12+82+376+249
```

Not only is this code simple, it is very fast. Here is a slightly more complex example that uses a similar technique to calculate the total of an invoice. The invoice looks like this. The big area on the right is a field named Items.



As you can see, if a line in the Items field contains a price it is always after a dollar ($) sign. We can use this fact to quickly calculate the total.

```
fileglobal linecalc
if Items≠""
    arrayfilter Items,linecalc,¶,array(import(),2,"$")
    linecalc=arraystrip(linecalc,¶)
    linecalc=replace(linecalc,¶,"+")
    execute "Subtotal="+linecalc
else
    Subtotal=0
endif
Tax=(Subtotal*7.75)/100
Total=Subtotal+Tax
```

Lines 3 thru 6 are the guts of this procedure.

Line 3 (`arrayfilter Items,linecalc,¶,array(import(),2,"$")`) uses the `arrayfilter` function (see "ARRAYFILTER" on page 5045) to scan the Items array and strip out only the prices (after the `$` symbol). The intermediate result in linecalc will be something like this.

```
4.99

5.99
3.50
6.00
```

Line 4 (`linecalc=arraystrip(linecalc,¶)`) strips out any extra carriage returns. The result looks now looks something like this.

```
4.99
5.99
3.50
6.00
```

Line 5 (`linecalc=replace(linecalc,¶,"+")`) converts the carriage returns (¶ - see "Special Characters" on page 1202) into plus symbols.

```
4.99+5.99+3.50+6.00
```

Line 6 (`execute "Subtotal="+linecalc`) is just like the previous example and calculates the total.

Do It Yourself Data Merge

Auto-wrap text objects allow you to merge data and formulas into a template to be displayed on the screen or printed in a report (see "Displaying Data in Auto-Wrap Text" on page 595). In this section we'll describe a technique using the `execute` statement that allows a procedure to do the same thing. For example, suppose that you have a template in a variable named Letter that contains this text.

```
«Name»
«Address»
«City», «State» «Zip»

Dear «Name»,

Your order (reference «Order») has been shipped. You should expect it to arrive within the
next three to five days.

Sincerely,

Acme Widgets
```

If we can turn this template into a formula we can evaluate it with the `execute` statement. Let's see how this can be done. The first step is to create two arrays. The first array, FieldNames, will contain a list of the fields, like this.

```
«Name»
«Address»
«City»
«State»
«Zip»
«Order»
```

The second array, FieldFormulas, will also contain a list of fields, but modified so that they can be included as part of a formula.

```
"}+«Name»+{"
"}+«Address»+{"
"}+«City»+{"
"}+«State»+{"
"}+«Zip»+{"
"}+«Order»+{"
```

Here's the code that can generate these two arrays (see "DBINFO(" on page 5150 and "ARRAYFILTER" on page 5045).

```
local FieldNames,FieldFormulas
FieldNames=dbinfo("fields","")
arrayfilter FieldNames,FieldNames,¶,"«"+import()+"»"
arrayfilter FieldNames,FieldFormulas,¶,“"}+”+import()+“+{""
```

Now we can use the `replacemultiple(` function (see "REPLACEMULTIPLE(" on page 5667) to transform the template into a formula.

```
local MergeFormula
MergeFormula=”{“+replacemultiple(Letter,FieldNames,FieldFormulas)+”}”
```

After the transformation the template has been turned into a valid Panorama formula.

```
{}+«Name»+{
}+«Address»+{
}+«City»+{, }+«State»+{ }+«Zip»+{

Dear }+«Name»+{,

Your order (reference }+«Order»+{) has been shipped. You should expect it to arrive within
the next three to five days.

Sincerely,

Acme Widgets}
```

Here's the complete procedure.

```
local FieldNames,FieldFormulas,MergeFormula
fileglobal FinalLetter
FieldNames=dbinfo("fields","")
arrayfilter FieldNames,FieldNames,¶,"«"+import()+"»"
arrayfilter FieldNames,FieldFormulas,¶,“"}+”+import()+“+{""
MergeFormula=”{“+replacemultiple(Letter,FieldNames,FieldFormulas)+”}”
execute “FinalLetter=”+MergeFormula
```

This procedure turns the template into a final letter with all of the data merged in as requested in the template. You could use a procedure like this to generate custom e-mail or as part of a CGI for a web server (in which case the template would contain HTML).

On-The-Fly Subroutine Error Checking

Just as with any other procedure, it's possible for an on-the-fly subroutine to contain grammar errors. If the subroutine contains a grammar error (for example `a++b`) the procedure will stop and an alert displaying an error message will appear. If you don't want that to happen you can place an `if error` statement after the `execute` statement. This example procedure executes whatever is in the PreFlight field and ignores any grammar error that occurs.

```
execute PreFlight
if error
   nop
endif
```

The program can find out what the problem was with the `info("error")` function (see "INFO("ERROR")" on page 5373). To find out exactly where the problem occurred check the special global variables ExecuteErrorStart and ExecuteErrorEnd. These variables contain numbers telling where within the subroutine Panorama thinks the error occurred. If the subroutine was generated with a complex formula it may be difficult to relate these numbers back to the formula that generated the subroutine.

The `if error` statement after the `execute` only catches grammar errors, not run-time errors. If you want to catch run-time errors (for example field or variable does not exist or numeric when text expected) you must build if error statements into the generated subroutine itself (see "Error Handling with if error" on page 1381).

Building Parameters on the Fly (Parameters in a Variable)

Many statements accept one or more formulas as parameters. These formulas are usually fixed at the time the procedure is written. For example the statement below has four parameters:

```
arrayfilter a,b,¶,str(seq())+": "+import()
```

Most statements also allow you to store the text of the formula itself in a variable. To do this, use `@variable` or `@¨variable¨` instead of the formula itself. For example, the previous example could be rewritten using this technique:

```
local myFormula
myFormula={str(seq())+": "+import()}
arrayfilter a,b,¶,@myFormula
```

In this example there is no advantage to this technique, but the power is that now you can change the formula "on the fly". Of course you could also do this with the `Execute` statement, like this:

```
local myFormula
myFormula={str(seq())+": "+import()}
execute {arrayfilter a,b,¶,}+myFormula
```

Using the @ technique has two benefits over using the `Execute` statement. First of all, it can be quite a bit faster. Secondly, it makes it much easier to work with local variables.

Warning: When using this technique, take extra care in checking your formulas. Panorama often cannot accurately report the source of a formula error when this technique is used.

Catching Program Errors (Especially for Web and other Server Applications)

The `if error` statement (see "Error Handling with if error" on page 1381) gives the programmer complete control of what happens when an error occurs at a specific point in the program. However it requires the programmer to explicitly handle every error that may occur. The `OnError` statement can be used to catch all errors that are not trapped by `if error` statements. This has two benefits when Panorama is used as part of a web server. First it allows the programmer to easily eliminate all error alert dialogs. This is very important for server applications because an alert dialog requires human intervention to get the server going again. Secondly, it makes it easy to build a log of errors.

The `OnError` statement has one parameter: a text string that contains one or more Panorama statements to be executed when an error occurs. Notice that this is not the name of a procedure, but the actual statements themselves (as a string of text). This is similar to the `execute` statement (see “Building Subroutines On The Fly (The Execute Statement)” on page 1399). Once an error has occurred these statements will run. Within these statements you can use the `info("error")` function to find out what the error was, if necessary.

The effect of the `OnError` statement ends when the main procedure stops running. In other words, `OnError` isn't a permanent error handler — you must specify it for each procedure you wish to have error trapping. If you plan to use `OnError`, it is probably best to put it in the first line of any procedure that needs error trapping. If you are going to use the same statements with `OnError` in several different procedures, you may want to set up the statements in a variable in your .Initialize procedure, then use that variable as the parameter to `OnError`.

It's important to consider the possible environment that may exist when an error is created. Depending on the flow of your main procedure, Panorama may not be in the same window or even in the same database. Your `OnError` program should generally not make any assumptions about what windows or databases will be active or available when the error occurs.

Here is an example of how `OnError` could be used in a CGI (web server) application. In this example if there is an error Panorama will return an error message to the web server and also log the error along with the date and time.

```
global cgiResult,errorLog
errorLog=errorLog                          /* make sure errorLog exists */
if error
   errorLog=""                               /* initialize errorLog */
endif
onerror {cgiResult="Panorama Error: "+info("error") }+
        {errorLog=sandwich("",errorLog,¶)+}+
        {datepattern(today(),"DD/MM/YYYY ")+}+
        {timepattern(now(),"hh:mm:ss")+}+
        {info("error")}

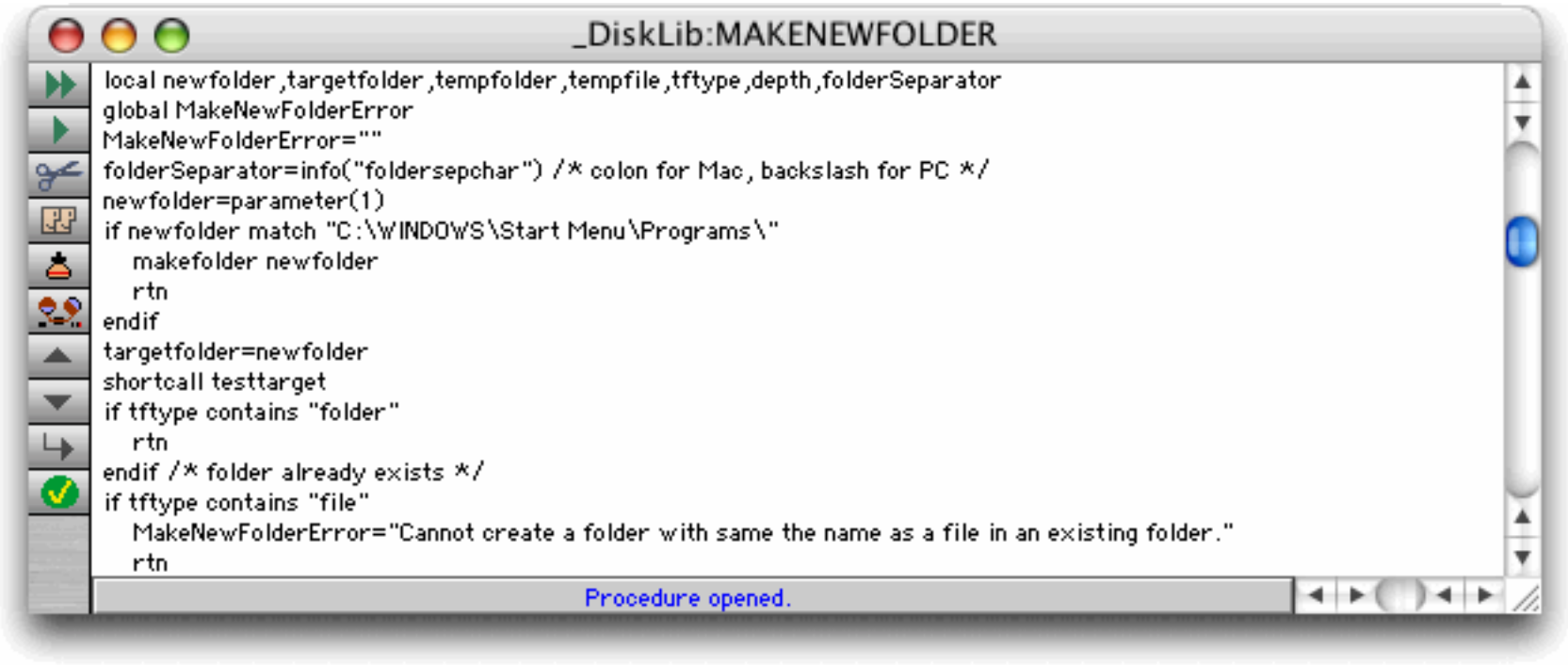
/* error logging is set up, now we can continue with our tasks */
...
... rest of this procedure
...
```

Custom Statements

Panorama comes with hundreds of ready to use built-in statements (see “Programming Reference Wizard” on page 1361), but you aren’t limited to these built-in statements. If you don’t find the statement you need you can build your own! If you are planning on using a particular sequence of steps frequently then it might pay to create a custom statement that you can use in any database.

As you might have guessed from the phrase “sequence of steps” in the previous paragraph, custom statements are very similar to subroutines. In fact, custom statements actually are Panorama subroutines, written using Panorama’s standard procedure editor and the Panorama programming language. The only difference is that these subroutines are placed in a special location, where Panorama automatically loads them so. As it loads the database containing these special subroutines, Panorama also adds them to the programming language so that they are always available to procedures in any database.

To illustrate this, consider the subroutine shown below, which will make a new folder. This procedure is called MAKENEWFOLDER and is contained in the _DiskLib database.



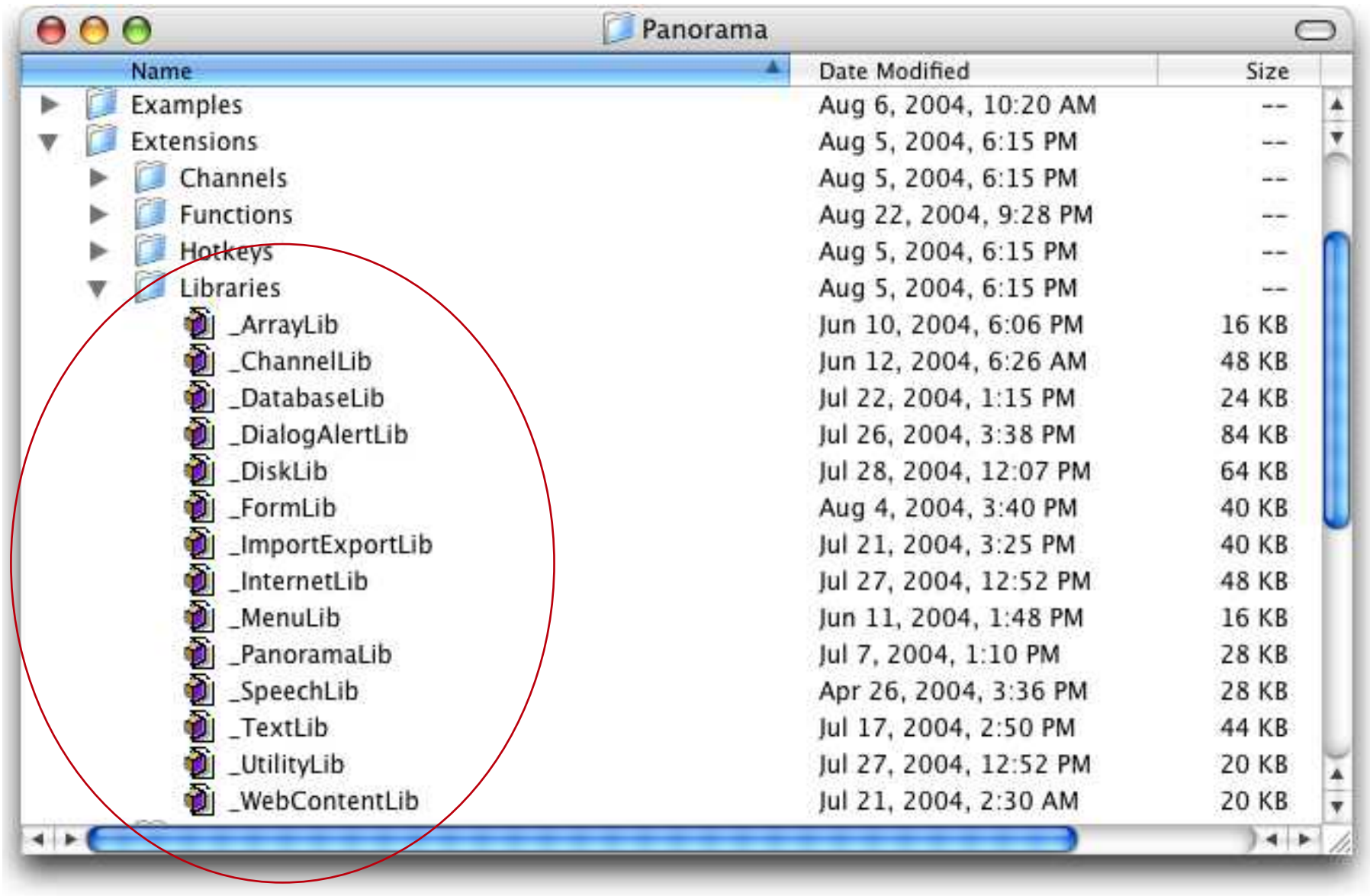
Even without the ability to create custom statements you can still call this subroutine from another database using the line shown below. (see “Calling a Subroutine in Another Database” on page 1392).

```
farcall "_DiskLib","MAKENEWFOLDER","My Drive:My Documents:MyImages:Zack:"
```

This technique has some drawbacks — you have to remember that this subroutine is in the _DiskLib database, you have to make sure that the _DiskLib database has actually been opened, and you have to use the exact capitalization MAKENEWFOLDER because MakeNewFolder or makenewfolder won’t work. Converting this procedure into a custom statement fixes these drawbacks. Here is the same subroutine used as a custom statement:

```
MakeNewFolder "My Drive:My Documents:MyImages:Zack:"
```

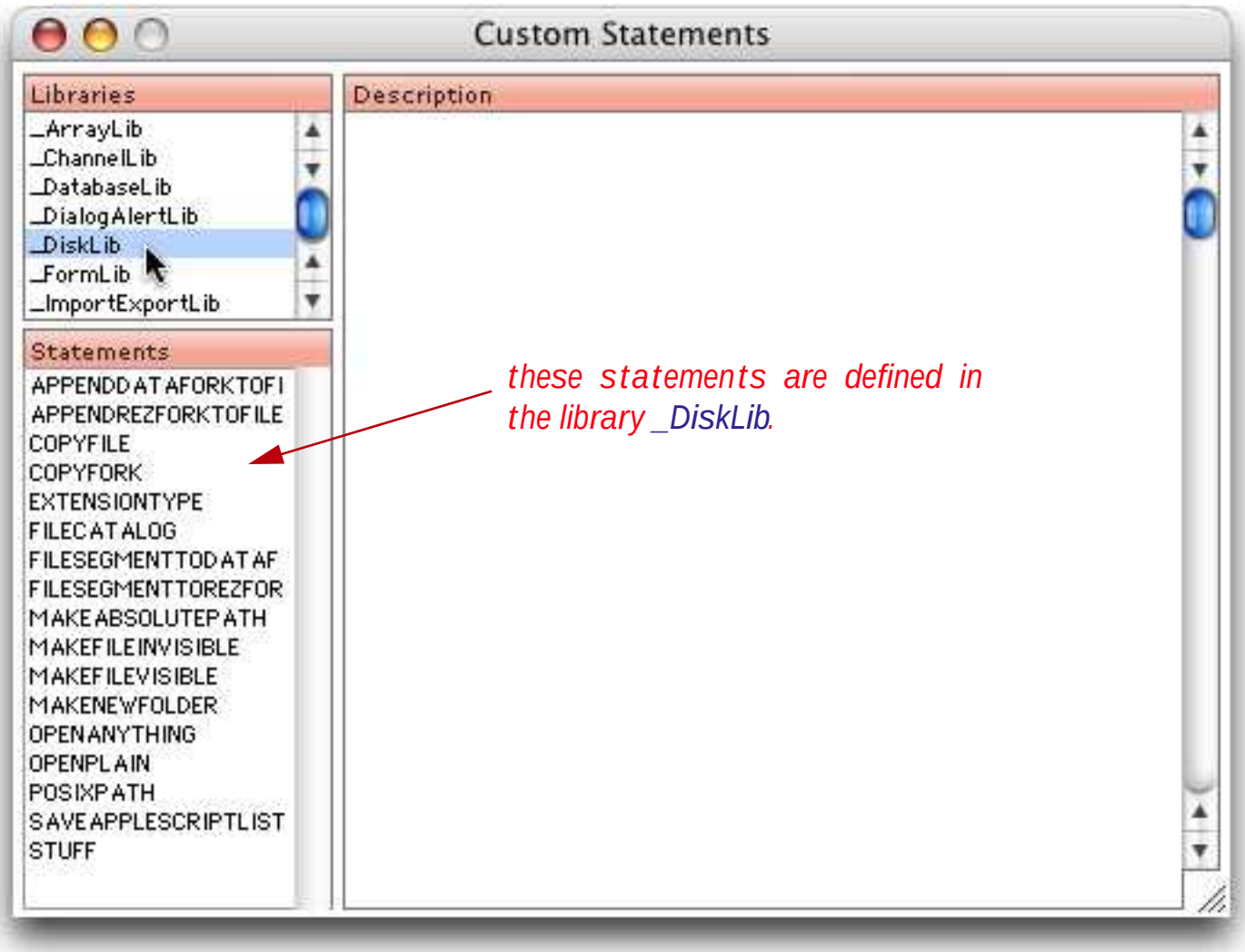
As illustrated in this example, the procedures used in custom statements are stored in one or more Panorama databases. These databases are called **procedure libraries**. These library databases are normally kept in the Extensions:Libraries folder within the Panorama folder. (Note: All of the databases in the folder shown below begin with _ and end with Lib, but that is not necessary. You can use any name you like as long as the database is placed within this folder.)



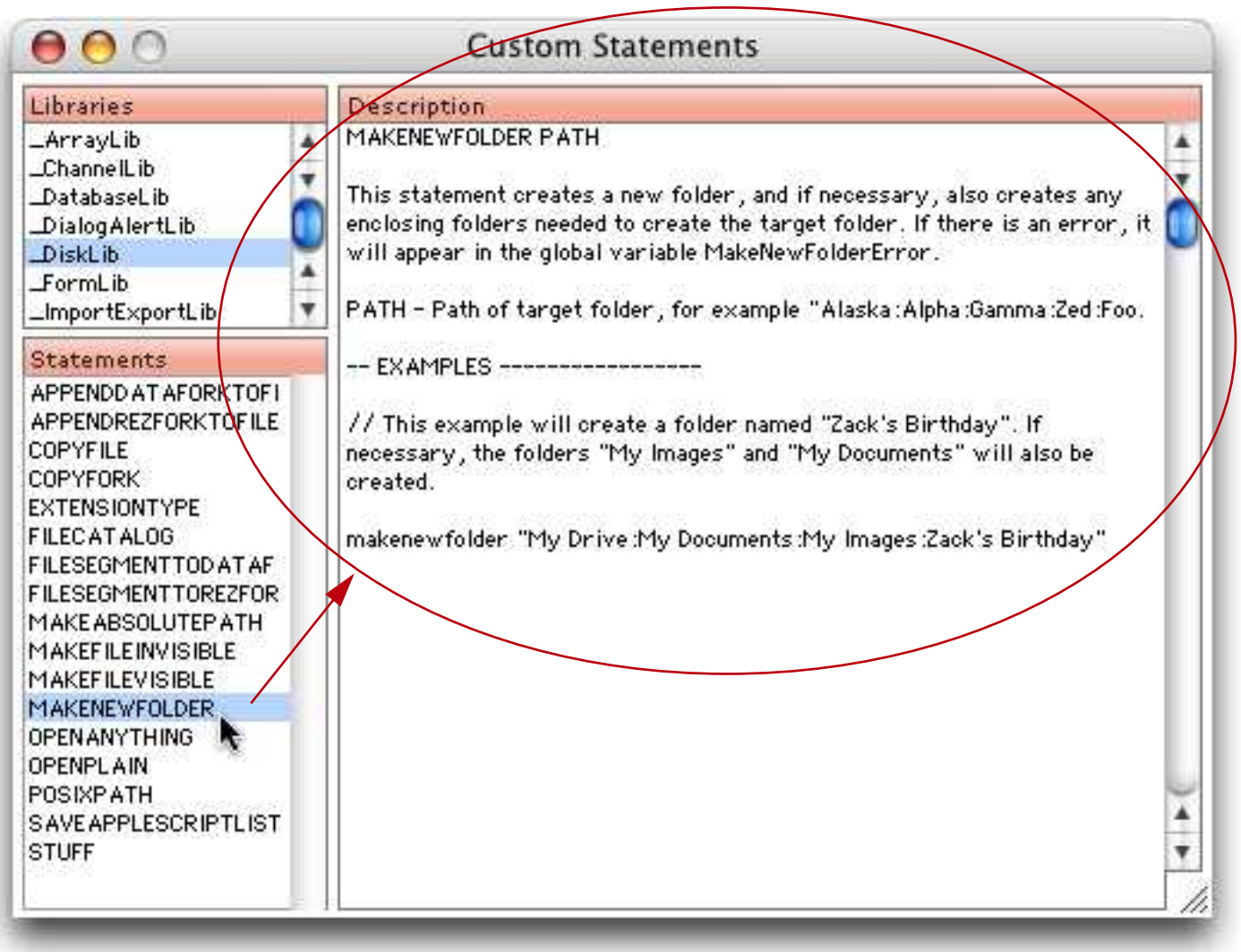
If a database is placed in this folder Panorama will automatically open it when Panorama launches (the database is opened secretly, with no windows). Panorama will then scan the database and register each procedure in the database as a custom statement. (Actually, only procedures with names that are all upper case letters will be registered, for example ZIP or ZAP but not Zip or ZAP23.)

The Custom Statements Wizard

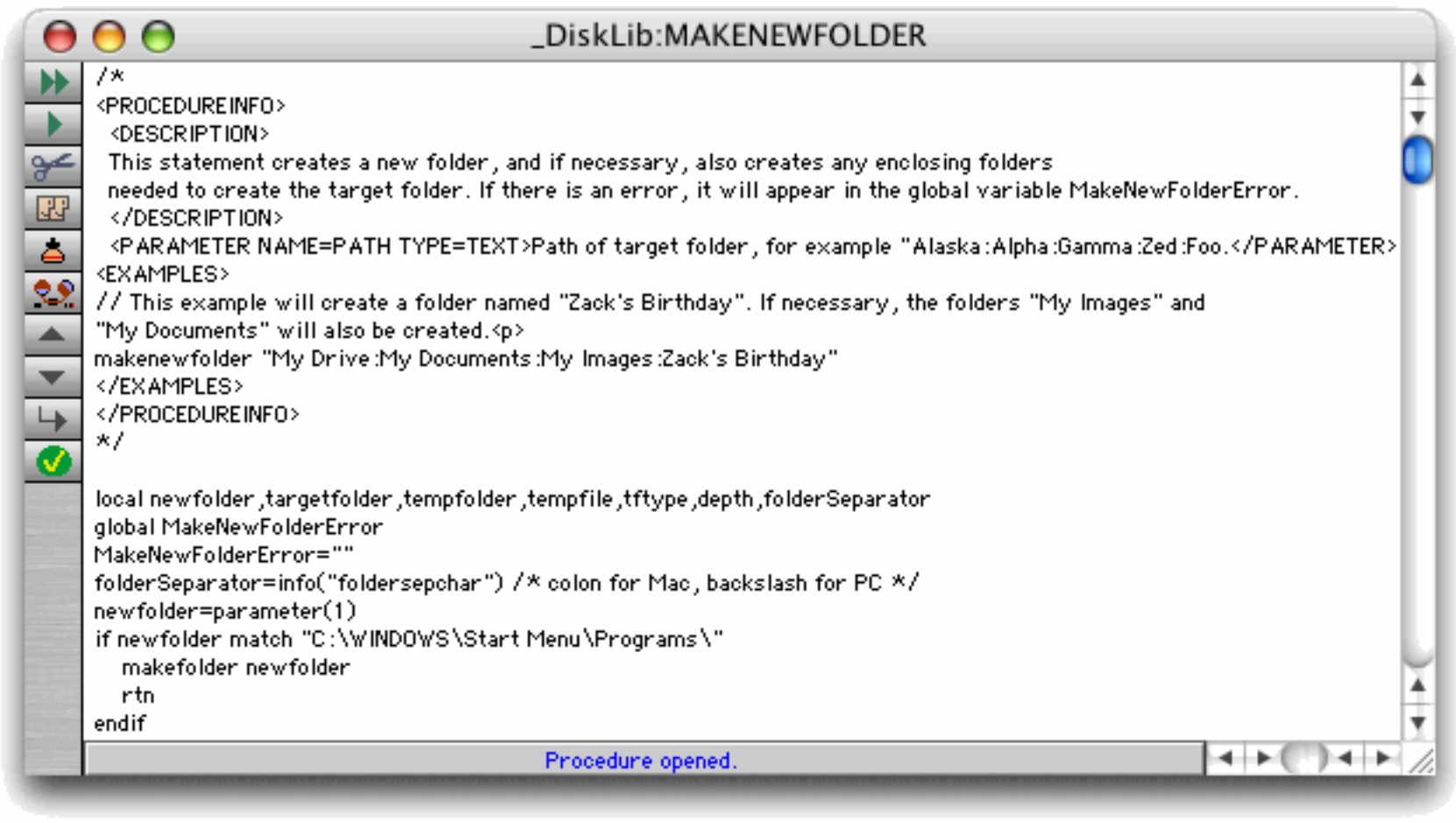
To learn about the custom library statements that are available for your use you can use the **Custom Statement** wizard. When you open this wizard it displays a list of the libraries that are currently available (on the top left). When you click on a library in this list, the wizard displays a list of statements defined in that library (on the bottom left).



You can click on a specific statement to see a description of that statement and its parameters.



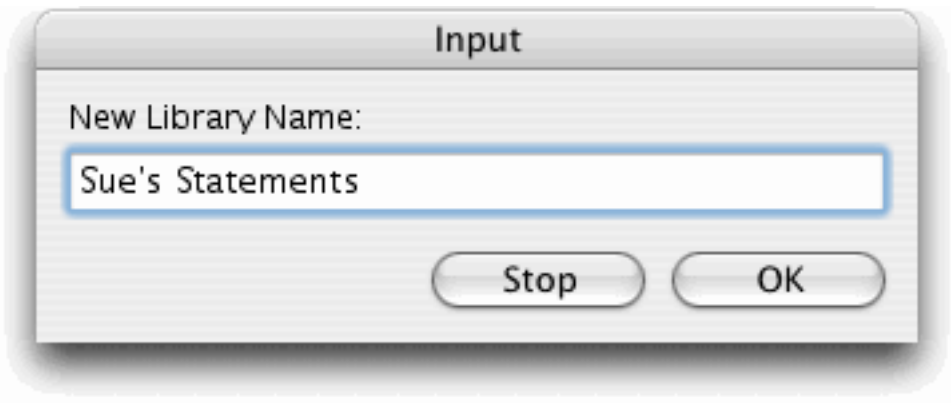
If you want to see (or even edit) the actual code of this statement, double click on the statement name (in the list of statements). The actual code for this statement will appear in a separate window.



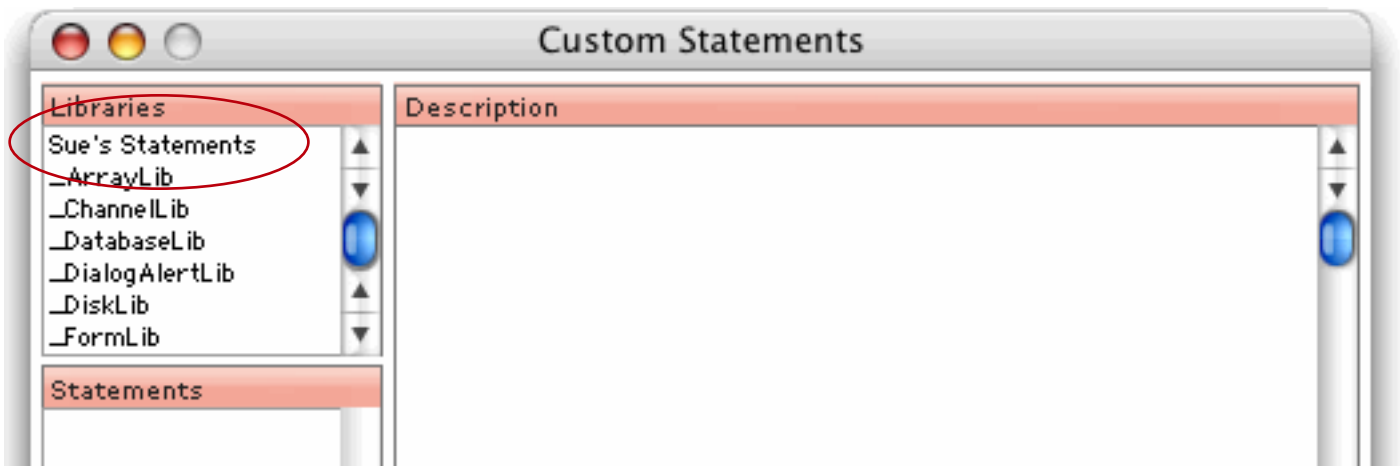
If you make any changes to a statement, please do so carefully (especially for statements that are supplied with Panorama). When you close this window the changes are automatically saved.

Creating Your Own Custom Statement Library

Panorama comes with about a dozen pre-built custom libraries that you can begin using right away, but you can also use the **Custom Statements** wizard to create your own. To do so, choose **New Library** from the Library menu. (You may have noticed that all of the libraries that come with Panorama have names that begin with _ and end with Lib, but that is not necessary.)



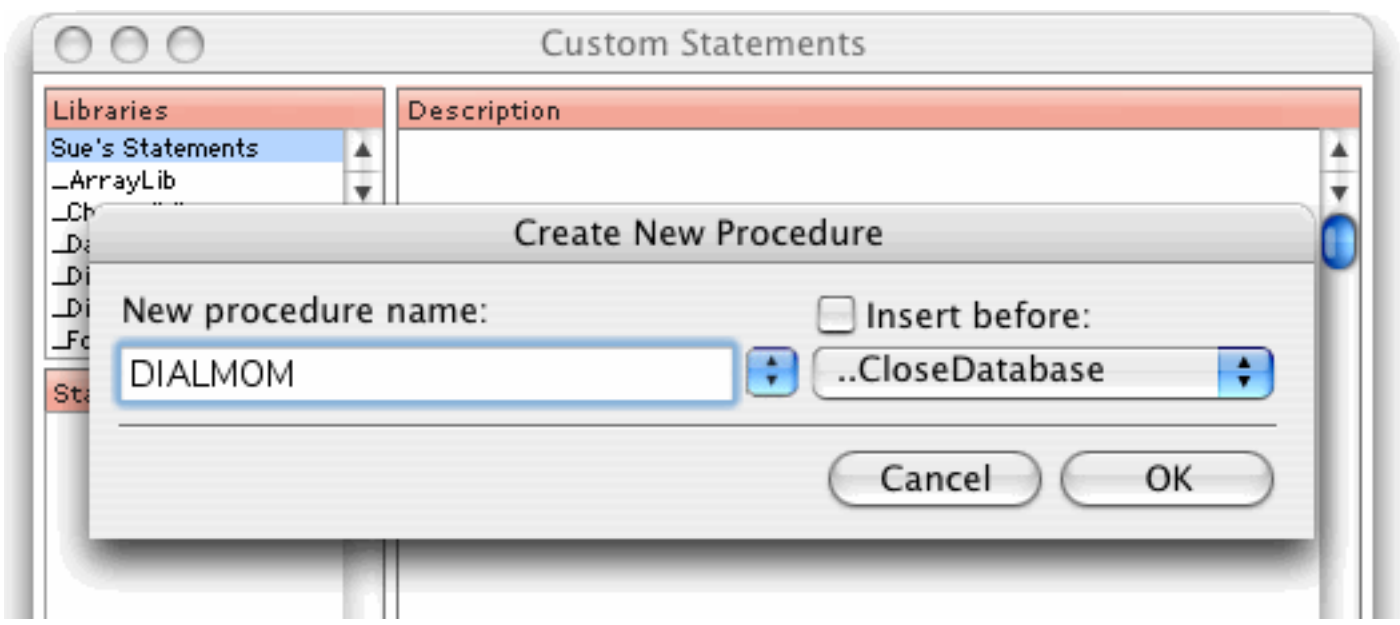
Once you've entered the name, press **OK**. Your new library will appear in the list of libraries in the upper left hand corner.



If you click on your new library you'll see that it doesn't contain any statements yet.

Creating a New Custom Statement

To create a new custom statement, first make sure that you have clicked on the name of the library you want the statement to be created in. ((Note: We do not recommend that you add new statements to the libraries that come with Panorama. If you do so, your new statements will be destroyed the next time you update Panorama. Instead, make sure to add your new statements to your own libraries.) Then choose **New Statement** from the Statement menu, and type in a name for your new statement.



The name must contain only the letters A through Z (uppercase only). You cannot include spaces, numbers, lower case letters, or any other kind of character except for upper case letters. (Note: The **New Statement** command won't stop you from entering these characters, but if you do, you cannot use this procedure as a custom statement.) Here are some examples of procedure names that may be used for custom statements.

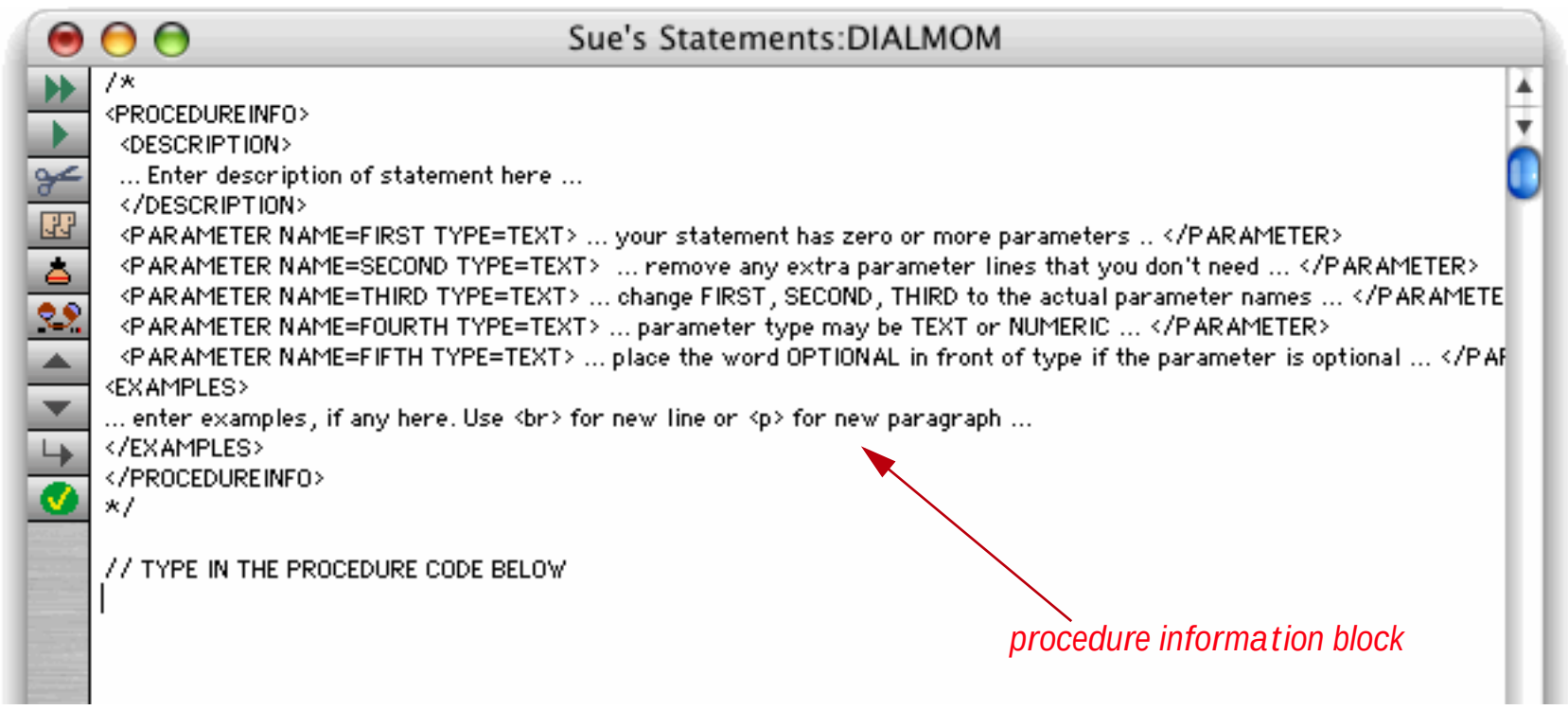
```
PARSENAME
SPLITINPUT
WRITELOG
```

Here are some names that *may not* be used for custom statements. (If your library database contains procedures with these names those procedures will be ignored when the library is initialized.)

```
ParseName
SPLIT INPUT
CARD#
```

To avoid conflicts with future versions of Panorama it's best to choose names that are unlikely to be used by Panorama itself. For example, `MYWRITELOG` or `ACCOUNTINGLOG` might be a safer choice than `WRITELOG`. If a future version of Panorama includes a statement with the same name as your statement then your statement will stop working (you'll have to rename your statement and locate and change every place that you have used it).

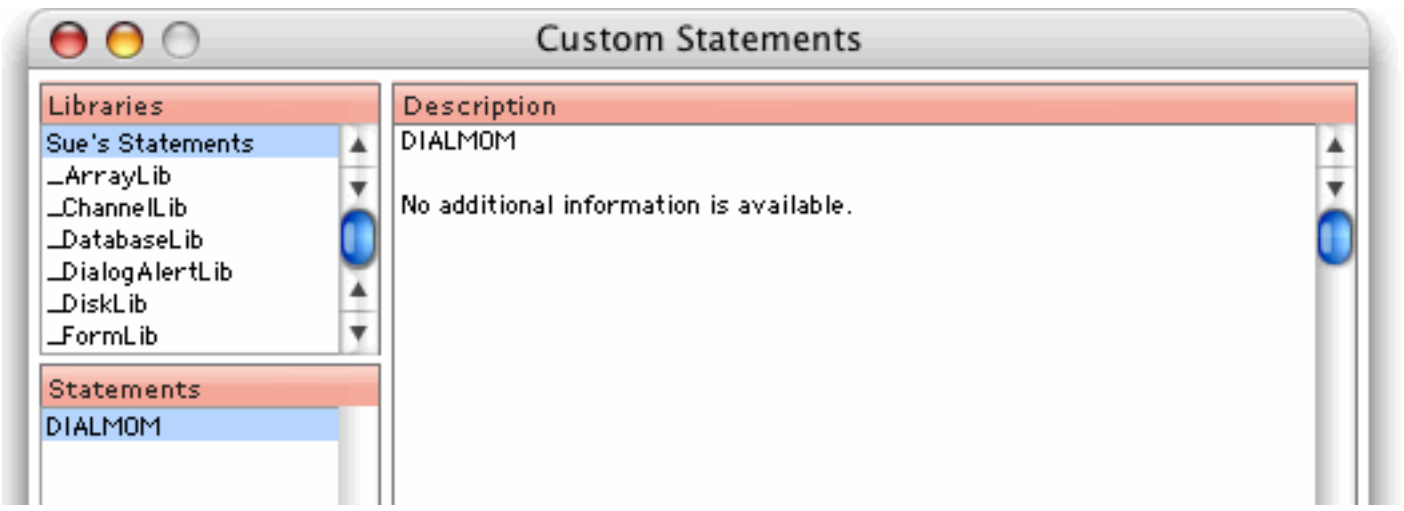
OK, back to the action. When you press the **OK** button, the wizard will create a brand new procedure with the specified name.



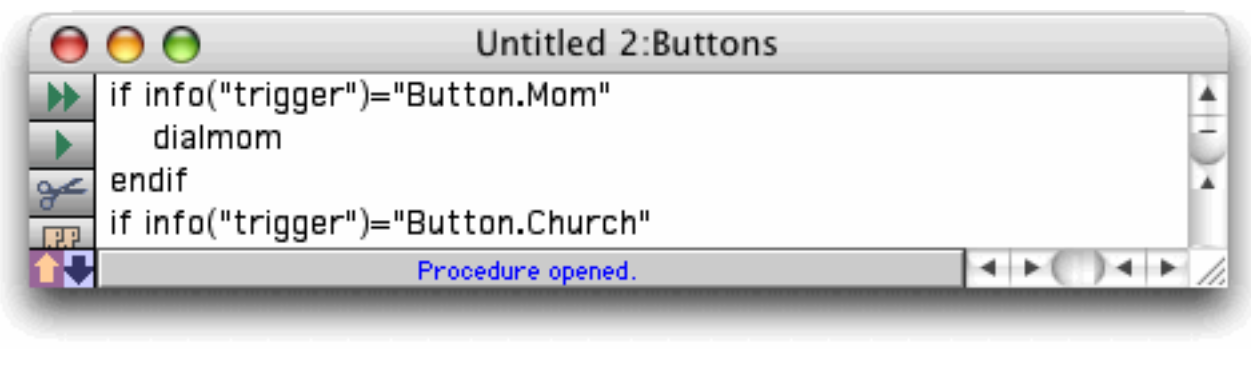
Although it's not required, it's a good idea to include a **procedure information block** in each procedure that will become a statement. In fact this is such a good idea that when you create a new statement a "typical" procedure information block is created for you, as shown above. However, for this simple example we won't use the information block, so erase it and type in the actual code of the procedure.



This simple statement simply dials Mom's phone number. However, the statement isn't quite ready to use. The final step is to "register" the statement with Panorama. To do this, click back to **Custom Statements** wizard and choose **Register New/Modified Statements** from the Statements menu. The new statement will appear in the list of statements, and is now ready to use. (Notice that in the description area it says No additional information is available. This is because there is no **Procedure Information Block** for this statement.)



To use this new statement, simply type it into any procedure in any database. (Notice that like any other statement, the `dialmom` statement is not sensitive, you can use `DIALMOM`, `dialmom`, `DialMom`, `diALmOM` or any other combination of upper and lower case letters.)



That's all there is to it. Later, if you quit and then re-launch Panorama, this statement will automatically be registered as part of Panorama's initialization process. From now on the `dialmom` statement is always available whenever you need it. (Of course you can always modify and/or delete it later if you wish, but be careful, as this will break any procedures that use your new statement.)

Setting Up a Procedure Information Block

Although it's not required, it's a good idea to include a **procedure information block** in each procedure that will become a statement. This block contains descriptive information about the statement and its parameters. This serves two purposes:

1) It allows the statement to be "self-documenting" through the Custom Statements wizard.

2) It allows Panorama to perform some error checking on the parameters of your statement, making debugging easier. For example, when this statement is used in a procedure Panorama can check to make sure that the proper number of parameters is supplied.

The procedure information block should be included in a comment within your procedure (usually at the top) and looks something like this.

```
/*
 <PROCEDUREINFO>
 <DESCRIPTION>
 --- put description of procedure here ---
 </DESCRIPTION>
 <PARAMETER NAME=name TYPE=TEXT>--- put description of parameter here</PARAMETER>
 <PARAMETER NAME=name TYPE=TEXT>--- put description of parameter here</PARAMETER>
 <PARAMETER NAME=name TYPE=TEXT>--- put description of parameter here</PARAMETER>
 <EXAMPLES>
 --- put examples (if any) here
 </EXAMPLES>
 </PROCEDUREINFO>
*/
```

When you create a new statement a “typical” procedure information block is created for you. You’ll need to modify this block to reflect the actual structure of your new statement. The most important part is to set up the parameter list to have the correct number of parameters.

Each parameter has a name, type and description.

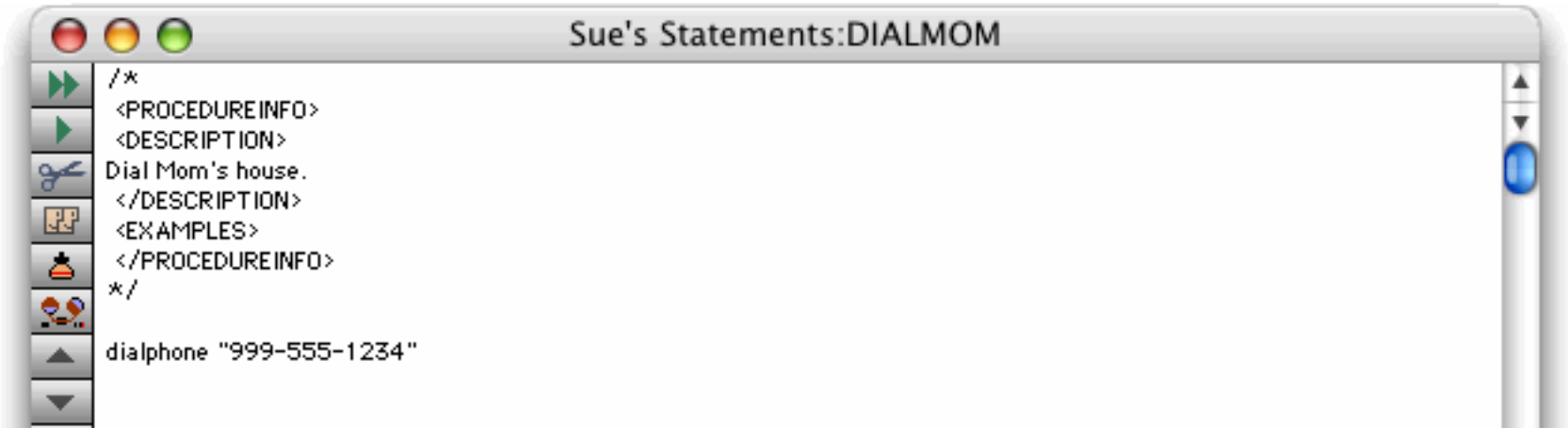
```
<PARAMETER NAME=name TYPE=TEXT>--- put description of parameter here</PARAMETER>
```

The current version of Panorama only supports one type: `TEXT`. The name is used only for documentation purposes.

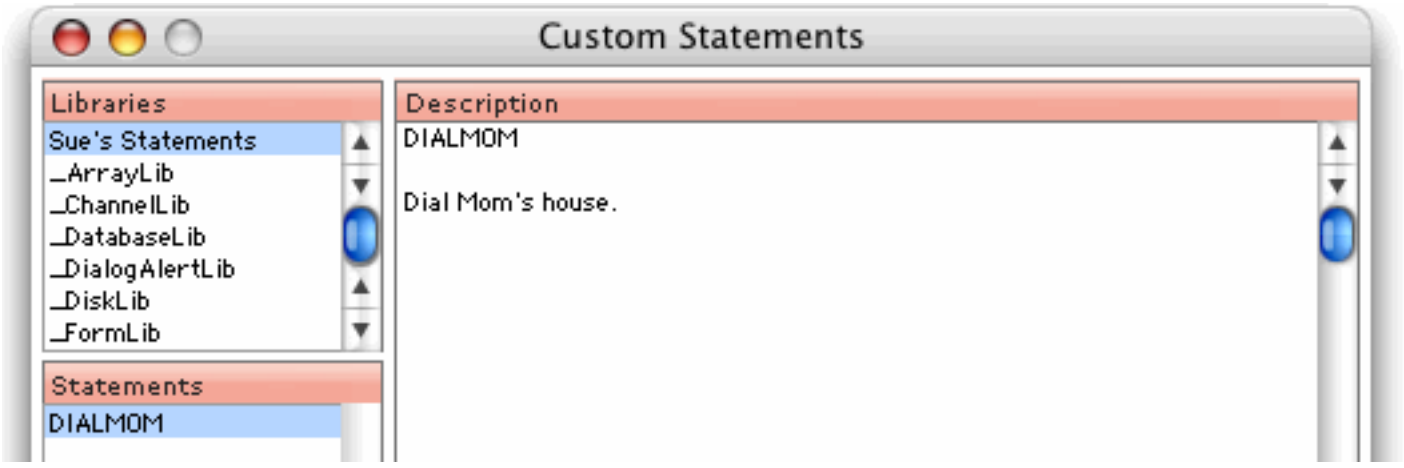
In the description area any extra whitespace will be ignored. In other words, when the description is displayed carriage returns will be turned into spaces and anywhere there are two or more spaces in a row they will be replaced by a single space (just like HTML text).

The examples section is optional. If included, whitespace is handled the same as the description area. However, you can also include `<p>` and `<br>` tags which work just like HTML. (However, `<P>` and `<BR>` are not supported, the tags must be lower case.)

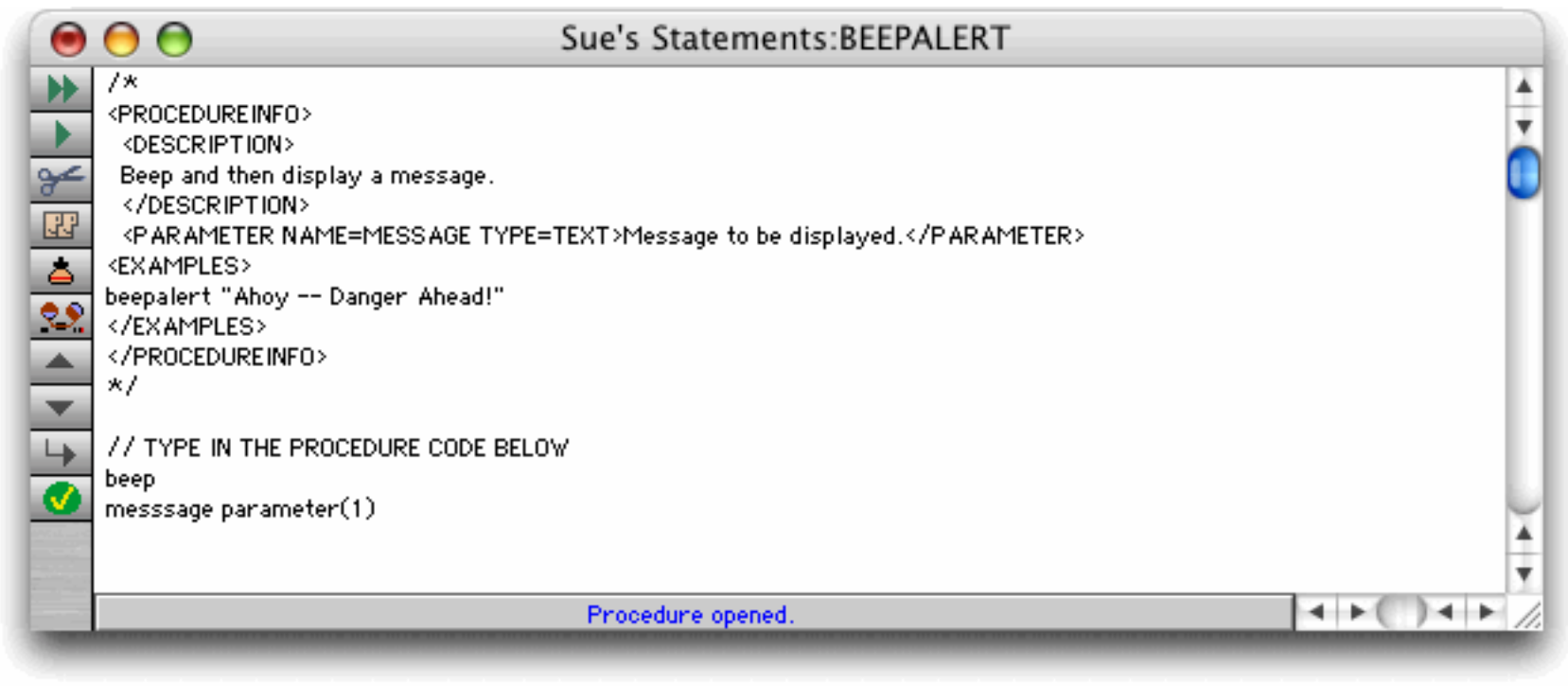
In the previous section a simple `DIALMOM` statement was created. This statement has no parameters at all, so the procedure information block would look like this:



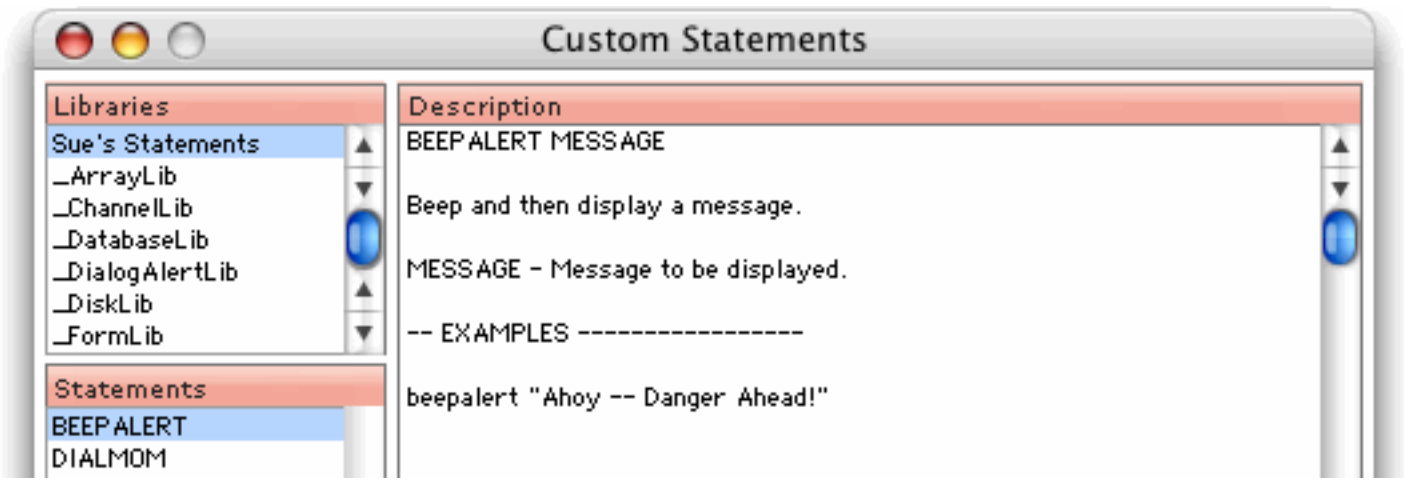
Whenever you change the procedure information block you must go back to the Custom Statements wizard and use the **Register New/Modified Statements** command from the Statements menu. This is especially true if you change the number of parameters. This command tells Panorama to update it's internal description of this statement. After using this command the wizard now shows the description from the information block.



Here is a slightly more advanced custom statement. This statement has one parameter, a message to be displayed.



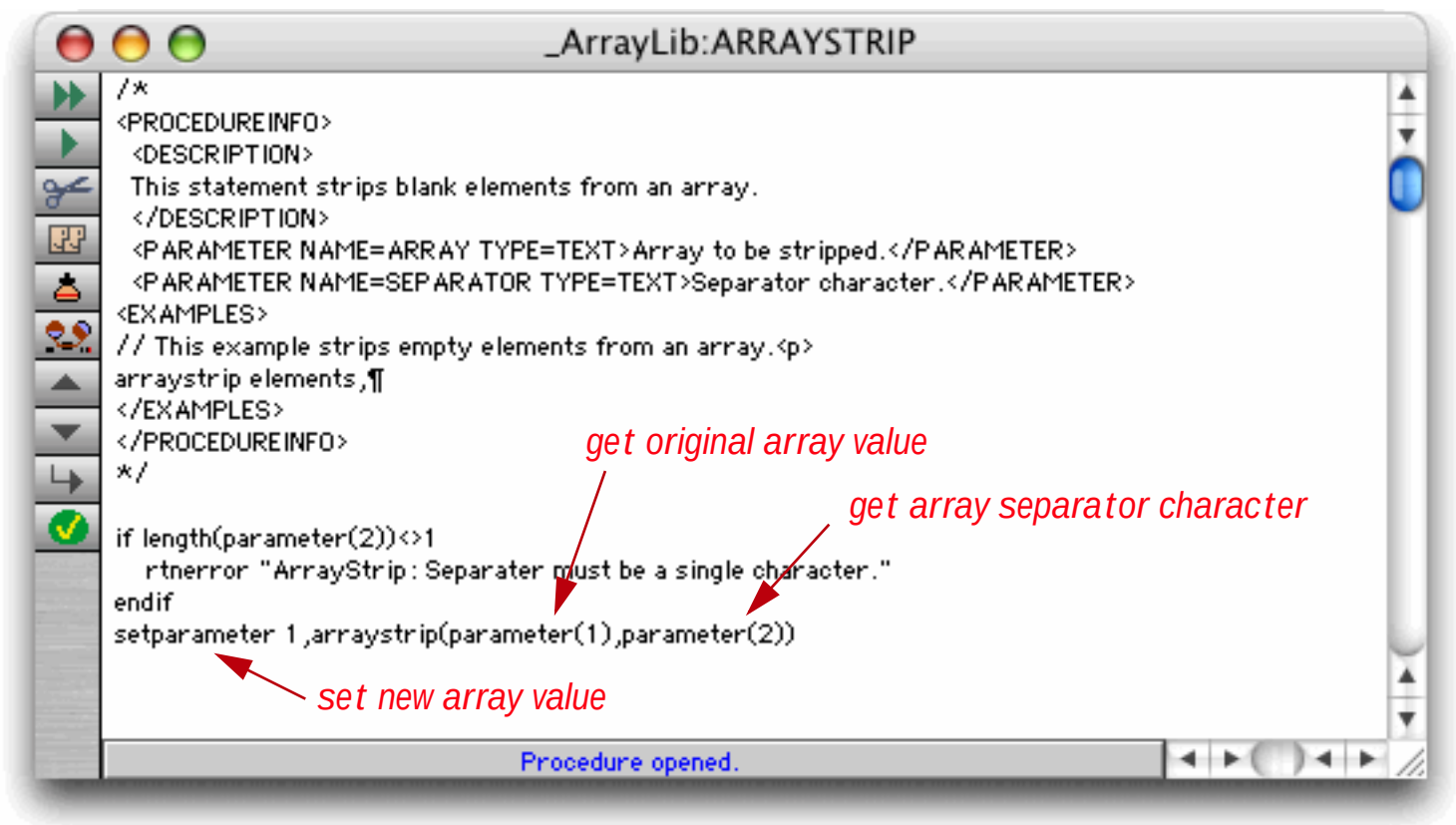
After the **Register New/Modified Statements** command is used, this statement will appear in the Custom Statements wizard like this.



If your procedure has additional parameters you need to document each one in the procedure information block.

Processing Parameters

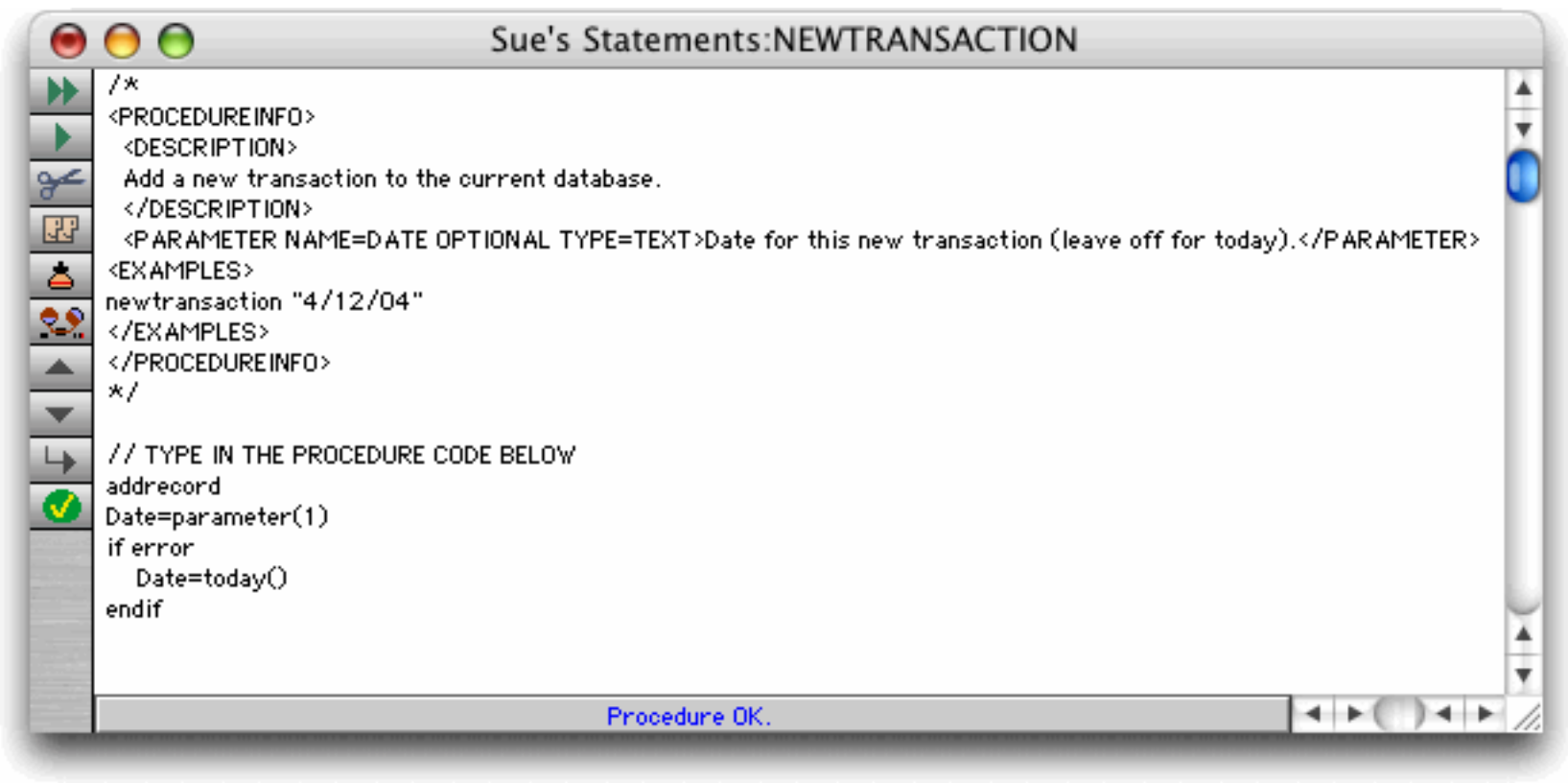
Within a custom statement parameters are usually handled with the parameter(function and setparameter statement, just like a normal subroutine (see "Passing Values to a Subroutine (Parameters)" on page 1386 and "Passing Values Back From a Procedure" on page 1387). For example, here is a custom statement that uses two parameters.



The custom statement libraries that come with Panorama are full of ideas and techniques that you can use in your own custom statements.

Optional Parameters

In some cases a parameter may be optional. If the parameter is not supplied then the statement will assume a default value. To make a parameter optional you need to adjust the procedure information block and the procedure itself. For example, the custom statement below has one parameter (the date), but it is optional.



To make this parameter optional, the word `OPTIONAL` may be placed anywhere within the `<parameter...>` tag, as shown above. Since the parameter is now optional, the `parameter(` function might not work. The procedure checks for this with the `if error` statement (see "Error Handling with if error" on page 1381). If the parameter is missing, this particular statement substitutes today's date.

Because the parameter is optional, the newtransaction statement can be used with or without a date parameter. If there is no parameter, the statement assumes today's date.

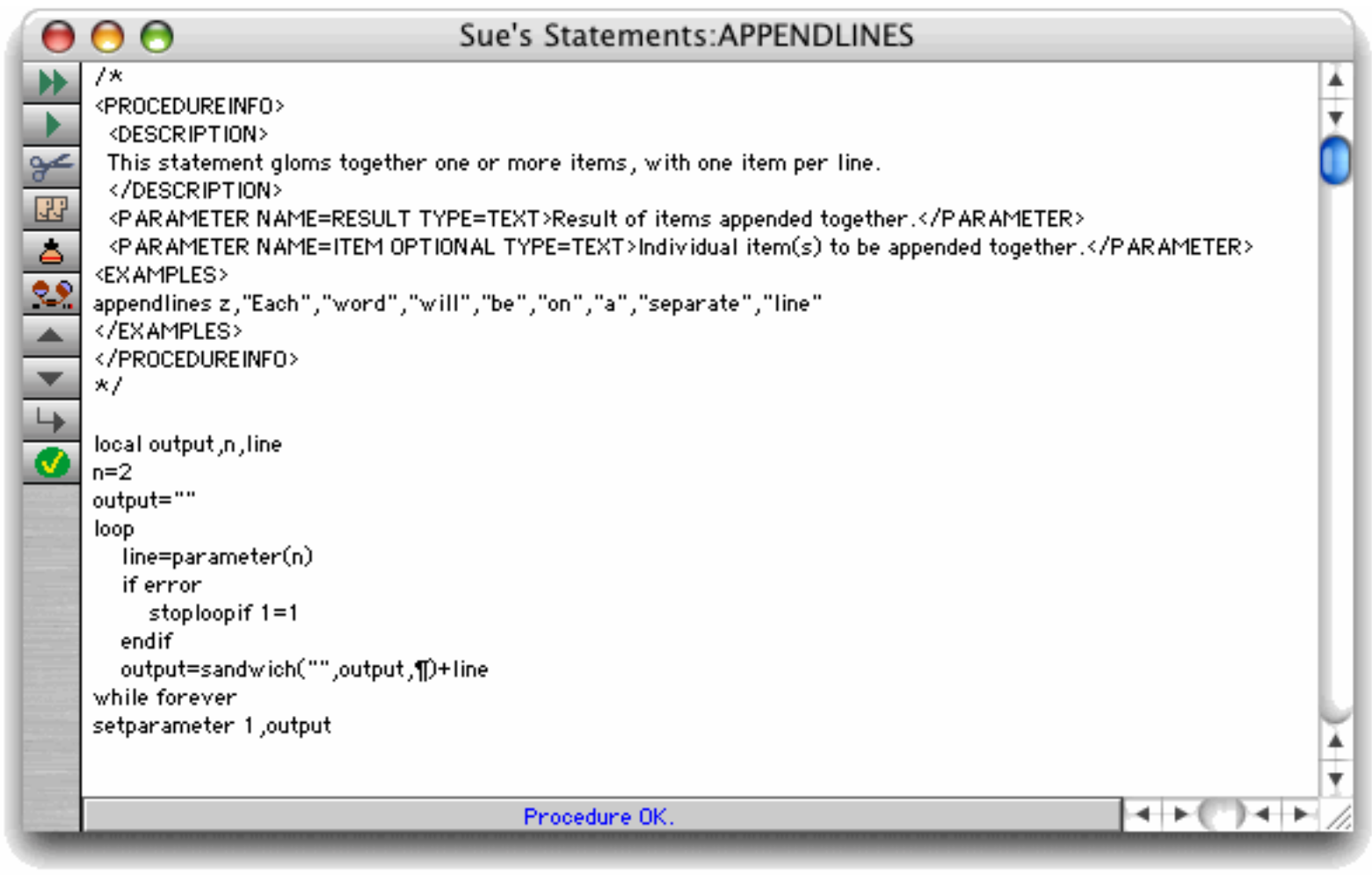
```
newtransaction
```

If there is a parameter, the statement will use that date instead.

```
newtransaction "8/27/03"
```

Repeating Parameters

Sometimes you may not know in advance how many parameters are needed. If placed at the end of the parameter list, an optional parameter can actually repeat over and over again. The second parameter of this custom statement is optional, so it can repeat over and over again.



Notice that the loop loads increasing parameters (2, 3, 4) until it finds a missing parameter, at which point it stops.

Raw Parameters

The `parameter(` function allows a procedure to determine the value of a parameter. Sometimes, however, a procedure needs to access the text of the parameter itself, rather than the value. This information is available in the `¨_RawParameters¨` local variable. This variable is a tab separated array with the raw text of all of the parameters passed to the subroutine.

If a statement needs to access the raw text of a parameter you'll usually need to add the keyword `RAW` to the procedure information block for that parameter, like this:

```
<PARAMETER NAME=ASSIGNMENT RAW TYPE=TEXT>Assignment statement.</PARAMETER>
```

The RAW option tells Panorama not to display an error message if it can't compute a value for the parameter (for example if a variable hasn't been defined yet). However, each parameter must still be a syntactically correct formula. In other words, it's ok to have a parameter of X even if X hasn't been created yet, but X+ can never be a legal parameter because that is a syntax error.

To see a some good examples of how raw parameters can be used check out the MAKEGLOBAL and ARRAYSUBSET custom statements in the _UtilityLib library.

Debugging a Custom Statement

A custom statement can be debugged like any other procedure. You can put a debug statement in the code and then single step through it (see “Debugging a Procedure” on page 1431). As long as the procedure editor window is open you'll be able to step through the code. You can even step through the custom statements that are supplied with Panorama.

Accessing Forms & Procedures in the Library Database

When you write a custom statement keep in mind that as the procedure runs the current database is normally the database that has the topmost window... not the library database. If you need to refer to an item in the current database you can use the info("proceduredatabase") function. For example, suppose you want to open a form called "my form" in the library database. Here's an example of how this could be done:

```
window info("proceduredatabase")+":SECRET"
openform "my form"
```

Here's how to access a fileglobal or permanent variable in the library database:

```
grabfilevariable(info("proceduredatabase"),"variableName")
```

Here's how to call another procedure in the library database:

```
farcall info("proceduredatabase"),theprocedurename,parameters,if,any
```

Of course if the other procedure has a name that is all capital letters it will be a statement and can be called that way as well.

Advanced Topic: Using Libraries In Other Folders

Panorama automatically loads the libraries in the Extensions:Libraries folder when Panorama opens. However, it is possible to have libraries in other locations. For example, you might want to do this for a commercial product so that you can locate the library in your own folder instead of having to install it into the Panorama folder. In this case, however, you must manually load the library. Naturally you do this with a custom statement - the LoadLibrary statement. This statement has two parameters, the path to the folder containing the library and the name of the library itself. Here's how you could load the library database My Library which is in the same folder as the current database.

```
loadlibrary folderpath(dbinfo("folder","")),"My Library"
```

(By the way, there's no harm in loading a library more than once, except for the small delay. If you want, you can check to see if the library is already loaded with the info("procedurelibraries") function.) This function returns a carriage return separated list of all of the libraries that are currently active.

Program Formatting

The way a program is formatted can make a big difference in how understandable it is. Panorama is very flexible in letting you format a program; you can have multiple statements on a single line, or split a single statement over multiple lines. Statements can be flush on the left or indented; it's all up to you.

For example, here's a sample procedure from earlier in this chapter with one line per statement.

```
select Debit>0
field Category
groupup
field Debit
total
outlinelevel 1
```

Here's the same procedure squished onto a single line. Panorama will understand this just fine, although you and I might have a more difficult time.

```
select Debit>0 field Category groupup field Debit total outlinelevel 1
```

You can even split individual statements across multiple lines, as long as you don't split a single word or constant in the middle. Here's another version of this same program.

```
select
Debit>0
field
Category
groupup
field
Debit
total
outlinelevel
1
```

Anyplace you can have a single blank or carriage return you can have more than one. Here's one final example.

```
select       Debit>0
field        Category
groupup
field        Debit
total
outlinelevel 1
```

In general, we recommend using one statement per line for readability. We also recommend indenting the statements between `if` and `endif`, `case` and `endcase`, and between `loop` and `until` or `while` (as seen in most of the examples throughout this manual). If you have multiple levels of nested `if` statements, each level should be indented further. This makes it easy to see which statements are associated with each `if`/`endif` pair.

Here's a program example with no indenting:

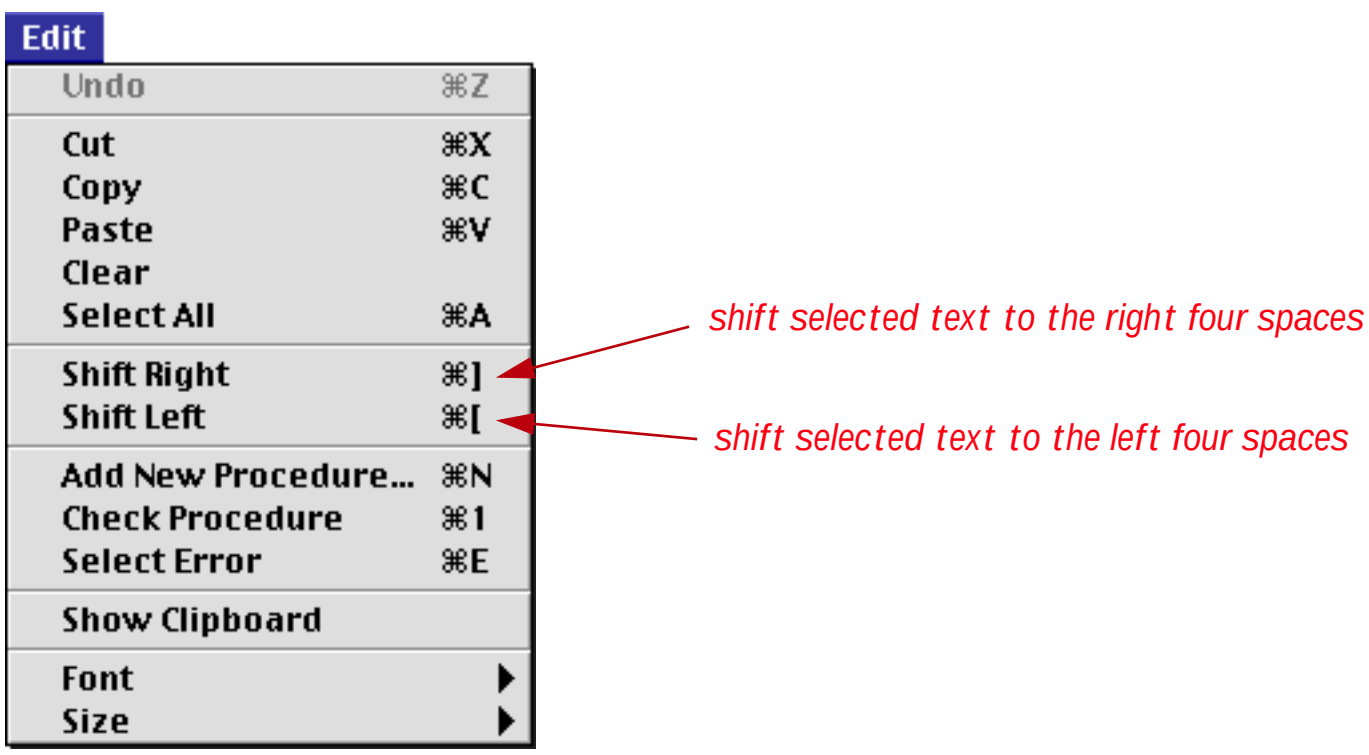
```
local CardLength
if PaymentMethod="Credit Card"
CardLength=length(CardNumber)
if CardLength<13 or CardLength>16
message "Sorry, invalid credit card number."
endif endif
```

Now the same procedure with the recommended indenting:

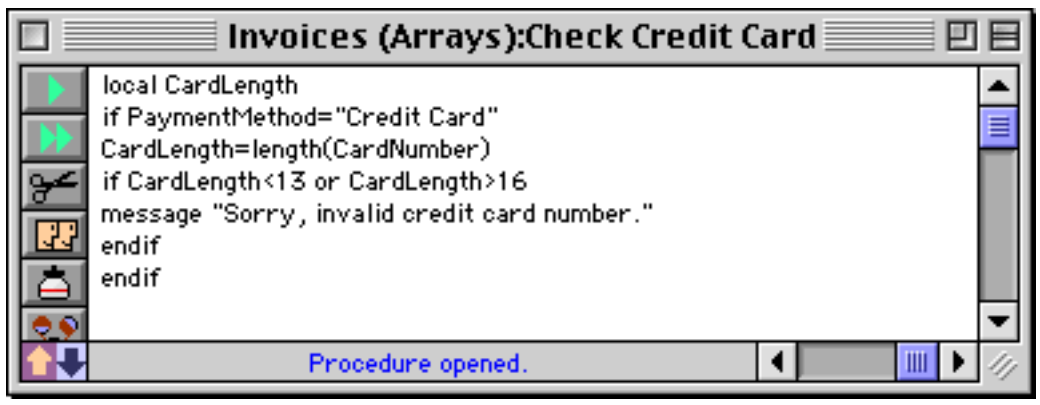
```
local CardLength
if PaymentMethod="Credit Card"
   CardLength=length(CardNumber)
   if CardLength<13 or CardLength>16
      message "Sorry, invalid credit card number."
   endif
endif
```

A lot easier to understand, isn't it?

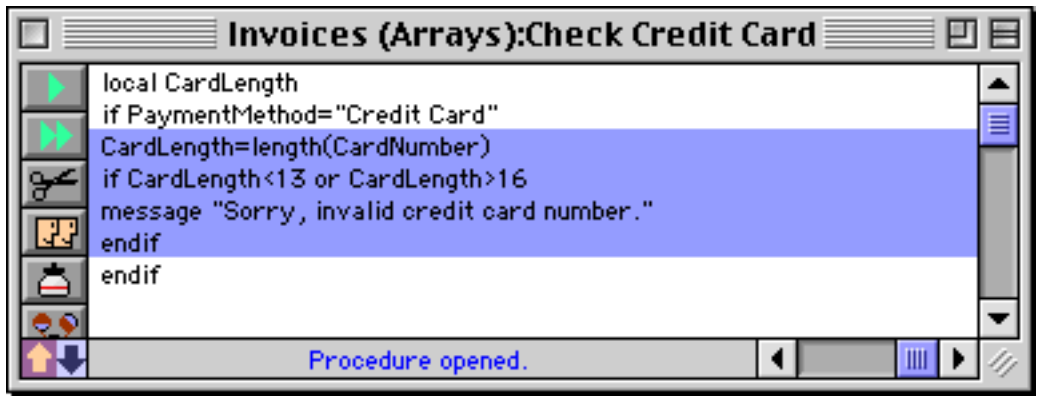
The **Edit** menu has two commands that can help you shift a section of text to the left or right.



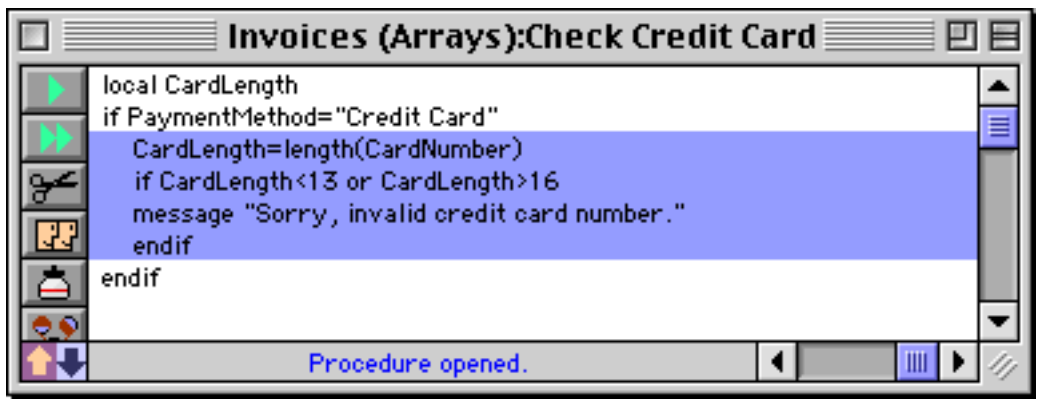
For example, suppose you start with this procedure, which is not indented at all (it's the same procedure listed earlier in this section).



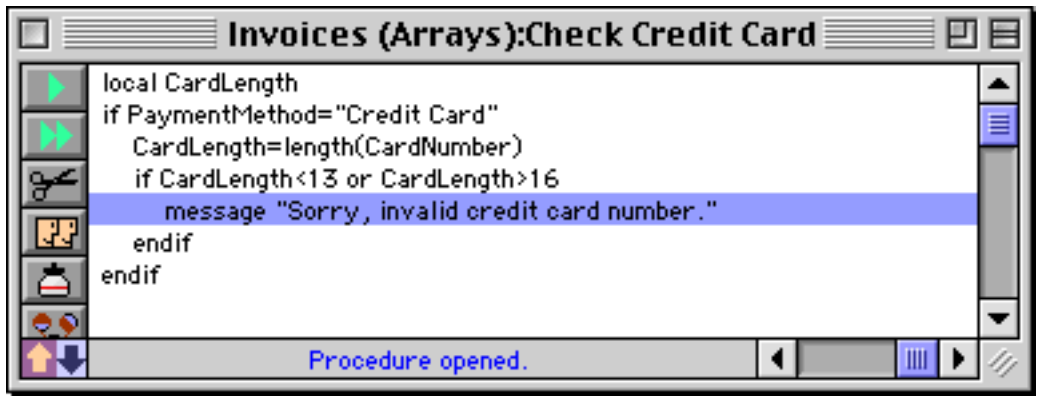
To indent the text, start by selecting the text between the `if` and `endif` statements.



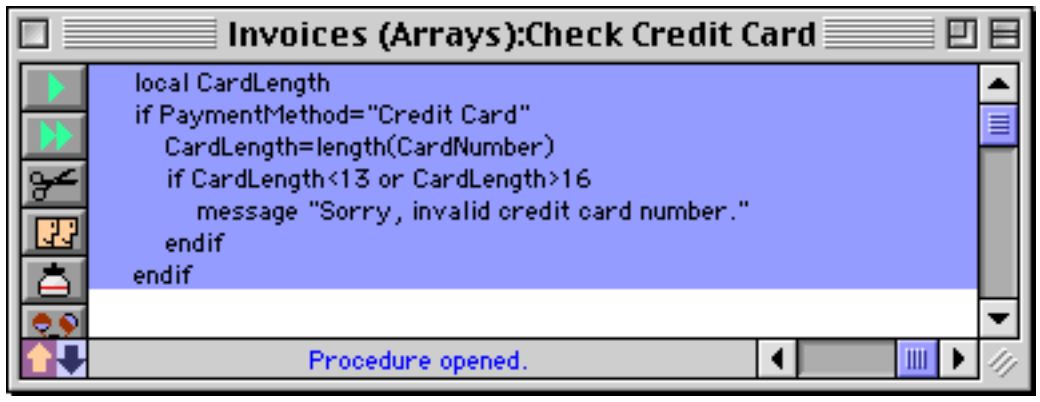
Now use the **Shift Right** command to indent the selected text.



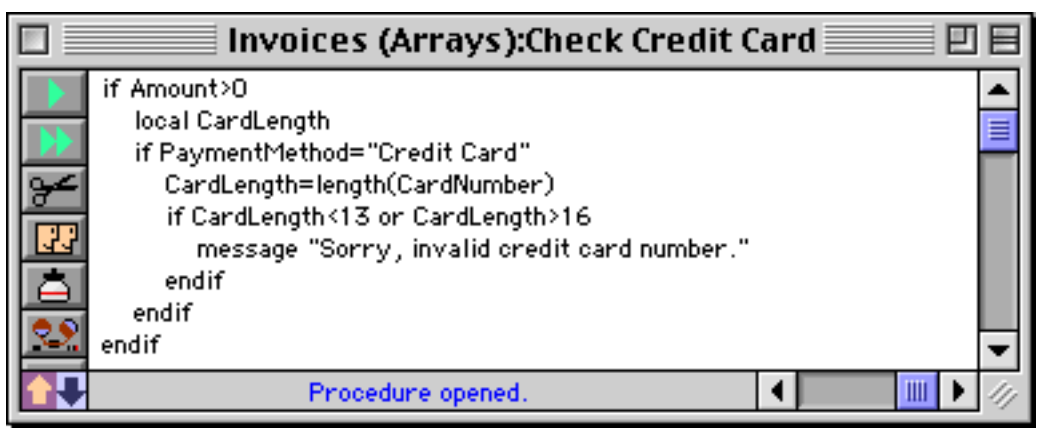
Repeat as necessary to indent any other text.



You can even use these commands to change the indentation of the entire procedure.



Now we can add another if statement around the entire procedure.



Consistent indentation can go a long way towards making your programs more readable and bug free.

Notes To Yourself

A **comment** is a note inside the program. Comments are very useful for documenting how a procedure works, what the variables are for, what the procedure parameters are, etc. Comments are totally optional, but you should use them to record anything you think you might forget about the operation of a procedure.

Panorama has three different comment styles: /* … */, / /, and ;.

/* … ***/** comments begin with /* and end with */ The advantage of this type of comment is that a single comment may be many lines long. You can also use this type of comment within a formula.

// comments begin with // and continue to the end of the line.

; comments begin with a semicolon and continue to the end of the line.

A comment can appear almost anywhere in a procedure. The only restriction on comments is that they cannot be inside the middle of a statement or formula; they must be between statements.

This example shows a procedure with lots of comments. If anyone comes back and takes a look at this procedure next year, they will have no problem telling what the procedure does and how it does it. To emphasize the comments they are shown in purple below. However, when actually editing a procedure the comments are black just like everything else.

```
/* Procedure: .Delay

 This procedure delays for a fairly precise time.
 The procedure has one parameter, the number of seconds to delay.

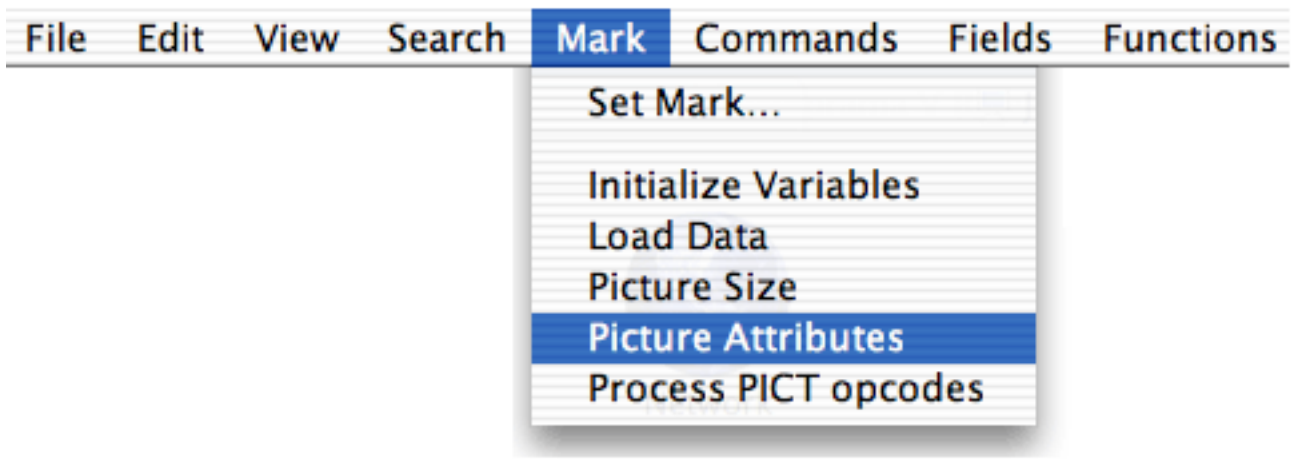
 Example:
   call .Delay,12; delay for 12 seconds
*/
local startTime
startTime=now()// record the time we started
loop
   nop   ; short delay
until now()>startTime+parameter(1)
```

“Commenting Out” Statements

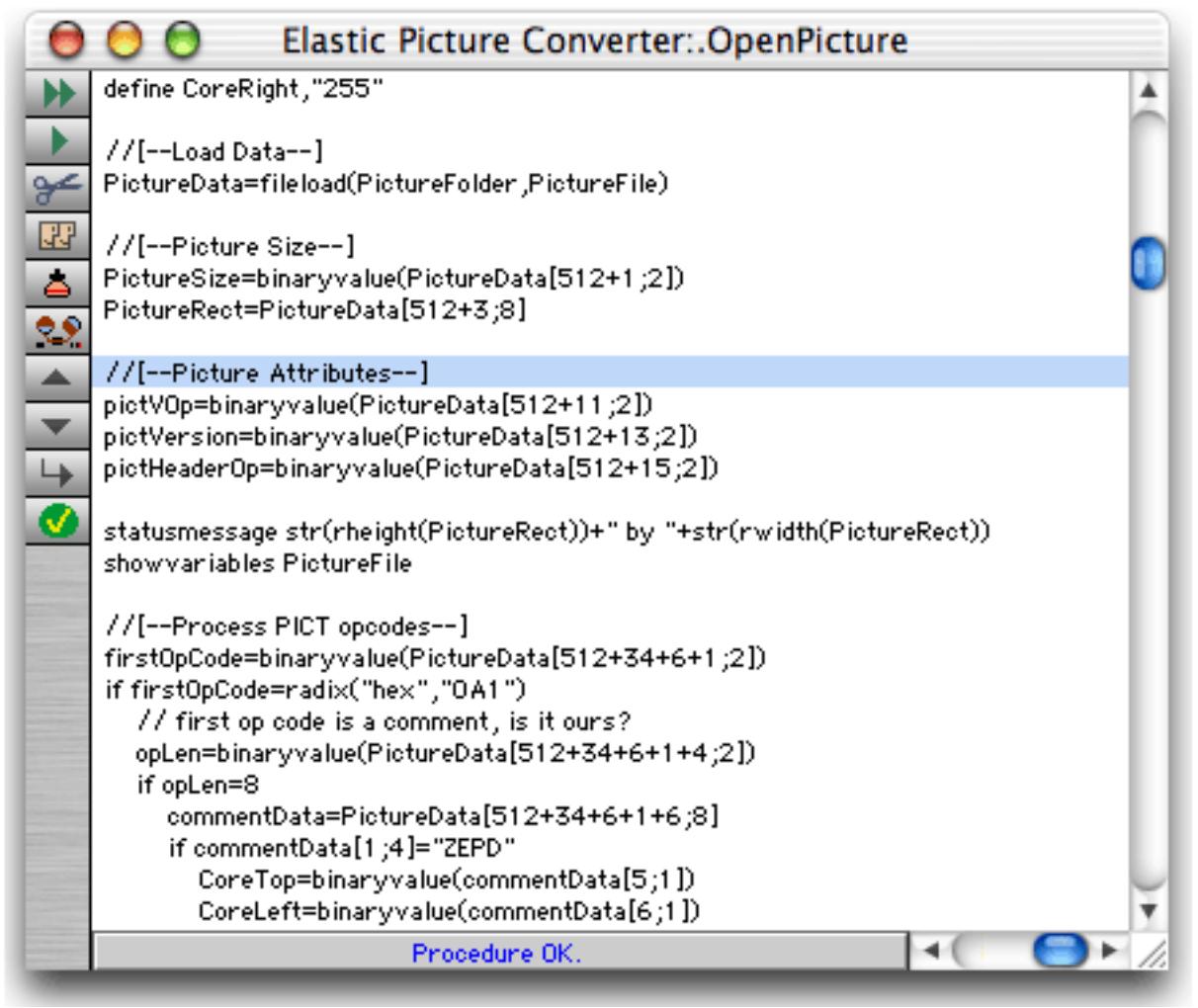
One handy use for /* … */ comments is to temporarily remove one or more statements from your program (usually for testing purposes). Simply put /* and */ around the statement or statements you want to remove, and those statements are effectively removed from your program without actually erasing them. (Programmers call this “commenting out” the statements, because they are temporarily “out” of the program.) To re-enable the statements simply remove the /* and */.

Organizing Large Procedures (The Mark Menu)

A single procedure may include up to 32,000 characters of text. A procedure that long would be more than 20 pages long if printed. As a procedure grows it can be difficult to navigate within the procedure itself. The **Mark** menu allows you to create "bookmarks" within the procedure. These marks are listed in the Mark menu.



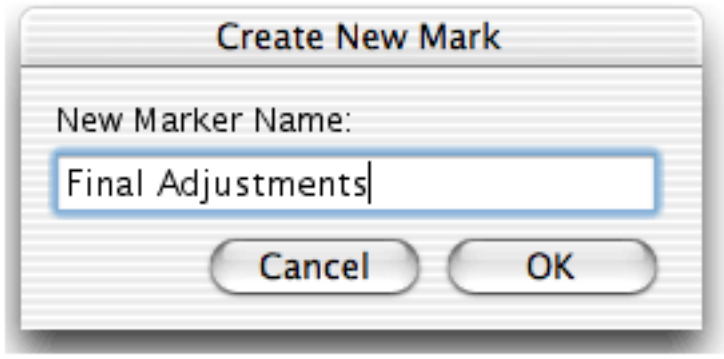
Choosing an item from this menu causes the editor to jump to the location in the procedure text corresponding to the mark.



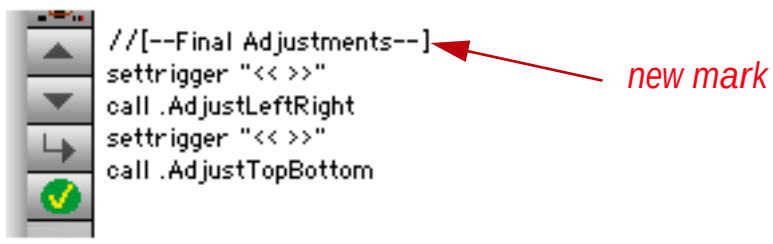
As you can see in the illustration above, a mark is simply a special comment (“Notes To Yourself” on page 1424) that looks like this:

```
//[--Mark Name--]
```

You can simply type in a mark yourself, or you can use the **Set Mark...** command in the Mark menu. This command prompts you for the mark name, then inserts it into the procedure at the current location.



Here is the inserted mark.



Each procedure may contain up to 100 marks. If there are more than 100 marks then any past the first 100 will not appear in the **Mark** menu.

Suppressing Display of Text and Graphics

As a program executes the windows belonging to the database will often flicker or even redisplay over and over again as the statements are performed. Often this redisplay serves no purpose except to slow the program down and annoy you. To disable display while a sequence of steps is performed you must bracket the steps with the noshow and endnoshow statements, like this.

```
noshow
   statement
   statement
   statement
   ...
endnoshow
```

The noshow statement tells Panorama to suppress all display of text and graphics by the following statements. For example the sort and formulafill statements normally cause some or all of the window to be redisplayed. But if these statements follow a noshow statement the window will not redisplay. The endnoshow statement cancels the effect of the noshow statement and resumes normal display operation.

Note: The noshow statement suppresses all display that results from changes to the database. It does not, however, suppress display that is causes by changes in the configuration of database windows. For example if a procedure moves a window to the front with the window statement the newly visible section of the window will always be displayed, with or without a noshow statement. This is true for any statement that changes the window configuration: opening or closing a window, changing the size of a window, or changing the stacking order of the windows.

Updating the Display After (or Within) a NoShow Block

The noshow statement is great for making procedures run faster without unnecessary window updating. There's just one problem though - since the window is not updated, it winds up being wrong! Consider the the procedure below.

```
noshow
   field Date
   groupup by month
   field Category
   groupup
   field Amount
   total
   outlinelevel 2
endnoshow
```

Without the noshow statement this procedure will cause the window to update four times. But with the noshow statement the final result is not displayed! To fix this you must add one of the seven statements in the table below.

Statement	Parameters	Description
showpage	none	Displays the entire database.
showline	none	Displays the current record.
showfields	list of fields	Displays the specified fields in the current record
showvariables	list of variables	Displays the specified variables
showcolumns	list of fields	Displays the specified fields in all visible records
showrecordcounter	none	Displays the number of records
showother	field,option	Depends on option, see documentation below

Using the showpage statement we can fix the program listed earlier so that it displays the final result at the end of the procedure.

```
noshow
   field Date
   groupup by month
   field Category
   groupup
   field Amount
   total
   outlinelevel 2
   showpage
endnoshow
```

The seven display statements are described in the following sections.

ShowPage

The showpage statement forces Panorama to redisplay all windows in the current database. Here is an example that performs several operations on the current database, but only updates the display once:

```
noshow
   field Date
   groupup by month
   field Category
   groupup
   field Amount
   total
   outlinelevel 2
   showpage
endnoshow
```

ShowLine

The showline statement forces Panorama to redisplay the current record in all windows in the current database. The example below clears the current record in the database, but doesn't display anything until it is completely finished.

```
noshow
   field array(dbinfo("fields",""),1,¶) /* go to first field */
   loop
      clearcell
      right
   until stopped
   showline
endnoshow
```

Without the noshow statement you would be able to watch as Panorama cleared each cell in the line. With the noshow statement all of the cells will appear to disappear simultaneously.

ShowFields field,field,…,field

The showfields statement forces Panorama to redisplay the specified fields in all windows in the current database. You may list as many fields as you want to display, with each field separated by a comma. (If you want to display all the fields it is easier to use the showline statement.) The example below modifies three fields but only displays the change made to the Balance field.

```
noshow
   Date=today()
   Time=now()
   Balance=Credit-Debit
   showfields Balance
endnoshow
```

ShowColumns field,field,…,field

The showcolumns statement forces Panorama to redisplay the specified fields in all windows in the current database. In a data sheet or view-as-list window the entire column is re-displayed, not just the current record. You may list as many fields as you want to display, with each field separated by a comma. (If you want to display all the fields it is easier to use the showpage statement.) The example below performs two calculations on the Balance field, but only redisplays the column a single time.

```
noshow
   field Balance
   Balance=Credit-Debit
   RunningTotal
   showcolumns Balance
endnoshow
```

ShowVariables var,var,…,var

The showvariables statement forces Panorama to redisplay the specified variables in all windows in the current database. You may list as many variables as you want to display, with each variables separated by a comma. The example below adds a new record to the database without changing the display, but does show the new record count.

```
noshow
   global myCount
   addrecord
   myCount=info("total")
   showvariables myCount
endnoshow
```

Warning: The showvariables statement is **always** required if you want to display changes to one or more variables, even if you are not using the noshow statement.

ShowRecordCounter

The showrecordcounter statement forces Panorama to redisplay the record count in all windows in the current database. The example below adds three new records to the database but only updates the display once.

```
noshow
   addrecord
   addrecord
   addrecord
   showpage
   showrecordcounter
endnoshow
```

ShowOther field,code

The `showother` statement forces Panorama to redisplay all windows in the current database. You specify a code that tells Panorama what portion of the window to update. This is the code that Panorama uses internally, so if Panorama becomes capable of a new mode of redisplay it will automatically become available. Some codes allow you to specify a specific field to update, you can use the field name or use All to specify all fields. The available codes are listed in this table.

Code	Action
0	Display current cell (should use `showfields` instead)
1	Display entire page (should use `showpage` instead)
2	Cursor moved, update data sheet
3	Cursor moved, data sheet already updated
4	Update window after insertline (data sheet or view-as-list)
5	Move cursor up / down (for example after a search)
6	New line with cursor move
97	Display record count (use `showrecordcounter` instead)
98	Display data sheet field header (after changing field name)
99	Display after database redesign (insert field, etc.)

We recommend that you avoid this command if one of the other show commands will do the job for you.

Disabling the Watch Cursor

As a procedure runs the cursor often flips from the arrow into a watch, or sometimes a pie chart. This helps let the user know that they may need to wait.

Arrow	Watch	Pie Chart
		

In some cases Panorama flips to the watch or pie chart cursor when it is not really necessary (especially on today's faster machines). If you want the mouse cursor to remain as an arrow while your procedure runs you can use the `nowatchcursor` statement (see "NOWATCHCURSOR" on page 5549). Here is a procedure that opens a database. This would normally cause the watch cursor to be displayed, but in this case the arrow remains active.

```
nowatchcursor
openfile "Reference Data"
watchcursor
```

The final statement re-enables the watch and pie chart mouse cursors. In this example the `watchcursor` statement isn't really necessary because Panorama always automatically re-enables these cursors at the end of any procedure.

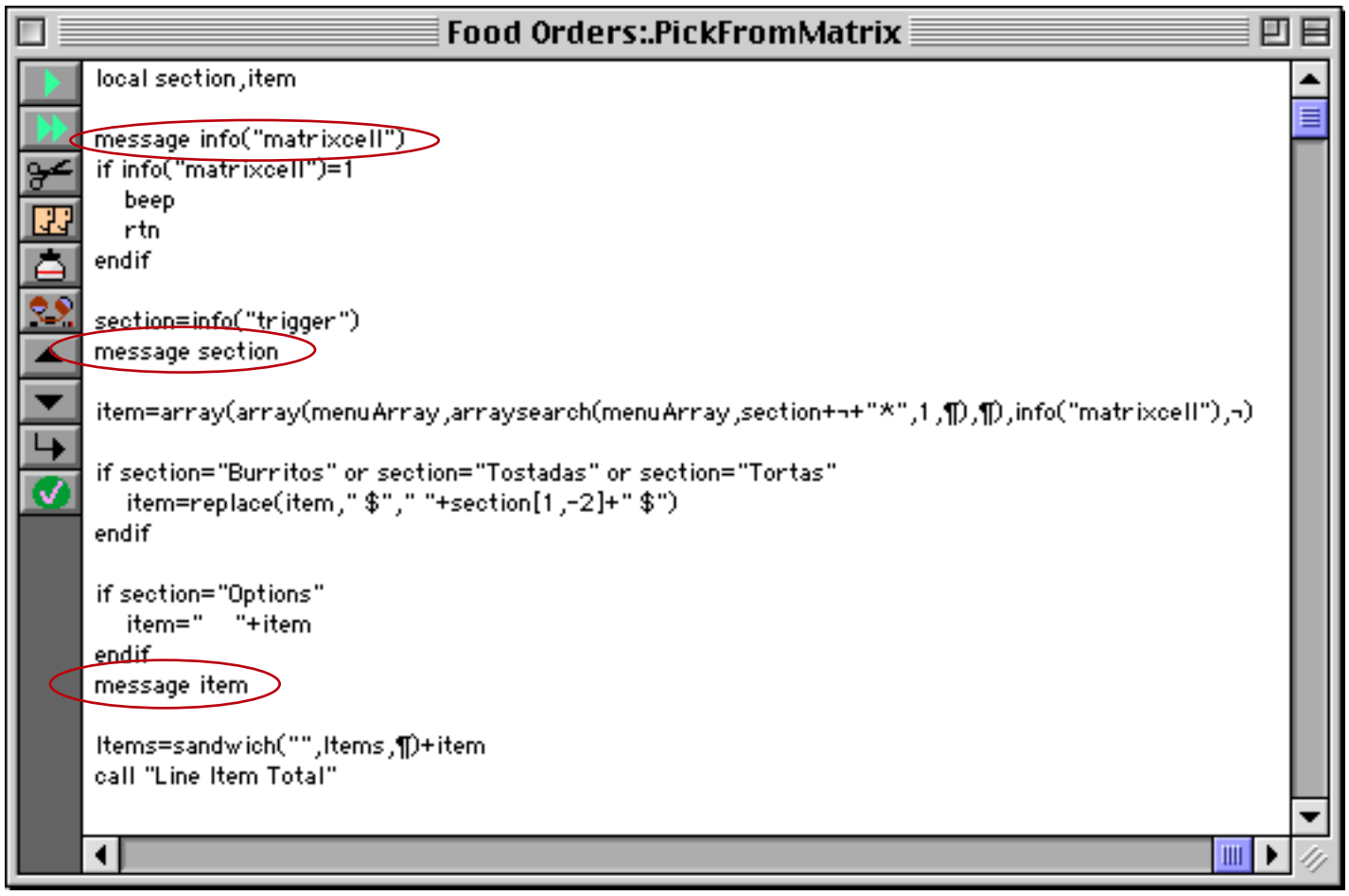
Hide and Show

Previous versions of Panorama (up to 3.0) include `hide` and `show` commands that allowed a programmer to turn off the display of text and graphics while the procedure was running. Unfortunately these commands did not give accurate control over the display, and worse, they could even crash if you attempted to use them across multiple windows. These commands are still available to retain compatibility with old databases, but we recommend that you avoid them for new applications.

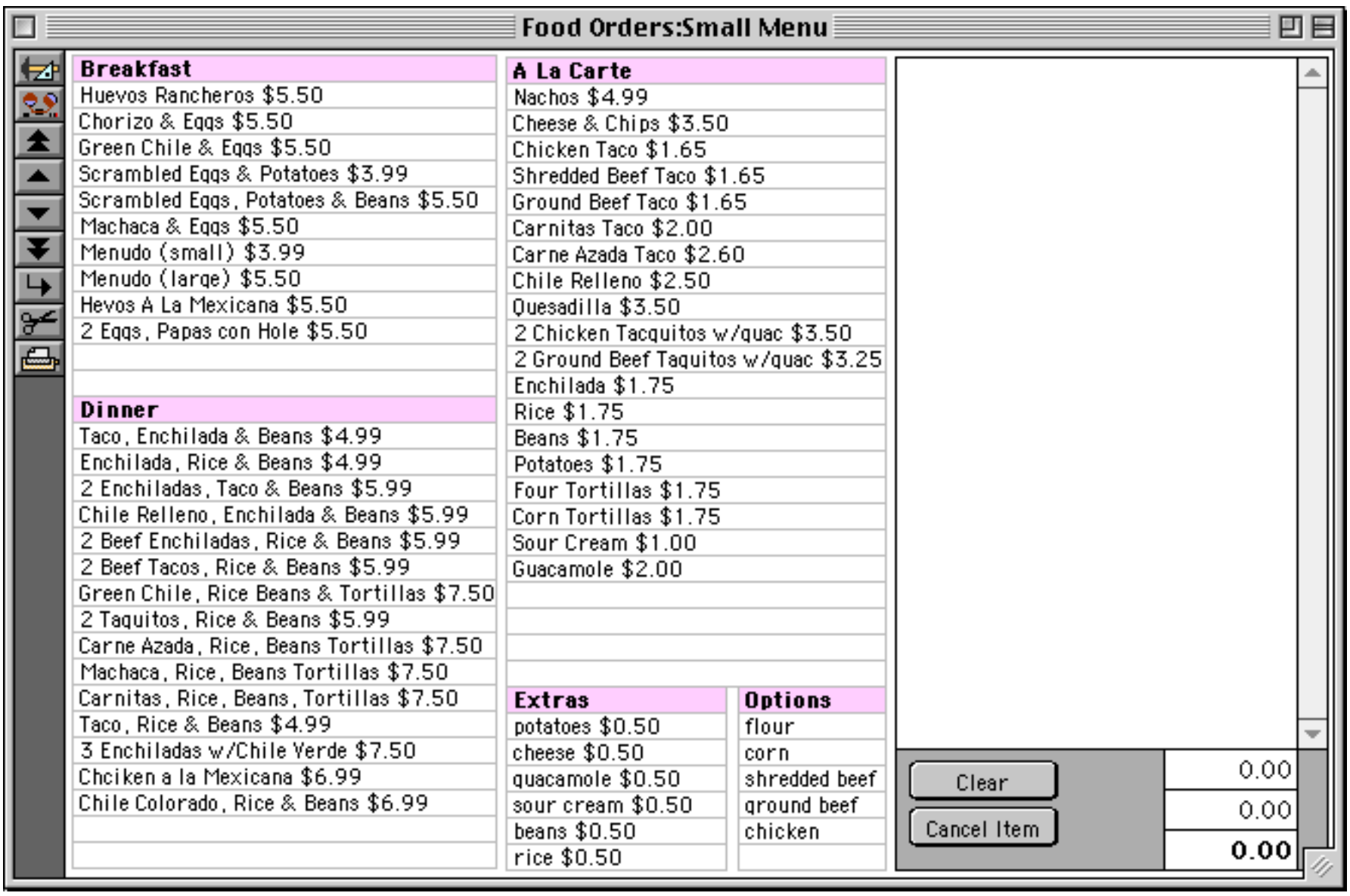
Debugging a Procedure

In the real world, programs often don't work correctly the first time. (Sometimes they don't work the second or the third time, either!) Panorama has a number of tools that you can use to help locate and correct the problem in a procedure.

One of the most basic tools you can use is the `message` statement. By inserting this statement at various points in your program you can display intermediate results and get a feel for what is happening in your program. For example, consider this procedure which has had three `message` statements inserted into it.



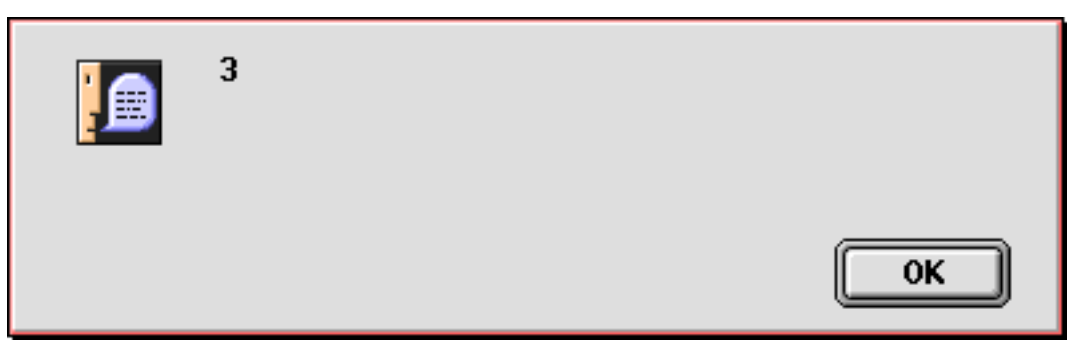
This procedure is designed to be triggered by a Matrix SuperObject (see "Super Matrix Objects" on page 929) which contains menu items.



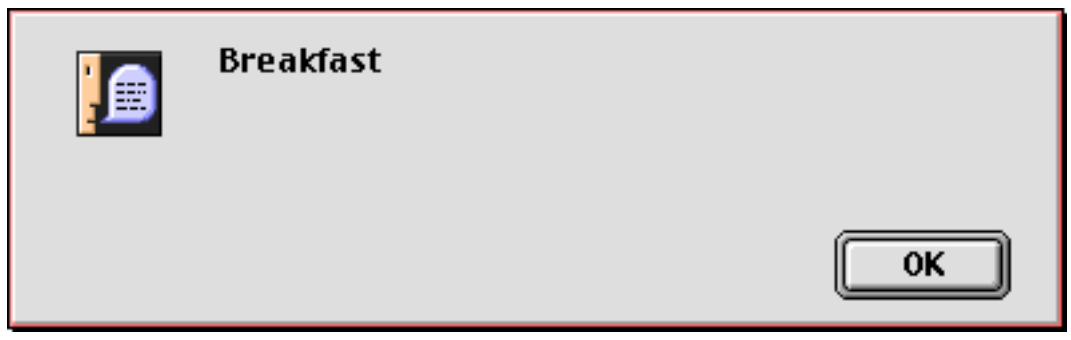
When you click on a particular matrix item the procedure is triggered.



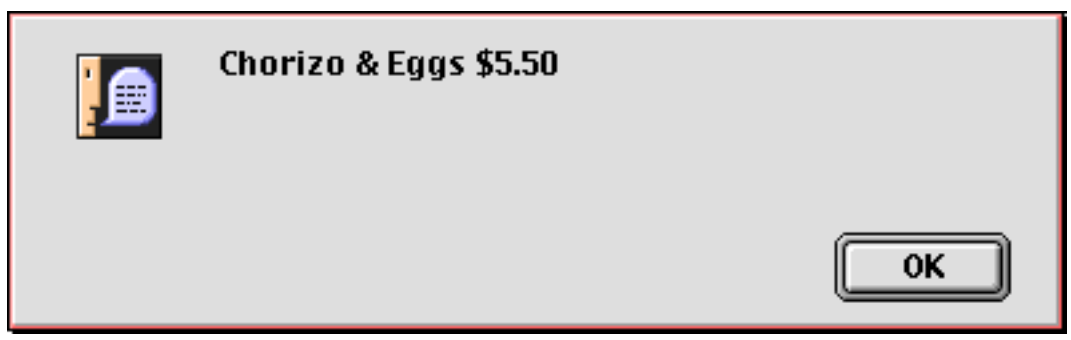
The first message statement, `message info("matrixcell")`, displays the number of the cell that was clicked on. (It also verifies that the procedure is being triggered correctly.)



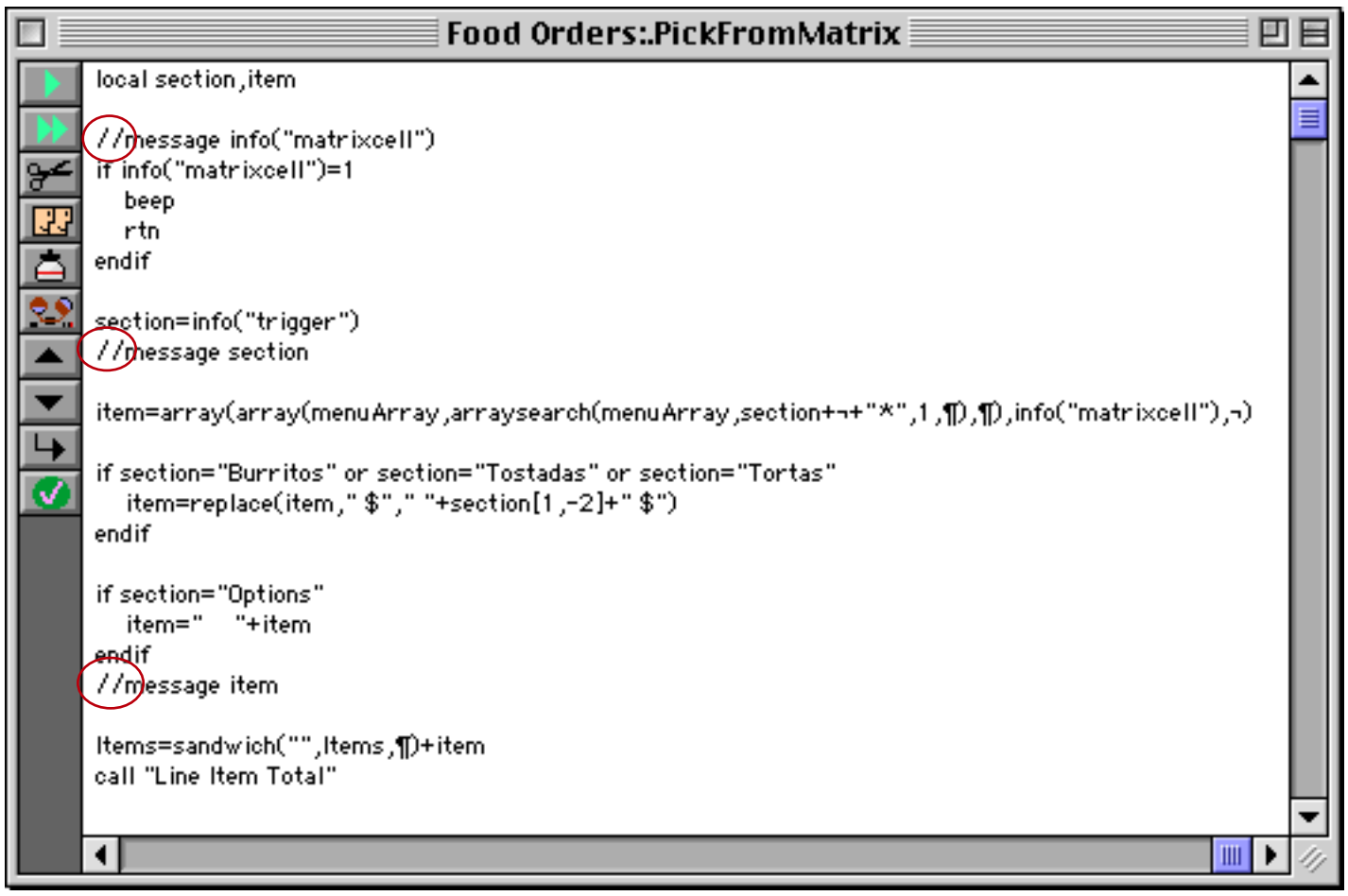
The next message statement, `message section`, displays the result of the `info("trigger")` function.



The final message statement, `message item`, displays the item that has been looked up and will be added to the order. This will allow you to quickly spot any errors in the code that retrieves the item and price.



Once the procedure is working correctly you can remove the `message` statements. If you think there is any chance you might need them again you can temporarily remove them by "commenting them out" (see "“Commenting Out” Statements" on page 1424) like this.



As long as the `//` is in front of the message statement the statement is disabled. Any time you want the statement back in again you simply need to remove the `//`.

Sometimes you may want to run a portion of the procedure, display a message and then stop. To do this simply add a `stop` statement after the `message` statement (see "Stopping the Program" on page 1397). If both statements are on the same line then they can both be disabled with a single `//` comment.

The Panorama Interactive Debugger

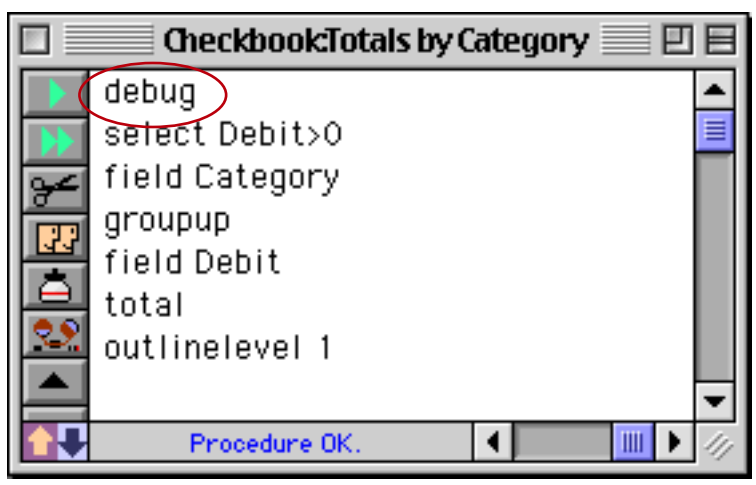
To help solve more stubborn problems Panorama includes a built in debugger. The debugger allows you to stop a procedure in the middle and execute statements one at a time (called **single-stepping**). You can actually watch as your program executes each statement, and you can check the value of fields and variables at any time.

The Debug Statement

The debug statement pauses the procedure so that you can examine fields and variables. You can insert a debug statement anywhere in a procedure, and a procedure may contain more than one debug statement. Usually debug statements are inserted into the procedure temporarily while you are getting the program running, and then removed when the procedure is operating properly.

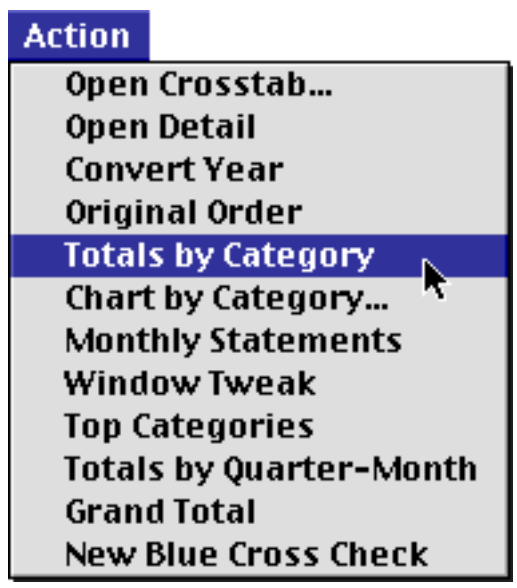
Using the Debugger

The first step in using the debugger is to add one or more debug statements to a procedure. In this example the debug statement has been inserted at the very top of the procedure, but it can be inserted anywhere.

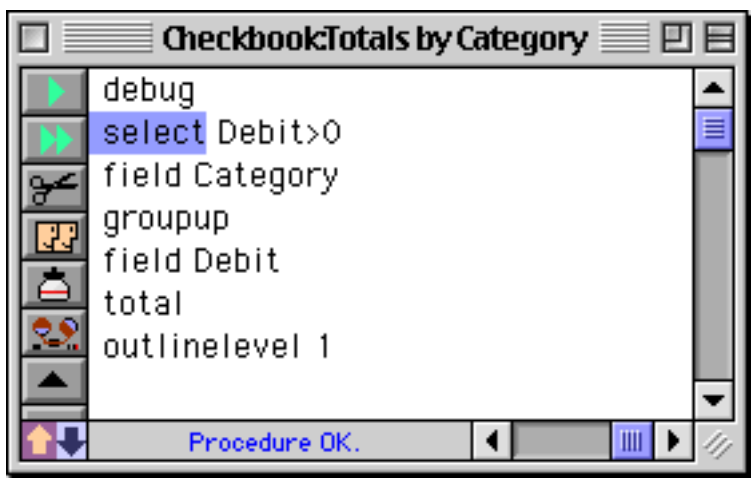


Once this is done, go to a form or data sheet window where you can test the procedure. Be sure to leave the procedure window open! If necessary, you can open an additional window in the same database - see “Opening More Than One Window Per Database” on page 223 and “The View Wizard” on page 227.

Now start the procedure normally. Usually you will click on a button or pull down a menu item. If this is a “hidden trigger” procedure you should perform whatever action triggers the procedure.



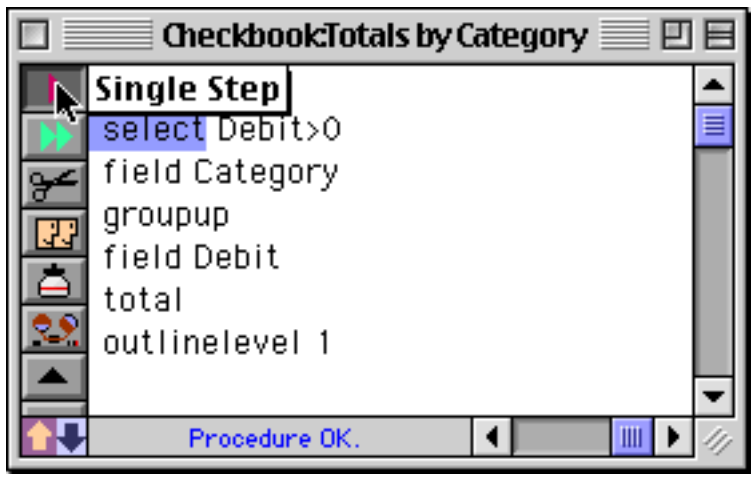
The procedure will run normally until it gets to the debug statement. At that point the procedure will stop and Panorama will bring the procedure window back to the front. The statement following the debug statement will be highlighted, indicating that it is the next statement to be performed when the procedure resumes.



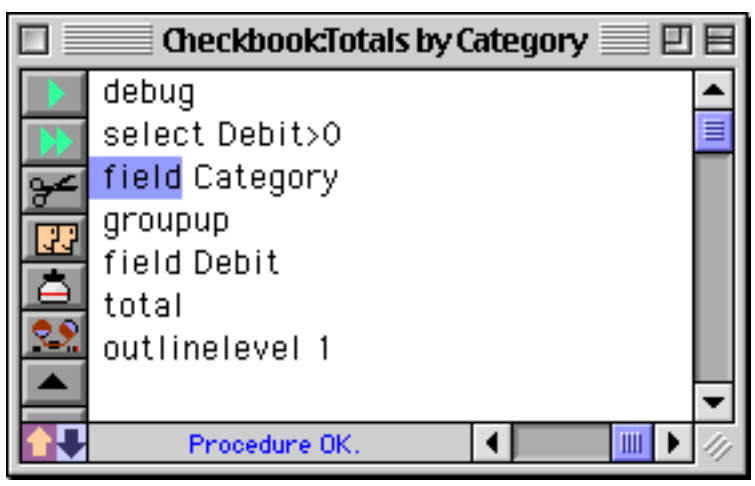
Important: If the procedure window is not open, the procedure will not stop. That's why it was so important to leave the procedure window open in the last paragraph. If you wish, you can leave debug statements permanently in a procedure. They won't affect the procedure unless the procedure window is open.

Single Stepping

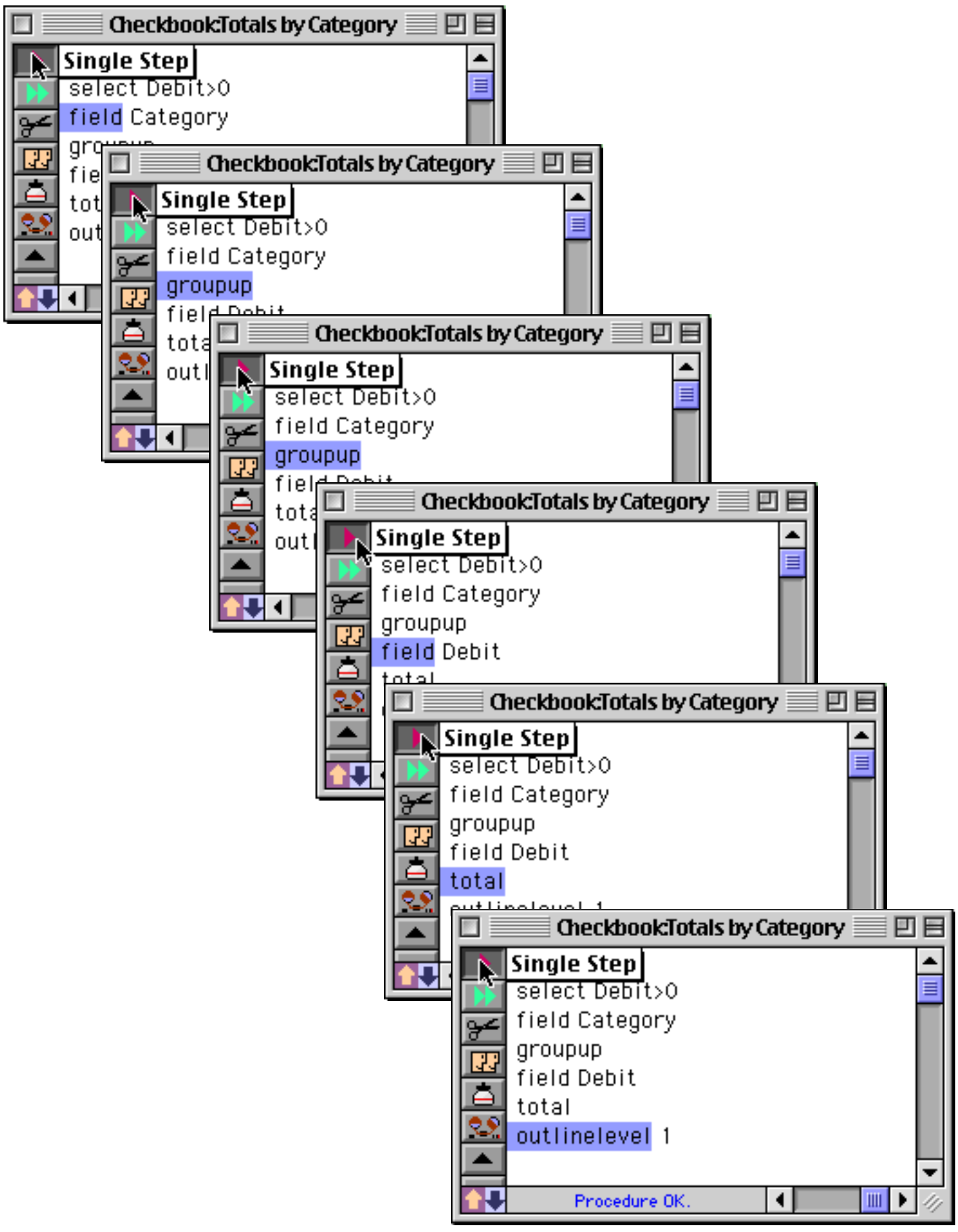
After the procedure has been stopped by a debug statement, you have the option of continuing the procedure one step at a time. You can watch to see what happens as each step is performed. To perform the next step, press the **Single Step** tool, or select **Single Step** from the Debug menu. (Note: you can also use the **Single Step** command without the debug statement simply by clicking on the tool. Panorama will start single stepping from the first line of the procedure.)



Before it performs the next statement, Panorama will move the procedure window to the back again, so that the form or data sheet (or whatever the current window was) is on top again. Panorama performs the statement, then brings the procedure window back on top again. The following statement is highlighted.



By single stepping again and again you can watch the program run. You can see as the procedure makes decisions at `if` statements, watch as a loop runs over and over again—everything your procedure does is instantly visible.



If the procedure uses `call` or `farcall` statement to trigger a subroutine (see “Subroutines” on page 1384), single stepping usually considers the subroutine to be a single step. In other words, in one step Panorama will perform the entire subroutine. However, if the window containing the subroutine procedure is open, Panorama will single step through the subroutine, letting you see each step in it.

Resuming Full Speed Execution

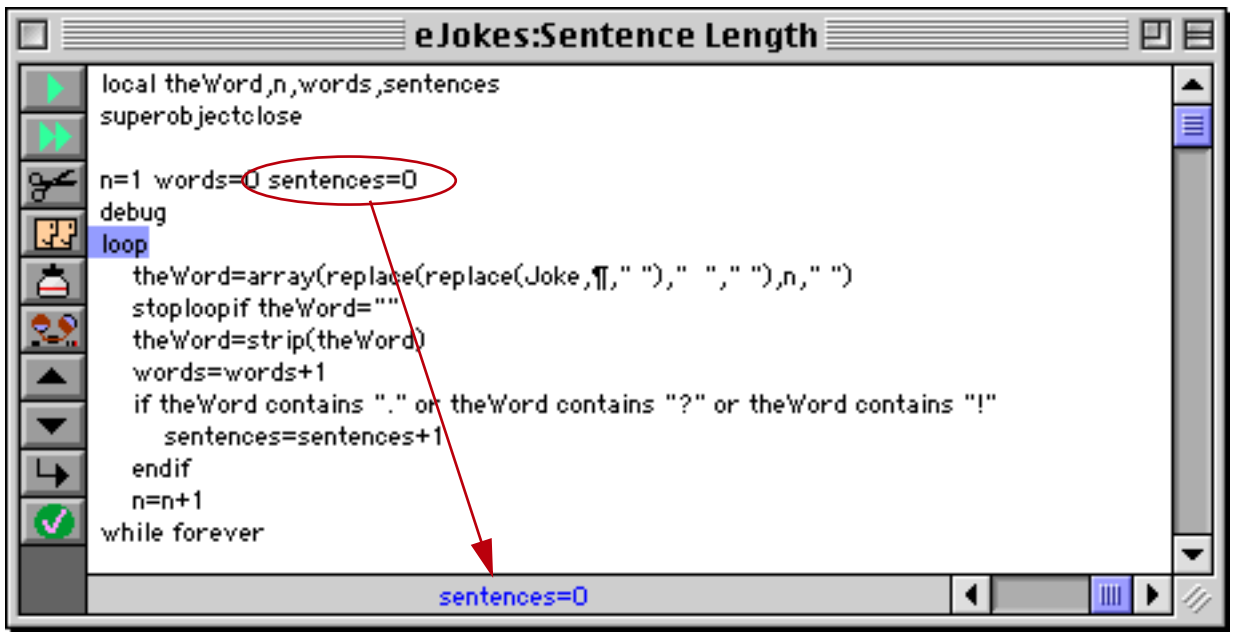
If you want the procedure to start up again at full speed, press the **Run** tool or select **Run** from the Debug menu. The procedure will start up again at full speed from the current spot. It will continue at full speed until it either reaches the end of the procedure, or it comes to another `debug` statement in an open procedure window.

Making Corrections to a Procedure

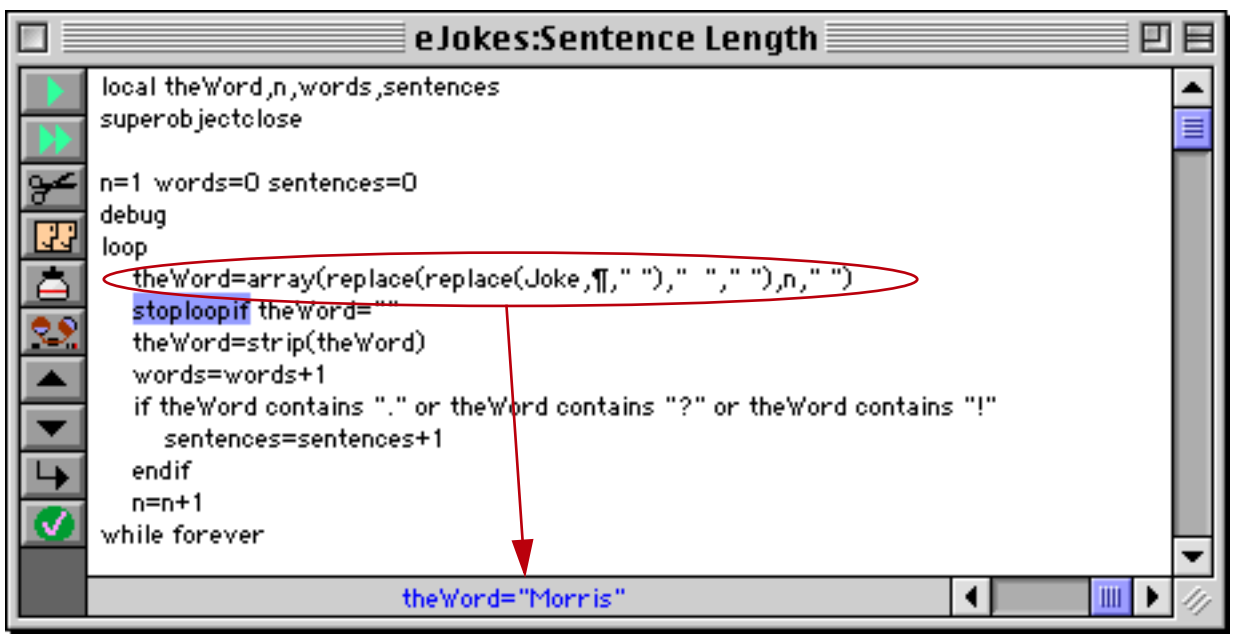
In the course of debugging you may find a problem with your procedure. To fix the problem, just edit the procedure. However, after you change the procedure you can no longer single step or continue the procedure. You must start over again from the top after any kind of change.

Watching Computations

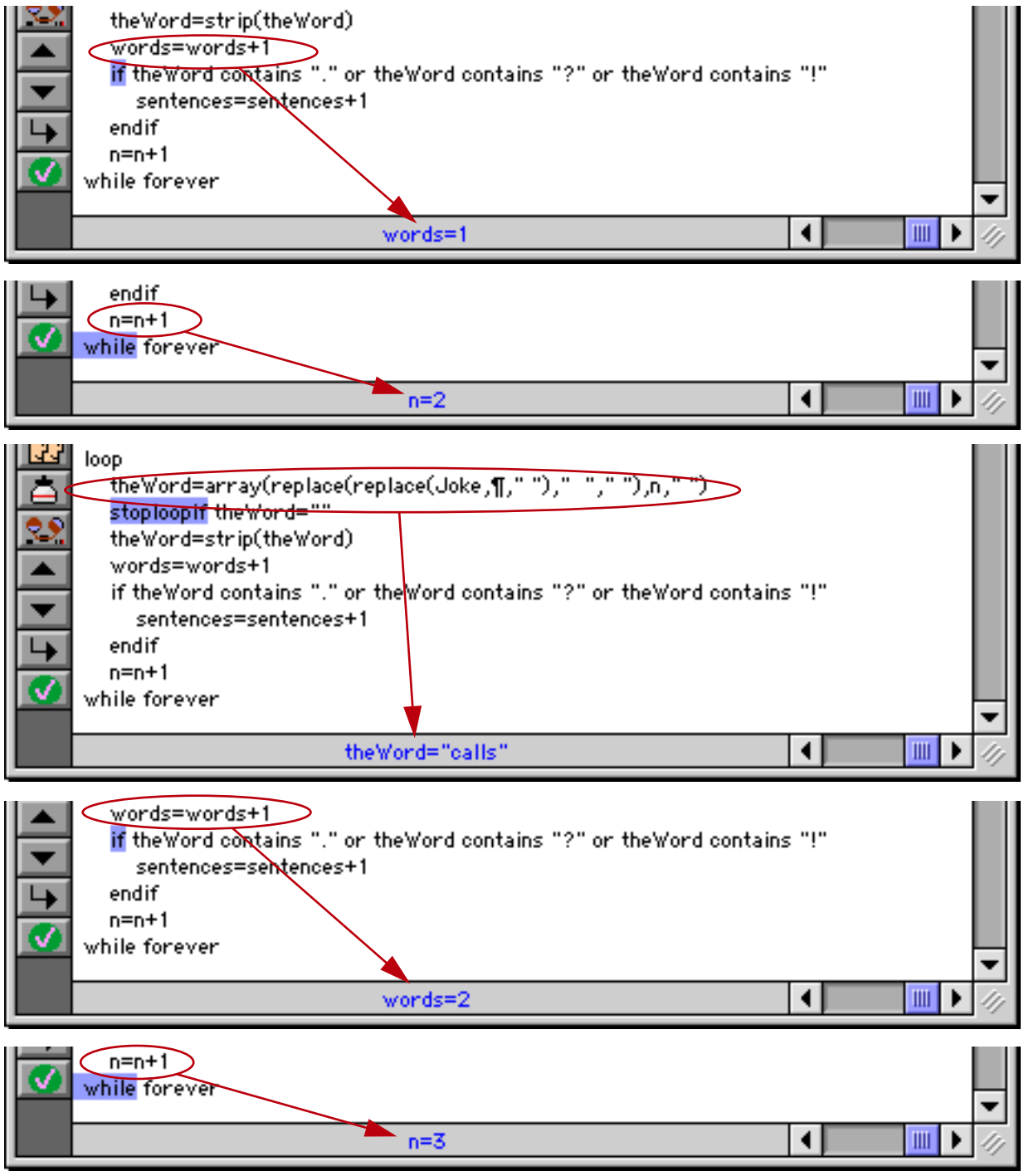
When you single step through a procedure Panorama updates the status bar to show the result of each assignment statement (see "Assignment Statements" on page 1368). This makes it easier to follow along with what is going on in the procedure. For example, the procedure shown below has just stopped after the `debug` statement. The status bar shows the result of the last assignment statement, `sentences=0`.



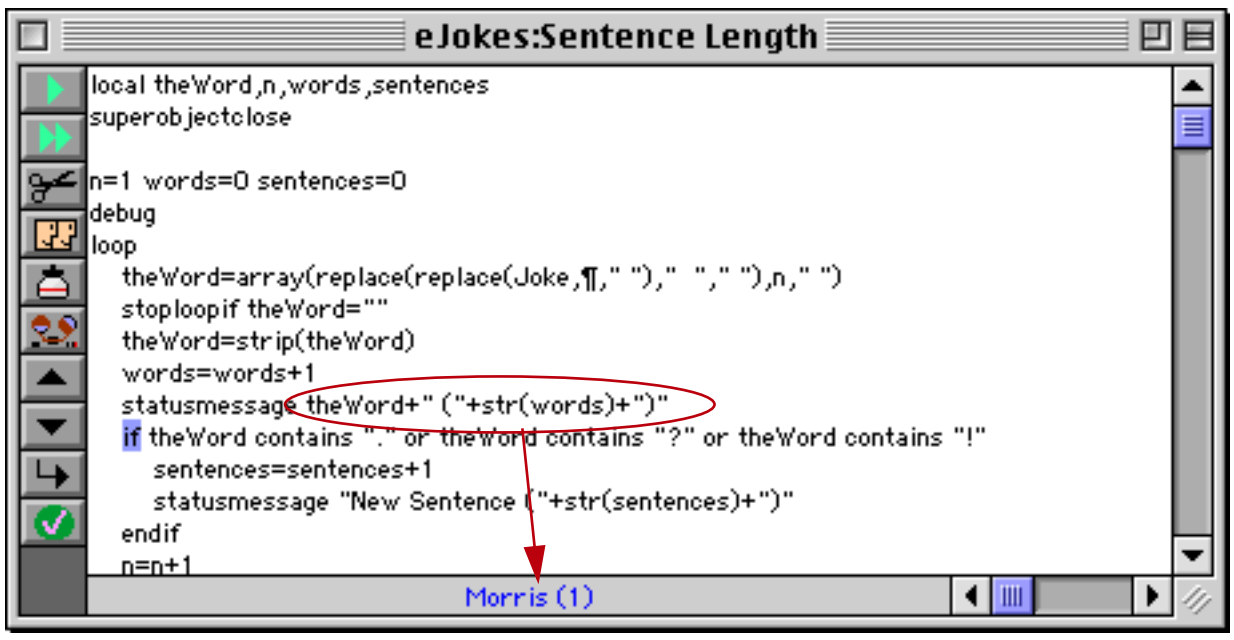
Single stepping twice executes the loop and assignment statements. Again, the status bar shows the result of the assignment, in this case the word Morris.



Each time you single step through an assignment statement the result is shown in the status bar.



You can use the `statusmessage` statement to display any formula in the status bar. This statement is similar to the `message` statement, but instead of displaying the message in an alert it displays it in the status bar, as shown below.



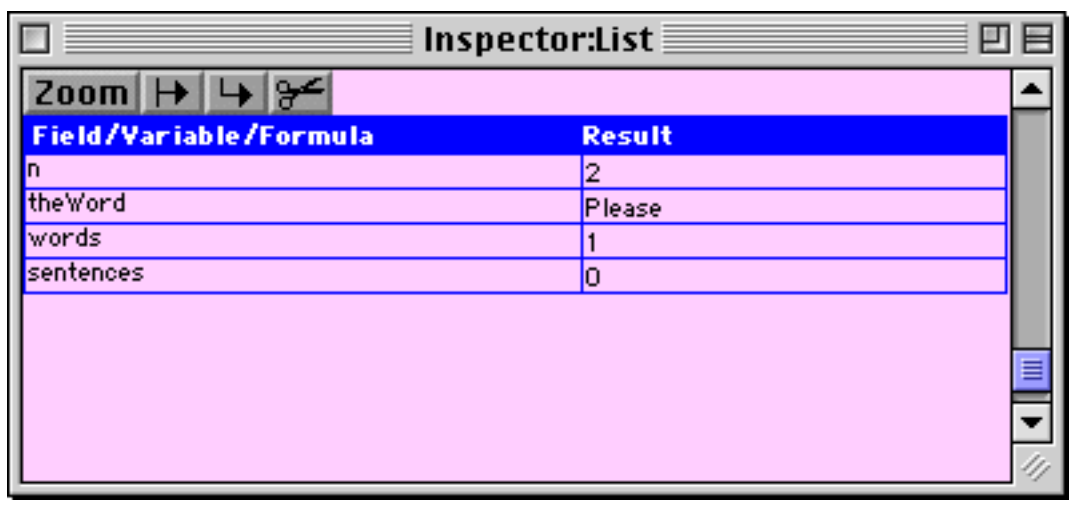
The `statusmessage` statement works any time the procedure window is open, even when the procedure is running at full speed. A strategically placed `statusmessage` statement can be ideal for watching the progress of a loop. For example the `statusmessage` statement in the procedure above will let you watch as the procedure counts the words in each sentence — 1, 2, 3 … . If the procedure window is closed the `statusmessage` statement is simply ignored, so it is safe to leave this statement in your final procedure in case you need it later. (When the window is open the procedure may run slower than normal due to the time taken to update the status bar.)

Using the Inspector to Examine Fields, Variables and Formulas

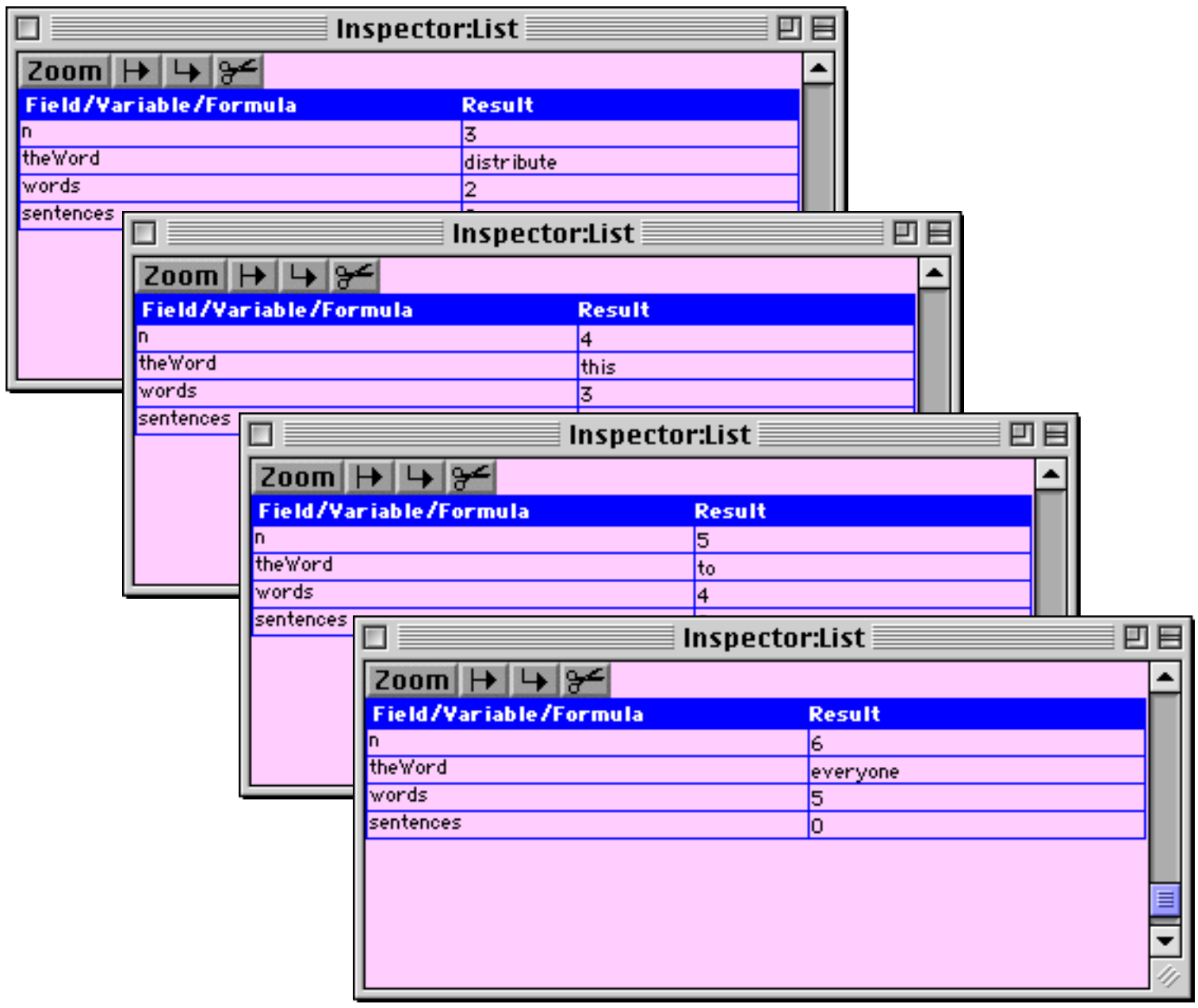
During debugging, you'll often need to examine the contents of fields and variables. If the fields and / or variables you are interested in are not already visible in a form or data sheet window, you can use the Inspector window to watch them. To access this window, choose the **Open Inspector** command in the Debug menu.



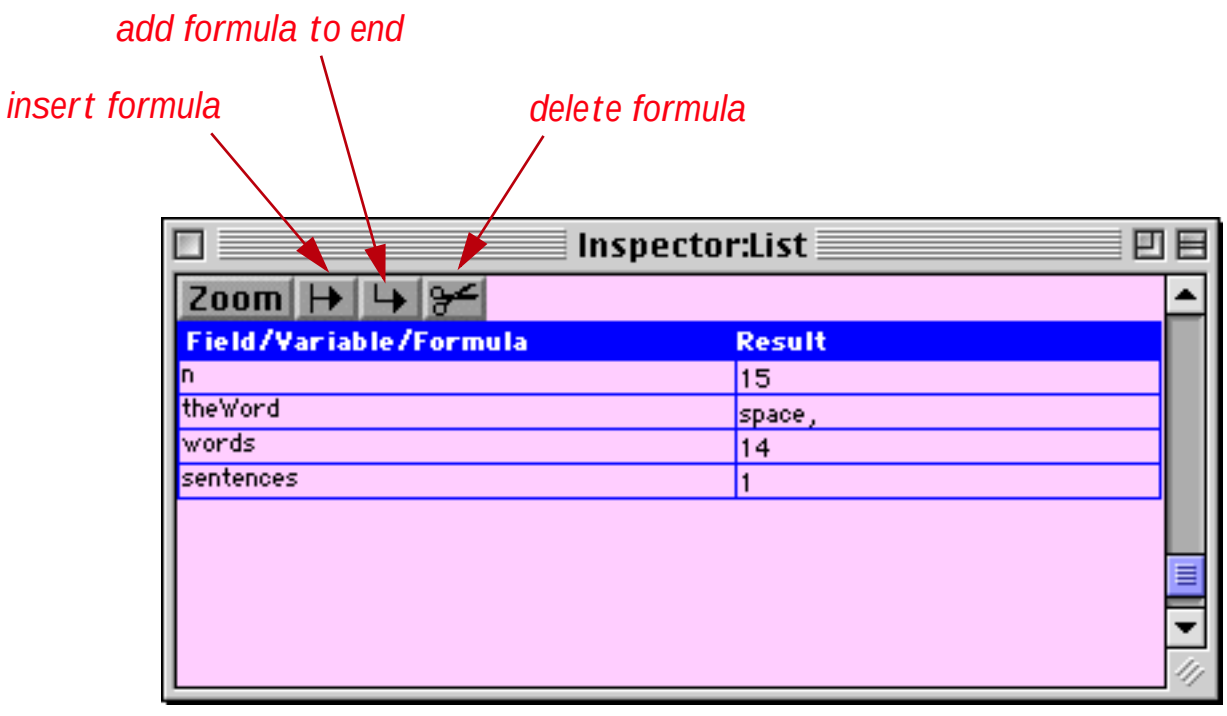
The **Inspector** window displays two columns. The left column is for formulas that you enter. The right column displays the result of each formula.



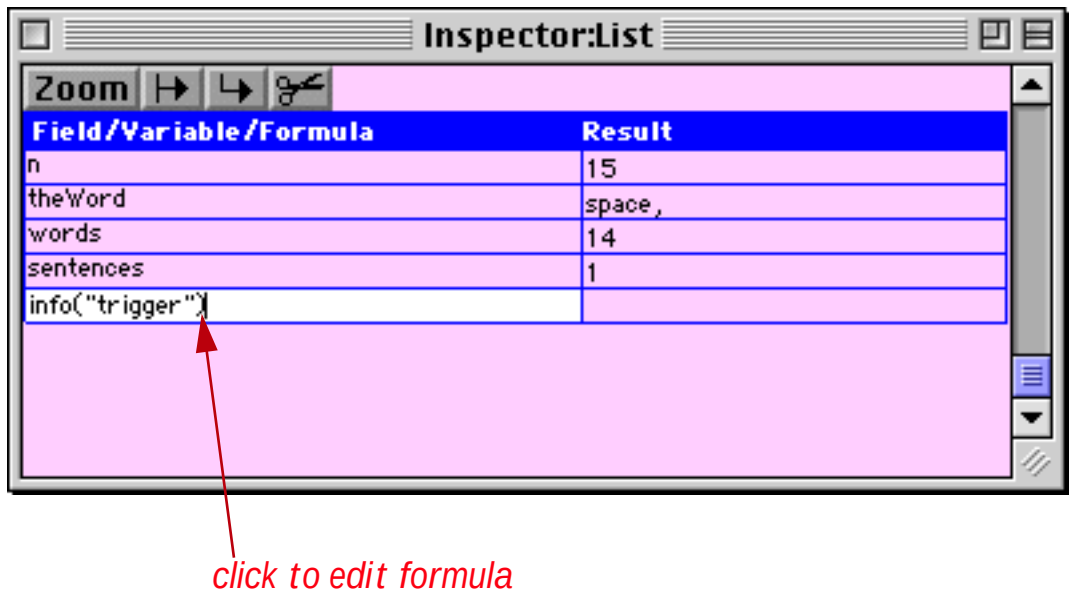
Each formula may consist of a field, a variable, or a more complex formula with fields, variables, and/or functions. The Inspector window calculates and displays the result of each formula. The calculations are updated every time a field or variable is changed, so you can actually watch the data change as you single step or proceed through the procedure.



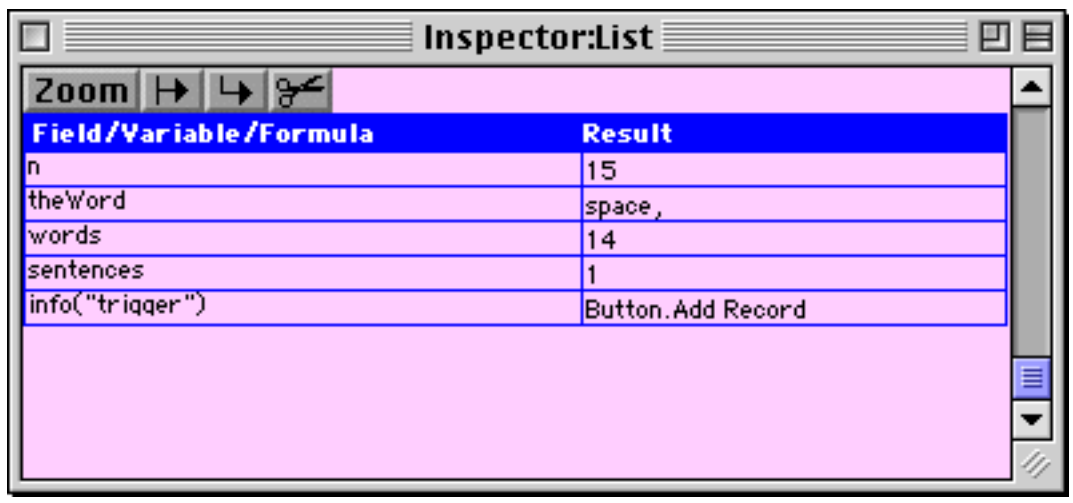
Use the buttons at the top of the window to add and remove formulas.



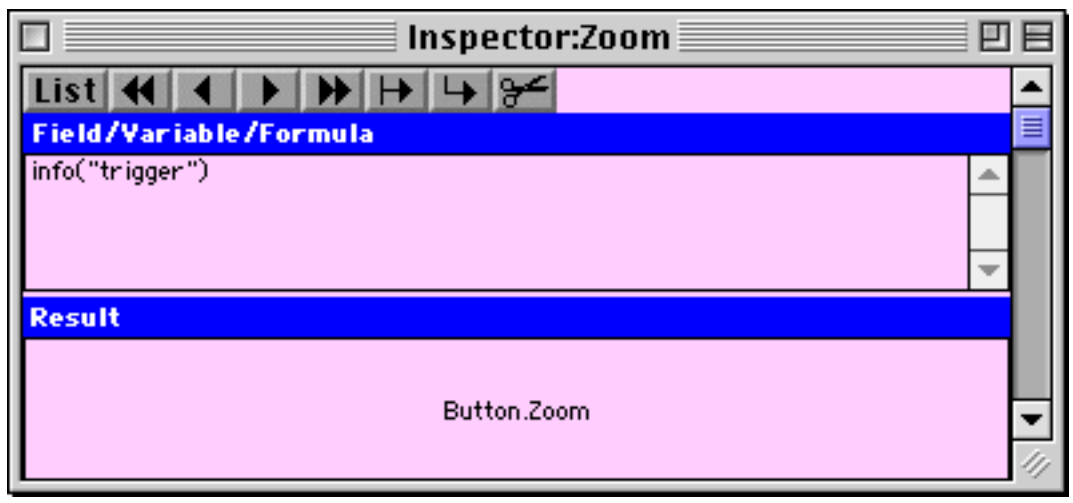
To edit a formula, click on it then begin typing.



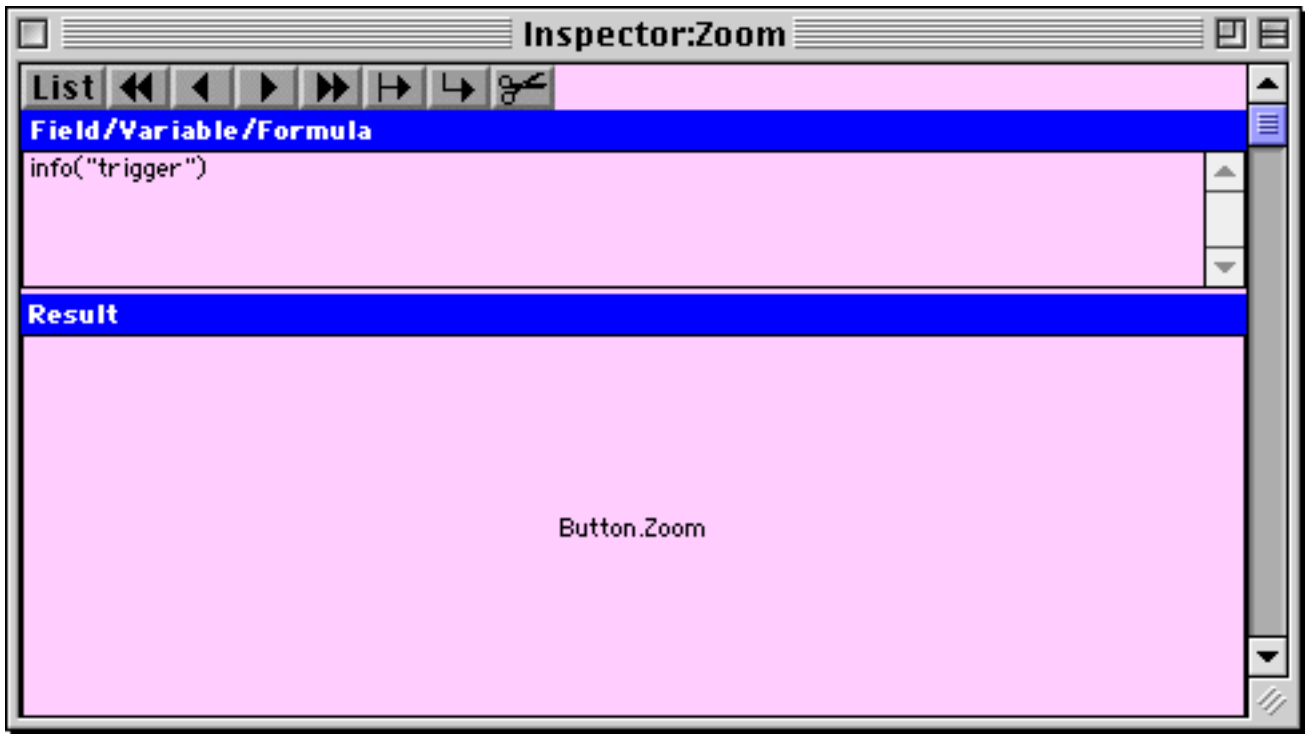
Press **Return** or **Enter** to see the result.



The **Inspector** window normally displays a list of formulas, one line per formula. If you need to display a formula or result that is more than one line high switch to the Zoom mode. In this mode only one formula and result is displayed at time.



If necessary you can enlarge the window to display a result that doesn't fit in the normal window size.



To switch back to the list mode press the **List** button.

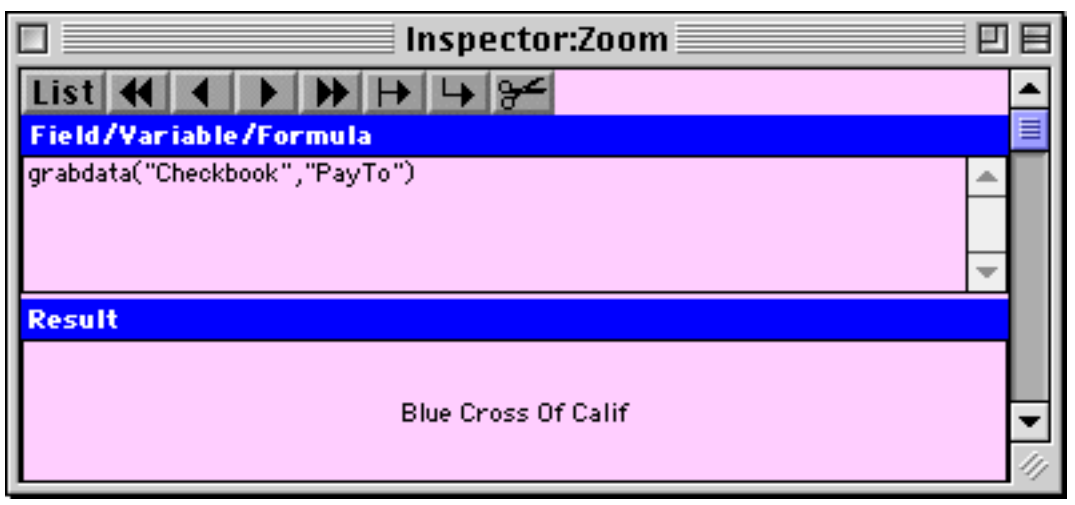
What Fields or Variables can be Displayed?

Sometimes you may enter a formula that looks correct to you, but no result appears in the **Inspector** window. Why does this error occur? This means that the field or variable is not currently accessible to the procedure being debugged. As a procedure runs, it may switch from database to database, and variables may be created and destroyed (see "Variable Accessibility" on page 1373). The **Inspector** window cannot display fields or variables that the procedure cannot access.

If you are trying to display a local variable, that variable may not exist. Local variables only exist while the procedure is actually running. You can only see the contents of a local variable while the procedure is stopped in the middle or single stepping. Before the procedure is started, or after it is finished, the local variable does not exist and cannot be inspected.

When displaying fields, the **Inspector** window always displays the value of the field in the current debug database. Usually this is the database that contains the procedure. However, if the procedure switches to a different database (with the `window` or `openfile` statement), the **Inspector** window will also switch to the new database. If no result appears, you are probably attempting to display a field in another database.

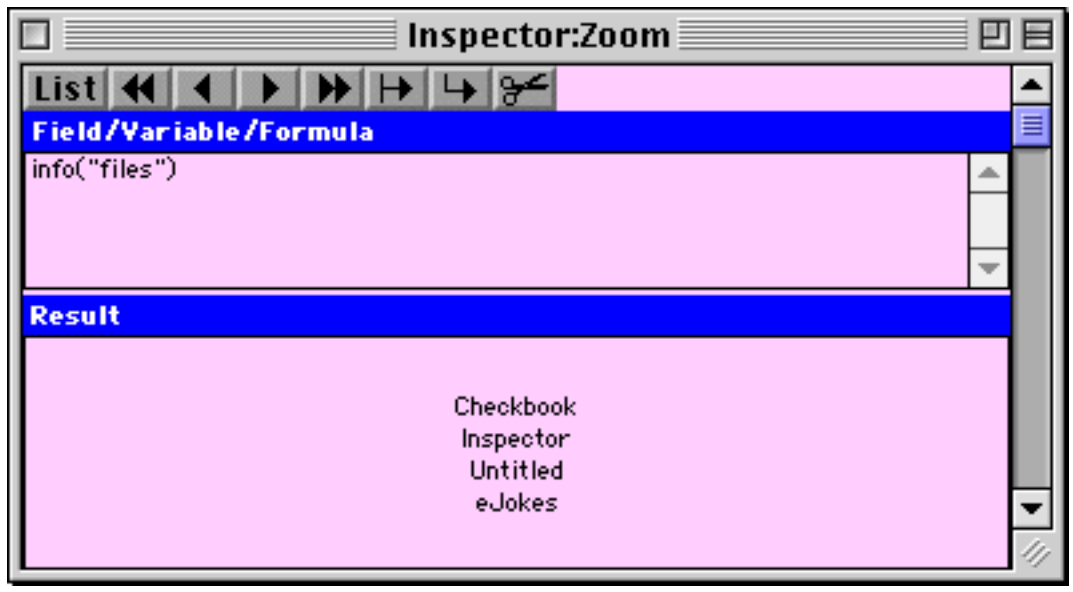
Use the `grabdata(` function to display the value of a particular field in a particular database no matter what database is being debugged. For example, to always display the PayTo field in the Checkbook database, type the formula `grabdata("Checkbook","PayTo")` into the **Inspector** window as shown below.



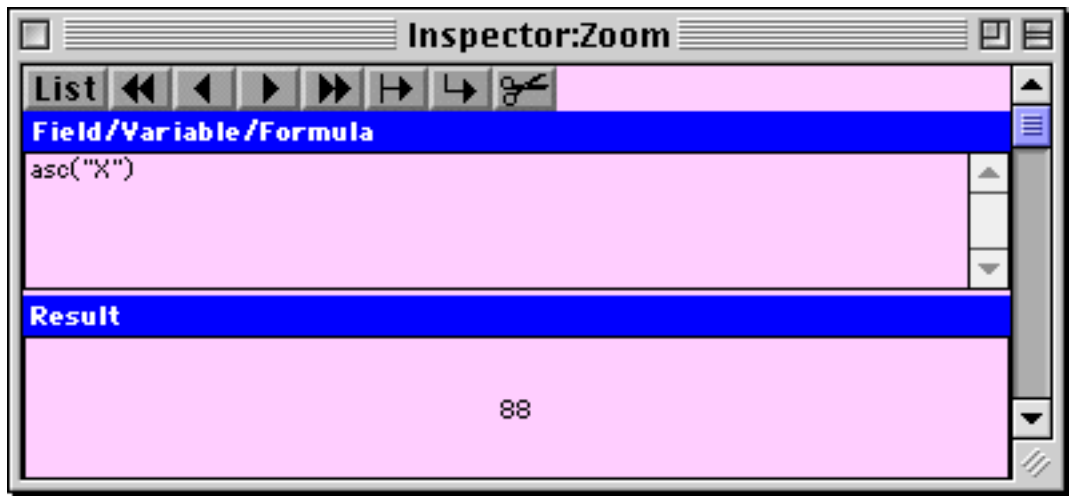
As long as the Checkbook database remains open in memory you'll be able to see the contents of this field.

Displaying Functions

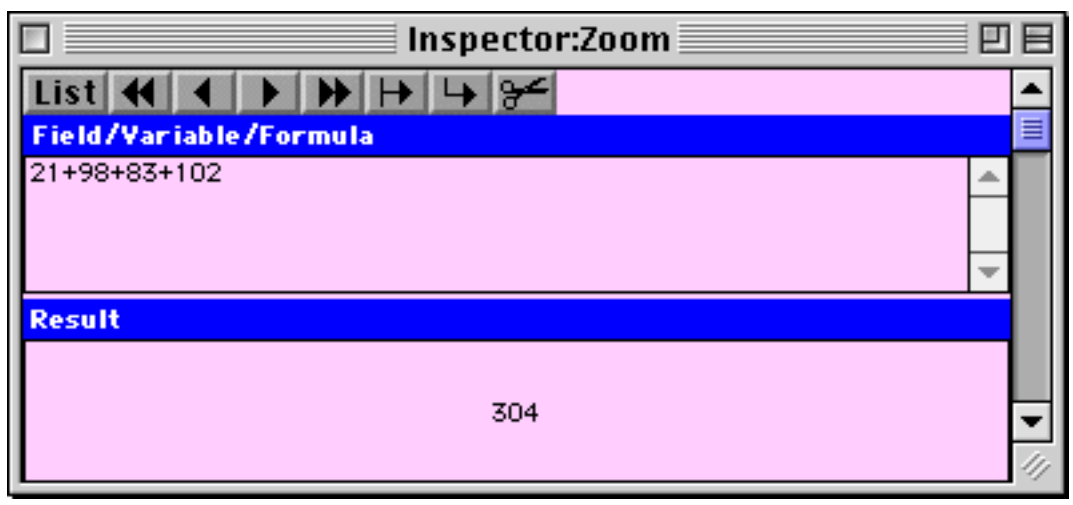
Don't forget that the **Inspector** window can display any formula, not just fields and variables. One handy application for the Inspector window is to look at the results of info(functions. For example, you can display the function `info("trigger")` to see how a procedure was triggered, or `info("files")` to see what databases are currently open.



You can use the `asc(` function to look up the ASCII value of a character (see "Characters and ASCII Values on page 1230").



Or you can use the Inspector as a handy calculator.



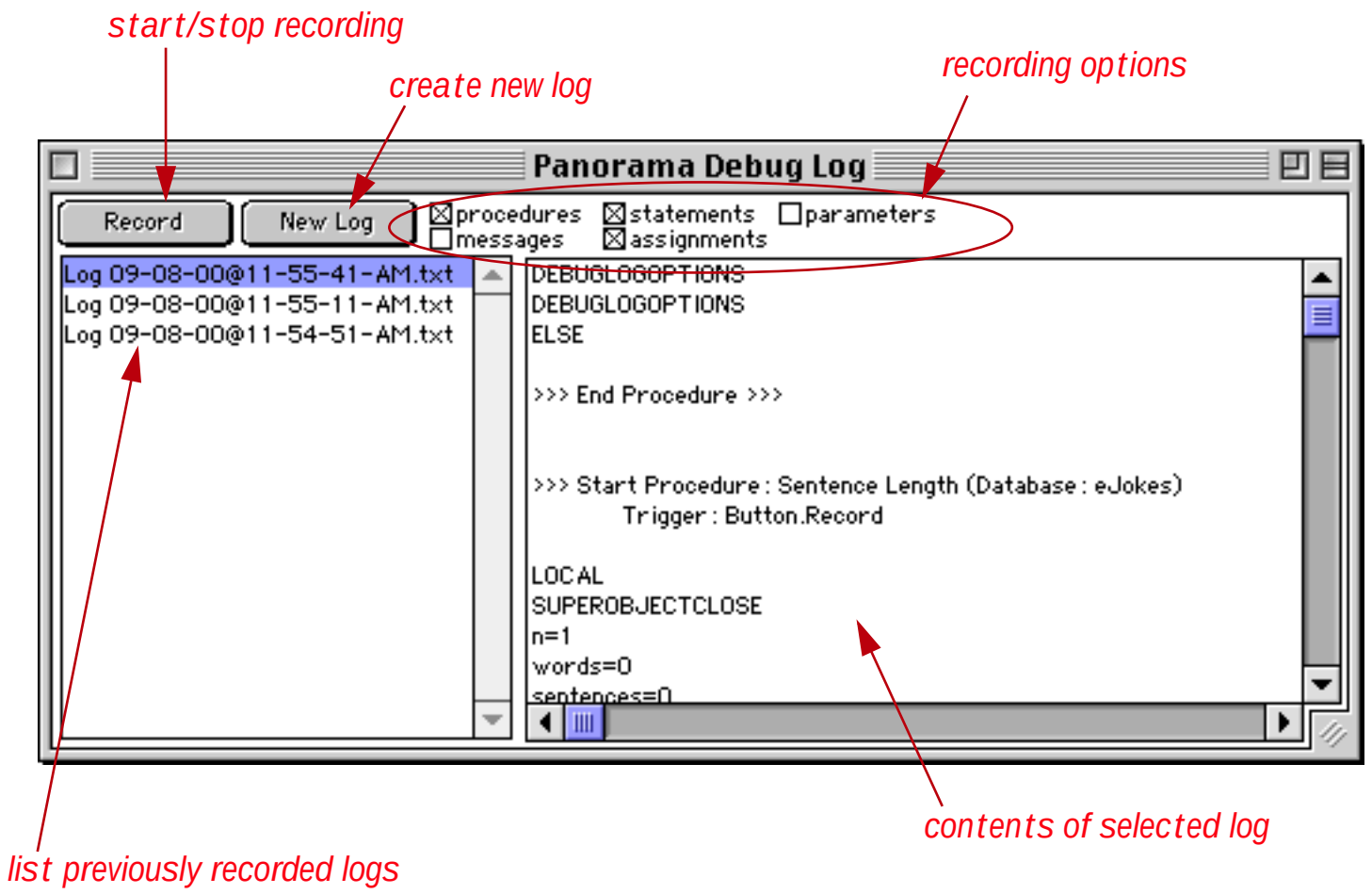
Note: The **Formula Wizard** also can be used as a handy calculator. See "Using the Formula Wizard on page 1175".

Procedure Debug Log

The procedure debug log was originally developed as an “in house” tool to help debug Panorama itself. It has proved so useful that we have decided to document and make it available for general use. When the debug log is in use Panorama records procedure activity in a text file. Later you can review the text file to trace the actions of your procedure. Although Panorama rarely crashes, when it does the debug log comes in very handy, because it will record the steps taken right up to the crash. This allows you to find out exactly what statement is causing the crash (which explains why this debug log is so useful for our in-house programming of Panorama itself.)

The Procedure Log Window

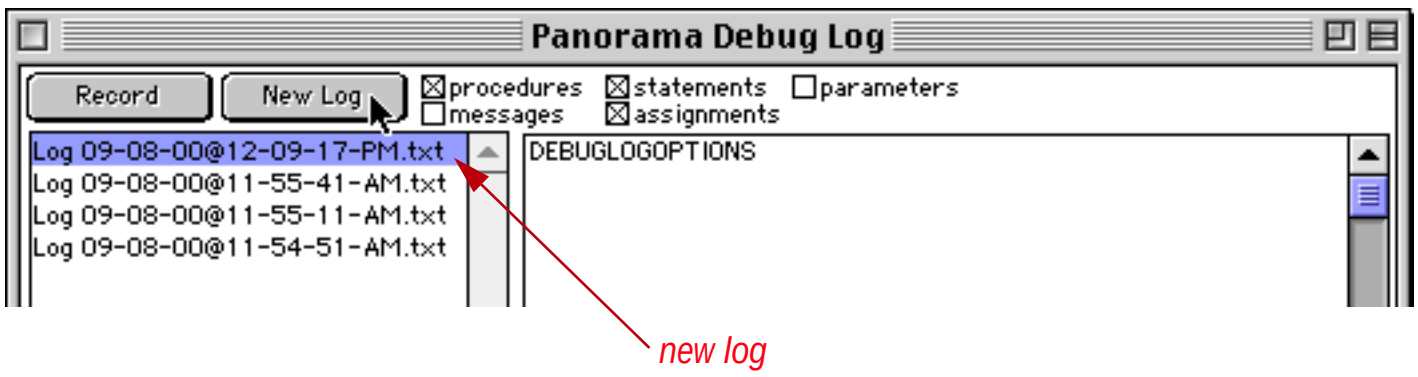
To open the log window choose **Debug Log** from the Wizards menu. When you first open the debug log it looks something like this.



The list on the left hand side of the window shows each of the previously recorded logs. Each log is date and time stamped.

Recording a New Log

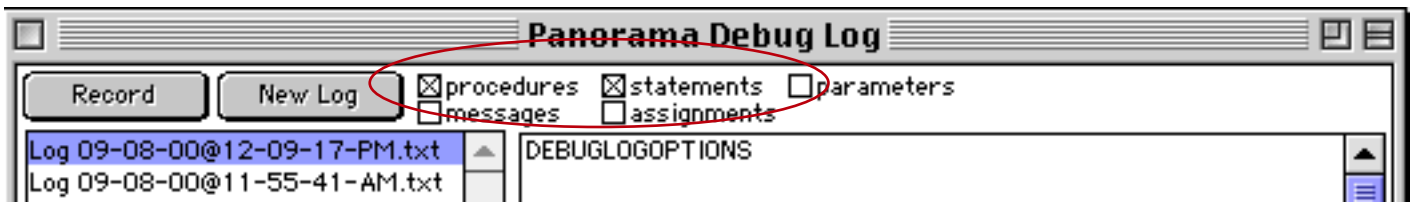
To record a new log, start by pressing the **New Log** button. A new log will be added to the top of the list.



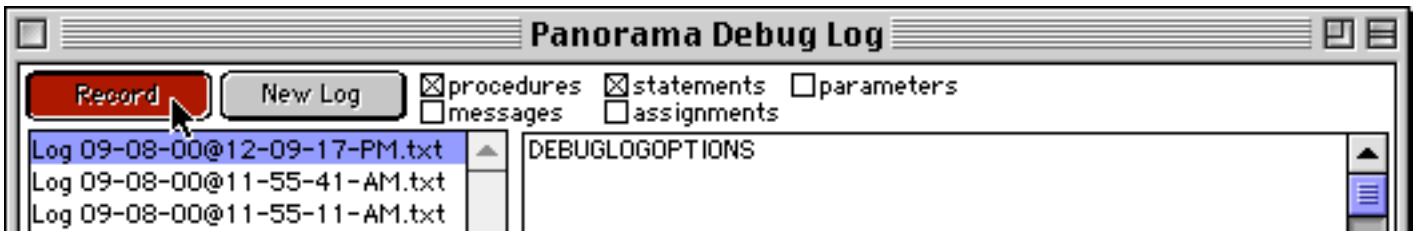
Next, select the recording options for the new log.

Option	Description
procedures	When this option is selected the log will record each time a new procedure starts or finishes, either by being triggered by a menu or button or as a subroutine call. This option can be handy if you are not sure what procedure is triggered by a button. Simply turn on the Debug Log, start recording and press the button. Then check the log to find out which procedure was triggered.
statements	When this option is selected the log will record each statement that is executed. Only the statement itself is recorded, not any parameters (see next section).
parameters	When this option is selected the log will record the values of each statement parameter (see “Decoding Parameters and Assignment Statements” on page 1449).
messages	When this option is selected the log will record each **logmessage** statement (see “The Log-Message Statement” on page 1450).
assignments	When this option is selected the log will record each assignment statement (A=B, etc.). See “Decoding Parameters and Assignment Statements” on page 1449.

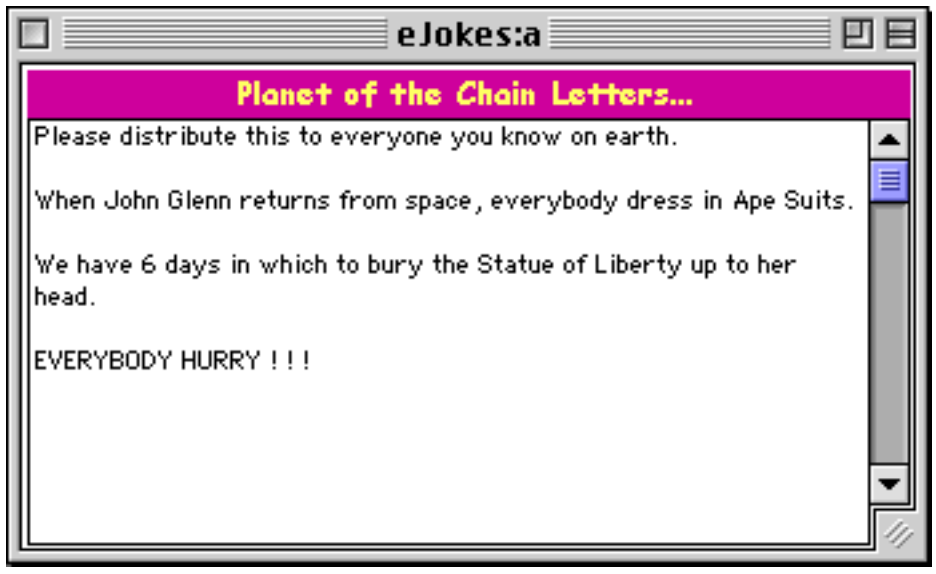
For our first log we’ll record only the procedures and statements.



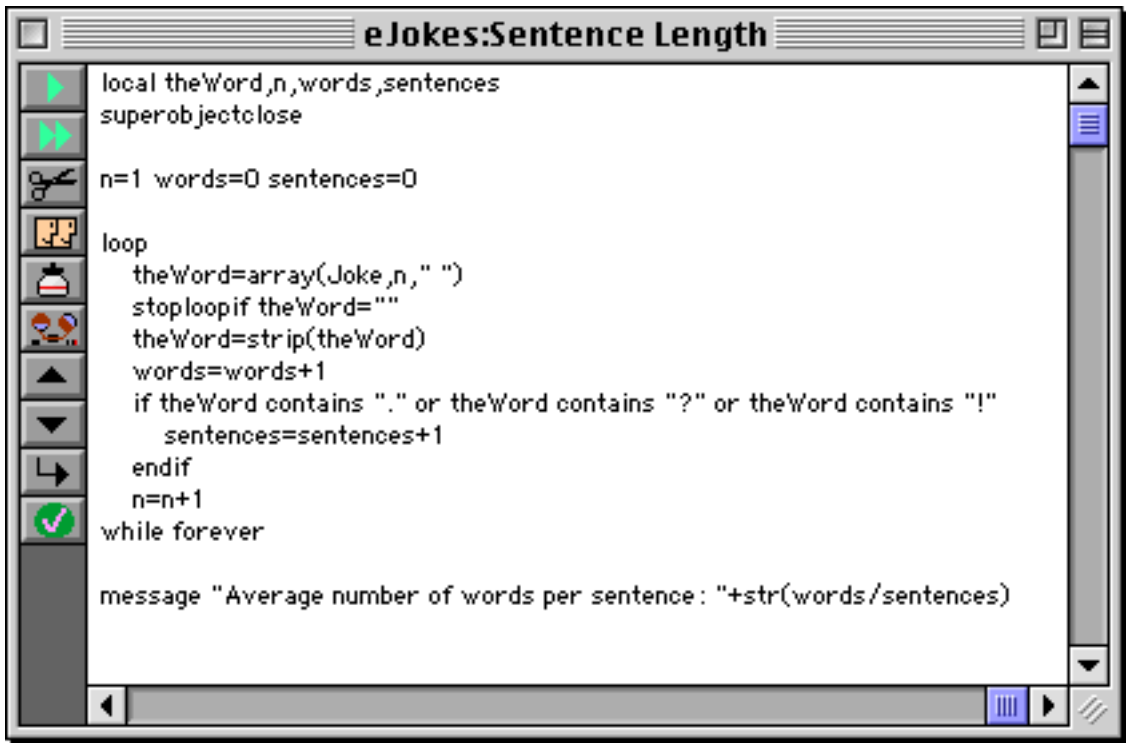
Once the options are set press the **Record** button to start recording. The button will highlight to show that it is recording.



Open the database that contains the procedure you want to test (if it is already open, click on it to bring it to the front.

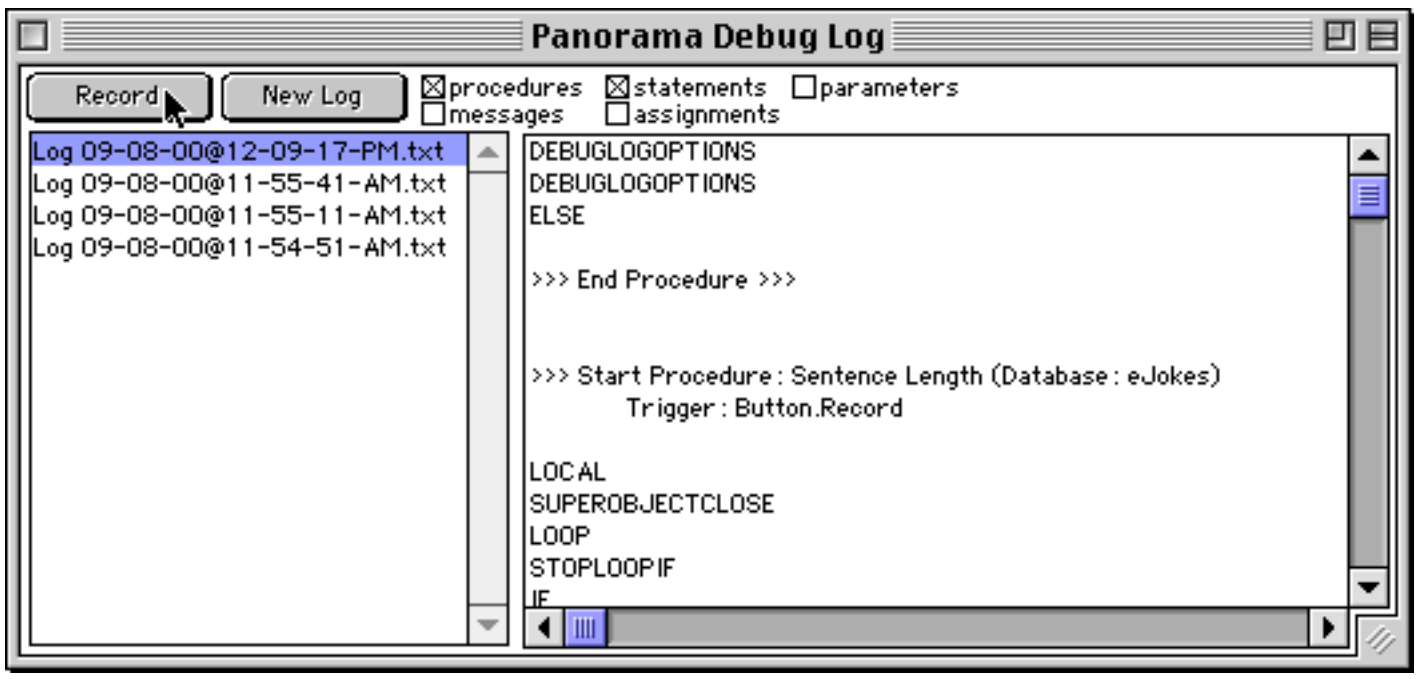


Now perform whatever action it takes to trigger the procedure you want to test — choose the procedure from the **Action** menu, press on a button, enter data, whatever (see "50 Ways to Trigger a Procedure" on page 1458). In this case we are going to test a procedure named **Sentence Length** in the **Action** menu. Here is the text of the procedure.

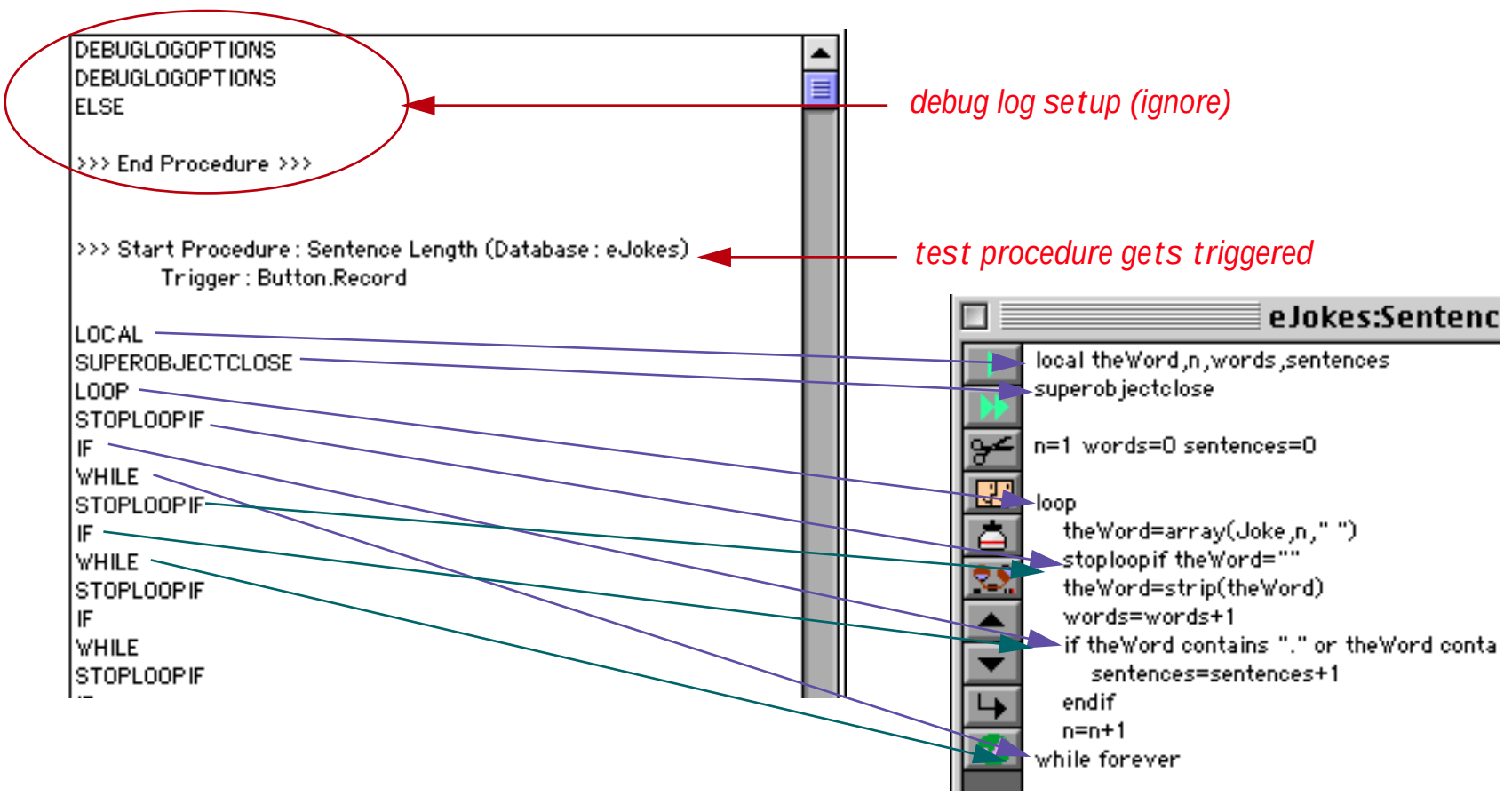


Note: Depending on the recording options you have selected, **the procedure may run much slower than it usually does.** The recording process slows down Panorama's speed by an order of magnitude or more.

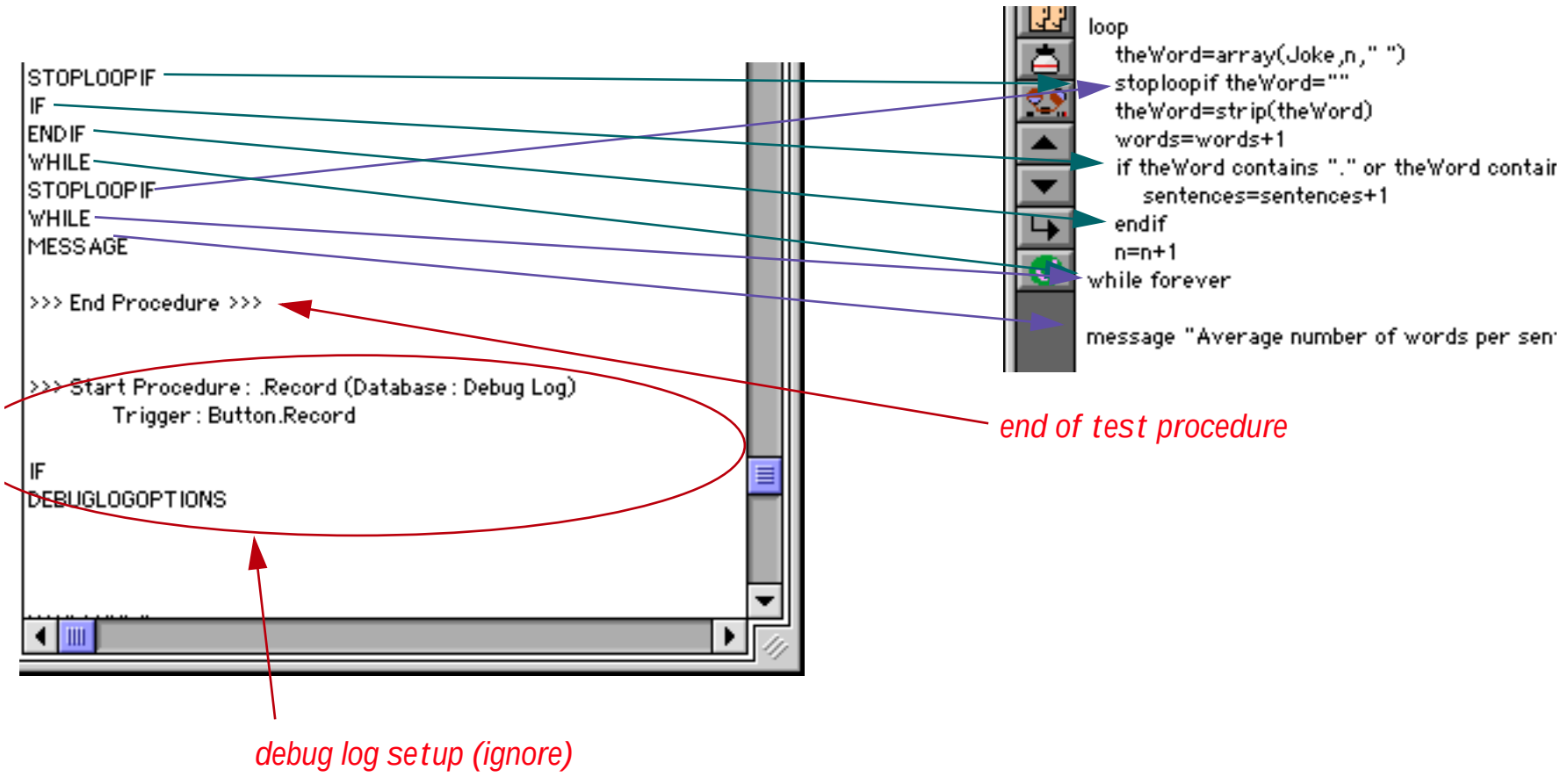
When the procedure has finished running, click on the Debug Log window and press the **Record** button. The newly recorded log appears on the right hand side of the window.



At the beginning of the log you may see a few lines caused by the debug log database actually recording itself. You should ignore these lines. Your recording starts with your test procedure being triggered. After that you will see a recording of each statement the procedure performed.

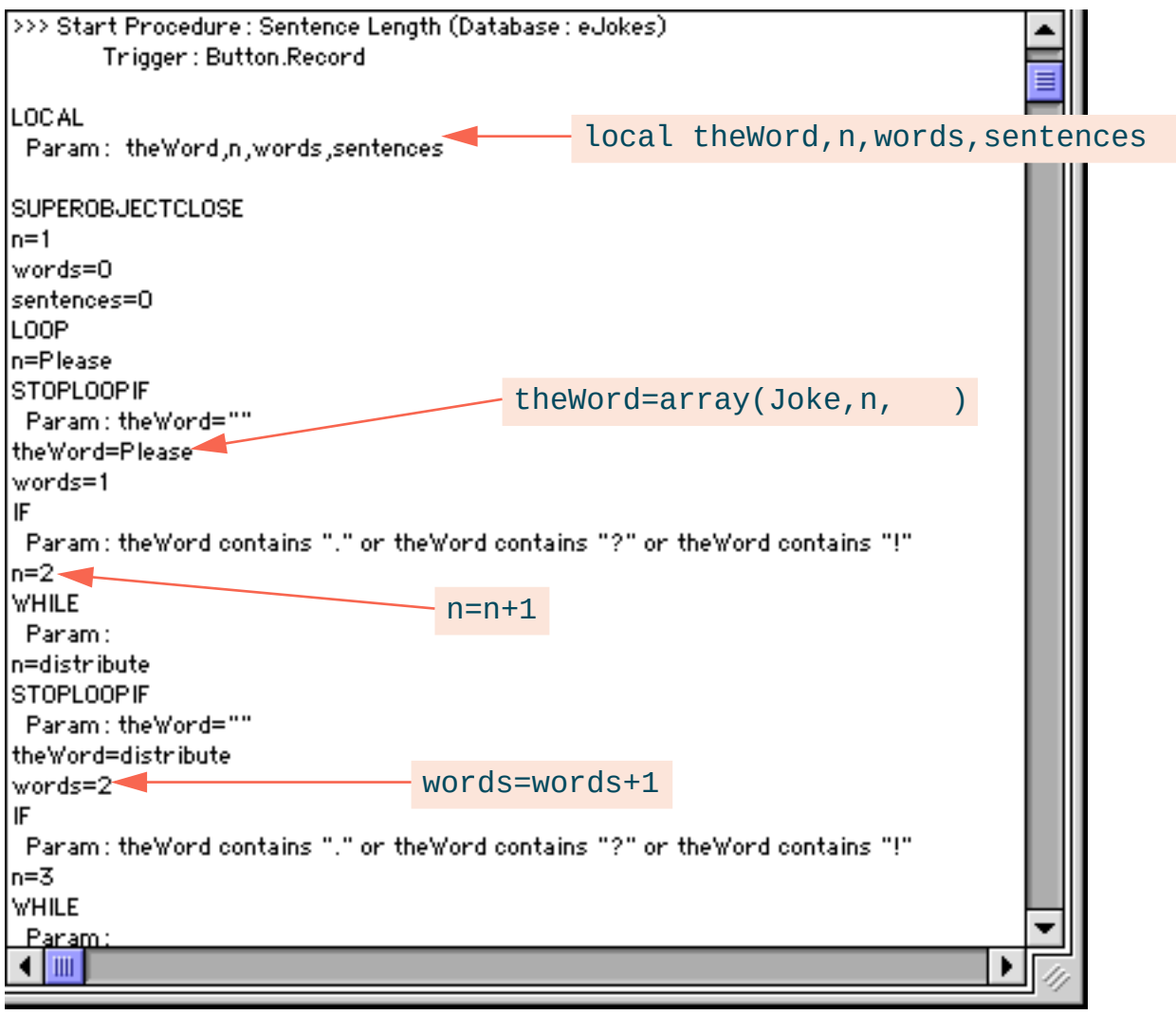


You can continue to trace the steps the procedure took all the way to the end.



Decoding Parameters and Assignment Statements

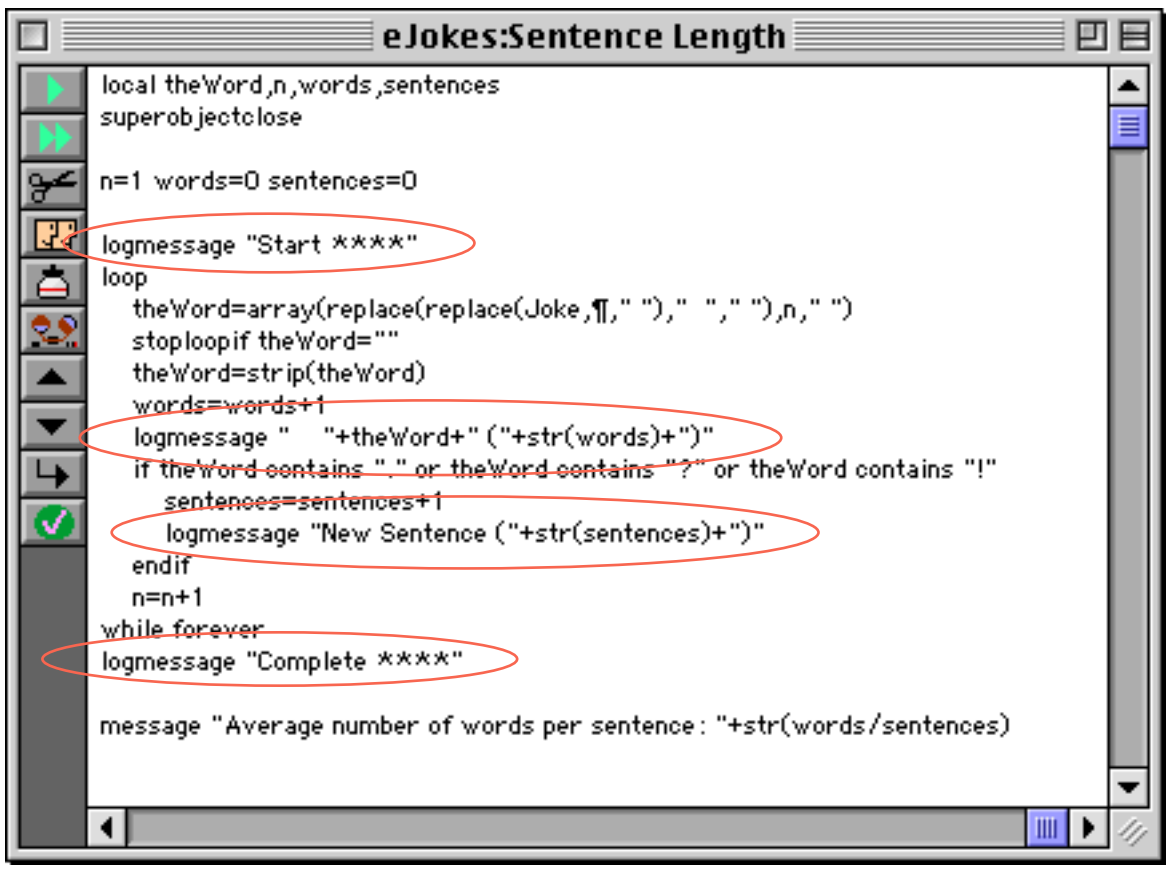
When the Parameters and / or Assignment options are enabled the log will contain much more information.



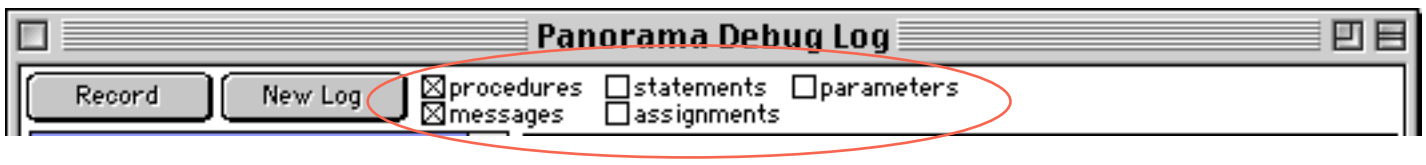
Each statement parameter is logged with the word Param: followed by the value of the parameter. Each assignment is logged as the destination (words=) followed by the value that the procedure is putting into the destination (Please). Notice that in either case the procedure is logging the value and not the formula used to produce the value, for example n=2, not n=n+1.

The LogMessage Statement

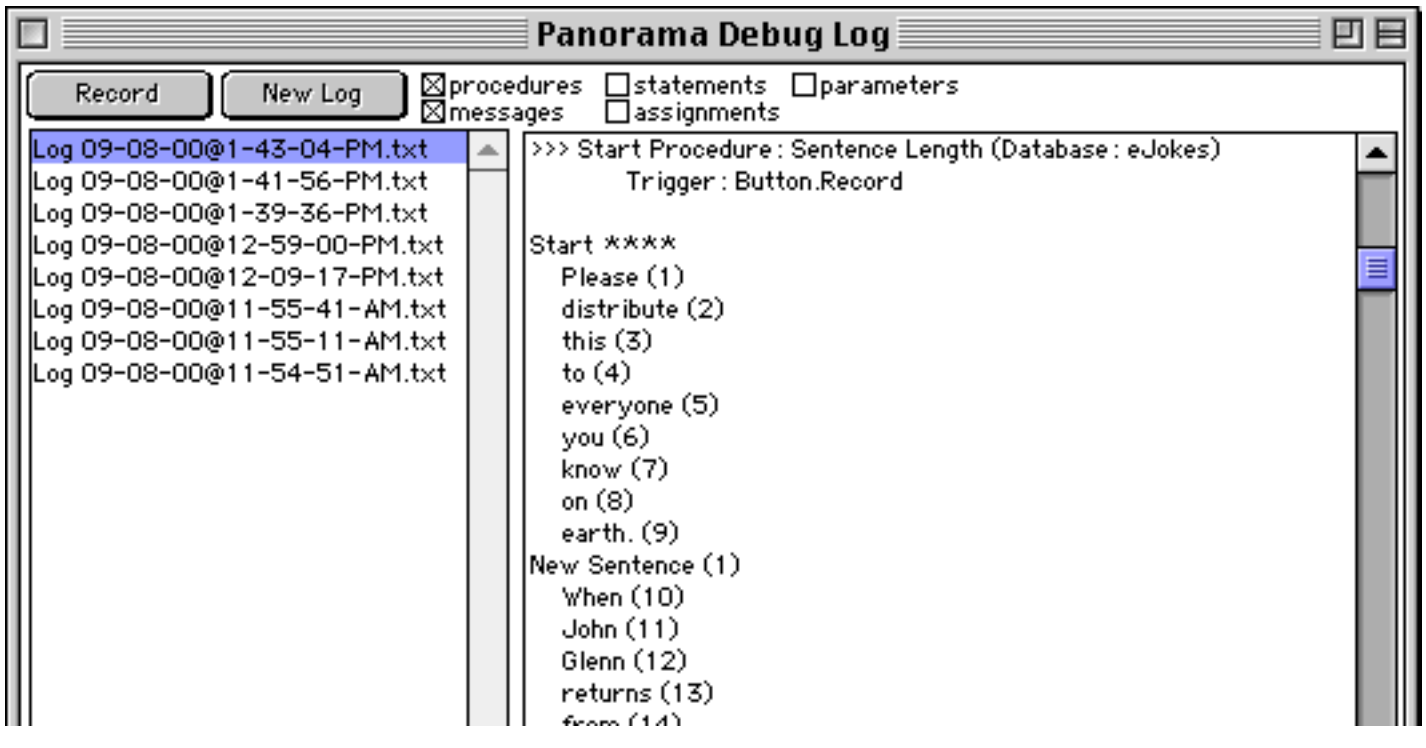
The debug log can quickly generate reams and reams of information that can be tedious to wade through. By inserting the `logmessage` statement in strategic locations you can create a log that shows only the information that is useful to you. Here is a revised version of the procedure with four `logmessage` statements added at strategic spots.



Before recording we'll adjust the log options to only record messages, not statements, parameters or assignments.



The revised log shows only the messages. You can easily see the flow of the procedure as it scans through each word and sentence.



If you look closely at the procedure above (with the `logmessage` statements) you'll notice that the assignment statement at the beginning of the loop is different than in the previous examples.

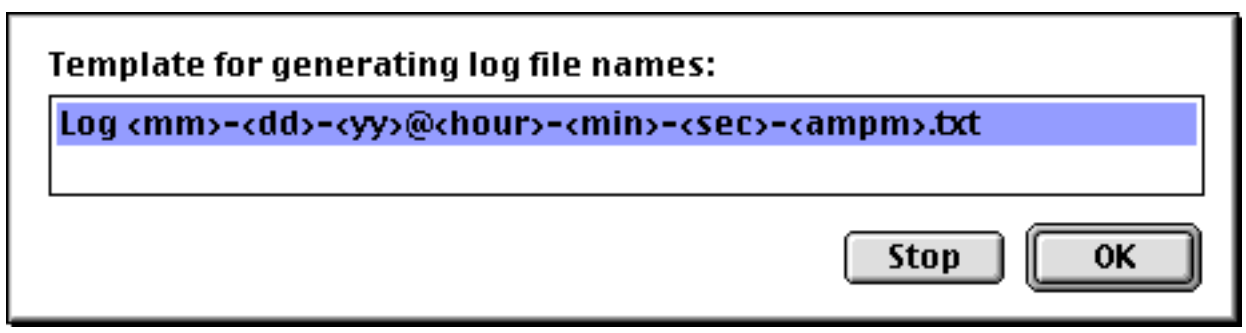
```
theWord=array(replace(replace(Joke,¶," "),"  "," "),n," ")
```

The reason for this change is that in the process of creating the screen shots to demonstrate the `logmessage` statement the log actually showed us that there was a bug in the procedure that caused it to count the number of words incorrectly! The log created with the `logmessage` statements made this bug instantly visible, and hopefully it can do the same for your bugs too!

The Log Menu

When you are finished with a log you can delete it by selecting the log and choosing the **Delete Selected Logs** command from the **Log** menu. To delete every log choose **Delete All Logs** from the **Log** menu.

Panorama normally date and time stamps each log file. You can customize how the log file is created by using the **Edit Log File Template** command from the **Log** menu. This command opens a template that allows you to customize the date/time stamp.



You can customize this template by re-arranging the items.

Cross Referencing

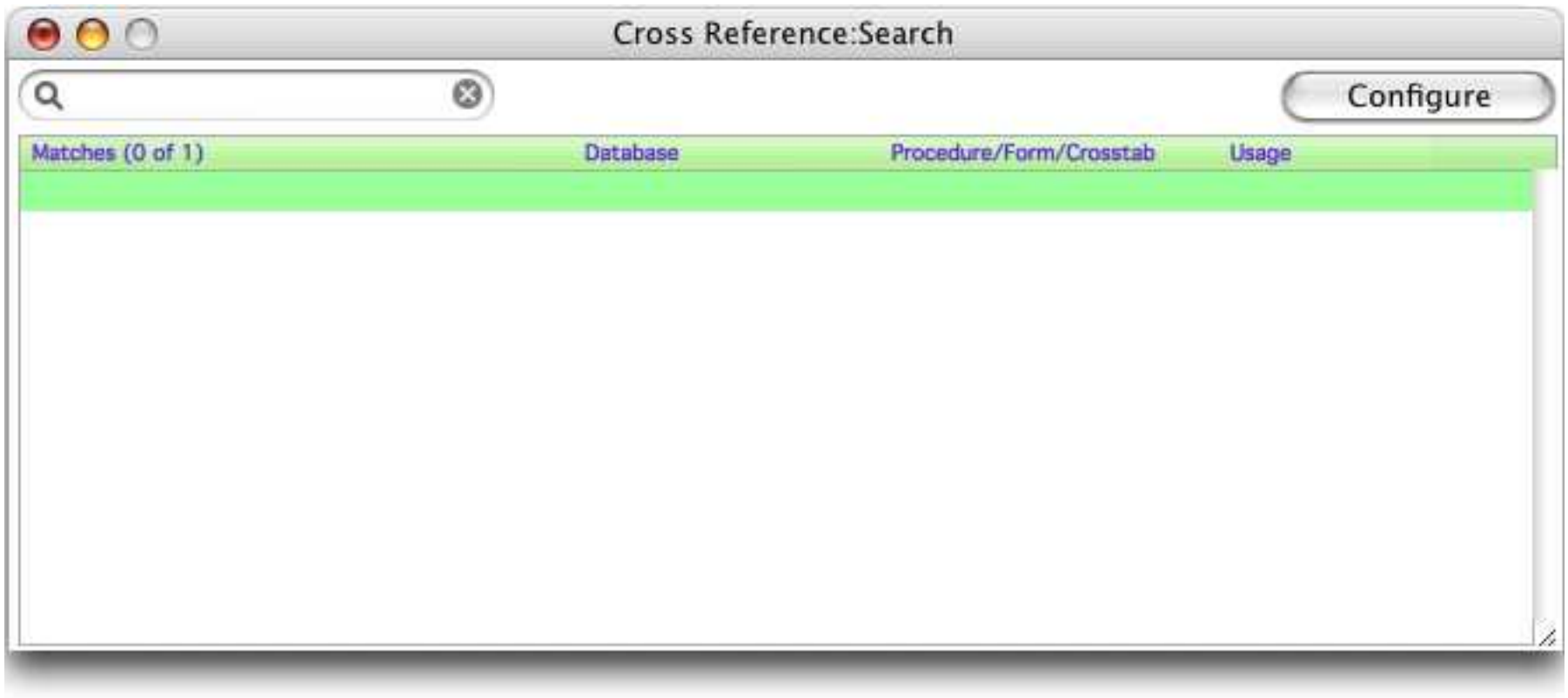
A complex real world system (accounting, reservations, order entry, etc.) created with Panorama may involve a dozen files with hundreds of fields, variables, procedures, forms, etc. Keeping track of all this information in your head can be a monumental task.

Panorama's **Cross Reference** database feature can help make this task manageable. A cross reference database keeps track of all the items in one or more databases: every field, every variable, every procedure, every form—every everything. Not only does the cross reference database keep track of where these items are defined, but also everywhere they are used. For example, suppose your database has a field named Title. A cross reference database can tell you that this field is used in the Entry, List, and Label forms, and is also used in the procedures .NewRecord and Search. Or you could use a cross reference database to find out that the .LastYear procedure is triggered by buttons in the Entry and Annual Report forms. As your database applications become more complicated you'll find that a cross reference database is an invaluable tool to help you sift through a mountain of databases and programming.

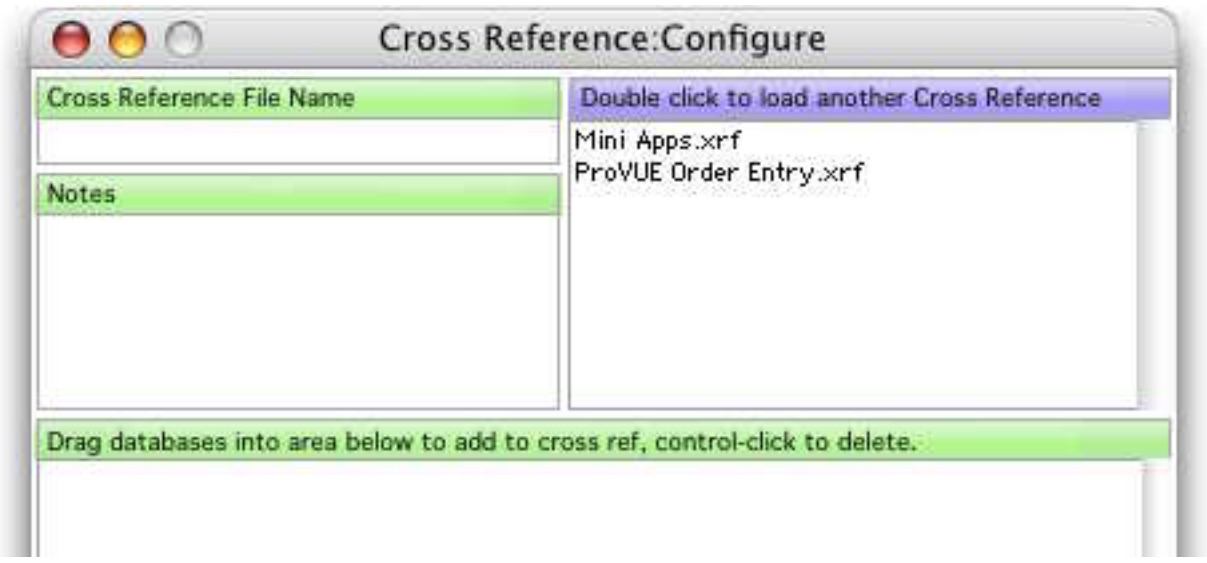
Note: You can also use the **View Wizard** to search through procedures (see "Searching All Procedures" on page 232). However unlike a **Cross Reference** database the **View Wizard** cannot search through forms, crosstabs or the design sheet.

The Cross Reference Wizard

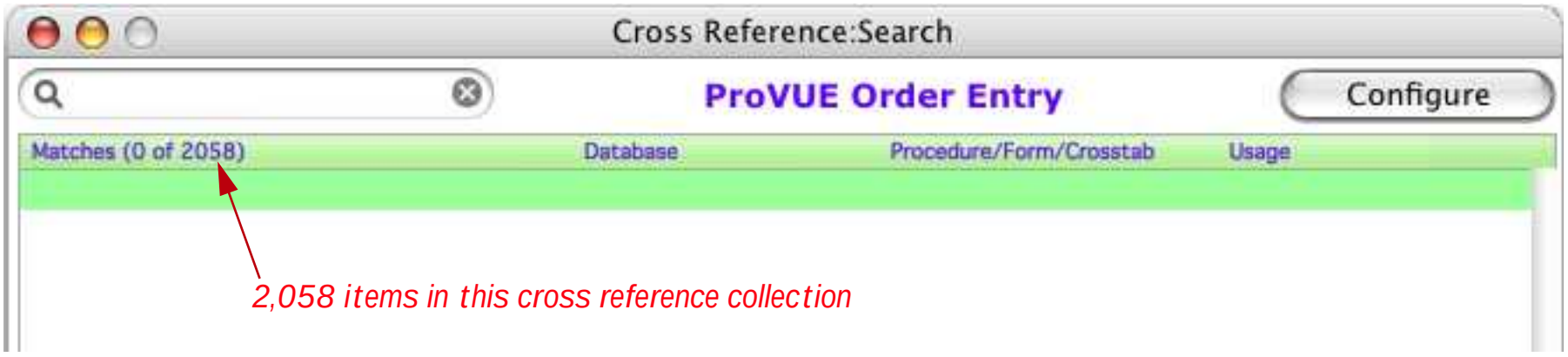
You'll find the **Cross Reference** wizard in the Developer Tools submenu of the Wizards menu.



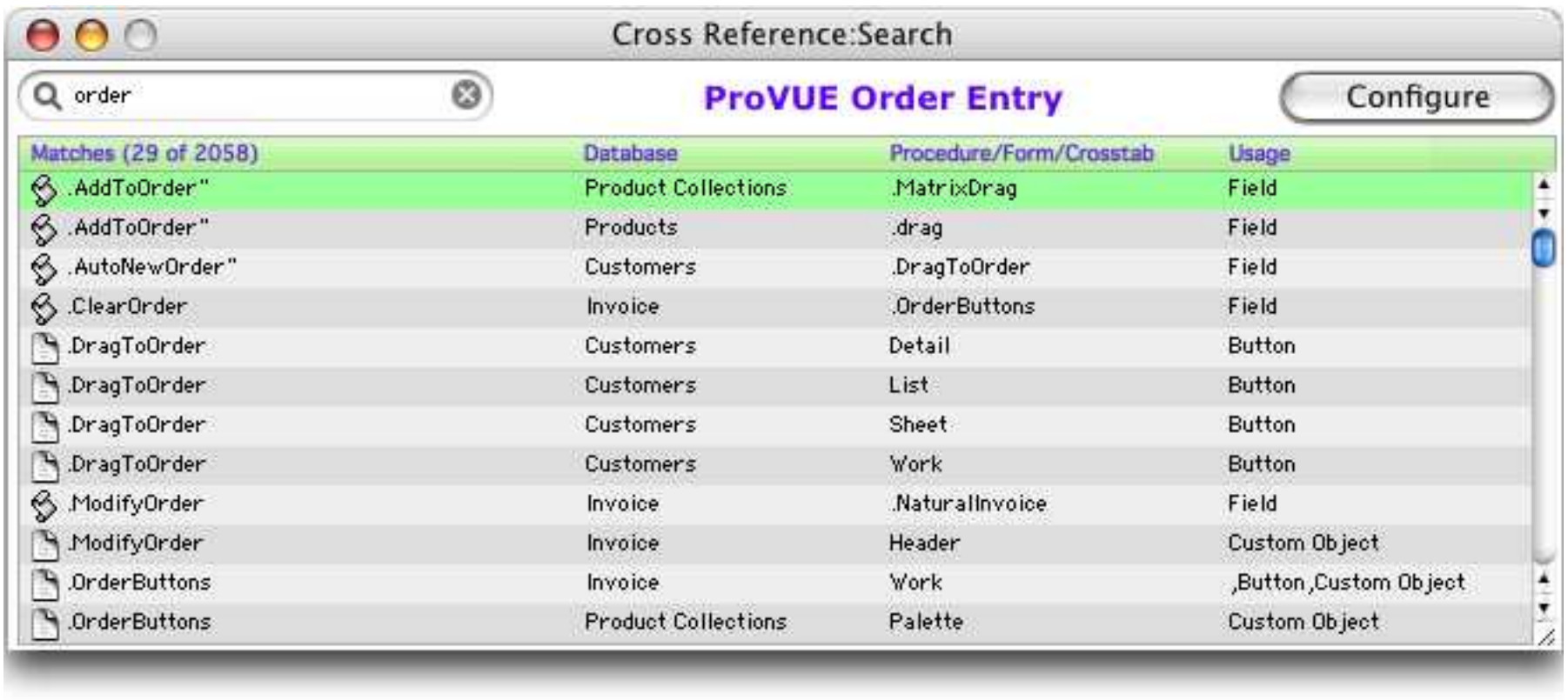
Before you can use this wizard you'll need to configure it, so press the **Configure** button.



For now we'll simply use one of the previously saved cross references that comes with Panorama. Double click on ProVUE Order Entry.xrf. The previously configured cross reference appears in the main window. This cross reference contains a combined index for six databases.



This cross reference contains a combined index for six databases. As you start typing into the search box in the upper left, the wizard shows you where that text appears in any of the forms, crosstabs, in the six databases. There are 29 fields, forms or procedures that contain the phrase "order."



The wizard display contains four columns. The leftmost column contains the full contents of the text that matched your search, along with an icon that identifies the type of view.

	Field (data sheet)
	Procedure (script)
	Form
	Crosstab

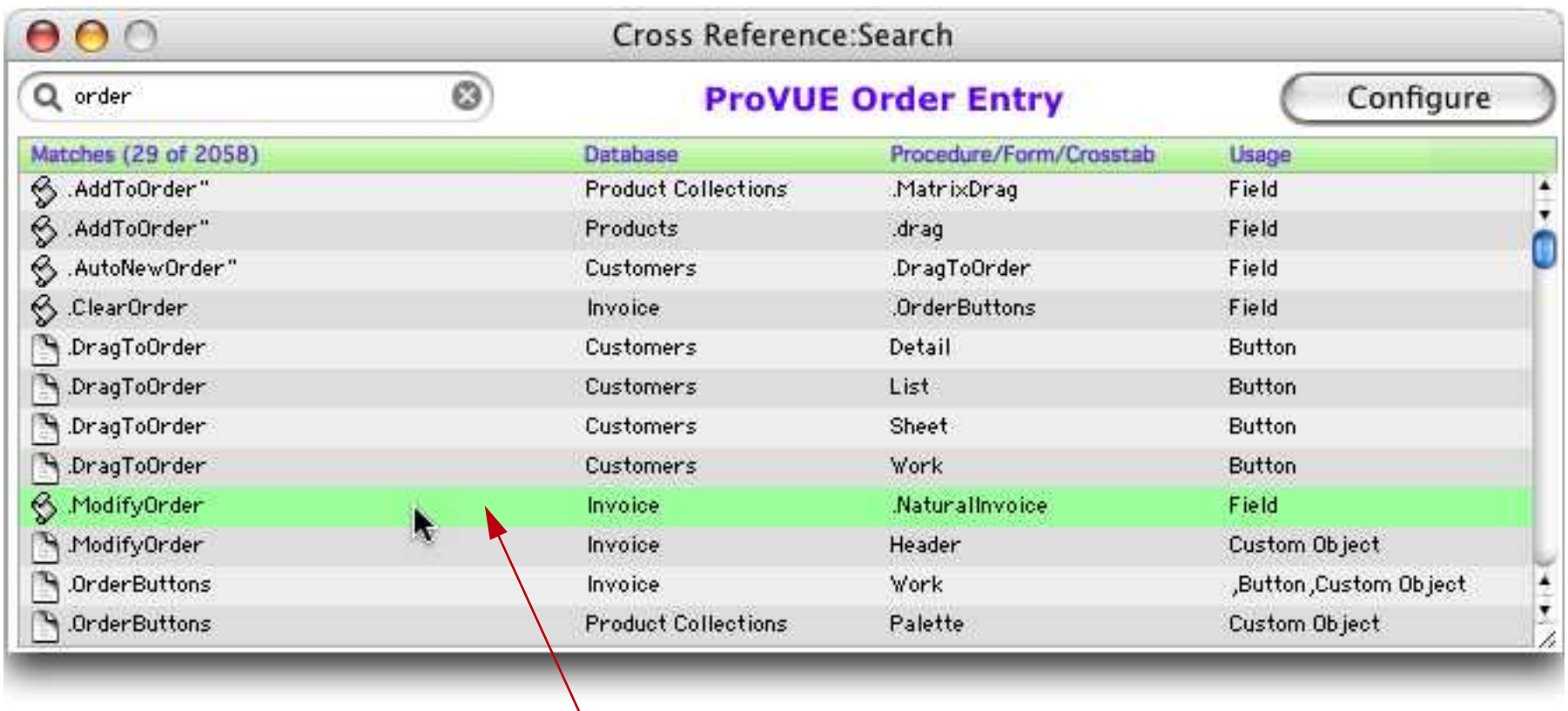
The Database column contains the name of the database that contains this item.

The Procedure / Form / Crosstab column contains the name of the procedure, form or crosstab that contains this item.

The Usage column tells how the matched item is used in this instance.

Opening a Form, Procedure or Crosstab

To open any of the items in the list, just double click on it.

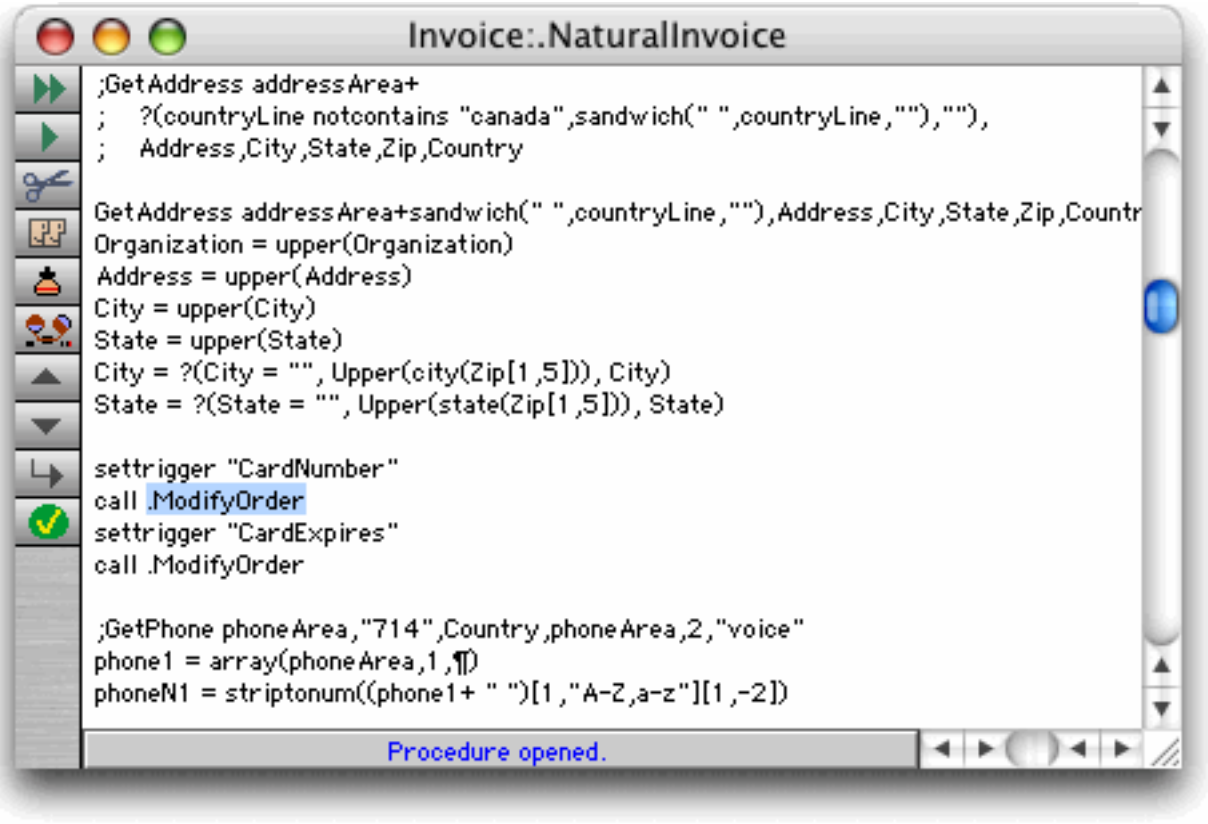


If the database is already open, the form, procedure or crosstab will open in a new window. If the database is not already open, the wizard will ask you if you want to open it.



If you press **Open Normal**, the database will open normally. Any saved window positions will open, and if the database has an .Initialize procedure it will be triggered (just as if you had double clicked on this database on your desktop, or opened it from the Open File dialog.) If you press **Open Secret**, the wizard will not open any saved windows, and it will not trigger the .Initialize procedure, if any. Only the requested form, procedure or crosstab will open.

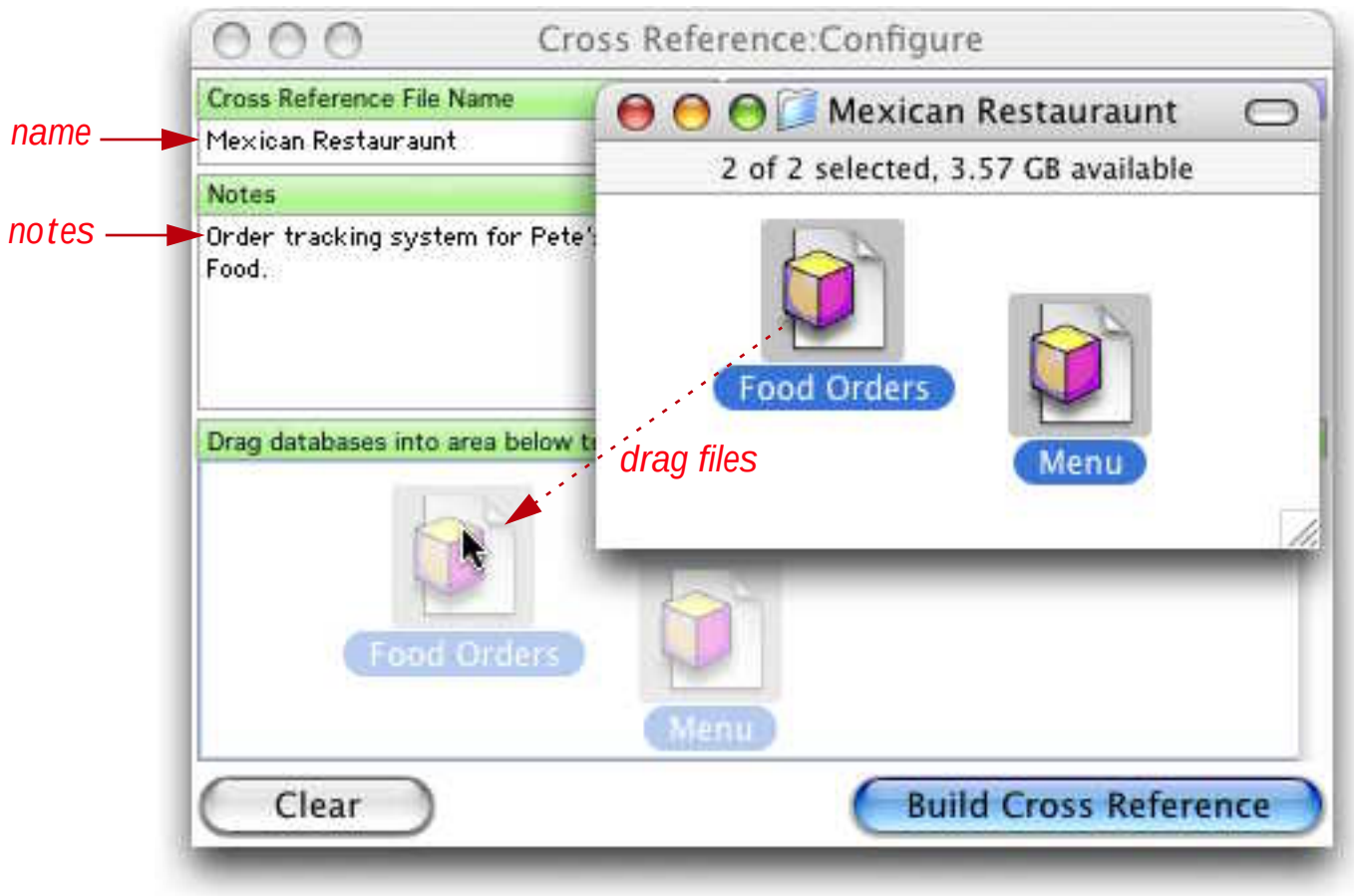
If you double clicked on a procedure, the wizard will automatically highlight the first occurrence of the text you originally searched for.



You can use the **Find Next** command to locate additional occurrences of this text.

Setting up a New Cross Reference

To set up a new cross reference, click on the **Configure** button. You'll see the current configuration in the Configuration window. Since you want to create a new configuration, press the **Clear** button, then type in the name for the new cross reference you are going to create (up to 27 characters). You can also optionally type in some notes. Finally, drag the files you want to be included in the cross reference into the bottom portion of the window.



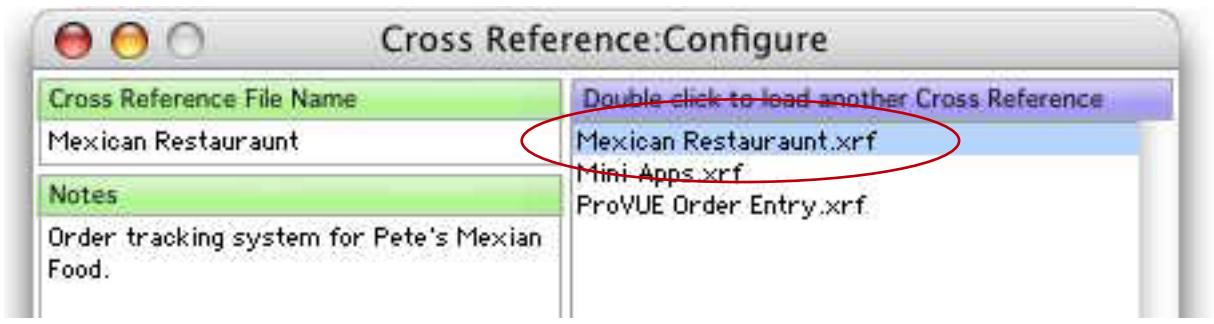
When you release the mouse the databases will be listed in the configuration window. If necessary you can drag additional files into this area.



When everything is set up, press the **Build Cross Reference** button. After a delay, the Configuration window will close and the new cross reference is ready to use. (The delay may be from a few seconds to a minute, depending on the complexity level of the databases involved). You can now search the cross reference as described in the previous section.



If you press the **Configure** button you'll see your new cross reference listed.



Updating a Cross Reference

A cross reference is a snapshot in time that reflects the contents of your database at the time you pressed the **Build Cross Reference** button. As you work on your databases the cross reference will gradually go out of date. When that happens you can update the cross reference by pressing the **Configure** button, then the **Build Cross Reference** button.

50 Ways to Trigger a Procedure

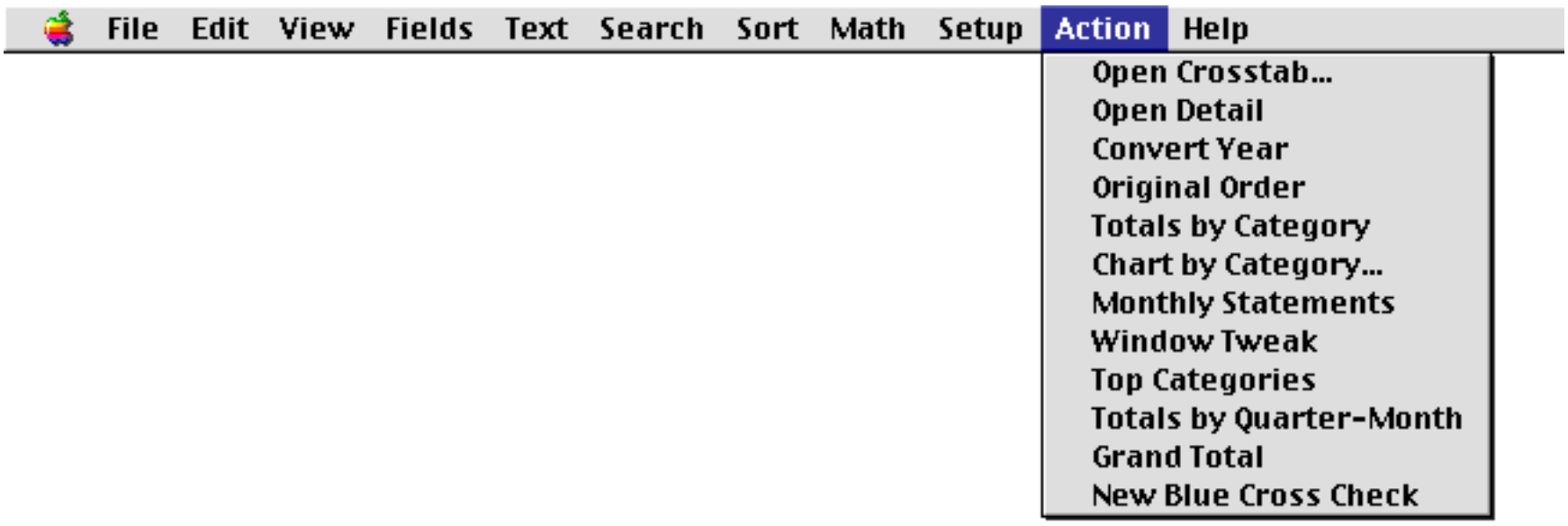
Procedures don't start up on their own — they must be triggered somehow. There aren't really 50 ways to trigger a procedure, but there are quite a few.

There are basically two types of triggers that can activate a procedure: **explicit triggers** and **hidden triggers** (implicit). Explicit triggers allow the user to deliberately trigger a procedure, for example by pressing a button or choosing an item from a menu. Hidden triggers activate a procedure automatically when the user performs some normal Panorama action. Examples of user actions that can cause a hidden trigger to activate include adding new records to a database, deleting records, opening a file, closing a window and many more. Procedures that are activated by hidden triggers can customize the way Panorama responds to these user actions, giving the programmer tremendous flexibility in creating a user interface that is appropriate for the task at hand.

The same procedure can be triggered different ways at different times. For example, the same procedure could be triggered both by a menu command or a button. If necessary, a procedure can use the `info("trigger")` function to find out how it was triggered.

The Action Menu

The **Action** menu is the simplest way to allow a procedure to be triggered. All you have to do is create the procedure, and it is automatically listed in the **Action** menu. The **Action** menu is added to the end of the standard menus, and the user can activate any procedure simply by selecting its name from the **Action** menu. (Advanced tip: If a procedure may be triggered other means in addition to the **Action** menu (for example, by a button), you can use the `info("trigger")` function to find out which way the procedure was triggered. If the procedure was triggered by the menu this function will return Action Menu.)



Panorama allows some variations from the basic one-size-fits-all **Action** menu. The programmer can give the **Action** menu a different name, or even split the **Action** menu into multiple menus. The programmer can also exempt some procedures so that they are not listed in the **Action** menu (a procedure that is not listed can only be triggered some other way, for example by a button).

The **Action** menu does have some significant limitations. **Action** menus can only be added to the standard menus, they cannot replace the standard menus. **Action** menus cannot have any submenus. The **Action** menu cannot change when the user switches from form to form—it always contains the same items (unless you switch to a different database). In addition, there must be a separate procedure for each menu item in the **Action** menu. It is not possible to have multiple menu items handled by a single procedure. With these restrictions in mind, the **Action** menu is by far the easiest way to set up your own menus in a Panorama database. For many custom Panorama databases, the **Action** menu is the only user interface. (Note: Prior to version 3.0 the **Action** menu was called the **Macro** menu.)

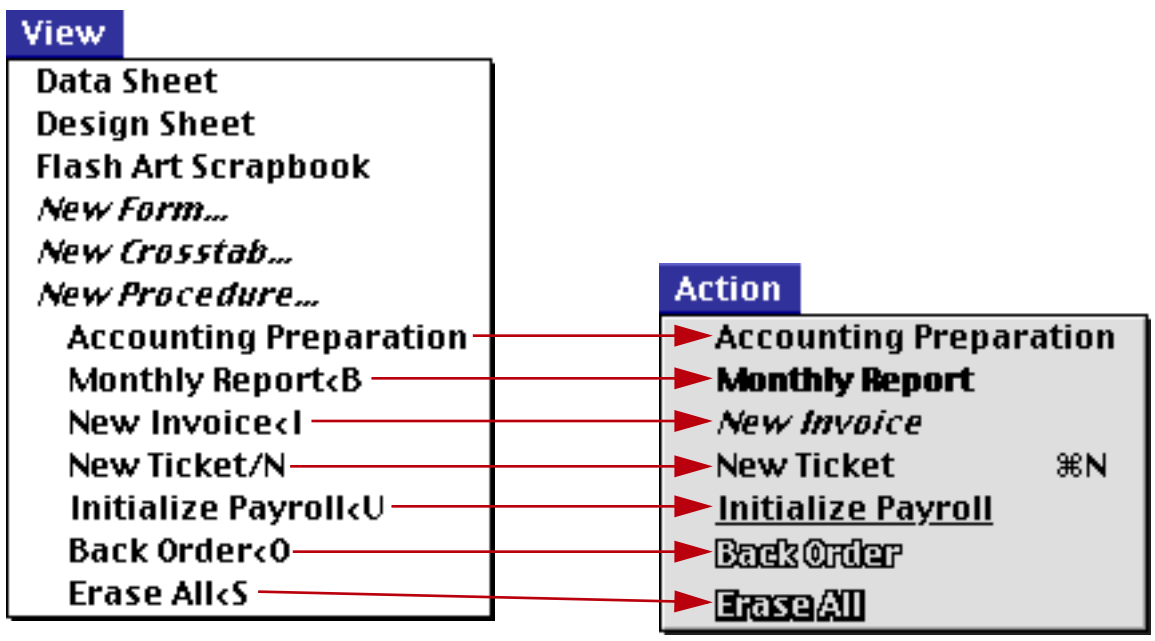
Action Menu Options

By adding special characters to a procedure's name, you can change the way the procedure is displayed in the **Action** menu, or even remove the procedure from the menu completely.

There are two special characters that should never be used in a procedure name that is listed in the **Action** menu: ^ and ; .

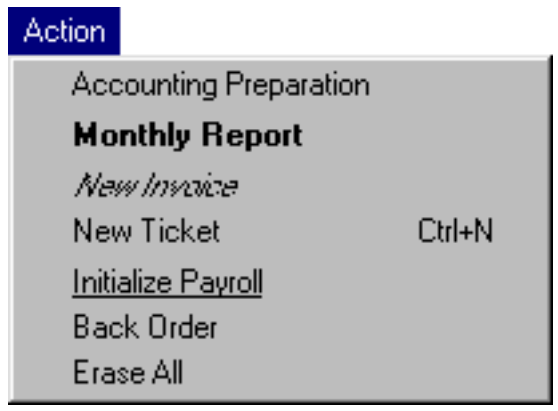
Setting Different Menu Item Styles (Bold, Italic, etc.)

You can make a procedure name appear in several different styles in the **Action** menu—bold, italic, underline, outline, shadow, or a combination of these styles. To change the style of a menu item you must add a special suffix to the end of the procedure name. The suffix consists of the < character followed by the letter B (bold), I (italic), U (underline), O (outline) or S (shadow). The action menu below show all six different styles (including plain) and the procedure names for creating those styles.



You can also combine styles with multiple suffixes, for example **Initialize Payroll<B<I** for both italic and bold. You can also combine a style with a command key equivalent, for example **Back Order<I/B**.

Here's the same menu on a Windows based system.



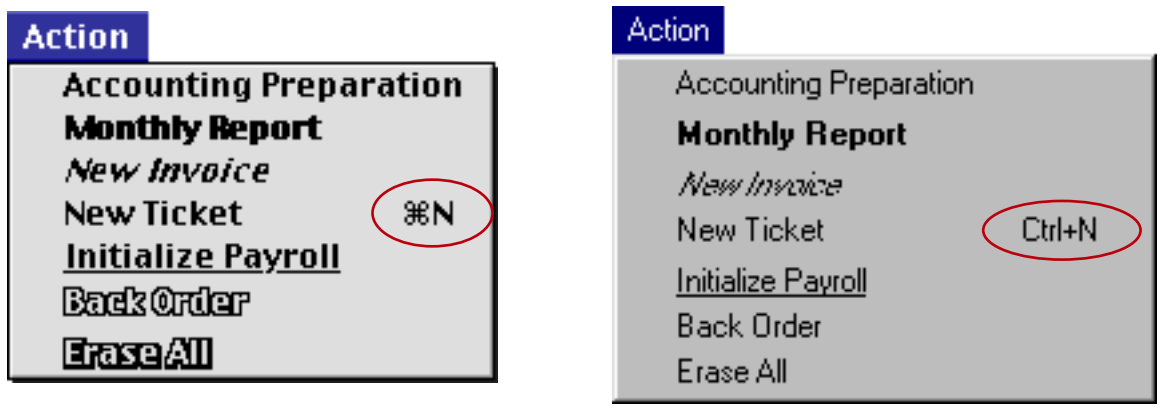
As you can see, the Outline and Shadow styles do not appear on Windows systems.

Shortcuts/Command Key Equivalents

Like other menu items, procedures in the **Action** menu can have keys on the keyboard assigned to them. On the Macintosh these are called **Command Key Equivalents**, on the PC (Windows) they are called **shortcuts**. To assign a key to a procedure you add a suffix consisting of a / character followed by the key you want to assign to the procedure. For example, a procedure named

```
New Ticket/N
```

will show up in the **Action** menu assigned to the N key. Here is what this menu looks like on both the Mac (left) and Windows (right).



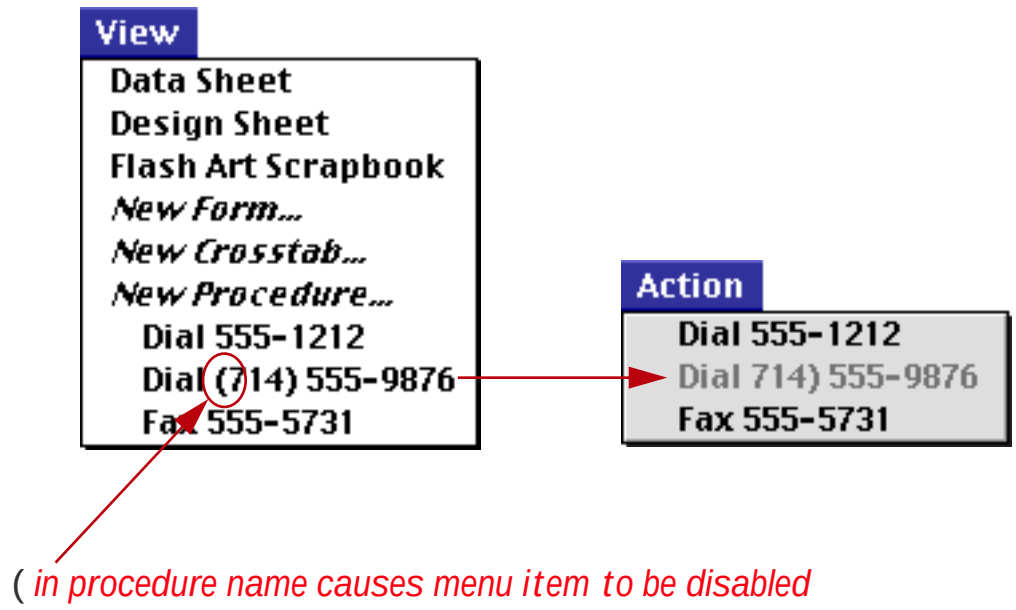
You can run this procedure by choosing it from the menu, or by pressing **Command-N** on the Macintosh or **Control-N** on a Windows based computer.

If a procedure's key assignment conflicts with one of Panorama's standard key assignments, the procedure will override the standard equivalent. For example, **Command/Control-P** is normally a command key equivalent for **Print**, but if you add a procedure called **Post Checks/P**, pressing **Command/Control-P** will trigger the procedure instead of printing.

Note: You cannot assign a command key equivalent to an "unlisted" procedure (one that begins with a period). Only procedures that appear in the **Action** menu can have command key equivalents.

Disabled Menu Items

If a procedure name contains the (character, the procedure name will appear in the menu but will be disabled (gray).

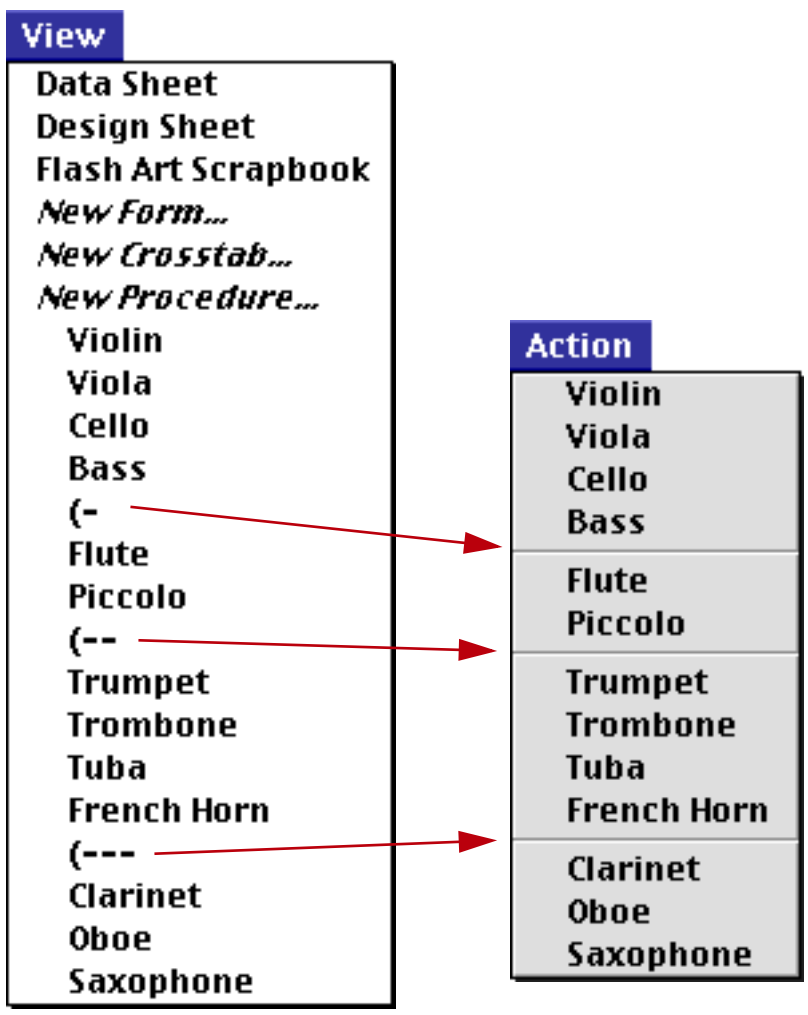


Don't use parenthesis in a procedure name unless you want the procedure to be disabled.

Separator Lines in a Menu

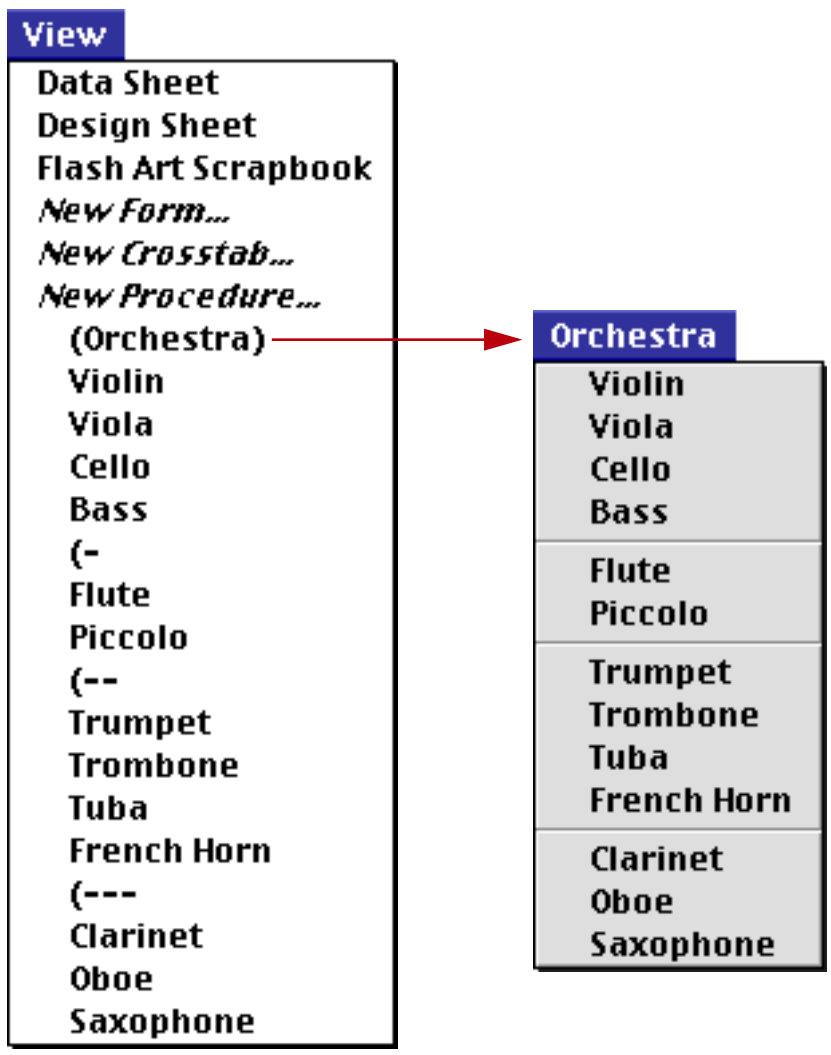
Many menus contain one or more gray lines separating different sections in the menu. To add a gray line to the **Action** menu, create a procedure with a name that start with (-. Use the **New Procedure** command in the View menu to create the procedure. Since the procedure will be disabled in the menu, it should not contain any statements.

To add a second gray line to the menu, create a procedure named (--. The third gray line should be named (---, the fourth (----, etc. (Each procedure name must contain a different number of dashes because Panorama does not allow duplicate procedure names.) Here's an example of an **Action** menu divided into four sections.



Renaming the Action Menu

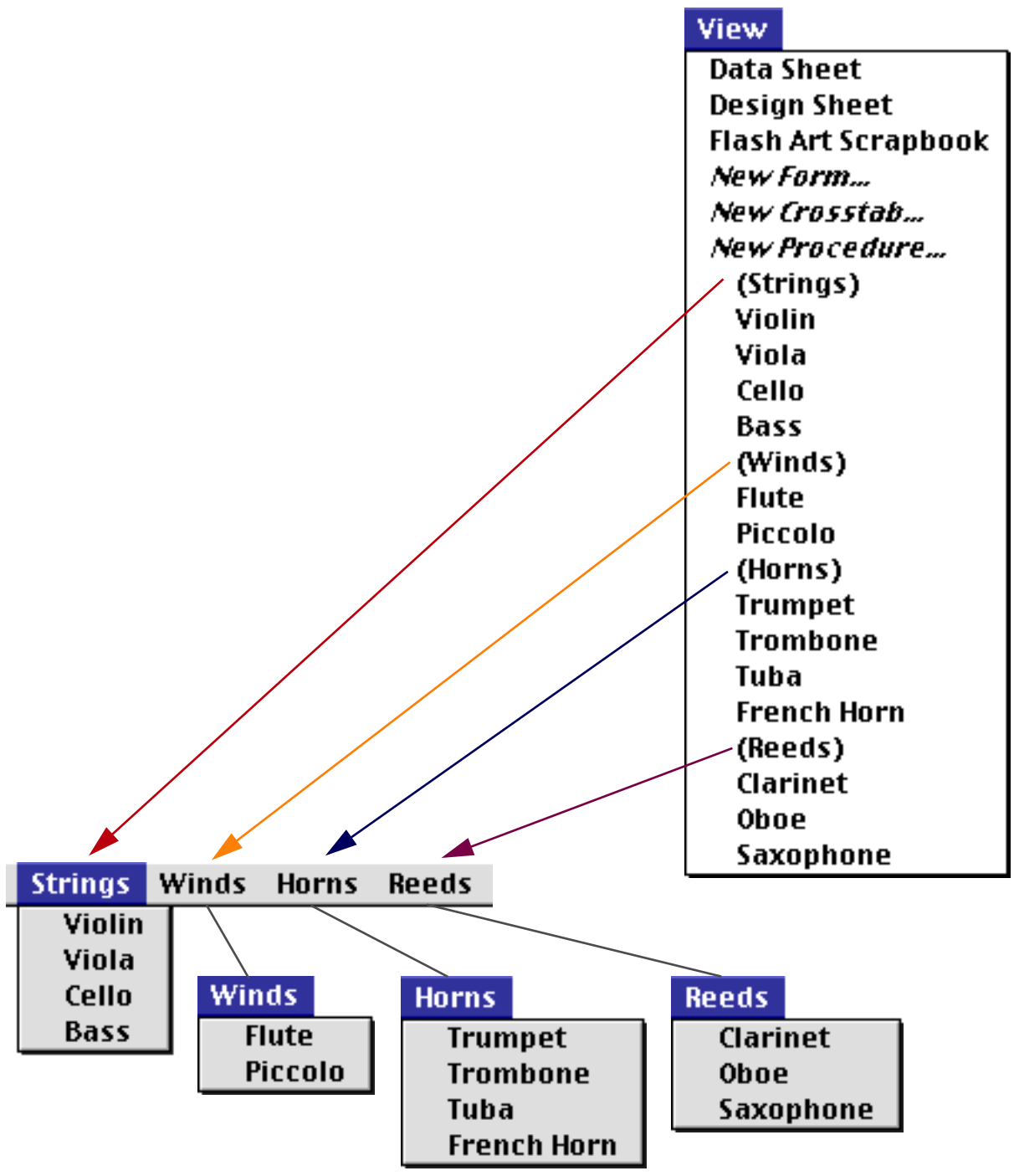
To give the **Action** menu a different name, insert an empty procedure with a name that begins and ends with a parenthesis as the very first procedure in the database. For example, inserting a procedure named `(Orchestra)` before the first procedure causes the **Action** menu to become the **Orchestra** menu.



See “Creating a New Form, Crosstab or Procedure” on page 236 to learn how to insert a procedure in any position.

Dividing the Action Menu into Multiple Menus

If your database contains lots of procedures you may want to split the **Action** menu into two or more separate menus. To split the **Action** menu into separate pieces, insert an empty procedure with a name that begins and ends with parentheses. For example, to start a new menu named People add a new procedure named (People). All of the procedures below this point will be listed in the People menu. You may split the **Action** menu into up to 12 separate menus. The example below shows an action menu split into four different menus.

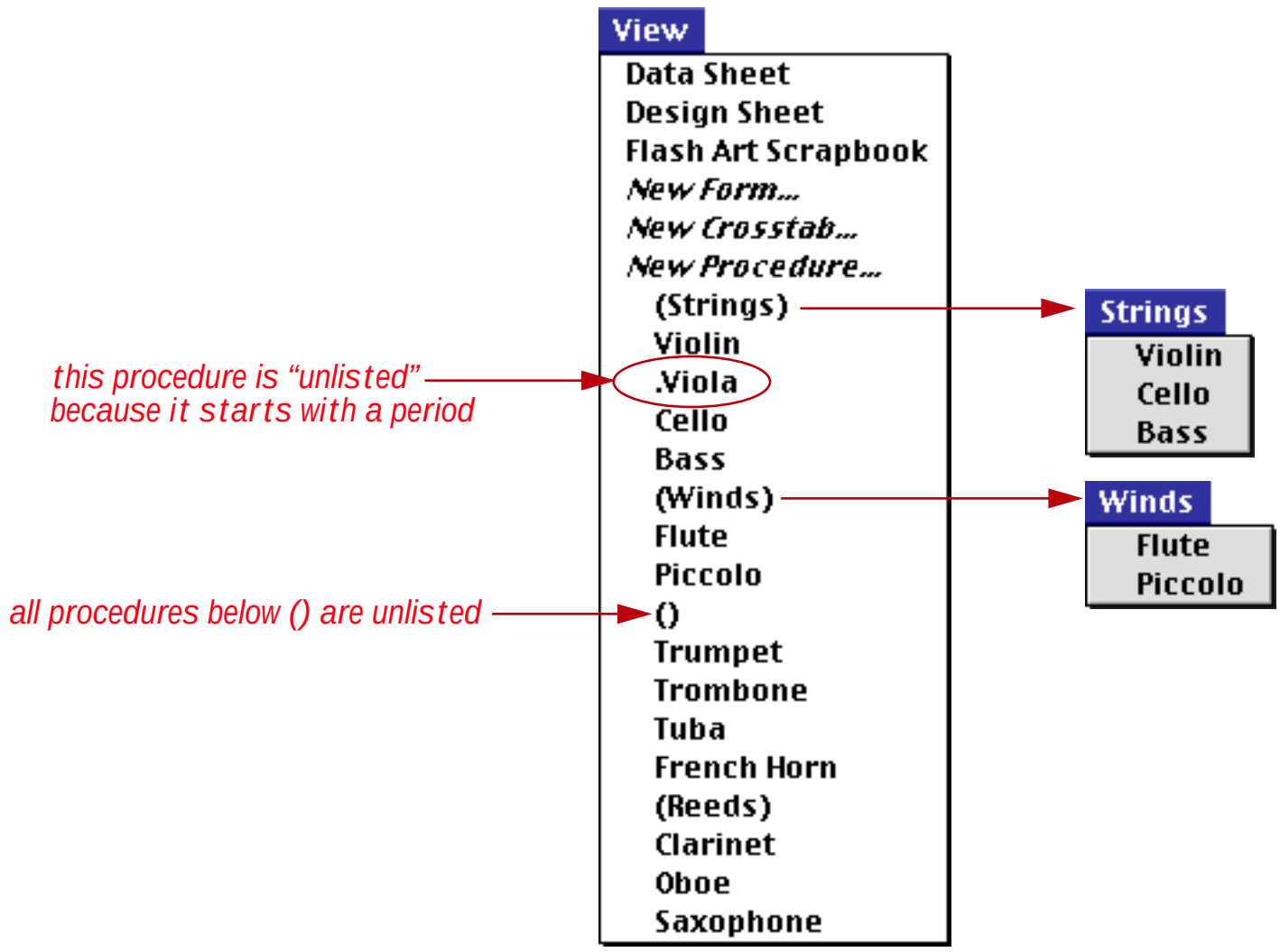


If necessary you can re-arrange the procedures to organize them into menus. See “Changing the Order of Forms, Crosstabs or Procedures” on page 237.

"Unlisted" Procedures

You may not want the **Action** menu to list the special procedures you create for buttons, automatic events, custom menus, or subroutines. To keep a single procedure out of the menu, add a period to the beginning of the procedure name. Any procedure name that begins with a period will be "unlisted," for example .Balance or .Prepare Chart.

To keep an entire group of procedures out of the menu insert a menu named (). Any procedure below a procedure named () will not appear in any menu.



Live Menus

The **Action** Menu is a very simple method for adding menus to your database. If you need more flexibility, however, you can use Panorama's "Live" menu feature. Live menus give you complete control over the content and appearance of each menu.

The secret behind the Live Menu system is a new special variable named `LiveMenuFormula`. If Panorama finds this variable it uses the contents to control the arrangement and content of the menus in the menu bar. This variable can be either a fileglobal variable (in which case it controls the menus for all windows in the current database) or a windowglobal variable (in which case it will control only the current window). In most cases, however, you will not manipulate this variable directly, but will access it with either the `FileMenuBar` or `WindowMenuBar` statement.

The FileMenuBar Statement

The `FileMenuBar` statement sets up the menu configuration for every window in the current database. Typically you might use this statement in the .Initialize procedure so that the menus will be set up as the database opens, but you can use it at any time to change the configuration. The statement has two parameters.

```
FileMenuBar StandardMenus,CustomMenus
```

The *StandardMenus* parameter is simply a list of the standard menus you want to include at the beginning of the menu bar. Choices available are `APPLE`, `FILE`, `EDIT`, `VIEW`, `TEXT`, `SEARCH`, `SORT`, `MATH` and `SETUP`. (Note: none of these choices are case sensitive.) You can also disable the action and wizards menu by using `-ACTION` and/or `-WIZARDS`. Items may be separated by spaces or commas. To include all standard menus, use ALL. To include the basic standard menus (Apple, File and Edit) use `BASIC`. (Note: On PC systems, the **Apple** menu is always excluded even if you ask it to be included.) If all you want to do is suppress some of the standard menus you can use this parameter and leave the *CustomMenus* parameter blank. For example, if you want just the **Apple**, **File**, **Edit**, **Action** and **Wizard** menus, use this statement.

```
FileMenuBar "Basic",""
```

If you want the **Apple**, **File**, **Edit**, **View** and **Action** menu, use this statement.

```
FileMenuBar "Apple File Edit View -Wizards",""
```

The *CustomMenus* parameter contains the formula for your live menus. There are a number of techniques for writing this formula, but the easiest is to use the `menu(` and `menuitems(` functions, like this:

```
FileMenuBar "Basic",
      menu("Colors")+menuitems("White;Yellow;Orange;Red;Green;Blue;Violet;Black")
```

As you have probably guessed, this creates a custom menu named **Colors** with eight items.



You can repeat to add as many menus as you like:

```
FileMenuBar "Basic",
      menu("Colors")+menuitems("White;Yellow;Orange;Red;Green;Blue;Violet;Black;")+
      menu("Texture")+menuitems("Flat;Matte;Smooth;Glossy;Ultra Glossy;")
```

This creates two menus — **Colors** and **Texture**:

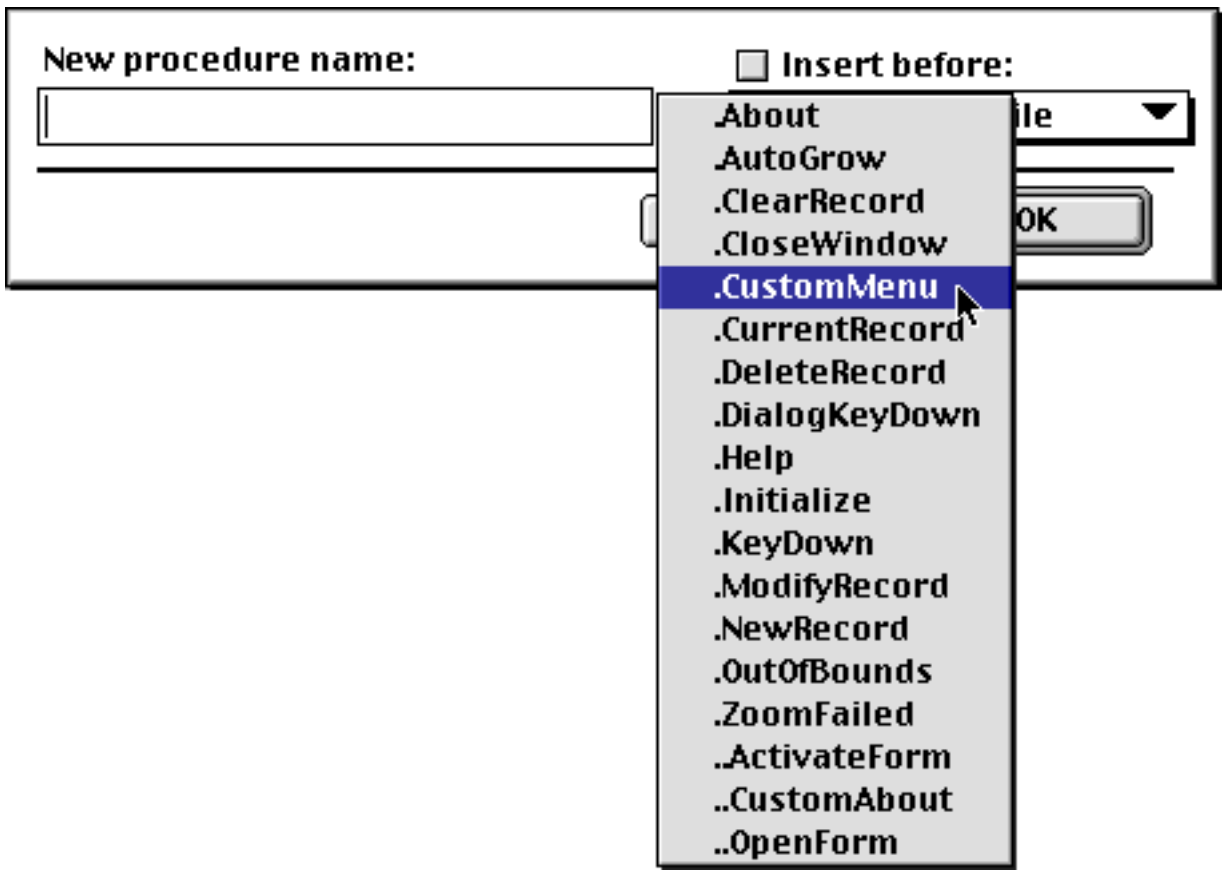


Remember that you must put these statements into a procedure and run the procedure before the live menus will appear. Until that point Panorama's standard menus will be used. To make sure that the live menus are used immediately when the database is opened you should put the `FileMenuBar` statement into the .Initialize procedure (see “.Initialize” on page 1511).

The .CustomMenu Procedure

What happens when a user pulls down a custom menu and selects a menu item? Choosing an item in a custom menu automatically starts a special procedure. This procedure must be called **.CustomMenu**. If the database does not have a **.CustomMenu** procedure then you'll still be able to pull down custom menus, but they won't do anything.

(Tip: When you create the **.CustomMenu** procedure, make sure that the procedure name is spelled and capitalized correctly. Don't forget the period at the beginning. The easy way to do it right is to select **.CustomMenu** from the pop-up menu in the **New Procedure** dialog (see “Creating Hidden Trigger Procedures” on page 1507).



If the **.CustomMenu** procedure name is not spelled correctly, Panorama won't be able to find and trigger the procedure when a custom menu item is used, and your custom menus won't work.)

Programming the .CustomMenu Procedure

Since the **.CustomMenu** procedure is triggered for all custom menu items, the procedure needs a way to figure out what item was chosen and act accordingly. For example, if the user pulls down a menu item you've created called **Sort by City**, you'll want something different to happen then if the user selects **Void Transaction**.

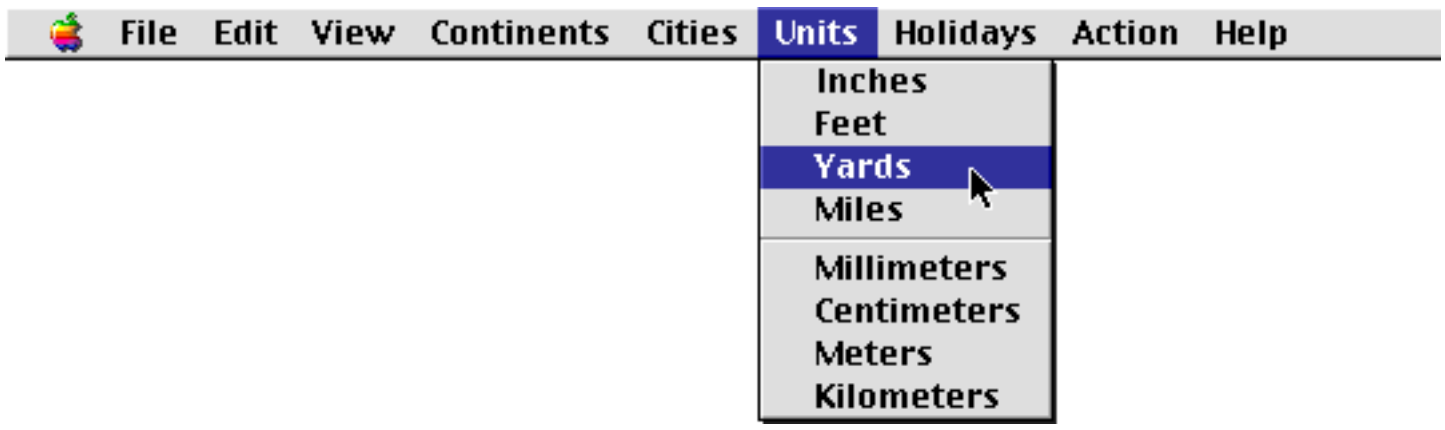
Whenever a custom menu item is chosen, Panorama stores the name of the custom menu and the name of the item. The programmer can retrieve this information using the `info("trigger")` function (see "INFO("TRIGGER")" on page 5433). By combining the `info("trigger")` function with `if` or `case` statements (see "IF Statements" on page 1380 and "CASE Statements" on page 1382) the programmer can create a **.CustomMenu** procedure that performs the correct action for every custom menu item. The following sections will illustrate several methods for programming **.CustomMenu** procedures to operate correctly.

The info("trigger") Function

The `info("trigger")` function can be used by any procedure to find out how that procedure was triggered. If the procedure was triggered by a custom menu, the `info("trigger")` function will return the word Menu followed by the menu name and menu item name, separated by periods.

```
Menu.<Menu Name>.<Menu Item Name>
```

For example, suppose you select **Yards** from the **Units** menu.



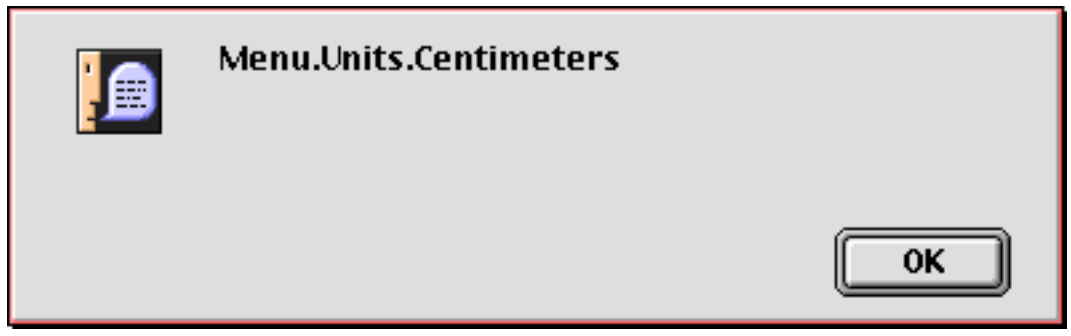
When this item is chosen the `info("trigger")` function will return the value:

```
Menu.Units.Yards
```

If you are ever in doubt about what value the `info("trigger")` function will contain for a menu item, temporarily insert the following line into the top of the **.CustomMenu** procedure.



Now choose the custom menu item in question. An alert will appear showing you the exact value produced by the `info("trigger")` function.



Once you have the value, be sure to go back and remove the temporary line from **.CustomMenu** procedure. A handy way to do this is to comment it out so that it can be easily re-activated later (see "“Commenting Out” Statements" on page 1424).

Processing Custom Menus with Simple IF's

The simplest way to process custom menus is to use the `if` statement (see “IF Statements” on page 1380). In this technique a similar block of statements is repeated over and over, once for each custom menu item. Each block starts with an if statement that uses `info("trigger")` to decode the menu item name. Then there are one or more statements that perform the actual operations for this menu item. Since we don't want any further actions for other menu items to be performed, this is followed by a `stop` statement (see “Stopping the Program” on page 1397). The `endif` statement terminates the entire block.

```
if info("trigger") = "Menu.<Menu Name>.<Item Name>"
   statement1
   statement2
   statement3
   ...
   stop
endif
```

The **.CustomMenu** procedure should contain one of these blocks for each custom menu item. Since each block of statements is completely self contained, the blocks can be in any order you want. The example below shows a **.CustomMenu** procedure written for two custom menus with two menu items apiece.

```
if info("trigger") = "Menu.Organize.SortByName"
   field LastName
   sortup
   field FirstName
   sortupwithin
   stop
endif
if info("trigger") = "Menu.Organize.SortByZip"
   field Zip
   sortup
   stop
endif
if info("trigger") = "Menu.Transaction.Add"
   AddRecord«
   stop
endif
if info("trigger") = "Menu.Transaction.Void"
   Description="Void"
   Amount=0
   stop
endif
```

Processing Custom Menus with Nested IF's

If your database has a lot of custom menu items, the technique described in the last section can be slow for items that are processed toward the bottom of the **.CustomMenu** procedure. There may be a noticeable delay as Panorama processes all the `if` statements. The solution to this delay is to group the blocks together by menus using nested `if` statements. For example, suppose your database uses 6 custom menus with 15 items apiece. Using the simple `if` statement technique there could be a delay of as many as 90 `if` statements before the statements that actually do the work get started. Using nested `if` statements this delay is reduced to a maximum of 21 `if` statements.

The example below shows the previous example rewritten to use nested `if` statements. The outer level of `if` statements selects what menu is being processed, while the inner level selects the individual menu items.

```
if info("trigger") beginswith "Menu.Organize."
   if info("trigger") endswith ".SortByName"
      field LastName sortup
      field FirstName sortupwithin
      stop
   endif
   if info("trigger") endswith ".SortByZip"
      field Zip sortup
      stop
   endif
endif
if info("trigger") beginswith "Menu.Transaction."
   if info("trigger") endswith ".Add"
      AddRecord
      stop
   endif
   if info("trigger") endswith ".Void"
      Description="Void"
      Amount=0
      stop
   endif
endif
```

Splitting the Trigger into Menu/Item Names

In some cases it may be advantageous to split the value returned from `info("trigger")` back into separate menu and menu item names. This can be done with the `array(` function as shown in the example below (see "Text Arrays" on page 1236).

This example assumes that the database contains a field called Carrier, and a custom menu called **Airlines** that contains menu items listing airlines: **American**, **Delta**, **Southwest**, etc. When the user selects an airline the name of the airline is copied into the **Carrier** field.

```
local MenuName,MenuItemName
MenuName=array(info("trigger"),2,".")
MenuItemName=array(info("trigger"),3,".")

if MenuName="Airlines"
   Carrier=MenuItemName
   stop
endif
/* other menu processing continues below */
…
```

An even easier method for doing this is to use the `splitmenutrigger` statement. See the **Programming Reference** wizard for more information on this statement.

Menus with Modifier Keys

Sometimes you may want to have a custom menu item perform a different action if a modifier key is pressed (**Shift**, **Control**, **Option**, **Command** or **Alt**). The program can test for these modifiers with the `info("modifiers")` function. This function returns the names of all the modifier keys that are pressed down.

The partial example below uses the `info("modifiers")` function to create a shortcut for the **Void** menu item. This procedure is programmed so that a confirmation alert normally appears before the transaction is voided, but if the user holds down the **Option** key the alert is skipped (**Alt** key on PC systems).

```
if info("trigger") = "Menu.Transaction.Void"
   if (not info("modifiers") contains "option")
      alert 1014,"Are you sure you want to void this transaction"
      if info("dialogtrigger")="No"
         stop
      endif
   endif
   Description="Void"
   Amount=0
   stop
endif
```

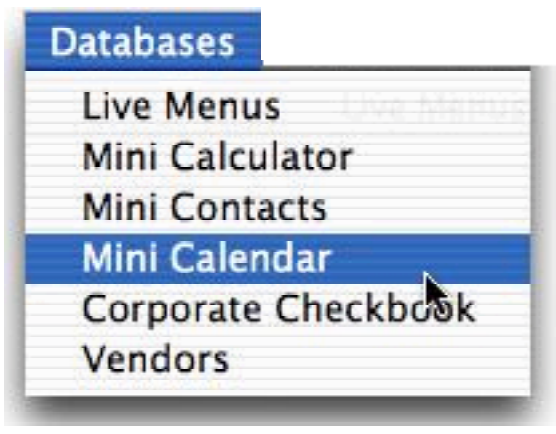
Since the user has no way to tell that a modifier key affects the operation of a menu item, this technique should be used with care. Don't make pressing a modifier key cause a completely different operation. In general this technique should only be used for slight variations (like the shortcut above), or to allow for secret undocumented operations that you don't want someone to stumble across accidentally.

Building Menus from Arrays

You'll often want to build a menu (or part of a menu) from an array. The `arraymenu(` function makes this easy to do. This function turns a carriage return delimited array into a series of menu items. This example builds a menu that lists all of the currently open databases.

```
FileMenuBar "Basic",menu("Databases")+arraymenu(info("files"))
```

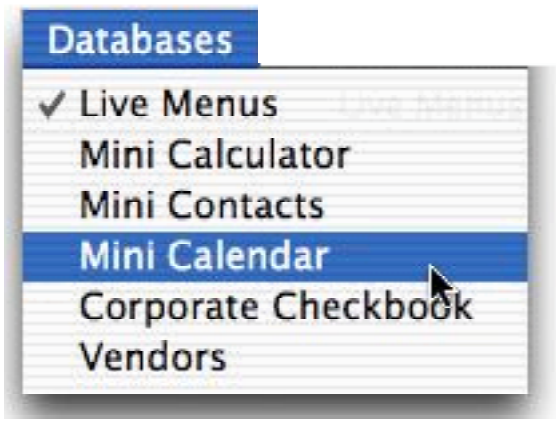
This menu will automatically adjust as databases are opened and closed.



The `checkedarraymenu(` function is similar, but allows one of the menu items to be checked. The second parameter specifies what item should be checked.

```
FileMenuBar "Basic",menu("Databases")+checkedarraymenu(info("files"),info("databasename"))
```

In this case, the currently opened database is checked.



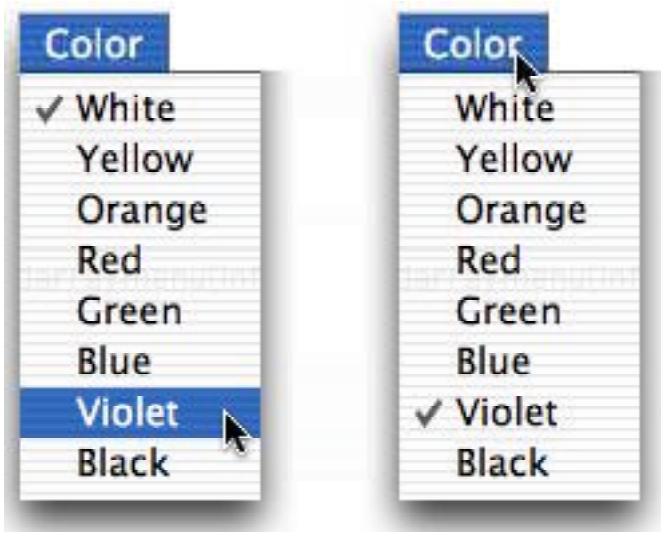
A more useful example uses a menu to set a variable, in this case allowing a color to be selected from a list of colors. To create the menu place the following statements in the .Initialize procedure:

```
FileGlobal colorOptions,colorChoice
colorOptions=replace("White;Yellow;Orange;Red;Green;Blue;Violet;Black",";",¶)
define colorChoice,"White"
FileMenuBar "Basic",menu("Color")+checkedarraymenu(colorOptions,colorChoice)
```

Here's the code that must be placed in the .CustomMenu procedure to actually operate the menu (see "The .CustomMenu Procedure" on page 1466).

```
if info("trigger") beginswith "Menu.Color."
     colorChoice=array(info("trigger"),3,".")
endif
```

That's all there is too it. Now when an item is selected from the **Color** menu, the item will be checked.



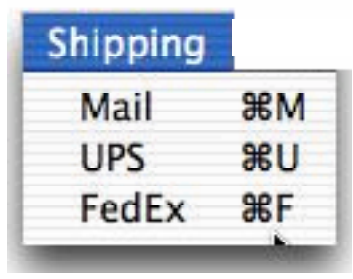
The value of the choice is in the colorChoice variable, where it can be used by other formulas or procedures.

Command Key Equivalents

Within the `menuitems(` function an item can be followed by the ´ character an one or more options. Everything before the ´ is the menu name, with the options after. The `/` option is used to set up command key equivalents. This example creates three menu items with corresponding equivalents.

```
FileMenuBar "Basic",
      menu("Shipping")+menuitems("Mail¬/M;UPS¬/U;FedEx¬/F")
```

To select **Mail** with the keyboard, type **Command-M** (**Control-M** on the PC). **UPS** is **Command-U** / **Control-U** while **FedEx** is **Command-F** / **Control-F**.



Menu Styles

Using the < option, menus can be bold, italic, underline, or a combination of these three choices.

```
FileMenuBar "Basic",
      menu("Currency")+menuitems("Dollar¬<B<U/D;Euro;Franc¬<I;Lira¬<I;Peso;Pound¬<U;Yen")
```

Here is the menu created by this statement.



Disabled Menu Items and Separator Lines

To disable a menu item make the first character in the menu name a left parenthesis (.

```
FileMenuBar "Basic",
      menu("Currency")+menuitems("Dollar;Euro;(Franc;(Lira;Peso;Pound;Yen")
```

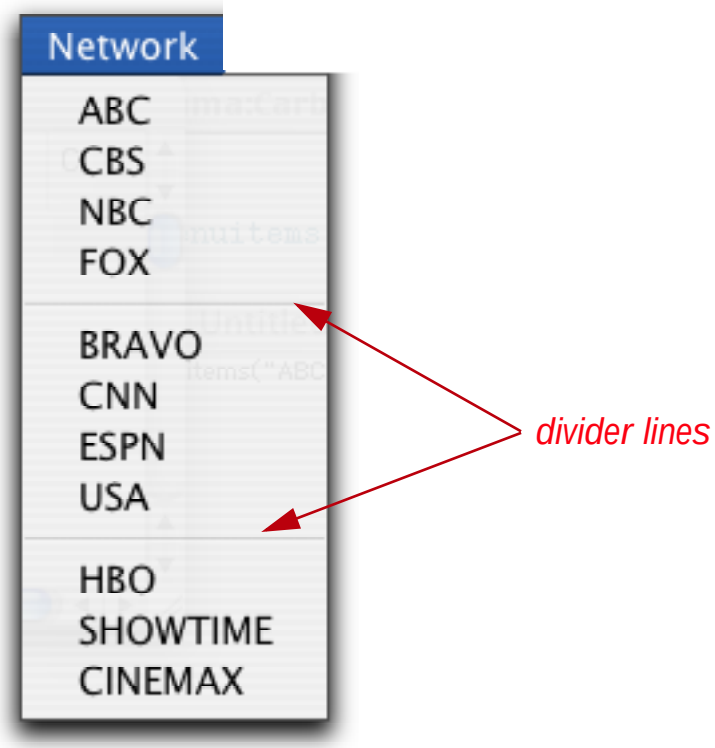
Here is the menu with the disabled items.



To create a divider line the menu item should be (-, like this:

```
FileMenuBar "Basic",
    menu("Network")+
    menuitems("ABC;CBS;NBC;FOX;(-;BRAVO;CNN;ESPN;USA;(-;HBO;SHOWTIME;CINEMAX")
```

This menu is divided into three sections: Broadcast networks, basic cable, and premium cable:

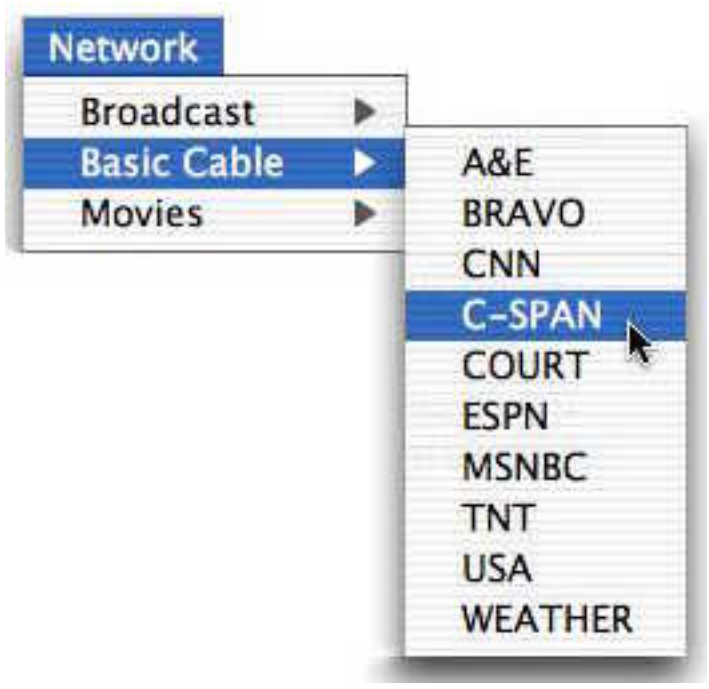


Submenus (Heirarchical Menus)

One of the advantages of the Live Menu system is that it makes it easy to attach a submenu to any menu item (even an item in a submenu for multiple menu levels of submenus. If you use submenus they must be defined before they are used. Submenus are defined with the `submenu(` function instead of the `menu(` function. To use a submenu you must place `(submenu name)` in the options area at the end of the menu item (after the ´ character).

```
FileMenuBar "Basic",
  submenu("Broadcast")+menuitems("ABC;CBS;NBC;FOX;WB;")+
  submenu("Basic")+menuitems("A&E;BRAVO;CNN;C-SPAN;COURT;ESPN;MSNBC;TNT;USA;WEATHER;")+
  submenu("Premium")+menuitems("HBO;SHOWTIME;CINEMAX;")+
  menu("Network")+menuitems("Broadcast¬(Broadcast);Basic Cable¬(Basic);Movies¬(Premium)")
```

This example creates a menu with three submenus.



Multiple Column Menus

On MacOS, a menu that has too many items to fit within the height of the display will scroll. By using the `columnmenu(` function you can create a menu that wraps to two , three, four or more columns as needed. Otherwise this function is the same as the `menu(` function.

```
FileMenuBar "Basic",columnmenu("Titles")+arraymenu(listchoices("Title",¶))
```

Multiple column menus are useful when your menus contain a lot of items.



On Windows computers menus always wrap to multiple columns, so this function is the same as the menu(function.

The WindowMenuBar Statement

The WindowMenuBar statement works exactly like the FileMenuBar statement except for the fact that it only specifies the menus for the current window instead of for all windows in the current database. Typically you would use this statement immediately after opening the window.

```
OpenForm "Currency Conversions"
WindowMenuBar "Basic",
   menu("Currency")+menuitems("Dollar;Euro;(Franc;(Lira;Peso;Pound;Yen")
```

You can combine the WindowMenuBar and FileMenuBar statements. In this case the WindowMenuBar overrides for the windows in which it is used.

Advanced Topic: Live Menus Behind the Scenes

The following sections describes the nitty gritty details of how Live Menus work. For most applications you won't need to know this information and you can simply skip this section.

Live menus are set up by creating a fileglobal or windowglobal variable named LiveMenuFormula. This variable needs to be filled with a formula that calculates the menus to display. An important distinction is that the variable must contain with the formula itself -- not the result of the formula. The FileMenuBar and WindowMenuBar statements automatically create this variable for you and fill it with the formula.

When switching to a new window or clicking on the menu bar Panorama will look for this variable. If the variable exists, Panorama will calculate the formula result, and use that result to display all of the menus. This means that all of the menu contents are calculated on the fly as you click on the menu bar!

The formula in the LiveMenuFormula variable must calculate a carriage return delimited array. Panorama will scan this array from top to bottom. A line that begins with (and ends with) signals the start of a new menu. All other lines represent individual menu items.

Menu Titles

Menu titles must begin and end with (and). Here are some examples of menu titles.

```
(File)
(Message)
(Windows)
```

A menu title can have one or more options. Options appear after the end of the menu title. A tab character must appear before the first option, immediately after the). The options available are:

***S** - This is a submenu. It will not appear in the menu bar, but may be attached to other menus (see below).

***E** - This is the edit menu. When a text or word processing object is clicked on, this menu will be replaced by the correct editing menu for that object.

***M** - This is a multiple column menu. Instead of scrolling if there are too many items to fit on the screen, the menu will wrap to two or more columns.

Here are some examples of menu titles with options:

```
(Fields)¬*S*M
(Recent)¬*S
```

Note: In these examples, the ´ character represents the tab character. The actual formula to create the titles above would be:

```
"(Fields)"+¬+"*S*M"
"(Recent)"+¬+"*S"
```

To simplify the examples we'll use ´ to represent the tab character in the examples in the following sections.

If a menu title is entirely numeric, Panorama will insert the corresponding standard menu into the menu bar at that location. For example, `(1)` is the **Apple** menu, while `(27)` is the standard **File** menu for use with a form. Note: If the user selects an item in a standard menu, the .CustomMenu procedure is not triggered. Instead, Panorama simply performs the normal action for that menu item. Note 2: A standard menu should not have any menu items (see below) following it. The standard menu should be immediately followed by another menu title.

Note: Panorama normally includes the **Action** and **Wizard** menus in your menu bar automatically. If you want to remove either of these menus you can by using the special menu titles `(-ACTION)` and `(-WIZARDS)`.

Menu Items

Any line that does not begin with a (and end with) is a menu item. Most menus have one or more items. The items should be immediately below the corresponding menu title. Here is an example of a simple menu:

```
(Shipping)
US Mail
UPS Ground
Fedex
DHL
```

If the first character of a menu item is √ or • the menu will be marked:

```
(Shipping)
US Mail
√UPS Ground
Fedex
DHL
```

Note: In this example, the name of the menu item is still "UPS Ground", not "√UPS Ground". This is the value that will appear in `info("trigger")` if this menu item is selected.

If the first character of a menu item is (the menu will be disabled (dimmed). In this example both Fedex and DHL are not available:

```
(Shipping)
US Mail
√UPS Ground
(Fedex
(DHL
```

If the first character of a menu item is a tab then special characters in the name are put into the menu name. If the LiveMenu formula results in the text shown below, the **(Alpha** menu item will not be dimmed, and the (character will appear in the menu. The **(Beta)** menu item is NOT a new menu title, and the (and) will appear in the menu. (Remember that in these examples ¬ represents the tab character.

```
¬(Alpha
¬(Beta)
```

You can combine the ^, ¥, (and ´ characters. In this example the Gamma menu is dimmed, checked, and contains the (and) characters.

```
¯(¬(Gamma)
```

By adding ´ at the end of the menu item you can specify options

```
MENU ITEM¬/F - Command-F
MENU ITEM¬/N - Command-N
MENU ITEM¬<B - Bold
MENU ITEM¬<I - Italic
MENU ITEM¬<U - Underline
```

All of these options can be combined in ways that make sense. The Omega menu item below is checked with a bullet (¥), is bold, italic and has an key equivalent of Command-F.

```
•¬(Omega)¬/F<B<I
```

Submenus

A submenu must be defined before it is used. Once it is defined it can be attached to any menu item like this:

```
MENU ITEM¬(SUBMENU)
```

Here is an example that shows how to create a Travel menu with three submenus, Air, Car, and Train

```
(Airlines)¬*S
American
Continental
Delta
Jet Blue
Southwest
United
(Train)¬*S
Amtrak
VIA
(Car Rental)¬*S
Avis
Hertz
National
(Travel)
Air¬(Airlines)
Car¬(Car Rental)
Train¬(Train)
```

Formula Errors

What if the formula in the LiveMenuFormula variable contains an error? In that case, Panorama simply displays the standard menus. There is no error message or indication of where the problem occurred. So if your custom menu doesn't appear you need to start looking very closely at the formula you have set up. (Note: Another advantage of the `WINDOWMENUBAR` and `FILEMENUBAR` statements is that they automatically check for and display formula syntax errors when you write your procedure. However, it is still possible to have a formula evaluation error (for example "number when text was expected") even when using these statements. Often the best approach is to simply remove parts of the formula until it works, then fix the portion that was most recently removed.)

The menu(, menuitems(, arraymenu(and checkarraymenu(functions

These functions were introduced earlier. Now you can easily see what these function actually do. The `menu(` function simply adds ( ) and a carriage return to the menu name. The `menuitems(` function simply converts a semicolon separated array into a carriage return separated array. It also converts ¬ characters into tab characters. The `arraymenu(` functions add a tab to the beginning of each array element so that (,), *, < and / characters in array values will be displayed properly in the menu.

Helper Functions for Standard Menus

The following functions help incorporate standard Panorama menus into your live menu bar. By using these functions you eliminate the need to look up the correct menu numbers. In addition, the functions automatically adjust the menu numbers depending on whether or not the current window is a form or data sheet.

`STANDARDFILEMENU()` - Generates the specification for the standard File menu (and the Window submenu). The specification will be adjusted depending on whether the current window is a form or a data sheet.

`STANDARDEDITMENU()` - Generates the specification for the standard Edit menu. The specification will be adjusted depending on whether the current window is a form or a data sheet.

`STANDARDVIEWMENU()` - Generates the specification for the standard View menu.

`STANDARDFIELDSMENU()` - If the current window is a data sheet, this function generates the specification for the standard Fields menu.

`STANDARDTEXTMENU()` - If the current window is a data sheet, this function generates the specification for the standard Text menu (as well as the Font and Size menus).

`STANDARDSEARCHMENU()` - Generates the specification for the standard Search menu.

`STANDARDSORTMENU()` - Generates the specification for the standard Sort menu.

`STANDARDMATHMENU()` - Generates the specification for the standard Math menu.

`STANDARDSETUPMENU()` - Generates the specification for the standard Edit menu. The specification will be adjusted depending on whether the current window is a form or a data sheet.

`NOACTIONMENU()` - When included in the formula the Action menu is removed from the menu bar.

`NOWIZARDMENU()` - When included in the formula the Wizards menu is removed from the menu bar.

Custom Menus (Resource Based)

Resource based Custom menus are a third option for setting up menus. Like Live Menus, which were introduced in Panorama V, resource based Custom Menus give you tremendous flexibility and complete control over your menus. However, resource based custom menus are much more difficult to create and maintain, and they require that you set up a separate file to hold the menu resources. This separate file must be kept with the database at all times. Because of these drawbacks, we strongly recommend that you not use resource based menus for new projects. If you are building a new project we suggest that you skip this section and use Live Menus instead (see “Live Menus” on page 1465). In fact, the advantages of Live Menus are so great that we recommend that, if possible, you change your existing databases from resource menus to Live Menus, especially if you do major maintenance on a database.

Important Note for Custom Menu Developers

Starting with Panorama V, resource based custom menus need to avoid the area around menu #200. Live menus use numbers from 200 up. Wizard menus use menus from 199 down. So for now that means custom menus should avoid the area between 190 and 210. For future expansion we recommend avoiding the area between 170 and 230. Or better yet, switch your custom menus to Live menus!

Custom Menu Overview

Unlike most other Panorama features, custom menus are not self-contained in a Panorama database. Instead you must create and set up a separate file, called a **resource file**, that contains the custom menus you want to use. Resource files are special files that were developed on the Macintosh uses for storing information the system needs to operate. Panorama also supports resource files on Windows based systems with the `.rsr` extension (for example My Menus.rsr). Within the resource file each menu is assigned a unique number between 128 and 20,000.

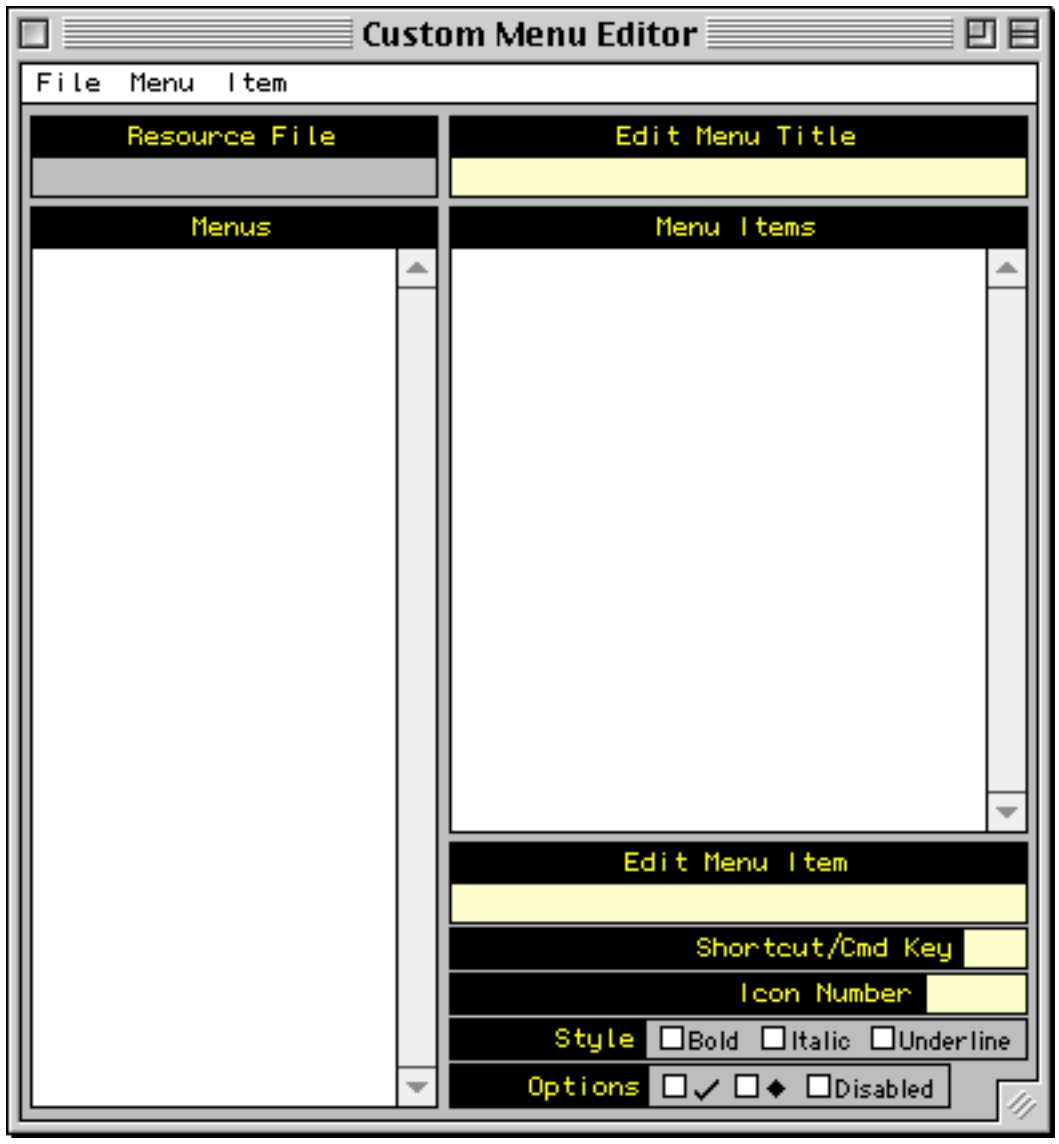
Once the resource file is created, Panorama can open it and use the menus inside. You'll need to assign the custom menus separately for each form and for the data sheet (if you don't make this assignment, Panorama will use the standard menus.) Once they are assigned, the appropriate custom menus appear in the menu bar automatically (in data mode only—standard menus always appear in graphics mode).

Once the menus are set up, the user can select items from a custom menu. When this happens Panorama automatically triggers a special procedure for handling custom menus. This procedure must be called the **.CustomMenu** procedure. This one procedure handles all the custom menus in the entire database—whether there is one or fifty. The procedure can use the `info("trigger")` function to find out what custom menu item was selected, then take the appropriate action.

For advanced applications, it is even possible for a Panorama procedure to modify the arrangement and appearance of the custom menus. Using special statements, a procedure can change which menus are assigned to a form, can enable or disable entire menus or individual items, can rename menu items, and can set and remove checkmarks and other symbols.

Preparing a Resource File

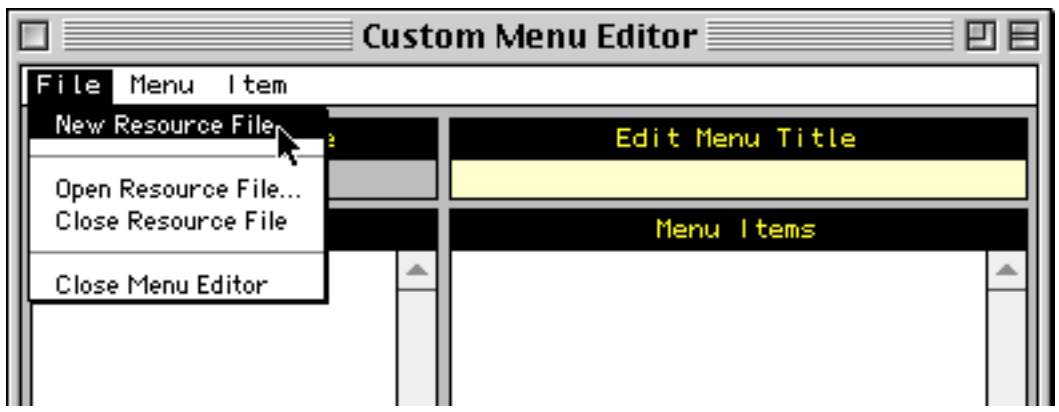
To create custom menus you'll need to create menu resources in a resource file. One way to do this is with Panorama's **Custom Menu Editor**, a database that is installed along with Panorama. To open the **Custom Menu Editor** simply locate it on the hard disk and open it just like any other database.



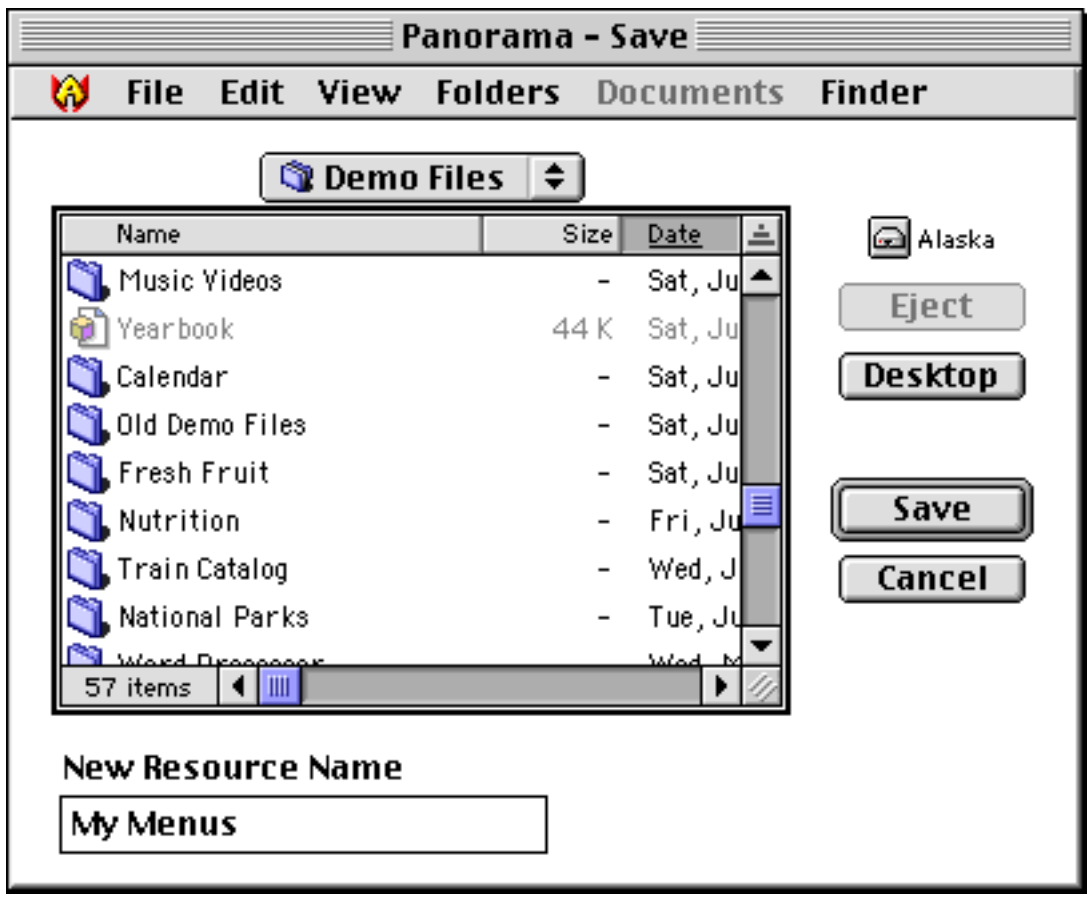
(Note: If you are using a Macintosh computer you can also create and edit menu resources using resource editing programs such as ResEdit or Resourcerer.)

Creating a New Resource File

To create a new resource file choose **New Resource File** from the File menu inside the editor window.

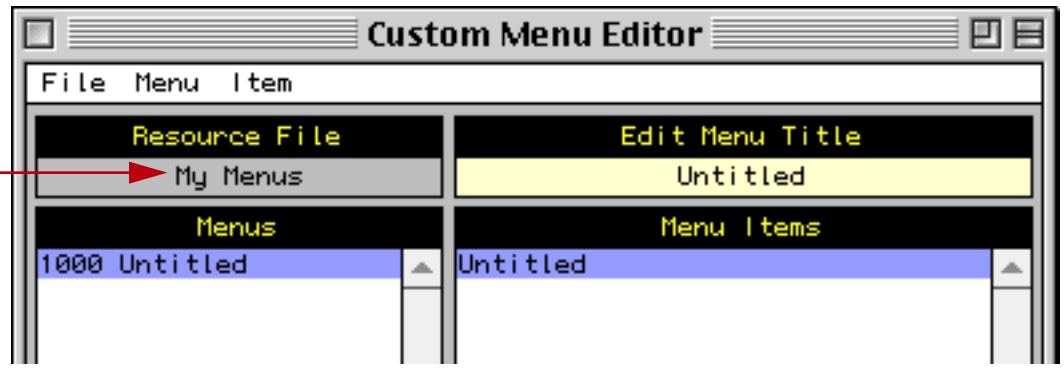


Use the dialog to enter the name of the new resource file and the folder (subdirectory) in which it will be placed.

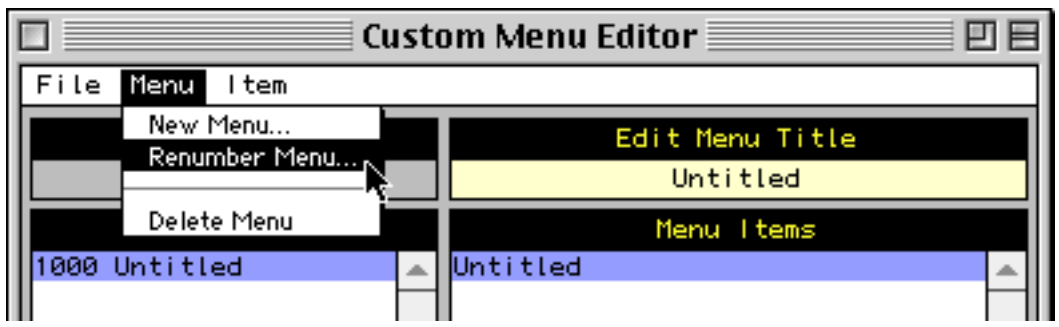


When you press the **Save** button the new resource file is created.

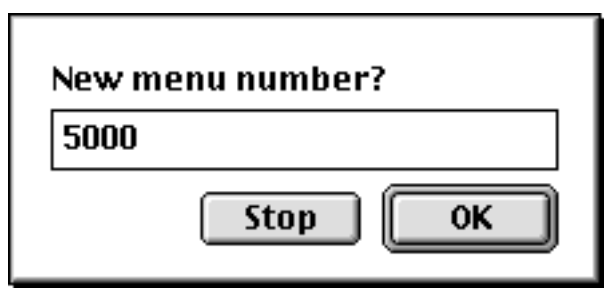
name of open resource file



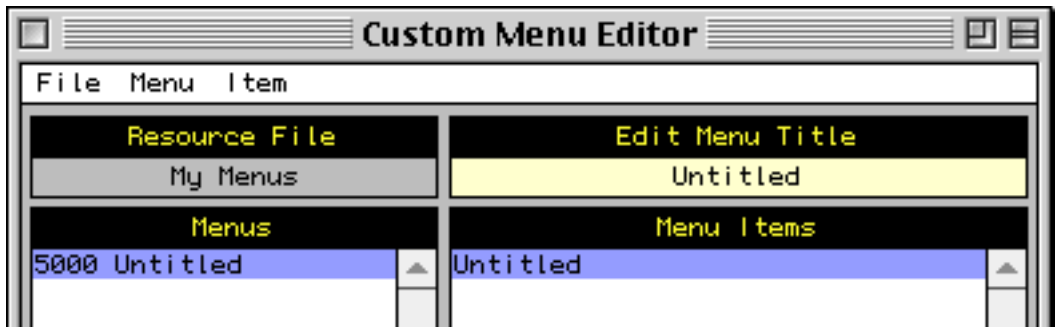
The new file starts with one menu named Untitled. The first thing you'll probably want to do is change the number of this menu. To do this use the **Renumber Menu** command.



The menu number can be anything from 160 to 65535. We recommend that you stick to numbers above 1000 (except for submenus, which must be from 160 to 255). You should also avoid the area around 200, see "Important Note for Custom Menu Developers" on page 1478.

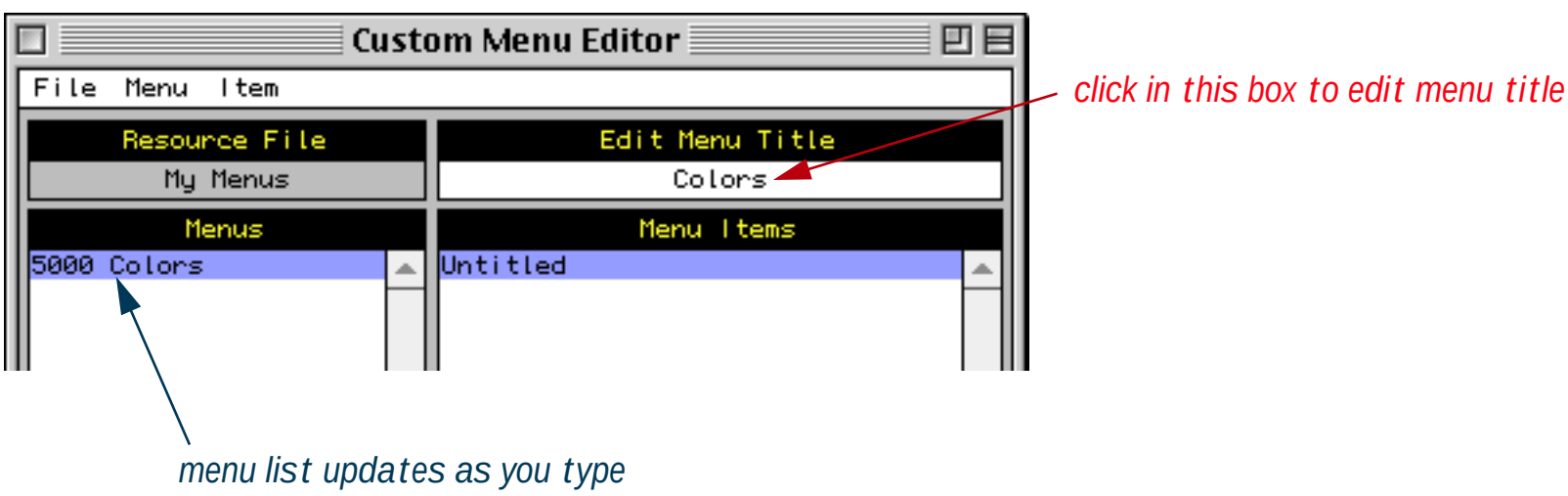


When you press **OK** the menu number will change, in this case to 5000.

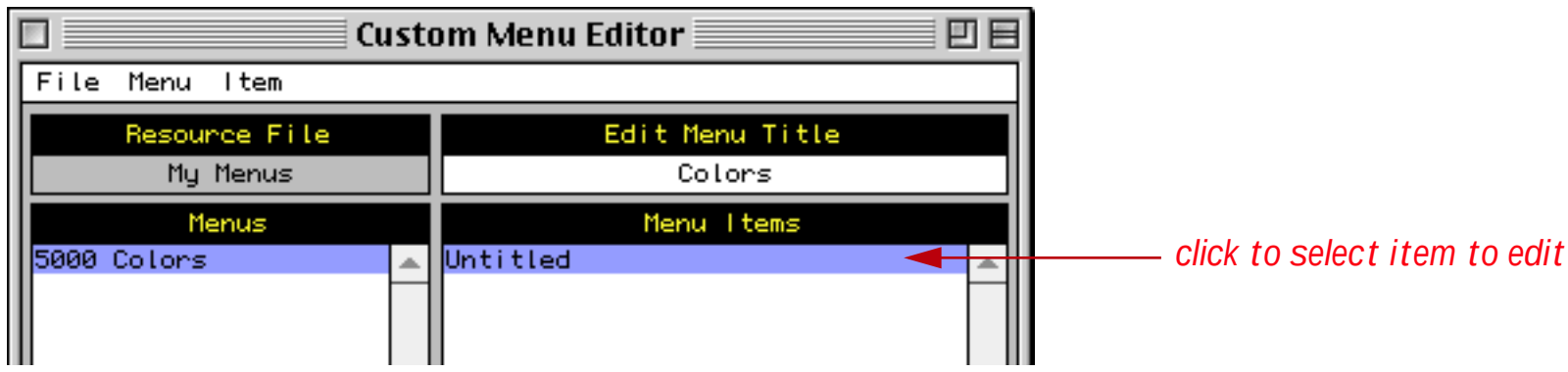


Editing Within a Menu

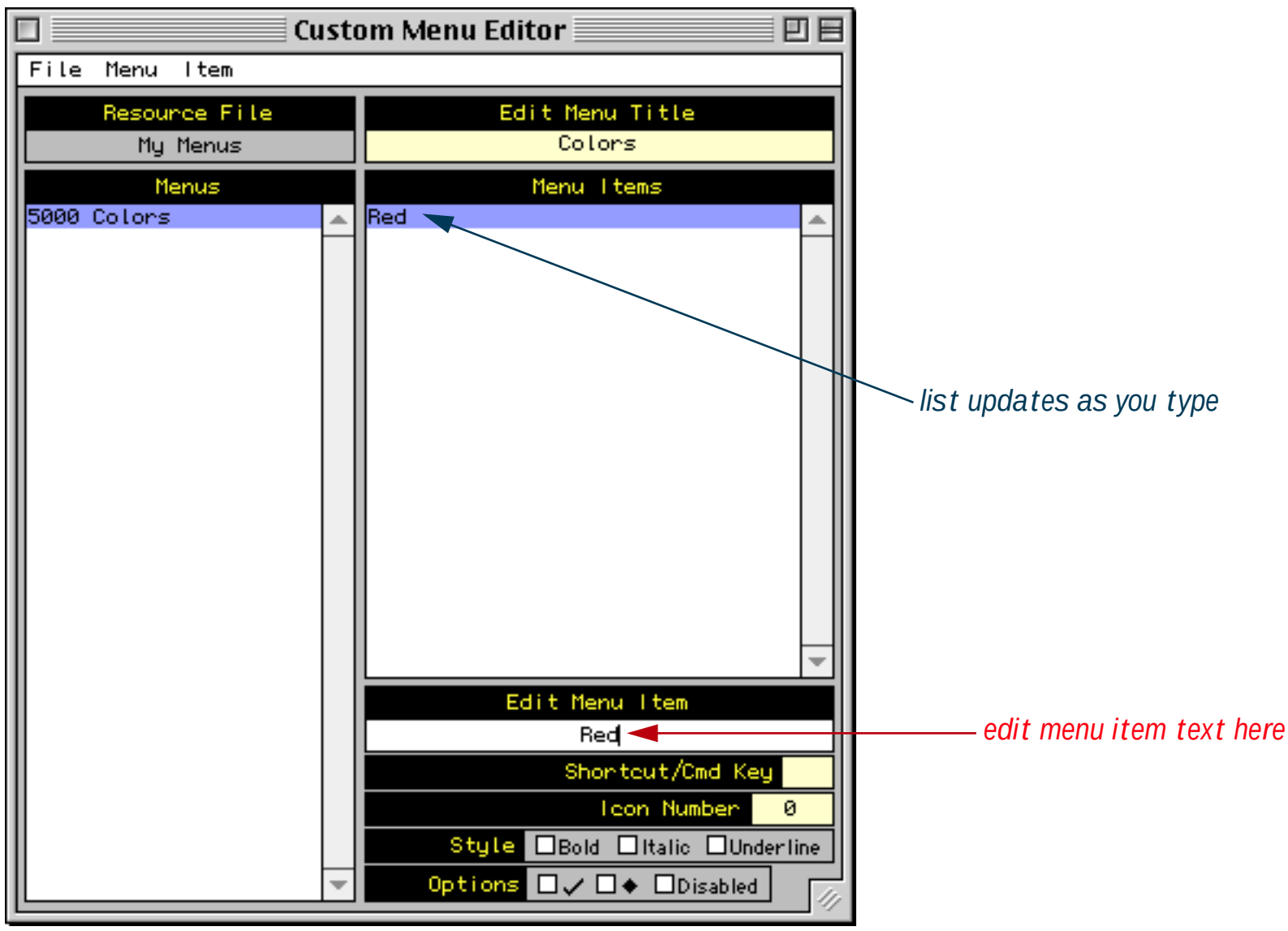
To edit the menu title simply click in the menu title and start typing.



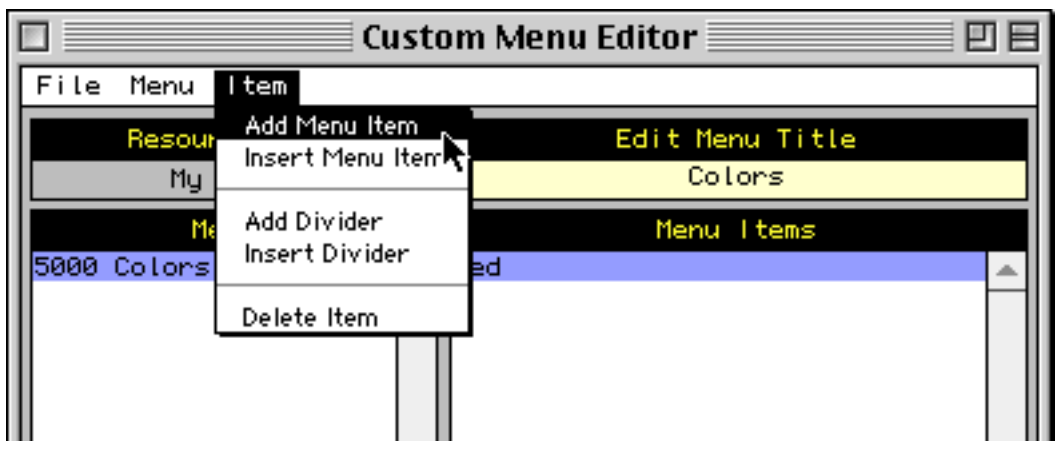
To edit an item within the menu start by selecting the item in the list. (In this case there is only one item so it is selected automatically).



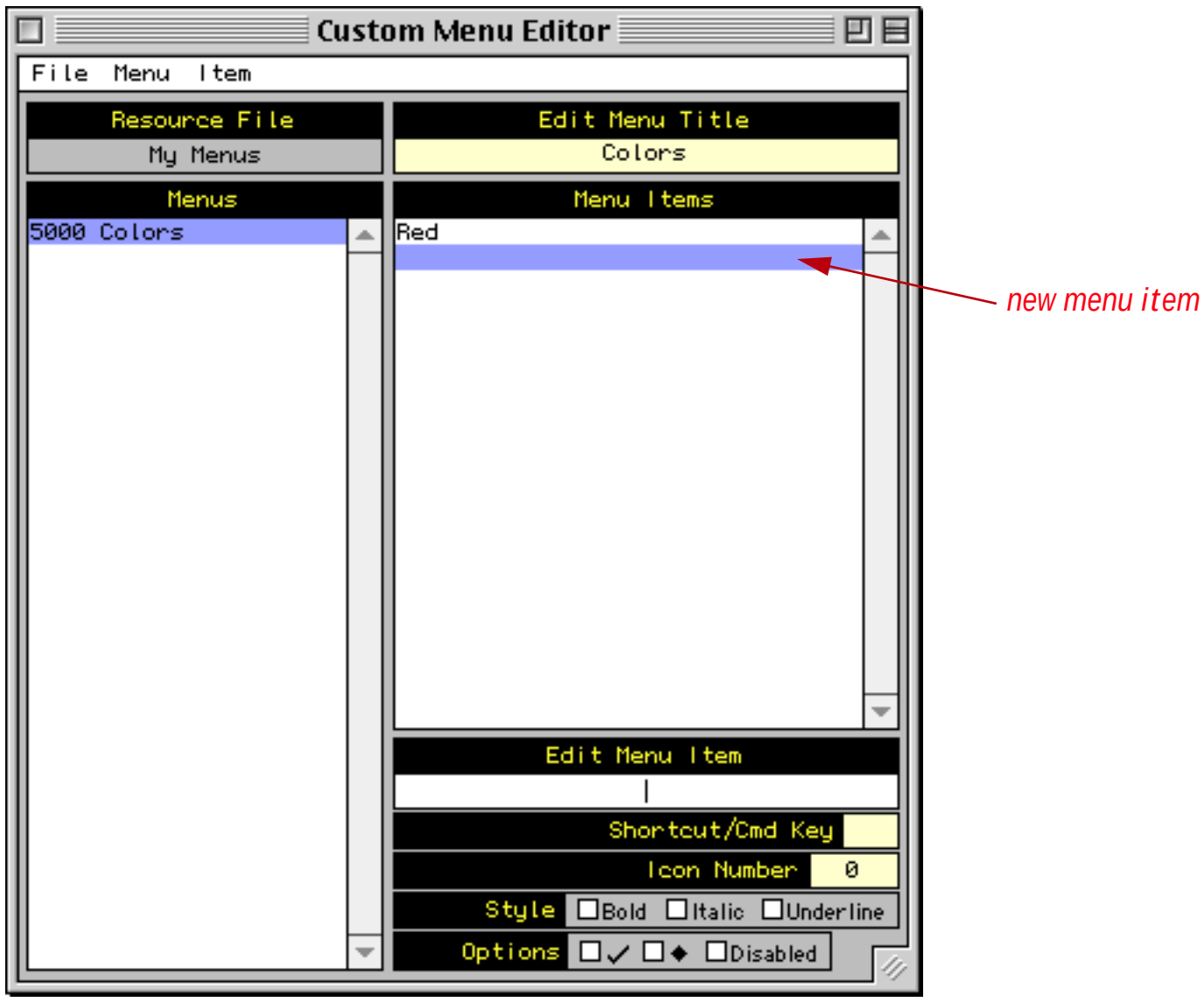
Once the item is selected you can edit it in the box below the bottom of the menu item list.



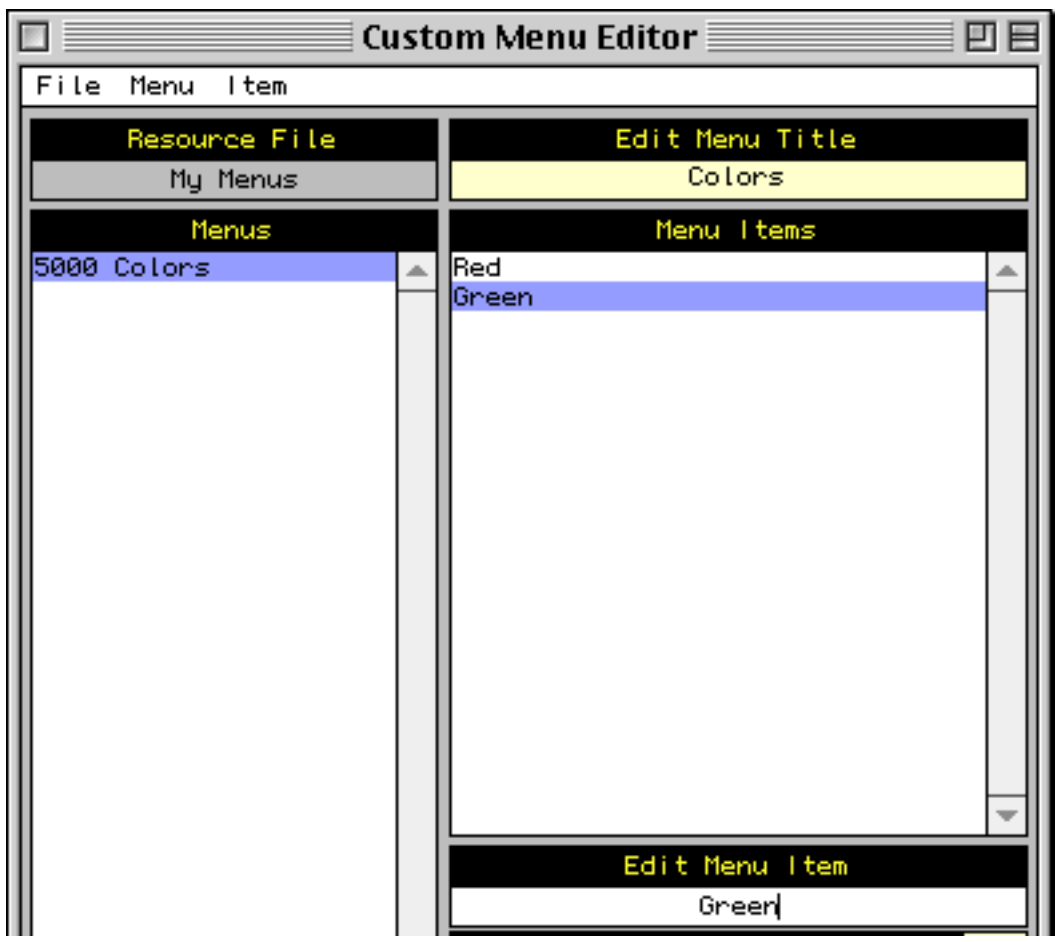
To add a new menu item choose **Add Menu Item** from the Item menu.



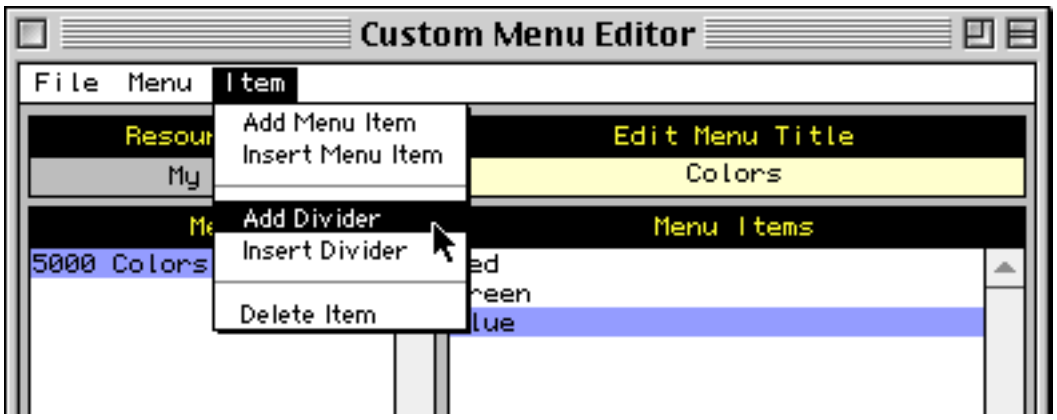
The new item is added to the end of the list and is initially empty.



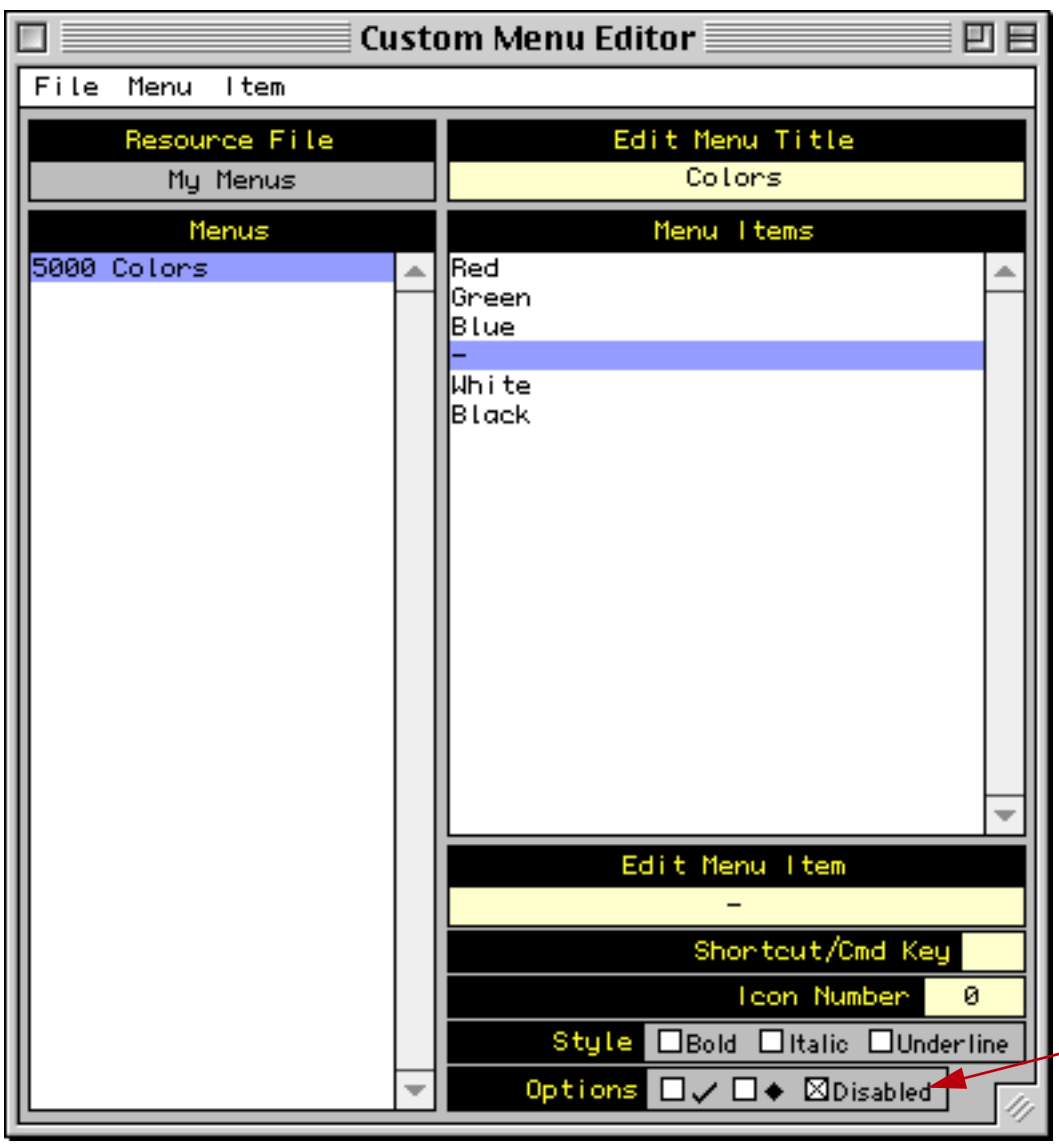
The new item is already selected so you can just start typing.



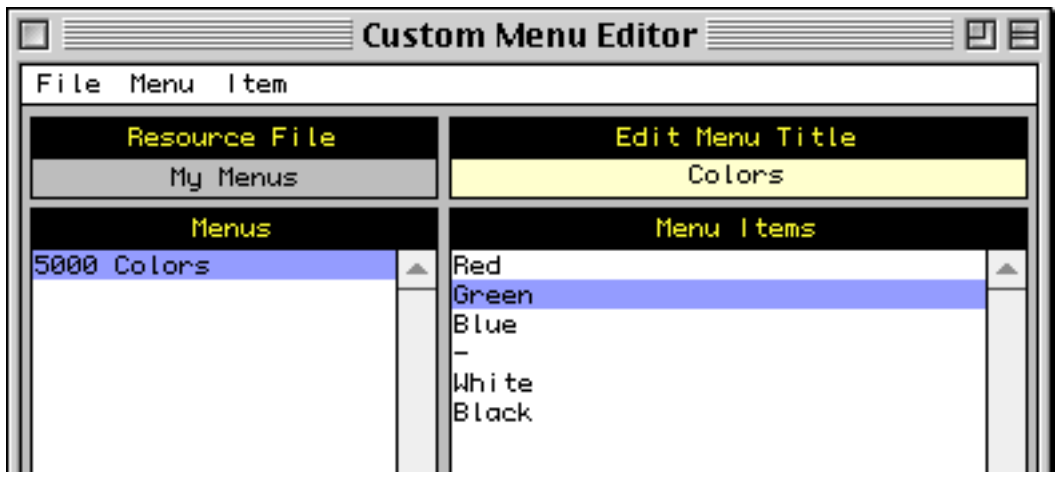
To add a divider line in the menu choose **Add Divider**.



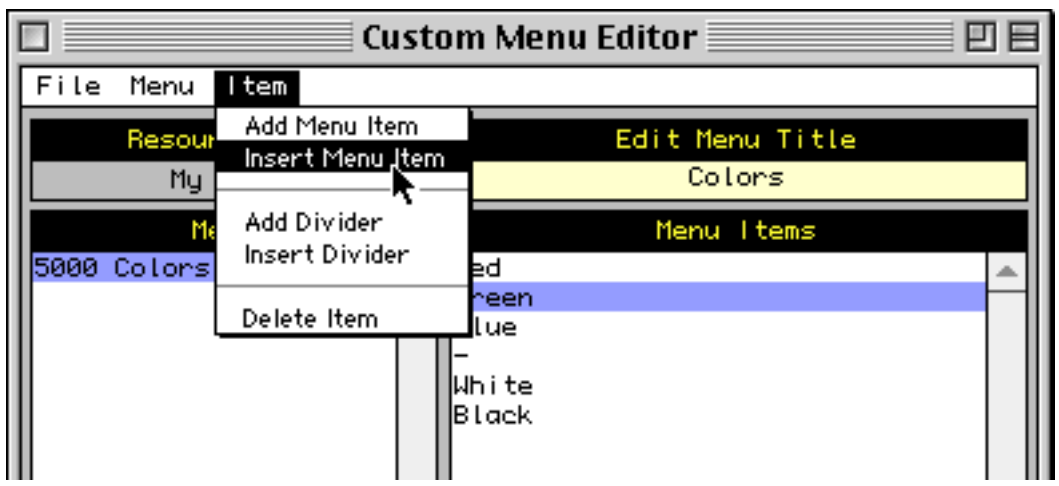
Here's a menu with a couple more items added after the divider. The divider is automatically disabled.



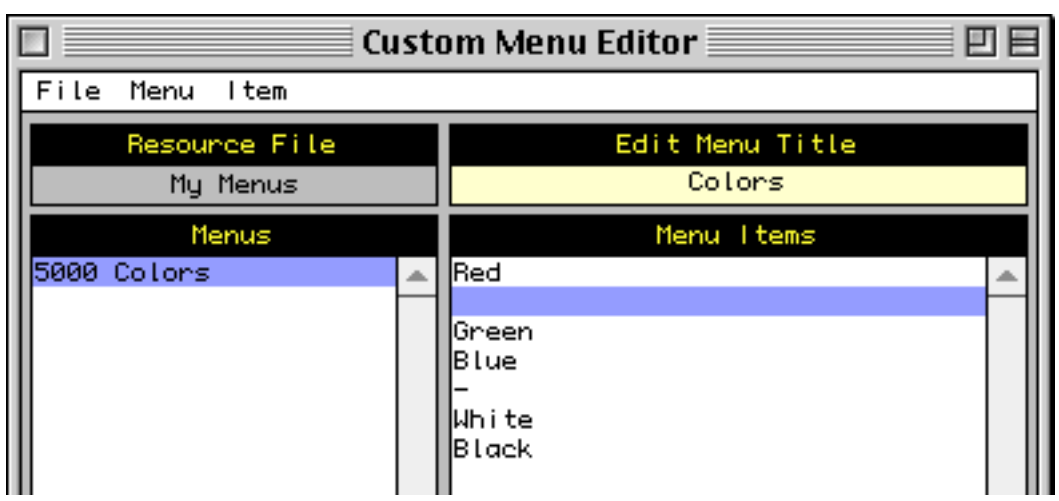
To insert a new item in the middle of the menu first select the spot where you want the new item to appear.



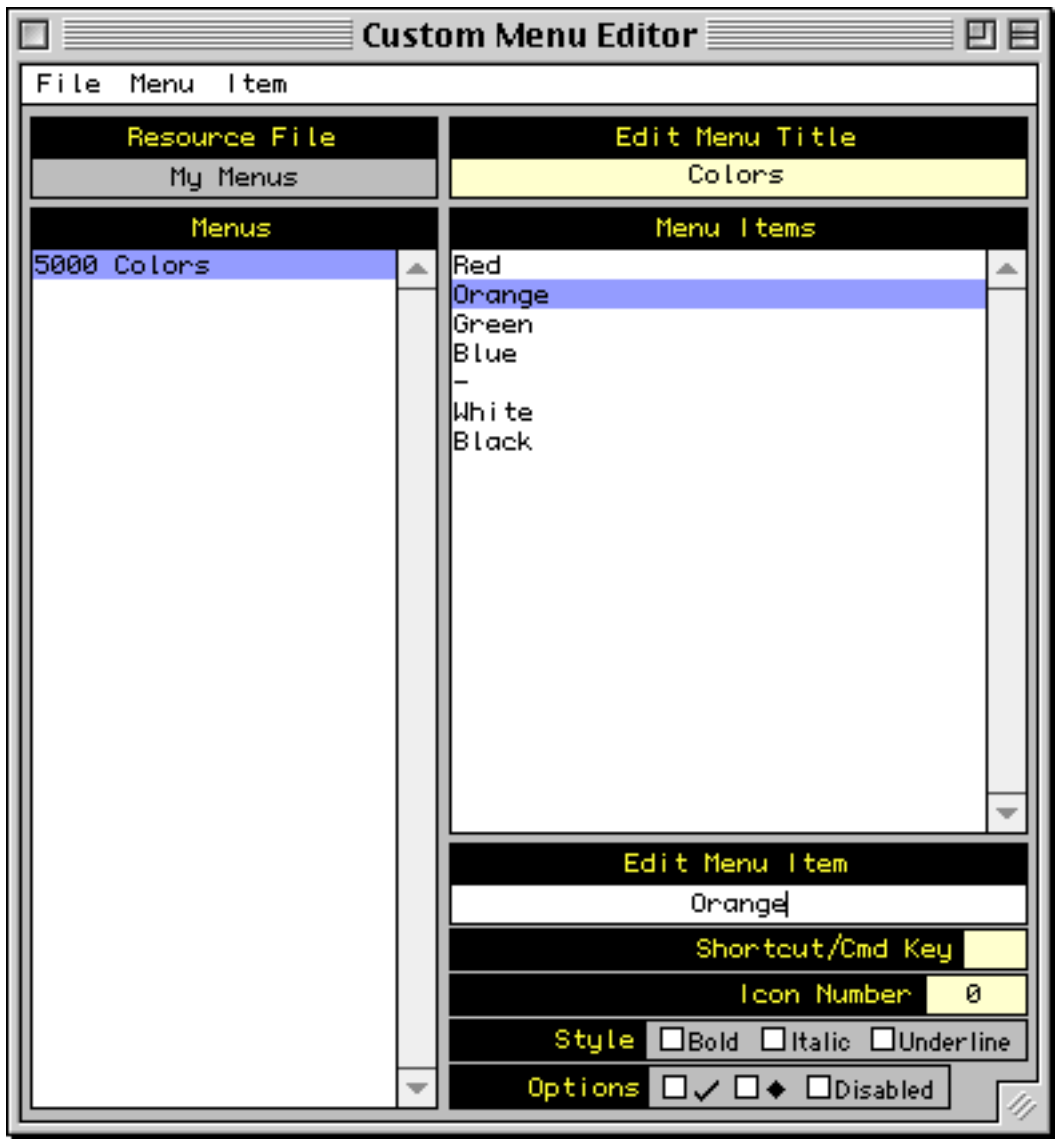
Choose **Insert Menu Item** from the Item menu.



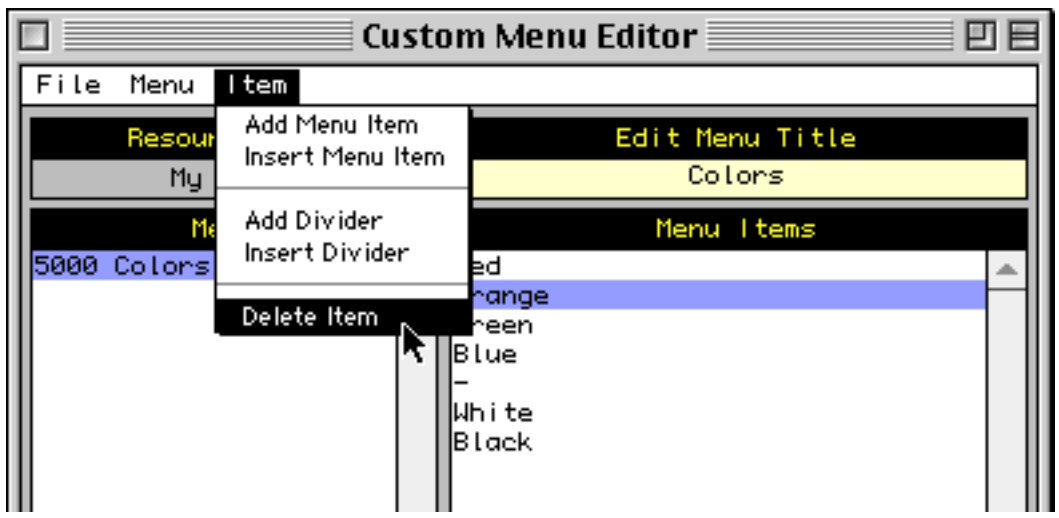
The new item appears in the spot you have selected.



Just type the text of the new menu item.



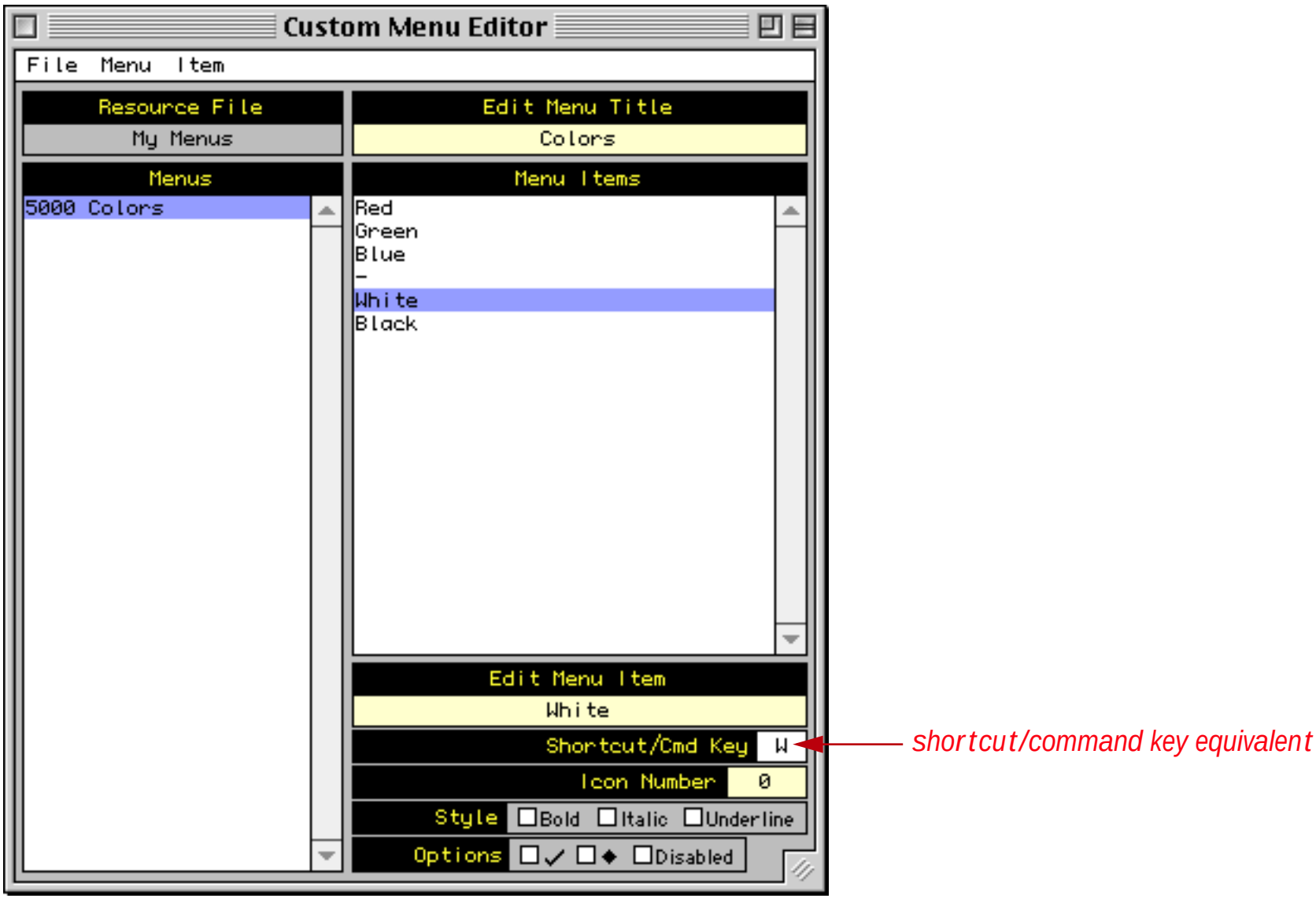
To delete a menu item first select the item then choose **Delete Item** from the Item menu.



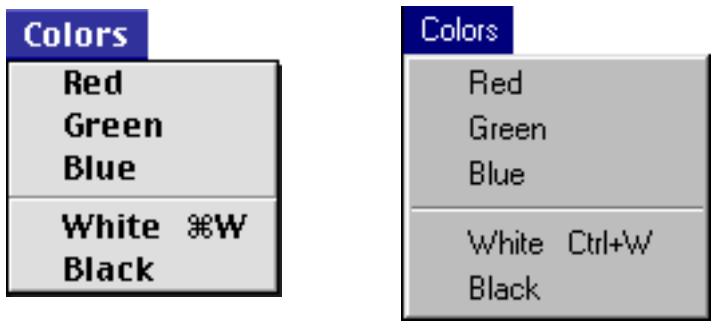
Don’t forget that you can edit any menu item simply by clicking on it and typing.

Command Key Equivalents/Shortcuts

Like other menus, custom menu items can have keys on the keyboard assigned to them. On the Macintosh these are called **Command Key Equivalents**, on the PC (Windows) they are called **shortcuts**. To assign a key to a menu item simply type the key into the appropriate box.



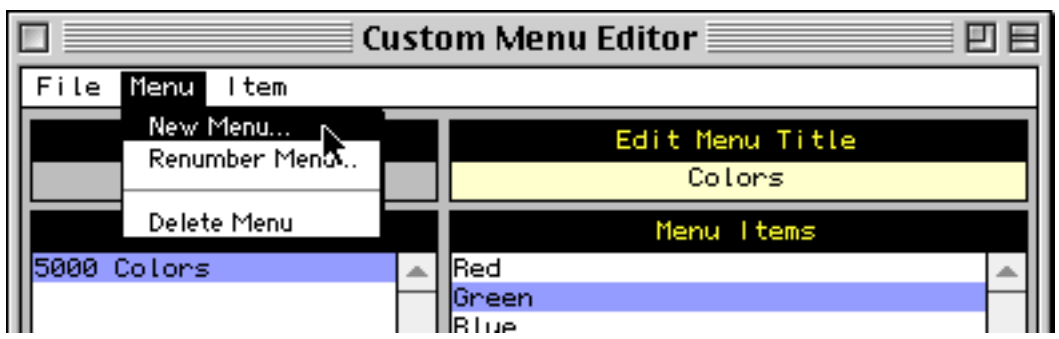
Here's what this menu looks like on both Macintosh and Windows.



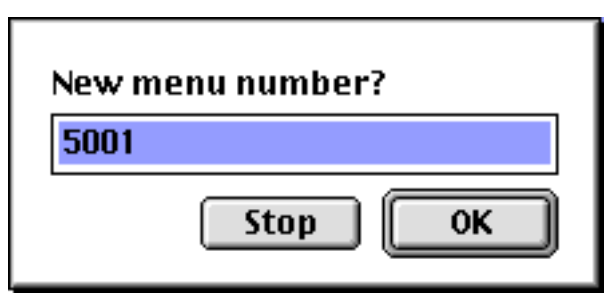
On the Macintosh the **White** menu item can be selected by pressing **Command-W**, while on the PC it can be selected by pressing **Control-W**.

Adding and Removing Entire Menus

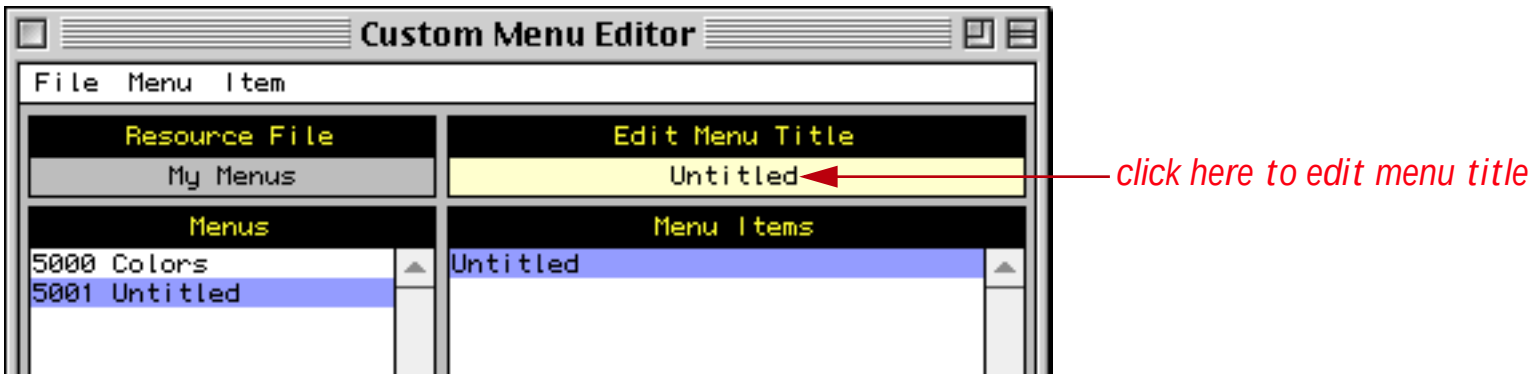
To add a new menu click on **New Menu** in the Menu menu (say that three times fast!).



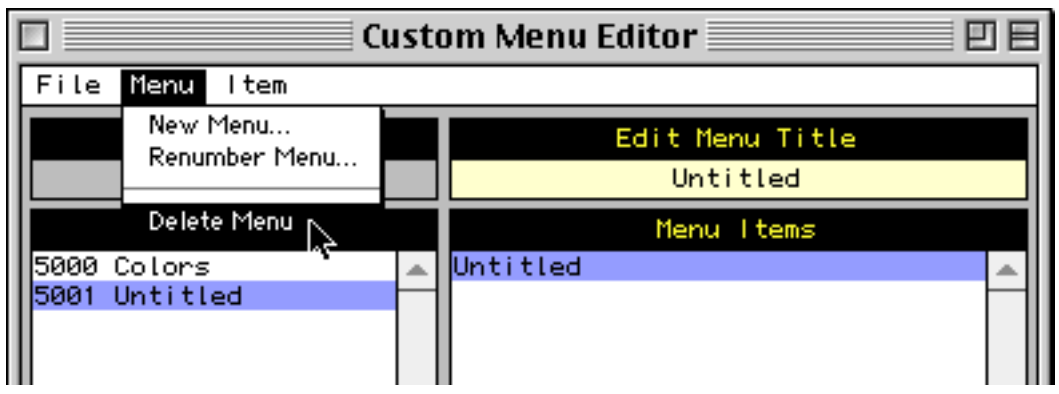
Panorama will ask you to select a number for the new menu. It will automatically assign the next available number, but you can select any number between 160 and 65535. We recommend that you stick to numbers above 1000 (except for submenus, which must be from 160 to 255).



When you press **OK** the new menu is created.



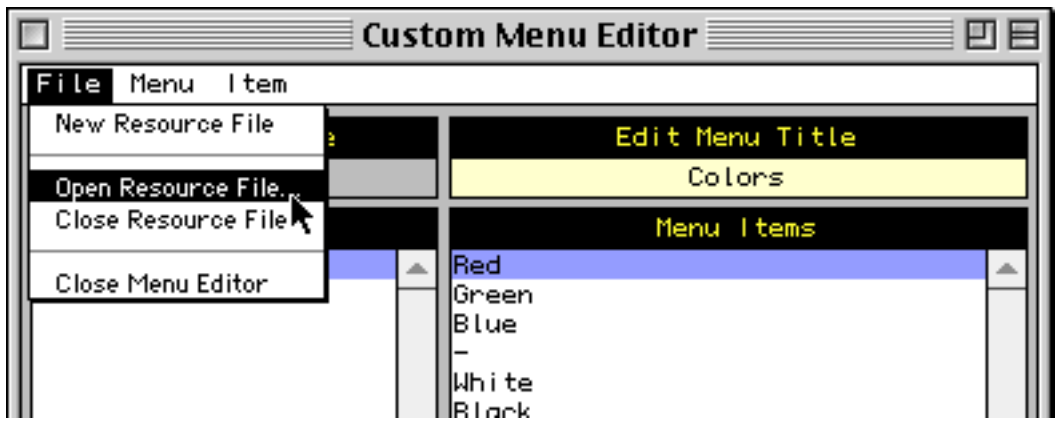
To delete a menu simply select it and choose **Delete Menu** from the Menu menu.



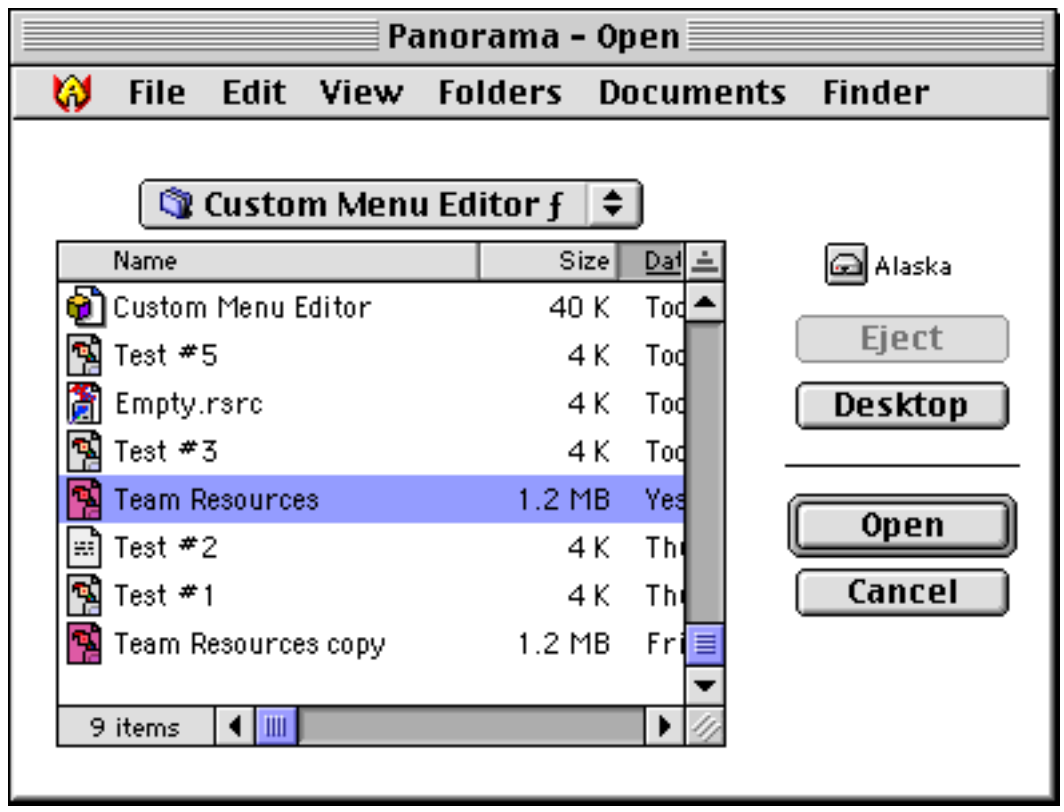
To change the number of a menu select the menu and choose **Renumber Menu**.

Opening and Closing Resource Files

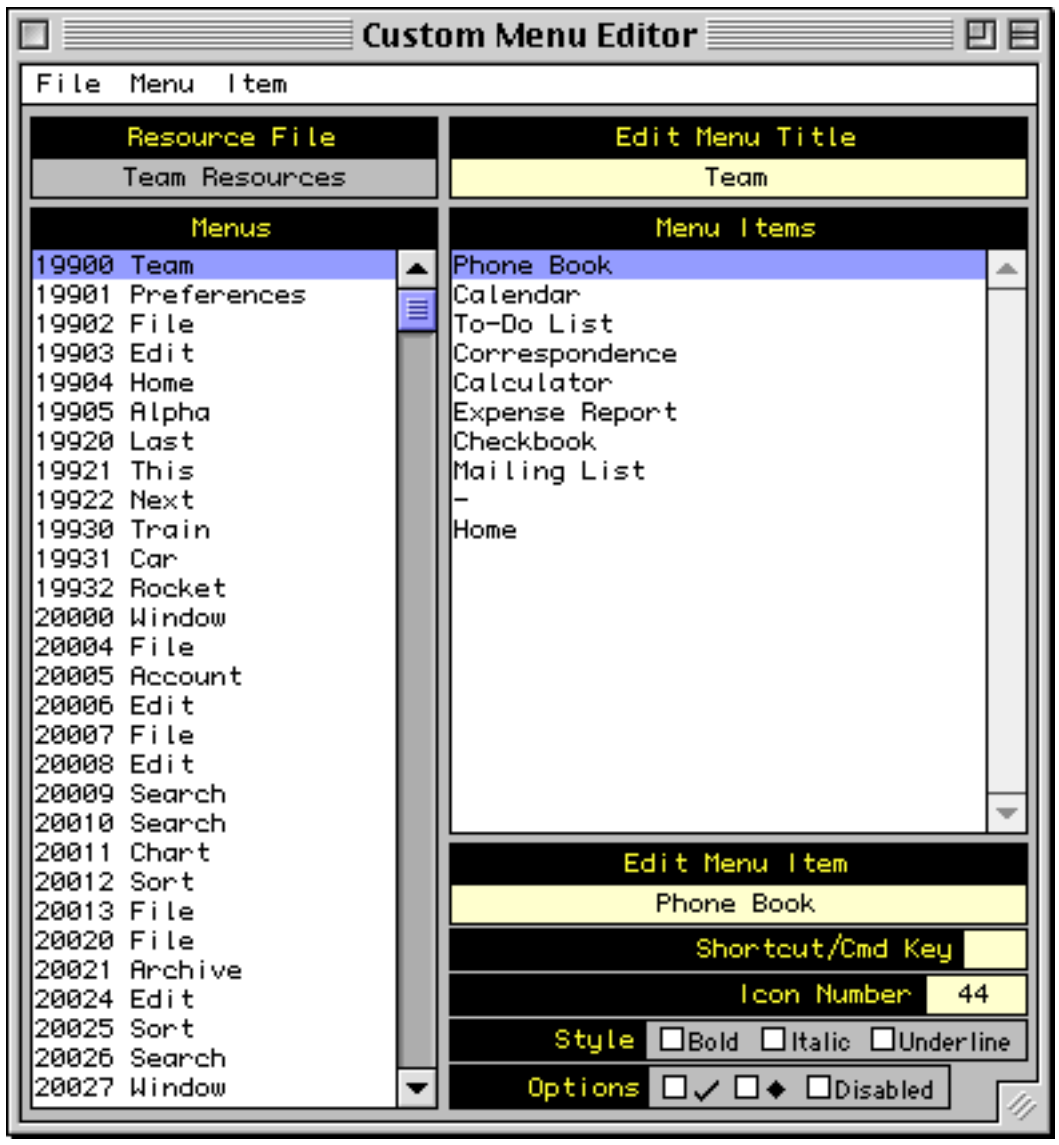
To open a resource file you have created previously choose **Open Resource File** from the File menu.



A dialog appears allowing you to select the folder and file.



Once the file is opened all the menus within that file will be listed.



When you are finished with the resource file you should close it by selecting **Close Resource File**. The current resource file is also closed automatically if you open or create another resource file.

Saving Resource Files

You may have noticed that there is no **Save** command in the File menu. Unlike Panorama databases, resources are saved immediately as you make each change.

Opening a Resource File in Panorama

Once a resource file containing menus has been prepared you can open and use it within Panorama. To open the resource file you must create a procedure which uses the `openresource` statement to open the resource file (see “OPENRESOURCE” on page 5580). To make sure that the resource file is opened immediately when the file is opened, we recommend that you place the `openresource` statement in the **.Initialize** procedure for the file (see “.Initialize” on page 1511 for more information on the **.Initialize** procedure).

The `openresource` statement requires one parameter—the name of the resource file to open. If you want the custom menus to appear immediately (the normal situation), the `openresource` statement must be followed by a `drawmenus` statement (see “DRAWMENUS” on page 5182). For example, if the resource file containing your menus is called Accounting Menus then the .Initialize procedure should contain the statements:

```
openresource "Accounting Menus"
drawmenus
```

Simply creating the .Initialize procedure does not open the resource file. The first time you create this procedure you must save the database, then close and re-open the database. The .Initialize procedure will open the resource file when you re-open the database, and you can begin using the custom menus. From then on the resource file will be opened automatically every time the database is opened.

Once a resource file has been opened, it will remain open until you quit from Panorama or until you close the resource file with the `closeresource` statement in a procedure (see “CLOSERESOURCE” on page 5111).

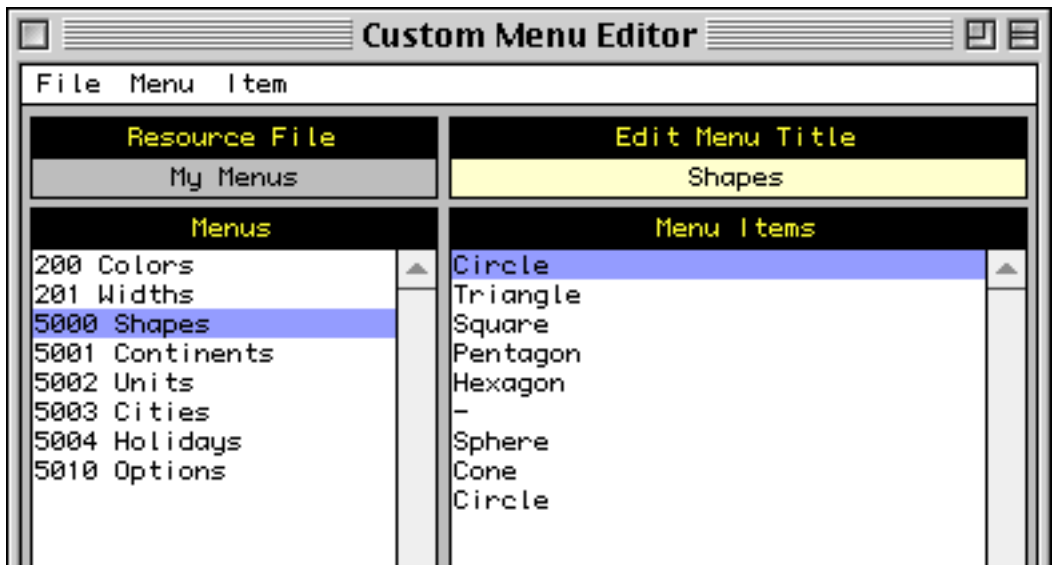
Sharing A Resource File Between Databases

If you have several databases that are usually used together you can create a single resource file containing the custom menus for all of the databases. This helps to reduce clutter on your disk, especially for more complex Panorama applications.

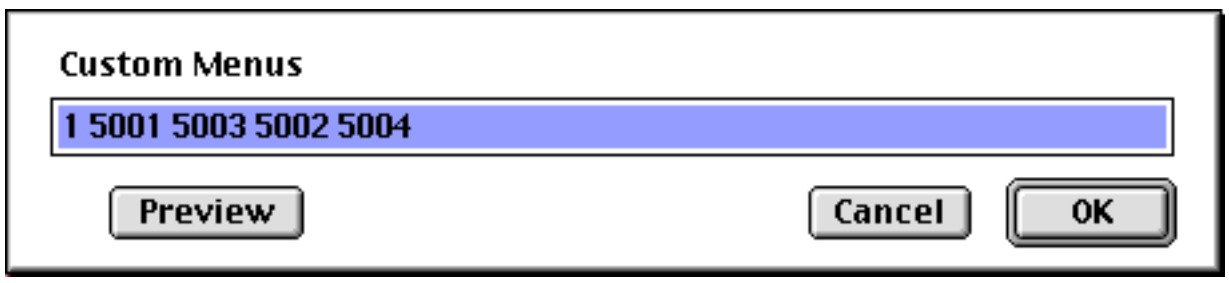
When a resource file is shared in this way, each database should contain `openresource` and `drawmenus` statements in their **.Initialize** procedures, as described in the last section. It doesn’t hurt to open the same resource file more than once. Panorama realizes that the resource file is already open so that it doesn’t have to open the file again.

Assigning Custom Menus to a Form

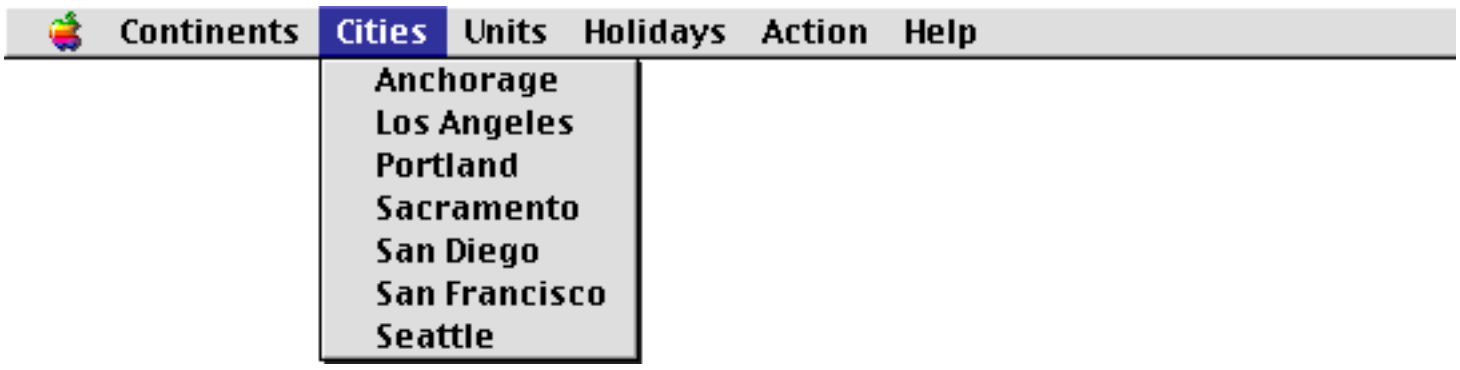
To install custom menus for a Panorama data sheet or form, use the **Custom Menu** command in Panorama’s Setup menu. The **Custom Menu** dialog allows you to specify the ID numbers of the menus to be included in the menu bar. For example, suppose you have prepared a resource file with these menus:



To set up custom menus for a form go into Graphic Design Mode, then choose the **Install Custom Menus** command from the Setup menu. In the dialog enter the menu numbers for each custom menu. (In this case the first menu is 1, the Apple menu.)

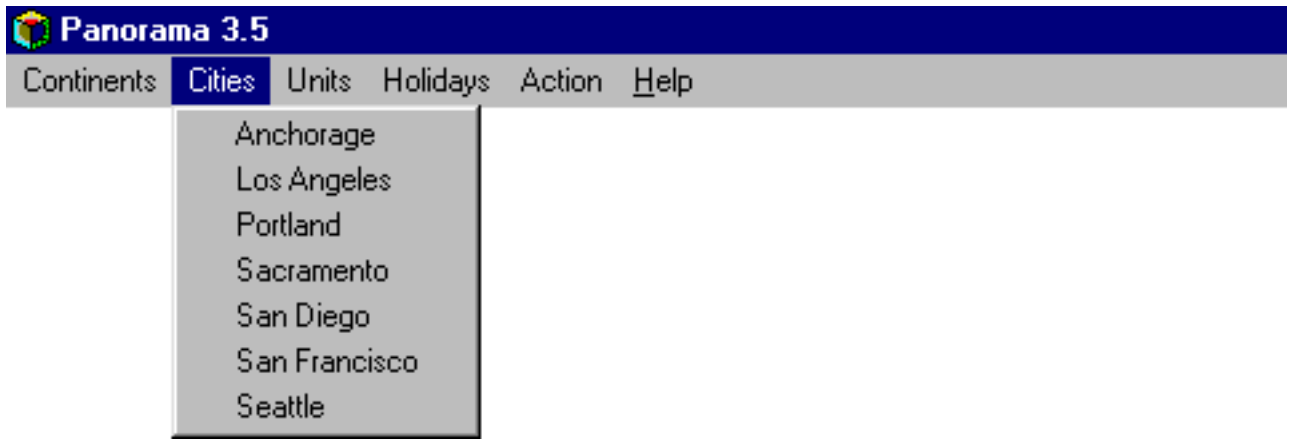


After the menu numbers have been entered press **OK** to close the dialog. Then switch the form to Data Access Mode to check out the menu configuration.



Notice that the menu bar includes the **Action** and **Help** menus. To lean how to get rid of the **Action** menu, see "“Unlisted” Procedures" on page 1464. There is no way to get rid of the **Help** menu.

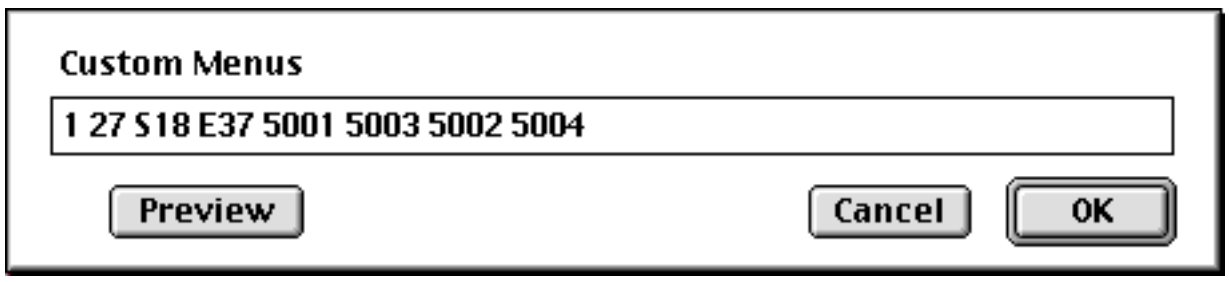
Here's what this same custom menu configuration looks like on Windows. Notice that the Apple menu has automatically been removed.



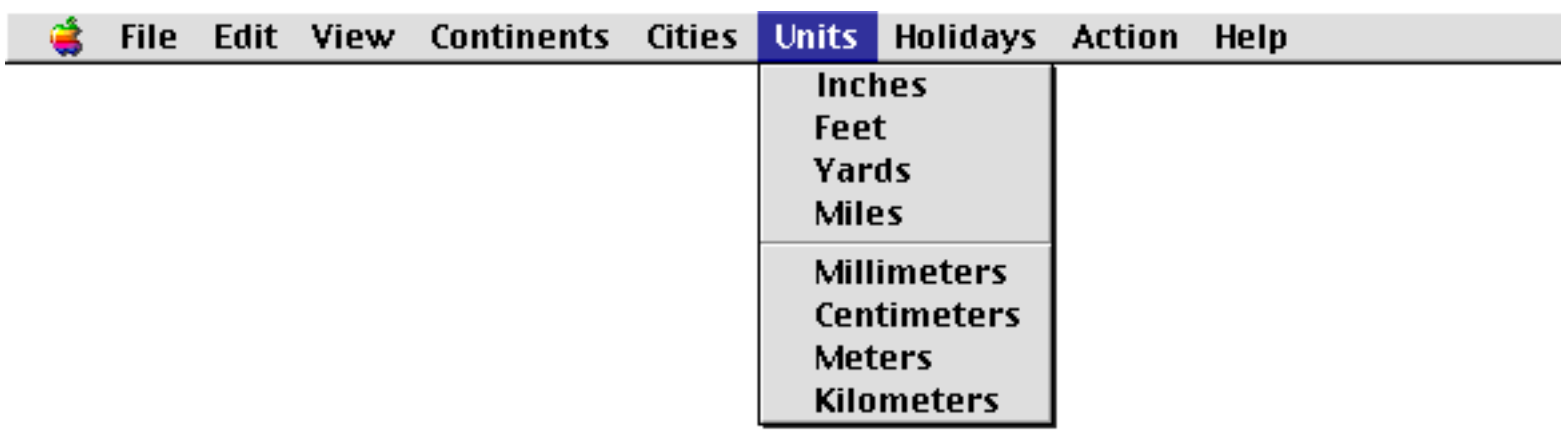
You can mix your own custom menus with Panorama's standard menus. For example, you may wish to combine Panorama's standard **Apple**, **File**, and **Edit** menus with your own custom menus. To use a standard Panorama menu, simply type its ID number into the **Custom Menu** dialog. The ID numbers of Panorama's standard menus are listed below. Notice that the menu ID for the Edit Menu must be preceded by the letter **E**. Menus that are submenus must be preceded by the letter **S**.

Number	Menu	Notes
1		Apple Menu
7	File	Data Sheet Only
27	File	Forms Only
S18		Arrange Submenu
E19	Edit	Data Sheet Only
E37	Edit	Forms Only
26	View	
28	Fields	Data Sheet Only
73	Text	Data Sheet Only
S3	Font	Data Sheet Only
S4	Size	Data Sheet Only
8	Search	
9	Sort	
10	Math	
68	Setup	Data Sheet Only
70	Setup	Form Only

Here is the configuration dialog for a typical menu that combines both standard menus and custom menus.

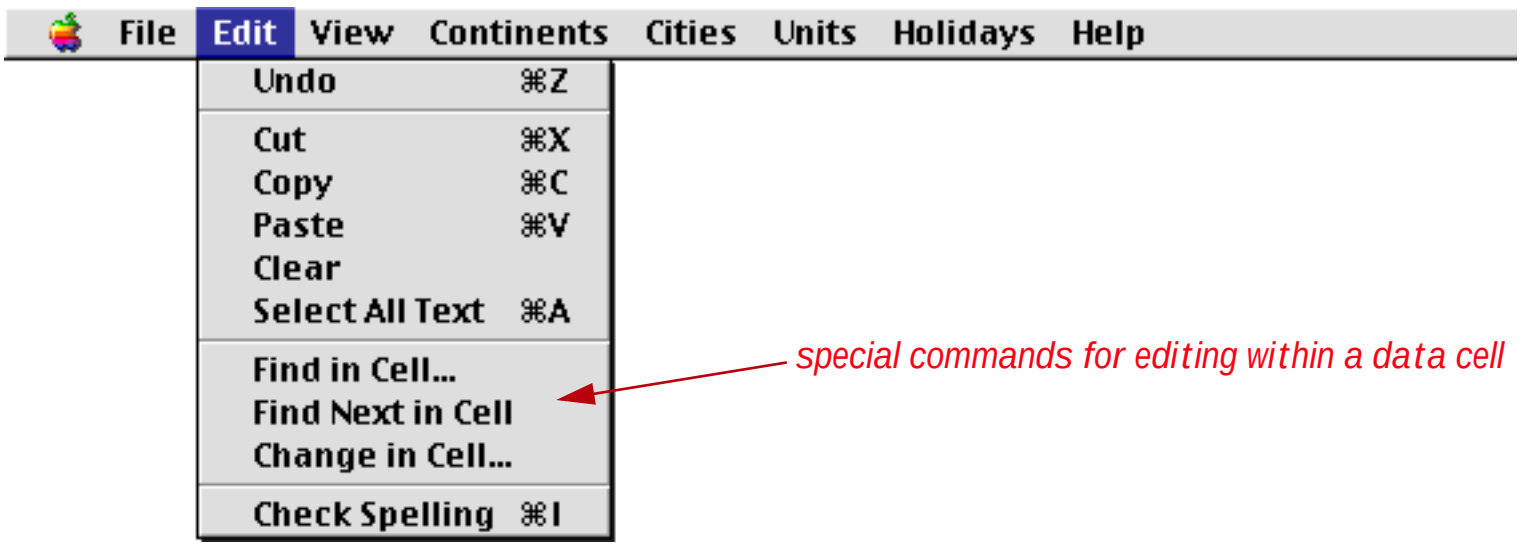


Here's what this menu configuration looks like.



Notice that Panorama has added a **View** menu after the **Edit** menu. Panorama will automatically add the **View** menu after the **Edit** menu if their is an **Edit** menu.

The **Edit** menu should always be prefixed with an **E**, even if you create your own custom **Edit** menu. The **E** identifies the location of the **Edit** menu for Panorama. Panorama uses this information two ways. First, it normally inserts the **View** menu after this location. Secondly, when a data cell is being edited Panorama temporarily substitutes a special **Edit** menu for editing data cells.



If your custom menu configuration doesn't include a menu that begins with an **E** then these commands will not be available when editing a data cell.

If the menu configuration does not include the **View** menu (26) it will normally be placed after the **Edit** menu, as described above. If you wish to include the **Edit** menu but not include a **View** menu then place X26 somewhere in the menu configuration. If you wish to position the **View** menu somewhere other than after the **Edit** menu then simply place the number 26 in the spot where you want the **View** menu to appear.

You can preview your custom menu configuration by pressing the **Preview** button. You can try out the new menus to see what they look like (they won't do anything while being previewed). (Note: The menu preview will not work if the resource file has not been opened—see "Opening a Resource File in Panorama" on page 1490.)

Each form can have its own separate menu configuration, as can the data sheet. You must use the **Custom Menu** dialog to set up the menu commands for each form or data sheet. If several forms use the same menus, you can use the **Copy** and **Paste** commands to transfer the menu ID numbers from one form to another. Also if you use the **Copy Form**/**Paste Form** command to copy an entire form (see "Copying an Entire Form" on page 566), the custom menu setup will automatically be copied to the new form.

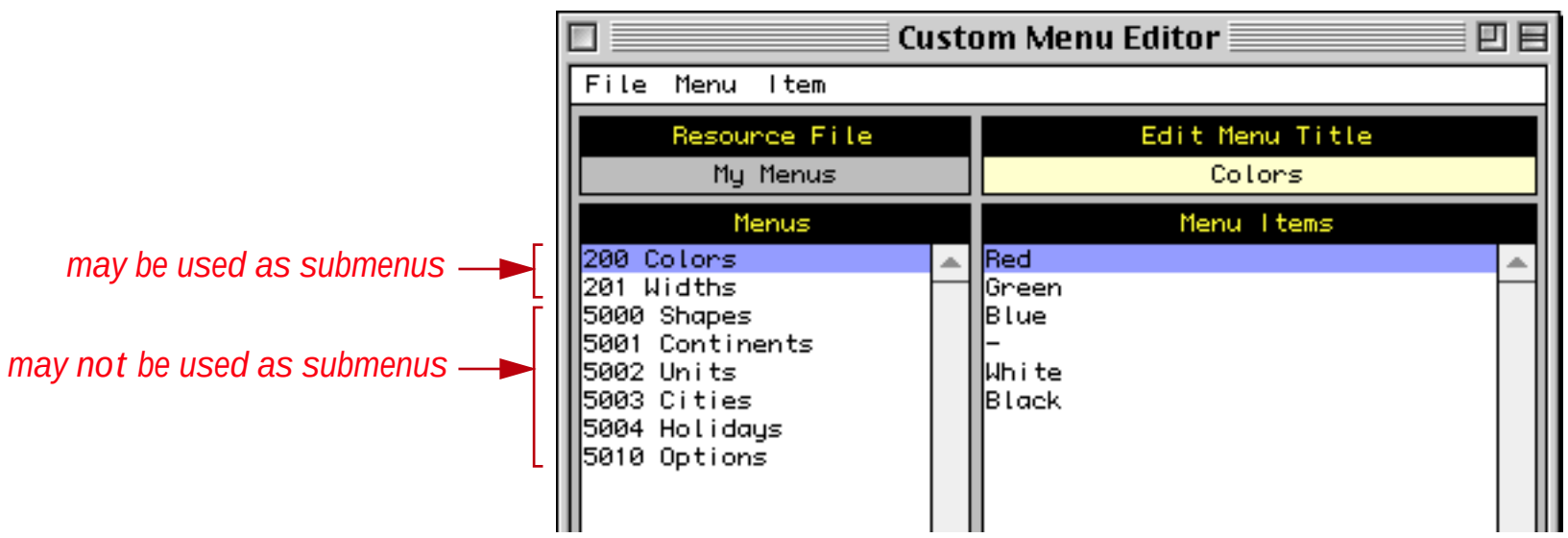
The .CustomMenuProcedure

What happens when a user pulls down a custom menu and selects a menu item? Choosing an item in a custom menu automatically starts a special procedure. This procedure must be called **.CustomMenu**. If the database does not have a **.CustomMenu** procedure then you'll still be able to pull down custom menus, but they won't do anything. To learn how to program the **.CustomMenu** procedure see "The .CustomMenu Procedure" on page 1466.

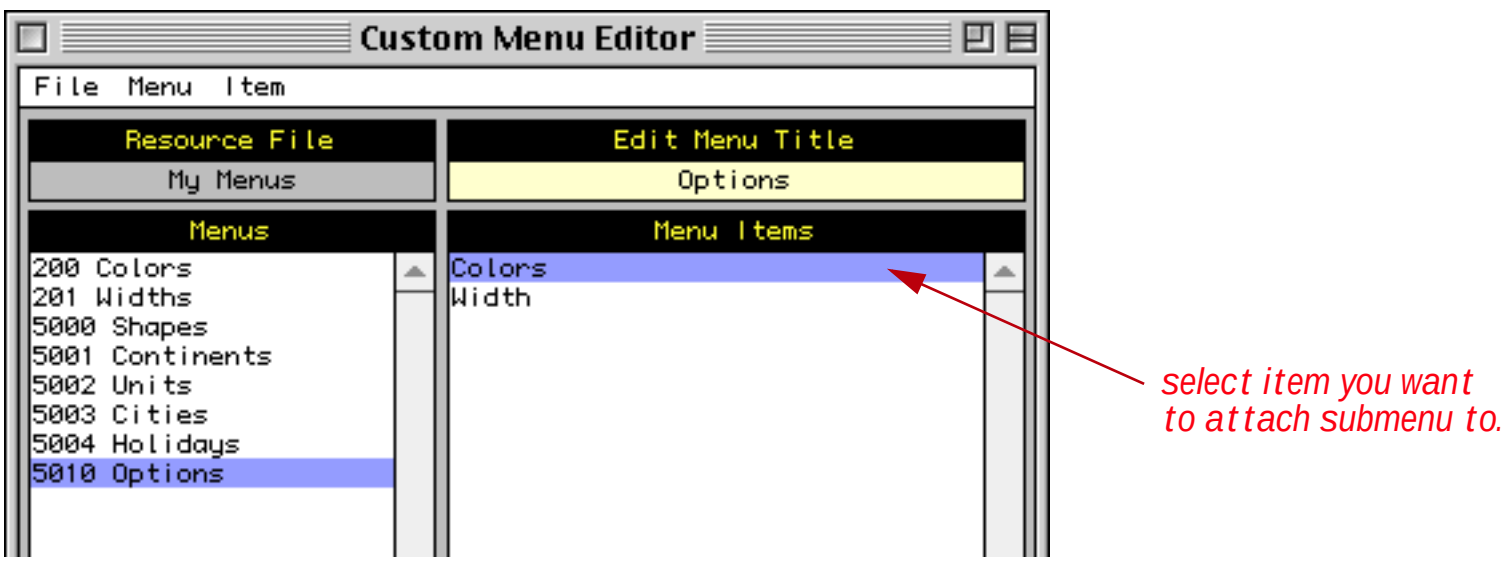
Submenus (Hierarchal Menus)

Custom menus can be nested up to five levels deep. This is usually called submenus or hierarchical menus.

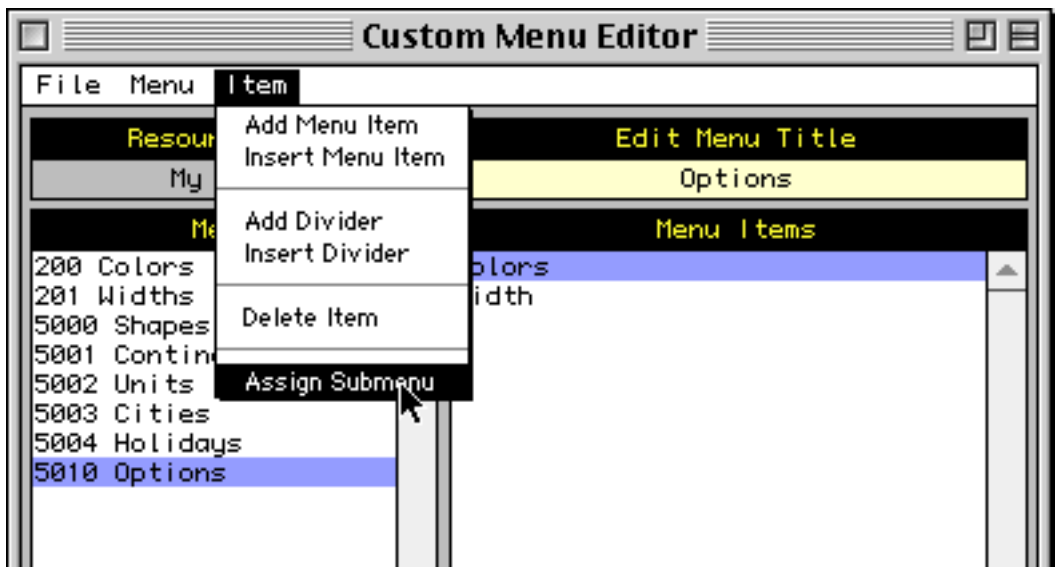
Submenus must be prepared in advance in a resource file, just like any other custom menu (see "Preparing a Resource File" on page 1479). However, submenus must have menu ID's between 200 and 255. Be sure to give the submenu a name, even though that name will not appear in the menu bar. (Even though the name doesn't appear in the menu bar, the name will still be returned by the `info("trigger")` command.)



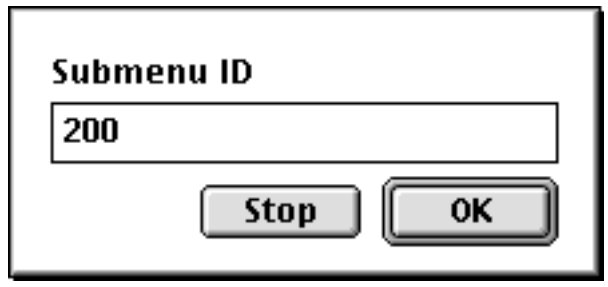
After the submenu is created, it must be attached to a menu item in another menu. Select the menu item that you want to attach the submenu to.



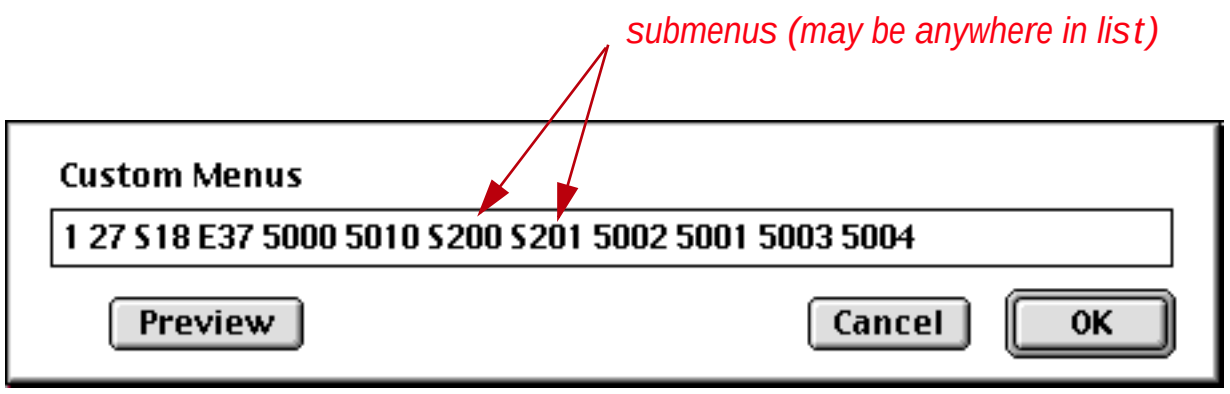
Once the item is selected choose **Assign Submenu** from the **Item** menu.



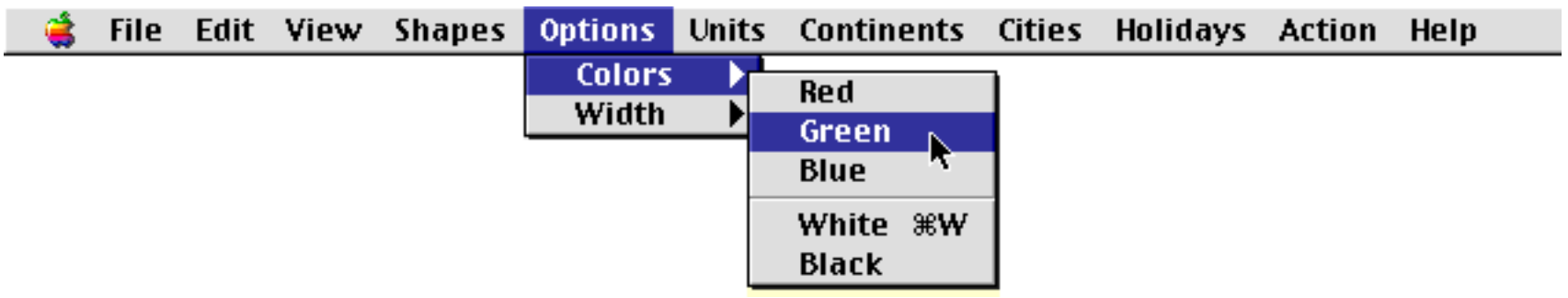
Type in the ID of the submenu you want to attach to this item. In this case we'll type 200 to attach the **Colors** menu as a Submenu.



Like other custom menus, submenus must be assigned to one or more views in the database using the **Custom Menu** dialog (in the Setup Menu). However, the menu ID number should be preceded by the letter **S**. For example, if your submenu has a menu ID of 200, it should be installed in the Custom Menu dialog as S200 (or s200). Preceding the menu ID with the letter **S** tells Panorama not to display the menu in the menu bar.



Once the submenu is installed, it can be used like any other custom menu.



If the user selects an item in the submenu it will trigger the **.CustomMenu** procedure (see “The .CustomMenu Procedure” on page 1466). The `info("trigger")` function will return the name of the submenu and the submenu item. For example, if the user selects **Green** in the **Colors** submenu, which is a submenu in the **Options** menu, the `info("trigger")` function will return Menu.Colors.Green.

Changing Custom Menus on the Fly

One of the advantages of custom menus is that they can be modified on the fly by the programmer. The programmer can add and remove checkmarks, gray out menu items (enable or disable), or even replace an entire menu with a different one.

(Warning: The commands described in the following section work only with custom menus. They do not work with Panorama's standard menus.)

Specifying Menus and Menu Items

To change a menu, Panorama needs to know exactly what menu and menu item you want to modify or access. There are two ways you can specify menus and menu items: 1) by name, or 2) by numeric ID.

For example, suppose you have a custom menu named **Transaction** with a menu ID of 3012. The fourth item in this menu is **Void**. There are four possible combinations that can be used to specify this menu item:

Example 1: `"Transaction" "Void"`

Example 2: `3012 4`

Example 3: `"Transaction" 4`

Example 4: `3012 "Void"`

In the first example, both the menu and the menu item are specified by name. In the second example, both are specified by numeric ID. In the final two examples the methods are mixed.

How does the programmer decide which method to use? If the name is known when the program is written, it should probably be used for clarity. However, in some cases the name cannot be known. In those cases the numeric ID must be used.

Menu Marks (Checkmarks, etc.)

A procedure can add or remove any kind of mark to a custom menu item. Any character can be used as a mark, but the most common marks are checkmarks (✓) and diamonds (◆). You cannot type these characters into a procedure, but you can use the `chr(` function to create these special characters.

For the checkmark (✓), use `chr(18)`

For the diamond (◆), use `chr(19)`

The `SetMenuMark` statement adds or removes a mark to a specific menu item (see "SETMENUMARK" on page 5742). The statement requires three parameters: 1) the menu, 2) the menu item, and 3) the mark character. If you supply an empty mark character, Panorama will remove any mark that is attached to the menu item.

The `GetMenuMark` statement allows a program to find out if and how a menu item is marked (see "GETMENUMARK" on page 5302). This statement has two parameters, 1) the menu, 2) the menu item. The statement finds out what character (if any) is used to mark the menu item, then places that character in the clipboard where it can be retrieved with the `clipboard()` function (see "CLIPBOARD(" on page 5107).

The `ClearMenuMarks` statement removes all the marks from every menu item in a menu. The statement requires one parameter: the menu name or ID number. This statement is especially useful when you don't know what menu items might be marked. The marks could also be cleared by using the `SetMenuMark` command over and over again in a loop, but `ClearMenuMarks` is much faster.

Checkmark On/Off Toggle

One of the most common reasons to mark a menu item is to indicate the status of something represented by the menu item: on / off, rush / normal, locked / unlocked, etc. For example, suppose a database had a custom menu item in the **Order** menu named **Rush**, and that this item could be either checked or unchecked.



The sample program fragment below shows the section of the **.CustomMenu** procedure that adds and removes the checkmark.

```
if info("trigger") = "Menu.Order.Rush"
   getmenumark "Order" "Rush"
   if clipboard() = ""
      setmenumark "Order","Rush",chr(18)
   else
      setmenumark "Order","Rush" ""
   endif
endif
```

The program fragment starts by seeing if **Rush** is checked already. If not, it adds the checkmark, otherwise it removes it.

Other procedures in this database can use `GetMenuMark` to check if the **Rush** status is turned on. This sample program fragment makes today the ship date if rush is on, otherwise the ship date is in 5 days.

```
getmenumark "Order","Rush"
if clipboard() ≠ ""
   ShipDate=today()
else
   ShipDate=today()+5
endif
```

Checking One Item in a Group

Another common use for menu checkmarks is to check one item from a group. The sample below is designed to work with a **Shipper** custom menu that contains a list of shipping companies (**UPS**, **US Mail**, **FedEx**, etc.).



The program fragment below should be part of the **.CustomMenu** procedure.

```
global PreferredShipper
local MenuName,MenuItemName
MenuName=array(info("trigger"),2,".")
MenuItemName=array(info("trigger"),3,".")

if MenuName="Shipper"
   clearmenumarks "Shipper"
   PreferredShipper = MenuItemName
   setmenumark "Shipper",MenuItemName,chr(18)
   stop
endif
```

When the user selects any item from the **Shipper** menu, the procedure starts by clearing all of the menu items in the menu. Then it saves the new preferred shipper in a global variable, where it can be accessed by other procedures at any time. Finally a checkmark is added next to the new preferred shipper name in the menu.

Groups with Other…

This example extends the previous example by adding an **Other…** item to the **Shipper** menu.



If the user selects this item, the procedure will display a dialog allowing the user to type in the name of any shipper.



After the new value is entered, the procedure changes the name of the Other menu to show the new selection, for example **Other (DHL)…**.



(Note: This sample assumes that the **Shipper** menu contains 6 items and that the **Other** item is the last one in the menu. The `SetMenuText` line will have to be adjusted if the menu contains more or fewer items.)

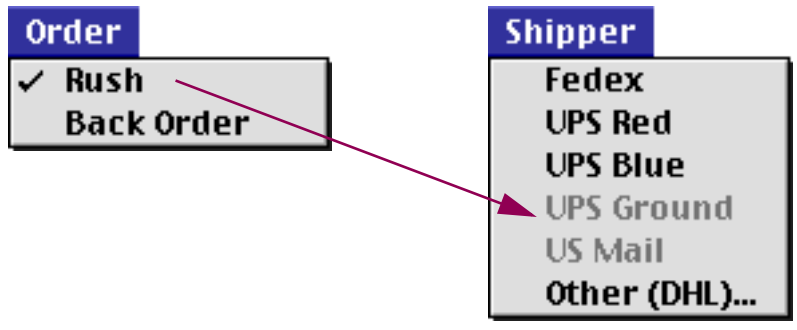
```
global PreferredShipper
local MenuName,MenuItemName
MenuName=array(info("trigger"),2,".")
MenuItemName=array(info("trigger"),3,".")

if MenuName="Shipper"
   clearmenumarks "Shipper"
   if MenuItemName beginswith "Other"
      gettext "Preferred Shipper…",PreferredShipper
      setmenutext "Shipper",6,"Other ("+PreferredShipper+")…"
      stop
   endif
   PreferredShipper = MenuItemName
   setmenumark "Shipper",MenuItemName,chr(18)
   stop
endif
```

Disabling Menu Items

A procedure can disable custom menu items when they would not be appropriate. The `MenuDisable` statement disables a menu item, turning it gray in the menu. The `MenuEnable` statement enables a menu item that has been disabled. Both of these statements have two parameters: 1) the menu name or ID number, and 2) the menu item name or number.

The sample program fragment below disables the **US Mail** and **UPS Ground** items in the **Shipper** menu when the **Rush** option is turned on.



These shipping options are re-enabled when the **Rush** option is turned off.

```
if info("trigger") = "Menu.Order.Rush"
   getmenumark "Order" "Rush"
   if clipboard() = ""
      setmenumark "Order","Rush",chr(18)
      menudisable "Shipper","US Mail"
      menudisable "Shipper","UPS Ground"
   else
      setmenumark "Order","Rush",""
      menuenable "Shipper","US Mail"
      menuenable "Shipper","UPS Ground"
   endif
endif
```

(Note: If a custom menu contains over 32 items, only the first 32 items can be disabled.)

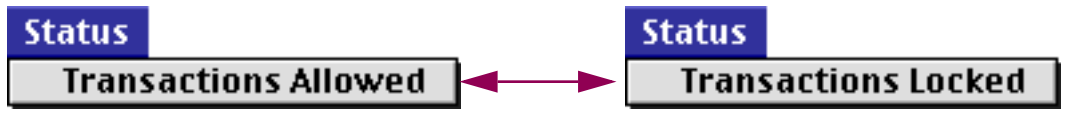
Changing Menu Text on the Fly

Panorama allows custom menu items to change to show changing conditions. For example, a menu item could be changed between **Locked**/**UnLocked**, **Inches**/**Centimeters**, or **Public**/**Private**.

To change a menu item use the `SetMenuText` statement. This statement has three parameters: 1) the menu name or ID number, 2) the menu item name (if known) or number, and 3) the new menu item text.

To find out what the current text of a menu item is, use the `GetMenuText` statement. This command has two parameters: 1) the menu name or ID number, and 2) the menu item number. (Notice that you must specify the menu item by number. This is because, by definition, you do not know what the menu item text is or you wouldn't be using this command!) The `GetMenuText` statement will put a copy of the menu item text into the clipboard, where it can be retrieved with the `clipboard()` function.

The code sample below assumes that the database has a custom menu named **Status**, and that the first item in this menu is either **Transactions Allowed** or **Transactions Locked**. This code sample starts by using the `GetMenuText` statement to find out the current state of the menu item.



Then it uses the `SetMenuText` statement to toggle the text in the menu to the opposite condition.

```
if info("trigger") matches "Menu.Status.Transactions*"
   getmenutext "Status" 1
   if clipboard() contains "locked"
      setmenutext "Status",1,"Transactions Allowed"
   else
      setmenutext "Status",1,"Transactions Locked"
   endif
   stop
endif
```

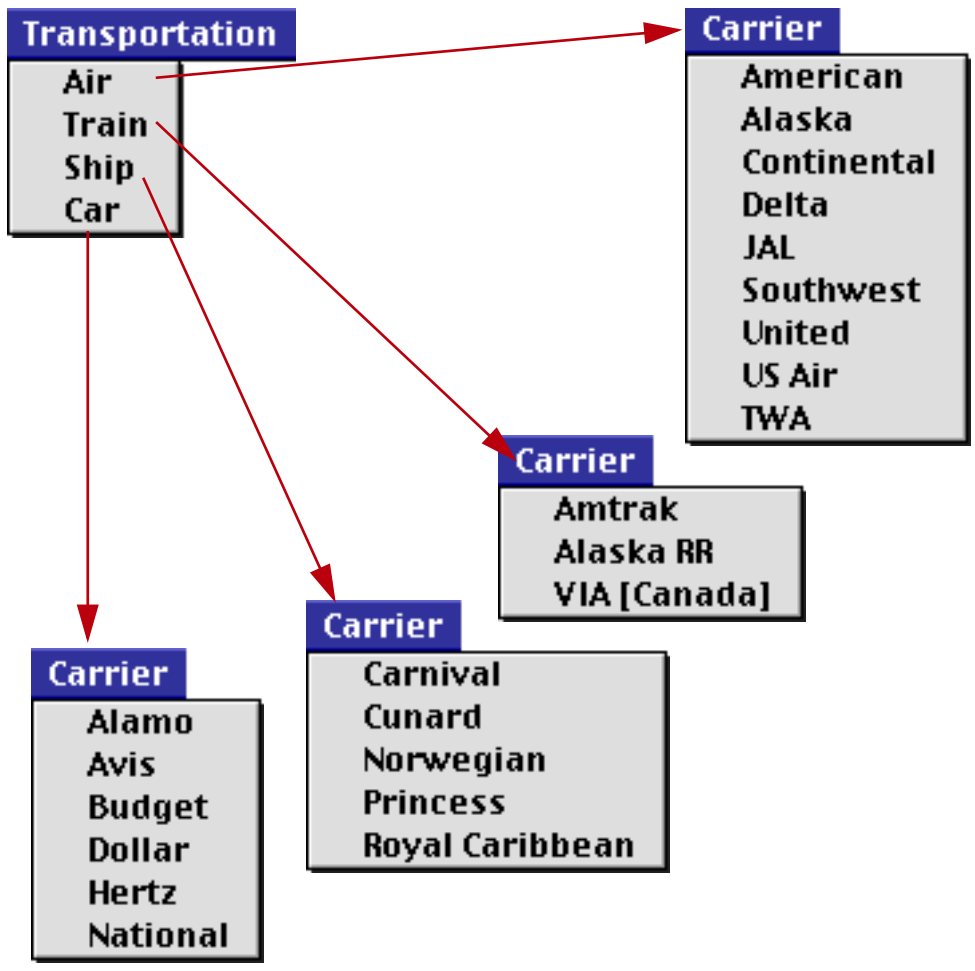
You should be very careful to avoid a confusing interface when using this menu changing technique. In the example above, it may not be clear to the user what this menu item does or shows. A better solution might be to have two menu items and mark one or the other with a checkbox. A previous section in this chapter, shows a more suitable use for changing menu items (see "Groups with Other…" on page 1499).

Rebuilding Entire Menus

The `menubuild` statement allows a programmer to completely rebuild a menu on the fly. This allows menu choices to be tailored to changing situations. The key word here is rebuild, because this command cannot build a menu from scratch. The custom menu must already exist in an open resource file, and it must be assigned to the current form or data sheet with the **Custom Menu** dialog.

The `menubuild` statement requires two parameters: 1) the menu name or ID number, and 2) a list of the new menu items, separated by semicolons.

The example below shows the `menubuild` statement in action. The database associated with this example has two custom menus, **Transportation** and **Carrier**. The Transportation menu has four items: **Air**, **Train**, **Ship** and **Car**. When the user selects one of these four items, the procedure changes the **Carrier** menu to show a list of carriers for that transportation mode. For example, if the user selects **Air**, the **Carrier** menu will be filled with a list of airlines. If the user selects **Train**, the **Carrier** menu will be filled with a list of railroads, etc.



Here's the procedure.

```
case info("trigger") = "Menu.Transportation.Air"
   menubuild "Carrier","American;Continental;Delta;Southwest;United;US Air;"
case info("trigger") = "Menu.Transportation.Train"
   menubuild "Carrier","Amtrak;Alaska RR;VIA [Canada]"
case info("trigger") = "Menu.Transportation.Ship"
   menubuild "Carrier","Carnival;Cunard;Norwegian;Princess;Royal Caribbean"
case info("trigger") = "Menu.Transportation.Car"
   menubuild "Carrier","Alamo;Avis;Budget;Dollar;Hertz;National"
endcase
```

Note: The `menubuild` statement normally builds all menu items as plain 12 point text in the default menu font. By adding a special suffix to menu item names in the menu item list you can change menu items to different styles: bold, italic, etc. The table below lists the different styles and corresponding suffixes.

Style	Suffix	Example
Bold	<B	Monthly Report<B
Italic	<I	New Invoice<I
Underline	<U	Initialize Payroll<U
Outline	<O	Back Order<O
Shadow	<S	Erase All Statements<S

It is also possible to assign a command key equivalent (shortcut) by adding a suffix. The suffix consists of a `/` character followed by the character you want to assign as a command key equivalent. The example below fills the **Ship** menu with five items that have command key equivalents from 1 thru 5.

```
menubuild "Ship","UPS/1;US Mail/2;FedEx/3;DHL/4;Airborne/5"
```

You can build separator lines in the custom menu by defining a menu item as `(-`. Unlike **Action** menus, all separator lines can be defined `(-`, duplicates are no problem. The example below would build a menu with two separator lines, one between **White** and **Red**, and another between **Blue** and **Orange**.

```
menubuild "Colors","Black;White;(-;Red;Green;Blue;(-;Orange;Violet;Yellow;Cyan;Brown"
```

Reassigning Menus in the Menu Bar

Normally custom menus are assigned to the menu bar with the **Custom Menu** dialog (in the Setup menu). The `setmenus` statement allows this same function to be performed by the programmer, allowing menus to be added, removed, or re-arranged in the menu bar at any time under program control. The `setmenus` statement requires one parameter, a list of the menu ID numbers you want to include in the menu bar, exactly as they would be typed into the **Custom Menu** dialog.

The `getmenus` statement allows a program to retrieve and examine the current menu bar configuration. This command places a copy of the menu ID number list into the clipboard, where it can be retrieved with the `clipboard()` function.

The example below saves the current menu configuration in the global variable fullMenus, then installs a minimum menu configuration of just the **Apple**, **File**, and **Setup** menus (S18 is the **Arrange** sub-menu).

```
global fullMenus
getmenus
fullMenus=clipboard()
setmenus "1 27 S18 68"
```

At a later time another procedure could restore the full custom menu configuration by simply including the line below:

```
setmenus fullMenus
```

Custom Menu Troubleshooting

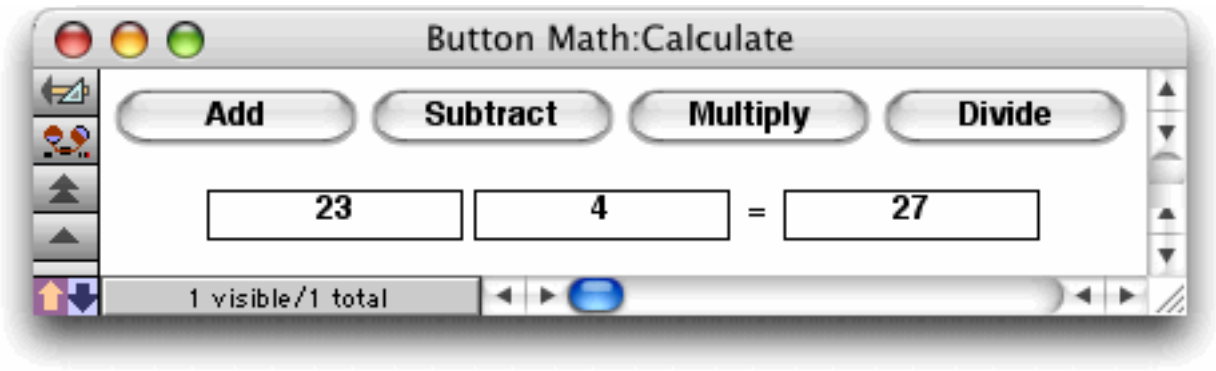
If you encounter problems while setting up custom menus, consult the troubleshooting guide below.

Problem	Solutions
No menus appear at all.	The most likely source of this problem is that the resource file is not open. To correct this problem make sure that the **.Initialize** procedure contains an `openresource` statement, that the resource file name matches the name specified by the `openresource` statement, and that the resource file is in the same folder as the Panorama database file (or the correct folder if a path has been specified by the `openresource` statement). If the resource file has been opened correctly, the menus may not be set up correctly in the resource file. Check the menus with the Custom Menu Editor (see "Preparing a Resource File" on page 1479), and make sure that the menu ID numbers match the numbers that have been set up in the **Custom Menu** dialog.
Only standard Panorama menus appear.	The most likely source of this problem is that the resource file is not open. See the previous section.
Custom menus appear and can be pulled down, but nothing happens when you select any custom menu item.	This problem could be caused if there is no **.CustomMenu** procedure, or if the **.CustomMenu** procedure name is spelled incorrectly. If you think you have set up the **.CustomMenu** procedure correctly but none of your menu items work, insert the following line at the top of the **.CustomMenu** procedure: `message info("trigger") stop` After this line is inserted, if you select any custom menu item a dialog should appear. If the dialog does not appear, check the name of the **.CustomMenu** procedure—it must be spelled exactly including upper and lower case. If the dialog does appear then the problem is in the logic of your **.CustomMenu** procedure.
A specific custom menu item does not work.	This indicates a problem in the logic of your **.CustomMenu** procedure—possibly an incorrect comparison with the `info("trigger")` function. Use the debugger to check the logic of your procedure (see "The Panorama Interactive Debugger" on page 1434).

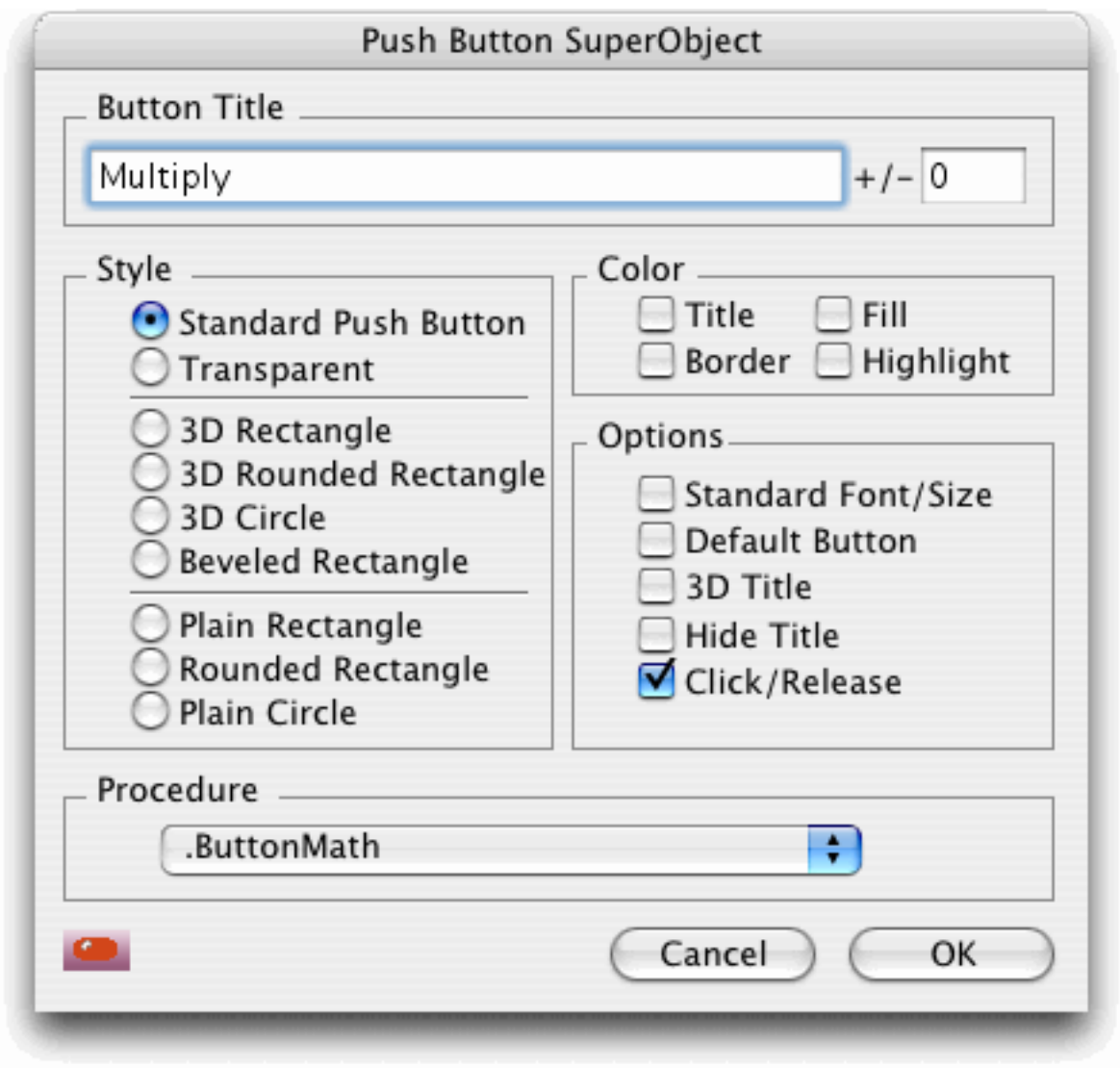
Buttons

Buttons are an important part of the today's modern graphic user interfaces. You can use a wide variety of buttons in any Panorama form. Panorama buttons come in three basic varieties: push buttons, data buttons (checkboxes and radio buttons), and pop-up menu buttons. All of these types of buttons can trigger a procedure. Use the configuration dialog for the button to select which button will be triggered when the button is pressed (see "Buttons & Widgets" on page 813).

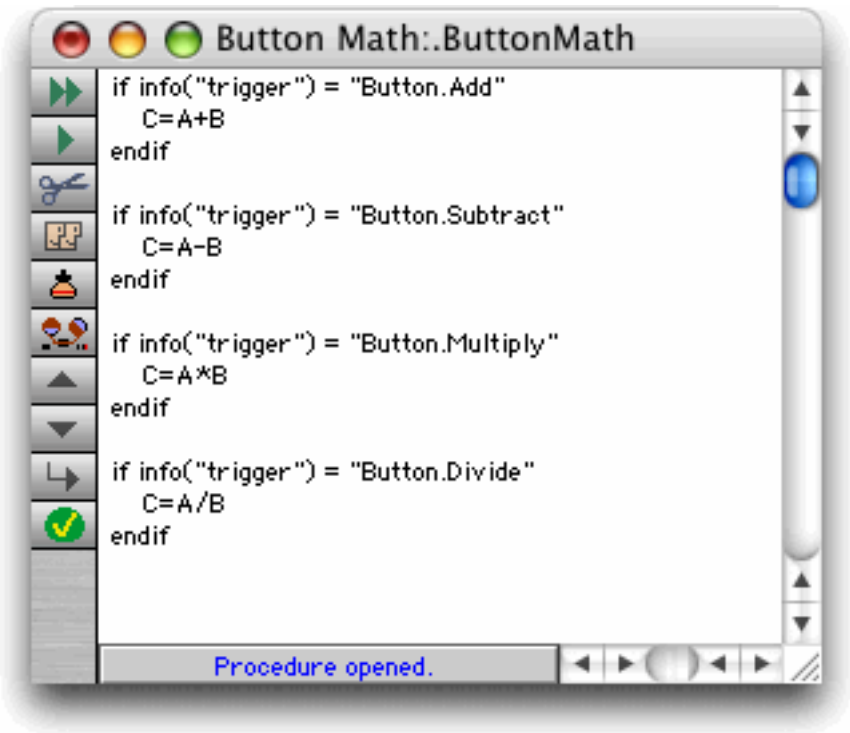
When a button is triggered by a procedure the `info( trigger )` function will return the title of the button. If you wish you may use a single procedure with many different buttons. For example, consider this form, which has four different buttons.



All four of these buttons trigger the **.ButtonMath** procedure. In fact, the only difference between these buttons is their titles. Here is the configuration dialog for one of these buttons.



The **.ButtonMath** procedure uses the `info("trigger")` function to decide which button was pressed.



Notice that the name of this procedure starts with a period. This makes this an “unlisted” procedure that does not appear in the **Action** menu (see See ““Unlisted” Procedures” on page 1464). It wouldn’t make any sense to trigger this procedure from the menu, so it’s best to make it unlisted.

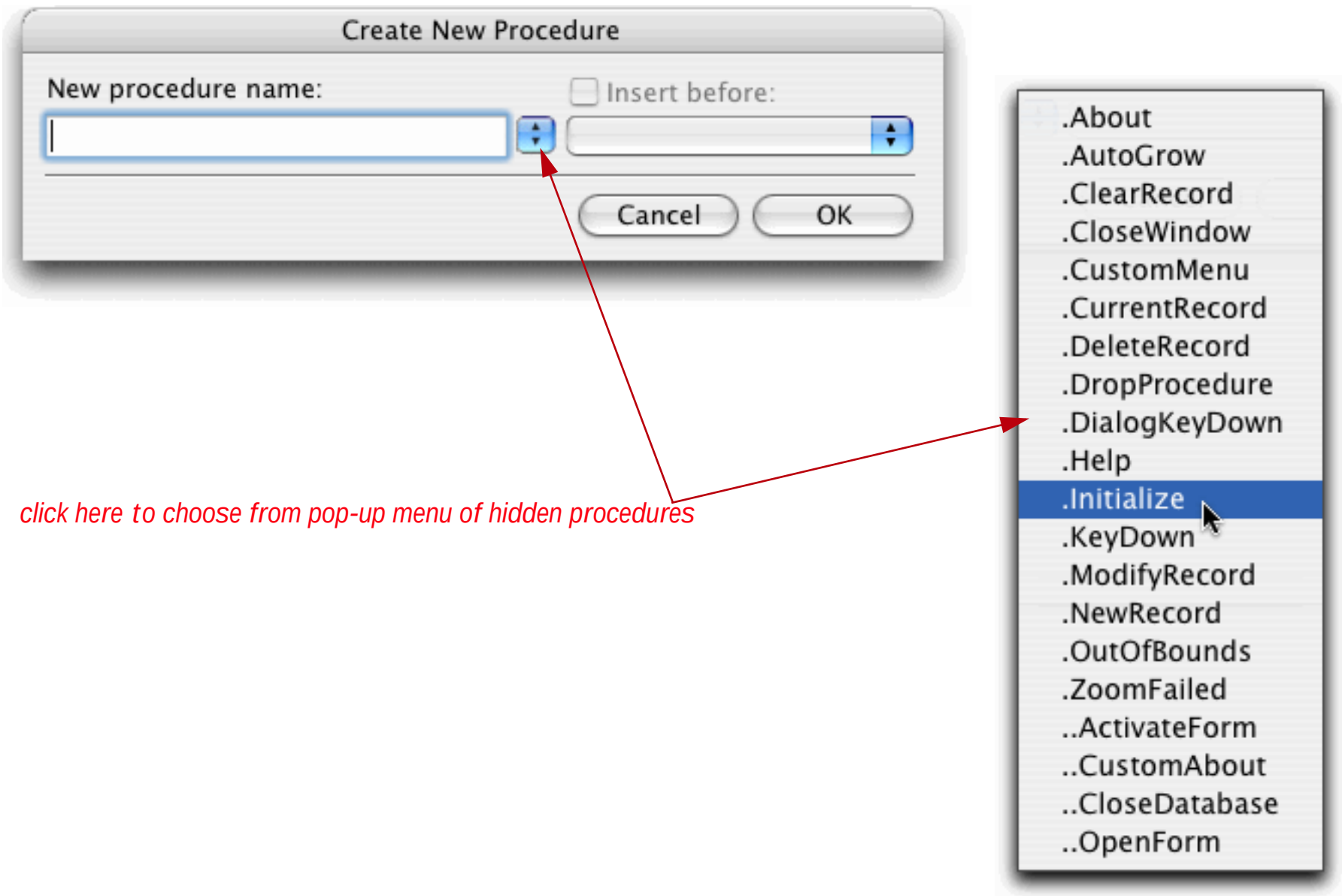
Hidden Triggers

Hidden triggers activate a procedure automatically when the user performs some normal Panorama action. Examples of user actions that can cause a hidden trigger to activate include adding new records to a database, deleting records, opening a file, closing a window and many more. The trigger is “hidden” because the user is not explicitly asking Panorama to activate a procedure by pressing a button or selecting a menu choice.

Procedures that are activated by hidden triggers can modify (or even override) the way Panorama reacts to many standard user actions. For example, when a user clicks on a window’s close box, Panorama normally responds by closing the window. But with a hidden trigger the programmer can activate a procedure whenever the close box is clicked. This procedure can do anything the programmer wants. For example, the programmer may want to save the window position before the window is closed. Or the programmer may not want to let the user even close the window until all the data on a form is filled in. Of course this kind of flexibility comes with a price. The user expects the window to close—so any other action must be carefully designed so that it doesn’t confuse or frustrate the user.

Creating Hidden Trigger Procedures

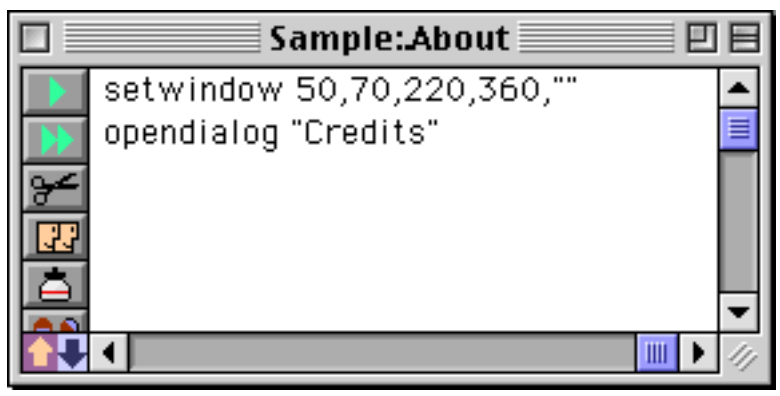
To create a procedure that is activated by a hidden trigger you must give the procedure a special name. For example, suppose you want to create a procedure that is triggered whenever a window is closed. That procedure must be named **.CloseWindow**. If a database contains a procedure with that name, it will always be triggered when the user clicks on the close box of a window in that database. To make it easier to create hidden trigger procedures, the **New Procedure** dialog contains a pop-up menu of the special procedure names required for hidden triggers.



Each of these possible hidden trigger procedures is explained in the following sections.

.About

This hidden trigger procedure will be triggered when the user selects **About Panorama** from the Apple menu. Normally selecting this menu item displays the “splash screen” and copyright message for Panorama. Using the .About hidden trigger procedure, you can display your own splash screen with information about your database. The following example shows a typical **.About** hidden trigger procedure that opens a form called Credits. This form would normally include a picture or logo for your database. It should also include a button that allows the user to close the window and resume normal use of the database (or you could even flip through multiple credit pages before closing the window).

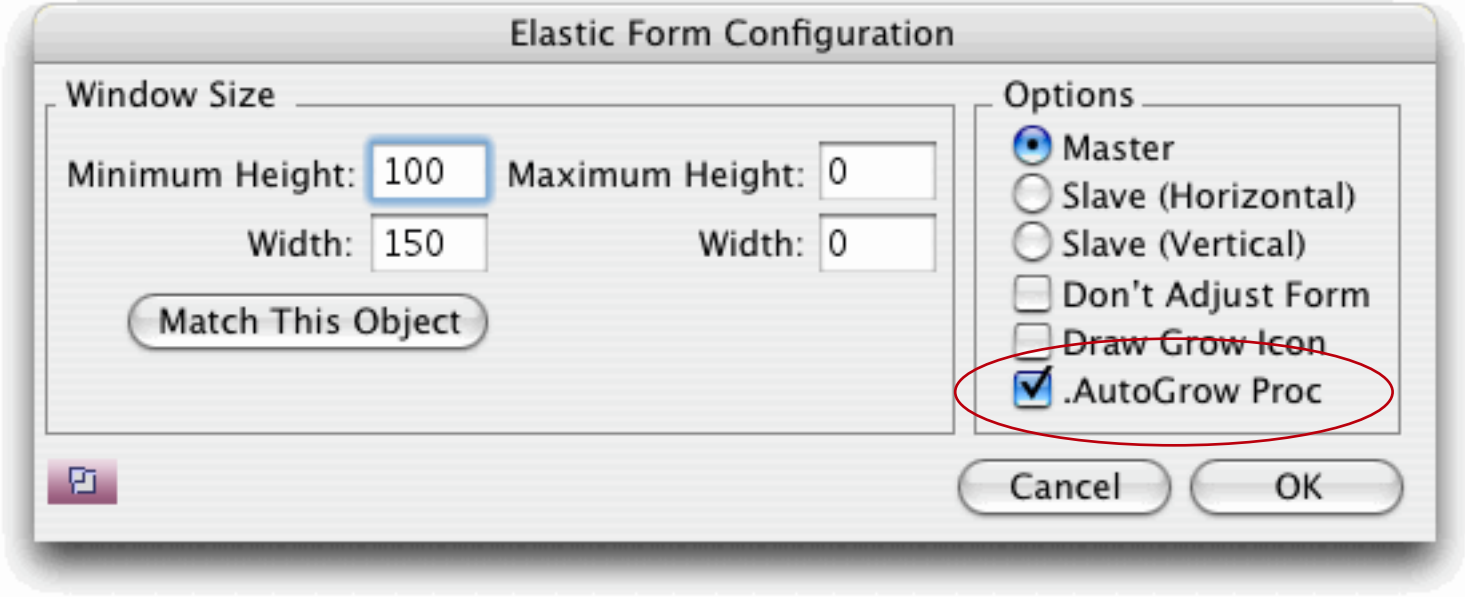


Warning: Your revised credit screen **must** include the ProVUE copyright notice in its entirety.

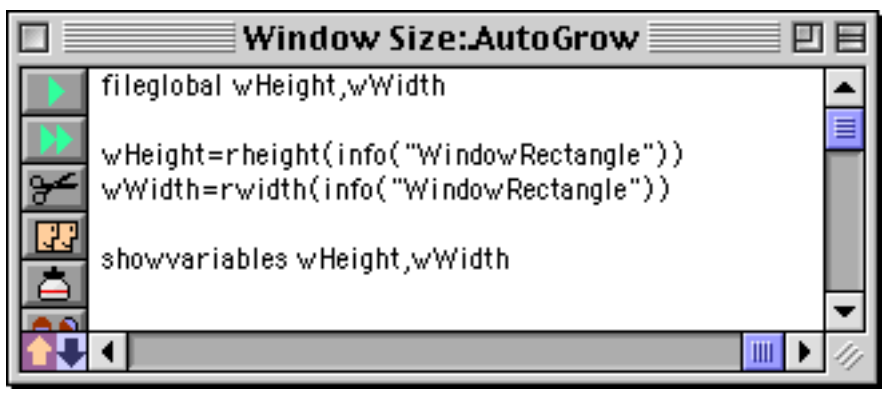
.Note: You can also change the name of the About Panorama menu item. See “..CustomAbout” on page 1525.

AutoGrow

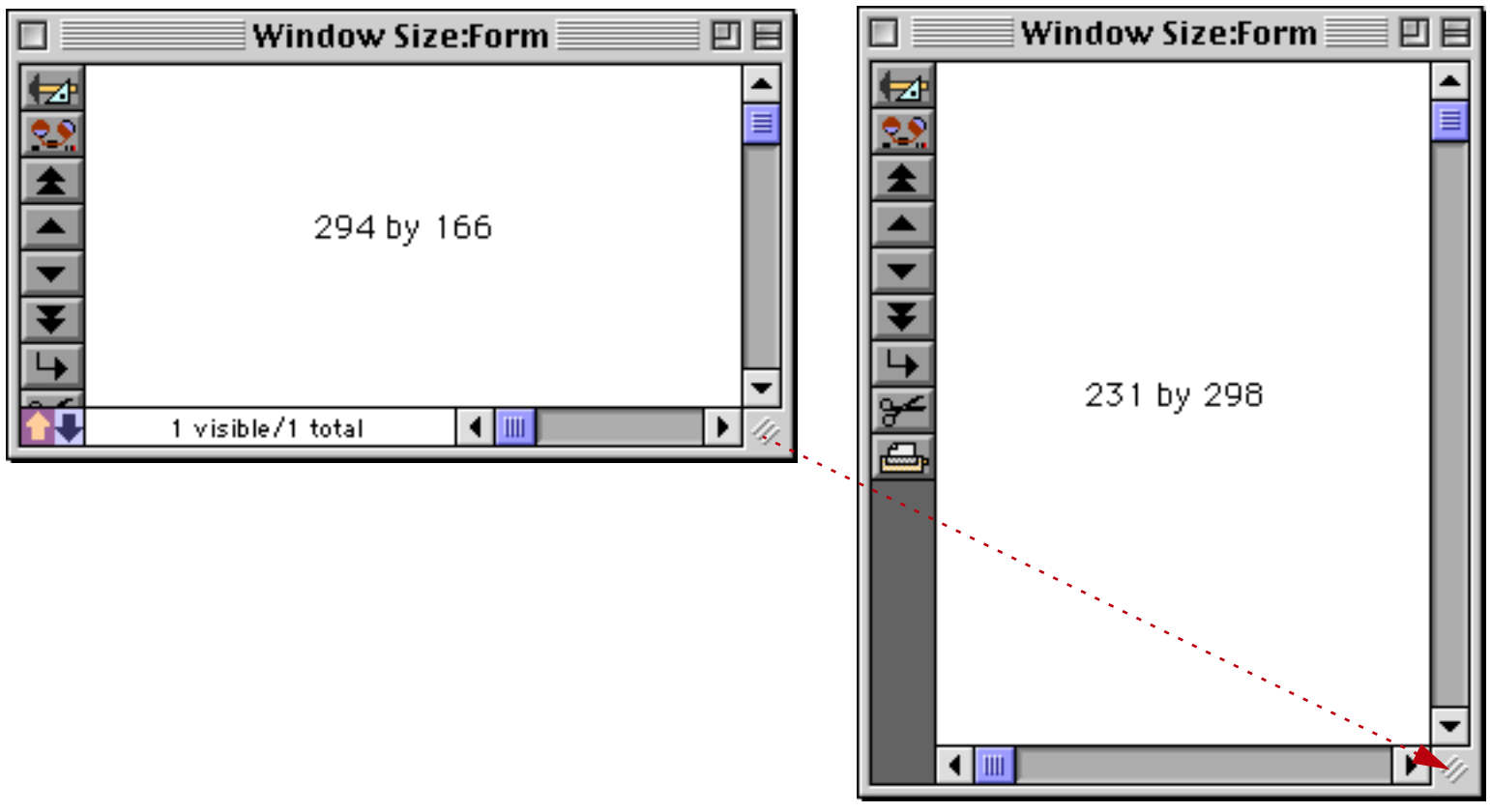
The **.AutoGrow** procedure is designed to work with Elastic forms (see “Elastic Forms” on page 912). To use this procedure you must enabled the option in the auto-grow objects, like this.



When this option is enabled the **.AutoGrow** procedure will be triggered every time the window changes size. Here is an **.AutoGrow** procedure that simply calculates the width and height of the window.



In this database the **.AutoGrow** procedure is simply used to display the size of the new window.



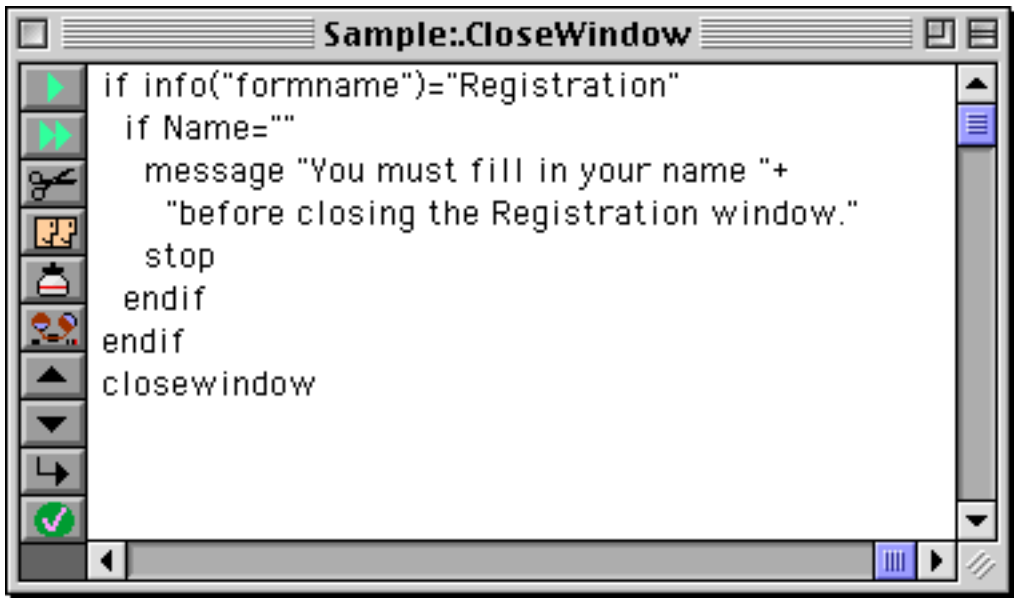
A more useful application would be to adjust elements of the form depending on the size of the window. See “Programming Graphic Objects on the Fly” on page 1753 to learn how to adjust form elements.

.ClearRecord

This procedure is triggered when you choose the **Clear** menu item from the Edit menu, but only in a crosstab. Frankly, we can’t remember why this feature was added!

.CloseWindow

This hidden trigger procedure will be triggered when the user clicks on the close box in the upper left hand corner of a window. Usually clicking on this box causes the window to close. Using the .CloseWindow hidden trigger procedure you can perform extra steps before closing the window, or even prevent the window from closing. Here is a typical example of how the .CloseWindow hidden trigger procedure is used. If the Registration form is open, the procedure checks to make sure that the Name field is not empty. If the Name field is empty, the procedure tells the user that they cannot close the form yet. Otherwise the procedure goes ahead and closes the window.



Notice that the last statement in this procedure, `closewindow`, actually closes the window. It does not trigger the procedure again. Only a user action, such as clicking or pressing a key, can trigger a hidden procedure.

Warning: The .CloseWindow procedure is triggered only by the user clicking on the close box of the window. It is not triggered by other actions that might close the window, such as closing the entire file or quitting from Panorama.

.CurrentRecord

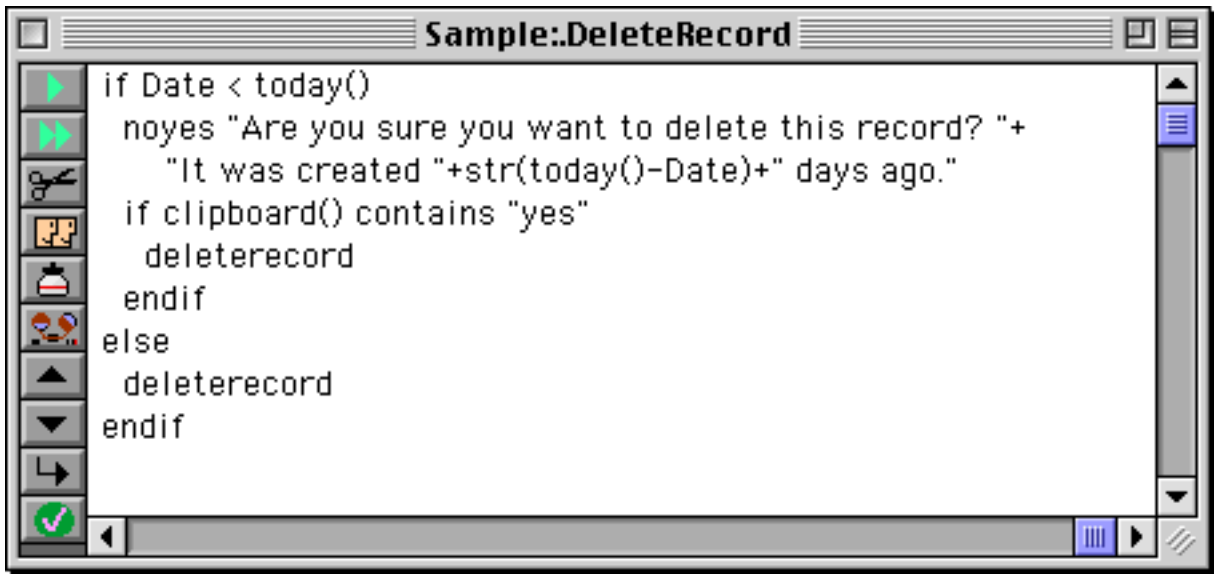
This hidden trigger procedure will be triggered when the database shifts to a different record. For example, this procedure is triggered when you move up or down in the database with the vertical scroll bar, or with the **Find** or **Find Next** commands. (Remember, like other hidden trigger procedures it is not triggered by procedure statements, only by user actions.) You can use this procedure to perform any special actions that are necessary to display or work with this record.

.CustomMenu

This hidden trigger procedure will be triggered when the user selects a Custom menu item. For a complete description of custom menus and the **.CustomMenu** hidden trigger procedure see “The .CustomMenu Procedure” on page 1466.

.DeleteRecord

This hidden trigger procedure will be triggered when the user attempts to delete a record from the database. This procedure could be triggered by the **Delete Record** tool, or by pressing the **Delete** key in data sheet or view-as-list windows. The example below allows records created today to be deleted immediately, but double checks before allowing older records to be deleted.



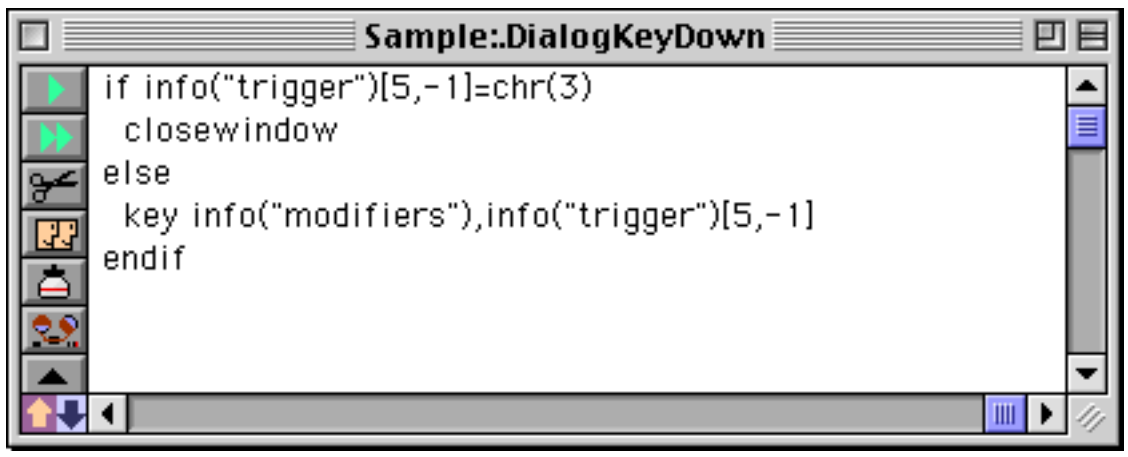
Notice that the record is not deleted unless the procedure deletes the record. The .DeleteRecord procedure interrupts the normal deletion process and takes over. This puts you, the programmer, in control.

.DialogKeyDown

This very specialized hidden trigger procedure will be triggered when the user presses a key in a form that has no drag bar (a form that looks like a dialog). However, this procedure will not be triggered if a Data Cell or SuperObject Text Editor is currently active. So if you are in a dialog created with a Panorama form, not editing text, and press a key, the .DialogKeyDown procedure will be triggered. (Another way to intercept keystrokes is with a hotkey procedure, see “Hot Key Procedures” on page 1518.)

The procedure can tell what key was pressed by using the `info("trigger")` function. For example, if the user presses **Y** the `info("trigger")` functions will return Key.Y.

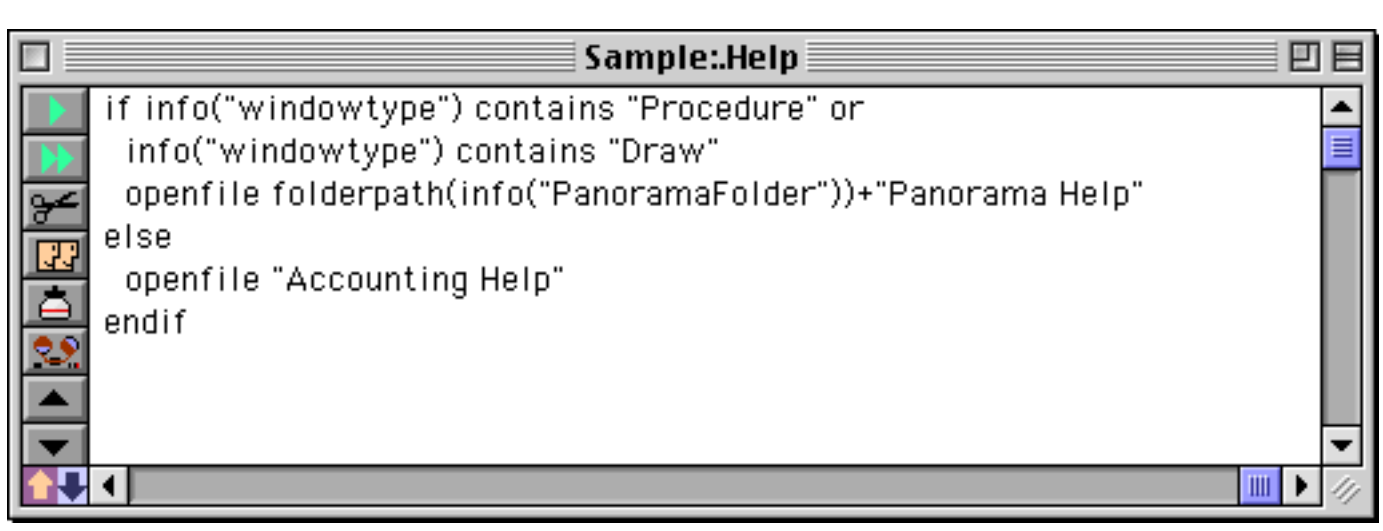
Here is a .DialogKeyDown procedure that closes the dialog window if the user presses the **Enter** key. All other keystrokes will be processed normally.



To process keystrokes normally this procedure uses the key statement. See “.KeyDown” on page 1511 for more information on this statement.

.Help

This hidden trigger procedure will be triggered when the user selects **Help** from the Apple menu. Normally selecting this menu item opens the Panorama help system. Using this hidden trigger procedure you can force Panorama to use your own custom help system for your database. The example shown below will open the normal Panorama Help if the current window is a procedure or a form in graphics mode. Otherwise it will open the special Accounting Help database.



.Initialize

This hidden trigger procedure will be triggered when the database is opened, either from the desktop or from the **Open** dialog. You can use this procedure to initialize global and permanent variables, open resource files, set up custom menus, pre-sort or pre-select the database…anything that needs to be done automatically whenever the database file is opened.

Warning: Most procedures can only be triggered from the data sheet or a form. However, the .Initialize procedure will start running immediately when the file is opened, in whatever window happens to be open. If this window is not a data sheet or form, the procedure may not operate correctly. Many procedure statements (sort, group, select, etc.) will not operate properly from a non-data window. If this may happen, the first thing the .Initialize procedure should do is open a form or the data sheet.

.KeyDown

This very specialized hidden trigger procedure will be triggered when the user presses a key in a form. However, this procedure will not be triggered if a Data Cell or SuperObject Text Editor is currently active. So if you are in a form, not editing text, and press a key, the **.KeyDown** procedure will be triggered. (Another way to intercept keystrokes is with a hotkey procedure, see “Hot Key Procedures” on page 1518.)

The procedure can tell what key was pressed by using the `info("trigger")` function. For example, if the user presses Y the info("trigger") functions will return Key.Y. Special keys will return the special values listed in this table:

Tab	tab	chr(9)
Enter	enter	chr(3)
Return	return	chr(13)
Esc	esc	chr(27)
Delete	delete	chr(8)
Left Arrow	←	chr(28)
Right Arrow	→	chr(29)
Up Arrow	↑	chr(31)
Down Arrow	↓	chr(30)

The procedure can use the `info("modifier")` function to tell what modifier keys have been pressed along with the primary key: shift, option, control, and command.

Here is a **.KeyDown** procedure that closes the window if the user presses the **Enter** key. If the user presses the **$** key Panorama jumps to the Amount data cell and begins to edit it. All other keystrokes will be processed normally.

```
local KeyStroke
KeyStroke=info("trigger")[5,-1]
case KeyStroke=chr(3)/* enter key */
   closewindow
case KeyStroke="$"
   field Amount
   editcellstop
defaultcase
   key info("modifiers"),KeyStroke
endcase
```

To process keystrokes normally this procedure uses the `key` statement. The `key` statement passes the keystroke back to Panorama for normal processing (see “KEY” on page 5461). This statement has two parameters: 1) the modifiers associated with the key, and 2) the key itself.

.ModifyRecord

This hidden trigger procedure will be triggered when the user modifies any field in the database. Warning: The **.ModifyRecord** procedure will not run if a procedure is already running, or if the field has its own procedure. In those cases you may want the other procedure to call the **.ModifyRecord** procedure as a subroutine (more on this in a moment). The **.ModifyRecord** procedure is also not called if the data is modified with a command in the Fill menu.

The **.ModifyRecord** procedure example below automatically marks the latest date and time when a record was modified. This example assumes that the database has two fields for time / date tracking: ModifyDate (a date field) and ModifyTime (a numeric field).

```
ModifyDate=today()
ModifyTime=now()
```

Note: This example illustrates the **.ModifyRecord** procedure, but a better way to perform this task would be to create a Time Stamp field in the design sheet. See “Automatic Time/Date Stamping” on page 338 for details on this process.

If your database has other procedures that modify the database they should call the **.ModifyRecord** procedure to make sure that the time stamp is kept up to date. For example, here is a procedure that automatically subtracts one from the QtyInStock field.

```
QtyInStock=QtyInStock-1
call .ModifyRecord
```

The example below selects all items that have over 500 in stock and have not been touched in 30 days. For those items, it reduces the price by 10%, then marks the modification date and time.

```
select QtyInStock>500 and today()-ModifyDate>30
field Price formulafill Price*0.90
field ModifyDate formulafill today()
field ModifyTime formulafill now()
```

Since the **.ModifyRecord** procedure is not triggered by `FormulaFill`, the procedure must update the modification date and time itself.

.NewRecord

This hidden trigger procedure will be triggered when the user attempts to add a new record to the database with the **Add New Record** or **Insert New Record** tool, or by pressing the **Return** key in the data sheet or a view-as-list form (see “Adding a New Record” on page 306). The `info("trigger")` function can be used to determine which of the three possible actions triggered the procedure:

"New.Add"	Add New Record tool or Add New Record menu item
"New.Insert"	Insert New Record tool
"New.Return"	Return key

This sample **.NewRecord** procedure won’t allow new records to be added if there already at 100 or more records in the database:

```
if info("records")≥100
   message "Sorry, this database is limited to 100 records."
else
   case info("trigger") = "New.Add"
      addrecord
   case info("trigger") = "New.Insert"
      insertrecord
   case info("trigger") = "New.Return"
      insertbelow
   endcase
endif
```

Here's another **.NewRecord** example that only allows new records to be added to the end of the database, not inserted in the middle. In this example the case for `info("trigger") = "New.Insert"` has been eliminated, because we know that a new record cannot be inserted. The case for `info("trigger") = "New.Return"` has been expanded to also check to see if we are on the last line of the database with the `info("eof")` function.

```
local AddFlag
AddFlag="no"
case info("trigger") = "New.Add
   addrecord
   AddFlag="yes"
case info("trigger") = "New.Return" and info("eof")
   insertbelow
   AddFlag="yes"
endcase
if AddFlag="no"
   message "Sorry, new records must be added"+
     " to the end of the database, not inserted in the middle."
   stop
else
   call .ModifyRecord
endif
```

At the end of this example procedure a message is displayed if the record could not be added. If the new record has been added this procedure calls **.ModifyRecord** (see ".ModifyRecord" on page 1512). The procedure could also calculate default values at this point.

Warning: The **.NewRecord** procedure is not triggered when new records are added by appending (with the **Open File** command).

.OutOfBounds

This very specialized hidden trigger procedure will be triggered when a form that has no drag bar is open (a form that looks like a dialog) and the user clicks outside of the window. In other words, if you create a dialog with a Panorama form and the user clicks outside of the dialog, this procedure will be triggered. (If there is no **.OutOfBounds** procedure, Panorama will simply beep when this happens.)

If you wish, the **.OutOfBounds** procedure could simply close the window if the user clicks outside of it:

```
closewindow
```

Or, the .OutOfBounds procedure could display a message:

```
message "Please click inside the dialog."
```

Of course, this message might make the user feel like they were back in kindergarten (please draw inside the lines!).

.ZoomFailed

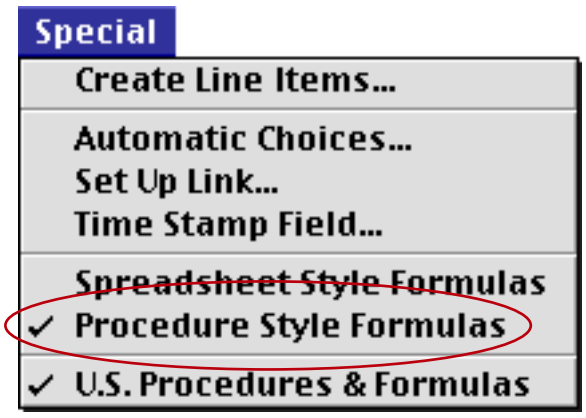
This very specialized hidden trigger procedure will be triggered when the user clicks on the zoom box, but Panorama cannot zoom the window. The only time this happens is if both the horizontal and vertical scroll bars are disabled. In that case you can use the **.ZoomFailed** procedure to make the window zoom, perhaps by switching to a different form as the example below shows:

```
case info("formname")="Letter
   goform "Full Letter"
   setwindow 20,2,760,500,""
   zoomwindow
case info("formname")="Full Letter"
   goform "Letter"
   setwindow 20,2,320,500,""
   zoomwindow
endcase
```

Data Entry Triggers

Panorama can automatically trigger a procedure whenever you enter new data into a database field. Unlike other hidden trigger procedures, data entry trigger procedures do not have a special name. Instead, the name of the procedure is specified in the design sheet.

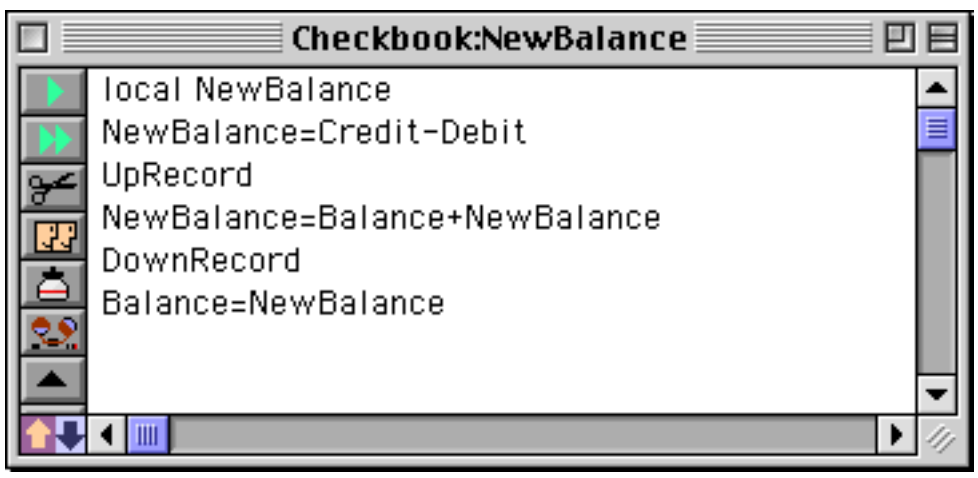
To set up a procedure that is triggered by data entry, first create the procedure. If you don't want the procedure to be listed in the **Action** menu the procedure name should start with a period. The procedure name must be a single word with no spaces. Then open the design sheet and make sure that the **Procedure Style Formulas** option is checked in the **Special** menu. If you have been using **Spreadsheet Style** formulas you will need to adjust the other formulas that have been set up in the design sheet (see "Automatic Calculations" on page 340).



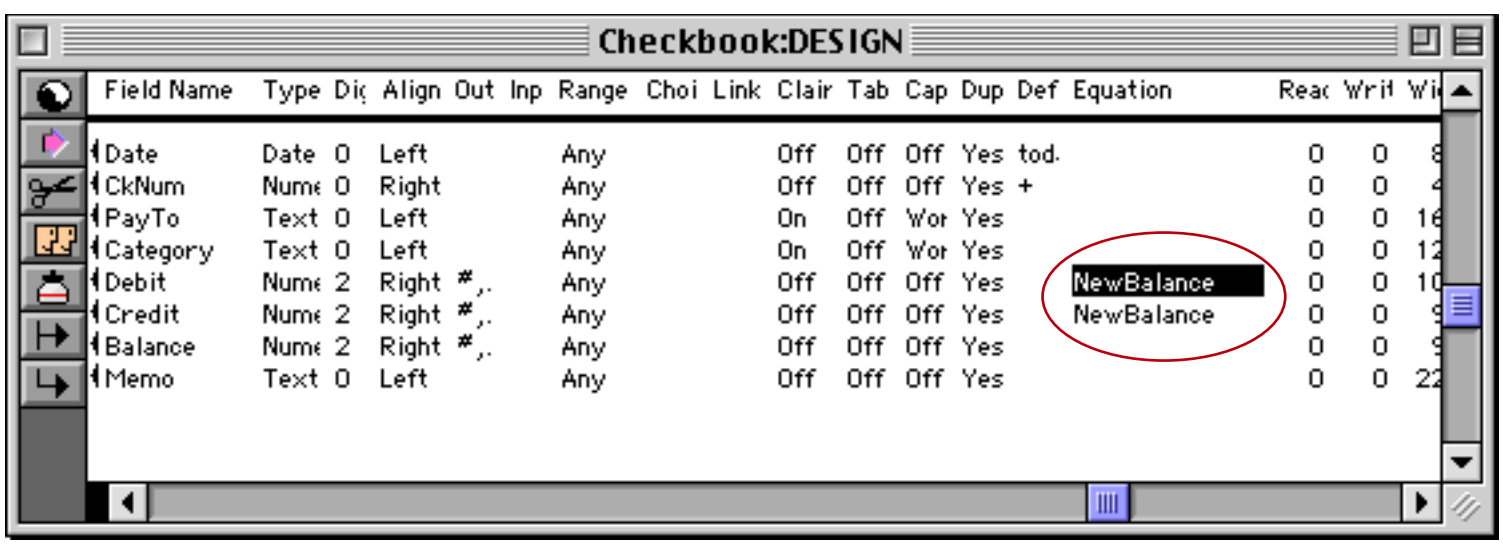
Then type the procedure name into the Equation column for the field. If there are already formulas for the field, the procedure name should be typed after the last formula.

Once the data entry trigger is set up, the procedure will be triggered automatically each time the user presses **Enter**, **Return** or **Tab** to enter data. The procedure can find out which key was pressed using the `info("trigger")` function. If the **Return** key was pressed, `info("trigger")` will be Key.Return. If the **Tab** key was pressed, `info("trigger")` will be Key.Tab.

Here is a sample data entry triggered procedure that calculates the new balance for a checkbook. The data sheet should be set up so that this procedure is triggered whenever the Credit or Debit fields are modified.



To set this procedure up to be triggered you must enter it into the Equation field of the design sheet, like this.



When the user presses the **Tab** key while editing data, Panorama normally skips to the next field. However, if the procedure uses the field statement to switch to a different field, Panorama's normal tab order is aborted. If you want to abort the tab order without moving to a different field, use the `stoptab` statement. Here is a sample data entry triggered procedure that filters out negative values in the Price field.

```
if Price < 0
   message "Negative prices are not allowed"
   stoptab
endif
```

If a data entry triggered procedure is triggered, the **.ModifyRecord** procedure (if any) for the database will not be triggered. If necessary, you should call to **.ModifyRecord** somewhere in your data entry triggered procedure. Here's a revised version of our check balance procedure.

```
local NewBalance
NewBalance=Credit-Debit
UpRecord
NewBalance=Balance+NewBalance
DownRecord
Balance=NewBalance
call .ModifyRecord
```

Another option is to get rid of the data entry triggered procedures completely and do all the work in the **.ModifyRecord** procedure. That option is described in the next section.

Data Entry Triggers (Part Two)

Instead of using separate data entry trigger procedures for each field as described in the last section, you combine all of the data entry procedures for the entire database into a single procedure: the **.ModifyRecord** procedure. Remember, this procedure is triggered whenever any field is modified. This procedure can use the `info("fieldname")` function to determine which field was modified, and take appropriate action. Here's an example of a **.ModifyRecord** procedure that performs special actions for the Date, Credit, and Debit fields in a checkbook database.

```
case info("fieldname")="Date"
   if Date>today()
      Date=today()
      stoptab
      message "Post-dated checks not allowed!"
   endif
case info("fieldname")="Credit" or info("fieldname")="Debit"
   local NewBalance
   NewBalance=Credit-Debit
   UpRecord
   NewBalance=Balance+NewBalance
   DownRecord
   Balance=NewBalance
endcase
ModifyDate=today()
```

This procedure also time stamps the ModifyDate field in the current record whenever any field is modified. With this technique it's all in one procedure and easy to keep track of. Another advantage of this technique is that it will work just fine with the **Spreadsheet Style Formulas** option.

Hot Key Procedures

Panorama can be configured so that an action is triggered automatically when a particular key or key combination is pressed. Unless you want a special effect (for example a hotkey that only works in a single window) the easiest way to set that up is to use the Hotkey wizard, which is in the Preferences submenu of the Wizard menu. Using this wizard you can set up a hotkey (or multiple hotkeys) that triggers a procedure in your database. The wizard automatically manages the necessary permanent variables for you. See “Hotkey Manager” on page 58.

If you do need a special effect that is not supported by the Hotkey wizard you can set up hotkeys in a procedure. Every time a key is pressed Panorama looks for a special variable. This variable is named HotKey[xx], where xx is the hexadecimal value of the keycode for the key that is pressed (see table below). For example, to create a hotkey procedure for the **F1** key the variable must be named HotKey[7A] If Panorama finds such a value it takes the contents of the variable and executes them, just as if the variable was part of an `execute` statement (see “Building Subroutines On The Fly (The Execute Statement)” on page 1399). The table below lists the codes for 96 different keys (the keys in green are on the numeric keypad).

A	00	M	2E	Y	10	⌘	37	0	1D	2	54	*	43	F9	65
B	0B	N	2D	Z	06	space	31	-	1B	3	55	+	4E	F10	6D
C	08	O	1F	[	21	\	2A	=	18	4	56	-	45	F11	67
D	02	P	23	]	1E	1	12	delete	33	5	57	ESC	35	F12	6F
E	0E	Q	0C	;	29	2	13	`	32	6	58	F1	7A	F13	69
F	03	R	0F	‘	27	3	14	return	24	7	59	F2	78	F14	6B
G	05	S	01	,	2B	4	15	↑	7E	8	5B	F3	63	F15	71
H	04	T	11	.	2F	5	17	↓	7D	9	5C	F4	76	home	73
I	22	U	20	/	2C	6	16	←	7B	enter	4C	F5	60	PgUp	74
J	26	V	09	shift	38	7	1A	→	7C	clear	47	F6	61	PgDn	79
K	28	W	0D	control	3B	8	1C	0	52	=	51	F7	62	END	77
L	25	X	07	option	3A	9	19	1	53	/	4B	F8	64	HELP	72

The procedure below sets up a hotkey procedure for the **F1** key. After this procedure has been used pressing the **F1** key will open the Favorite Databases wizard.

```
global «HotKey[7A]»
«HotKey[7A]»={
   openfile folderpath(info("panoramafolder"))+"Wizards:Favorite Databases"
}
```

Because this hotkey procedure was set up as a global variable it will be active no matter what database is currently open. If you want to restrict the hotkey procedure to a particular database you should use a fileglobal variable. If you want to restrict the hotkey procedure to a particular window you should use a windowglobal variable.

If you define a hotkey procedure with a global or fileglobal variable keep in mind that the procedure may be triggered at **any time** in **any window**, so you must not make any assumptions about what window or even what type of window will be active when the procedure is triggered. It could be a procedure window, crosstab window, input window, you name it. Also, if you used a global variable you cannot even assume that the original database is still open.

Hotkey procedures do not work in Panorama dialogs. For example, pressing a hot key has no effect when you are using the **Find/Select** dialog, or the **Print** dialog, or any other dialog that is built into Panorama. However, hot keys do work in dialogs that you have created with a form (see “Custom Dialogs” on page 1616).

To disable a hotkey procedure you can either set the hotkey variable to "" or you can destroy the variable with the undefine statement (see “Destroying a Variable” on page 1372).

HotKeys with Modifiers

If you want to assign an action to a key with a specific modifier key combination, create a variable named HotKey[kkmm], where kk is the hexadecimal value of the keycode for the key that is pressed (see table above) and mm is sum of the hexadecimal values for each modifier key. The table below lists the hex values for the four modifiers keys. (Please keep in mind that these are hexadecimal, not decimal values. You can use the RPN Programmer's Calculator wizard to calculate the sum of two or more hexadecimal numbers.)

Hex	Macintosh	Windows
01	Command	Control
02	Shift	Shift
08	Option	Alt
10	Control	Right Click

For example, to create a hotkey procedure for **Shift-F1** key the variable must be named HotKey[7A02]. To create a hotkey procedure for **Command-Option-A** the variable must be named HotKey[7A09].

Universal HotKey Procedure

If Panorama does not find a hotkey for the specific key that has been pressed it will check to see if there is a variable named HotKey[*]. If there is, Panorama will execute the code it finds inside. Be very careful with this variable. For example, the procedure below will completely disable the keyboard until you **Quit** from Panorama. No matter what database you are in pressing the keyboard will no longer have any effect. Usually you will want to selectively apply a universal hotkey with a fileglobal or windowglobal variable.

```
global «HotKey[*]»
«HotKey[*]»="rtn"
```

The **.KeyDown** and **.DialogKeyDown** procedures provide another method to intercept keystrokes and process them yourself. See “.KeyDown” on page 1511 and “.DialogKeyDown” on page 1510.

Triggering a Procedure Every Second

Once every second Panorama checks for a special variable named ExecuteEverySecond. If it finds this variable, Panorama takes its contents and executes them, just as if the variable was part of an execute statement (see “Building Subroutines On The Fly (The Execute Statement)” on page 1399). The most common use for this feature is to create animation within a form. For example, you could use this feature to make an item on the form blink, or you could use it to update a stopwatch display every second.

The ExecuteEverySecond variable may be a global, fileglobal, or windowglobal variable. If it is a windowglobal variable, the procedure will only be executed when that window is the front window. If it is a fileglobal variable the procedure will only be executed when that file is the active file (one of its windows is on top). If it is a global variable the procedure will be executed no matter what database is active, even if the original database that created the variable is no longer open. It’s possible to have more than one ExecuteEverySecond variable active at a time, for example a windowglobal and a global. In that case Panorama will execute both of the procedures every second.

Here is an example that will cause a bullet to blink on and off once per second. The form Blink Demo should contain a Text Display SuperObject that displays the blinkValue variable, or a Flash Art object that uses this value as part of the formula. The `nowatchcursor` statement makes sure that the mouse arrow doesn't flip into a watch once per second as the procedure runs (see "Disabling the Watch Cursor" on page 1430).

```
openform "Blink Demo"
windowglobal ExecuteEverySecond,blinkValue
blinkValue="•"
ExecuteEverySecond={
   nowatchcursor
   if blinkValue="•"
      blinkValue=""
   else
      blinkValue="•"
   endif
   showvariables blinkValue
}
```

Because the ExecuteEverySecond variable was defined as a windowglobal variable the object will only blink when that window is the front window. When another window comes to the front the blinking will pause. It will resume when the Blink Demo window is brought to the front again.

Since this example uses a windowglobal variable the procedure can assume that the Blink Demo window is on top when the procedure executes each second. If you use a fileglobal or global variable there is no guarantee what window will be on top. If you need to access a specific database you should use the secret window feature to temporarily activate it during the procedure (see "Temporary "Invisible" Windows" on page 1581).

If you use a global variable you should be careful to be a good neighbor, since other databases may also be using the same variable. Instead of simply assigning the text of your procedure to the ExecuteEverySecond procedure you should append it. When you are done you should remove your code while leaving any other code. In addition, your procedure should not assume that the original database that activated it is still open — it may have been closed.

Panorama includes a Stopwatch wizard that can be used as a timer.



The three buttons in this form are tied to a single procedure which is listed below. When the **Start** button is pressed the procedure appends the code in StopwatchCode to the ExecuteEverySecond variable. This code will execute every second, causing the form to update as the timer runs. (Notice that the procedure also checks to make sure that the original database is still open using the `arraysearch(` and `info("files")` functions.) When the **Stop** button is pressed the procedure uses the `replace(` function to remove the code that it added without touching any other code that may have been added by other databases. In fact, you can make copies of this Stopwatch database and run them all at the same time. Each will keep it's own time, and they will all keep running no matter what database is currently active.

```
global ExecuteEverySecond
fileglobal ElapsedTime,CumulativeTime,StartTime
local StopwatchCode
define ElapsedTime,0
define ExecuteEverySecond,""

StopwatchCode=
{/* }+info("databasename")+{ Wizard */
nowatchcursor
if arraysearch(info("files"),}+"{"+info("databasename")+"}"+{,1,¶)<>0
   local wasWindow
   wasWindow=info("windowname")
```

```
   window }+"{"+info("databasename")+"}"+{+":SECRET"
   ElapsedTime=CumulativeTime+now()-StartTime
   showvariables ElapsedTime
   window wasWindow
endif
}

if info("trigger") contains "Reset"
   ElapsedTime=0
   showvariables ElapsedTime
   rtn
endif
if info("trigger") contains "Start"
   if ExecuteEverySecond notcontains StopwatchCode
      CumulativeTime=ElapsedTime
      StartTime=now()
      ExecuteEverySecond=ExecuteEverySecond+StopwatchCode
   endif
endif
if info("trigger") contains "Stop"
   ExecuteEverySecond=replace(ExecuteEverySecond,StopwatchCode,"")
endif
```

If the code you assign to the ExecuteEverySecond variable contains an error Panorama will display an alert with the error message. It will then wait 20 seconds before it trys to execute the variable again. This delay gives you a chance to do something (perhaps simply quitting Panorama) before the error occurs again.

By the way, Panorama is not guaranteed to execute the procedure in the ExecuteEverySecond variable every single second. The procedure will not be executed when you are editing data, graphics, or a procedure. The procedure will not be executed when you hold down the mouse for an extended time, or if an operation like sorting, selecting or opening a file takes more than a second. Also, the procedure will not be executed if Panorama is not the frontmost program (in other words, if another program is on top).

Triggering a Procedure Every Minute

Once every minute Panorama checks for a special variable named ExecuteEveryMinute and executes the contents, if any. This variable is just like ExecuteEverySecond except that it only runs once per minute. At that rate this variable isn't much use for animations, but it can be used to check for reminders. This example beeps and displays the time once per hour.

```
fileglobal ExecuteEveryMinute
ExecuteEveryMinute={
   if timepattern(now(),"mm")="00"
      beep
      message "It's "+timepattern(now(),"hh")+" o'clock"
   endif
}
```

Panorama will execute this procedure as close to the top of the hour as possible. If it cannot execute the procedure exactly at the top of the hour (because another program is on top, or you are editing data, graphics or a procedure) it will execute the procedure as soon as possible after the hour.

Triggering a Procedure As Soon As Possible

The ExecuteASAP variable is similar to the ExecuteEverySecond and ExecuteEveryMinute variables. To use this variable, create it and then fill it with the text of a procedure. The procedure will run as soon as possible when the current procedure is finished. This is especially handy in Handler procedures to allow them to trigger things they normally couldn't do, like open or close windows (see "Event Handler Procedures" on page 1522).

The ExecuteASAP procedure should undefine the variable itself as part of the procedure. Otherwise the procedure will run over and over again. Just make sure the following line appears somewhere within the procedure.

```
undefine ExecuteASAP
```

To see an example of the ExecuteASAP variable in action, check out the CLOSEWINDOWKEEPSECRET statement in the _UtilityLib Library.

Event Handler Procedures

Panorama procedures are usually triggered by relatively "hi-level" events like clicking on a button or choosing from a menu. However there is a special type of procedure that is triggered by more low level events like simply bring a window to the front. These “event handler” procedures (also called simply “handler procedures”) let you control how Panorama responds to these low level events.

Internally, event handler procedures work slightly differently than regular procedures. When an event handler procedure is triggered, Panorama stops everything and runs that procedure immediately. If a regular procedure causes the event handler procedure to trigger, the regular procedure will pause and wait for the event handler procedure to finish before continuing.

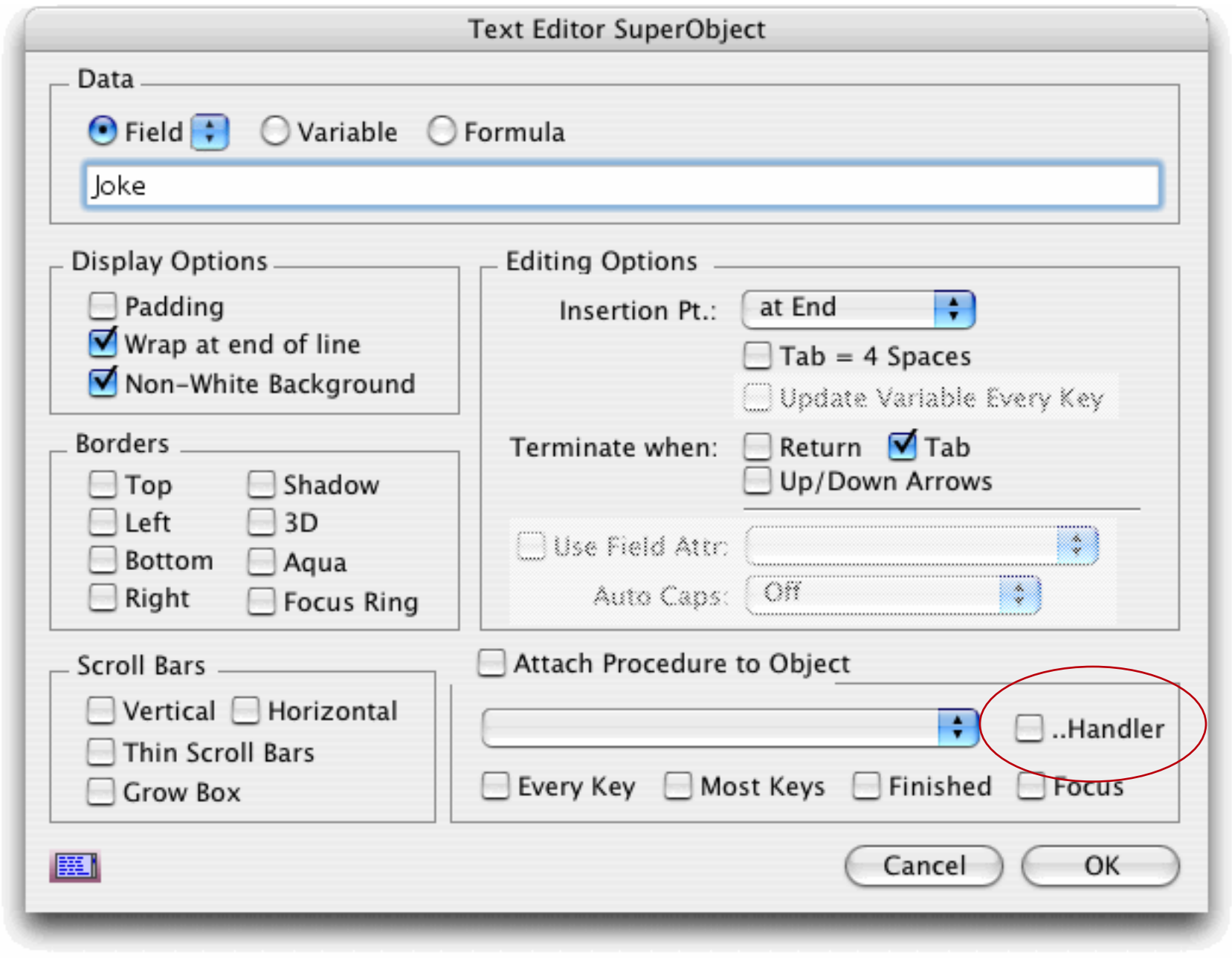
Event handler procedures are intended for changing the way Panorama responds to various low level events. An event handler procedure cannot contain any statements that would cause more low level events. In practical terms this means that an event handler procedure cannot change the arrangement of windows on the screen in any way—it cannot bring another window to the top, open a new window, close a window, or open or close any files. You cannot use the debugger with event handler procedures, because the debugger itself generates low level events. An error dialog will appear if your event handler procedure attempts to perform a statement that would cause another low level event. Event handler procedures should not pause for user input unless you really want to annoy your users. (However, it can sometimes be convenient to use a `message` statement to help debug an event handler procedure.)

Event handler procedures should be as short as possible. Extra delays in processing low level events will be very noticeable. The event handler procedure should only deal with the event in question and should not contain any other logic for your application. If possible an event handler procedure should be written with one or two simple statements or formulas.

Most event handler procedures have names that begin with two periods (**..ActivateForm**, **..CustomAbout**, etc.) to help distinguish them from ordinary procedures. The following sections describe each type of event handler procedure in detail.

Text Editor SuperObject ..Handler Option

The text editor SuperObject has always been able to trigger procedures when various events occur: pressing a key, pressing most keys, and terminating the editing of the object (see “Text Editor Options” on page 643). However, users of early versions of Panorama encountered problems with this feature, since it would not work properly when other procedures were running, and would not work properly if the user terminated editing by clicking on another window. In addition, there was no way to automatically run a procedure when editing started. The ..Handler option (in the Text Editor SuperObject Object Properties dialog) solves all of these problems.



When the ..Handler option is turned on, all procedures triggered by the Text Editor SuperObject are treated as event handler procedures. The benefit of using event handler procedures is that these procedures are guaranteed to trigger and work properly under all conditions, no matter how the user started or stopped editing and whether or not another procedure is currently running. The only downside is that event handler procedures cannot open or close windows (see the previous section). To retain compatibility with databases created with earlier versions of Panorama you are allowed to turn the ..Handler option off.

Focus Procedure

Panorama 3.1 added a new condition that may cause the Text Editor SuperObject to trigger a procedure. This condition is called **Focus**. When the Focus option is turned on, the text editor will trigger its procedure whenever you start editing that field. In other words, when you click on the field or tab into the field, the procedure will be triggered. When the procedure is triggered this way, the `info("trigger")` function will begin with Focus. followed by the name of the object, for example Focus.TimeEditor or Focus.HTML.

One use for a focus procedure is to implement Undo for editing. (Of course as of Panorama V this would be redundant, since Panorama already has Undo for editing within a cell.) Here is a procedure that saves the data in the field as the editing begins.

```
if info("trigger") beginswith "Focus."
   undoCell=«»   /* «» is the current field */
   undoField=info("fieldname")
endif
```

The Undo procedure would look like this.

```
if undoField≠""
   set undoField,undoCell
endif
```

For completeness you may wish to add the following line to your **.CurrentRecord** procedure. This line ensures that you cannot undo after moving to a different record.

```
undoField=""
```

Another use for the **Focus** procedure is to memorize the selection point when editing was terminated and re-set the selection when editing resumes again. This example assumes that the database has two numeric fields named textStart and textEnd.

```
if info("trigger") contains "focus"
   activesuperobject "setselection",textStart,textEnd
else
   activesuperobject "getselection",textStart,textEnd
endif
```

The Focus option cannot be used if the ..Handler option is turned off. This is not a big handicap, since you obviously don't want to change window just as editing begins. You should also keep the procedure as short as possible to minimize delay.

..OpenForm

The **..OpenForm** event handling procedure is triggered when any form in the same database as the procedure is opened. It doesn't matter how the form is opened—manually by the user, in a procedure, or automatically as part of opening the database. The most common uses for the **..OpenForm** procedure are initializing window variables and initializing SuperObjects. The **..OpenForm** procedure below will automatically open the Text Editor or Word Processing SuperObject named Letter when the Editor form is opened.

```
if info("formname")="Editor"
   superobject "Letter","open"
endif
```

..ActivateForm

The **..ActivateForm** event handling procedure is triggered when any form in the same database as the procedure is activated (brought to the front). It doesn't matter how the form is activated—manually (by clicking on it) or as part of a procedure (usually the `window` statement).

The example below shows how this procedure can be used with clone windows (see "Window Clones" on page 1584). It assumes that your database contains a field called ID with unique values for each record. When a clone window is activated (brought to the front), the procedure automatically searches for the record that corresponds to that window.

```
windowvariable windowID
if info("formname") beginswith "Clone"
   find ID=windowID
endif
```

The careful reader may wonder if opening a form also activates it. The answer is yes. When a form is opened, both the **..OpenForm** and **..ActivateForm** procedures will be triggered (if they exist), in that order.

..CustomAbout

The **..CustomAbout** procedure allows you to change the name of the **About Panorama** item in the Apple menu (Mac) or Help menu (PC). This item normally says **About Panorama...** or **About Panorama Direct...**, but you can customize it to display any text you want when your database is active, for example **About This Database...** .

The first step in customizing the **About** menu item is to create a new procedure in the database called **..CustomAbout**. You must spell this name exactly as shown, including upper and lower case.

The **..CustomAbout** procedure should only have a single statement in it: `SetAboutMenu`. This statement is followed by a text parameter, which could contain a formula, that specifies what text should be displayed. For example, if you want the Apple Menu to display **About Office 97...**, the **..CustomAbout** procedure should look like this:

```
setaboutmenu "About Office 97…"
```

Panorama runs the **..CustomAbout** procedure every time you click on a form or data sheet window. This means that you can adjust the Apple Menu for changing conditions. Here is a **..CustomAbout** procedure that displays the name of the current form:

```
setaboutmenu "About "+info("formname")+"…"
```

The **..CustomAbout** procedure only applies to the forms and data sheet in the same database. As you click from window to window, the About item in the Apple Menu may change. When a procedure, flash art, or design sheet window is open the Apple Menu will display the standard **About Panorama...** or **About Panorama Direct...** message. The standard message will also be displayed when a window from any other database is active (unless that database also has a **..CustomAbout** procedure).

Note: By using the **..CustomAbout** and **.About** procedures (see ".About" on page 1508) you can almost completely hide the fact that your application is created in Panorama. (However, on the Macintosh the Application Menu in the upper right hand corner of the screen will always show the name Panorama, while on the PC the master window and the task bar will always show the name Panorama.) Don't forget that you must include the Panorama copyright message in any custom About window that you create with the **.About** procedure!

..CloseDatabase

The **..CloseDatabase** procedure allows you to customize the actions that occur when a database is closed. Whenever a database is closed (either by clicking on the close box of the last open window, using the **Close File** command, or the **Quit** command), Panorama checks to see if the database contains a procedure called **..CloseDatabase**. If this procedure exists it will be called as a "handler" procedure.

If the procedure contains a `ContinueClose "yes"` or `ContinueClose "no"` statement, Panorama will not display the normal "Do you want to save changes" dialog, and will not save the database. Instead it is up to the procedure to decide what to do and to go ahead and do it. For example, if you just want to close the database without saving it the procedure can be simply one line:

```
ContinueClose "yes"
```

Please note that if the procedure contains a `ContinueClose "no"` statement, Panorama may ignore this under certain situations. For example, if a Quit has been requested by an Apple Event, the Quit will continue no matter what.

If necessary, the procedure can use the `info("quitinprogress")` to check whether Panorama is in the process of shutting down.

We've written some custom statements that automate the most common ways of handling database closing. The CLOSEWINDOWKEEPSECRET statement (in the _UtilityLib Library) doesn't allow Panorama to close the database. Instead, it makes the database secret (all windows closed) but leaves it in memory. However, the database will close when Panorama quits. This statement also saves the database. (Note: a ..CloseDatabase procedure with this statement has been added to any custom statement library database created by the Custom Statement wizard. This is why the custom library is saved automatically when you close all of the windows.)

The STANDARDCLOSEALERT statement mimics the normal way Panorama closes a database. It displays the dialog asking the user whether they want to save, not save, or cancel, then does the appropriate operation. This statement is useful if you want the standard operation but with some additional processing before or after.

Chapter 25: Programming Techniques



The previous chapter covered the basics of working with procedures - how to create and edit procedures, set up and use variables, and basic control flow. This chapter builds on these basics and shows how to automate a wide variety of tasks within Panorama.

The structure of this chapter is a miniature version of the entire manual. We'll start with Chapter One and show how to automate file handling, then work our way through the entire manual showing how to automate all of the different operations available in Panorama.

Although this chapter is quite extensive, Panorama is so rich that there are many statements, functions and techniques that are not covered here. You'll find a gold mine of additional information in the Programming Reference wizard (see "Programming Reference Wizard" on page 1361) and in the wizards and example files included with Panorama. You'll also find a very vibrant on-line community of Panorama developers on the "QNA" e-mail discussion list. See the tech support page on **www.provue.com** for information on subscribing to this list.

Accessing Files

A Panorama procedure has a wide variety of statements available for working with disk files. A procedure can open and close database files, import and export data to/from a database, or even read and write disk files directly (including resource files and registry entries).

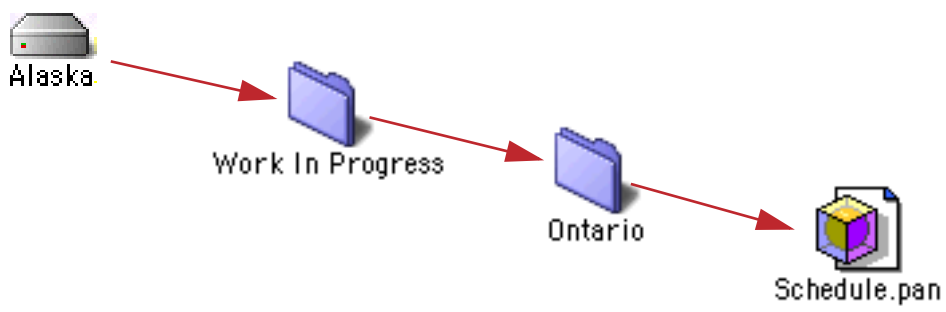
Files and Folders

On both Macintosh and Windows PC systems files are contained in folders, which may themselves be nested inside other folders. There are two different methods that Panorama uses to identify the name and location of a file. Method 1 is to combine both the name and location in a single string of text. Method 2 is to specify a separate file name and folder ID separately. Some statements and functions use method 1, some use method 2. Panorama also include functions that can convert back and forth between these two methods.

Combined Folder Location and File Name

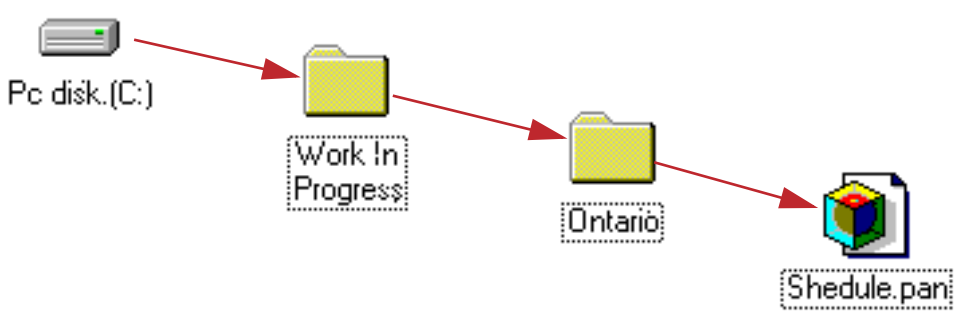
On the Macintosh, the exact location of any file can be specified by stringing together the name of the volume (disk) and the folders, each separated by a colon.

```
Alaska:Work in Progress:Ontario:Schedule.pan
```



Windows systems are similar, but backslashes (\) are used instead of colons, and drive names always consist of letters followed by a colon (A:, B:, C:, etc.).

```
C:\Work in Progress\Ontario\Schedule.pan
```

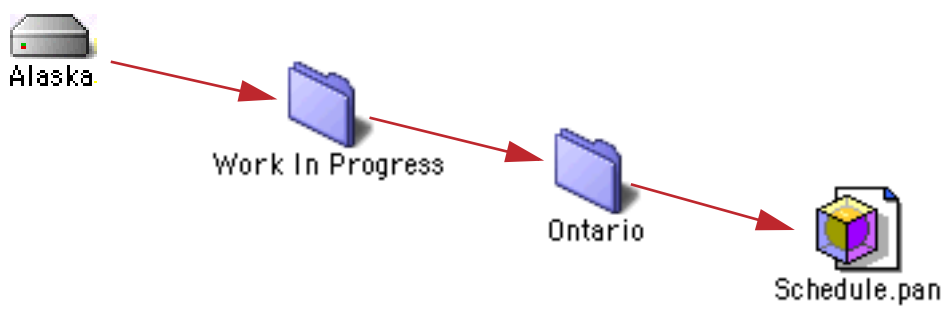


For cross platform compatibility, Panorama also allows you to use colons when using Panorama for Windows, like this:

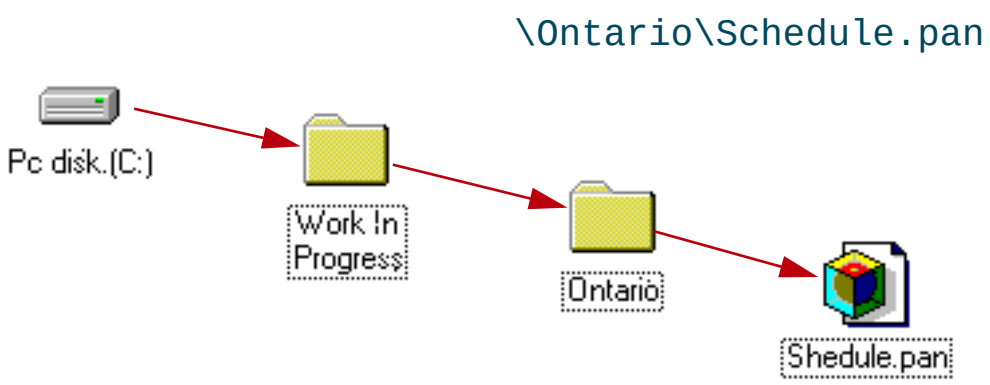
```
C::Work in Progress:Ontario:Schedule.pan
```

A file's location may also be specified relative to the current database. For example, suppose the current database was in the Work in Progress folder. In that case you could specify the location of the Shedule.pan file by simply leaving off left hand portion of the specification. The specification must begin with a colon or backslash to indicate that it is relative to the current folder and not an absolute location.

```
:Ontario:Schedule.pan
```



On PC systems you can specify this relative location like this:



However, keep in mind that on PC systems Panorama will accept : instead of \. Therefore, the specification

```
:Ontario:Schedule.pan
```

will work on both Windows and MacOS based computers. You should use colons if your database might ever be used on both Macintosh and Windows computers.

Folder ID's and Paths

A folder's **path** is simply a list of folders within folders, each separated by a colon (Macintosh) or backslash (Windows PC). The list always starts with the name of the disk drive. Here are some examples of folder paths:

```
GigDrive:Work In Progress:

F:\Clients\Taco Bell\

HD:System Folder:Preferences:

C:\Windows\
```

A **folder ID** is a 6 byte binary value that uniquely identifies a folder. Folder ID's are used by several of the Panorama functions and statements. Panorama can convert a path to a folder id with the `folder(` function, like this:

```
folder("MyDisk:Clients:")
```

The `folderpath(` function converts a folder ID back into a path, for example:

```
folderpath(dbinfo("folder"),"Checkbook"))
```

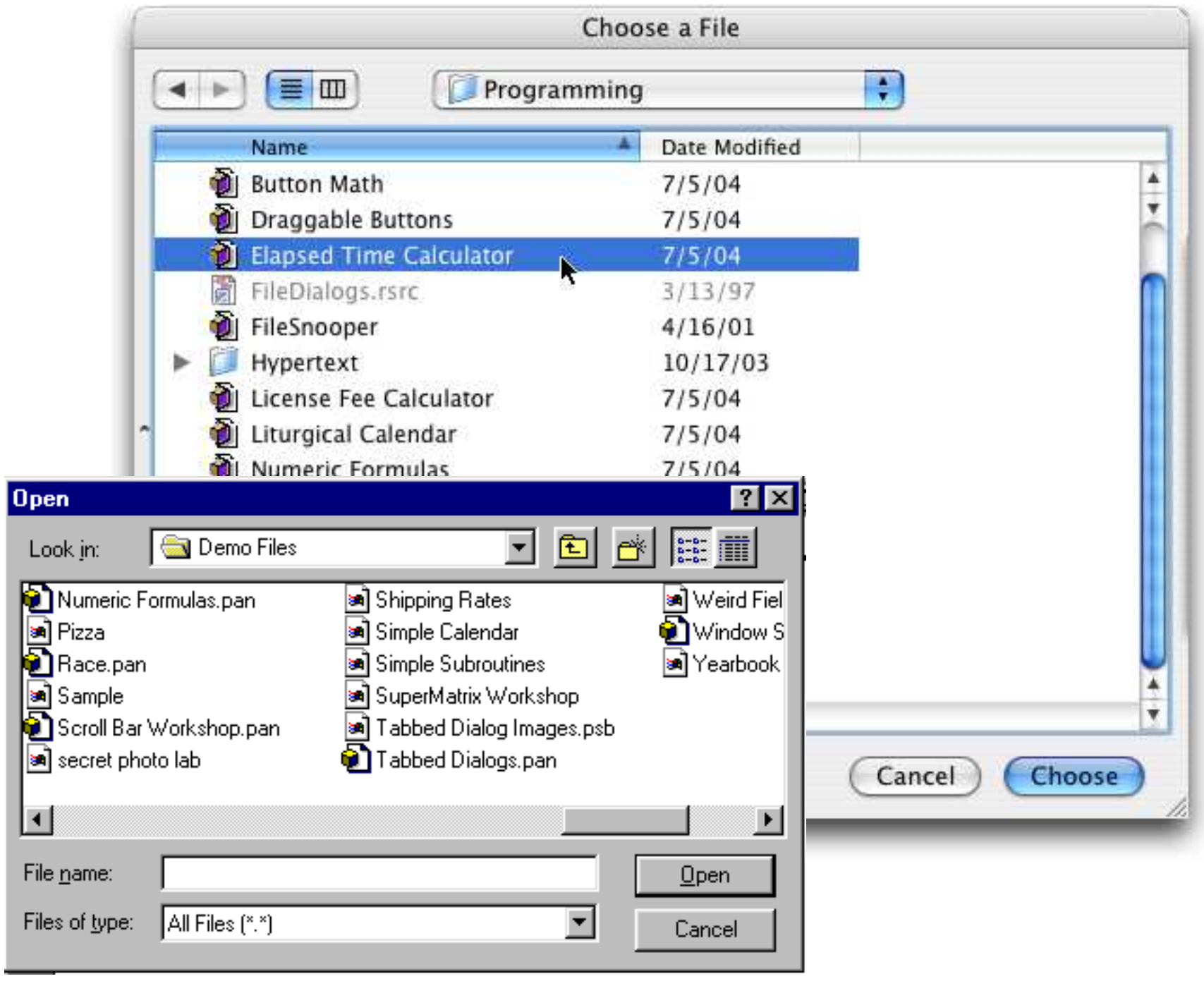
Panorama has numerous functions for working with folders and paths. See "Disk Files and Folders" on page 1304 for detailed information on these functions.

Tip: Most functions or statements that use a folder ID will accept `""`, which means "the folder that contains the current database."

Locating a File with Standard Dialogs

Both the Macintosh and Windows PC's have standard dialogs for locating a file. With the `openfiledialog` and `savefiledialog` statements, your procedures can use these dialogs also.

The `openfiledialog` statement allows the user to select the file using a standard file open dialog (see "OPENFILEDIALOG" on page 5574). This illustration shows what this dialog looks like on both Windows PC and Macintosh computers (the exact appearance may vary depending on what operating system version and extensions you are using).



The file may be anywhere on any disk drive mounted on the computer. The `openfiledialog` statement has four parameters.

```
openfiledialog folder,file,type,typelist
```

The `folder` parameter is a folder ID (see "Folder ID's and Paths" on page 1529). This specifies what folder should be listed when the dialog first opens.

The `file` parameter is the name of the file the user selected. If this parameter is empty the user pressed the **Cancel** button.

The `type` parameter is the type of the file the user selected. This is a four letter descriptor, for example TEXT for text files or ZEPD for Panorama database files.

The `typelist` parameter specifies what kind of files you want listed in the file dialog. If this parameter is empty (`""`), then all files will be listed. Otherwise, this should be one or more 4 letter “file type” descriptions. Here are two file type descriptions you may find useful.

Type	Description
ZEPD	Panorama database
TEXT	Text file

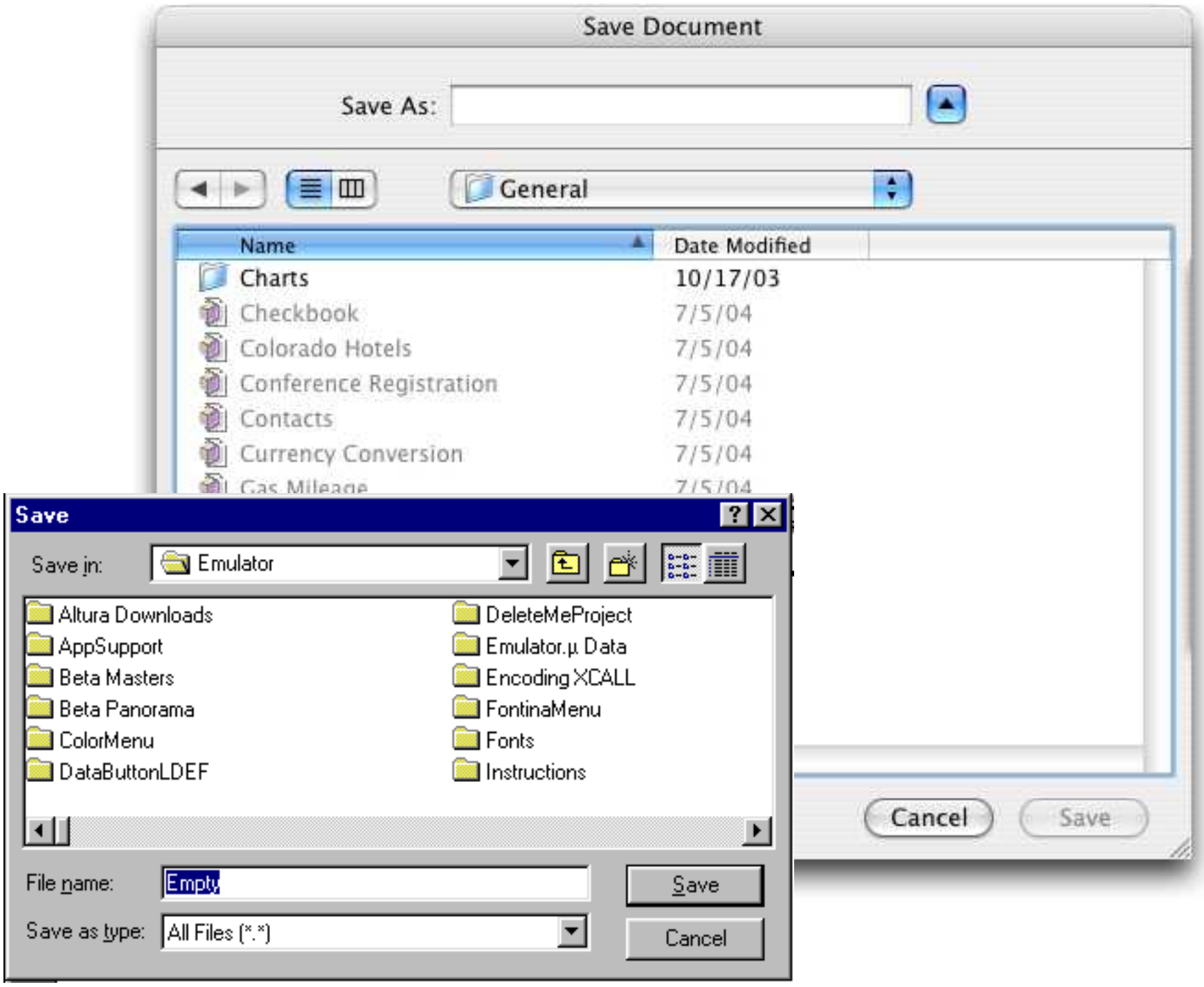
The file type must be entered exactly as shown. Only files whose types match exactly (including upper / lower case) will be shown.

Here is an example of a procedure that allows the user to select a Panorama file, then appends that file to the current file.

```
local fileFolder,fileName,fileType
openfiledialog fileFolder,fileName,fileType,"ZEPD"
if fileName="" stop endif ; user pressed cancel
openfile "+"+folderpath(fileFolder)+fileName
```

To change this example so that it can be used for importing text files, change `"ZEPD"` to `"TEXT"`. Or, to allow either choice, change this parameter to `"ZEPDTEXT"` (or `"TEXTZEPD"`).

The `savefiledialog` statement allows the user to select a file using a standard file save dialog (see "SAVEFILEDIALOG" on page 5698). This illustration shows what this dialog looks like on both Windows PC and Macintosh computers (the exact appearance may vary depending on what operating system version and extensions you are using).



The user may type in the name of the file, and can select what folder the file will be saved into. The file may be anywhere on any disk drive mounted on the computer. The `savefiledialog` statement has three parameters.

```
savefiledialog folder,file,prompt
```

The `folder` parameter is the ID of the folder the user selected.

The `file` parameter is the name of the file the user entered. If this parameter is empty, the user pressed the ˘ button.

The `prompt` parameter is a short phrase that you supply. When using MacOS Panorama will display this phrase in the dialog title, for example `Enter file name`.

Here is an example of a procedure that allows the user to specify a file name and location, then exports to that file.

```
local fileFolder,fileName
fileName=""
savefiledialog fileFolder,fileName,"Export file name:"
if fileName="" stop endif ; user pressed cancel
export folderpath(fileFolder)+fileName,replace(exportline(),¶,chr(11))+¶
```

If the user specifies a file that already exists, Panorama will warn them and ask if they really want to erase the existing file.

Customizing the Standard File Dialogs

Versions of Panorama prior to 5.0 allowed you to customize the dialog that appears when the `openfiledialog` and `savefiledialog` statements are used. However, this only worked on Macintosh computers, and only with the older OS 6/7 style dialogs. Since Panorama now uses Apple (relatively) new File Navigation Services system, the open and save dialogs can no longer be customized on either the Macintosh or on Windows.

Opening a Panorama Database

To open a Panorama database use the `openfile` statement (see “OPENFILE” on page 5572). If the database to be opened is in the same folder as the current database you can simply supply the filename.

```
openfile "Contacts"
```

If the file is in a different folder you must supply a combined path and file name (see “Combined Folder Location and File Name” on page 1528).

```
openfile "C:\My Documents\Organizer\Contacts"
```

On a Windows PC system the file would actually be named Contacts.pan, but you may omit the .pan from the name.

Here is an example of a procedure that allows the user to select a Panorama file and then open it (see “Locating a File with Standard Dialogs” on page 1530).

```
local fileFolder,fileName
openfiledialog fileFolder,fileName,"ZEPD"
if fileName=""  /* user pressed cancel */
   stop
endif
openfile folderpath(fileFolder)+fileName
```

Suppressing the Default Extension

On Macintosh systems the behavior of the `openfile` statement changes slightly if the current database name ends with .pan. In that case the `openfile` statement will automatically add .pan to any file name that doesn’t already have an extension. This makes it easier to set up a set of database files that can work on both the Macintosh and the PC. However, this also means that you will not be able to open a database that doesn’t have any extension at all with this statement (of course you can always open such a file manually using the **Open File** dialog in the File menu).

If you need to use the `openfile` statement to open a file without an extension when the current file does have the .pan extension you must use the `nodefaultextension` statement just before the `openfile` statement. Here is a modified version of the procedure from the last section that allows the user to select any Panorama file (with or without the extension) and then open it.

```
local fileFolder,fileName
openfiledialog fileFolder,fileName,"ZEPD"
if fileName=""  /* user pressed cancel */
   stop
endif
nodefaultextension
openfile folderpath(fileFolder)+fileName
```

Appending Databases End-to-End

If you have two database files with identical field structures (the same fields in the same order), it's easy to append them together. Essentially appending sticks the second database right onto the end of the current database. To append a database use the `openfile` statement, but put a `+` symbol in front of the name of the database. The example below appends the file MoreCustomers to the end of the current database. (Note: The MoreCustomers file does not have to be open, but it's ok if it is.)

```
openfile "+MoreCustomers"
```

If the file you want to append doesn't have identical field order, but the field names are the same, use two `+` symbols, like this:

```
openfile "++MoreCustomers"
```

Only fields with matching names will be appended together (this is the same as checking the Match Fields by Name option in the **Open File** dialog, see "Appending One Database to Another" on page 141). If there are any extra fields in the current database that do not match, they will be left blank in the new appended data.

The `openfile` statement can also be used to append a text file to the current database. See "Importing Text Files" on page 1537.

Eliminating Duplicates in Appended Data

If you need to eliminate duplicates between the newly appended data and the original data, you can use the `sortup` statement to bring the duplicates together, followed by the `unpropagate` statement to clear out the duplicate company names (see "Using UnPropagate to Eliminate Duplicates" on page 470). The `select` and `removeunselected` statements actually eliminate the duplicate records. The example below appends to the customer file, then eliminates any duplicates that were appended. Newly appended records that are not duplicates of the existing data will be kept.

```
openfile "+MoreCustomers"
field Company
sortup
unpropagate
select Company<>""
removeunselected
```

If you want to keep the newly appended data and remove the original data when a duplicate is found, use the `unpropagateup` statement instead of the `unpropagate` statement. For example, you would do this if you felt the newly appended data was more up-to-date or more reliable than the original data.

Replacing the Data in a Database

Occasionally you may want to throw out all the data in the current database and replace it with the data in another database. For example, the data in the current database may be out-of-date. To do this use the `openfile` statement and put an `&` symbol in front of the file name:

```
openfile "&NewCustomerList"
```

You may be wondering, if the data is already in a database, “Why not use that database instead of replacing the data in the current database?” Usually it’s because the file that contains the correct data does not have the forms and procedures set up they way you want them. You can easily transfer the data, transferring forms and procedures is a much more tedious job.

If the database file you want to load doesn’t have the fields in the same order, but the field names are the same, use two & symbols in a row.

```
openfile "&&NewCustomerList"
```

Only fields with matching names will be loaded with data (this is the same as checking the Match Fields by Name option in the **Open File** dialog, see “Appending One Database to Another” on page 141). If there are any extra fields in the current database that do not match, they will be left blank in the newly loaded data.

The `openfile` statement can also be used to replace the current data with data from a text file (see “Importing Text Files” on page 1537). The & option becomes very handy because you can load the raw text into a pre-prepared database that contains ready-to-use forms and procedures.

Saving a Panorama Database

To save the current database use the `save` statement (see “SAVE” on page 5692). This procedure adds a new record and then saves the database.

```
addrecord
save
```

To save all open databases use the `saveall` statement (see “SAVEALL” on page 5694).

To save the current database with a new name or in a new location use the `saveas` statement (see “SAVEAS” on page 5695). The new file becomes the current version of the file. The procedure below will save the current file under the name Monday, Tuesday, Wednesday, etc. depending on the day of the week.

```
saveas datepattern(today(),"DayOfWeek")
```

The `saveacopyas` statement also saves the database with a new name or location, but the new file does not become the current version of the file (see “SAVEACOPYAS” on page 5693). This procedure saves a backup copy of the current file in a folder named Backups.

```
saveas "C:\Backups\"+info("databasename")
```

A procedure can also save the current file as a text file, this is called **exporting** the file. See “Exporting Text Files” on page 1541 to learn how to do this in a procedure.

A procedure can use the `revert` statement to revert to the last previously saved copy of the database (see “REVERT” on page 5676).

Closing a Database

To close the current database use the `closefile` statement (see “CLOSEFILE” on page 5109). All windows associated with the current file will also be closed.

If the `closefile` statement is in the middle of a procedure it will immediately close the file without stopping to ask if you want to save the file. For example, this procedure will close the Checkbook and Payments databases immediately, without saving them, and then open the Bills database.

```
window "Checkbook"
closefile
window "Payments"
closefile
openfile "Bills"
```

A safer version of this procedure might look like this.

```
window "Checkbook"
save
closefile
window "Payments"
save
closefile
openfile "Bills"
```

There is one exception to this rule. If the `closefile` statement is the very last statement in the procedure Panorama will stop and ask if the file should be saved. If you don't want that to happen, add a `nop` statement (see "Doing Nothing for a While" on page 1399) after the `closefile` statement, like this.

```
closefile
nop
```

Shutting Down Panorama

To shut down Panorama use the `quit` statement (see "QUIT" on page 5624). The `quit` statement will not normally ask the user if they want to save changes in any open databases before stopping Panorama. However, if the `quit` statement is the last statement in the procedure, or is followed by a `stop` statement, it will ask the user if he or she wants to save each file, and if they say **yes**, save the files for them.

Importing Text Files

Panorama cannot directly access information in files created by other database or spreadsheet programs. Exchanging information between Panorama and other programs requires an intermediate file which is called a “text” or “ASCII” file (see “Working with Text Files” on page 144). A text file is very basic because it contains just the data—no forms, procedures, formulas, pictures or anything else. Because text files are so simple they provide a common interchange format for different programs. Virtually all database, spreadsheet, and word processing programs on both the Macintosh and the PC can read and write text files.

Like the Panorama data sheet, a text file is divided into records and fields. Each line in the text file is a single record. The fields may be separated by commas or by tabs. You will often hear that a file is “tab delimited” or “comma delimited,” because the tab or comma character delimits the boundaries between each field. When you export from Panorama you must specify whether to use commas or tabs. When Panorama imports a text file it automatically checks for tabs. If it doesn’t find any, it assumes that this text file is comma delimited.

Carriage Returns in the Data

One complication with using text files is handling carriage returns that are actually part of the data in a cell. Usually a carriage return signals the end of the record and the start of a new record. If a carriage return is in a cell, Panorama thinks a new record is starting, and the rest of the import will be misaligned.

The best solution to this problem is to convert carriage returns in cells into vertical tabs (ASCII value 11) in the exported text file. Many programs do this automatically. When the data is imported into Panorama the vertical tabs are automatically converted back into carriage returns again.

If you are manually exporting data from Panorama (using the **Save As** command, see “Exporting a Text File” on page 165) and want the carriage returns converted to vertical tabs, make sure the Tabs w/o quotes option is turned on, and the Output Patterns option is turned off.

Importing a Text File into an Existing Database

The `openfile` statement can also be used to import a text file into the current database (the text can either be appended to or replace the current data). This is called **importing** the text. In a procedure, the technique for importing text is exactly the same as appending or replacing two Panorama databases. Panorama checks the file you have specified to see if it is a database or text file, and automatically imports it if it is a text file.

This example will append the data in a text file named RawData.txt to the end of the current database.

```
openfile "+RawData.txt"
```

This example will erase all the data in the current database, then import the data in a text file named NewRawData.txt.

```
openfile "&NewRawData.txt"
```

Both of these examples have shown files with the .txt extension. On the Macintosh this is optional. On Windows PC systems this extension is required. If you need to import a text file that does not have the .txt extension you can use the `opentext` statement (see “OPENTEXTFILE” on page 5585). This statement is the same as the `openfile` statement except that it treats all files as text files, no matter what extension they have.

The text file must be separated into columns with either tabs or commas, and into rows with carriage returns or carriage return/line feed pairs. (If the first line of the file contains a tab Panorama assumes the file is tab delimited, otherwise comma delimited.) If any vertical tab characters are encountered (character value: 11) they are converted to carriage returns within the actual cell (in other words, this allows you to import carriage returns within a cell without starting a new record.)

Importing from a Variable

If the filename begins with the @ symbol the openfile statement will import from a variable instead of from a text file. For example, the procedure below will import the text contents of the variable MyData into a new database.

```
openfile "@MyData"
```

Adding the + symbol causes the text contained in the variable to be appended to the end of the current database.

```
openfile "+@MyData"
```

Adding the & symbol causes the text contained in the variable to be appended to the end of the current database.

```
openfile "&@MyData"
```

A useful technique is to build an array of data from one database (see “Building an Array from a Database” on page 1719) and then import that data directly into another database.

Importing HTML Tables

When the openfile statement imports a text file (or variable) it checks the text to see if it contains one or more HTML <table> tags. If a <table> tag is found the other text is ignored and Panorama simply imports the data inside the table. The text will be divided into rows and columns based on the <tr> and <td> tags within the table. Here’s an example of how to import a table.

```
openfile "Financial_News.html"
```

If the text file (or variable) contains more than one table only the first will be imported. Any additional tables will be ignored. If you need to import a different table the procedure can use the fileload(function to read the text into a variable (see “Reading Data Files” on page 1550), then use the tagdata(function to extract the table you want to import (see “Tag Parameter Functions” on page 1245). The example procedure below imports the 2nd table from the HTML file (instead of the 1st).

```
local rawHTML,theTable
rawHTML=fileload("","Financial_News.html")
theTable="<table"+tagdata(rawHTML,"<table","</table>",2)+"</table>"
openfile "@theTable"
```

Re-Arranging the Order of Imported Data

Panorama normally imports text directly into the database, column for column. In other words, the first column in the text file goes into the first field in the database, the second column into the second field, etc. This is great if the text file is set up the same way as your database. If not, you’ll need to process the data as it comes in, possibly re-arranging and combining data as you go. This processing is done with a formula you design. The formula is set up by the importusing statement (see “IMPORTUSING” on page 5355), which should be the statement immediately before the openfile statement. The importusing statement has one parameter, the formula for processing the data. Here’s the general idea of how these statements must be used together (several more specific examples follow below).

```
importusing formula
openfile "+MyTextFile.txt"
```

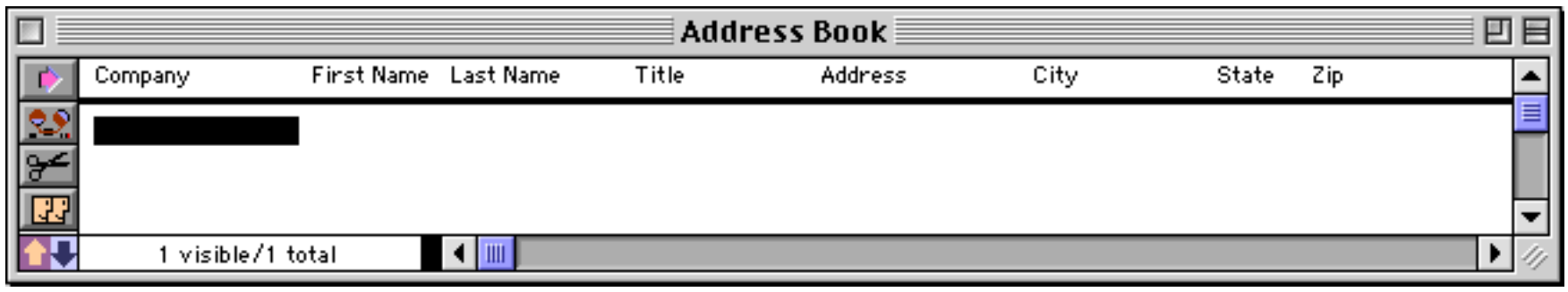
By itself, the `importusing` statement doesn't do anything except stash the formula you provide where the `openfile` statement can find it. Here's what happens. First `openfile` statement reads one line from the text file. But it doesn't actually import the line into the database. Instead, it evaluates the `importusing` formula. The formula takes the line of data and re-arranges it. The `openfile` statement then takes the result of this formula and imports that result into the database. This process is repeated over and over again for each line in the text file: read, calculate, import.

To process the line that the `openfile` statement has read in, the formula needs to be able to access the data in that line. There are two special functions that allow you to read this line. The `import()` function (see "IMPORT(" on page 5352) accesses the entire line that has been read in. The `importcell(columnNumber)` function (see "IMPORTCELL(" on page 5353) accesses an individual cell in the line (tab or comma delimited). The columnNumber starts with 0 for the first column, 1 for the second column, etc.

Suppose you have a text file named Sam's Contacts.txt that contains data like this (each column of data represents fields separated by a tab):

```
Smith    John   World Widgets        124 W. Olive St      San Jose  CA 95134
Lee      Susan  Industrial Metals    2347 N. Riverside    Cambridge MA 02139
Marklee  Lance  Zipper Technologies  687 E. Dorothy Lane  Bothell   WA 98011
Anders   Fred   Acme Fireworks       5672 Lakewood Drive  Salinas   CA 93908
```

You want to import this data into a database that contains these fields:

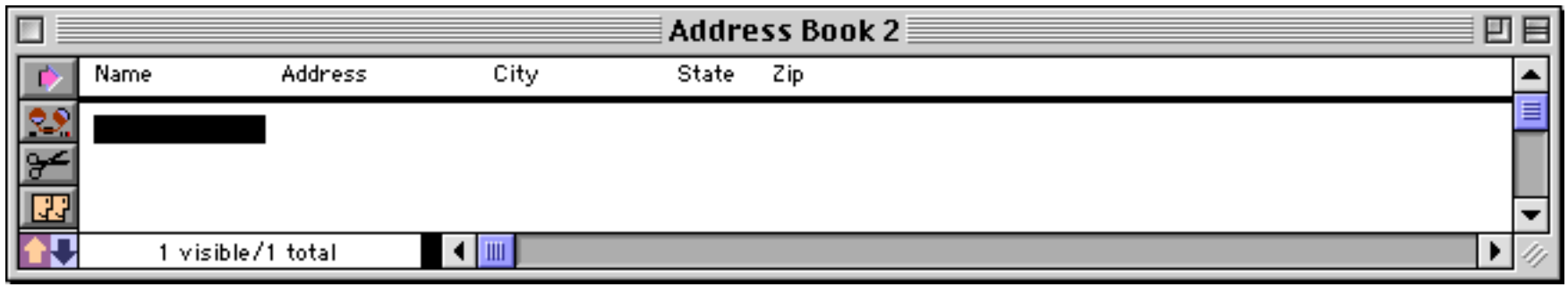


Here's a procedure that will append the data in Sam's Contacts.txt into this database. The tabs (¬) in the formula divide the output into separate columns again so it can be imported (see "Special Characters" on page 1202).

```
importusing importcell(2)+¬+importcell(1)+¬+importcell(0)+¬+¬+
              importcell(3)+¬+importcell(4)+¬+importcell(5)+¬+importcell(6)
openfile "+Sam's Contacts.txt"
```

The formula re-arranges the incoming data so that third column in the input text goes into the first field, the 2nd column goes into the 2nd field, the first column goes into the 3rd field, the 4th field is empty, the 4th column goes into the 5th field, the 5th column goes into the 6th field, the 6th goes into the 7th field and the 7th column goes into the 8th field.

In this example, each column in the input corresponds with one field in the final database. However, you could split up a column into multiple fields, or combine multiple columns in the input text into a single field in the final database. For example, suppose your Address Book database only had five fields, like this.



Here is a procedure that imports Sam's Contacts.txt into this version of the Address Book database.

```
importusing importcell(1)+" "+importcell(0)+¬+
   importcell(3)+¬+importcell(4)+¬+importcell(5)+¬+importcell(6)
openfile "+Sam's Contacts.txt"
```

This formula simply concatenates the first and last names with a space, but you can use any function you want, including the `?(`, `sandwich(`, `upper(`, `lower(`, even `lookup(` functions.

Building the ImportUsing Formula on the Fly

The `importusing` formula is usually hard-coded into the procedure. What if however, you don't know how the data should be re-arranged in advance? In that case you might want to design a dialog that would allow the user to configure how the data is re-arranged. The dialog would build the formula for importusing. To pass the formula to the importusing statement, you must put the formula into the clipboard and then use the following statement:

```
importusing clipboard
```

Warning: You must enter this statement exactly as shown here. Do not put `()` after the word `clipboard`. Do not add anything else at all to this statement.

Here is a very simplified example that shows how it is done. First the procedure asks for the name of the file to import, then it asks for the order of the fields. The field order should be entered as numbers separated by spaces, for example 4 1 2 3 8 9 10.

```
local importFile,importOrder,importFormula,X
importFile="" importOrder="" importFormula=""
gettext "Import what file?",importFile
gettext "Field order? (ex: 4 1 2 3)",importOrder
X=1
loop
   importFormula=sandwich("",importFormula,"+¬+")+
      "importcell("+array(importOrder,X," ")+")"
   X=X+1
while X<=arraysize(importOrder," ")
clipboard=importFormula
importusing clipboard
openfile "+"+importFile
```

As the program goes through the loop it builds up the formula. For example, if the user entered 3 2 4 the procedure would generate the formula:

```
importcell(3)+¬+importcell(2)+¬+importcell(4)
```

When the formula is completely assembled it is placed into the clipboard, and then the text file is appended to the current file.

Although this example illustrates how the technique works, the user interface is lousy. Instead of having the user type in the field order, you'll probably want to let the user choose the list order with a List or Pop-Up SuperObject.

Exporting Text Files

You can manually export a database to a text file with the **Save As** command (see "Exporting a Text File" on page 165). To export a database from a procedure, use the `export` statement ("EXPORT" on page 5207). This statement has two parameters:

```
export file,formula
```

The `file` parameter is the name of the text file to export. If this file already exists, it will be erased and the new file will replace it. If the file needs to be in a different folder than the current database then a combined folder and file must be supplied (see "Combined Folder Location and File Name" on page 1528).

The `formula` parameter is a formula that controls how the data is formatted as it is exported. To do its job, the `export` statement scans each visible (selected) record in the database. For each record it calculates the result of the formula, and adds that result to the text file being exported.

Usually the export formula consists of a series of fields separated by tabs and ending with a carriage return. Remember, in a formula ¬ represents a tab and ¶ represents a carriage return (see "Special Characters" on page 1202).

Here is a typical formula that exports a name and address list.

```
export "Addresses.txt",Name+¬+Address+¬+City+¬+State+¬+Zip+¶
```

Numeric and date fields must be converted to text before they can be exported. The functions listed in this table can perform these conversions.

Function	Reference Page	Description
exportcell(value)	Page 5209	This function converts a value into text without any special formatting. For numeric values this function is the same as the **str(** function (see below). The advantage of this function is that it works with any kind of value - text, numeric or date. Use this function when for some reason you don't know what kind of data you need to convert.
pattern(number,string)	Page 5599	This function converts a number into text, using the string as an output pattern. For example the formula pattern(Price,"$#.,##") will convert the price 3458.23 into the string $3,458.23. The pattern adds the $ and the comma. For more information on numeric output patterns see "Numeric Output Patterns" on page 290.
str(number)	Page 5799	This function converts a number into text without any special formatting. If you want to format the number (add commas, set # of digits, etc.) use the pattern(function.
datepattern(number,pattern)	Page 5148	This function converts a number representing a date into a formatted text string. The pattern parameter is an output pattern telling the function how to format the date. For more information on date output patterns, see "Date Output Patterns" on page 295. Use the **datepattern(** function to store a date in a text field, or to display a formatted date in an auto-wrap text object or Text Display SuperObject. For example, the formula: `datepattern(«Ship Date»,"Month ddnth, yyyy")` can be used to display the date an order was shipped in the format May 12th, 2003.

Here is a function that exports data from a checkbook database using these functions.

```
export "Checks Archive.txt",str(«Check#»)+¬+datepattern(CheckDate,"MM/DD/YY")+¬+
   PayTo+¬+Category+¬+str(Debit)+¬+str(Credit)+¶
```

If one or more data cells might contain carriage returns, you may wish to convert the carriage returns into vertical tabs as they are being exported. The generic formula for this conversion is **`replace(<text>,¶,chr(11))`** (see "REPLACE(" on page 5665 and "CHR(" on page 5099). Here's a specific example.

```
export "Addresses.txt",replace(Name+¬+Address+¬+City+¬+State+¬+Zip,¶,chr(11))+¶
```

If you simply want to export all the fields in the same order that they appear in the data sheet, use the `exportline(` function (see Page 5210). This function produces tab delimited output of all the fields.

```
export "GenericText",replace(exportline(),¶,chr(11))+¶
```

Don't forget the ¶ on the end of the formula. Without this the exported text file will all be on a single line!

Exporting Line Items as Separate Records

Usually when Panorama exports data there is one line in the exported text file for each selected record in the database. So if the database has 67 records, the exported text file will have 67 lines.

This one-to-one correspondence does not apply if the export formula contains one or more line item fields with the Ω symbol (see "Special Characters" on page 1202), for example QtyΩ or PriceΩ. If the formula contains line items, the export file will contain multiple lines for each record—one line for each line item.

For example, consider an invoice database with 6 line items: Qty1, Qty2, … Qty6, Description1, Description2, … Description6, Price1, Price2, … Price6 etc. The procedure below can be used to export the line items from this database:

```
export "Items",«Invoice#»+¬+
   str(QtyΩ)+¬+DescriptionΩ+¬+str(PriceΩ)+¬+str(TotalΩ)+¶
```

The text file (Items) will contain 6 lines for each invoice. The first line will contain the invoice number, Qty1, Description1, Price1, and Total1. The second line will contain the invoice number, Qty2, Description2, Price2, and Total2. The output continues for each line item, then starts over at Qty1 for the next record.

Here's a sample of how the exported data might look. This shows the data from three invoice records. Invoice 17882 has three line items. The remaining 3 line items are blank. The next two invoices have two line items each.

```
178822  Widget           4.00  8.00
178821  Mini Widget      2.50  2.50
178824  Modern Widget    5.00  20.00
17882
17882
17882
178831  Art Deco Widget  7.50  7.50
178833  Thingy           3.00  9.00
17883
17883
17883
17883
178841  Modern Widget    5.00  5.00
178842  Micro Thingy     4.00  8.00
17884
17884
…
…
```

Warning: The line item export feature will only work if all the line item fields in the database have the same number of line items. If some line item fields have more line items than others, only a single record will be exported. For example if the database contains Address1, Address2, Qty1, Qty2, Qty3, Qty4, Qty5 the line item export feature will not work. The solution is to rename Address1 and Address2 to names that don't end with a number (perhaps Address and Suite, for example).

Analyzing Line Items

One great application for exporting line items as separate records is that you can re-import them into another database and analyze them. For example, from the invoice database it is very difficult to find out how many widgets or doo-dads you've sold. The information is split across all the line item fields. But if you export the line items as separate records and re-import this file into another database, it's easy to sort or select line item data.

The procedure below assumes you have set up a database called Line Item Analysis in advance. This database has five fields: Invoice#, Qty, Description, Price and Total. The procedure exports the line item data from the Invoice database, then re-imports it into the Line Item Analysis database.

```
export "Items",«Invoice#»+¬+str(QtyΩ)+¬+DescriptionΩ+¬+str(PriceΩ)+¬+str(TotalΩ)+¶
window "Line Item Analysis"
openfile "&Items"
select Description≠""
removeunselected
field Description groupup
field Qty total
field Price average
field Total total
outlinelevel 1
```

After the data is imported, the procedure removes all of the empty records (with the `select` and `removeunselected` statements). Then the procedure uses Panorama's standard analysis tools (`groupup`, `total`, `average`, etc.) to calculate how many of each type of item has been sold at what average price.

Exporting Array Elements as Separate Records

Usually when Panorama exports data there is one line in the exported text file for each selected record in the database. So if the database has 67 records, the exported text file will have 67 lines.

This one-to-one correspondence does not apply if the export formula contains one or more `arrayscan(` functions. This function allows you to export the contents of arrays (see "Text Arrays" on page 1236) with one element per exported line. The `arrayscan(` function (see "ARRAYSCAN(" on page 5052) has two parameters:

```
arrayscan(field,separator)
```

The field parameter is the name of a database field that contains an array. (A variable will also work, but usually doesn't make sense.) The separator parameter is the separator character for this array (see "Picking a Separator Character" on page 1236).

For example, suppose your database has a Phones field which contains an array of one or more phone numbers, separated by a carriage return. Each array element contains the type of phone number, a comma, and the phone number itself, like this:

```
home,(714) 555-1212
office,(714) 555-8932
fax,(714) 555-8938
```

The procedure below will export the phone numbers with one record per phone number:

```
export "Phone List",
   Name+¬+array(arrayscan(Phones,¶),1,",")+¬+array(arrayscan(Phones,¶),2,",")+¶
```

This procedure will output a text file something like this:

```
Joan Selbyhome(714) 555-1212
Joan Selbyoffice(714) 555-8932
Joan Selbyfax(714) 555-8938
Sally Rogersoffice(508) 777-8922
Sally Rogersfax(508) 777-8910
Chris Robertsoffice(909) 874-1234
```

Notice that, unlike the line item example in a previous section (see “Exporting Line Items as Separate Records” on page 1542), no blank lines are exported. Panorama counts the number of elements in the array, and outputs exactly that number of lines. If you use multiple `arrayscan(` functions in the formula, Panorama will export enough lines to handle the largest array.

The `arrayscan(` function can also be used in the formula for the `arraybuild`, `arrayselectedbuild`, or `arraylinebuild` statements (see “ARRAYBUILD” on page 5038). The `arrayscan(` works exactly the same as it does with the `export` statement, but the final result is an array instead of a text file.

Opening a Document in Another Application

You can use the `openanything` statement to open any document on your hard drive in it’s native application. For example, you can use this statement to open a *.doc* file in Microsoft Word, or a *.pdf* file in Adobe Acrobat (or Apple’s Preview program, if that is the default viewer for *.pdf* files on your system.

This example opens the Adobe Acrobat document Manual.pdf in the folder My Documents.

```
openanything folder("C:\My Documents"),"Manual.pdf"
```

This example opens the HTML document Roadshow.html in the folder My Test Site. The HTML document will be opened using your default web browser, which is usually Internet Explorer on Windows computers and Safari on Macintosh computers.

```
openanything folder("My Drive:My Test Site"),"Roadshow.html"
```

You can also use `openanything` to directly open an application without opening any document, like this:

```
openanything folder("My Drive:Applications"),"Marine Aquarium.app"
```

This opens the applications just as if you had double clicked on it.

Smart Merge Synchronization

If you have the same database on more than one machine (for example on a desktop computer and a portable computer) you may sometimes wonder which contains the most up-to-date information. If you build **smart merge** into the database, you won't have to wonder anymore. Panorama will keep track of which information is most up-to-date on a record by record database. When you run a special merge procedure, Panorama will merge the two databases, picking the most up-to-date information from each. It's quite easy to add this feature to almost any Panorama database.

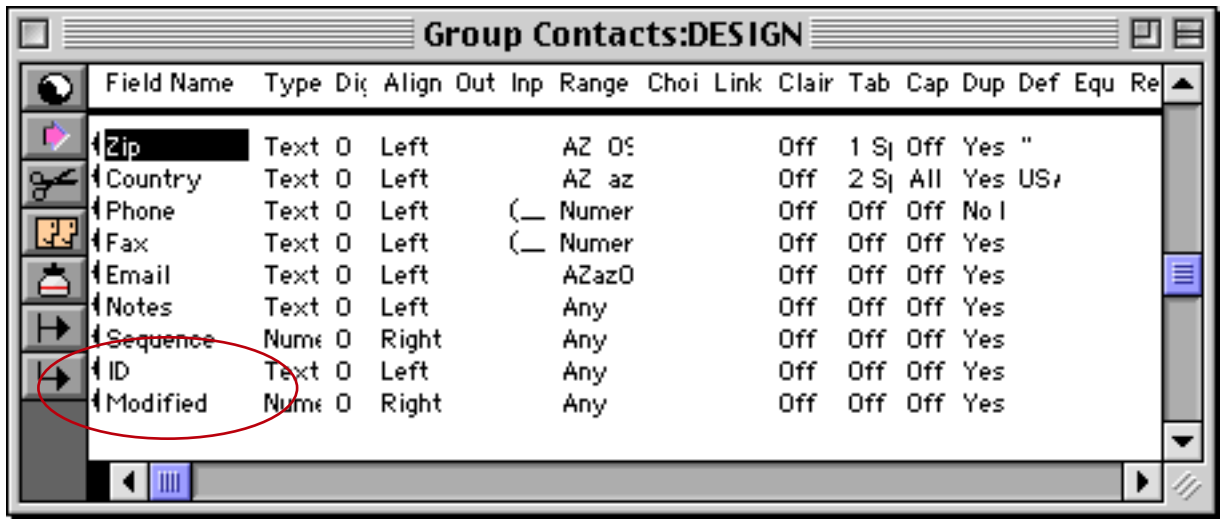
(Note: If you are using Panorama's multi-user Partner/Server capabilities, you do not need to add separate **smart merge** synchronization—Panorama automatically synchronizes all copies of the database using the server.)

How Smart Merge Synchronization Works

Smart merge works by comparing two database files record by record, keeping only the most recently modified version of each record. To do this it must be able to match up the corresponding records in the two database files, even if the database files have been sorted or otherwise rearranged. To do this it uses a special ID field. This field contains a unique ID value for every record in the database. The unique ID value is assigned when the record is first created, and never changes no matter how many times the record is modified or copied to other computers. To guarantee that the ID value is unique it is created by combining the name of the person creating the record along with a serial number, for example Lisa267 or John3091. Because of this, every computer you plan to use must have a different user name configured.

Adding Smart Merge to Your Database

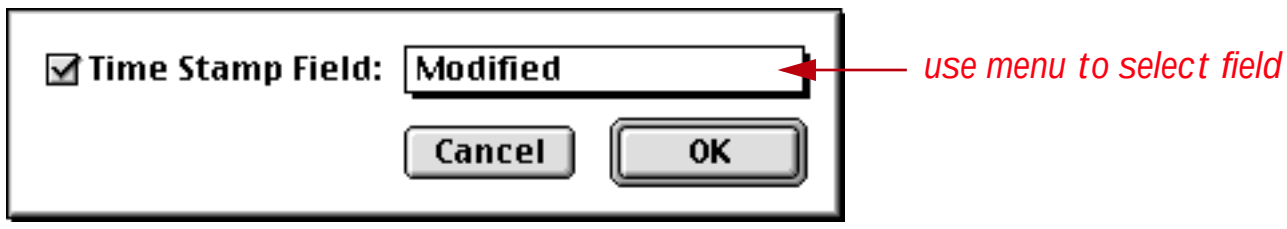
The first step is to add two new fields to your database. We usually call these fields **ID** and **Modified**. This illustration shows these fields added to the design sheet of an address book database.



The ID field will contain the unique ID value for each record, and should be a text field. The Modified value will contain the most recent modification date and time for each record, and should be a numeric field. You may also want to have a creation date field, but this is not necessary for the operation of the smart merge feature.

The Modified Field

Panorama can automatically update the Modified field with the current date and time whenever the database is modified. This feature is called time stamping. To enable this feature, open the design sheet for your database. Choose the **Time Stamp Field** command from the Special menu (see "Automatic Time / Date Stamping" on page 338). This opens a dialog box with a pop-up menu. The pop-up menu lists all the integer fields (Numeric 0 digits) in your database — use the menu to select the Modified field.



(If you don't see the Modified field listed, close the dialog, press the **New Generation** tool, then open the **Time Stamp Field** dialog again.)

Once you have designated the Modified field as the time stamp field, Panorama will automatically place a SuperDate containing the latest date and time into the field every time a new record is added, or whenever any other cell in the record is modified (see "SuperDates (combined date and time)" on page 1259).

Adding New Records

Whenever a new record is added to your database, you must make sure that the ID field is filled in. The best way to do this is to add a **.NewRecord** automatic procedure to your database (see ".NewRecord" on page 1513). The line shown below will fill in the proper value in the ID field.

```
ID=uniqueid("ID",info("user"))
```

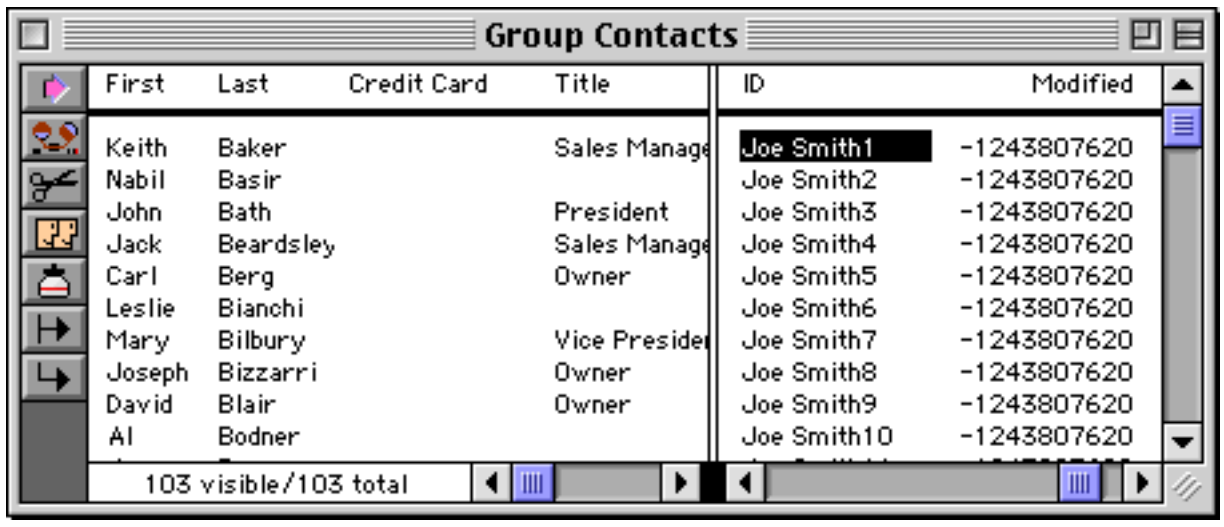
Although you may find other uses for it, the `uniqueid(` function was designed specifically for creating unique smart merge serial numbers (see "UNIQUEID(" on page 5870). This function has two parameters: the name of the field containing the ID serial numbers and a root name. You can get the root name by using the `info("user")` function. The `uniqueid(` function will scan the ID field to find the next serial number available. For example, if you are using a computer belonging to Sam and the highest Sam serial number is 296, the `uniqueid(` function will return the value Sam297.

Creating an **.NewRecord** procedure may not be enough to insure that the ID field is always filled in. If your database has procedures that create new records, the **.NewRecord** procedure will not automatically be called. You must modify these procedures to call the **.NewRecord** procedure (using the `call` statement).

Another possible problem area is imported data. When you import data into the database you must make sure that the ID and Modified fields are filled in. The procedure listed below will do the job. You should also run this procedure when you first add smart merge to your database, so that all your existing data will be properly identified.

```
select ID=""
field Modified
formulafill superdate(today(),now())
field ID
formulafill uniqueid("ID",info("user"))
selectall
```

When you first run this procedure after adding the ID and Modified fields it will initialize the fields something like this.



The Smart Merge Procedure

Once the ID and ModDate fields are set up, you're ready to build the actual smart merge procedure. This procedure performs three basic operations.

1. Append second file to the current file.
2. Sort by ModDate within ID.
3. Remove records with duplicate ID's

The first step is to append the file you want to merge with the current file. A procedure can do this with the `openfile` command by putting a plus sign in front of the file name, for example, `openfile "+"+MergeFile` (see "Appending Databases End-to-End" on page 1534). The big question is, what file do you want to merge with? If you know the filename in advance, you can simply enter the name into the procedure itself. For example, if you always want to merge with a file named Invoices on a floppy named Data, the procedure should contain the statement

```
openfile "+Data:Invoices"
```

Usually you'll want to let the user choose what file he or she wants to merge with. The example below shows how this can be done with the `openfiledialog` statement (see "Locating a File with Standard Dialogs" on page 1530).

The next step is to sort the database (see "Sorting" on page 1676). The database must be sorted by both the Modified field and the ID field. If two records have the same ID value they will now be right next to each other, with the more up-to-date record closer to the bottom.

The final step is to remove the duplicates (see “Using UnPropagate to Eliminate Duplicates” on page 1709). The `unpropagateup` statement identifies the duplicate records (see “UnPropagate” on page 469). If the same ID value occurs more than once in a row, this statement will clear all but the last value. Once the duplicates are identified, the `select` and `removeunselected` statements delete the duplicates from the file.

```
local PathFile,mergeFile, mergePath,mergeType, DoneID
openfiledialog mergePath,mergeFile,mergeType,"ZEPD"
if mergeFile=""
   stop /* user pressed cancel */
endif
PathFile=pathstr(mergePath)+mergeFile
if mergeFile<>info("databasename")
   alert 1014,"Are you sure you want to merge "+
   mergeFile+" with "+info("databasename")+"?"
   if info("dialogtrigger") contains "no"
      stop
   endif
endif
DoneID=ID   ; so we can go back to this record when finished
openfile "+"+PathFile
noshow
field Modified
sortup
field ID
sortup
unpropagateup
select ID<>""
removeunselected
field SortName
sortup
find ID=DoneID
showpage
endnoshow
```

This **Smart Merge** procedure will work for any database that has the ID and Modified fields properly set up. To use this procedure simply select **Smart Merge** from the **Action** menu.

Directly Reading and Writing Disk Files

Disk files are used for permanent storage of information. Panorama normally takes care of saving information in disk files, and reading it back in again later as needed. However Panorama also gives you the flexibility of accessing disk files directly.

Your disk may contain hundreds or thousands of files. When you create a new file, the operating system (MacOS or Windows) allocates a space on the disk for it. As the file grows, more space is made available as needed (until the disk is full). The operating system keeps track of the exact location and size of each file for you, and makes sure that each file is kept separately and doesn't interfere or overlap with any other file.

What's in a File?

Before launching into the actual business of reading and writing files a little background is in order. Different types of files contain different types of information. A particular file may contain a program, a picture, text, a database a spreadsheet, etc. When reading and writing files it's often important to know what kind of file it is and what it is supposed to contain. On Windows PC systems a file's type can be surmised from the three or four letter extension at the end of the file name — **.exe** for programs, **.txt** for text files, etc. On the Macintosh file extensions are not used for this purpose. Instead, each file has two invisible **tags** that identify what type of file is, and how the file was created. Each invisible tag is four characters long, for example TEXT, APPL, or KASX. You cannot normally see or modify these invisible tags, but you can access them via program statements and functions.

There are literally thousands of different tags and extensions for identifying different types of files. On the Macintosh the tag that identifies the type of file is called the **file type tag**.This table lists a few of the extensions and corresponding file type tags that you may encounter. (Note: Some of these tags, for example PDF, are only three characters long - in that case a space must be added to the end.)

Extension (PC)	Tag (Mac)	File Contents
.exe	APPL	Application (program)
.txt	TEXT	Text file
.pan	ZEPD	Panorama Database
.pnz	KSET	Panorama File Set
.pwp	paig	Panorama Word Processing document (see "Word Processor Document Storage Strategies" on page 696)
.pct	PICT	Macintosh PICT graphic (image)
.png	PNGf	Portable Network Graphic (image)
.jpg	JPEG	JPEG image
.tif	TIFF	TIFF image
.eps	EPSF	Encapsulated Postscript image (EPS)
.pdf	PDF	Adobe Acrobat Document
.mov	MOOV	Quicktime Movie
.wav	WAVE	Sound file
.aif	AIFF	Sound file
.sit	SITD	Stuffit Archive (compressed file)

Files on a Macintosh also include an additional four character tag that identifies what application created the file. This is called the **file creator tag**. Windows doesn't really have a corresponding information. The computer uses this tag to decide what application to launch when you double click on the file. Here are a few of the file creator tags you may encounter.

Tag	Application
KASX	Panorama (3.5 or later)
KAS1	Panorama (3.1 or earlier)
ttxt	SimpleText (text editor)
R*ch	BBEdit (text editor)
CWIE	CodeWarrior (text editor)
MPS	Macintosh Programmers Workshop
ToyS	AppleScript Editor
8BIM	Adobe Photoshop
XCEL	Microsoft Excel
WDBN	Microsoft Word
SIT!	Stuffit Archive

Many Panorama functions use the type and creator tags to select files to be displayed or processed.

Reading Data Files

Panorama has three functions for getting information from data files: `fileload(`, `fileloadpartial(`, and `filesize(`.

The `fileload(` function reads an entire file (see "FILELOAD(" on page 5231). The data in the file can be copied directly into a field or variable, or processed further using the formula. The function has two parameters: a folder ID (see "Folder ID's and Paths" on page 1529) and a file name. This example loads the contents of the system Note Pad into a field or variable named Notes.

```
local Notes
Notes=fileload(info("systemfolder"),"Note Pad File")
```

The `fileload(` function can read the data in any file on your disk. It's up to you to interpret what the data means, however—the `fileload(` function simply reads the raw data, exactly as it appears on the disk. Many files (if not most) will contain unintelligible gobbledygook.

If Panorama cannot read the file because of an error, the function will result in an error. In a procedure, this error can be trapped with the `if error` statement (see "Error Handling with if error" on page 1381).

The `fileloadpartial(` function is similar to `fileload(`, but it reads only a section of the file (see "FILELOADPARTIAL(" on page 5233). It has two additional parameters, the starting and ending positions within the file. These positions are measured in characters from the beginning of the file, with 0 being the first character. The example below reads the first 500 characters from the file My Data in the current folder, then extracts the first line from the data.

```
local LineOne
LineOne=array(fileloadpartial("","My Data",0,500),1,¶)
```

This same example of extracting the first line would also work with the `fileload(` function, but only if there is enough scratch memory to load the entire file (see "Changing Scratch Memory Size (Macintosh OS 9)" on page 192). By using `fileloadpartial(` this procedure requires only 500 bytes of scratch memory no matter how large the file My Data is.

The `filesize(` function calculates the size of a file (see "FILESIZE(" on page 5238). It has two parameters, the folder id (see "Folder ID's and Paths" on page 1529) and file name.

```
if filesize("","Sample File")=0
   message "The file is empty!"
endif
```

Note: If a file named Sample File does not exist in the current folder, the procedure above will display the error message File not found. Use the `if error` statement to trap this error if you want to display your own error or handle the error differently (see "Error Handling with if error" on page 1381).

```
local size
size=filesize("","Sample File")
if error
   message "Sample File does not exist!"
   rtn
endif
message "Sample File contains "+str(size)+" bytes."
```

Writing Data Files

The `filesave` statement copies data into a file (see "FILESAVE" on page 5236). If the file does not already exist, it is created. If the file already contained information, that information is lost.

The `filesave` statement has four parameters

```
filesave folder,file,type,data
```

The first parameter, folder, is the folder ID where the file is to be saved. The second parameter is the name of the file.

The third parameter, type, is an 8 character text item combining the **file type tag** and **file creator tag** for the file (see "What's in a File?" on page 1549). If you are using a Windows PC you can simply use "" for this parameter. If you are using a Macintosh the type and creator tags determine what icon, if any, this file will have, and what application will be launched when you double click on this file. If you use an empty string ("") for this parameter, the file will be set up as a **SimpleText** text file. Although you can create any type of file the most common application is to create a text file, as shown in this table.

Icon	Type / Creator	Description
	TEXTttxt	SimpleText text file
	TEXTR*ch	BBEdit text file
	TEXTCWIE	CodeWarrior text file
	TEXTMPW	MPW text file

The fourth parameter, data, is a formula that produces the data to be saved into the file. This may be a field, variable, or more complex formula.

Here is an example that saves the contents of the Notes field (the current record only) into a text file called Notes.txt.

```
filesave "","Notes.txt","TEXTR*ch",Notes
```

The `fileappend` statement adds data to the end of an existing file (see "FILEAPPEND" on page 5224). If the file does not already exist, it is created. If the file already contained information, that information is retained and the new information is added to the end of the file. The procedure below adds a line to the end of the file Deleted Records Log.txt every time it is triggered.

```
fileappend "","Deleted Records Log.txt","","Record "+str(ID)+" deleted on "+
   datepattern(today(),"Month, ddnth, yyyy")+" at "+timepattern(now(),"hh:mm:ss am/pm")+¶
deleterecord
```

If this procedure is named **.DeleteRecord** it will be triggered every time a record is deleted (see ".DeleteRecord" on page 1510). As records are deleted a log file will be created that looks something like this.

```
Record 4738 deleted on June 4th, 2001 at 3:12:47 PM
Record 392 deleted on June 4th, 2001 at 4:45:02 PM
Record 6133 deleted on June 5th, 2001 at 9:23:20 AM
```

As more records are deleted the log file will get bigger and bigger and bigger.

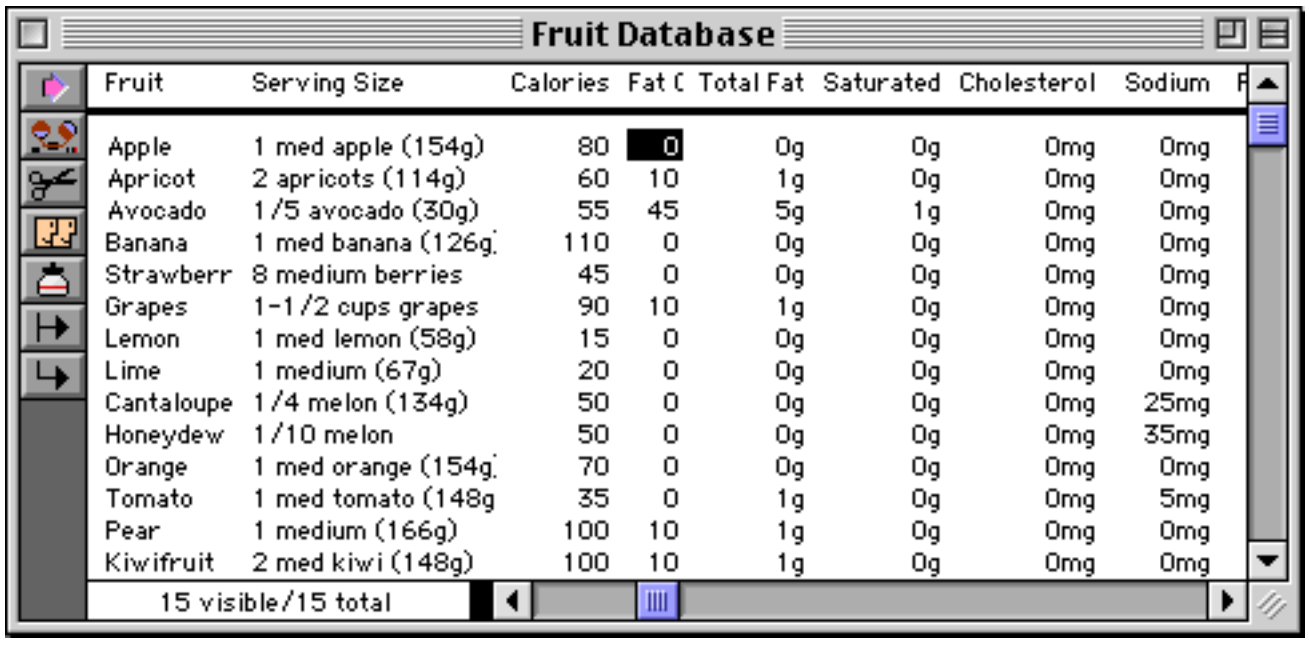
Copying Data Files

Panorama has several built in statements for copying files or parts of files. To learn more about these commands see the `copyfile` and `copyfork` statements in the **Programming Reference** wizard.

Using FileSave and ArrayBuild to Export Data

Earlier in this chapter we learned how to use the `export` statement to export data from the database (see "Exporting Text Files" on page 1541). Another technique is to use the `arraybuild` statement to scan the database and build an array (see "ARRAYBUILD" on page 5038) and then use `filesave` to export the array into a file. The advantage of this technique is that it is more flexible than using `export` because it allows you to add headers and footers to the data. When using Mac OS 9 the disadvantage is that this technique requires at least enough scratch memory to contain the entire exported file, so it will not work for exporting large databases (see "Changing Scratch Memory Size (Macintosh OS 9)" on page 192). On Windows and Mac OS X Panorama can take advantage of virtual memory for this, so it is less of a problem.

To illustrate this technique we will use it to export a file as an HTML table. We'll start with a database of fruit nutrition.



Here is the procedure that takes this database and creates an HTML page. The heart of the procedure is the `arraybuild` statement, which scans the database and creates each line of the table. See "ARRAYBUILD" on page 5038 to learn about the parameters to this statement.

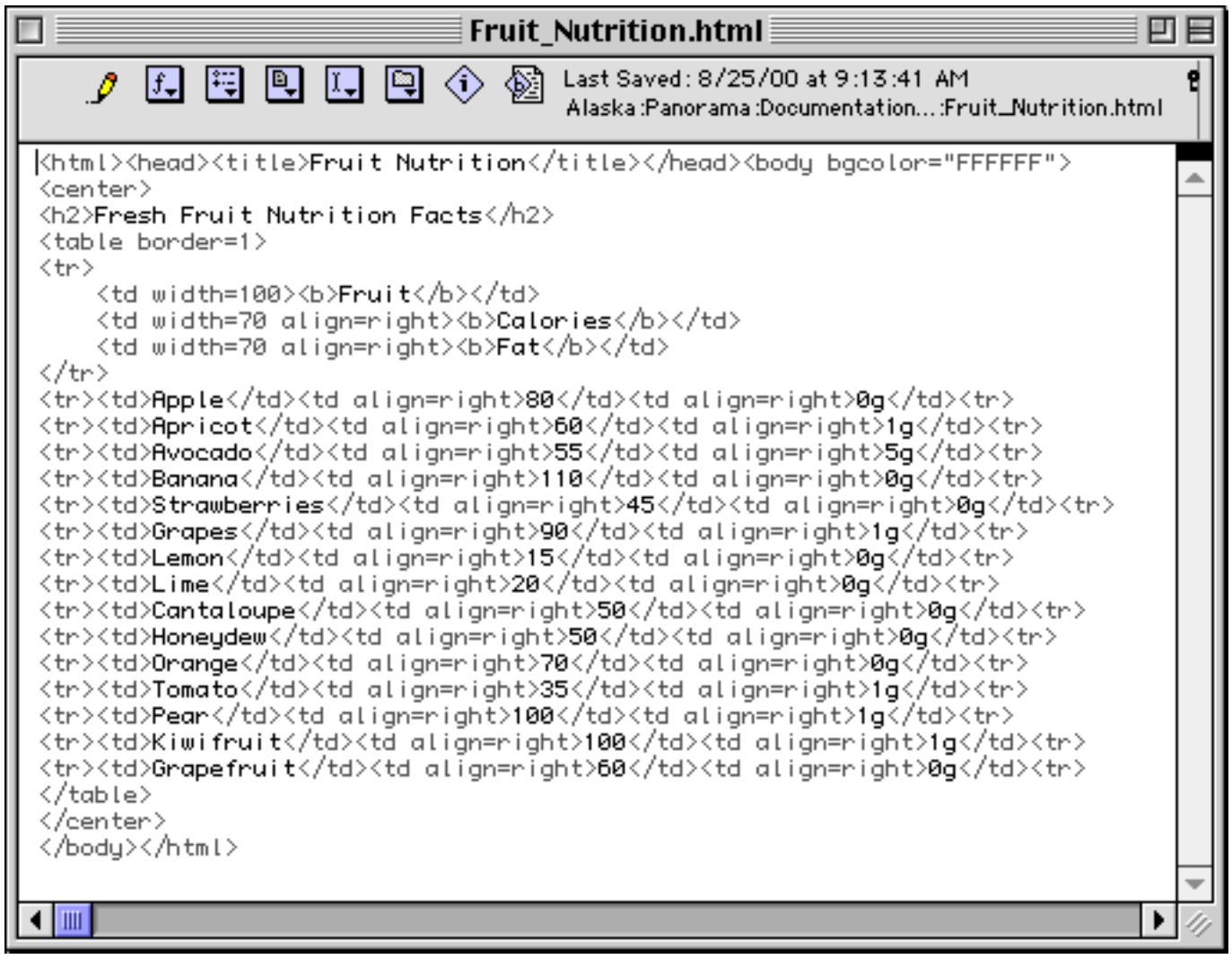
```
local webPage,webTable

arraybuild webTable,¶,"",
   "<tr><td>"+Fruit+"</td>"+
   "<td align=right>"+str(Calories)+"</td>"+
   "<td align=right>"+str(«Total Fat»)+"g</td><tr>"

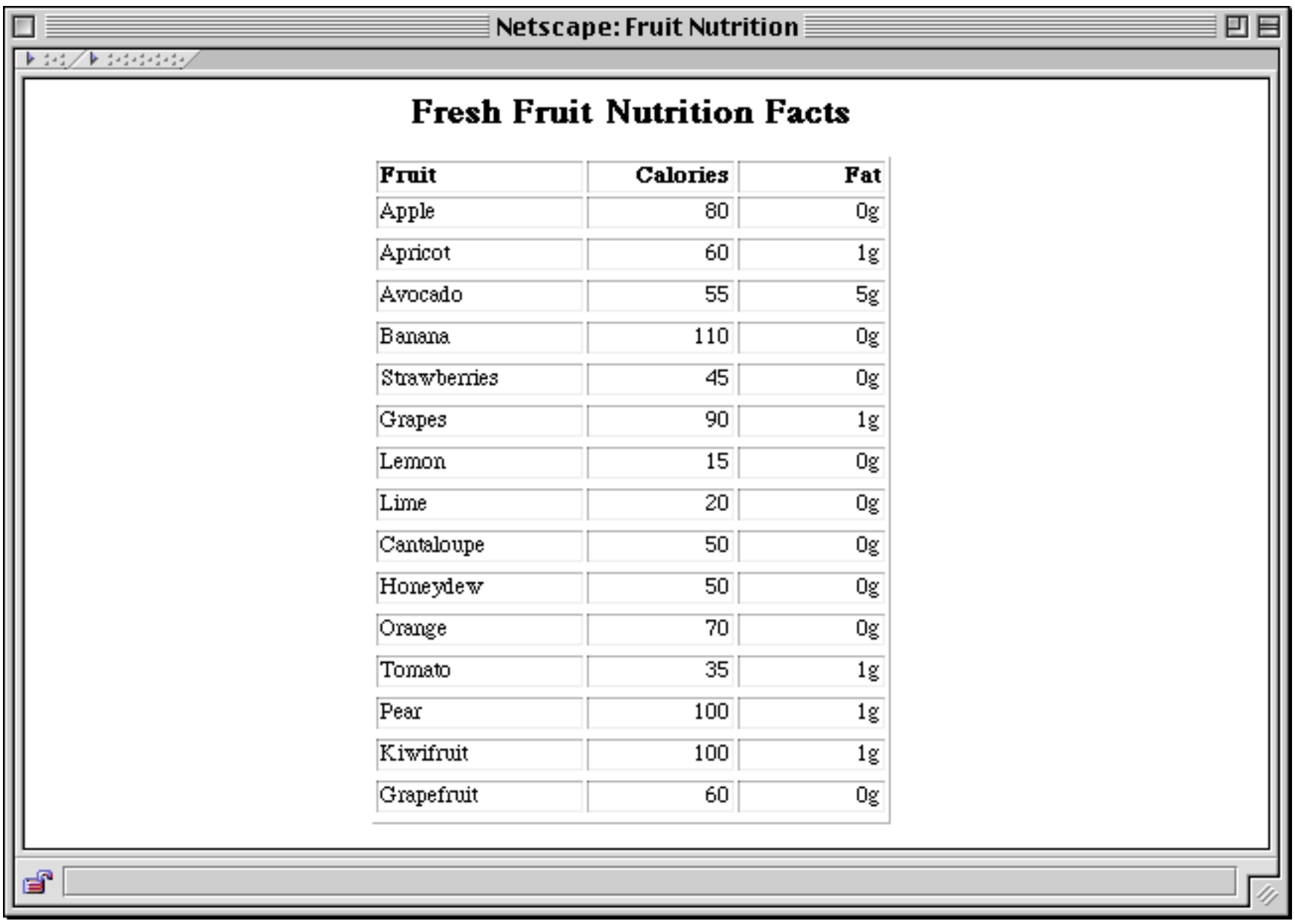
webPage={<html><head><title>Fruit Nutrition</title></head><body bgcolor="FFFFFF">
<center>
<h2>Fresh Fruit Nutrition Facts</h2>
<table border=1>
<tr>
   <td width=100><b>Fruit</b></td>
   <td width=70 align=right><b>Calories</b></td>
   <td width=70 align=right><b>Fat</b></td>
</tr>
<DATA>
</table>
</center>
</body></html>}

filesave "","Fruit_Nutrition.html","TEXTR*ch",replace(webPage,"<DATA>",webTable)
```

The middle section of the procedure places a web page template into the variable webPage. Notice how the { and } characters are used around the text so that " may be used within the text (see "Constants" on page 1195). The final line uses the replace(function to merge the table body into the web page template and then saves the file, which will look something like this.



When displayed in a web browser the finished result looks like this.



This example used the `arraybuild` statement to export the entire file. If you wanted to export only the currently selected records you would need to use the `arrayselectedbuild` statement.

Reading and Writing Resource Forks

On the Macintosh each file may consist of two separate partitions called **forks**, the **data fork** and the **resource fork**. The resource fork is normally accessed only indirectly through special statements (see "Working with Resources" on page 1560) and not through the standard file i/o statements and functions. However, sometimes it is necessary to read and write the resource fork directly (for example to copy a file you must copy both forks). To do this you must use the `resourefork` statement to switch Panorama into resource fork mode (see "RESOURCEFORK" on page 5668). In this mode all of Panorama's normal file i/o statements and functions access the resource fork instead of the data fork. To go back to normal data fork mode use the `datafork` statement (see "DATAFORK" on page 5141).

Here is a procedure that makes a copy of a file named **My File**. The copy is called Copy of My File, and includes both the data and resource forks from the original file.

```
local theOriginalFile,typecreator,data

theOriginalFile="My File"

typecreator=array(fileinfo("",theOriginalFile,2,¶)

datafork
data=fileload("",theOriginalFile)
filesave "","Copy of "+theOriginalFile,typecreator,data

resourcefork
data=fileload("",theOriginalFile)
filesave "","Copy of "+theOriginalFile,typecreator,data

datafork
```

On Windows PC systems files do not have resource forks and the `resourcefork` statement does absolutely nothing. If you want your program to work on both Mac and PC systems you must check which system you are using and only copy the resource fork if the database is on a Macintosh. Here is a revised copy of the procedure which shows one way to perform this check.

```
local theOriginalFile,typecreator,data,computerType

computerType="Macintosh"
if folderpath(dbinfo("folder","")) match "?:\*"
   computerType="Windows"
endif

theOriginalFile="My File"

typecreator=array(fileinfo("",theOriginalFile,2,¶)

datafork
data=fileload("",theOriginalFile)
filesave "","Copy of "+theOriginalFile,typecreator,data

if computerType="Macintosh"
   resourcefork
   data=fileload("",theOriginalFile)
   filesave "","Copy of "+theOriginalFile,typecreator,data
   datafork
endif
```

Erasing a File

A procedure can erase any file by using the `filetrash` statement (see "FILETRASH" on page 5239). This statement has two parameters: the folder ID (see "Folder ID's and Paths" on page 1529) and the name of the file to be erased. (If you use `""` as the folder ID, the folder containing the current database is assumed.) This procedure will erase the file Temp Data File.txt in the System folder.

```
filetrash info("systemfolder"),"Temp Data File.txt"
```

Keep in mind that the `filetrash` statement is the same as dragging the file into the trash can or recycle bin, then choosing the **Empty Trash** or **Empty Recycle Bin** command. Once a file has been deleted with this statement, you cannot get it back, so be careful.

Changing a File's Name

A procedure can change the name of any file by using the `filerename` statement (see "FILERENAME" on page 5235). This statement has three parameters: the folder ID (see "Folder ID's and Paths" on page 1529), the original name of the file, and the new name of the file. (If you use `""` as the folder ID, the folder containing the current database is assumed.) This procedure will rename the file Temp Data File.txt in the same folder as the current database, giving it the new name Permanent Data File.txt.

```
filerename "","Temp Data File.txt","Permanent Data File.txt"
```

If there is already a file named Permanent Data File.txt, this statement will not be able to rename the file.

Changing a File's Type and Creator

On the Macintosh each file has two four character tags that identify what type of file it is and what application it belongs to (see "What's in a File?" on page 1549). A procedure can change these tags by using the `filetypecreator` statement (see "FILETYPECREATOR" on page 5240). This statement has three parameters: the folder ID (see "Folder ID's and Paths" on page 1529), the name of the file, and the new tags for the file. (If you use `""` as the folder ID, the folder containing the current database is assumed.)

The procedure below examines a file that has been transferred from a PC computer to a Macintosh. Depending on the three character "extension" at the end of the filename, it converts the file into a text file, a Panorama file, or a Photoshop picture file.

```
case myfile endswith ".txt"
   filetypecreator myfolder,myfile,"TEXTttxt"
case myfile endswith ".pan"
   filetypecreator myfolder,myfile,"ZEPDKASX"
case myfile endswith ".pct"
   filetypecreator myfolder,myfile,"PICT8BIM"
endcase
```

Since PC files do not have type/creator tags the `filetypecreator` statement is simply ignored when used on a Windows PC.

Creating a New Folder

If it is necessary for a procedure to create a new folder it can do so with the `makenewfolder` statement. This statement has one parameter, the full path of the new folder. Here is an example. This example creates a folder named Project Foo.

```
makenewfolder "C:\Plans\1999\Project Foo"
```

If you want to check whether or not a folder already exists you can use the `folderexists(` function.

Getting Information about a File

Your computer keeps track of a number of attributes for each file on the disk, including the size of the file, the time and date when it was created, the time and date when it was last modified, and (on the Macintosh) the file type tag and file creator tag. Panorama has several functions for accessing this information: `fileexists(`, `filedate(`, `filetime(`, `filetypecreator(` and `fileinfo(`. Each of these functions have two parameters: the folder ID (see "Folder ID's and Paths" on page 1529) and the filename.

The `fileexists(` function returns true or false depending on whether the file exists or not.

The `filedate(` function returns the date when this file was last modified (see "Date Arithmetic" on page 1248).

The `filetime(` function returns the time when this file was last modified. This time is the number of seconds after midnight (see "Time Arithmetic" on page 1255).

The `filetypecreator(` function returns the 8 characters defining the **type tag** and **creator tag** for the file (4 characters each - see "What's in a File?" on page 1549). You can use this information to determine the type of file. For example, a Panorama database is ZEPDKASX. On PC systems Panorama will attempt to simulate the type and creator tags for some types of files based on the file's extension (.txt, .pan, etc.)

The `fileinfo(` function returns a text array with 8 items of information. The eight items are combined together in an array with carriage return separators, so you can use the `array(` function to extract the information you want.

The first element in the array returned by `fileinfo(` is the **type of item**, which may be either File or Folder.

The second element in the array returned by `fileinfo(` is 8 characters defining the **type tag** and **creator tag** for the file (4 characters each - see "What's in a File?" on page 1549). You can use this information to determine the type of file. For example, a Panorama database is ZEPDKASX. On PC systems Panorama will attempt to simulate the type and creator tags for some types of files based on the file's extension (.txt, .pan, etc.)

The third element in the array returned by `fileinfo(` is a number representing the **creation date** of the file. The formula below displays the creation date of the file named Sample.

```
datepattern(val(array(fileinfo("","Sample"),3,¶)),"Month ddnth YYYY")
```

The fourth element in the array returned by `fileinfo(` is a number representing the creation time of the file. The formula below displays the creation time of the file named Sample.

```
timepattern(val(array(fileinfo("","Sample"),4,¶)),"HH:MM:SS AM/PM")
```

The fifth element in the array returned by `fileinfo(` is a number representing the modification date of the file. The sixth element in the array returned by `fileinfo(` is a number representing the modification time of the file.

The seventh element in the array returned by `fileinfo(` is the size of the file.

The eighth element in the array returned by `fileinfo(` is the status of the file, either Locked or Unlocked.

Here is a typical array returned by the `fileinfo(` function for the file Panorama (the application itself).

```
File‹       Type of item (File or Folder)
APPLKASX‹   File type tag (APPL) and file creator tag (KASX)
2450070‹    Creation date. Use val( function to convert to number.
54233‹      Creation time. Use val( function to convert to number.
2450075‹    Modification date. Use val( function to convert to number.
71028‹      Modification time. Use val( function to convert to number.
1092657‹    File size (just over 1 megabyte)
Unlocked‹   File options (Locked or Unlocked)
```

Here is another example of information for a Panorama database file:

```
File‹       Type of item (File or Folder)
ZEPDKASX‹   File type tag (APPL) and file creator tag (KASX)
2450035‹    Creation date. Use val( function to convert to number.
401‹        Creation time. Use val( function to convert to number.
2450035‹    Modification date. Use val( function to convert to number.
923‹        Modification time. Use val( function to convert to number.
2320‹       File size (just over 1 megabyte)
Unlocked‹   File options (Locked or Unlocked)
```

Getting and Setting Additional File Information

The `getfilefinderinfo` statement (see “GETFILEFINDERINFO” on page 5296) retrieves a collection of information about a file, including when it was created and last modified and its position within the window.

```
getfilefinderinfo folderID,filename,type/creator,position,flags,creationdate,moddate
```

The first two parameters, folderID and filename, tell Panorama which file you are interested in. See “Folder ID’s and Paths” on page 1529 for more information about folder ID’s.

The next five parameters are all filled in by the statement. You should supply variables for each of these values. Type/Creator is the two four character tags that identify what type of file this is. See “What’s in a File?” on page 1549 for more information about these tags. Position is the visual x-y position of this file within the folder (see “Points” on page 1288). Flags contain a number of operating system specific options for this file. If bit 14 of this value is set then the file is invisible. CreationDate and ModDate contain the creation date/time and modification date/time of the file. Both of these values are SuperDates (see “SuperDates (combined date and time)” on page 1259).

The `setfilefinderinfo` statement (see “SETFILEFINDERINFO” on page 5741) modifies a collection of information about a file, including when it was created and last modified and its position within the window.

```
setfilefinderinfo folderID,filename,type/creator,position,flags,creationdate,moddate
```

The first two parameters, folderID and filename, tell Panorama which file you want to modify. See “Folder ID’s and Paths” on page 1529 for more information about folder ID’s.

The next five parameters specify the new values for each file option. The parameter descriptions are the same as for the `getfilefinderinfo` statement (see above). If you don’t want to change the type/creator value you can simply specify `""`. If you don’t want to change the position, flags, creationdate or moddate value specify 0. The procedure below sets the creation date/time and modification date/time to 9 am today.

```
setfilefinderinfo "","Sunset.jpg","",0,0,
   superdate(today(),time("9am")),superdate(today(),time("9am"))
```

All of the other file options (type/creator, position and flags) are left undisturbed.

Accessing and Modifying File Permissions

The MacOS X operating system is based on UNIX, and uses UNIX style file permissions. File permissions allow the system administrator to control who can read and write any file or folder. Panorama can access and modify these permissions with the `getfilepermissions` and `setfilepermissions` statements. To learn more about these statements see the **Programming Reference** wizard.

Building a List of Files or Folders

The `filecatalog` statement can build a list of files in folders and subfolders. This statement has four parameters: Path, WildCard, TypeCreator and Files.

The Path parameter is the path of the primary folder to be cataloged, for example "Computer:Alpha:Gamma:Zed:Foo". The statement will scan this folder, along with any subfolders inside that folder (and any subfolders inside the subfolders, etc.).

The WildCard parameter lets you specify for matching file names that you want to catalog. The * character will match any sequence of characters, while the ? character will match a single character. For example if you wanted to catalog only JPEG images you could use *.jpg, while if you wanted to catalog only web pages you could use *.html. The default is *, which matches all files.

The TypeCreator parameter is another method for specifying the types of files you want to catalog. For example, if this parameter is TEXT???? then only text files will be included in the final list, if this parameter is ????KASX then only files created by Panorama will be included. The default is ????????, which matches all files.

The Files parameter must be a field or variable. This is where the final list of files will be placed. Each line contains the file path and file name of a single file.

This example will create a list of all JPEG images inside the My Images folder (including subfolders), then displays that list in a dialog.

```
filecatalog "My Drive:My Documents:My Images:","*.jpg","",jpegArray
displaydata jpegArray
```

The `listfiles(` function builds a list of the files in a single folder. It is not as flexible as the `filecatalog` statement, but since it is a function it can be used to display information in forms. The list is a text array with a carriage return separator between each file name (see “Text Arrays” on page 1236). The `listfiles(` function has two parameters.

```
listfiles(folder,tags)
```

The folder parameter is a folder ID (see “Folder ID’s and Paths” on page 1529) that identifies what folder you want to list the contents of.

The tags parameter specifies what types of files you want to list. Leave this parameter empty if you want to list all files and folders regardless of type.

If you wish, you may use the tags parameter to specify one or more types of files to include in the list. Each type of file is specified by an 8 character combination of type and creator tags (see the previous section). For example, to list Panorama database files the tags parameter should be ZEPDKASX. You may use the ? character in the tags parameter when you don’t need to match. For example, many different applications can create text files. To list all text files no matter what application created them, use the tags parameter TEXT????. You can combine multiple tag specifications to list more than one type of file, for example ZEPDKASXTEXT???? to list both Panorama databases and text files.

The `listfiles(` function does not normally include folders in the list of files. However, there are two cases where folders will be listed: 1) if the tags parameter is empty, or 2) if the tags parameter starts with the ~ character (see “Special Characters” on page 1202). For example, to list Panorama databases and folders use the tags parameter ~ZEPDKASX. If you want to list only folders without any files, use type creator tags that are not used by any kind of file, for example ~ZZZZZZZZ.

Since `listfiles(` is a function, it can be used in any formula in a procedure, auto-wrap text object, or SuperObject formula. Here is an example formula that lists all the picture files in the same folder as the current database:

```
listfiles(dbinfo("folder",""),"PICT????")
```

Building a List of Disks (Volumes)

The `info("volumes")` functions creates a list of disks (volumes) that are currently mounted (active). The list is a text array with a carriage return separator between each file name (see “Text Arrays” on page 1236). The example below uses this function to check to see if the World Facts Reference CD is currently available.

```
if 0=arraysearch(info("volumes","World Facts Reference",1,¶)
   message "World Facts Reference CD is not currently available."
endif
```

Working with Resources

On the Macintosh files are split into two partitions, called **forks**. These forks are the **data fork** and the **resource fork**. The data fork corresponds to a normal file as used on other operating systems (Windows, UNIX, etc.) The resource fork, however, is not handled like a normal file. Instead, the operating system further subdivides this fork into components called **resources**. These resources are like miniature "files within a file" and are used to hold objects needed by programs like menus, images, text, templates, and even program code. Instead of accessing the resource fork directly as a file, programs use the operating system to access these components. Each resource component may be anything from a single character to tens of thousands of bytes of information.

Because resources play such an important part in the operation of Macintosh programs (including Panorama) we have created a mechanism by which resources can be used on Windows PC systems as well. Since Windows PC files do not have two forks the resources must be kept in a separate file. This file must have a name ending with the extension **.rsr**, for example My Menus.rsr. Unless specified otherwise, all of the functions and statements described in this section work equally well on both the Macintosh and the PC.

Just as a file is identified by it's location (folder) and filename, each resource is identified by its **type** and **ID number**. The **type** is a four letter designation that identifies what type of data is stored in that resource. There are hundreds of different types of resources, with more new types being created all the time. However, the most common types were defined by Apple in 1984 and are still in use today. This table describes some of the most common types.

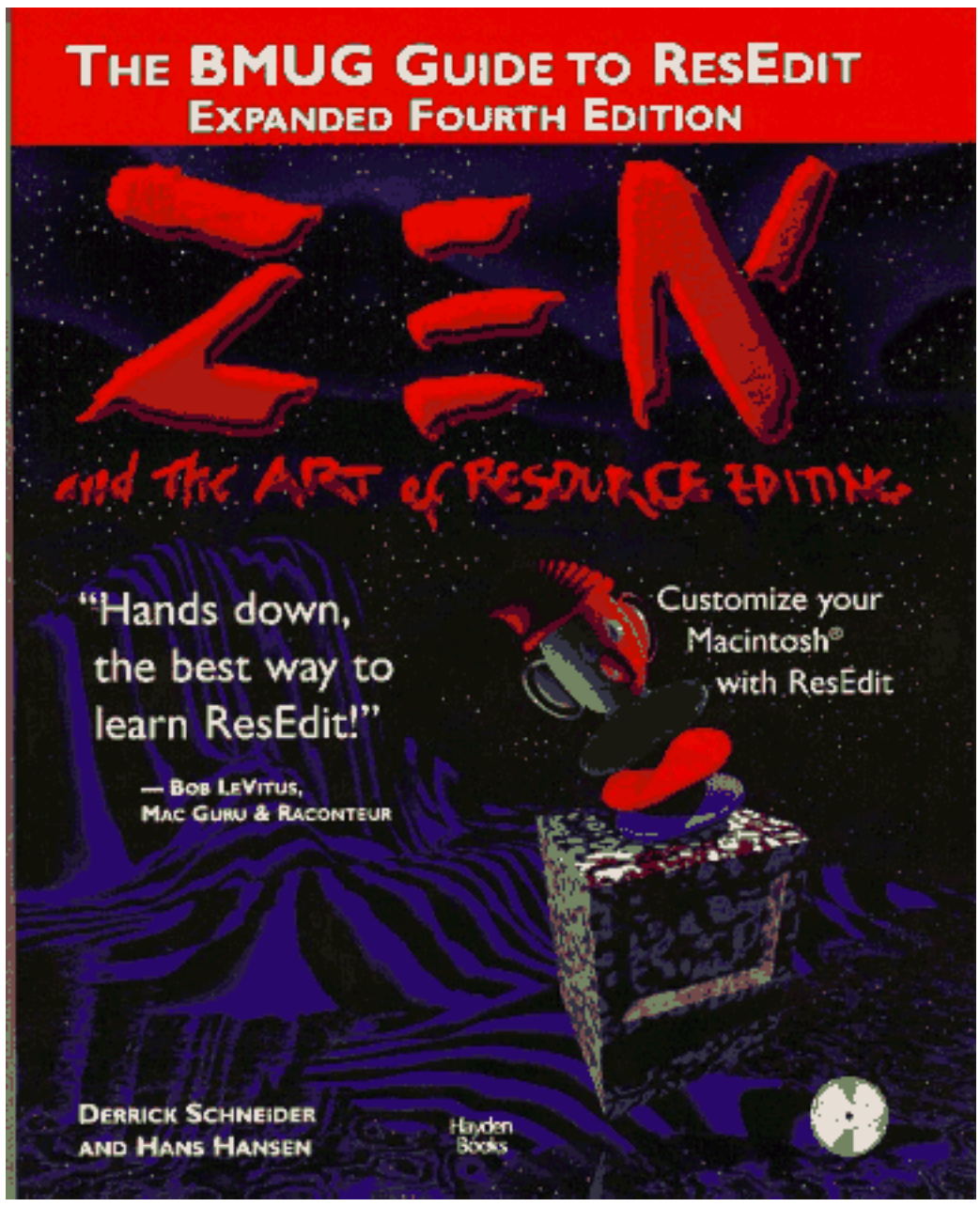
Type	Contents
CODE	Machine code (a program)
MENU	List of items in a menu
STR	A single item of text
STR#	Multiple items of text
DLOG	Template describing a dialog
DITL	List of items within a dialog
PICT	Picture
ICON	A single icon
ICN#	Multiple icons
cicn	Color icon
CURS	Cursor design (mouse pointer)

The **resource ID** is simply a number between 0 and 65535.

Just as a file is identified by its folder and file name, a resource is identified by its type and ID. For example, you may refer to a resource as MENU 97 or ICON 2544.

In addition to a type and ID, a resource may also have a **name**. However, the name is completely optional. If a resource does have a name, you can identify the resource by its type and name as well as by its type and ID. For example you may refer to a resource as ICON 2544 or as ICON Empty Trash Can.

On the Macintosh resource files may be created and edited with resource editing programs. The most popular such program is **ResEdit**, a freeware utility written by Apple and distributed by the Apple Programmers and Developers Association (APDA). ResEdit has appeared in several different versions since the Macintosh was released in 1984. If you'd like to learn more about **ResEdit**, we recommend that you get Zen and the Art of Resource Editing, available from many sources, including **amazon.com**. The book includes a CD with a copy of ResEdit along with many example files.



If you do a lot of resource editing you might want to check out another resource editor, **Resorcerer**, from Mathemaesthetics (http://www.mathemaesthetics.com/).



Unlike ResEdit, Resorcerer is not free, but it does have advanced features that are not included in ResEdit. We use Resorcerer instead of ResEdit here at ProVUE.

There are no resource editor programs available for Windows PC's. However, you can still create and modify resources using Panorama statements and functions. Panorama includes an editor for creating and modifying menu resources on both Macintosh and PC systems ("Preparing a Resource File" on page 1479).

Opening and Closing Resource Files

Before a Panorama procedure can access the objects inside a resource file it must open the resource file. This can be done with the `openresource` or `openresourcerw` statements.

The `openresource` statement opens a resource for read only access — you cannot modify resource objects when a file is opened this way (see "OPENRESOURCE" on page 5580). The `openresource` statement requires one parameter—the name of the resource file to open. For example, if the resource file containing your text is called Background Items then the procedure should contain the statement

```
openresource "Background Items"
```

This statement may be used on both Macintosh and Windows PC computers. On Windows PC computers the filename extension of **.rsr** is assumed, so the example above will actually open the resource file Background Items.rsr if used on a PC computer.

If the resource file is not in the same folder as the current database you must specify the location as well as the name of the resource file, like this (see "Combined Folder Location and File Name" on page 1528).

```
openresource "C:\Accounting\Background Items"
```

The `openresourcerw` statement also opens a resource file, but allows both reading and writing (see "OPENRESOURCERW" on page 5581).

Unlike opening a database window, there is no visible indication when a resource file is opened.

To close a resource file use the `closeresource` statement (see "CLOSERESOURCE" on page 5111). The statement must be followed by the name of the resource to close.

```
closeresource "Background Items"
```

If the resource was opened from a different folder you must specify the entire path, like this.

```
closeresource "C:\Accounting\Background Items"
```

It's often not necessary to bother with closing a resource file. If you leave any resource files open Panorama will automatically close them when you exit (**Quit**) from Panorama. If you attempt to re-open a resource file that is already open Panorama simply leaves it open and continues. (However, if a resource is open for reading only you must close it if you want to open it for read / write access.)

It is possible to open more than one resource file at once. In fact, there is always more than one resource file open, because the system has a resource file open, and Panorama has its own resource file open. If two different resource files contain resources with the same type and ID, Panorama will use the copy of the resource from the most recently opened resource file.

To get a list of currently open resource files use the `info("openresourcefiles")` function.

Opening a Resource File in the .Initialization Procedure

If a resource file is required for operation of the database (for example for custom menus, see "Custom Menus (Resource Based)" on page 1478), we recommend that you place the `openresource` statement in the **.Initialize** procedure for the file (See ".Initialize" on page 1511 more information on the **.Initialize** procedure).

Simply creating the **.Initialize** procedure does not open the resource file. The first time you create this procedure you must save the database, then close and re-open the database. The **.Initialize** procedure will open the resource file when you re-open the database, and you can begin using the resources immediately. From then on the resource file will be opened automatically every time the database is opened.

Reading a Resource

The `getresource(` function gets a resource from an open resource file and copies it into a field or variable. You can read any resource with this function, although making sense of the contents of the resource is up to you.

The `getresource(` function has two parameters: type and ID. The type is the resource type. This must be a four letter text item (see "Working with Resources" on page 1560). Standard resource types include `"STR "` (Pascal String), `"STR#"` (multiple strings), `"DLOG"` (dialog template), `"DITL"` (dialog items), `"MENU"` (menu). The ID is the identification for the resource. The resource ID can be a number (from 0 to 65535) or a name (a text item).

This example loads the contents of TEXT resource number 415 into the field LetterBody.

```
openresource "Letter Templates"
LetterBody=getresource("TEXT",415)
```

Remember, all resource have numbers, but they do not all have names. If the resource does have a name, you can use the name for the ID. This example loads the contents of the TEXT resource named Thank You #2 into the field LetterBody.

```
openresource "Letter Templates"
LetterBody=getresource("TEXT","Thank You #2")
```

Reading STR and STR# Resources

Panorama has three special functions for reading with string resources: `getstring(`, `getnstring(` and `getstringmatch(`.

To read a `STR` resource (which contains one text item up to 255 characters long), use the `getstring(` function (see "GETSTRING(" on page 5314). This function has two parameters: type and ID. Type is the resource type. This must be a four letter text item (see "Working with Resources" on page 1560). You can specify any resource type you like here, but strings are usually stored in resources of type `"STR "` (Pascal String). (If you specify `""` for the type, Panorama will assume `"STR "`.) The ID parameter is the identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item).

This example displays the contents of STR resource number 1296.

```
message getstring("",1296)
```

All resource have numbers, but they do not all have names. If the resource does have a name, you can use the name for the ID. This example displays the text in the resource named Overflow Error.

```
message getstring("","Overflow Error")
```

To access a `STR#` resource(which contains multiple text items, each up to 255 characters), use the `getnstring(` function (see "GETNSTRING(" on page 5307). `STR#` resources hold multiple text items, so the `getnstring(` function extracts one of them. This function has three parameters: type, ID and number. The type is the resource type. This must be a four letter text item. You can specify any resource type you like here, but strings are usually stored in resources of type `"STR#"` (multiple Pascal Strings). (If you specify `""` for the type, Panorama will assume `"STR#"`.) The ID is the identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item). The number is the number of the string item within the collection. For example, if the collection contains 6 strings they will be numbered 0, 1, 2, 3, 4, and 5.

This example displays the contents of STR# resource #693 item 12.

```
message getnstring("",1296,11)
```

Here is another example that displays the 12th item in the Errors STR# resource.

```
message getnstring("","Errors",11)
```

Another function that access STR# resources is the getstringmatch(function (see "GETSTRING-MATCH(" on page 5315). The getstringmatch(function searches through a collection of multiple strings in a STR# resource. If it finds a match with the text you supply, it returns the number of the text item within the collection.

The getstringmatch(function has three parameters: type, ID and text. The type and ID are the same as for the getnstring(function (see above). The text parameter is the text you want to search for. For a match, this text must be exactly the same as one of the text items in the STR# collection.

This function returns a number. If the text does not match any of the text items in the STR# collection, the function will return 0. If there is a match, the function will return the number of the item that matched, starting with 1 for the first item. (Notice that this numbering system is different than the getnstring(function, which starts with 0 for the first item.)

One application for this function is looking up commands or keywords. Suppose you have a STR# resource number 320 that contains the following text items.

```
DIAL
APPOINTMENT
TODO
LETTER
CHECK
```

Now suppose the database has a global variable called CommandLine. The user types a command into this variable with a Text Editor SuperObject™. Here is part of a procedure that can process these commands using the STR# 320 resource.

```
local commandWord, commandNumber, commandExtras
openresource "Accounting Extras"
commandWord=upper(strip(CommandLine[1," "]))
commandExtras=strip(CommandLine[" ",-1])
commandNumber=getstringmatch("",320,commandWord)
if commandNumber=0 stop endif
if commandNumber=1
      dial commandExtras
endif
if commandNumber=2
   …
endif
if commandNumber=3
   …
   …
```

Notice that the example converts the text the user types in into all upper case. This is to make sure that the text will match the commands in the STR# resource. Remember, the text must match exactly, including upper and lower case.

Writing a Resource

If a resource file has been opened with the openresourcerw statement (see "OPENRESOURCERW" on page 5581) a procedure can use the writeresource statement to create and/or modify a resource object (see "WRITERESOURCE" on page 5908). The writeresource statement has three parameters: type, ID and data. The type is the resource type. This must be a four letter text item (see "Working with Resources" on page 1560). The ID is the identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item). The data parameter is the data to be placed in the resource. The data should be text, not a number. (The text may represent a binary value, see "Raw Binary Data" on page 1296.)

Here is a procedure that writes some text (for example Last update: 10/18/02) into STR resource number 2000. If the resource does not exist it will be created.

```
writeresource "STR ",2000,string255("Last update: "+datepattern(today(),"mm/dd/yy")
```

If there is more than one resource file open the resource will be written into the file that was most recently opened (see “Working with Multiple Resource Files” on page 1567).

Deleting a Resource

If a resource file has been opened with the `openresourcerw` statement (see “OPENRESOURCERW” on page 5581) a procedure can use the `deleteresource` statement to permanently remove a resource (see “DELETERESOURCE” on page 5163).

```
deleteresource type,id
```

The type is the resource type. This must be a four letter text item (see “Working with Resources” on page 1560). The ID is the identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item).

Renumbering a Resource

If a resource file has been opened with the `openresourcerw` statement (see “OPENRESOURCERW” on page 5581) a procedure can use the `renameresource` statement to change the number of a resource and / or change it’s name (see “RENAMERESOURCE” on page 5659).

```
renameresource type,id,number,name
```

The type is the resource type. This must be a four letter text item (see “Working with Resources” on page 1560). The ID is the original identification for the resource. The resource id can be a number (from 0 to 65,535) or a name (a text item). The number is the new `number` for the resource. The name is the new name for the resource. This procedure changes resource MENU 498 into MENU 1917, and gives the resource the name Option Menu.

```
renameresource "MENU",498,1917,"Option Menu"
```

If the name is `""` Panorama will leave the name alone (no change). This procedure changes resource MENU 498 into MENU 8367 while leaving the name alone (as Option Menu in this case).

```
renameresource "MENU",1917,8367,""
```

If you want to change the name to empty text then use ¶ as the menu name (see “Special Characters” on page 1202). This procedure erases the name from resource MENU 8367.

```
renameresource "MENU",8367,8367,¶
```

Listing Resources

Panorama has two functions that can help you build a list of the resources available in the currently open resource files: `resourcetypes(` and `resources(`.

The `resourcetypes(` function creates a text array containing a list of the resource types in the currently open resource files (see “RESOURCETYPES(” on page 5670). The function has no parameters.

The `resourcetypes(` function returns a carriage return delimited text array (see “Text Arrays” on page 1236). Each element in the array contains a resource type. Each resource type is a four letter text item, for example `"STR "` (Pascal String), `"STR#"` (multiple strings), `"DLOG"` (dialog), `"DITL"` (dialog items), `"MENU"` (menu) (see “Working with Resources” on page 1560).

You can use this function to check if a particular resource type exists, or you can use the function with a pop-up menu or List SuperObject™ to allow the user to select a type of resource for any reason. The procedure below will create a text array with resource types.

```
local rezTypes
rezTypes=resourcetypes()
```

The rezTypes variable will be filled with a list of resource types, like this:

```
CNTL
CURS
FKEY
INIT
KCAP
KCHR
LDEF
MACA
PACK
PTCH
ROv#
TPLT
SIZE
LBAR
octb
DLGX
dctb
cocm
TEXT
STR#
PICT
PAT#
PAPR
MENU
MDEF
```

As you can see, the resource types are not listed in any particular order.

The `resources(` function creates a text array containing a list of resources of a particular types (see "RESOURCES(" on page 5669). This function has one parameter: `type`, which is the resource type. This must be a four letter text item (see "Working with Resources" on page 1560).

The `resources(` function returns a text array containing a carriage return delimited list of all the resources of the specified type. Each element of this list is itself a tab delimited array. The first item is the resource item number. The second item is the resource name (if any).

This example builds a list of the `TEXT` resources in the currently open resource files. (The currently open resource files include Panorama itself and the Macintosh system file, as well as any resource files you have opened with the `openresource` statement.)

```
local rezStrings
rezStrings=resources("TEXT")
```

This will fill rezStrings with an array like this. (There is a tab between the number and the first character of the resource name, if any.)

```
2001Error Messages
2002Command List
2003Conversion Options
2100
2103
2104
2140Day
2141Month
2142Year
```

The `resourcetypes(` and `resources(` functions normally list all resources in all open resource files. See the next section to learn how to make these function list only the resources in a single file.

Working with Multiple Resource Files

It's possible to open and work with multiple resource files at once. For example you could open three resource files like this.

```
openresourcerw "alpha"
openresourcerw "beta"
openresourcerw "gamma"
```

When reading resources Panorama always searches the most recently opened file first. For example, if a procedure contained the statement

```
temp=getresource("DATA",2000)
```

Panorama would start by searching for this resource object in the gamma resource file. If it didn't find it there it would look in the beta resource file, and if not there then it would look in the alpha file.

When writing resources Panorama always writes in the most recently opened file. For example if a procedure contained the statement

```
writeresource "DATA",2000,"This is not a test"
```

Panorama would always write this resource in the gamma resource file.

Sometimes it may be necessary to focus in on only a single resource file, temporarily disabling the other open files. This can be done with the `activeresource` statement (see "ACTIVERESOURCE" on page 5011). This statement temporarily makes only one resource file active. For example, the program below will write a resource into the beta resource file instead of the gamma file.

```
activeresource "beta"
writeresource "DATA",2000,"This is not a test"
activeresource ""
```

The second `activeresource` statement (with the `""` parameter) re-enables the other resource files.

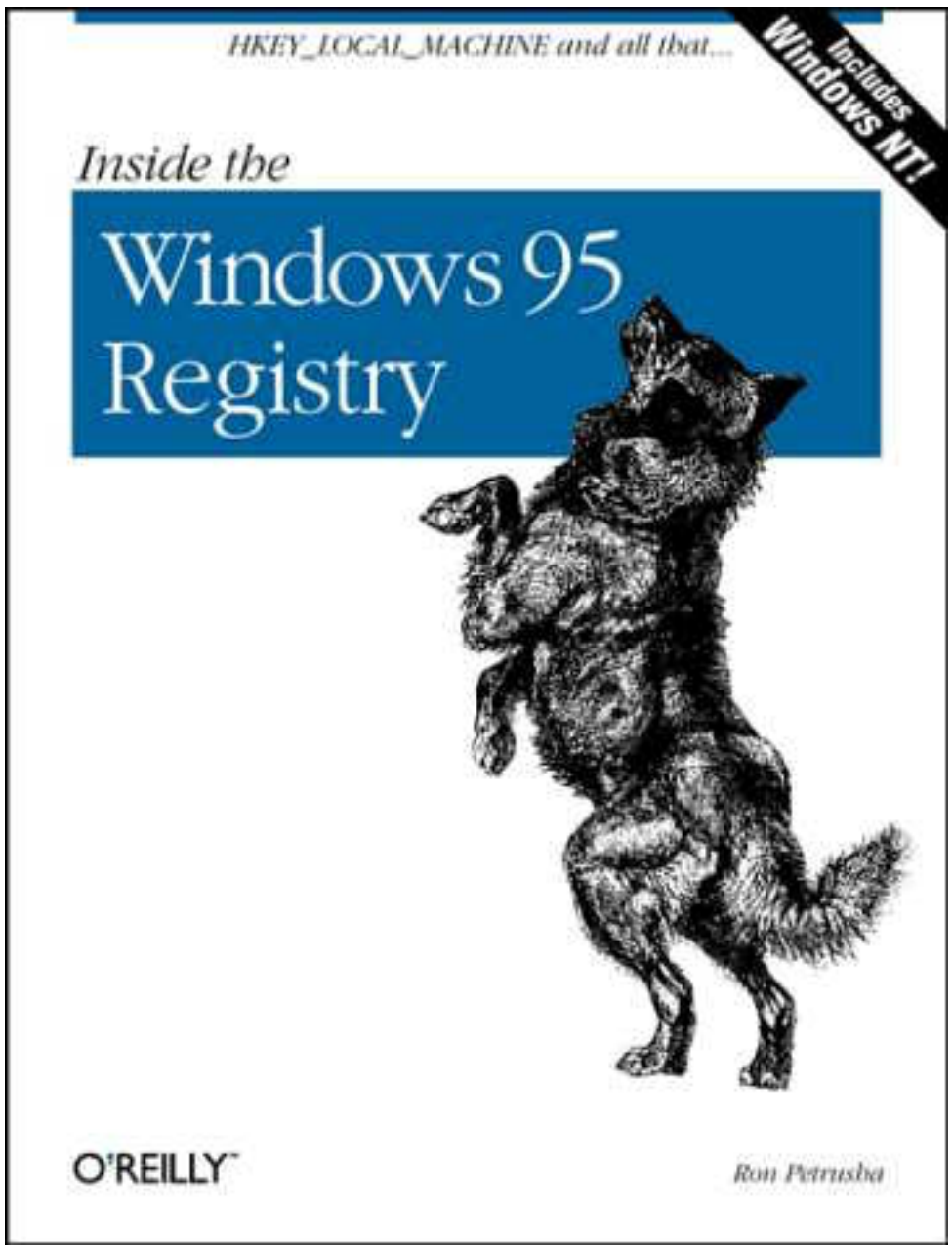
The `activeresource` statement can also be used for reading resources. The program below will only read the DATA 2000 resource from the beta file, not alpha or gamma.

```
activeresource "beta"
temp=getresource("DATA",2000)
activeresource ""
```

Finally, the `activeresource` statement can also be used when listing resources. The `resourcetypes(` and `resources(` functions normally list all resources in all open resource files (see "Listing Resources" on page 1565). When an `activeresource` statement is in use they will only list resources and resource types in the active resource file, ignoring the other files.

Accessing the Windows Registry

The **Registry** is a master database that Windows uses as kind of a giant system-wide preferences file. If you'd like to learn about what's in the registry and how to work with it, I recommend that you pick up a copy of "Inside the Windows 95 Registry" by Ron Petrusha.



Be sure you know what you are doing before you mess with the Registry. You can easily disable your system beyond repair if you are not careful.

Getting Information About Registry Items

The `registryinfo(` function allows you get a directory of subkeys, a directory of values within a registry key, or a specific value within a key (see "REGISTRYINFO(" on page 5640). For example, this formula returns a list of control panels.

```
registryinfo("HKEY_CURRENT_USER\Control Panel")
```

To get a directory of the values contained in a key (instead of the subkeys), add a colon to the end of the path. This example will list three values: MouseSpeed, MouseThreshold1, and MouseThreshold2.

```
registryinfo("HKEY_CURRENT_USER\Control Panel\Mouse:")
```

To retrieve a specific value, add the value name to the end of the path.

```
registryinfo("HKEY_CURRENT_USER\Control Panel\Mouse:MouseSpeed")
```

To get the default value for the key, use the name <DEFAULT>. This example will tell you that a **.aif** file is a **QuickTime** movie file (if you have QuickTime installed).

```
registryinfo("HKEY_CLASSES_ROOT\*\.aif:<DEFAULT>")
```

Panorama allows the six root keys to be abbreviated, as shown in the table below.

Root	Abbreviation
HKEY_CLASSES_ROOT	HKCR
HKEY_CURRENT_USER	HKCU
HKEY_LOCAL_MACHINE	HKLM
HKEY_USERS	HKUS
HKEY_CURRENT_CONFIG	HKCC
HKEY_DYN_DATA	HKDD

Using these abbreviations the examples given previously above could be rewritten as shown below:

```
registryinfo("HKCU\Control Panel")
registryinfo("HKCU\Control Panel\Mouse:")
registryinfo("HKCU\Control Panel\Mouse:MouseSpeed")
registryinfo("HKCR\*\.aif:<DEFAULT>")
```

Modifying Registry Entries

The `registrywrite` statement allows you to create registry keys and registry values, or to modify an existing registry value. This statement has three parameters.

```
RegistryWrite path,type,data
```

The first parameter is a registry path. This path uses the same format as the `registryinfo(` function (see “Getting Information About Registry Items” on page 1568).

The second parameter is the type of data being written. The possible choices are shown below. If you specify `""`, Panorama will default to REG_SZ (text).

0	REG_NONE
1	REG_SZ
2	REG_EXPAND_SZ
3	REG_BINARY
4	REG_DWORD
5	REG_DWORD_BIG_ENDIAN
6	REG_LINK
7	REG_MULTI_SZ
8	REG_RESOURCE_LIST
9	REG_RESOURCE_LIST
10	REG_RESOURCE_REQUIREMENTS_LIST

The third parameter is the actual data being written. No matter what data format you are writing, this should be text. For other data types you can fill the text item with binary values (see “Raw Binary Data” on page 1296).

This example changes the mouse speed. Notice that this statement supports the same abbreviations allowed by the `registryinfo(` function (see “Getting Information About Registry Items” on page 1568).

```
registrywrite "HKCU\Control Panel\Mouse:MouseSpeed","","2"
```

You can also change the default value associated with a registry key.

```
registrywrite "HKCR\*\.aif:<DEFAULT>","","Quick Time Movie"
```

This example creates a registry entry named Acme, but does not create or modify any values associated with that key.

```
registrywrite "HKLM\Software\Acme","",""
```

Deleting a Registry Entry

The `registrydelete` statement may be used to delete a registry key or registry value (see “REGISTRYDELETE” on page 5639). This statement has one parameter, the path of the registry item to be deleted (see “Getting Information About Registry Items” on page 1568).

This example deletes a registry value:

```
registrydelete "HKLM\Software\Acme\SuperWidget:WindowLocation"
```

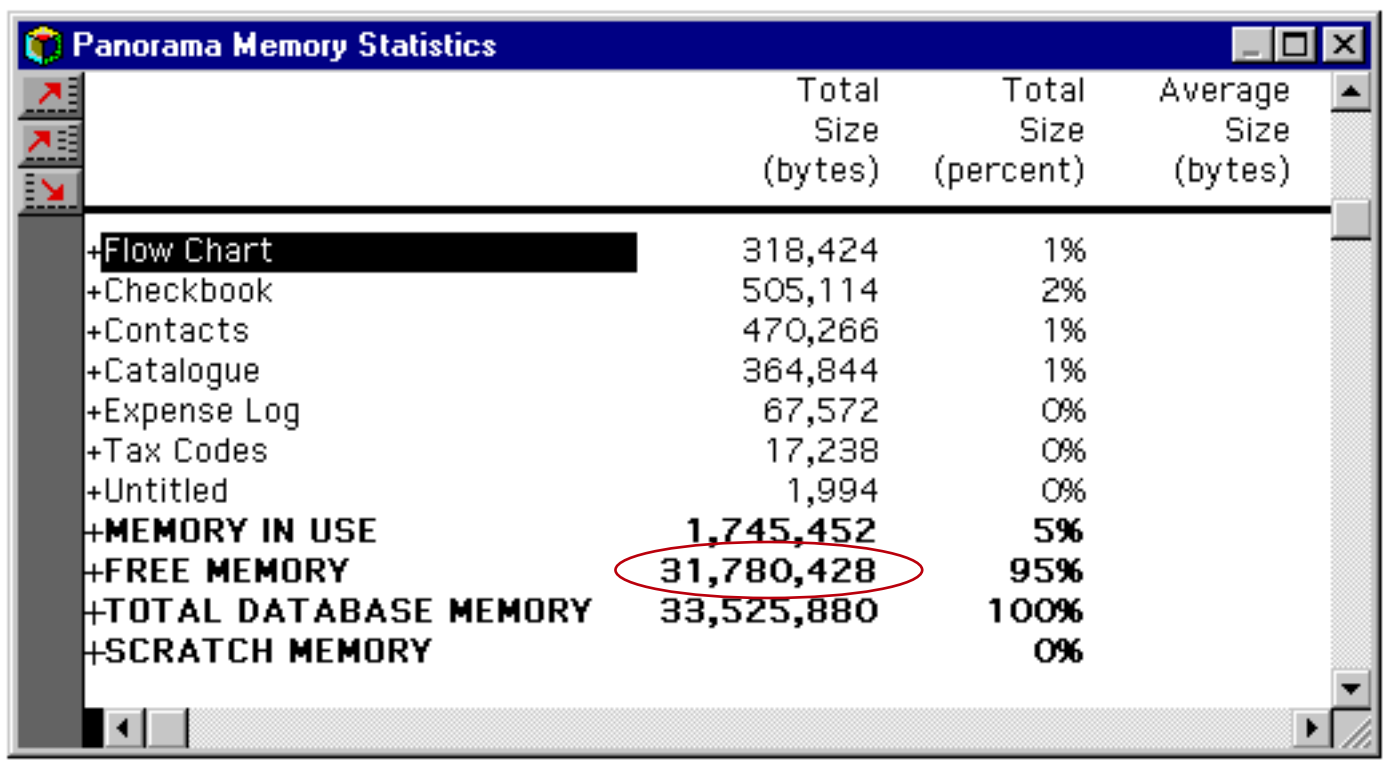
This example deletes a registry key, along with any values associated with it:

```
registrydelete "HKLM\Software\Acme\SuperWidget"
```

Monitoring Memory Usage

Since Panorama is a memory based program memory usage can be very important. Several statements allow you to monitor memory usage.

The `info("freememory")` function returns the amount of memory available for additional data (see "INFO("FREEMEMORY")" on page 5382). This corresponds to the value shown by the **Memory Usage** window (see "Monitoring Memory Usage" on page 186).



To learn how to change Panorama's overall memory allocation see "Adjusting Panorama's Memory Allocation (Windows and OS X)" on page 189 or "Adjusting Panorama's Memory Allocation (Macintosh OS 9)" on page 190.

The `info("scratchmemory")` function returns the amount of scratch memory available. On Macintosh OS X and Windows PC systems this function always returns 1,000,000 (one million).

Changing the Scratch Memory Allocation

On Macintosh OS 9 systems Panorama's Scratch Memory allocation can be modified manually using a dialog (see "Changing Scratch Memory Size (Macintosh OS 9)" on page 192). It can also be modified using the `scratchmemory` statement (see "SCRATCHMEMORY" on page 5703). This statement has one parameter, the number of bytes to be allocated for scratch memory. This example allocates about 2.5 megabytes for scratch memory.

```
scratchmemory 2500000
```

To temporarily modify the scratch memory allocation use the `tempscratchmemory` statement. When this statement is used the new scratch memory allocation only lasts until you exit from Panorama (or change the allocation again). The next time Panorama is launched the original scratch memory allocation will be used.

Windows

Windows are where the action is. Except for the menu bar, everything Panorama does happens inside of windows. A procedure can open windows, close them, move them around and control their appearance.

Opening a Window

Opening a window in a procedure generally requires at least two statements. The first statement sets up the location of the window on the screen, along with any window options. The second statement actually opens the window.

There are six primary statements that open a new window: `opensheet`, `openform`, `opendialog`, `opencrosstab`, `openprocedure` and `opendesignsheet`. The procedure can specify exactly where the new window will open (see the next section) or it can simply allow Panorama to open the window using the default location and size.

The `opensheet` statement opens the data sheet for the current database in a new window (see "OPENSHEET" on page 5583). If the data sheet window is already open, that window will simply be brought to the front (no new window will be opened).

The `openform` statement opens a form in a new window (see "OPENFORM" on page 5577). The specified form must be a form in the current database, not some other database. The `openform` statement has one parameter, the name of the form to open. The procedure below opens a form in a new window, prints a report, then closes the new window.

```
openform "Monthly Report"
field PayTo
groupup
field Amount
total
print dialog
removesummaries 7
closewindow
```

If the specified form is already open in a window, that window will simply be brought to the front (no new window will be opened).

A procedure can use the `dbinfo(` function to find out what forms (or crosstabs or procedures) are in a database (see "DBINFO(" on page 5150). The procedure below opens up every form in the current database.

```
local windowCount,formCount,nextForm
windowCount=arraysize(listwindows(""),¶)
formCount=arraysize(dbinfo("forms",""),¶)
if windowCount+formCount>23
   message "Too many forms, cannot open them all"
   formCount=23-windowCount
endif
nextForm=1
loop
   openform array(dbinfo("forms",""),nextForm,¶)
   nextForm=nextForm+1
until nextForm>formCount
```

The `opendialog` statement also opens a form in a new window (see "OPENDIALOG" on page 5570). However, this statement opens the window without any drag bar, tool palette, or scroll bars. In other words, the new window will look (and act) like an old style, non-movable dialog window. Since this type of dialog is now obsolete we recommend that you avoid the `opendialog` statement. When a form is opened with the `opendialog` statement, Panorama will not allow any other window to be moved on top of the dialog window. Panorama simply ignores clicks in other windows, as well as clicks on the desktop. The only way to close this window is with the `closewindow` statement (see "CLOSEWINDOW" on page 5113). The form

should have at least one button that triggers a procedure that will close the window.The example below opens a dialog window, then pauses. In this case the window should contain at least one button that has a `resume pState` statement in it, and it should have SuperObjects™ for editing the PrintStartDate and PrintEndDate global variables.

```
global pState
setwindowrectangle rectangle(100,150,300,450),""
opendialog "Print Options"
pause pState
closewindow
select Date>=PrintStartDate and Date≤PrintEndDate
print dialog
```

For information on a better way to create dialogs with Panorama forms see “Custom Dialogs” on page 1616.

The `opencrosstab` statement opens a crosstab in a new window (see “OPENCROSSTAB” on page 5568). The specified crosstab must be a crosstab in the current database, not some other database. The `opencrosstab` statement has one parameter, the name of the crosstab to open.

The `openprocedure` statement opens a procedure in a new window, so that the procedure can be edited (see “OPENPROCEDURE” on page 5579). The procedure must be a procedure in the current database, not some other database. The `openprocedure` statement has one parameter, the name of the procedure to open.

The `opendesignsheet` statement opens the design sheet for the current database in a new window (see “OPENDESIGNSHEET” on page 5569). If the design sheet window is already open, that window will simply be brought to the front (no new window will be opened). Once the design sheet is open the procedure can modify the structure of the database (be careful!). A procedure can also modify the structure of the database with the `fieldname`, `fieldtype`, `addfield`, `insertfield` and `deletefield` statements (see “Modifying Field Structure Directly” on page 1633).

Specifying the New Window Location

There are three statements that can define the location and options for a new window: `setwindowrectangle`, `setwindow` and `windowbox`. The `setwindowrectangle` statement (see “SETWINDOWRECTANGLE” on page 5755) has two parameters: a rectangle defining the location of the new window (relative to the upper left hand corner of the main screen, see “Rectangles” on page 1290), and the window options. (The window options will be discussed in the next section, but if you use `""` you will get a standard window.)

Here is a procedure that opens a 100 by 200 pixel form window in the upper left hand corner of the main screen.

```
setwindowrectangle rectangle(25,2,125,202),""
openform "Status"
```

If you want the new window to fill the entire screen you can use the `info("screenrectangle")` function (see “INFO("SCREENRECTANGLE")” on page 5414), like this:

```
setwindowrectangle info("screenrectangle"),""
openform "Status"
```

The `setwindowrectangle` statement will allow you to open the window partially or completely off the screen, where it won’t be visible. If you don’t want that to happen you can use the `fitwindow` statement in between the `setwindowrectangle` statement and the statement that opens the new window (`openform`, `opensheet`, etc.). The `fitwindow` statement adjusts the new window position to make sure that it is com-

pletely visible on the screen (see "FITWINDOW" on page 5253). The procedure below positions the Status window in the lower right hand corner of an XGA (1024 by 768) monitor. But on an older 640 by 480 monitor the window would be partially off the screen. The `fitwindow` statement adjusts the window location so it will be completely visible even on these small monitors.

```
setwindowrectangle rectangle(568,624,768,1024),""
fitwindow
openform "Status"
```

You'll often want to center a new window on top of the current, window, or center a new window on the screen. The subroutine procedure below, called **.CenterRectangle**, is handy for these situations.

```
local newRect,oldHeight,oldWidth,newHeight,newWidth
local newTop,newLeft
oldHeight=rheight(parameter(1))
oldWidth=rwidth(parameter(1))
newHeight=parameter(2)
newWidth=parameter(3)
newTop=max(20,rtop(paramter(1)+int((oldHeight-NewHeight)/2))
newLeft=rleft(parameter(1)+int((oldWidth-newWidth)/2)
newRect=rectanglesize(newTop,newLeft,newHeight,newWidth)
setparameter 1,newRect
```

Once you've added the **.CenterRectangle** procedure to your database you can use it to open windows. Here's a procedure that centers a new 200 pixel high by 450 pixel wide window in the middle of the screen.

```
local myWindowBox
myWindowBox=info("screenrectangle")
call .CenterRectangle,myWindowBox,200,450
setwindowrectangle myWindowBox,""
openform "Schedule"
```

This next procedure will center the new window in the middle of the current window.

```
local myWindowBox
myWindowBox=info("windowrectangle")
call .CenterRectangle,myWindowBox,200,200
setwindowrectangle myWindowBox,""
openform "Options"
```

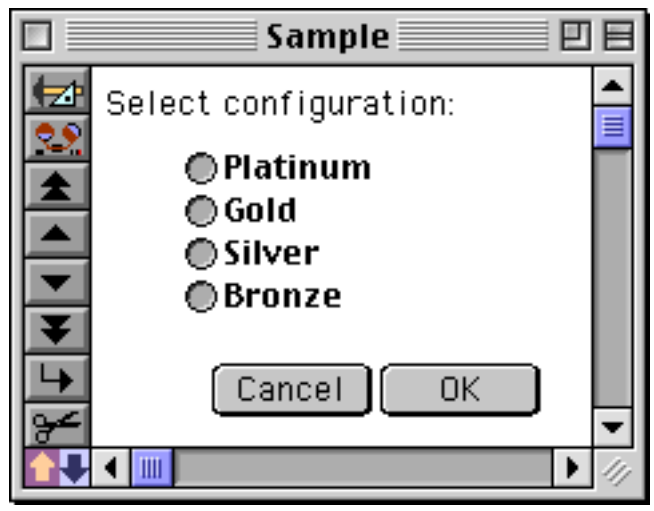
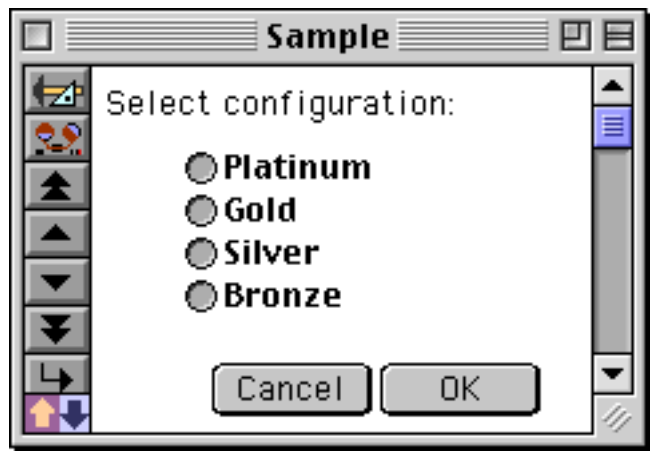
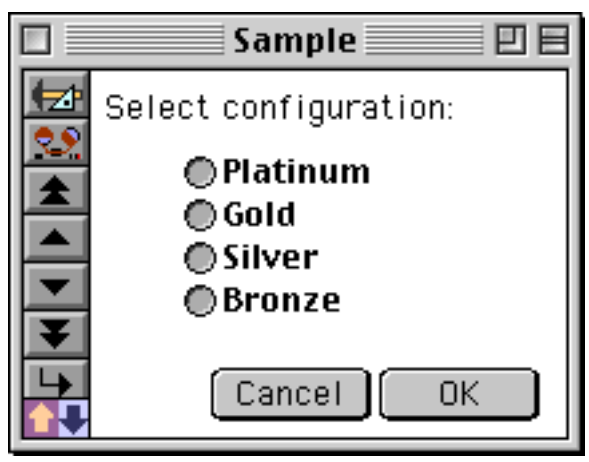
The `setwindow` statement is similar to `setwindowrectangle`, but uses four separate numbers as parameters instead of using a rectangle (see "SETWINDOW" on page 5753). The `windowbox` statement also uses four numbers, but they are combined into a single text item (see "WINDOWBOX" on page 5894). (Also, the `windowbox` statement only has one parameter, so you cannot set the window options.) The following three statements will work identically to each other.

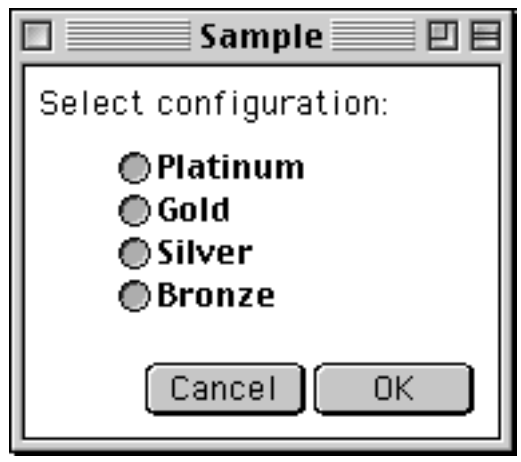
```
setwindowrectangle rectangle(100,120,250,400),""
setwindow 100,120,250,400,""
windowbox "100 120 250 400"
```

You may wonder why Panorama has three statements for doing the same thing. The answer is that as newer statements are added to Panorama in new versions, the older statements are retained for compatibility with existing procedures. In general, the `setwindowrectangle` statement is the best for new procedures because of all the functions Panorama now has for manipulating rectangles as a single unit (instead of having to deal with four separate numbers). See "Rectangles" on page 1290 to learn more about these functions.

New Window Options

Normal Panorama windows have a drag bar across the top, a tool palette down the left hand side, and two scroll bars down the right and bottom edges of the windows. Using the `setwindowrectangle` or `setwindow` statement, a procedure can open a window with or without any or all of these elements. The second parameter of these statements is a text argument that allows you to suppress these four window elements. The text may include any combination of these four components nodragbar, nopalette, novertscroll and nohorzscroll. Here are some examples of combinations of these options.

Option	Example
""	
"nohorzscroll"	
"nohorzscroll novertscroll"	

Option	Example
`"nopalette nohorzscroll novertscroll"`	
`"nodragbar nopalette nohorzscroll novertscroll"` NON-MOVABLE WINDOWS ARE NO LONGER RECOMMENDED!	

Here is a procedure that opens a 100 by 200 pixel form window in the upper left hand corner of the main screen. The window has no tool palette and no scroll bars.

```
setwindowrectangle rectangle(25,2,125,202),"nopalette novertscroll nohorzscroll"
openform "Status"
```

You can use these options to open a form as a dialog window with no drag bar, no tools, and no scroll bar. Another way to do this is with the `opendialog` statement, which is described later in this chapter. If a window is opened with the `opendialog` statement Panorama will treat it as a dialog and will not allow any other window to be brought on top of it (including windows of other currently running applications).

Non Standard Window Styles

In addition to standard windows Panorama supports several custom window styles. In fact, if you are using a Macintosh computer and you are an advanced C, Pascal or assembly language programmer you can even create your own window types. The `windowproc` statement allows you to override the normal default window type and use any type of window (see “WINDOWPROC” on page 5898).

The `windowproc` statement should be placed just in front of the statement that actually opens the window (see the example below). The `windowproc` statement has one parameter—a window type number.

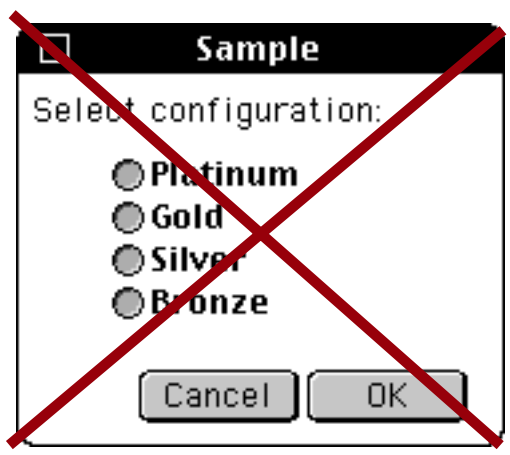
Window type 2 is a plain dialog box with no border and no drop shadow.



Window type 3 is also a plain dialog with no border. However, type 3 windows do have a drop shadow.



In previous versions of Panorama window type 16 was a rounded corner window with a black drag bar. This window type is no longer available.



Once a window has been opened, the window type is permanent. The only way to change a window type is to close the window and then re-open it with a new type.

Changing a Window's Position/Options

If a window has a drag bar, the user can move the window around on the screen. Using the `zoomwindow` statement, a procedure can also move a window, and/or change the display options for that window (see "ZOOMWINDOW" on page 5916).

The `zoomwindow` statement has five parameters:

```
zoomwindow top,left,bottom,right,options
```

These are the exact same options used by the `setwindow` statement.

The procedure below moves the current window to the right by 50 pixels. (The example assumes the current window has standard window options.)

```
local deltaV,deltaH
deltaV=0
deltaH=50
zoomwindow
   deltaV+rtop(info("windowrectangle")),
   deltaH+rleft(info("windowrectangle")),
   deltaV+rbottom(info("windowrectangle")),
   deltaH+rright(info("windowrectangle")),
   "" /* change this line to use non-standard window options */
```

If the window was originally close to the right hand edge of the screen, this procedure may push the window partially off the screen. If you want to prevent this, place a `fitwindow` statement (see "FITWINDOW" on page 5253) just in front of the `zoomwindow` statement.

Changing a Window's View

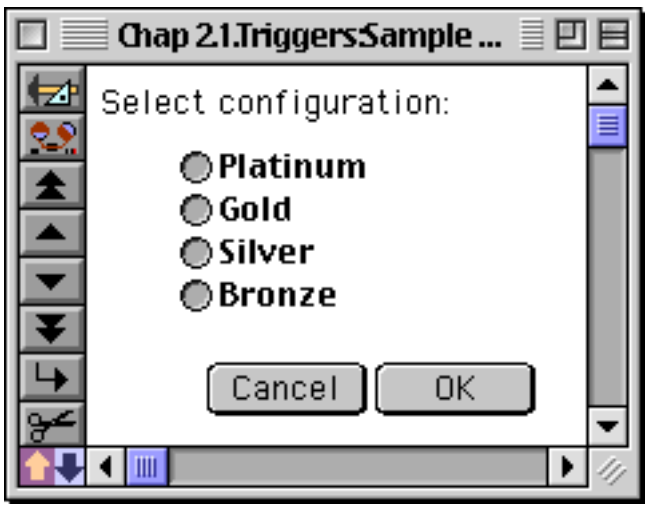
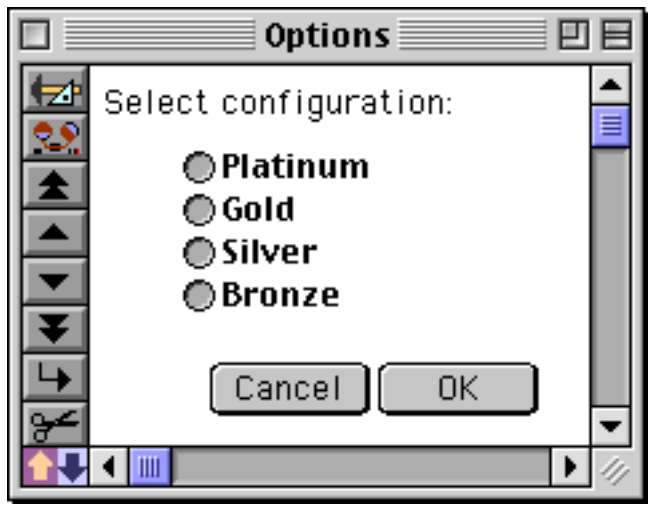
A procedure can change what is inside a window at any time. There are five primary statements that change what appears in the current window: `gosheet`, `goform`, `gocrosstab`, `goprocedure` and `godesignsheet`. These statements are the same as `opensheet`, `openform`, etc. except that instead of opening a new window, they simply open the requested form, crosstab, etc. inside the current window. If the requested view is already open in another window (or even the current window) the go commands simply bring that window to the front. In that case, the original window will continue to display whatever it was displaying before.

The procedure below switches the current window to a form named List.

```
goform "List"
```

Changing the Name of a Window

Panorama windows usually have a name that combines the database name with the name of the form, crosstab, or procedure being displayed. However, using the `windowname` statement a procedure can rename any window to anything you want (see "WINDOWNAME" on page 5897).

Before `windoname` statement…	After `windoname` statement.
	

The `windowname` statement has one parameter, the name for the window. The only restriction is that the name must be less than 31 characters long.

Suppose you have a database named Team with a form named Status. Normally the name of this window would be Team:Status. The procedure below opens the Status window and renames it to Status Display.

```
openform "Status"
windowname "Status Display"
```

The new window name is only temporary. Panorama will forget the new window name if the window is closed, or if the contents of the window are changed to a different view (either with the **View** menu or with a `goform`, `gosheet`, or other `go<view>` statement.)

Scrolling Inside a Form Window

If a form is larger that the window, the user can scroll to different parts of the form using the scroll bars. Using the `formxy` statement, a procedure can also scroll the form within the window (see "FORMXY" on page 5280). This statement has two parameters, the vertical and horizontal position:

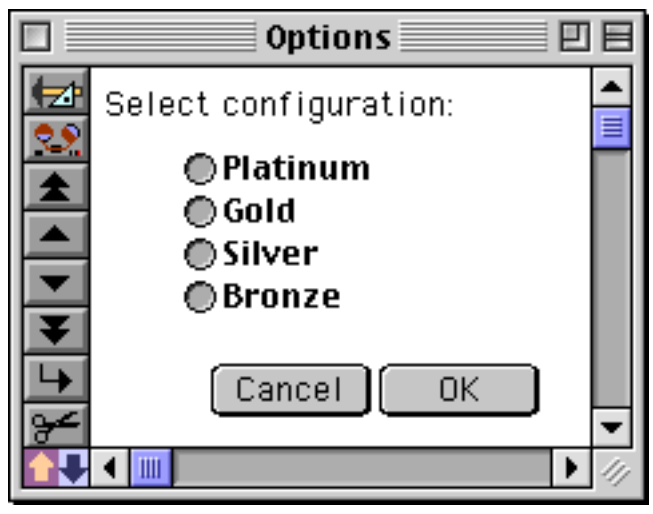
```
formxy vertical,horizontal
```

The vertical parameter is the vertical spot on the form that you want to appear at the very top of the window. If you want to see the top of the form this should be zero. The position is specified in pixels (1 pixel = 1/72 inch).

The horizontal parameter is the horizontal spot on the form that you want to appear at the very left edge of the window. If you want to see the left edge of the form, this should be zero.

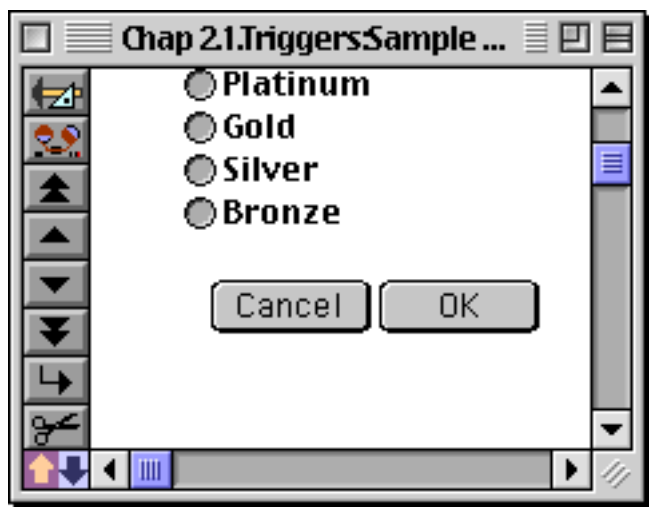
This simple example procedure makes sure that the upper left hand corner of the form is visible.

```
formxy 0,0
```



The next procedure slides the form down 1/2 inch.

```
formxy 72/2,0
```



Sometimes the `formxy` statement may not seem to slide the form to the exact position you specified. There are two possible reasons for this. First of all, the form cannot be scrolled farther than the bottom right object on the form. Once this object is visible, the form cannot be scrolled any farther.

The other possible discrepancy is that forms are always scrolled in increments of 8 pixels. This insures that patterns (which repeat every 8 pixels) are always displayed consistently. The `formxy` statement will always round the position you specify to the nearest multiple of 8.

Closing a Window

The user can usually close a window by clicking on the close box (in the upper left hand corner of the drag bar). A procedure can close a window with the `closewindow` statement (see “CLOSEWINDOW” on page 5113). This statement has no parameters—it simply closes the current (top) window.

When Panorama closes the last window associated with a database, it normally asks the user if he or she wants to save changes before closing the database. However, the `closewindow` statement does not usually do this. It simply closes the window and the database without saving any changes. Sometimes this is very convenient. A procedure can open a database, sort it, select, etc., then print a report and close the database, throwing away the temporary changes it has made. However, if you want the procedure to ask the user if they want to save changes, there are two ways to do this. The first is to write this into the procedure itself.

```
if info("changes")>0 and arraysize(listwindows(info("databasename")),¶)=1
   alert 1013,"Do you want to save changes?"
   if clipboard() contains "yes"
      save
   endif
endif
closewindow
…
… procedure continues
```

The second method is simply to make sure that the `closewindow` statement is either the last statement in the procedure, or that it is immediately followed by a `stop` statement. In either case Panorama will check for changes and ask the user if they want to save changes if necessary. (If `closewindow` is the last statement and you don't want Panorama to ask to save changes, simply put an extra `nop` statement (no-operation) after the `closewindow` statement.)

Trapping the Close Box

If the window has a drag bar, the user can close the window at any time simply by clicking on the close box. You may want to prevent the user from closing the window under certain conditions, or you may want to perform some special operation before the window is closed. These situations call for a **.CloseWindow** procedure (see ".CloseWindow" on page 1509). Whenever the user clicks on a close box, Panorama checks to see if there is a **.CloseWindow** procedure in this database. If there is, Panorama triggers the procedure instead of closing the window.

For example, suppose you want to allow the user to close any window except for the data sheet window. This **.CloseWindow** procedure will do the trick:

```
if info("windowname")=info("databasename")
   message "Sorry, you cannot close the data sheet"
else
   closewindow
endif
```

Notice that if it wishes to close the window, the **.CloseWindow** procedure must explicitly use the `closewindow` statement.

It's important to keep it mind that the only time the **.CloseWindow** procedure is triggered is when the user clicks on the close box. It is not triggered when the user selects the **Quit** or **Close File** commands from the File menu, or by the `closewindow` statement.

In addition to trapping the close box, you may wish to trap the action of closing the entire database. You can do this with the **..CloseDatabase** procedure, see "..CloseDatabase" on page 1525.

Changing The Window Order (Who's on Top?)

The user can bring any window to the front by clicking on the window. A procedure can bring a window to the front with the `window` statement (see "CLOSEWINDOW" on page 5113). This statement has one parameter, the name of the window that needs to be brought to the front. The name must be spelled and capitalized exactly as it appears in the window title at the top of the window.

```
window "Price List:Report"
```

If there is no such window, the procedure will normally stop and display an error. The procedure can trap and respond to this error with the `if error` statement, like this (see "Error Handling with if error" on page 1381).

```
window "Price List:Report"
if error
   openfile "Price List"
   openform "Report"
endif
```

Use the `info("windowname")` function to get the name of the currently active (top) window (see "INFO("WINDOWNAME")" on page 5445). One application for this function is when you want to jump to a different window, then jump back to the original window. This example jumps to the price list window, sorts the price list, then jumps back to the original window.

```
local wasWindow
wasWindow=info("windowname")
window "Price List:Prices"
field Description
sortup
window wasWindow
```

Use the `info("windows")` function to get a text array listing of open windows (see "INFO("WINDOWS")" on page 5447). The text array is carriage return separated (see "Text Arrays" on page 1236), and it lists the windows in order from front to back (the top window is array element 1, the next window is element 2, etc.) The procedure below swaps the order of the top and second from top window.

```
window array(info("windows"),2,¶)
```

Here is another procedure that brings the bottom window to the front. Using this procedure over and over again will cycle through all of the windows.

```
local windowCount
windowCount=arraysize(info("windows"),¶)
window array(info("windows"),windowCount,¶)
```

The `listwindows(` function is similar to `info("windows")` but allows you to list only the windows that belong to a particular database (see "LISTWINDOWS(" on page 5478). This function has one parameter, the name of the database. (If you leave the database name as an `""` empty string, the `listwindows(` function will list all windows, just like the `info("windows")` function.) The procedure below displays the number of open Price List windows.

```
local windowCount
windowCount=arraysize(listwindows("Price List"),¶)
message "The Price List has "+str(windowCount)+" windows open."
```

The `windowtoback` statement is the opposite of the `window` statement. It sends a specified window all the way to the bottom of the pile (see "WINDOWTOBACK" on page 5900).

```
window "Price List:Report"
print dialog
windowtoback "Price List:Report"
```

When the procedure is done with the Price List:Report window, it moves it out of the way, below all of the other windows. (Of course another option would be to close the window.)

Temporary "Invisible" Windows

Often a procedure needs to flip back and forth between windows in different databases. Although this gets the job done, it is very annoying and slow if the windows overlap each other.

To get around this problem a procedure can work with **secret windows**. From a procedure's point of view, a secret window is just like any other window that contains a form. The good news is that secret windows are invisible! A procedure can flip to a secret window in another database, perform some operation on that database (search, sort, etc.), then go back to the original window—all without the flashing and updating that usually occurs when you flip from window to window.

Secret windows are temporary. A secret window can be created by the `window` statement (see "WINDOW" on page 5892). The secret window ceases to exist as soon as the procedure brings another window to the top (or as soon as the procedure stops).

To create a secret window and make it the current window, use the `window` statement with the parameter <database>:secret. For example, to open a secret window for the Price List database use this statement:

```
window "Price List:Secret"
```

The Price List database must be open or this statement will not work. The word secret may be capitalized any way you want: secret, Secret, or even SECRET.

Here is a procedure that opens the price list in a secret window, sorts the price list, then goes back to the original window (all without any flashing).

```
local wasWindow
wasWindow=info("windowname")
window "Price List:Secret"
field PartDescription
sortup
window wasWindow
```

(Note: If your database actually contains a form named Secret (or secret or SECRET) and it is open, the window statement will bring this window to the front instead of creating a secret invisible window.)

Databases Without Windows

Most Panorama databases have at least one window at all times. However, it is possible to create a database that has no windows at all. Such a database can be used for lookups, or it can be used with secret windows (see "Temporary "Invisible" Windows" on page 1581).

To create a database that has no windows, use the **Save As** command and check the No Windows option (see "Saving Window Positions" on page 132). The next time you open this database (either by double clicking on it or with the **Open File** dialog) it will open without any windows.

To temporarily open a database without its normal windows use the `opensecret` statement (see "OPENSECRET" on page 5582). This statement is identical to the `openfile` statement (see "Opening a Panorama Database" on page 1533), but it does not open the windows. (It also does not trigger the **.Initialize** procedure, see ".Initialize" on page 1511.) The example below opens the Price List database without any windows.

```
opensecret "Price List"
```

If you open a database with `opensecret` and then later decide that you want the windows after all you can use the `openfile` statement to open the windows. This will also trigger the files **.Initialize** procedure if it has one (see ".Initialize" on page 1511).

```
opensecret "Price List"  /* open file without windows */
…
…
…
openfile "Price List"    /* file is already open, just open the windows */
```

The `makesecret` statement makes all the windows for the current database vanish (see "MAKESECRET" on page 5522). The database is still open and can be used for lookups or with secret windows. (Note: Since all the windows for the current database vanish, some other database will be the top window after this statement.) Here is an example **.CloseWindow** procedure that makes the database invisible if the user closes the last window:

```
if arraysize(listwindows(info("databasename")),¶)=1
   makesecret
else
   closewindow
endif
```

If this is not the last window for this database, the procedure simply closes the window. If it is the last window, it closes the window with `makesecret`, so the database is still open in memory.

To re-open a window in a database that has no visible windows you need to use a secret window. The procedure below opens the Distributor window in the Price List database.

```
window "Price List:Secret"
setwindowrectangle rectanglesize(20,20,300,180),""
openform "Distributor"
```

If you want to actually close a file with no windows you must again use a secret window. This procedure removes the Price List file from memory.

```
window "Price List:Secret"
closefile
```

Using secret windows, a procedure can access an invisible database just as easily as a visible database. The procedure can read, modify, sort, or even save the database. There's no need to open a visible window unless the user needs to see the database.

"Magic" Windows

Panorama has a number of `info(` functions and graphic statements that work with the currently active window. In some circumstances you may want to use one of these functions or statements to work with one of the other open windows. Panorama's "magic window" feature allows you to temporarily designate any open window as the currently active window for use with these `info(` functions (see "Window, Form and Report Information" on page 1325) and graphic statements (see "Programming Graphic Objects on the Fly" on page 1753). The designated window doesn't actually move to the front, so you can use this feature without unnecessary window "flashing."

There are two statements available for designating an open window as the "magic" active window.

```
magicwindow windowname

magicformwindow database,form
```

In either case the window or form must already be open. From this point on in the program all `info(` functions and graphic commands will refer to this window, instead of the "real" current window. To remove the magic window designation and go back to the "real" current window use the statement

```
magicwindow ""
```

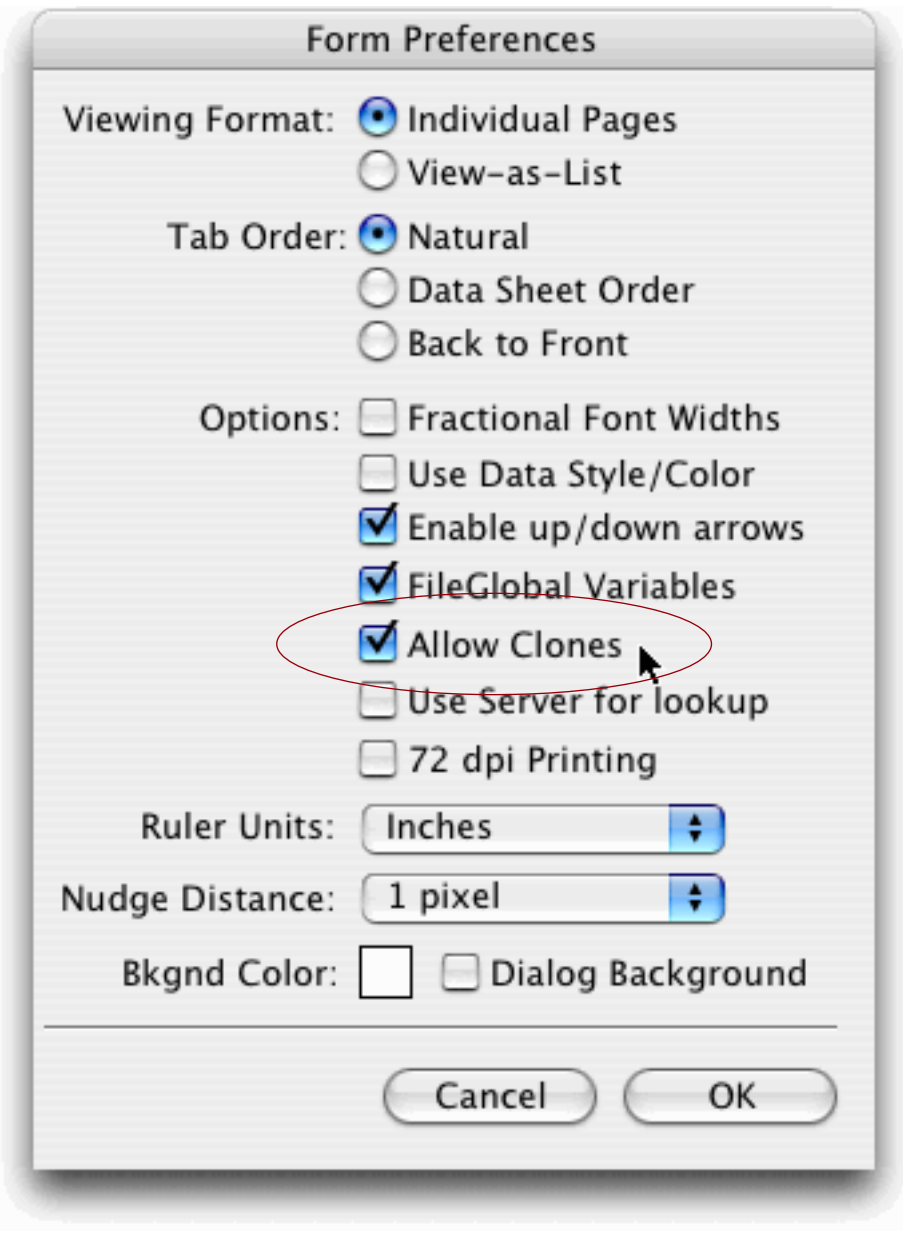
The `info("magicwindow")` function can be used to check the name of the currently designated “magic” window, if any. For example this little program will update the People List list object in the window Contacts:List. The `if` statement is used in case the specified window is not open at all.

```
magicwindow "Contacts:List"
if info("magicwindow")<>""
   superobject "People List","Fill List"
   magicwindow ""
endif
```

You may confuse this “magic window” feature with the “secret window” feature, but they are quite different. The “secret window” feature creates a virtual, invisible window that is not tied to any actual window or form. This secret window may be used for any database operation, including data entry, searching, sorting, etc. The “magic window” feature does not create an invisible window, but works only with windows that are actually open. It does not affect most database operations (these still take place in the “real” active window), but only affects `info(` functions (see “Window, Form and Report Information” on page 1325) and graphic statements (see “Programming Graphic Objects on the Fly” on page 1753).

Window Clones

Panorama normally allows only a single window per form. However a form can be designed to be opened over and over again into multiple windows. This is called **window cloning**. To allow a form to be cloned you must open the **Form Preferences** dialog and select the Allow Clones option.



A window clone cannot be opened manually…clone windows must be created with the `openform` statement in a procedure. Here is a typical procedure that opens a slightly offset clone of the current window:

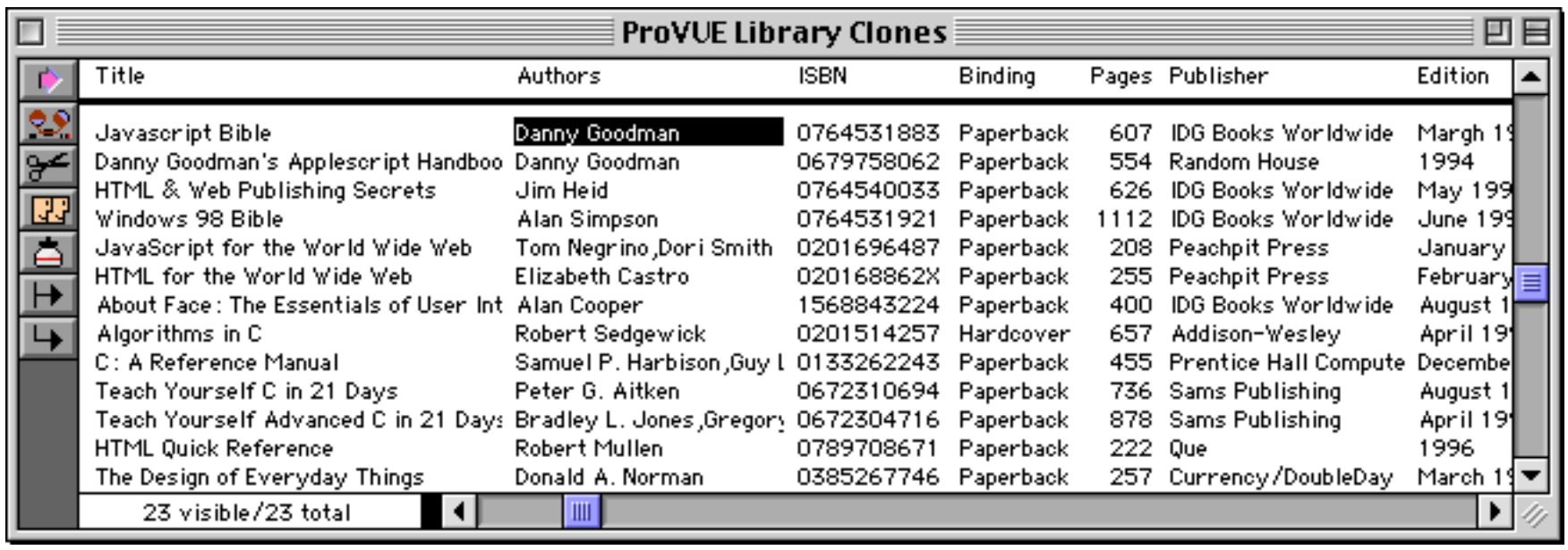
```
setwindowrectangle rectangleadjust(info("rectangle",10,10,10,10))
openform info("formname")
```

This procedure will not create a clone window unless the Allow Clones option is turned on.

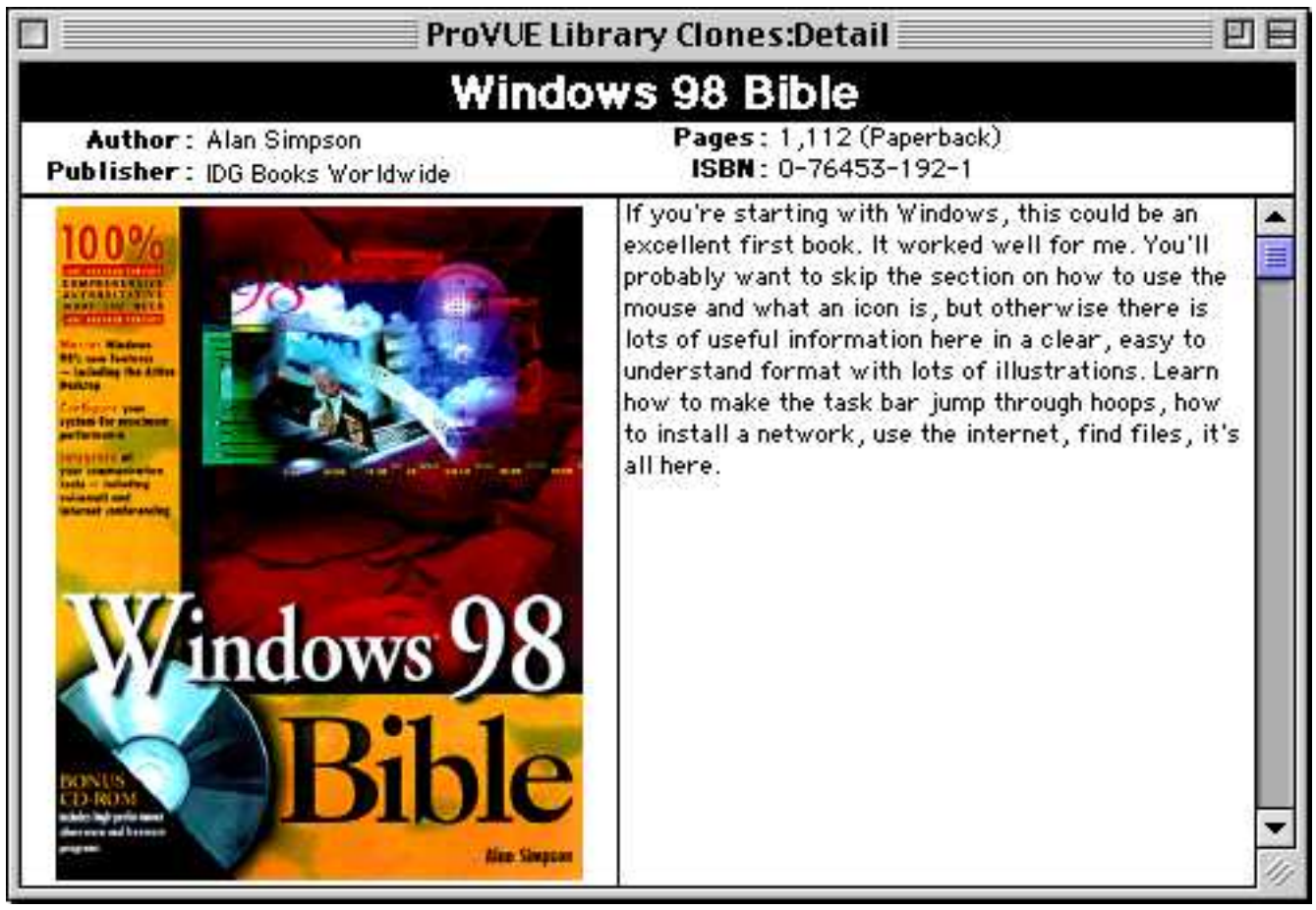
Designing A Clone Window Application

Although any form can be cloned if the Allow Clones option is turned on, most forms will not work very intelligently if they are cloned. In general, a form that is designed to be cloned should not contain any fields or global variables, only `windowglobal` variables (see “Variable Accessibility” on page 1373). If your form contains Text Editor, Data Button, Pop-Up Menu or List SuperObjects and the Allow Clones option is turned on, these SuperObjects will automatically create `windowglobal` variables instead of global variables. Since the `windowglobal` variables can be manipulated separately for each clone window you can control each clone window individually, even though all the clone windows use the same form template.

To illustrate clone windows we’ll use this database that contains a list of books. Here is the data sheet.



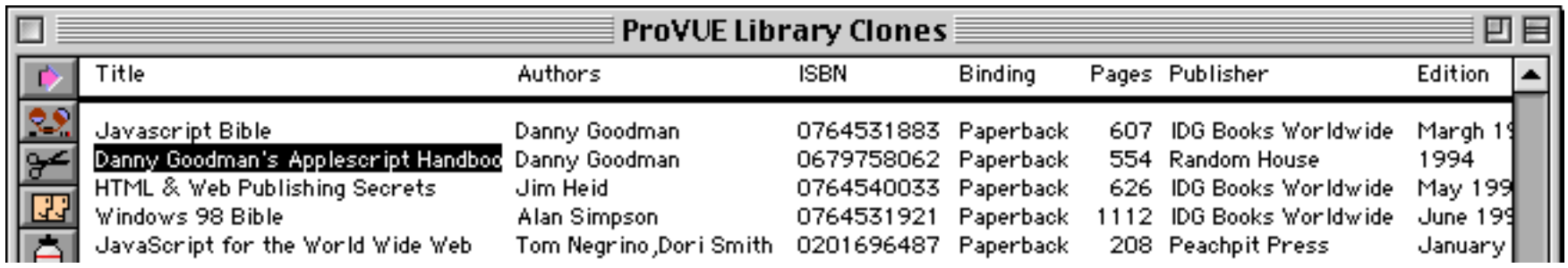
In this database we've created a form called Detail that displays the information from a single record in the database.



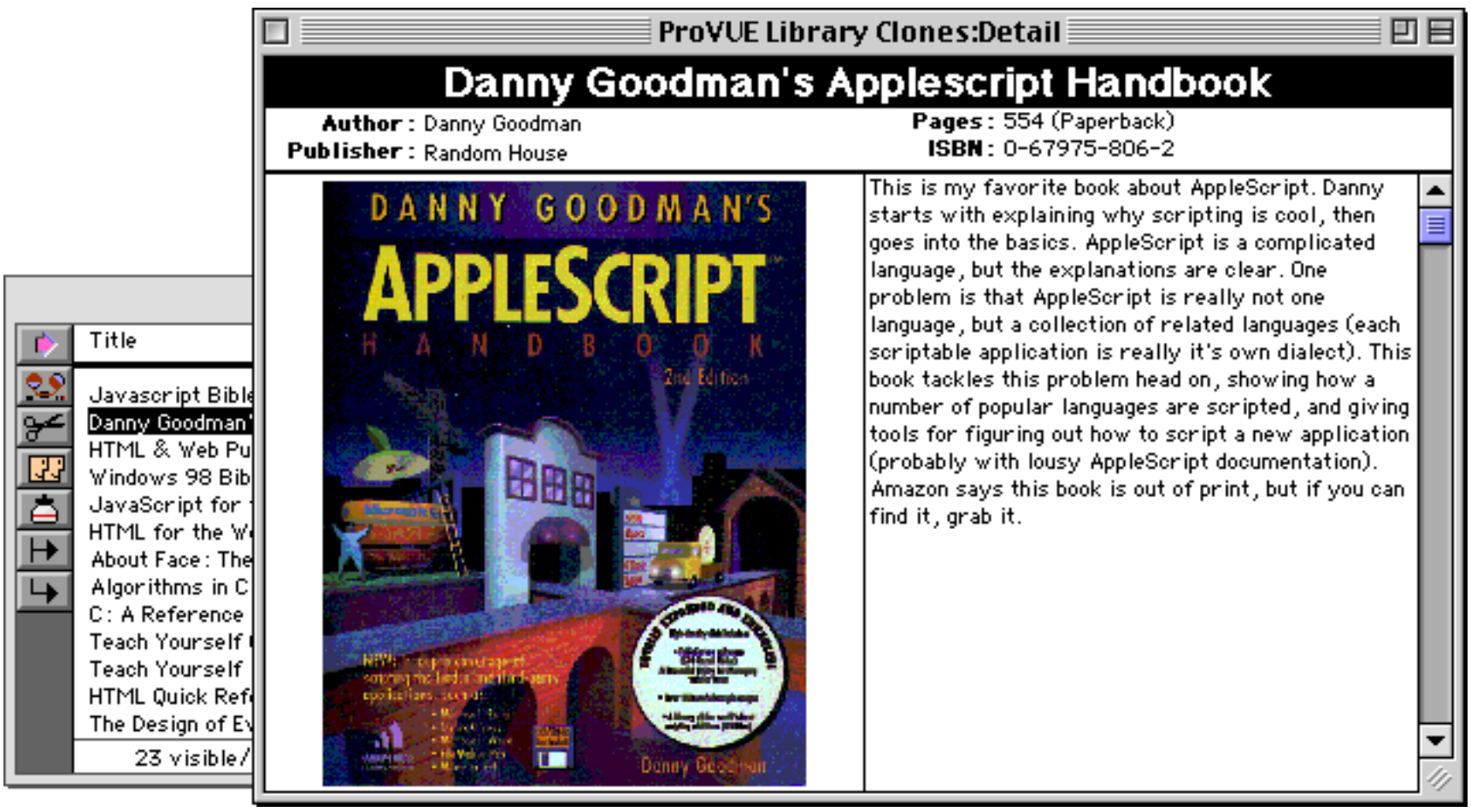
This form doesn't display the information directly from the database fields (Title, Authors, Publisher, etc.). Instead it has been set up to display information from a series of variables with the same name as the fields but with an x added to the beginning (xTitle, xAuthors, xPublisher, etc.). These variables are created in this procedure which opens the clone form, creates the variables and fills them with the data from the database fields.

```
local dRect
/* open the clone window */
dRect=rectanglesize(
   100+rtop(info("windowrectangle")),200+rleft(info("windowrectangle")),340,520)
setwindowrectangle dRect,"noVertScroll noHorzScroll nopalette"
fitwindow
openform "Detail"
/* create new variables for THIS window */
windowglobal xTitle,xAuthors,xISBN,xPages,xBinding,xPublisher,xDescription
/* fill the variables with the data from the current record */
xTitle=Title
xAuthors=Authors
xISBN=ISBN
xPages=Pages
xBinding=Binding
xPublisher=Publisher
xDescription=Description
/* display the variables */
showvariables xTitle,xAuthors,xISBN,xPages,xBinding,xPublisher,xDescription
```

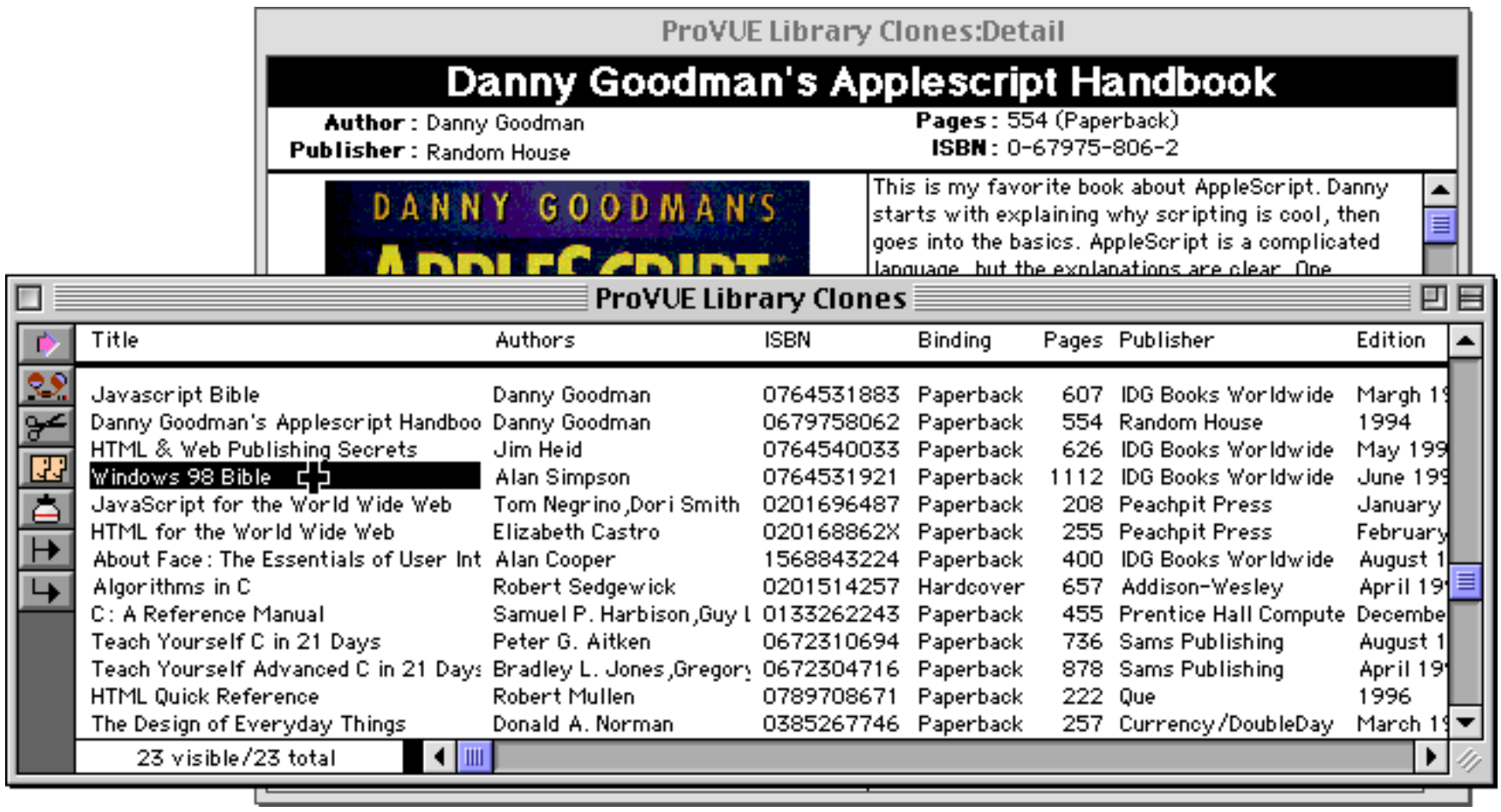
Let's see how this procedure works. Start by selecting a record in the data sheet.



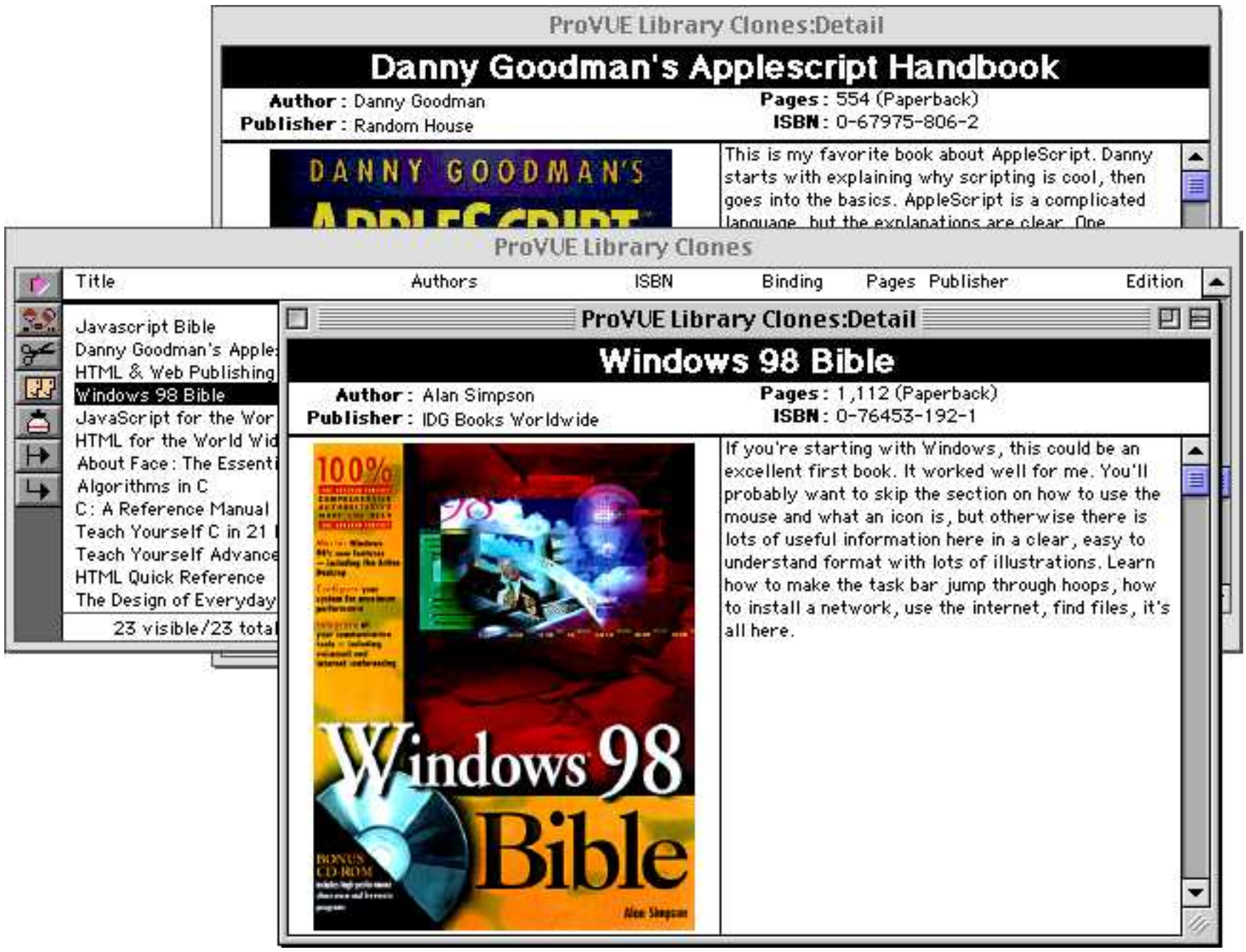
Now select the procedure from the **Action** menu (we have called it **Open Clone**, but you can call it anything you want). The procedure will open a new window which displays the information for Danny Goodman's Applescript Handbook.



Now click on the data sheet to bring it back to the front and then click on another record. Usually when you do this any open forms will automatically synchronize to show the new record. But in this case, the form is not displaying the information directly from the database but instead is displaying the data we have stored in the `windowglobal` variables. These variables have not changed, so the form continues to display the data from the original record.



Now if we select the **Open Clones** procedure again the procedure will open a second copy of the Detail form. This new copy shows the information from the current record, while the original Detail window continues to show the information from the original window.



Using the **Open Clone** procedure we can continue to open additional "clone" copies of the form — as many as we want up to Panorama's 32 window limit.



There is no special handling necessary for closing clone windows. The window simply closes when you click on the close box. All of the windowglobal variables associated with the window are destroyed when the window is closed.

Alerts

Alerts are very simple dialogs that simply display a message and allow the user to press a button. Alerts are usually used to "alert" the user of a situation (a problem, perhaps). Panorama has several off-the-shelf alerts. (You can also create your own alerts with a form, just like any other dialog.)

The simplest way to alert the user to a situation is to use the `beep` statement (see "BEEP" on page 5072). This statement, which has no parameters, simply causes the computer to make its standard beep sound.

Displaying a Message

The simplest way to display a short piece of text is with the `message` statement (see "MESSAGE" on page 5529). This simply displays an alert with any message you want. The alert stays on the screen until the user presses the **OK** button. The message statement has one parameter, the text of the message to be displayed. The example below displays the number of records in the database.

```
message "Total records: "+str(info("records"))
```

Here is what this alert looks like when this procedure runs.



There are also three variations on the `Message` statement — `BigMessage`, `GiantMessage` and `TallMessage`. If you use these, keep in mind that you can only display 256 characters in these dialogs. So the `TallMessage` statement, for example, is useful for a narrow list of items, but you definitely can't fill the dialog with text (the `DisplayData` and `SuperAlert` statements described below will allow that).



If you really want to display a lot of text use the `DisplayData` statement. For example, this program will ask you to select a file, then display the entire file.

```
local folder,file,type,filetext
openfiledialog folder,file,type,""
if file="" rtn endif
filetext=fileload(folder,file)
displaydata filetext
```

This statement will allow you to display up to 32,768 characters. The dialog normally takes up most of whatever screen size you have available (the illustration below has been reduced to 60%) and has a scroll bar so that you can view additional text.

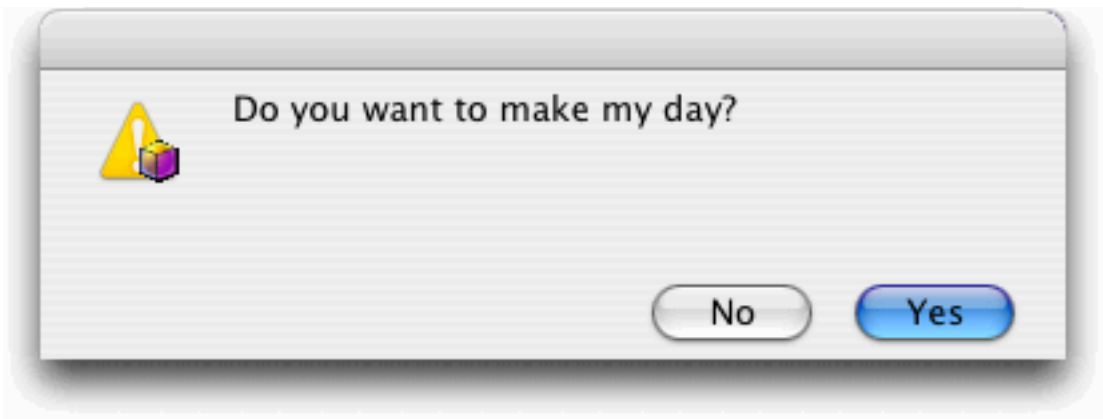


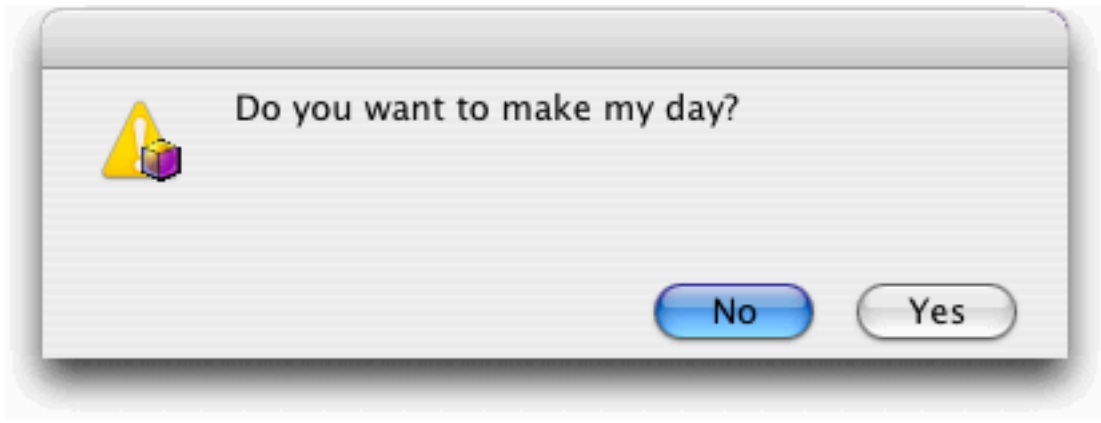
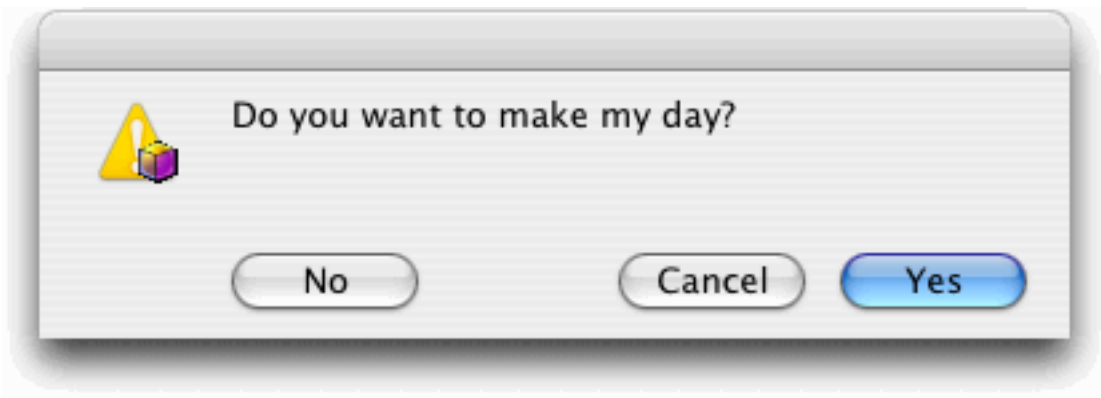
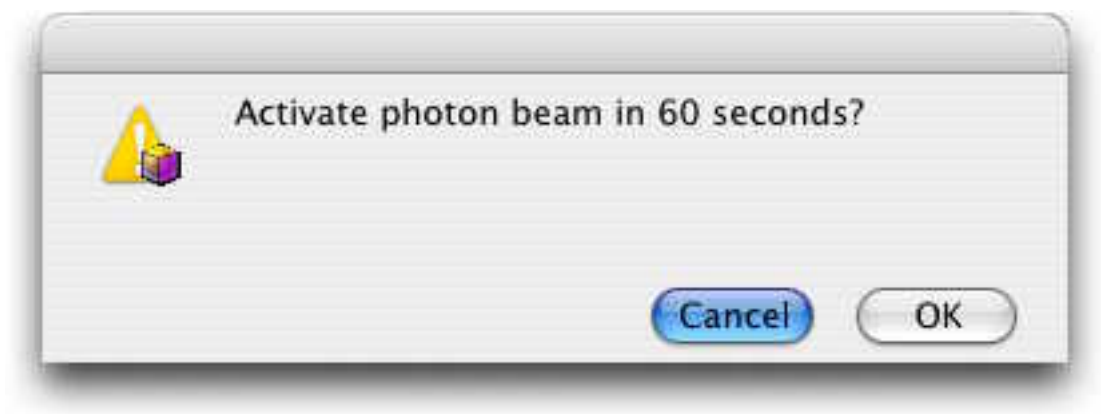
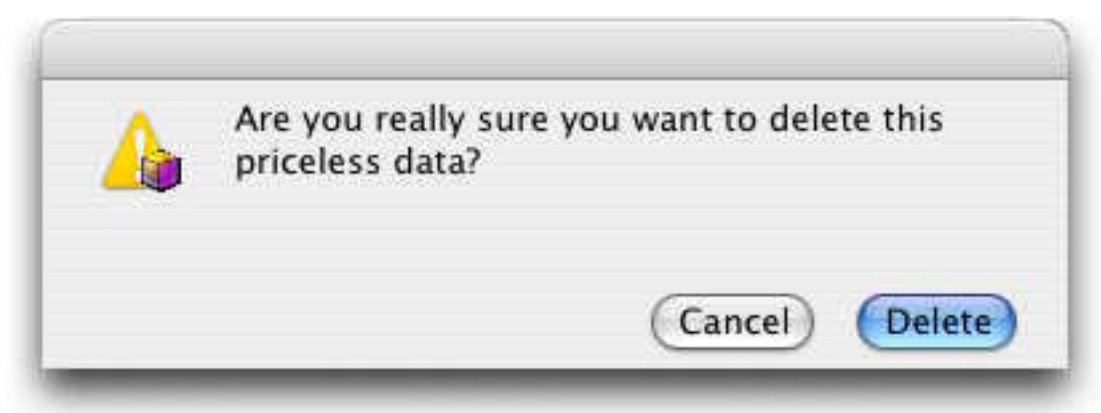
There are three buttons at the bottom of the dialog. To simply continue, press the **OK** button. To stop the procedure, press the **Stop** button. Press the **Copy** button to copy the text that is displayed to the clipboard (this also stops the procedure).

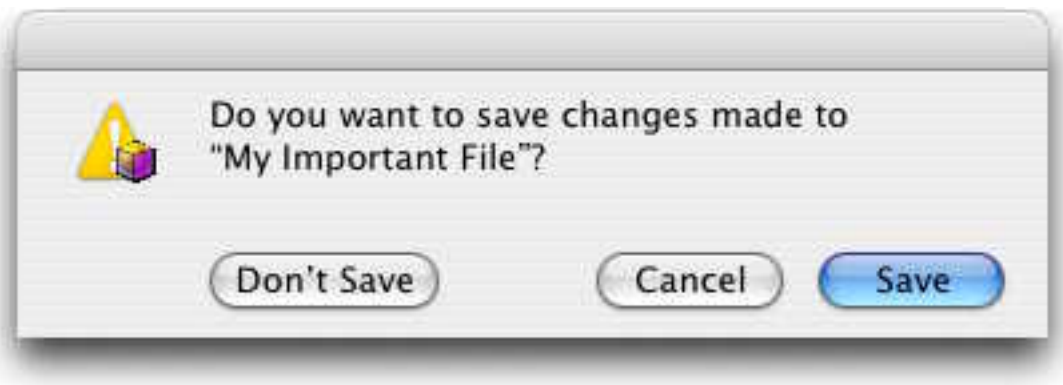
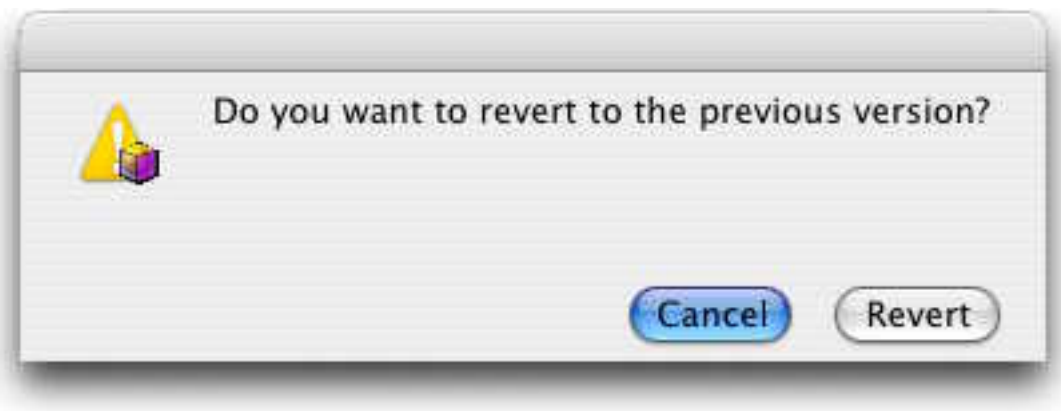
The `displaydata` statement has an optional second parameter that allows you to customize the appearance of the dialog. You can change the size, title, font, text size, color and many other options. For more information see “The SuperAlert Statement” on page 1599.

Alerts With Multiple Buttons

Panorama has several statements for creating standard alerts with multiple buttons. After this statement the program can use the `info("dialogtrigger")` function to find out what button was pressed. This function will return the text of the button, for example, Yes, No, Cancel, etc.

Code	Sample
`alertyesno "Do you want to make my day?"`	

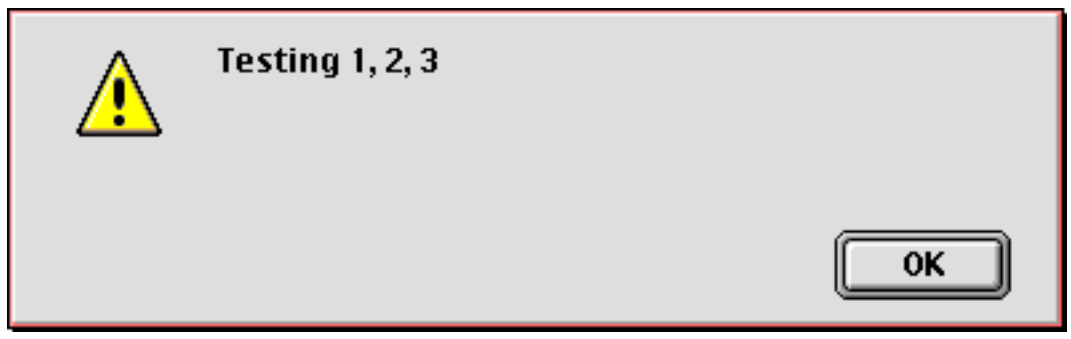
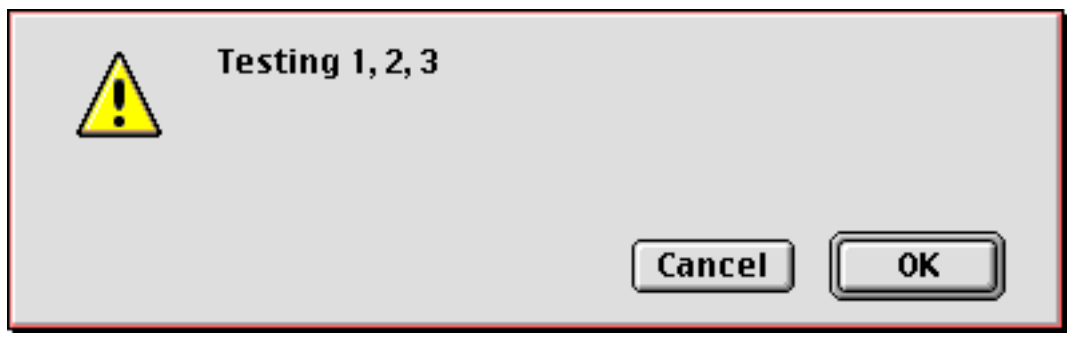
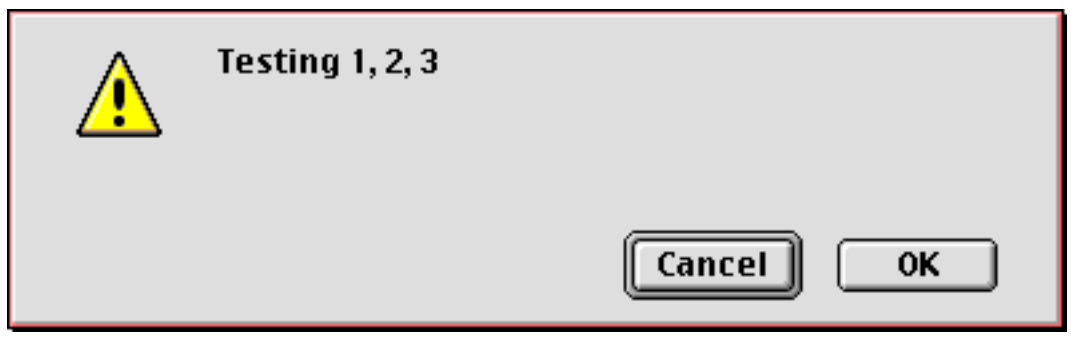
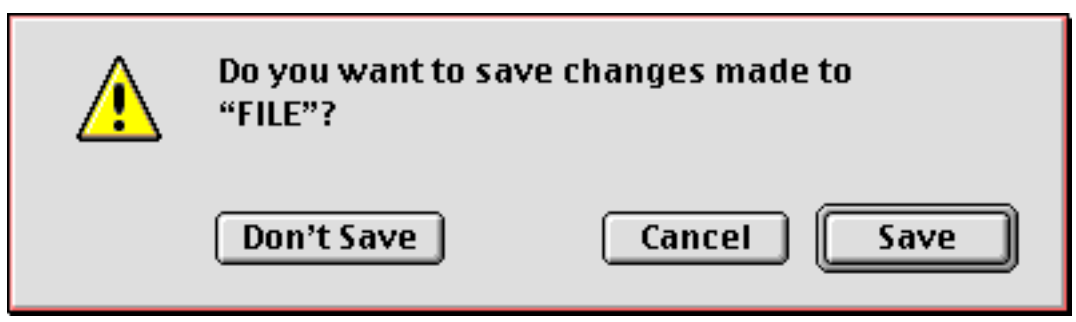
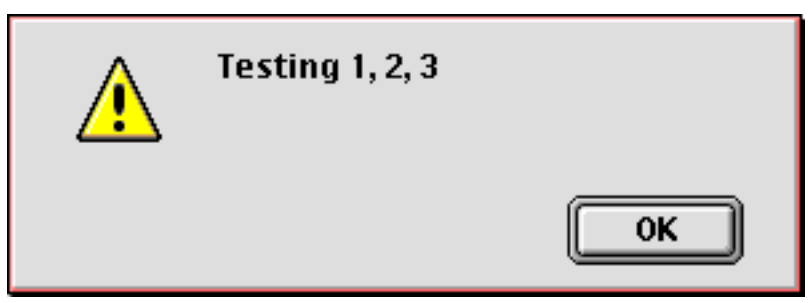
Code	Sample
`alertnoyes "Do you want to make my day?`	
`alertyesnocancel "Do you want to make my day?"`	
`alertokcancel "Activate photon beam in 60 seconds?"`	
`alertcancelook "Activate photon beam in 60 seconds?"`	
`alertdeletecancel "Are you really sure you want to delete this priceless data?""`	

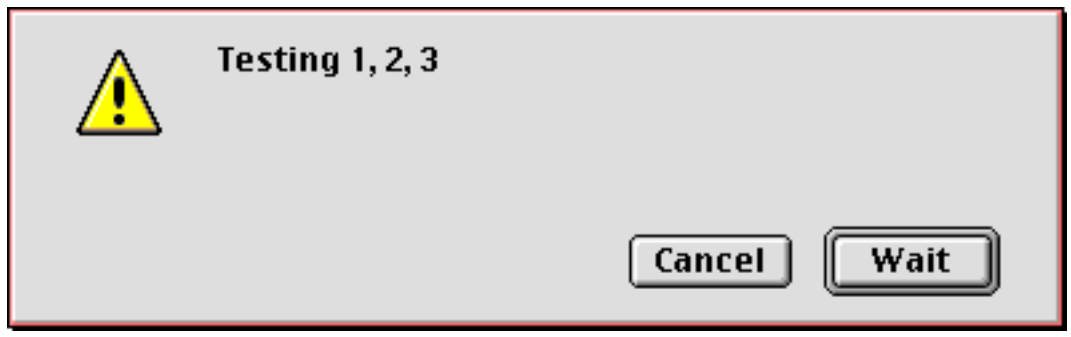
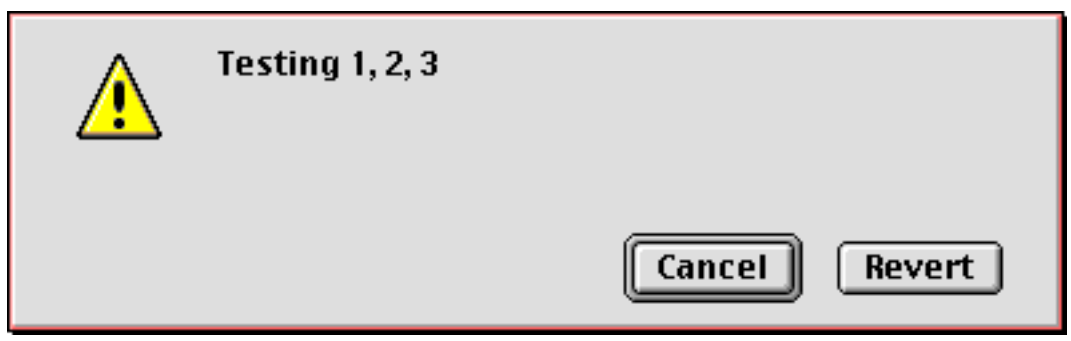
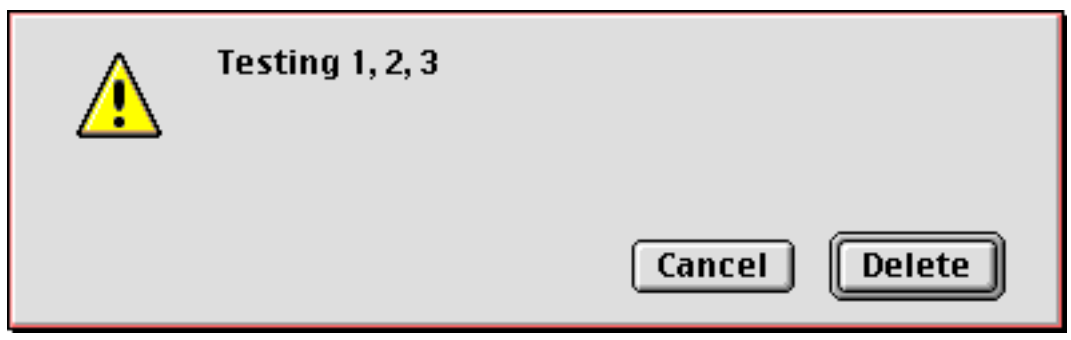
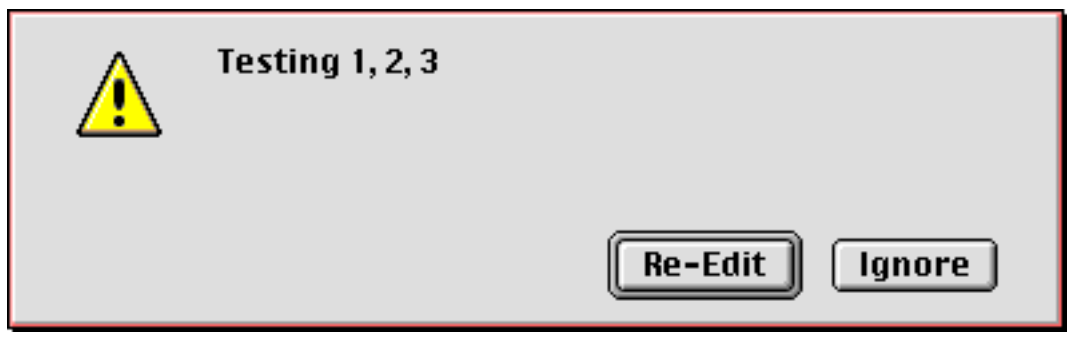
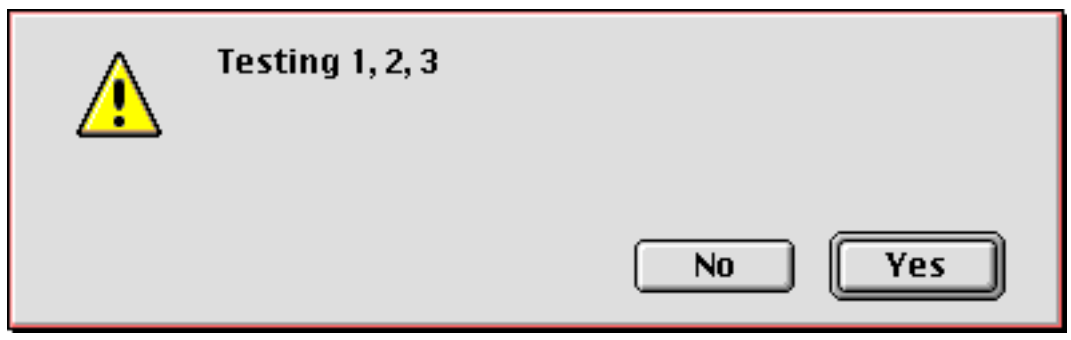
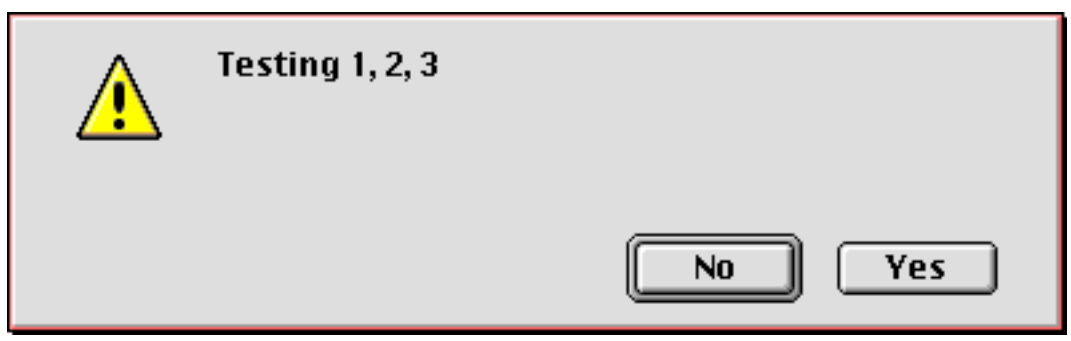
Code	Sample
`alertcanceldelete "Are you really sure you want to delete this priceless data?"`	
`alertsavecancel "My Important File"`	
`alertrevertcancel "Do you want to revert to the previous version?"`	
`alertok "This is identical to the message statement except for the icon."`	
`alertoksmall "When you have very little to say, this one is perfect!"`	

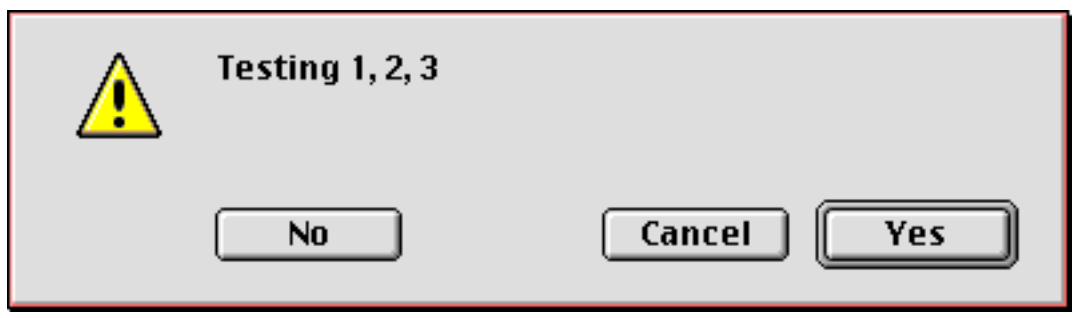
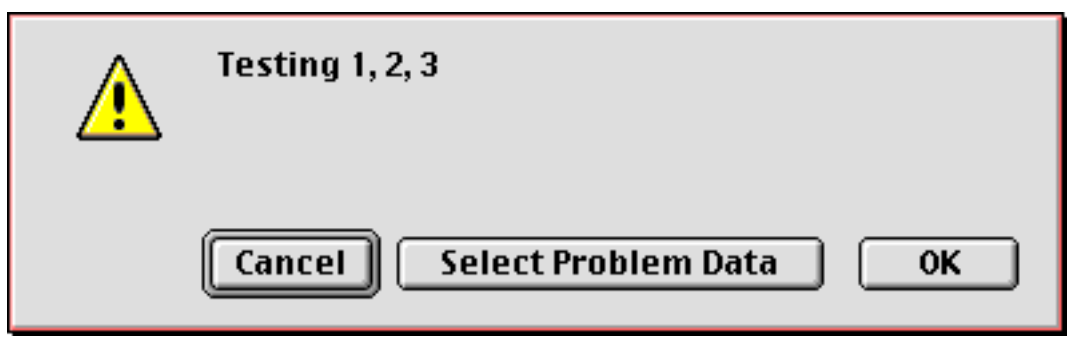
The Alert Statement

The `alert` statement allows you to build your own custom alerts with **ResEdit** (see "Working with Resources" on page 1560). However, we no longer recommend this statement — use the `SuperAlert` statement instead (see "The SuperAlert Statement" on page 1599) . The `alert` statement has two parameters: id and message. The id is the resource number of the alert template you have built. The message is the text you want to display in the alert. Like the statements above, to find out what button was pressed use the `info("dialogtrigger")` function (see "INFO("DIALOGTRIGGER")" on page 5370).

There are several alert templates built into Panorama that you can use with the alert statement.

Code	Sample
`alert 1000,"Testing 1, 2, 3"`	
`alert 1001,"Testing 1, 2, 3"`	
`alert 1002,"Testing 1, 2, 3"`	
`alert 1003,"FILE"`	
`alert 1005,"Testing 1, 2, 3"`	

Code	Sample
`alert 1008,"Testing 1, 2, 3"`	
`alert 1009,"Testing 1, 2, 3"`	
`alert 1010,"Testing 1, 2, 3"`	
`alert 1012,"Testing 1, 2, 3"`	
`alert 1013,"Testing 1, 2, 3"`	
`alert 1014,"Testing 1, 2, 3"`	

Code	Sample
`alert 1015,"Testing 1, 2, 3"`	
`alert 1018,"Testing 1, 2, 3"`	
`alert 1101,"Testing 1, 2, 3"`	

The example below uses the `alert` statement instead of the `noyes` statement.

```
alert 1014,"Do you really want to remove the old data?"
if info("dialogtrigger")="Yes"
   select Date>today()-90
   removeunselected
endif
```

If you create your own alert templates with **ResEdit**, make sure the resource file is open (use the `openresource` statement) before you attempt to use the alert (see “Opening and Closing Resource Files” on page 1562).

Obsolete Alert Statements

Since Panorama was first released in 1988 it has evolved and improved in many ways. For compatibility, we usually leave in older features even when new features make them obsolete, as is the case with the statements described in this section. However, they are still described here because you may encounter them in older databases. These statements display a message and allow the user to make a choice: **Yes** or **No**, **Ok** or **Cancel**, etc. The statements that display these alerts are `yesno`, `noyes`, `okcancel`, and `cancelok`. These statements all display an alert with two buttons. For example, the `yesno` statement displays a dialog with **Yes** and **No**

buttons. Notice that the first button is the default button, so the difference between yesno and noyes is which button is the default. All of these statements have one parameter: the text of the message to be displayed. The statements will put the name of the button clicked into the clipboard. The example below uses the noyes statement to confirm that the user really wants to delete data from the database.

```
noyes "Do you really want to remove the old data?"
if clipboard()="Yes"
   select Date>today()-90
   removeunselected
endif
```

When this procedure is run the alert looks like this.



Using the newer alertnoyes statement, this procedure would look like this:

```
alertnoyes "Do you really want to remove the old data?"
if info("dialogtrigger")="Yes"
   select Date>today()-90
   removeunselected
endif
```

The advantage of the alertnoyes statement is that it doesn't disturb the clipboard.

Suppressing Alerts

In some applications (particularly web servers) you may want to suppress all alerts so that the program never stops and waits for someone to do something. You can do this with the alertmode statement (see "ALERTMODE" on page 5028). This statement has one parameter, which controls whether alerts will appear. If the parameter is "yes", "true", or "on", alerts will be displayed. If the parameter is "no", "false", or "off", alerts will not be displayed. The program will simply continue as if the default button had been pressed.

The SuperAlert Statement

Panorama V introduced this new statement that displays a configurable alert. You can control over a dozen different options, including the overall alert size (height and width), the font, text size, title, color, background color, and more. You can even place images on the alert and specify up to three buttons on the alert. In addition, the alert may display up to 32,768 characters (it is *not* limited to 255 characters like the other alerts described in this section) and can even have a scroll bar.

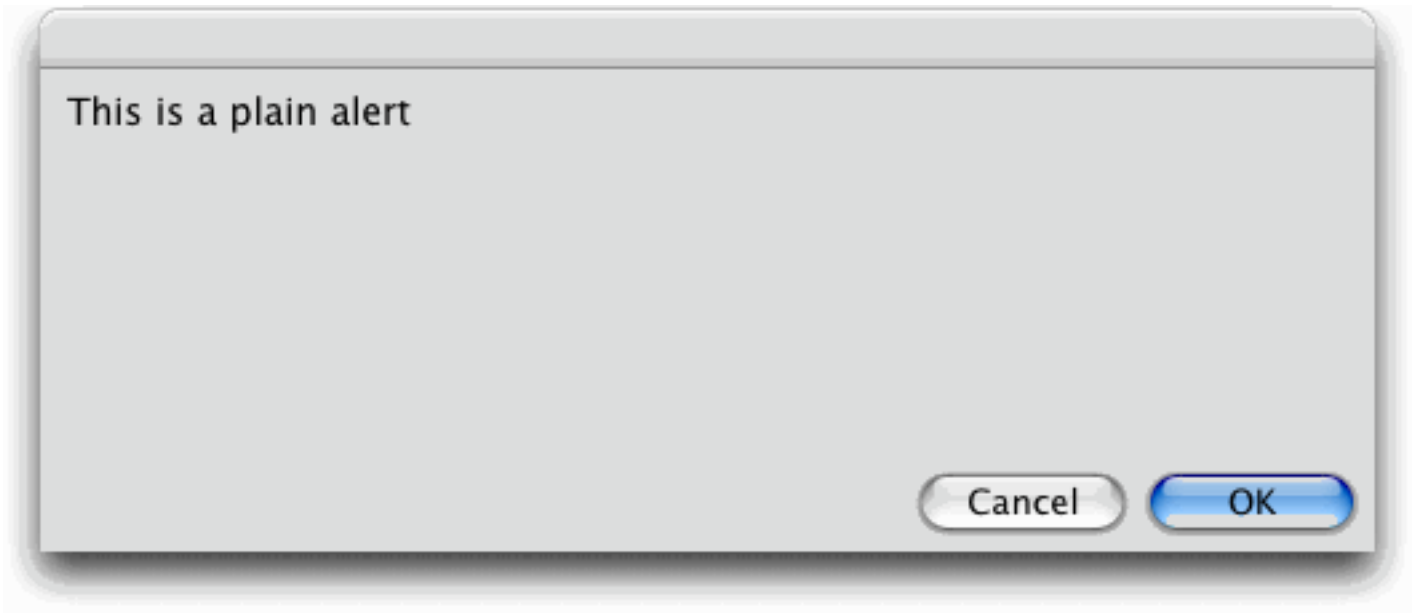
The SuperAlert statement has two parameters

```
superalert message,options
```

Message is simply whatever text you want to display in the alert. Options is a text parameter that uses a syntax similar to an HTML tag to specify one or more options. If you just want to use the default options you don't have to specify any options at all.

```
superalert "This is a plain alert",""
```

This plain alert looks like this:



Each option is specified as an `option=value` pair, for example `height=4in`, `color=red`, etc. Here is an example of a dialog with several options set. (Tip: If you enclose the entire option parameter in “smart quotes” as shown below you will be able to use regular " quotes for the individual options, if necessary.)

```
superalert "WARNING!",
  “height=4in font=helvetica size=36 color=red style=bold textalign=center buttons=Danger”
```

This alert should get some attention!



The table below describes each of options available with this statement.

Option	Examples	Description
title=	title="Latest Information"	With this option you can set the title of the alert. This appears in the title bar at the top of the alert window.
height=	height=400 height=5in height=4.5" height=10cm height=80% height=-50 height=-2cm	This option specifies the height of the alert. The default is 200 pixels. If you supply just a number then the height is specified in pixels (72 pixels = 1 inch). If the number is followed by in or " then it is specified in inches. If the number is followed by cm it is specified in centimeters. If the number is followed by % it is specified as a percentage of the screen height, for example 50% is 1/2 of the screen height. If the height is negative then this value is the distance of the border between the top of the screen and the alert (also the bottom of the alert and the bottom of the screen). For example if your screen is 8 inches high then a height of -1in will make the alert 6 inches high (one inch at the top and one inch at the bottom).
width=	width=600 width=7in width=5.5" width=12cm width=80% width=-50 width=-2cm	This option specifies the width of the alert. The default is 500 pixels. If you supply just a number then the width is specified in pixels (72 pixels = 1 inch). If the number is followed by in or " then it is specified in inches. If the number is followed by cm it is specified in centimeters. If the number is followed by % it is specified as a percentage of the screen width, for example 50% is 1/2 of the screen width. If the width is negative then this value is the distance of the border between the left of the screen and the alert (also the right edge of the alert and the right side of the screen). For example if your screen is 10 inches wide then a width of -1in will make the alert 8 inches wide (one inch on the left and one inch on the right).
font=	font=Tekton font="Comic Sans" font=Verdana	This option specifies the font to be used. The default is whatever the system font is on your system.
size=	size=9 size=18	This option specifies the text size (in points). The default is whatever the default system text size is on your system.
style=	style=italic style=bold style="bold italic"	This option specifies the text style. You can choose one or more options from bold, italic, underline, outline and shadow. All of the text in the alert will be displayed in the same style, you cannot mix styles within the alert.
textalign=	textalign=center	This option specifies how the text will be positioned within the alert. The default is *topleft*. The available choices are: *topleft, topcenter, topright, leftcenter, center, rightcenter, bottomright, bottomcenter,* and *bottomleft*.
scroll=	scroll=yes scroll=thin	This option enables a scroll bar to allow viewing of large quantities of text. The options are *yes, thin* and *no*.
buttons=	buttons="Yes;No;Cancel" buttons="One:50;Two:50"	This option allows you to specify up to three buttons (the default is two buttons: *Ok* and *Cancel*). Each button is separated by a semicolon, and the first button listed is the default button. The default button width is 80 pixels, you can also specify a button width in pixels by placing a colon followed by the width after the button name. In the second example to the left the buttons will be 50 pixels wide.
timeout=	timeout=20	If this option is used the alert will close automatically after the specified number of seconds. If the user doesn't press a button before the specified time the alert will automatically press the default button so the procedure can continue.

Option	Examples	Description
color=	color=#00FF00 color=red color=royalblue	This option specifies the text color (the default is black). If the color begins with # then you can specify any color using HTML style color tags (#RRGGBB, for example #00FF00 for green). You can also choose from this list of colors: *aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkseagreen, darkslateblue, darkslategray, darkturquoise, darkviolet, deeppink, deepskyblue, dimgray, dodgerblue, firebrick, floralwhite, forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod, gray, green, greenyellow, honeydew, hotpink, indianred, indigo, ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral, lightcyan, lightgoldenrodyellow, lightgreen, lightgrey, lightpink, lightsalmon, lightseagreen, lightskyblue, lightslategray, lightsteelblue, lightyellow, lime, limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid, mediumpurple, mediumseagreen, mediumslateblue, mediumspringgreen, mediumturquoise, mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy, oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen, paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple, red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen, seashell, sienna, silver, skyblue, slateblue, slategray, snow, springgreen, steelblue, tan, teal, thistle, tomato, turquoise, violet, wheat, white, whitesmoke, yellow, yellowgreen.*

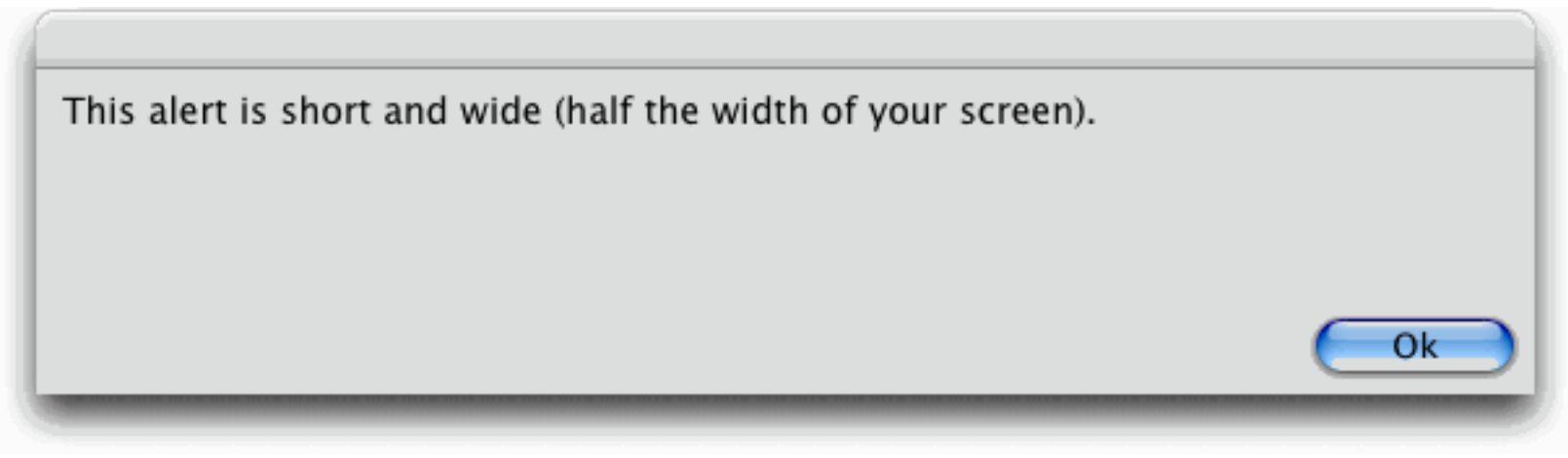
Option	Examples	Description
bgcolor=	bgcolor=#FFDDFF bgcolor=lightskyblue bgcolor=##983	This option specifies the background color (the default is gray). If the color begins with # then you can specify any color using HTML style color tags (#RRGGBB, for example #FFCCFF for lite pink). You can also choose from the same list of colors available for the color= option (see above). A third option is to use ## followed by a resource number to display a background image from Panorama's resources. You can use the **Icons & Backgrounds** wizard to find out what background images are available.
icon=	icon=stop icon=caution icon=note	This option will display a stop sign, caution sign, or note icon in the alert.
image=	image="Smiley Face"	This option allows you to display any image you want within the alert. The image may be in a disk file or in the flash art gallery of the current database. If the image is in a disk file the default location is the folder containing the current database, but you can specify any folder you want by specifying the complete path.
imageedge=	imageedge=top imageedge=left imageedge=both imageedge=none	This option specifies how the image should placed relative to the text. *Left* means that the image will be placed to the left of the text. *Top* means that the image will be placed above the text. *Both* means the text will be placed both below and to the right of the image (the image in the upper left corner with the text in the lower right corner). *None* means that the text will be drawn on top of the image, essentially making the image a background. For all of these options the alert will adjust the text location based on the size of the image.
imagealign=	imagealign=topcenter imagealign=leftcenter	This option specifies how the image should be aligned relative to the edges of the alert. The options are *topleft* (the default), *topcenter*, *topright*, *leftcenter* and *bottomcenter*.
texttopadjust=	texttopadjust=10	The text position is automatically adjusted based on the image size. If the image is placed at the top (imageedge=top) the text is normally placed 3 pixels below the image. This option allows you to change that spacing.

Here are some examples illustrating the use of the `superalert` statement.

```
superalert "Are you talking to me?",
    {height=120 width=200 font="Verdana" size=9 color="red" buttons="Yes:60;No:60"}
```




```
superalert "This alert is short and wide (half the width of your screen).",
    {height=2in width=50% buttons="Ok"}
```




```
superalert "Look Out!",{image="Stop Sign"}
```



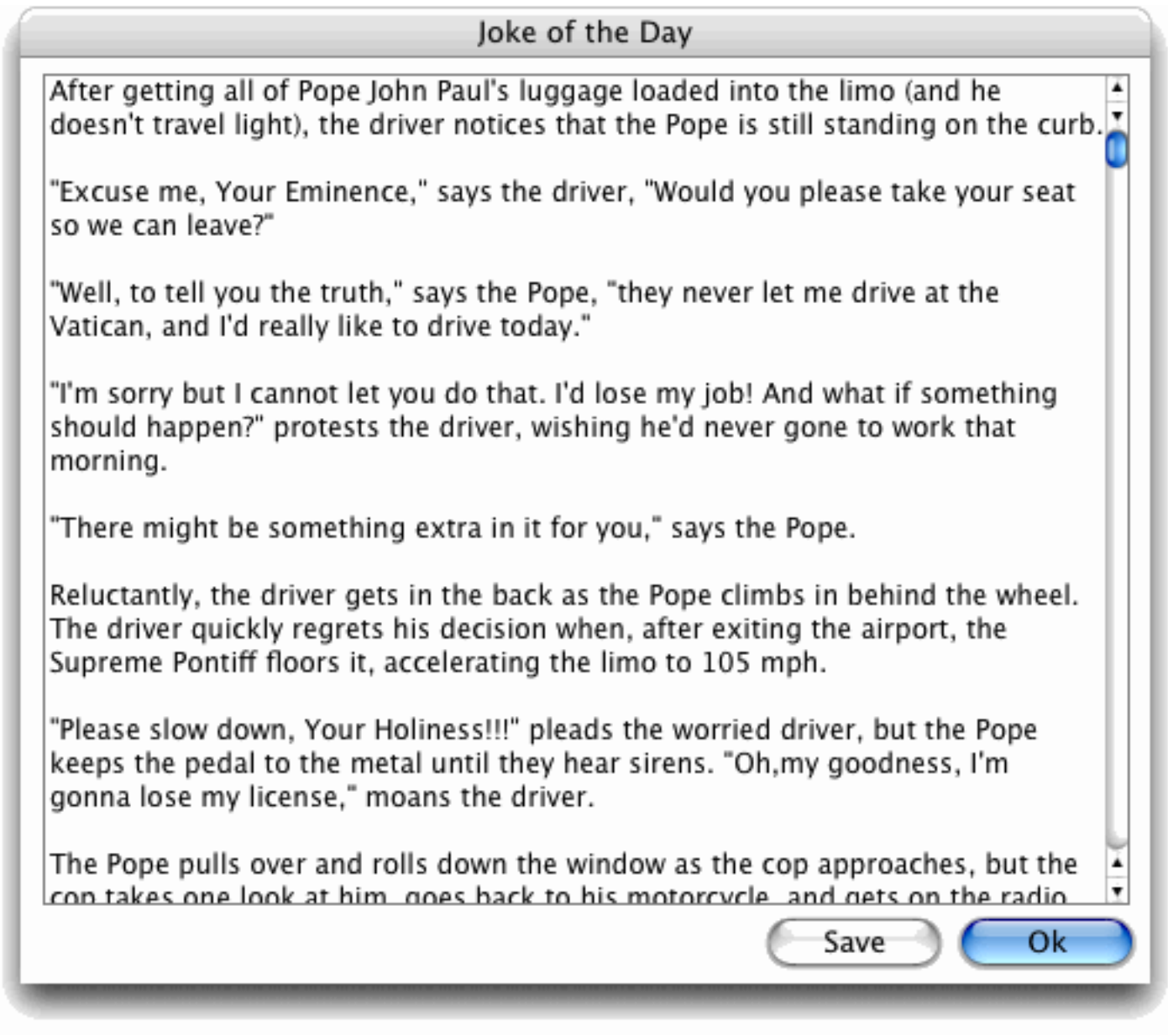
```
superalert "Look Out!",{image="Stop Sign" imageedge=top bgcolor=white}
```




```
superalert "Look Out!",{image="Stop Sign" imagealign=topcenter imageedge=top bgcolor=white}
```



```
superalert Joke,“Title="Joke of the Day" height=6in width=7in scroll=thin size=12
bgcolor=white buttons="Ok;Save"”
```



The DisplayData Alert

The `DisplayData` alert (see “Displaying a Message” on page 1591) is based on the `SuperAlert` statement. The second parameter to the `DisplayData` statement can take all of the same options as the `SuperAlert` statement, but has different default values. You can override these defaults by supplying a second parameter with the options you want changed.

```
displaydata myFile,“title=”My File” font=Courier size=12”
```

The default options are white background, Monaco 9 point font, window height 100 pixels less than screen height, width 80% of screen width, with three buttons: **Ok**, **Stop** and **Copy**. (You can eliminate the **Stop** and / or **Copy** buttons, but if you add other buttons they will not work. Use the `superalert` statement directly if you need to add other buttons.)

Dialogs

Dialogs are a special type of window. A dialog window is usually temporary, and usually modal. In other words, the dialog must be filled in and dismissed before the user can continue with his or her work.

Basic Text Entry Dialogs

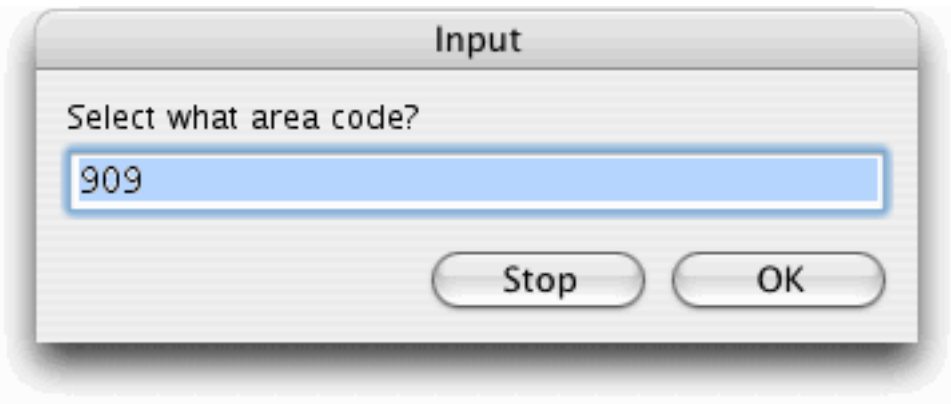
The most basic dialog prompts for a single item of text. Panorama has several options for easily creating this type of basic dialog, starting with the `gettext` statement. This statement has two parameters.

```
gettext prompt,input
```

The prompt parameter should be a short message that will be displayed in the dialog. This message should explain what needs to be entered. The input parameter must be the name of a field or variable that contains text. The value in this field or variable will be displayed in the dialog—the user can use it as is, edit the value, or erase it completely and type in a new value. The example below uses `gettext` to find out what area code to search for. The default area code is 909.

```
local whatArea
whatArea="909"
gettext "Select what area code?",whatArea
select Phone match "("+whatArea+")*"
```

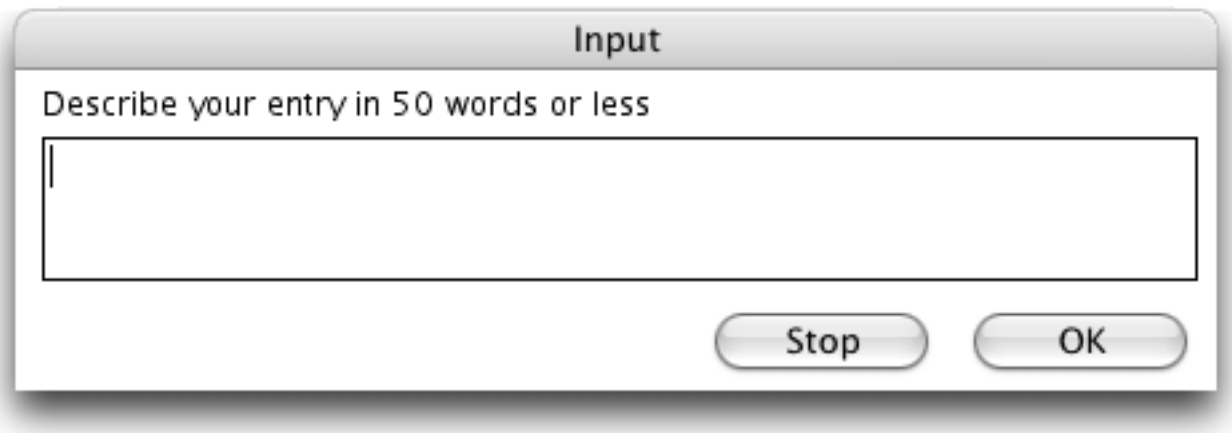
When this procedure is run a dialog like this is displayed.



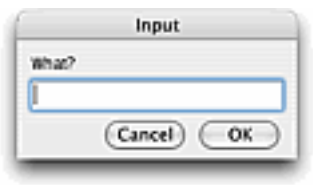
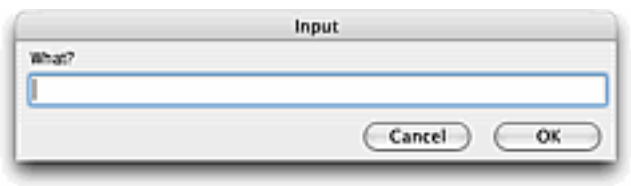
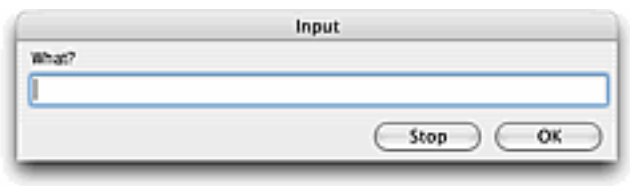
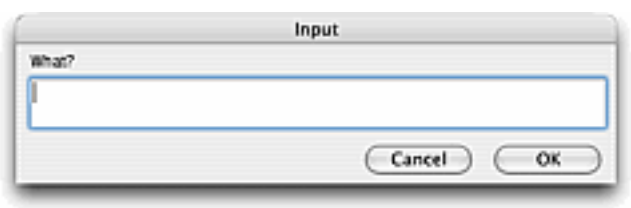
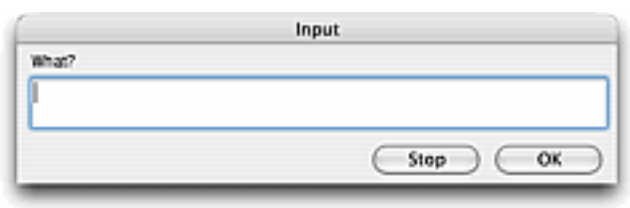
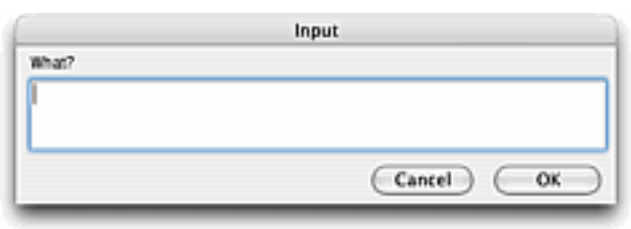
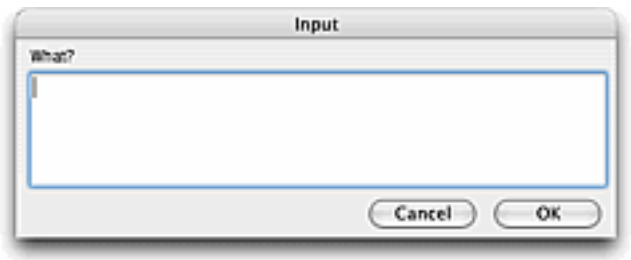
You can customize the appearance of this basic dialog by using the `customdialog` statement (see "CUSTOMDIALOG" on page 5135). This statement has one parameter, the resource ID number of a dialog template. You can create dialog templates with **ResEdit** (see "Working with Resources" on page 1560), or you can use one of several templates supplied with Panorama. Here's a procedure that uses one of Panorama's built in templates.

```
customdialog 3103
gettext "Describe your entry in 50 words or less",Description
```

Here is the alert that will appear.



This table shows the different templates that are available as part of Panorama. (To save space, these images have been reduced to 50%).

#	Sample	#	Sample
3131			
3121		3101	
3122		3102	
3123		3103	
3125		3105	
3120		3100	

As you may have noticed, some of these dialogs contain a **Cancel** button and some contain a **Stop** button. If the dialog contains a **Stop** button the procedure will stop immediately if the button is pressed.

If the dialog contains a **Cancel** button the procedure will continue no matter what button is pressed. The procedure can use the `info("dialogtrigger")` function to find out which button was pressed (see “INFO("DIALOGTRIGGER")” on page 5370). If the **Cancel** button was pressed Panorama will ignore whatever the user typed into the field or variable, leaving the original value untouched. Here is a procedure that uses the `info("dialogtrigger")` function to find out which button was pressed.

```
local whatArea
whatArea="909"
customdialog 3131
gettext "Select what area code?",whatArea
if info("dialogtrigger")="OK"
   select Phone match "("+whatArea+")*"
else
   selectall
endif
```

If you create your own custom resource templates make sure the resource file is open before you use the dialog (see “Opening and Closing Resource Files” on page 1562).

The SuperGetText Statement

Panorama V introduced this new statement that displays a configurable dialog for entering text. You can control over a dozen different options, including the overall dialog size (height and width), the font, text size, title, color, background color, and more.

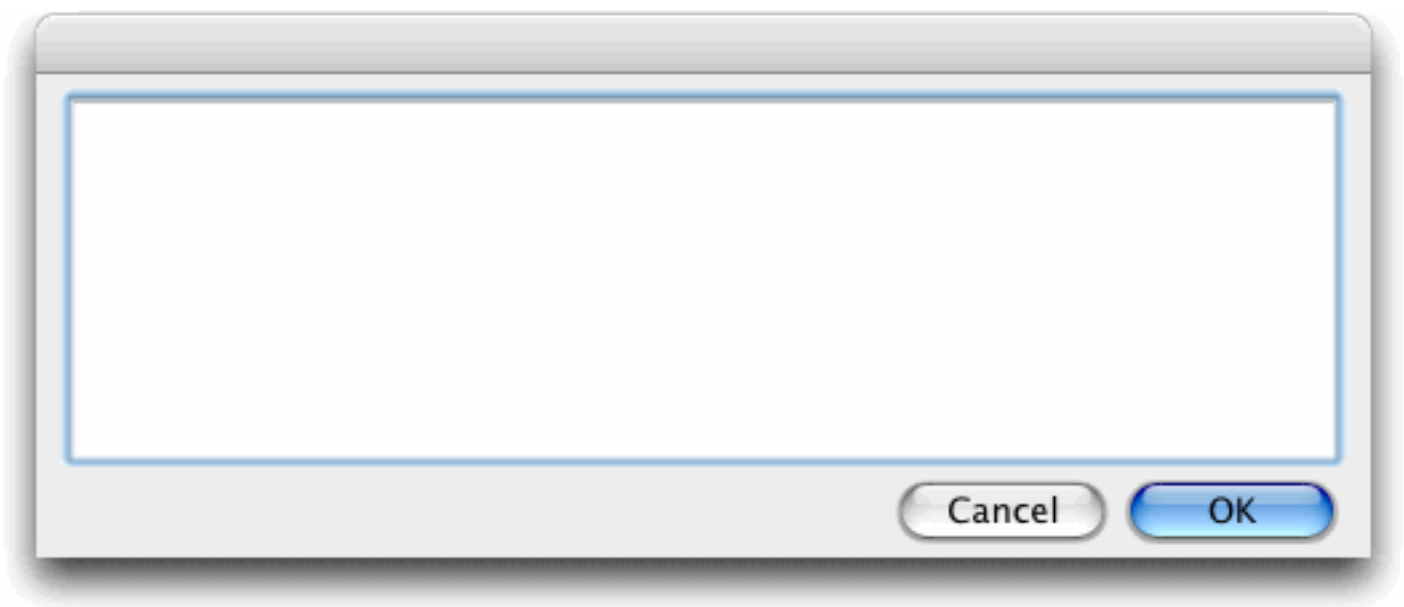
The SuperGetText statement has two parameters

```
supergettext input,options
```

Input is a field or variable that will contain the text that is typed into the dialog. Options is a text parameter that uses a syntax similar to an HTML tag to specify one or more options. If you just want to use the default options you don’t have to specify any options at all.

```
supergettext Query,""
```

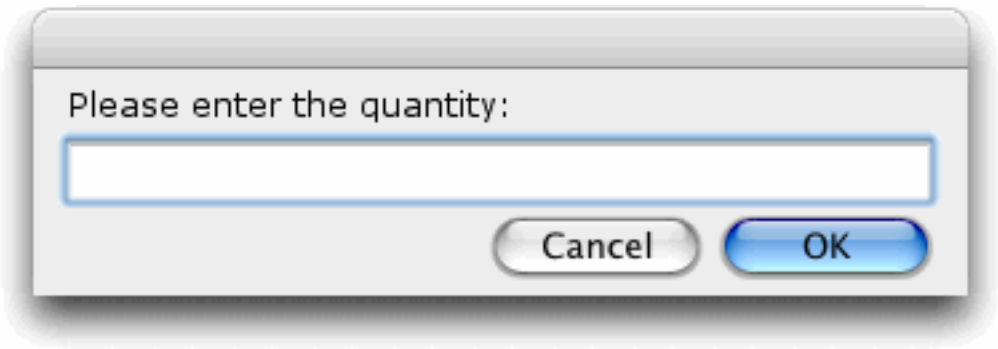
This plain dialog looks like this:



Each option is specified as an `option=value` pair, for example `height=4in`, `caption="Enter Quantity"`, etc. Here is an example of a dialog with several options set. (Tip: If you enclose the entire option parameter in “smart quotes” as shown below you will be able to use regular " quotes for the individual options, if necessary.)

```
supergettext Query,“caption="Please enter the quantity:" height=104 width=350”
```

This dialog will look like this.



The table below describes each of options available with this statement.

Option	Examples	Description
title=	title="Latest Information"	With this option you can set the title of the dialog. This appears in the title bar at the top of the dialog window.
height=	height=400 height=5in height=4.5" height=10cm height=80% height=-50 height=-2cm	This option specifies the height of the dialog. The default is 200 pixels. If you supply just a number then the height is specified in pixels (72 pixels = 1 inch). If the number is followed by in or " then it is specified in inches. If the number is followed by cm it is specified in centimeters. If the number is followed by % it is specified as a percentage of the screen height, for example 50% is 1/2 of the screen height. If the height is negative then this value is the distance of the border between the top of the screen and the alert (also the bottom of the alert and the bottom of the screen). For example if your screen is 8 inches high then a height of -1in will make the alert 6 inches high (one inch at the top and one inch at the bottom).
width=	width=600 width=7in width=5.5" width=12cm width=80% width=-50 width=-2cm	This option specifies the width of the dialog. The default is 500 pixels. If you supply just a number then the width is specified in pixels (72 pixels = 1 inch). If the number is followed by in or " then it is specified in inches. If the number is followed by cm it is specified in centimeters. If the number is followed by % it is specified as a percentage of the screen width, for example 50% is 1/2 of the screen width. If the width is negative then this value is the distance of the border between the left of the screen and the alert (also the right edge of the alert and the right side of the screen). For example if your screen is 10 inches wide then a width of -1in will make the alert 8 inches wide (one inch on the left and one inch on the right).
font=	font=Tekton font="Comic Sans" font=Verdana	This option specifies the font to be used. The default is whatever the system font is on your system.
size=	size=9 size=18	This option specifies the text size (in points). The default is 12 points.
scroll=	scroll=yes scroll=thin	This option enables a scroll bar to allow viewing of large quantities of text. The options are *yes*, *thin* and *no*.
caption=	caption="Start date:"	This option specifies a message to be displayed at the top of the dialog, above the text. (This message is displayed within the dialog, not in the title bar.)
captionheight=	captionheight=2	This option specifies how many lines should be used for the caption (see above). This value defaults to 1, but you can use a larger value if you have a long caption.

Option	Examples	Description
captionfont=	captionfont=Tekton captionfont="Comic Sans" captionfont=Verdana	This option specifies the font to be used in the caption. This allows you to specify a different font for the caption and the text editing portions of the dialog.
captionsize=	captionsize=9 captionsize=18	This option specifies the text size (in points) to be used in the caption. This allows you to specify a different font size for the caption and the text editing portions of the dialog.
captionstyle=	captionstyle=italic captionstyle=bold captionstyle="bold italic"	This option specifies the text style to be used in the caption. You can choose one or more options from bold, italic, underline, outline and shadow. All of the text in the caption will be displayed in the same style, you cannot mix styles.
captioncolor=	captioncolor=#00FF00 captioncolor=red captioncolor=royalblue	This option specifies the caption text color (the default is black). If the color begins with # then you can specify any color using HTML style color tags (#RRGGBB, for example #00FF00 for green). You can also choose from this list of colors: *aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkseagreen, darkslateblue, darkslategray, darkturquoise, darkviolet, deeppink, deepskyblue, dimgray, dodgerblue, firebrick, floralwhite, forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod, gray, green, greenyellow, honeydew, hotpink, indianred, indigo, ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral, lightcyan, lightgoldenrodyellow, lightgreen, lightgrey, lightpink, lightsalmon, lightseagreen, lightskyblue, lightslategray, lightsteelblue, lightyellow, lime, limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid, mediumpurple, mediumseagreen, mediumslateblue, mediumspringgreen, mediumturquoise, mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy, oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen, paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple, red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen, seashell, sienna, silver, skyblue, slateblue, slategray, snow, springgreen, steelblue, tan, teal, thistle, tomato, turquoise, violet, wheat, white, whitesmoke, yellow, yellowgreen.*
buttons=	buttons="Yes;No;Cancel" buttons="One:50;Two:50"	This option allows you to specify up to three buttons (the default is two buttons: *Ok* and *Cancel*). Each button is separated by a semicolon, and the first button listed is the default button. The default button width is 80 pixels, you can also specify a button width in pixels by placing a colon followed by the width after the button name. In the second example to the left the buttons will be 50 pixels wide.
bgcolor=	bgcolor=#FFDDFF bgcolor=lightskyblue bgcolor=##983	This option specifies the background color (the default is gray). If the color begins with # then you can specify any color using HTML style color tags (#RRGGBB, for example #FFCCFF for lite pink). You can also choose from the same list of colors available for the color= option (see above). A third option is to use ## followed by a resource number to display a background image from Panorama's resources. You can use the **Icons & Backgrounds** wizard to find out what background images are available.

After the dialog is complete the program can use the `info("dialogtrigger")` to find out which button was clicked.

Obsolete Text Entry Statements

Since Panorama was first released in 1988 it has evolved and improved in many ways. For compatibility, we usually leave in older features even when new features make them obsolete, as is the case with the statements described in this section. However, they are still described here because you may encounter them in older databases. The `getscrap` statement displays a simple dialog (see “GETSCRAP” on page 5312).



The user types in one item of text, then presses **OK** or **Stop**. Panorama will put whatever text the user types into the clipboard. The `getscrap` statement has one parameter, the text that you want to appear at the top of the dialog. The example below uses `getscrap` to find out what check to search for.

```
getscrap "Find what check #?"
select «Check#»=val(clipboard())
```

The `getscrapok` statement is similar to `getscrap`, but the dialog has no **Stop** button (see “GETSCRAPOK” on page 5313).

The SuperChoiceDialog Statement

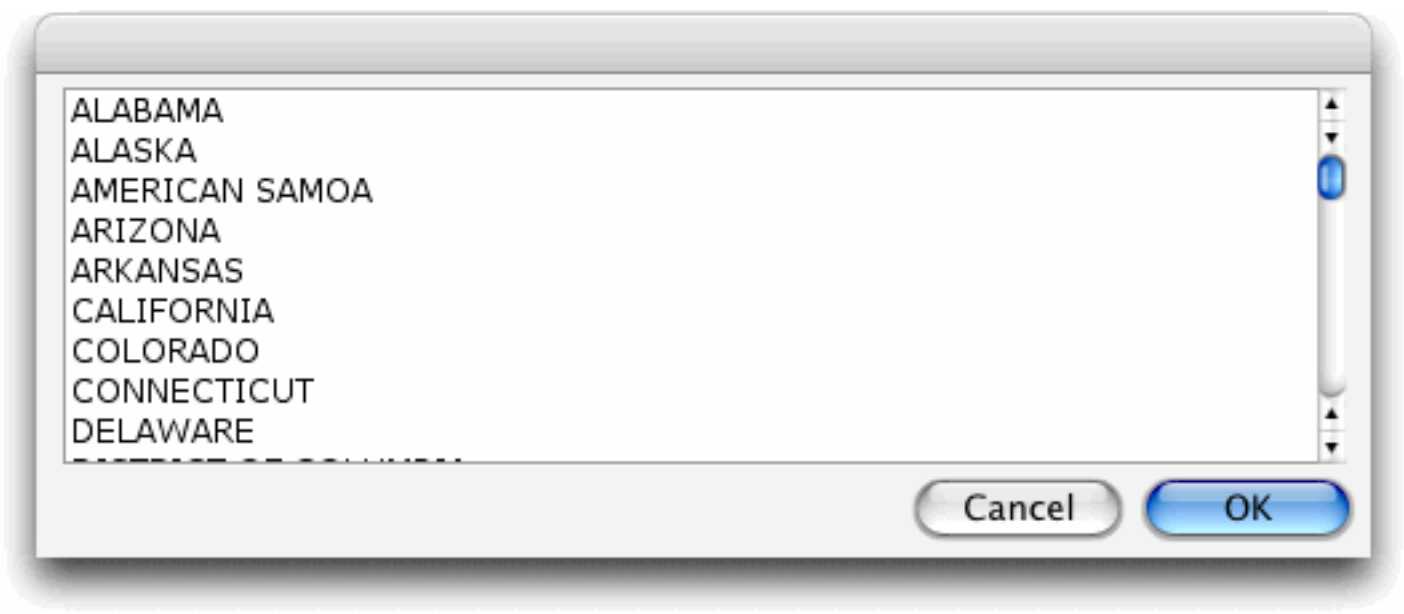
This statement displays a configurable dialog that displays a list of choices. This statement has three parameters:

```
superchoicedialog list,selection,options
```

The list parameter must be a text delimited array containing the lists of choices that will be displayed. The selection parameter must be a field or variable to accept the choice. The options is a text parameter that uses a syntax similar to an HTML tag to specify one or more options. If you just want to use the default options you don’t have to specify any options at all.

```
superchoicedialog States,theState,""
```

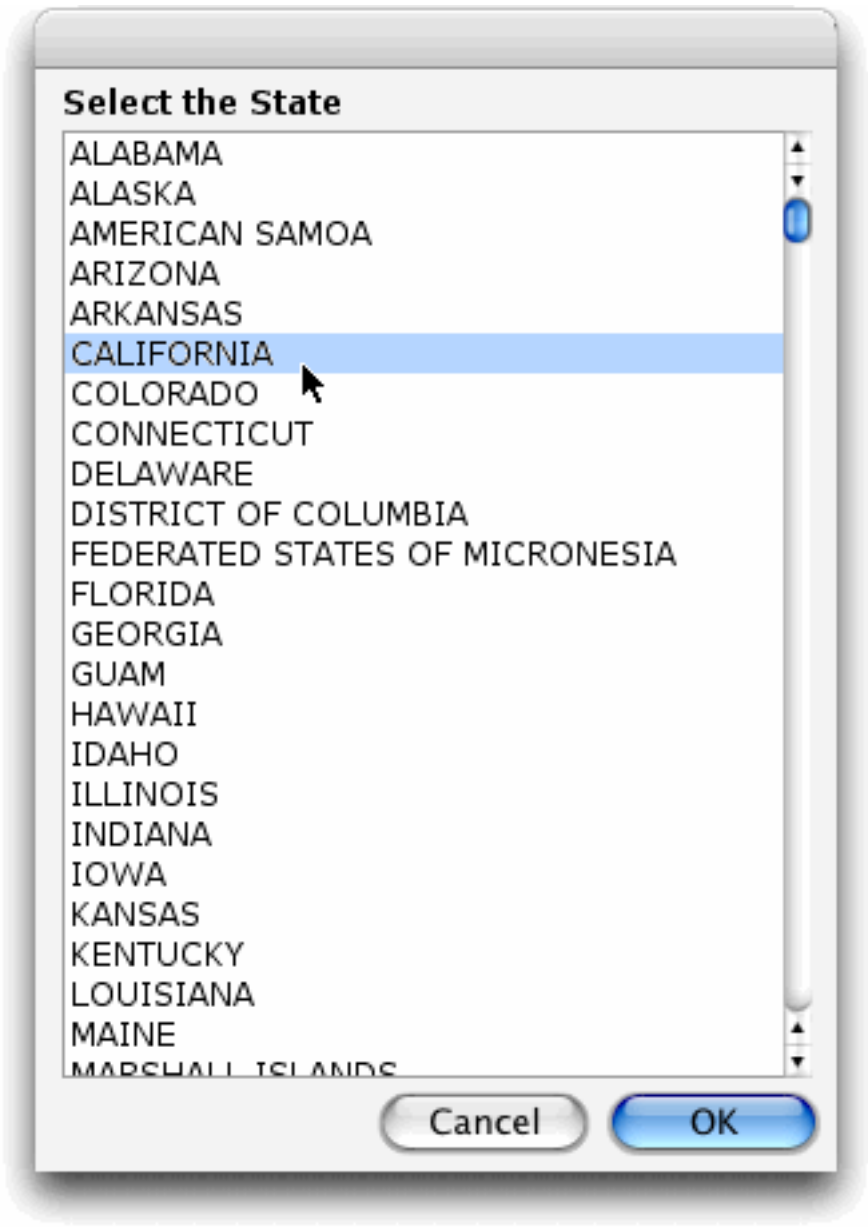
This plain dialog looks like this:



Each option is specified as an `option=value` pair, for example `height=4in`, `caption="Enter Quantity"`, etc. Here is an example of a dialog with several options set. (Tip: If you enclose the entire option parameter in “smart quotes” as shown below you will be able to use regular " quotes for the individual options, if necessary.)

```
superchoicedialog States,theState,
    “Caption="Select the State" captionstyle=bold height=6in width=3in”
```

This dialog will look like this.



The table below describes each of options available with this statement.

Option	Examples	Description
title=	title="Latest Information"	With this option you can set the title of the dialog. This appears in the title bar at the top of the dialog window.
height=	height=400 height=5in height=4.5" height=10cm height=80% height=-50 height=-2cm	This option specifies the height of the dialog. The default is 200 pixels. If you supply just a number then the height is specified in pixels (72 pixels = 1 inch). If the number is followed by in or " then it is specified in inches. If the number is followed by cm it is specified in centimeters. If the number is followed by % it is specified as a percentage of the screen height, for example 50% is 1/2 of the screen height. If the height is negative then this value is the distance of the border between the top of the screen and the alert (also the bottom of the alert and the bottom of the screen). For example if your screen is 8 inches high then a height of -1in will make the alert 6 inches high (one inch at the top and one inch at the bottom).
width=	width=600 width=7in width=5.5" width=12cm width=80% width=-50 width=-2cm	This option specifies the width of the dialog. The default is 500 pixels. If you supply just a number then the width is specified in pixels (72 pixels = 1 inch). If the number is followed by in or " then it is specified in inches. If the number is followed by cm it is specified in centimeters. If the number is followed by % it is specified as a percentage of the screen width, for example 50% is 1/2 of the screen width. If the width is negative then this value is the distance of the border between the left of the screen and the alert (also the right edge of the alert and the right side of the screen). For example if your screen is 10 inches wide then a width of -1in will make the alert 8 inches wide (one inch on the left and one inch on the right).
font=	font=Tekton font="Comic Sans" font=Verdana	This option specifies the font to be used for the list of choices. The default is whatever the system font is on your system.
size=	size=9 size=18	This option specifies the text size (in points) for the list of choices. The default is 12 points.
caption=	caption="Start date:"	This option specifies a message to be displayed at the top of the dialog, above the text. (This message is displayed within the dialog, not in the title bar.)
captionheight=	captionheight=2	This option specifies how many lines should be used for the caption (see above). This value defaults to 1, but you can use a larger value if you have a long caption.

Option	Examples	Description
captionfont=	captionfont=Tekton captionfont="Comic Sans" captionfont=Verdana	This option specifies the font to be used in the caption. This allows you to specify a different font for the caption and the text editing portions of the dialog.
captionsize=	captionsize=9 captionsize=18	This option specifies the text size (in points) to be used in the caption. This allows you to specify a different font size for the caption and the text editing portions of the dialog.
captionstyle=	captionstyle=italic captionstyle=bold captionstyle="bold italic"	This option specifies the text style to be used in the caption. You can choose one or more options from bold, italic, underline, outline and shadow. All of the text in the caption will be displayed in the same style, you cannot mix styles.
captioncolor=	captioncolor=#00FF00 captioncolor=red captioncolor=royalblue	This option specifies the caption text color (the default is black). If the color begins with # then you can specify any color using HTML style color tags (#RRGGBB, for example #00FF00 for green). You can also choose from this list of colors: *aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkseagreen, darkslateblue, darkslategray, darkturquoise, darkviolet, deeppink, deepskyblue, dimgray, dodgerblue, firebrick, floralwhite, forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod, gray, green, greenyellow, honeydew, hotpink, indianred, indigo, ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral, lightcyan, lightgoldenrodyellow, lightgreen, lightgrey, lightpink, lightsalmon, lightseagreen, lightskyblue, lightslategray, lightsteelblue, lightyellow, lime, limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid, mediumpurple, mediumseagreen, mediumslateblue, mediumspringgreen, mediumturquoise, mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy, oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen, paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple, red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen, seashell, sienna, silver, skyblue, slateblue, slategray, snow, springgreen, steelblue, tan, teal, thistle, tomato, turquoise, violet, wheat, white, whitesmoke, yellow, yellowgreen.*
buttons=	buttons="Yes;No;Cancel" buttons="One:50;Two:50"	This option allows you to specify up to three buttons (the default is two buttons: *Ok* and *Cancel*). Each button is separated by a semicolon, and the first button listed is the default button. The default button width is 80 pixels, you can also specify a button width in pixels by placing a colon followed by the width after the button name. In the second example to the left the buttons will be 50 pixels wide.
bgcolor=	bgcolor=#FFDDFF bgcolor=lightskyblue bgcolor=##983	This option specifies the background color (the default is gray). If the color begins with # then you can specify any color using HTML style color tags (#RRGGBB, for example #FFCCFF for lite pink). You can also choose from the same list of colors available for the color= option (see above). A third option is to use ## followed by a resource number to display a background image from Panorama's resources. You can use the **Icons & Backgrounds** wizard to find out what background images are available.

After the dialog is complete the program can use the `info("dialogtrigger")` to find out which button was clicked.

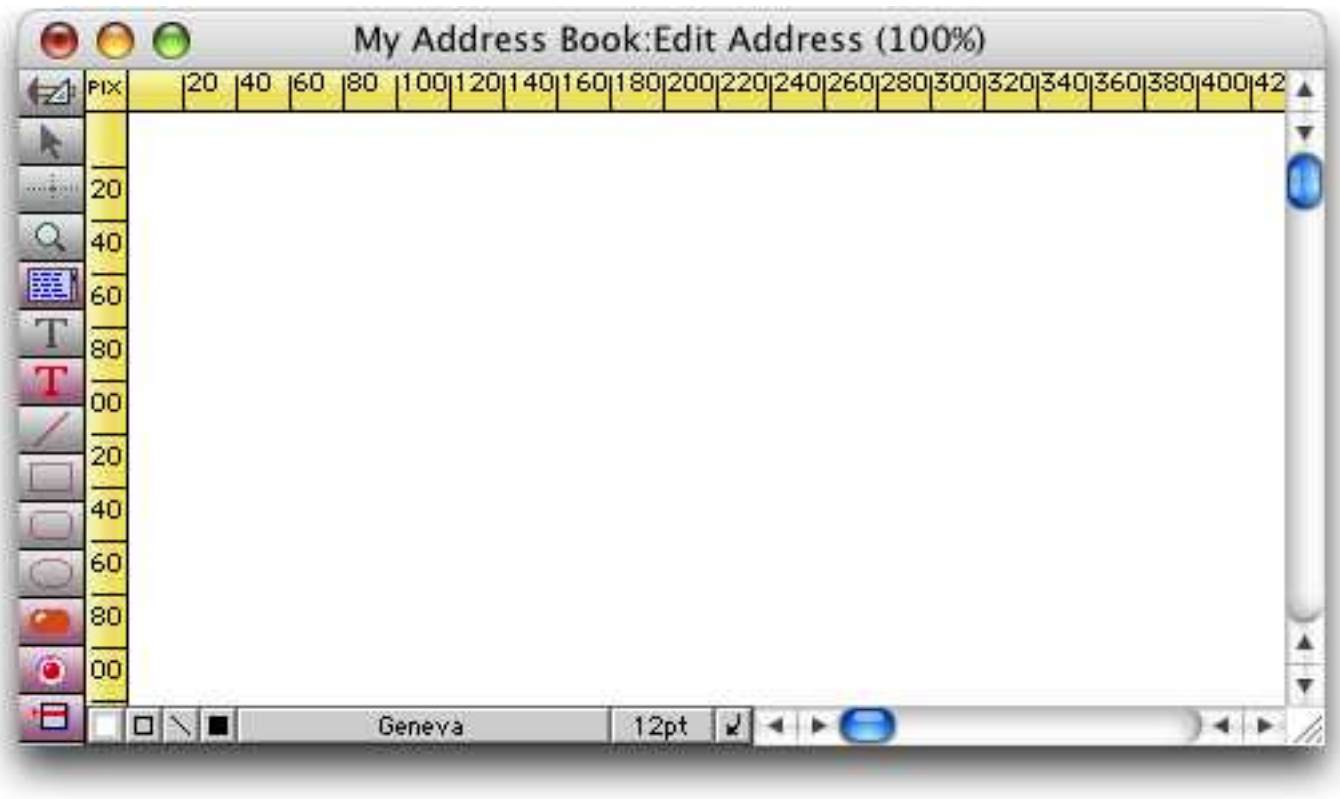
Custom Dialogs

Most of the dialogs you will need will not fit into the “off the shelf” category described in the last section. When an off the shelf dialog won’t cut it, you can build your own dialogs using standard Panorama forms. Forms used as dialogs are created just like any other form, using text objects, buttons, lists, pop-up menus, pictures, etc. In fact, any form can be used as a dialog.

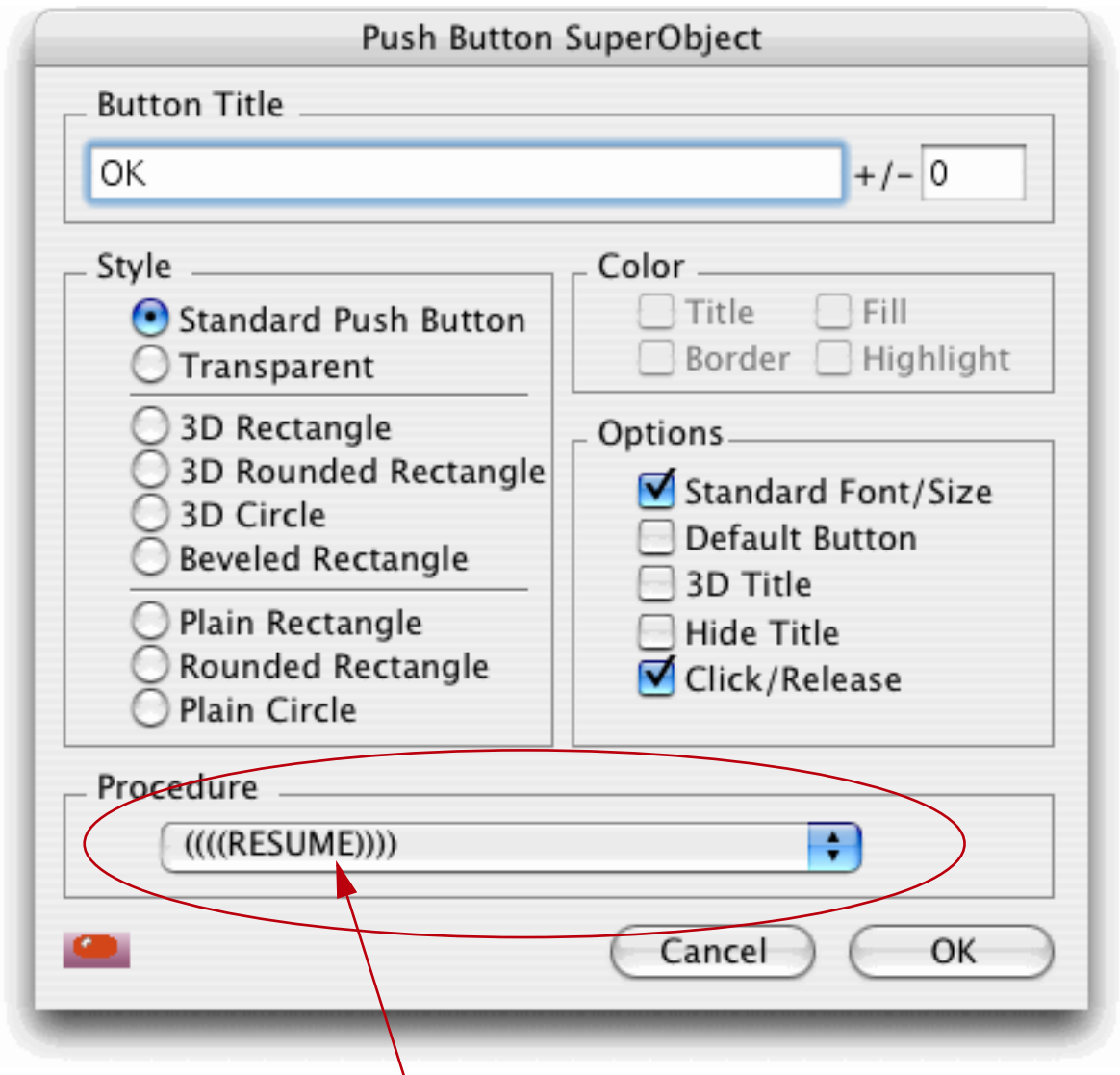
If a dialog is going to be used to collect information that is independent from the database (i.e. not in a database field) your dialog should use SuperObjects™ that are linked to global variables. You can use the SuperObject Text Editor, Pop-Up Menus, Data Buttons and Lists with global variables. When the dialog is closed, the procedure can use the information the user entered into these global variables any way it wants to.

Preparing a Form for Use as a Dialog

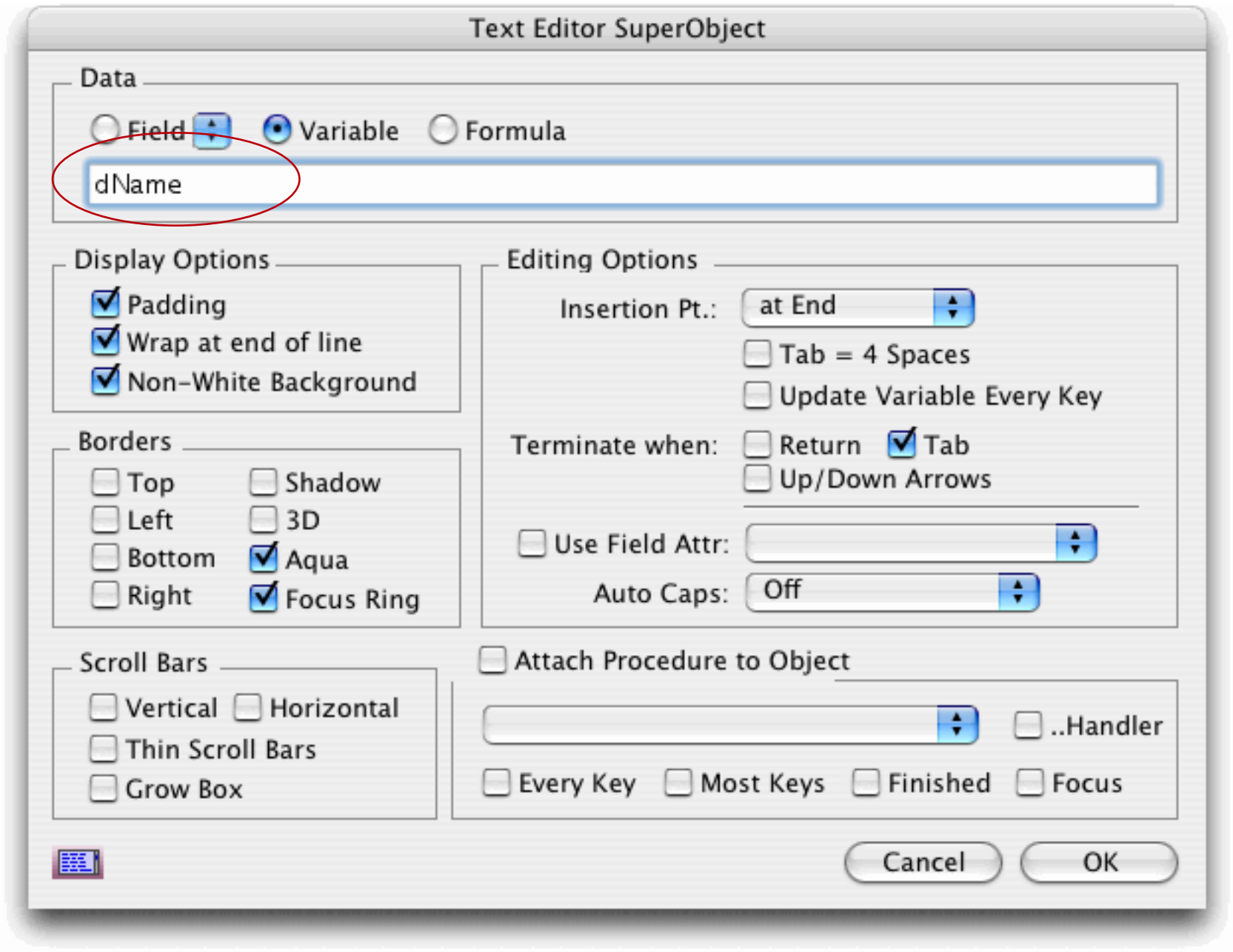
The first step in designing a custom dialog is to create a normal form (see “Creating a New Form, Crosstab or Procedure” on page 236).



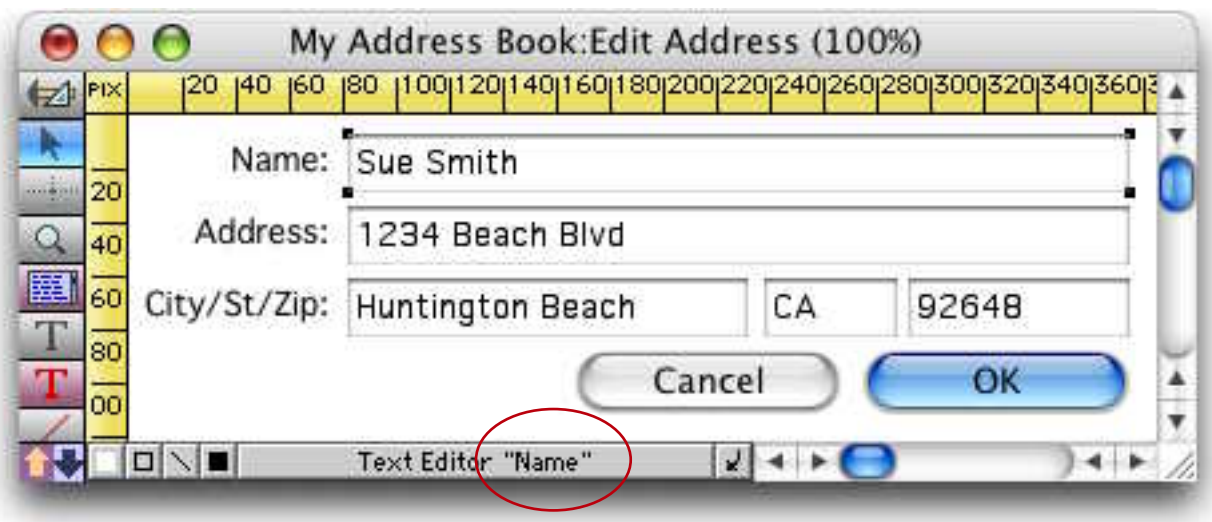
Next use the **Push Button** tool (see "Super Object Push Button" on page 813) to create the **OK** and **Cancel** buttons. Both of these buttons should be configured to resume the current procedure when clicked. To set up this configuration on the Macintosh hold down the **Control** key and click on the Procedure pop-up menu. To setup this configuration on Windows simply right click on the Procedure pop-up menu.



The next step is to add any text editing boxes that are needed using the Text Editor SuperObject (see "Text Editor SuperObject" on page 639). If a text editing box is going to be used to edit a database field it should be configured to edit a variable. The variable should have the same name as the field but with a prefix of **d**. The example below shows the configuration for editing the Name field.

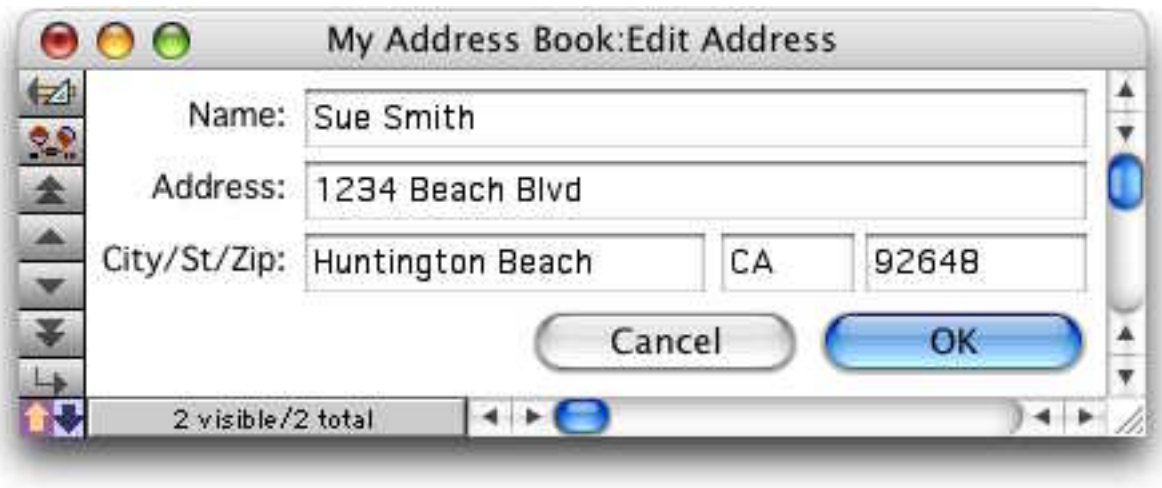


An optional step at this point is to give some or all of the text editor objects a name (see "Object Type / Object Name" on page 533 to learn how to assign a name to any object). In particular you'll probably want to assign a name to the upper left object, the object that will become the default for text entry when the dialog is first opened.

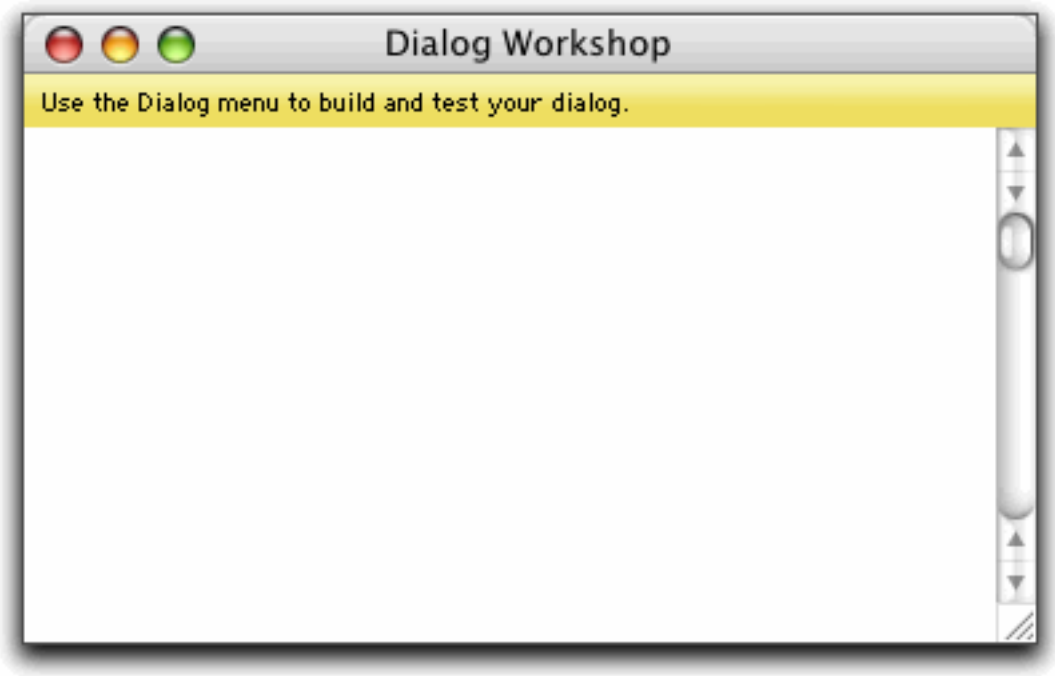


At this point you can add text captions and any checkboxes or radio buttons. To configure a checkbox or radio button to edit a database field it should be a variable with the same name as the field but with a **d** prefix, just as for the Text Editor SuperObjects. Otherwise the variable can be any name you like.

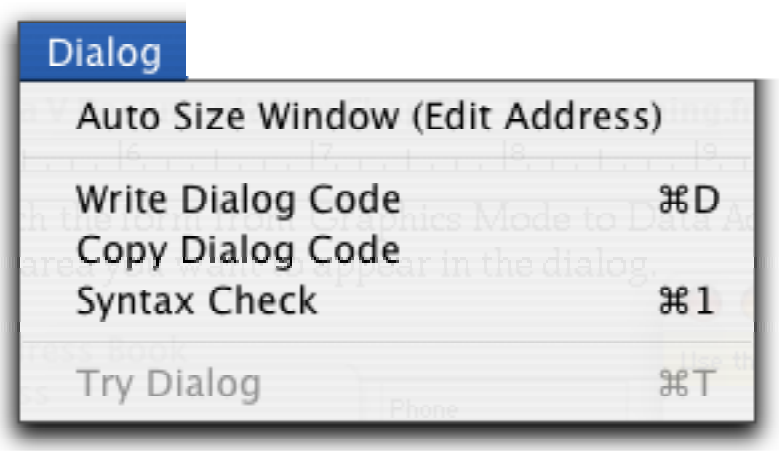
Once all the objects on the form are complete, switch the form from Graphics Mode to Data Access Mode. Then adjust the size of the window to show just the area you want to appear in the dialog.



Now open the **Dialog Workshop** (in the Developer Tools subfolder of the Wizard menu). This open a small window.

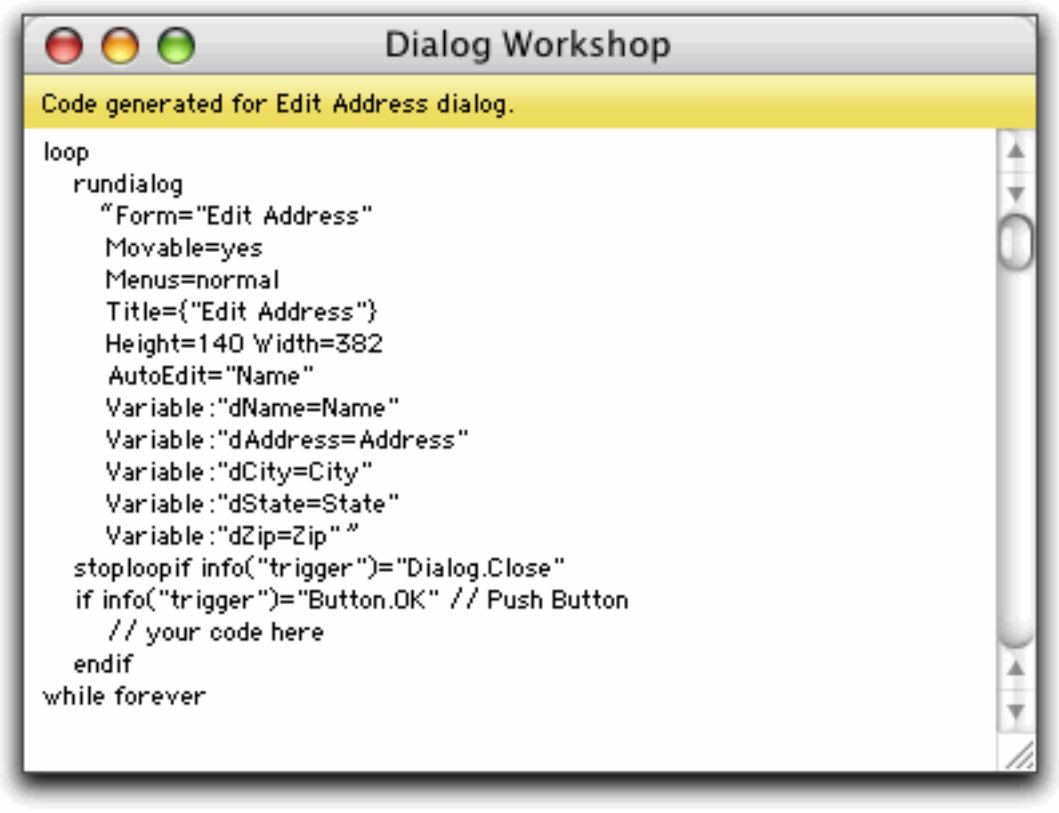


The guts of this workshop is in the **Dialog** menu.

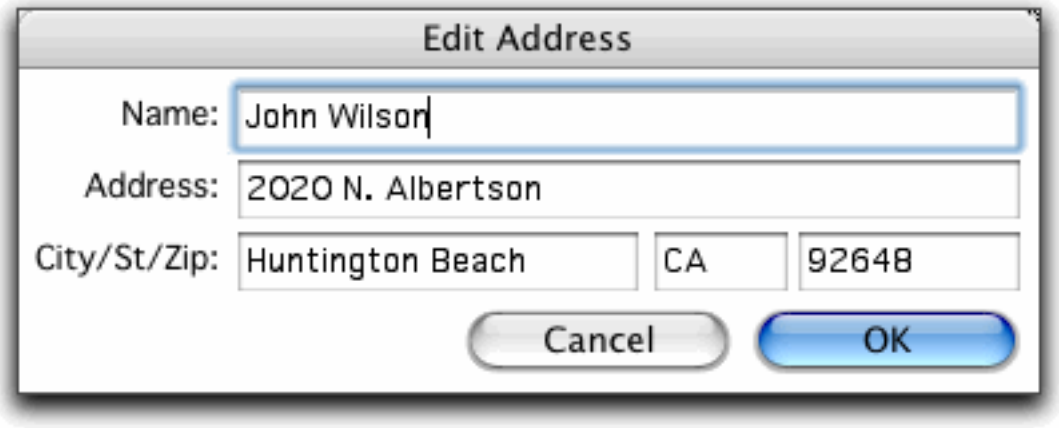


If you haven't already set the size of your window you can use the **Auto Size Window** command to adjust the form size. The wizard will try its best to adjust the form to an appropriate size.

When your form is ready to go choose the **Write Dialog Code** command. This command will analyze the form and write the code for a new procedure for you!

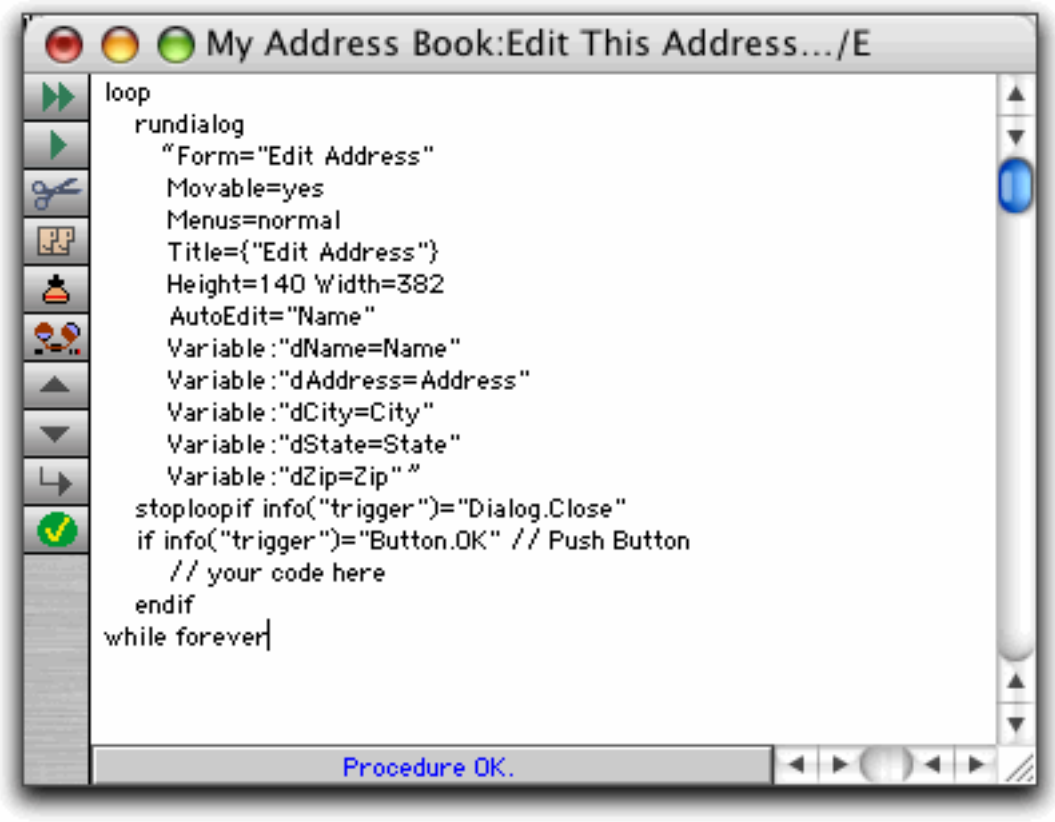


To try out your new dialog use the **Try Dialog** command. This allows you to see exactly what your new dialog looks like and how it will work.



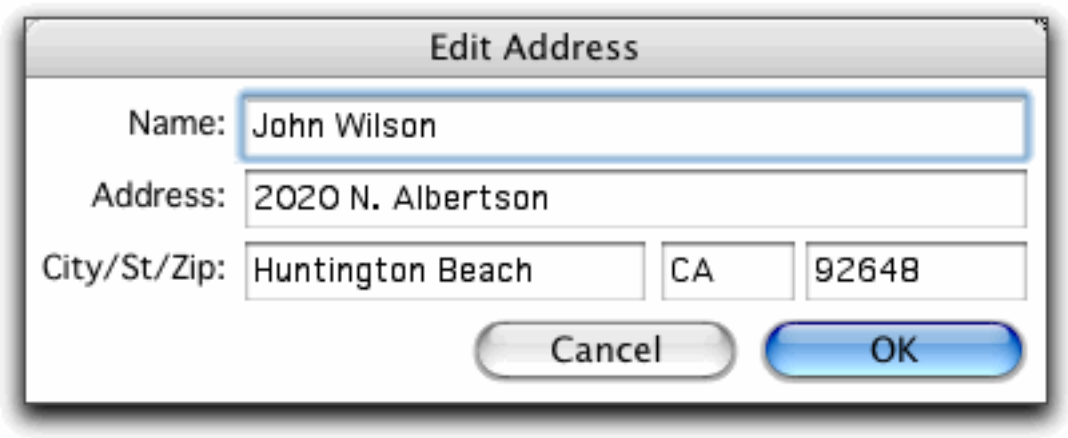
Next, close the form window. However, make sure that you leave another window in the database open, so that the database itself doesn't become closed.

If your dialog works the way you expected the next step is to copy the procedure into the database itself. You can copy your new procedure to the clipboard with the **Copy Dialog Code** command. If the dialog will be opened by clicking on a button or selecting an item in the **Action** menu then you'll need to create a new procedure (see "Writing a Procedure from Scratch" on page 1352). Once the procedure is opened you can use the **Paste** command to insert the automatically generated code into it.



For most dialogs, that's it! Once you've tried one or two dialogs you'll find that you can create a new dialog in just a few minutes.

Before you actually use the dialog you'll need to close the form you created. Make sure that some other window in this database remains open. To use the dialog you can simply select the new procedure from the **Action** menu (or click on the button or whatever). The dialog will appear, and will automatically be centered over the current window (or, if the window is too small, centered in the middle of the screen). Any cells or buttons associated with a field will automatically be filled in with the original data from the current record.



When you press the **OK** or **Cancel** buttons the dialog will close automatically, and if the **OK** button was pressed any fields that were modified will be updated.

Customizing the Dialog Code

The program automatically generated by the **Dialog Workshop** will handle many common dialogs as-is. However, this program is designed to be flexible and to allow you to modify almost all aspects of it's behavior. The key to this procedure is the `RunDialog` statement. Let's start by looking at the automatically generated code to see how it works. Here is the simplest possible dialog processing code — just four lines.

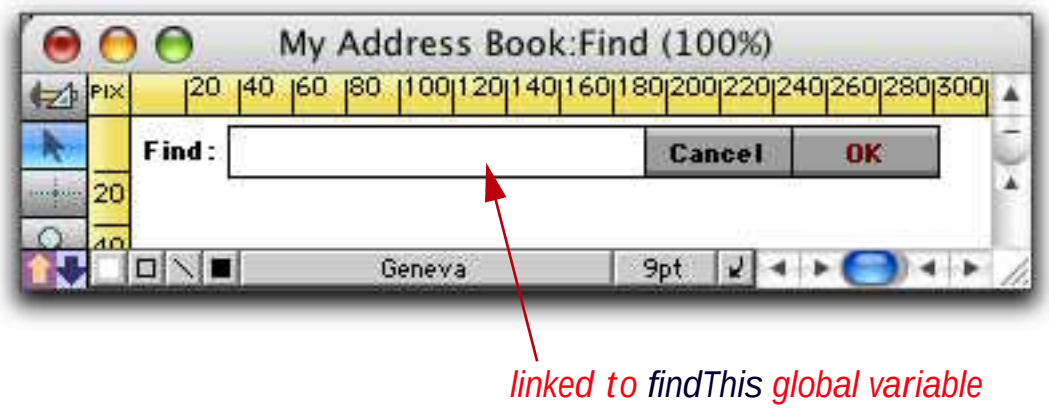
```
loop
   rundialog,{Form=Address Height=120 Width=400 Movable=yes Menus=normal}
   stoploopif info("trigger")="Dialog.Close"
while forever
```

In this most basic form the code is a simple loop that executes the `rundialog` statement each time through the loop. This statement handles most of the work. Whenever something happens (a button is pressed, the **Enter** or **Tab** key is pressed), the `rundialog` statement analyzes it and then returns to the loop. Your code in the loop can find out what is happening by examining the result of the `info("trigger")` function.

The `rundialog` statement has one parameter — a list of options. The options is a text parameter that uses a syntax similar to an HTML tag to specify one or more options. Each option is specified as an `option=value` pair, for example `form="Entry"`, `title="Enter Quantity"`, etc. (Tip: If you enclose the entire option parameter in "smart quotes" you will be able to use regular " quotes for the individual options, if necessary.)

This very simple program shown above only does one thing — stop the loop if `info("trigger")` becomes Dialog.Close. Our code doesn't need to do anything else because the `rundialog` statement will do everything for us including opening and closing the dialog.

To illustrate how this procedure can be expanded, let's consider the dialog show below. This dialog is designed to allow the user to type in a word or phrase they want to search for.



Here's the procedure that can handle this dialog. The automatically generated code is shown in blue, the custom code in purple.

```
global findThis
findThis=""
loop
   rundialog {Form="Find" Height=45 Width=346 movable=yes menus=normal AutoEdit="Find"}
   stoploopif info("trigger")="Dialog.Close"
   if info("trigger") contains "Dialog.OK"
      find exportline() contains findThis
   endif
while forever
```

The first two lines simply create the global variable named findThis and assign it a value. This is the variable the user will type into. The other new code checks to see if the **OK** button has been pressed, and if so, performs the search.

We can modify this code further to perform error checking. This version of the program checks to make sure that the user has typed something to search for. If not a message is displayed. More importantly, the `settrigger` statement (see "SETTRIGGER" on page 5752) is set to `""` instead of `"Dialog.OK"`. This tells the `rundialog` statement not to close the dialog window.

```
global findThis
findThis=""
loop
   rundialog {Form="Find" Height=45 Width=346 movable=yes menus=normal AutoEdit="Find"}
   stoploopif info("trigger")="Dialog.Close"
   if info("trigger") contains "Dialog.OK"
      if findThis=""
         message "You must enter something to search for!"
         settrigger ""
      else
         find exportline() contains findThis
      endif
   endif
while forever
```

The way the code above is written the find operation happens while the dialog is still open. If you wanted the dialog to close first you would need to rewrite the program like this.

```
global findThis
findThis=""
loop
   rundialog {Form="Find" Height=45 Width=346 movable=yes menus=normal AutoEdit="Find"}
   stoploopif info("trigger")="Dialog.Close"
   if info("trigger") contains "Dialog.OK"
      if findThis=""
         message "You must enter something to search for!"
         settrigger ""
      endif
   endif
while forever
if dlgResult="Ok"
   find exportline() contains findThis
endif
```

When the dialog is finished the dlgResult value will contain either the value Ok or Cancel.

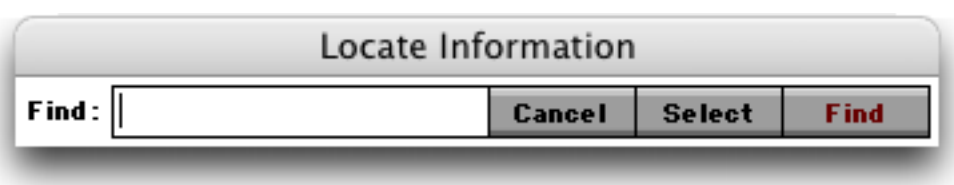
You can add additional buttons to the dialog that perform some action within the dialog. For example, you could add a **Clear** button to this dialog.



When a button other than the **OK** or **Cancel** button is pressed the `info("trigger")` function will return Button. followed by the title of the button.

```
global findThis
findThis=""
loop
   rundialog {Form="Find" Height=45 Width=346 movable=yes menus=normal AutoEdit="Find"}
   stoploopif info("trigger")="Dialog.Close"
   if info("trigger") contains "Dialog.OK"
      if findThis=""
         message "You must enter something to search for!"
         settrigger ""
      else
         find exportline() contains findThis
      endif
   endif
   if info("trigger") = "Button.Clear"
      superobject "Find","Open"
      activesuperobject "setselection",0,32767
      activesuperobject "clear"
   endif
while forever
```

Sometimes you may want to have more than one button that terminates the dialog successfully. In this case both the **Find** and **Select** buttons cause the dialog to close.



Here is the revised program to handle this dialog.

```
global findThis
findThis=""
loop
   rundialog
      {Form="Find" Height=45 Width=346 movable=yes menus=normal
         AutoEdit="Find" OkButton="Find"}
   stoploopif info("trigger")="Dialog.Close"
   if info("trigger") contains "Dialog.OK"
      if findThis=""
         message "You must enter something to search for!"
         settrigger ""
      else
         find exportline() contains findThis
      endif
   endif
   if info("trigger") contains "Button.Select"
      superobjectclose
      if findThis=""
         message "You must enter something to select!"
      else
         select exportline() contains findThis
         settrigger "Dialog.OK"
      endif
   endif
while forever
```

The first thing to notice is the option `OkButton="Find"` on the fourth line. Options are discussed in more detail in the next section, but for now this option tells the `rundialog` statement to treat the **Find** button as if it was the **OK** button. That means that when you press the **Enter** or **Return** key it will be treated just as if you had pressed the **Find** button. It also means that when the Find button is pressed the `info("trigger")` function will return Dialog.OK, not Button.Find (see line 6).

The additions to handle the **Select** button are fairly routine. However, notice the fourth line from the bottom, `settrigger "Dialog.OK"`. This line tells the `rundialog` statement to go ahead and close the dialog window.

In some cases you may need to perform some initialization after the dialog window has opened. Usually this involves some sort of graphic manipulation — moving an object or changing a font (see “Programming Graphic Objects on the Fly” on page 1753). (Almost any other kind of non-graphic initialization can simply be performed before the loop begins.) We don’t have an example of this, but the basic idea is to check for the trigger value of Dialog.Initialize.

```
loop
   rundialog {Form="Find" Height=45 Width=346 movable=yes menus=normal AutoEdit="Find"}
   stoploopif info("trigger")="Dialog.Close"
   if info("trigger") contains "Dialog.Initialize"
      /*
         ... code to initialize procedure goes here ...
      */
   endif
while forever
```

Sometimes you may want to handle the **Cancel** button in a special way. The revised procedure below checks to see if the user has typed anything in, and if so, asks them to confirm that they really do want to cancel.

```
global findThis
findThis=""
loop
   rundialog {Form="Find" Height=45 Width=346 movable=yes menus=normal
      AutoEdit="Find" OkButton="Find"}
   stoploopif info("trigger")="Dialog.Close"
   if info("trigger") contains "Dialog.OK"
      if findThis=""
         message "You must enter something to search for!"
         settrigger ""
      else
         find exportline() contains findThis
      endif
   endif
   if info("trigger") contains "Button.Select"
      superobjectclose
      if findThis=""
         message "You must enter something to select!"
      else
         select exportline() contains findThis
         settrigger "Dialog.OK"
      endif
   endif
   if info("trigger") = "Dialog.Cancel"
      superobjectclose
      if findThis≠""
         alert 1014,"Are you sure you want to cancel?"
         if info("dialogtrigger") contains "no"
            settrigger ""     /* tell .dialog to stop the cancel! */
            superobject "Find","Open"
         endif
      endif
   endif
while forever
```

All of the examples have shown push buttons, but you can also check for and handle any type of button, list, or even a Text Editor SuperObject that triggers the `rundialog` statement using *((((Resume))))*. Any object that has the *((((Resume))))* option set can be handled by your custom dialog code.

Options to the RunDialog Statement

The `rundialog` statement has one parameter. This parameter contains a series of `name=value` pairs that tell the **.dialog** subroutine how to process the dialog. At a minimum this parameter must include three parameters: the form name, the form height, and the form width.

```
{Form=Address Height=120 Width=400}
```

If the form name (or any value) contains a space it must be surrounded with quotes, like this.

```
{Form="Time Card" Height=120 Width=400}
```

In addition to the three basic name / value pairs there are also about a dozen other optional name / value pairs that you can specify to customize the appearance and behavior of your dialog.

The `height` and `width` options are normally specified in pixels. (One inch is equal to 72 pixels). However, you can also specify these values in inches (for example `2"` or `3in`) , centimeters (`12cm`) or as a percentage of the screen width or height (`75%`). If the value is negative then the dialog size will depend on the height and / or width of the screen. For example, `height=-72` specifies a dialog with 1 inch above and below the dialog. If the dialog has a variable size (either a percentage or negative value) then you should make the form elastic (see “Elastic Forms” on page 912) so that it can adjust to whatever screen size is available.

The `movable` option allows you to create a dialog with a drag bar that can be moved around on the screen. The value for this option should be yes or no, for example

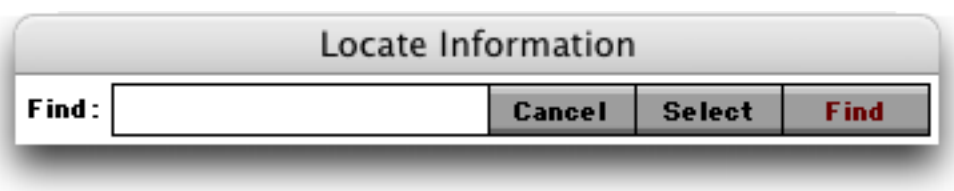
```
movable=yes
```

The default is yes, and we no longer recommend turning this option off.

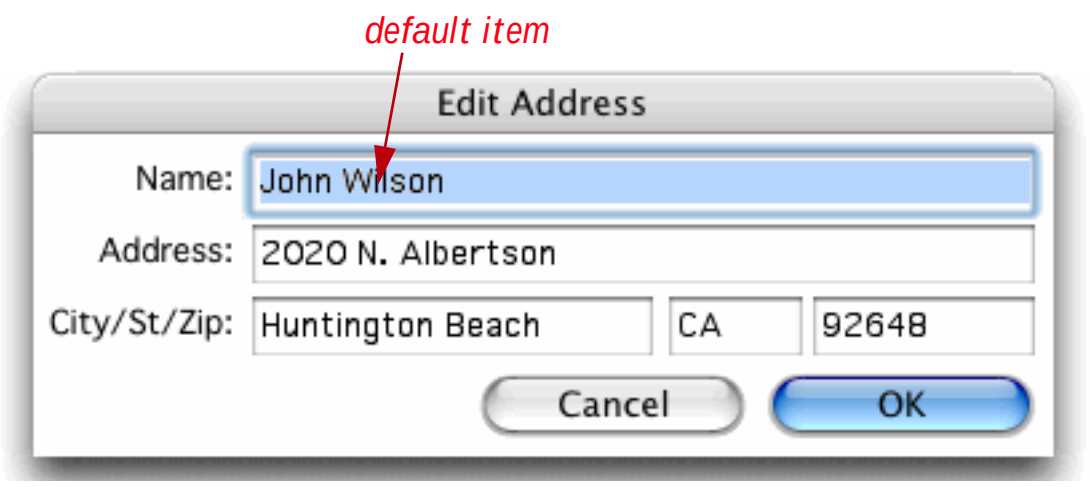
If you don’t give the dialog a title it will use the name of the form, as shown above. You can override this and specify another title using the `windowtitle` option, like this.

```
windowtitle="Locate Information"
```

The dialog will appear with the specified name in the title bar.



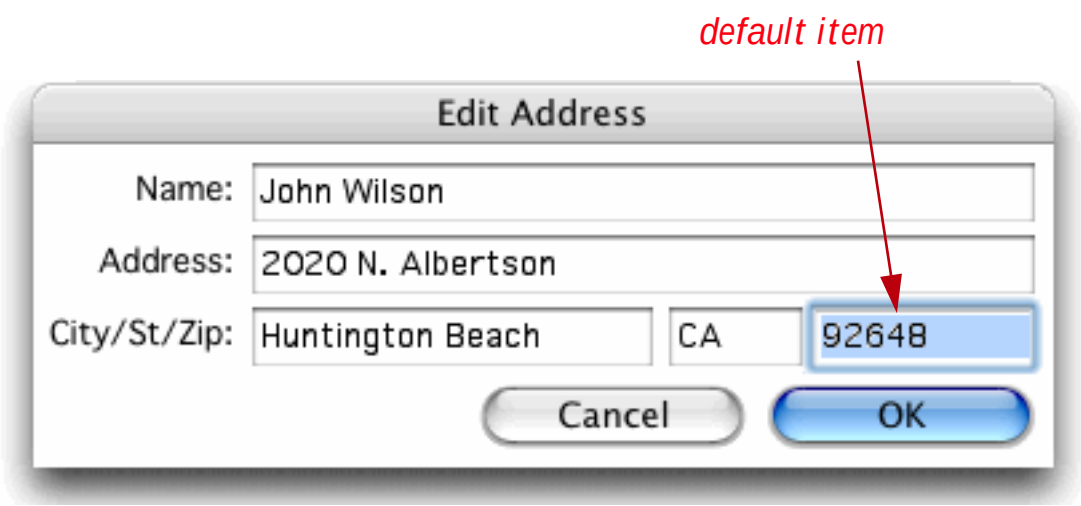
When a dialog has more than one editable text item normally the top left item is the default item where you will begin typing.



If the Text Editor SuperObjects are named (see "Object Type / Object Name" on page 533) you can override this default and specify a different default editing item with the `autoedit` option.

```
autoedit=Zip
```

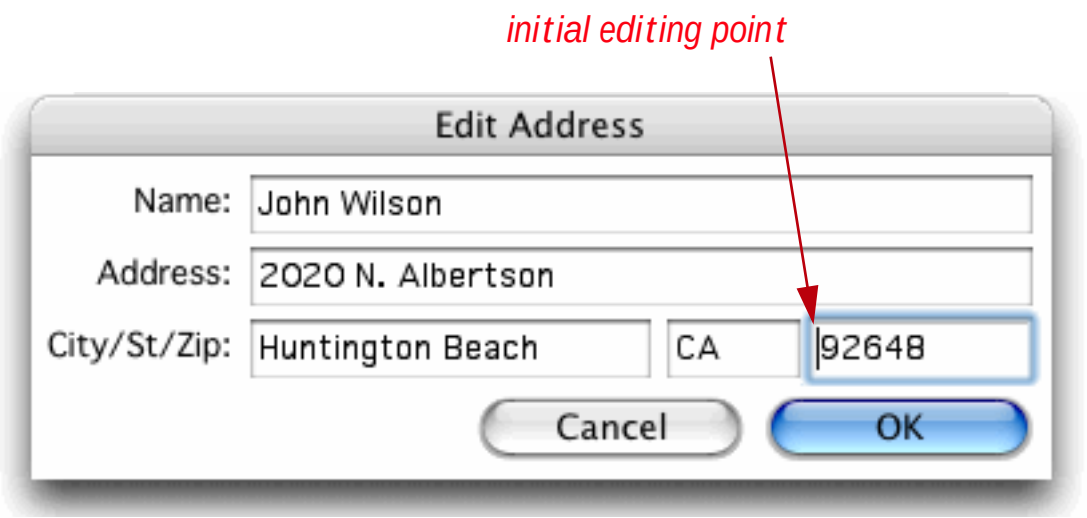
With this option set the Zip Code becomes the default item.



The `autoeditstart` and `autoeditend` options control what text is initially selected in the default item.

```
autoeditstart=0 autoeditend=0
```

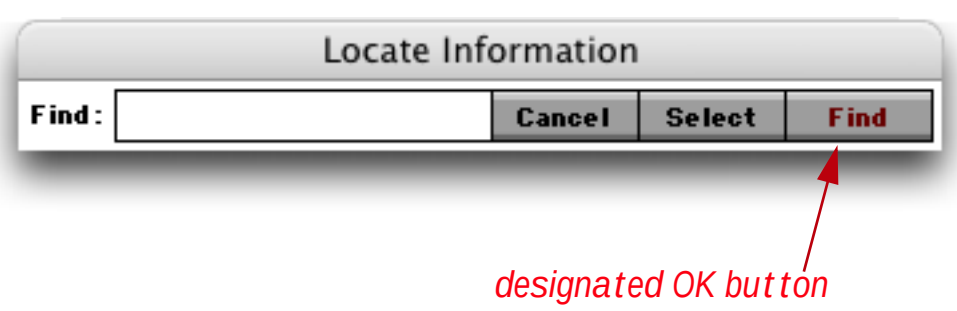
If both of these values are set to zero the initial editing point will be at the beginning of the text. If both of these are set to a large value like 9999 the initial editing point will be at the end of the text.



The `okbutton` option allows you to change what button is considered the **OK** button.

```
okbutton=Find
```

The **OK** button is usually named OK but it can be changed to any button on the form. When the button designated as the **OK** button is pressed the `info("trigger")` function will return Dialog.OK, even if the actual button has a different name. In addition, pressing the **Enter** or **Return** key will be treated the same as clicking on whatever button has been designated as the **OK** button.



The `cancelbutton` option allows you to change what button is considered the **Cancel** button.

```
cancelbutton=Stop
```

The **Cancel** button is usually named Cancel but it can be changed to any button on the form. When the button designated as the **Cancel** button is pressed the `info("trigger")` function will return Dialog.Cancel even if the actual button has a different name.

The `timeout` option allows you to put a time limit on a dialog. The timeout value is specified in seconds, so this option will cause the dialog to time out in two minutes.

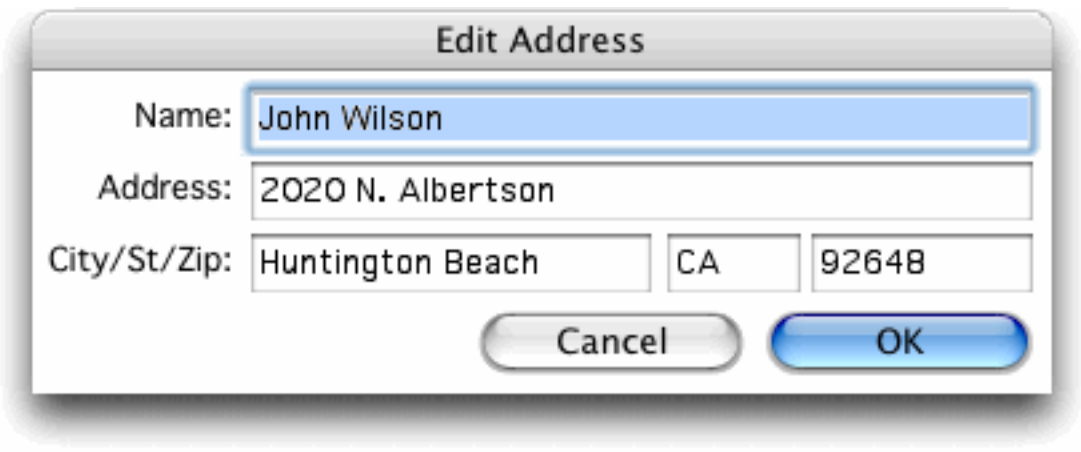
```
timeout=120
```

If the dialog is still open after two minutes it will close automatically as if the designated Ok button was pressed.

Editing Database Information with a Dialog

Editing database information with a dialog that has **OK** and **Cancel** buttons takes some extra effort. You can't simply edit the data directly because if the user presses the **Cancel** button you must be able to restore the original data. The solution is to copy the data from the database into variables, edit the variables, and then only copy the data back into the database if the **OK** button is pressed. You can write code for all this yourself, but it's easier to let the `rundialog` statement take care of it all for you.

To illustrate this, consider this dialog for editing an address.



The dialog edits five database fields — Name, Address, City, State and Zip. If you followed instructions carefully you have set up the SuperObject Text Editors in this dialog to edit five corresponding variables — dName, dAddress, dCity, dState and dZip (see "Preparing a Form for Use as a Dialog" on page 1616). Now one way to set this up would be to write the code to transfer the data back and forth yourself.

```
global dName,dAddress,dCity,dState,dZip
dName=Name
dAddress=Address
dCity=City
dState=State
dZip=Zip
loop
   rundialog {Form=Address Height=120 Width=400 menus=normal AutoEdit=Name}
   stoploopif info("trigger")="Dialog.Close"
   if info("trigger") contains "Dialog.OK"
      Name=dName
      Address=dAddress
      City=dCity
      State=dState
      Zip=dZip
   endif
while forever
```

This is a lot of extra work, though, because you have to type the field names twice and the variable names three times! Another option is to declare the relationship between the fields and variables as part of the option parameter to the `rundialog` statement. Each declaration takes the form

```
Variable:"<variable>=<field>"
```

Here is our revised procedure. A lot shorter, eh? Make sure that these declarations are in the parameter to the `rundialog` statement, between the `{` and `}` characters. On the other hand, if you set up your form correctly the **Dialog Workshop** will write all of the declarations for you, completely automatically!

```
loop
   rundialog {Form=Address Height=120 Width=400 AutoEdit=Name
      Variable:"dName=Name"
      Variable:"dAddress=Address"
      Variable:"dCity=City"
      Variable:"dState=State"
      Variable:"dZip=Zip"}
   stoploopif info("trigger")="Dialog.Close"
while forever
```

Sometimes the data needs to be converted in addition to being copied. Any time a dialog needs to edit a numeric or date field the declaration needs to include the functions for converting in both directions. Here's how a numeric Amount field and date StartDate field would be handled.

```
Variable:"val(«dAmount»)=str(«Amount»)"
Variable:"date(«dStartDate»)=datepattern(«StartDate»,“mm/dd/yy”)"
```

When conversion functions are used the variable and field names must always be enclosed in « and » chevrons (see “Special Characters” on page 1202). The chevrons must be included even if the variable or field name doesn't contain any blanks or punctuation. If the chevrons are omitted an error will occur when you try to open the dialog.

The `rundialog` statement normally creates the variables you specify as global variables when the dialog is opened (see “Long Life Variables” on page 1372). Using the `variabletype` option you can specify that another type of variable be created instead. The only option that makes any sense here is fileglobal.

```
variabletype=fileglobal
```

Custom Dialog Menus

The `rundialog` statement normally displays only the **Apple** (Mac only) and **Edit** menus when a dialog is open. This allows you to cut, copy and paste text within the menu. If you want to turn off all the menus, add `menu=none` to the list of options.

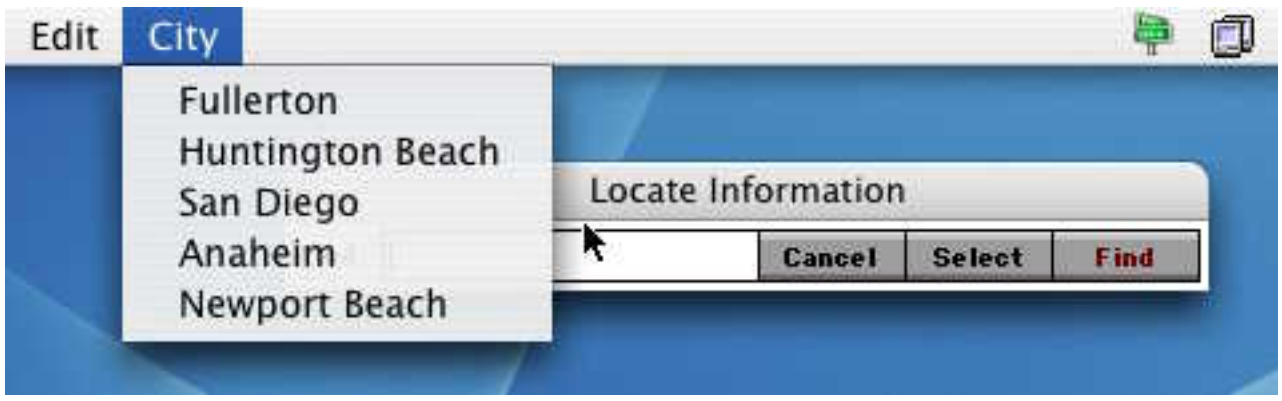
You can also specify your own completely custom menus using Panorama's Live Menu feature (see "Live Menus" on page 1465). Use the menu= option followed by the formula you want to use, like this:

```
rundialog " ... other options ... menu={formula} ... other options ... "
```

In designing the formula you'll need to use the same rules as the for the second parameter of the `filemenubar` and `windowmenubar` statements (see "The FileMenuBar Statement" on page 1465). Specifically, you'll usually want to use the `menu(`, `menuitems(`, `arraymenu(` and other similar functions to assemble the custom menu items. The example below adds a custom **City** menu when the dialog is open. This menu will list all of the cities in the database.

```
rundialog "Form="Find/Select" Height=45 Width=346 AutoEdit="Find" OkButton="Find"
    menus={menu("City")+arraymenu(listchoices(City,¶))}
    windowtitle="Locate Information""
```

Here is the dialog with the **City** menu pulled down. (Notice that the `rundialog` statement automatically includes the **Edit** menu.)

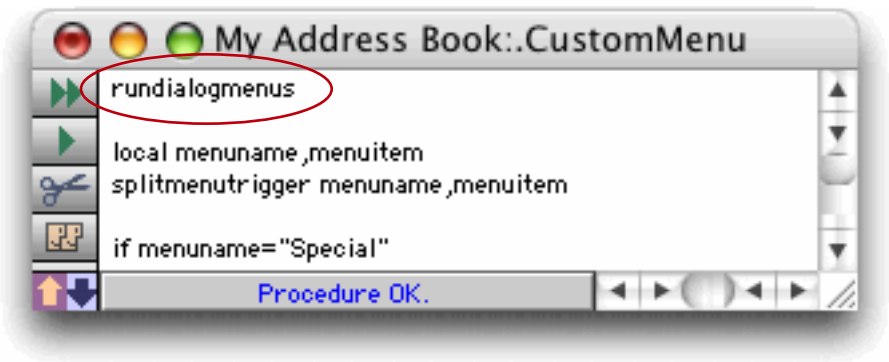


To make this custom menu work you'll need to add code to the dialog loop to check for the menu. This code is identical to the code used for the **.CustomMenu** procedure (see "The .CustomMenuProcedure" on page 1494). The purple code below shows a typical way to handle the menu.

```
global findThis
findThis=""
loop
    rundialog "Form="Find/Select" Height=45 Width=346 AutoEdit="Find" OkButton="Find"
        menus={menu("City")+arraymenu(listchoices(City,¶))}
        windowtitle="Locate Information""
    stoploopif info("trigger")="Dialog.Close"
    if info("trigger") beginswith "menu."
        local menuname,menuitem
        splitmenutrigger menuname,menuitem
        if menuname="City"
            superobject "Find","Open"
            activesuperobject "inserttext",menuitem
        endif
    endif

    ... rest of dialog event loop ...
```

You'll also need to make one change to the **.CustomMenu** code for this database. At the beginning of this procedure add one line — `rundialogmenus`.



Now the custom menu code is complete. When you open this dialog, you can use the **City** menu to type the name of a city into the dialog.

Accessing and Modifying the Database Structure (Fields)

Usually the database field structure is set up in advance, and procedures simply work with the fields as they have been defined. However, for single user databases it is possible for a procedure to add new fields, delete fields, or change the properties of existing fields. For example you may want to temporarily add a field to perform a calculation, then remove the field when the procedure is finished.

There are two techniques for modifying field structure in a procedure. The procedure can modify the structure directly using special statements, or it can open and modify the design sheet.

Getting Information About Field Structure

Before you actually modify the field structure you might want to know something about it. There are several functions that a procedure can use to learn about the structure of a database.

The `dbinfo(` function can gather a variety of information about any open database (see “DBINFO(” on page 5150). This function has two parameters:

```
dbinfo(option,database)
```

The second parameter is the name of the database that you want information about. If the parameter is empty (`""`) the current database is assumed. This database must be currently open. If the database is not open the `dbinfo(` function will not return any information.

The first parameter specifies the type of data you are requesting. Information types you may request include fields, forms, procedures, crosstabs, flash art, and folder. When the option is `"fields"` the function will return a carriage return separated list of the fields in the database (see “Text Arrays” on page 1236). For example, the procedure below will display the number of fields in the current database:

```
message "The database "+info("databasename")+" contains "+
                str(arraysize(dbinfo("fields",""),¶))+" fields."
```

The `datatype(` function returns the data type of a field or variable—text, numeric, date, etc. (see “DATATYPE(” on page 5142). This function has one parameter: the name of the field or variable in question. Depending on the data type this function will return one of the 10 values listed below.

Text	Integer
Choice	Fixed 1 Digit (#.#)
Picture	Fixed 2 Digits (#.##)
Date	Fixed 3 Digits (#.###)
Floating Point	Fixed 4 Digits (#.####)

The procedure listed below uses the `datatype(` function and the `dbinfo(` function to build a list of all the numeric fields in the current database.

```
local X,XField,XType,AllFields,NumericFields
X=1 NumericFields=""
AllFields=dbinfo("fields","")
loop
   XField=array(AllFields,X,¶)
   stoploopif XField=""
   XType=datatype((XField))
   if XType beginswith "F" or XType beginswith "I"
      NumericFields=sandwich("",NumericFields,",")+XField
   endif
   X=X+1
while forever
message "Numeric Fields: "+NumericFields
```

Notice that XField is surrounded by an extra pair of parentheses when it is used in the `datatype(` function. Without this extra pair the `datatype(` function would return the type of the XField variable itself, instead of the field whose name is contained inside of XField.

The `info("fieldname")` function returns the name of the currently active field (see "INFO("FIELD-NAME")" on page 5375). You can use this function to save the current field name in a variable, go somewhere else, then return to the original spot.

Modifying Field Structure Directly

There are five statements that allow a procedure to modify the structure of a database directly: `addfield`, `insertfield`, `deletefield`, `fieldname`, and `fieldtype`.

The `addfield` statement adds one field to the end of the database (the extreme right edge of the data sheet). This statement has one parameter—the name of the new field:

```
addfield fieldname
```

The fieldname can be defined with any formula. To simply define a fixed field name, enclose that name in quotes like this:

```
addfield "Tax Rate"
```

The new field is always a text field. You can change it to a different data type with the `fieldtype` statement (see below). To make this new tax rate field a numeric two digit field the procedure would be modified like this.

```
addfield "Tax Rate"
fieldtype "Fixed 2 Digits (#.##)"
```

Panorama also makes this new field the active field, so the procedure can jump right in with `formulafill` or some other statement that fills the new field with data.

The `insertfield` statement is exactly the same a the `addfield` statement except that the new field is inserted in front of the current field instead of being added to the end of the data sheet (see "INSERTFIELD" on page 5456).

Both the `addfield` statement and the `insertfield` statement can be programmed to display a dialog allowing the user to set up the field name, type, etc. To use this option put the word dialog (no quotes) after the statement, like this:

```
insertfield dialog
```

To change the data type of the currently active field use the `fieldtype` statement (see "FIELDTYPE" on page 5222). This statement has one parameter, the new data type:

```
fieldtype type
```

The type parameter is actually a text item that names the parameter. Legal field types are shown in this table.

Text	Integer
Choice	Fixed 1 Digit (#.#)
Picture	Fixed 2 Digits (#.##)
Date	Fixed 3 Digits (#.###)
Floating Point	Fixed 4 Digits (#.####)

If the current field has any data in it, Panorama will attempt to convert the data to the new data type. If some of the data can't be represented in the new data type, that data will be thrown away; so be careful! For example if a text field is converted to date or number, data values like John Smith or San Francisco in that field will be tossed. Don't change the field type unless you are sure the data currently in the field can be converted, or unless you don't care!

The procedure below shows how these statements and functions can be used together. This procedure makes an exact copy of the current field. First it copies the current field name and type into the local variables theField and theType (the `datatype(` function is described in the previous section). Then it attempts to move one field to the right. If it can't (because the current field is the last field of the database) it adds a new field, otherwise it inserts a new field in the middle of the database. Finally, it sets the new field to the same data type as the original field and copies the data from the original field into the new field.

```
local theField,theType
theField=info("fieldname")
theType=datatype(info("fieldname"))
right
if stopped /* could also use if info(“stopped”) */
   addfield "Copy of "+theField
else
   insertfield "Copy of "+theField
endif
fieldtype theType
formulafill grabdata("",theField)
```

To change the name of the current field use the `fieldname` statement (see "FIELDNAME" on page 5220). You can either specify the new name for the field using a formula, or use the word `dialog` to allow the user to enter the name in a dialog (they will also be able to modify other field properties.) (Note: Panorama will not prevent you from creating two or more fields with the same name. However, you should avoid this if possible. You can use the `dbinfo(` function to get a current list of the field names; see the previous section.)

To delete the current field, use the `deletefield` statement. Panorama won't display any warning—it will simply delete the field and all the data in it. Be careful because once you delete a field it's gone…you can't get it back. (Exception: If you have saved the database you might be able to get the field back with the **Revert To Saved** command.)

Working With the Design Sheet

For the ultimate control the procedure can open the design sheet and change it just like any other database. There are two statements a procedure can use to open the design sheet: `opendesignsheet` and `godesignsheet`. The `opendesignsheet` statement opens the design sheet in a new window (see "Opening a Window" on page 1572 for more information on opening windows). The `godesignsheet` statement opens the design sheet in the current window.

Once the design sheet is open, the procedure can locate any field it wants using the `find` statement (see "Finding Information" on page 1677). Once the correct line is selected the procedure can change elements with assignment statements (`Default="Acme"`, etc.). When the changes are complete, the procedure must use the `newgeneration` statement to actually change the structure of the database (see "NEWGENERATION" on page 5537).

The example below opens the design sheet and changes the output pattern for the Price field.

```
opendesignsheet
find «Field Name»="Price"
if info("found")
   «Output Pattern»="$#,.##"
   newgeneration
endif
closewindow
```

If you don't want the user to be able to see the shenanigans with the design sheet, use the `setwindow` or `setwindowrectangle` statements to make the window open outside the visible screen area (see "Specifying the New Window Location" on page 1573).

Updating Database Structure From Another Database

Panorama includes a mechanism that lets you copy the structure of a database from another database while retaining the original data. Let's say that you have created a database and distributed it to many users far and wide...perhaps you are even selling the database. Your many users are each filling their databases with their own data. In the meantime, you are creating a new version. This new version of the database may have new fields, new forms, new procedures, and there are probably changes to existing forms / procedures / fields as well. Once you have finished your update you need a way to distribute the update and let each user update his or her copy of the database so that it has the new structure but retains the old data. The `changename`, `detachname` and `hijack` statements make this possible. The example procedure below shows how to do it. This procedure assumes the old version of the database is currently open. The procedure allows the user to locate the update file, then updates the structure.

```
local oldFile,newFolder,newFile,newFType
/* let user locate the update file */
openfiledialog newFolder,newFile,newFType,"KASXZEPD"
if newFile="" stop endif /* user pressed cancel */
/* save name of original database*/
oldFile=info("databasename")
/* change name of original database IN MEMORY ONLY */
changename oldFile+".old"
/* open the file with the new structure */
openfile folderpath(newFolder)+newFile
/* suck the data from the old file into the new structure */
openfile "&"+oldFile+".old"
/* name of new database=name of old database (IN MEMORY ONLY) */
detachname oldFile
/* now connect to the original databases file on disk */
/* it's a "filejacking"! */
hijack oldFile+".old"
/* save the new, update file */
save
/* finally close the old database - we're done with it */
window oldFile+".old:SECRET"
closefile /* this file is really gone now */
```

If you look closely at this example, you will see that it doesn't really update the structure of the original database. Instead, it loads the data from the old database into the new database using Panorama's standard "append with matching names" feature (see "Replacing the Data in a Database" on page 1534). Once this is done the old database is "detached" from its disk file. The new file then takes over or "hijacks" the detached disk file. As part of this "hijack" process Panorama also copies the auto-increment value, so if the database uses auto-numbering the numbers will continue to be generated in sequence.

Transferring Permanent Variables

If the original database has permanent variables that you want to keep, insert the following statements just before the `detachname` statement in the procedure above. This procedure assumes that the new updated database has at least the same permanent variables as the original database, and it copies the values from the old database to the new.

```
local oldPermanentVariables,opv,pv
/* build a list of the permanent variables */
oldPermanentVariables=dbinfo("permanent",oldFile+".old")
opv=1
loop
   /* get name of permanent variable */
   pv=array(oldPermanentVariables,opv,¶)
   stoploopif pv=""
   /* transfer value from old to new */
   set pv,grabfilevariable(oldFile+".old",pv)
   opv=opv+1
while forever
```

For more information about the `grabfilevariable(` function, see “Accessing “Dormant” Variables” on page 1373.

The procedure above will not work if the old database is not in author mode, since the `dbinfo(` function will not be able to build a list of permanent variables. In that case you must rely on your knowledge of the original database and hard code the permanent variable names, like this:

```
pAreaCode=grabfilevariable(oldFile+".old","pAreaCode")
pDialingPrefix=grabfilevariable(oldFile+".old","pDialingPrefix")
pCallingCard=grabfilevariable(oldFile+".old","pCallingCard")
```

This procedure transfers three permanent variables from the old database to the new: pAreaCode, pDialingPrefix and pCallingCard.

Verifying Database Identity

The procedure listed above for updating the database relies on the user to pick the correct update database. If they pick the wrong database, there will be a big problem. You can use permanent variables to create a database identity system that will permanently identify a database, even if it has been renamed. We recommend creating three permanent variables with the names dbVendor, dbName and dbVersion. Here is an example showing how these variables can be created in the **.Initialize** procedure (see “.Initialize” on page 1511).

```
permanent dbVendor,dbName,dbVersion
dbVendor="ProVUE Development"
dbName="Power Team Phone Book"
dbVersion="2.0"
```

Once these variables have been created they can be used to verify the identity of a database. In the database update routine you can add verification code in between the two `openfile` statements (see “Updating Database Structure From Another Database” on page 1635). This verification code will stop the update if the user selected the wrong database.

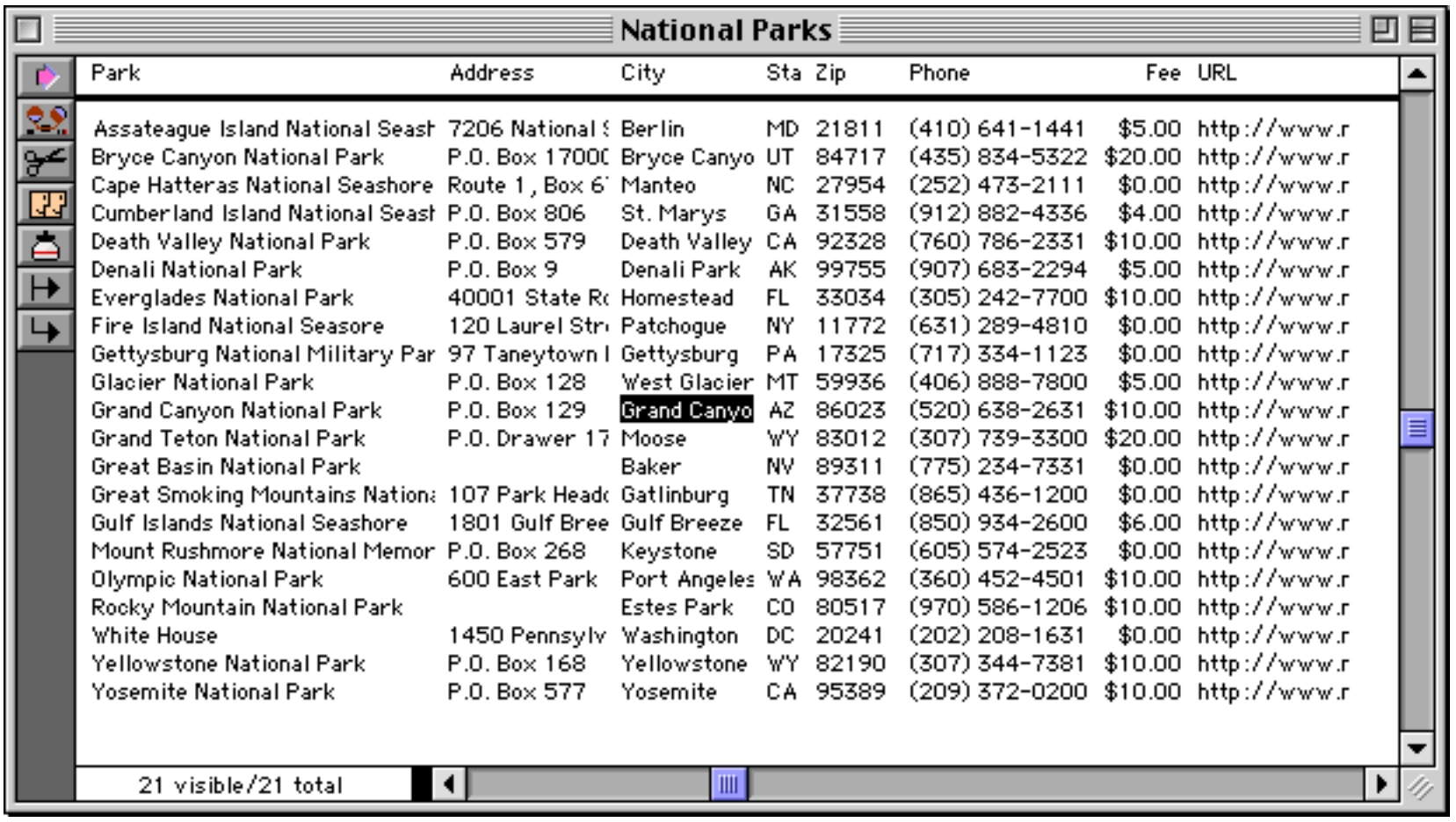
```
if dbVendor≠grabfilevariable(oldFile+".old",dbVendor) or
           dbName≠dbVendor≠grabfilevariable(oldFile+".old",dbName)
   message "Please pick another database. "+
           "The file you picked is not an update for "+oldFile+"."
   closefile /* close the bogus update file */
   changename oldFile /* and restore the original name */
endif
```

You could make this procedure even more robust by adding a check to make sure that the version number of the update file is newer than the version number of the old file.

Database Navigation and Editing

When you are manually working with a database you can use your eyes to see what you are clicking on and modifying. A procedure doesn't have eyes to see with, but it can still navigate and modify the database. Since the procedure can't see what it is doing you have to give it exact instructions to get the job done correctly. Imagine giving directions to a blindfolded person (go 23 paces, turn left, go 14 paces, turn right, etc.) Using a procedure to navigate and edit the database requires the same type of precise instructions.

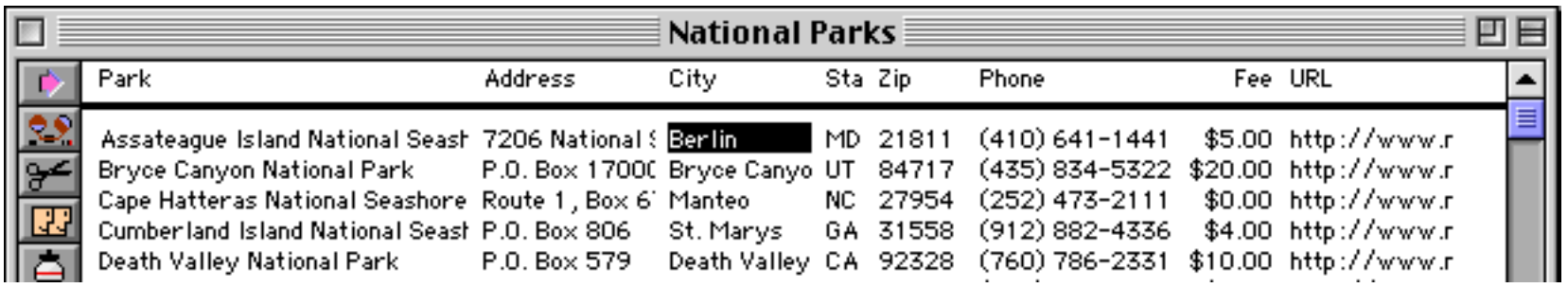
To illustrate how a procedure can navigate and move around the database we'll use this database of national parks.



As shown above we'll start out with the database on the record for Grand Canyon National Park. The current field is the City field.

Moving Up and Down in the Database

The basic statements for moving the currently active record are `firstrecord`, `lastrecord`, `uprecord` and `downrecord`. The `firstrecord` statement makes the very first visible record in the database the currently active record (the record at the top of the data sheet).



The `lastrecord` statement makes the very last visible record in the database the currently active record (the record at the bottom of the data sheet).



The `uprecord` and `downrecord` statements move the currently active record either one record up (towards the top of the data sheet) or one record down (towards the bottom of the data sheet).

To find out if the currently active record is the first or last visible record in the database, use the `info("bof")` and `info("eof")` functions (bof stands for beginning of file and eof stands for end of file).

Here's an example that uses the statements and functions described in this section to count the number of parks with no access fee.

```
local freeparks
freeparks=0
firstrecord
loop
   if Fee=0
      freeparks=freeparks+1
   endif
   stoploopif info("eof")
   downrecord
while forever
```

Although this procedure will work, it will also be unnecessarily slow. Avoid scanning through the database whenever possible. Here's another way to write this same procedure that will be much, much faster. For a small database with a couple of dozen records like our example the speed difference isn't too important, but for a large database with thousands of records we're talking about the difference between seconds vs. minutes.)

```
local freeparks
freeparks=0
formulasum freeparks,?(Fee=0,1,0)
```

Another way to reposition the currently active record is to search for something using the `find` statement (see “Finding Information” on page 1677). The `find` statement has one parameter, a formula. Starting from the top of the selected records, Panorama scans down the database until it finds a record that makes this formula true. For example, this procedure will scan down the database until it finds a record where the Park field contains Everglades.

```
find Park contains "Everglades"
```

Notice that the active field stays the same (City) even though the formula searches the Park field.



After the `find` statement you can check to see if Panorama actually found anything with the `info("found")` function (see “INFO("FOUND")” on page 5381).

To find the next match use the `next` statement (see “Finding Information” on page 1677). This is just like the `find` statement except there is no formula…it re-uses the formula supplied with the `find` statement. You can continue to use the `next` statement over and over again until the `info("found")` function tells you there are no more matches.

Here’s an example that locates all parks with no access fee and deletes them from the database:

```
find Fee=0
loop
   stoploopif (not info("found"))
   deleterecord
   next
while forever
```

Once again, this procedure will work but will be slow. Here’s a faster solution:

```
select Fee<>0
removeunselected
```

Why do I keep showing you these alternate examples? If you are a C or a Pascal programmer you are probably used to solving many problems with loops. In Panorama, a loop is often not the best solution because it is too slow. It may take some research, but you can usually find a Panorama statement that will do the same job much faster.

Moving Left and Right

The basic statements for moving the currently active field are `field`, `left` and `right`. The `field` statement moves directly to the specified field (see “FIELD” on page 5217). For example

```
field Park
```

will make the Park field the current field.



If a field name has spaces or other unusual punctuation you must surround it with quotes (see “Constants” on page 1195). Be sure to use quotes and not « » (chevrons).

```
field "Phone Number"
```



You can also use a formula to calculate the field name. If you do so you must surround the formula with (and) parentheses. For example this procedure will move to the last column in the database — any database.

```
field (array(dbinfo("fields",""),arraysize(dbinfo("fields",""),¶),¶))
```

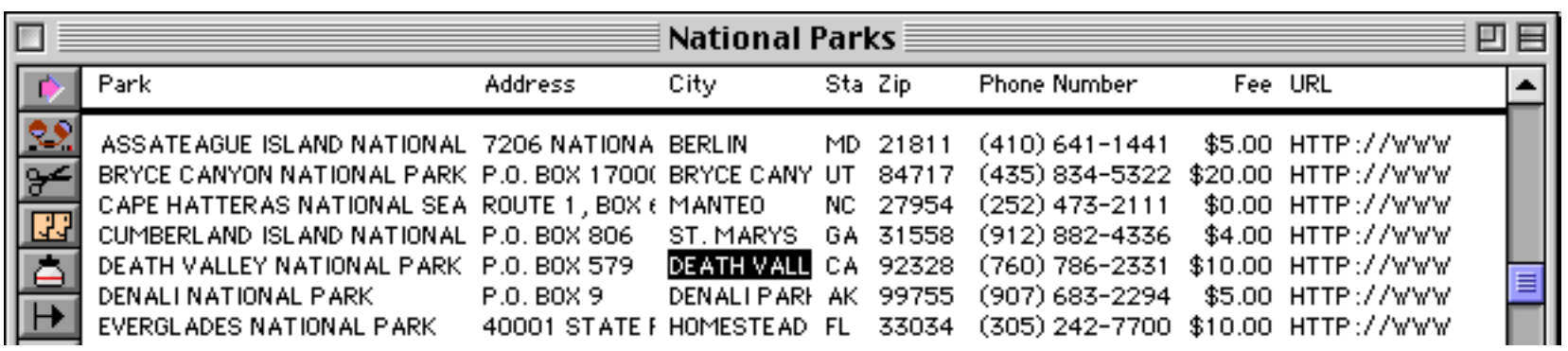
The formula uses the `dbinfo(` function (see “Getting Information About Field Structure” on page 1632) to calculate the name of the last field (in this case URL) and then the field statement jumps to that field.



To move left one field (column in the data sheet) use the `left` statement (see “LEFT” on page 5466). To move right one field (column in the data sheet) use the `right` statement (see “RIGHT” on page 5679). You can find out if this operation succeeded by using `if stopped`. This will be true if the procedure tried to move to the left of the first column or to the right of the last column. Here is a procedure that scans the entire database and converts every text field to upper case.

```
field (array(dbinfo("fields",""),1,¶))    /* move to first field */
loop
   if datatype(info("fieldname"))="Text"   /* is this a text field? */
      formulafill upper(«»)                /* if yes, convert to upper case */
                                           /* Note: «» is shorthand for current field */
   endif
   right                                   /* move to next field */
until stopped    /* stop if we just tried to move past last field */
                /* could also use until info(“stopped”) */
```

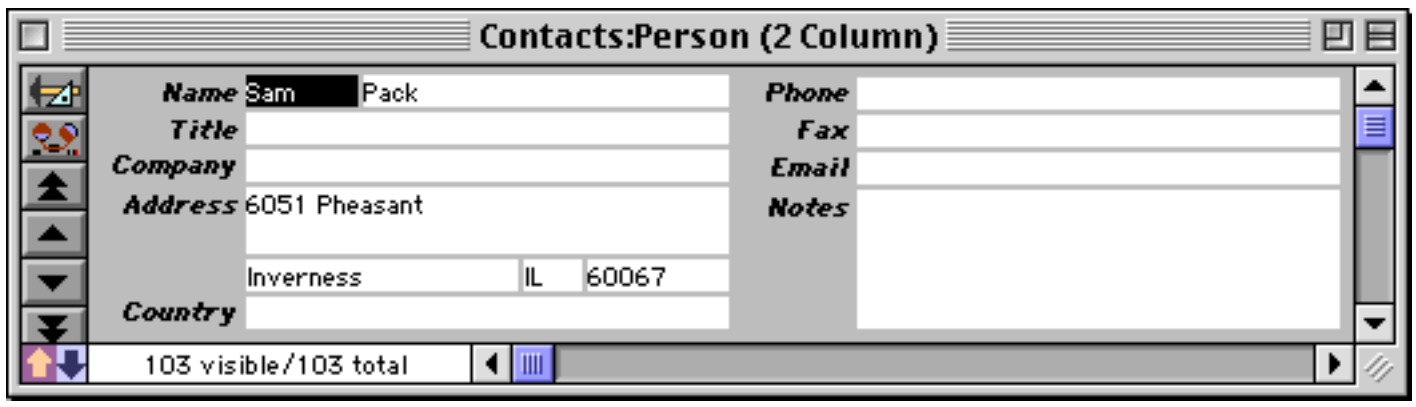
Here’s the result of running this procedure on our National Parks sample database.



Notice that this procedure doesn’t reference any specific field names in the National Parks database. This procedure will actually work on any database.

Moving “Left” and “Right” on a Form

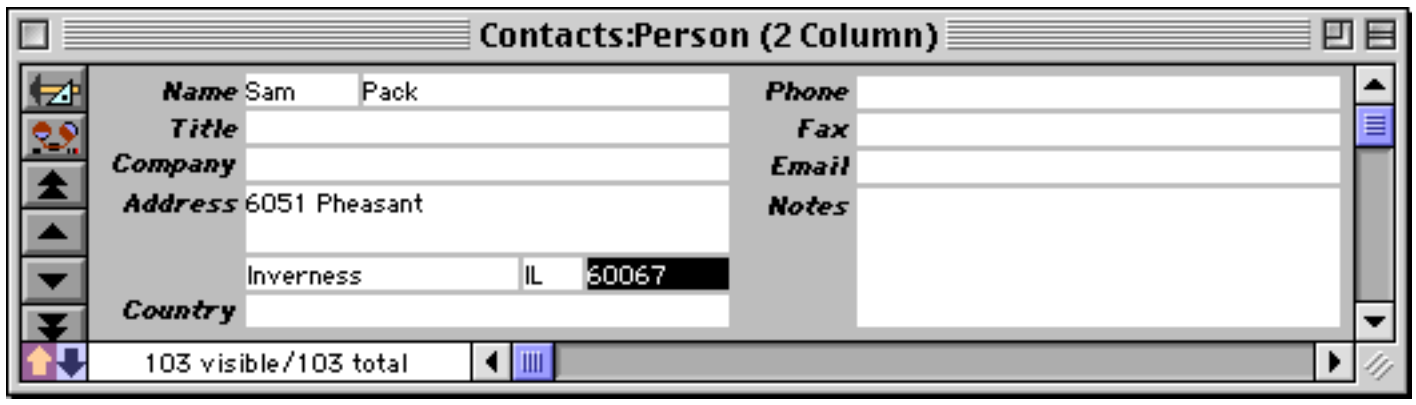
The examples in the previous section all showed the data sheet, but a procedure can also move to a specific field within a record on a form. To illustrate this we’ll use this form from a Contacts database.



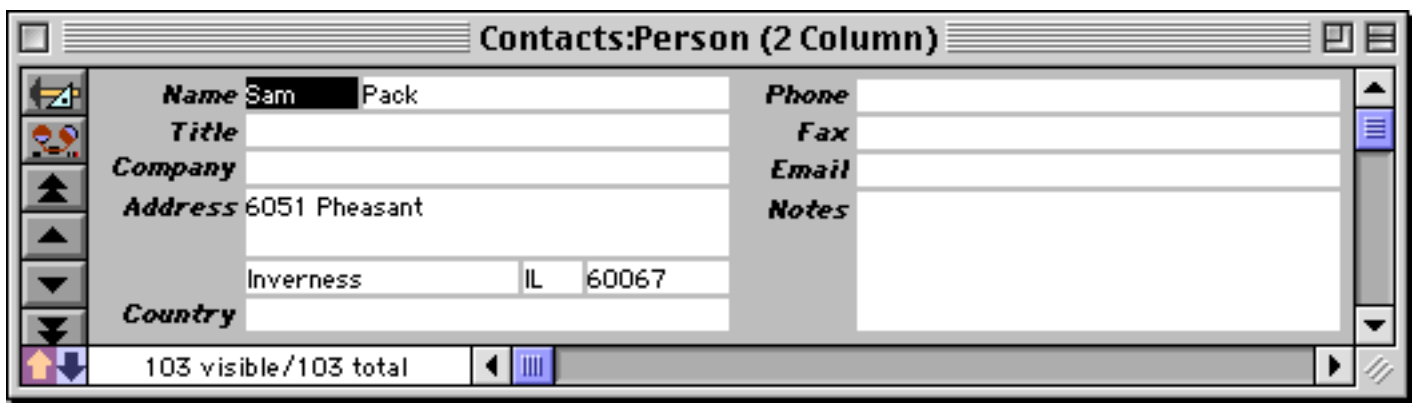
Just as when the data sheet is open a procedure can move to a specific field with the `field` statement.

```
field Zip
```

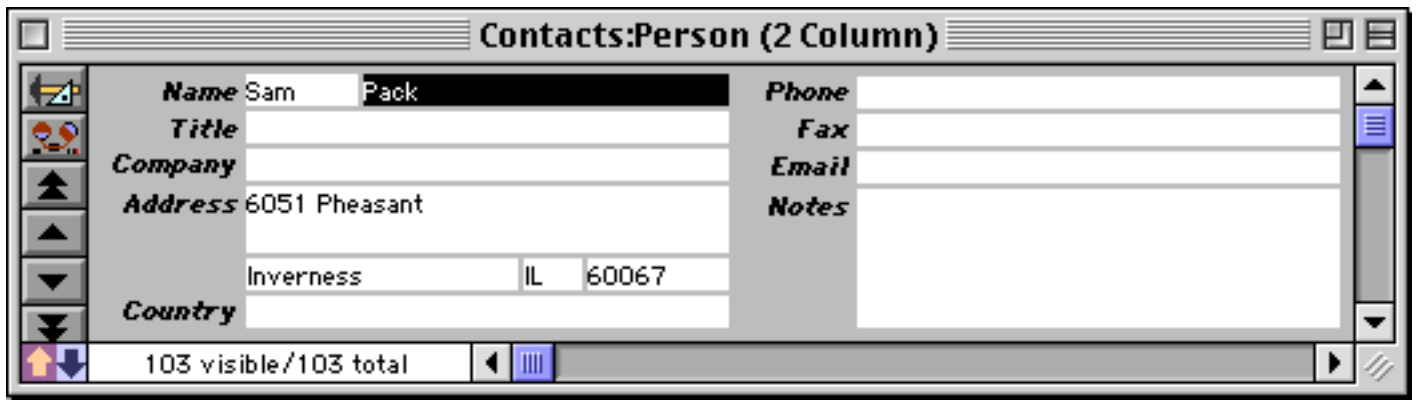
Panorama will jump to the specified field.



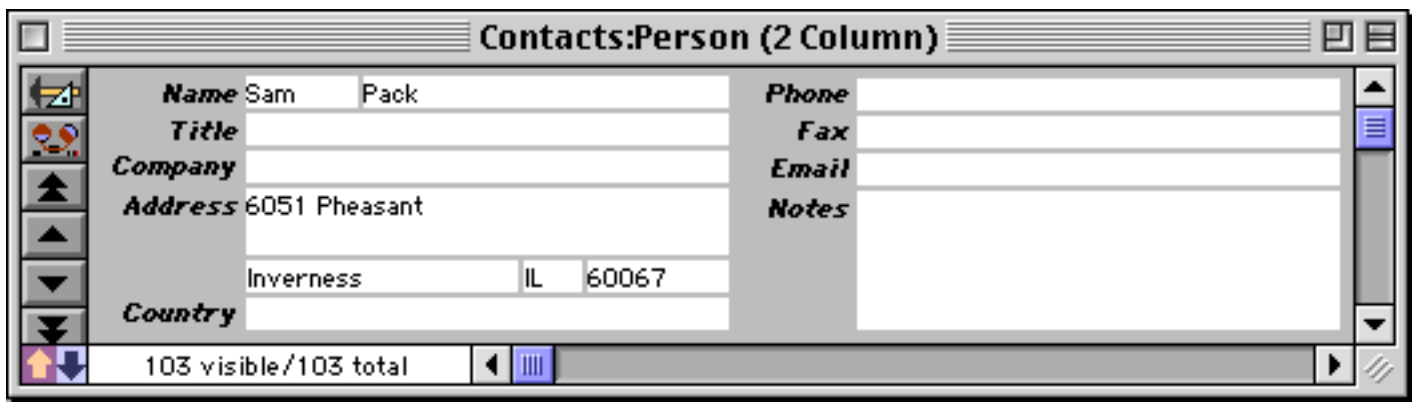
The `left` and `right` statements don't move left and right on the form, but move to the next column based on the order of the fields on the data sheet. For example, suppose you start on the First field.



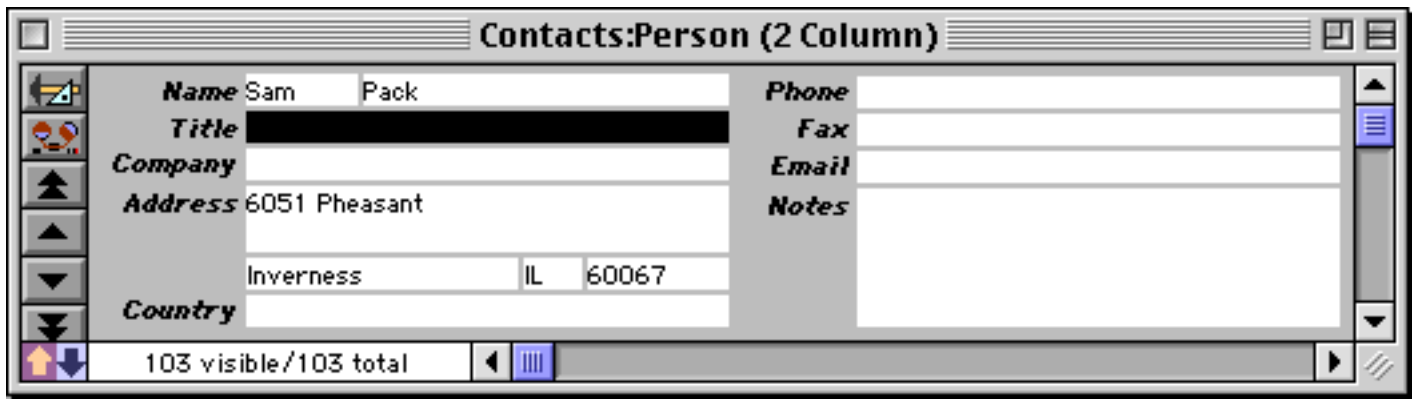
The `right` statement will cause Panorama to jump to the next field to the right in the data sheet. In this case that field (Last) is also to the right on the form.



The next field to the right in the data sheet, Credit Card, does not exist on this form, so at this point the current field selection is invisible. The Credit Card field is the current field, however, and would be modified if the procedure used a statement like `formulafill` at this point.



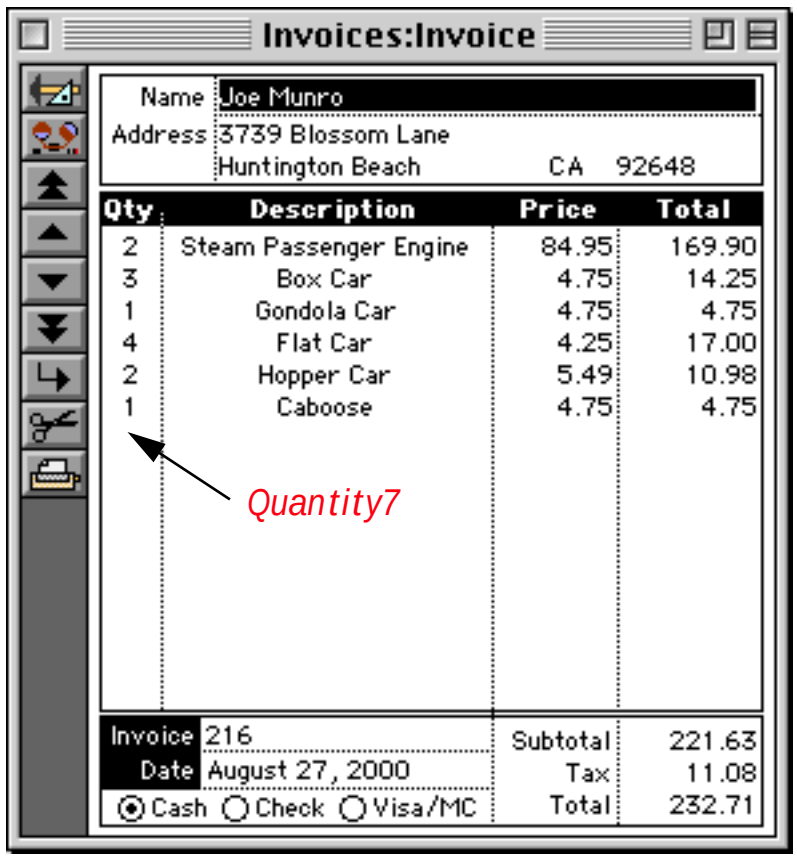
The next field to the right in the data sheet, Title, does have a data cell on this form.



The procedure can continue moving to the “right” until it gets to the last column in the data sheet. The `left` statement moves in the opposite direction.

Moving to an Empty Line Item Field

Line items are used for repeating items within a record (see “Repeating Fields (Line Items)” on page 262). When creating a new line item line in a procedure you will want to move to the first empty line item field. To illustrate this, consider this simple invoice form.



To add a new line item to this invoice the procedure must move to the Quantity7 field. One way to do this would be with a loop.

```
local n
n=1
loop
   field ("Quantity"+str(n))
   stoploopif «»=""
   n=n+1
while forever
```

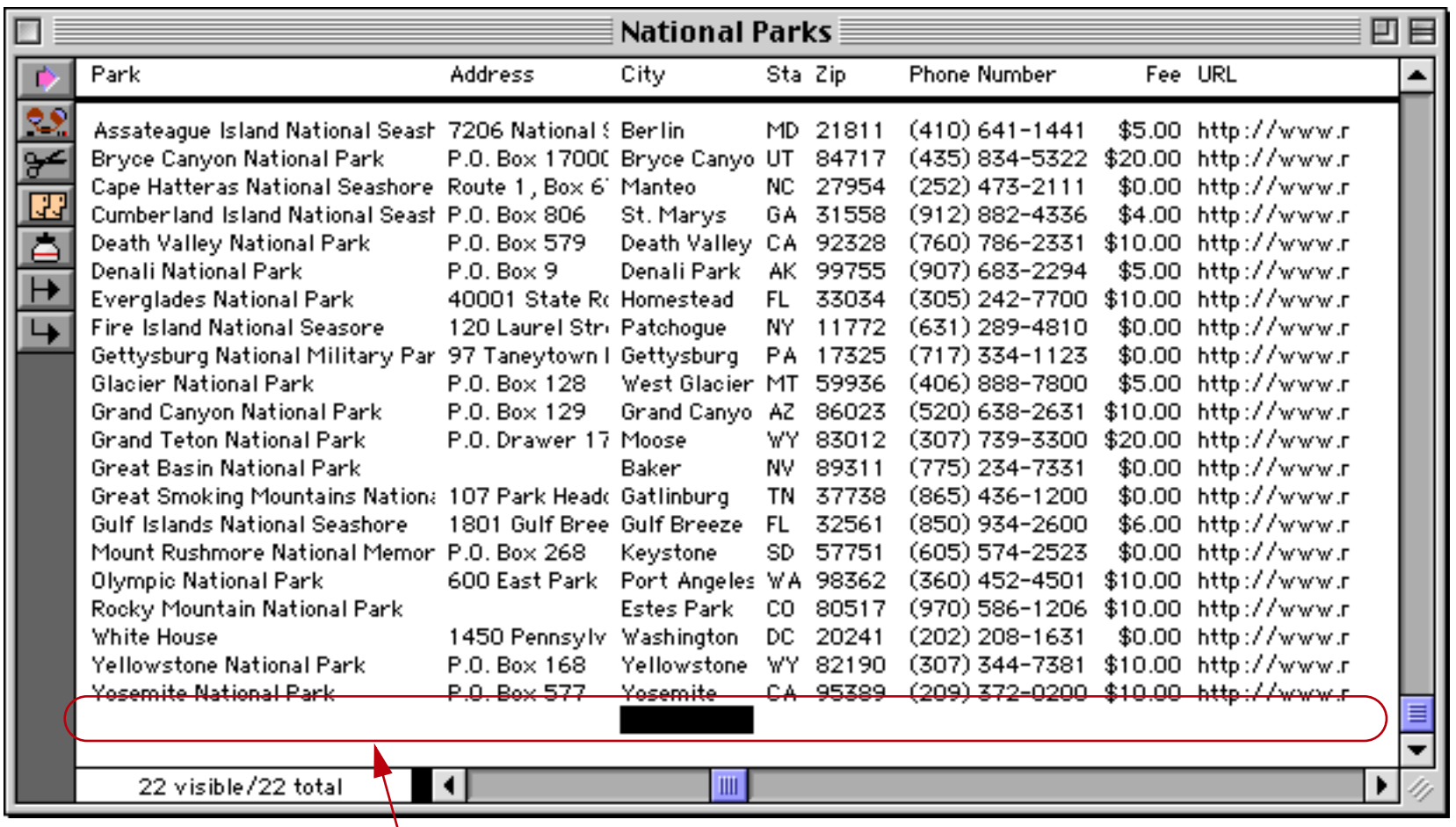
Since this is such a common operation when working with line item fields Panorama has a built in statement to do this job. The statement is called `emptyfield` (see "EMPTYFIELD" on page 5191). The `emptyfield` statement has one parameter, the name of the line item field to jump to. This field name must be surrounded with quotes and must be followed by the Ω character (see "Special Characters" on page 1202).

```
emptyfield "QuantityΩ"
```

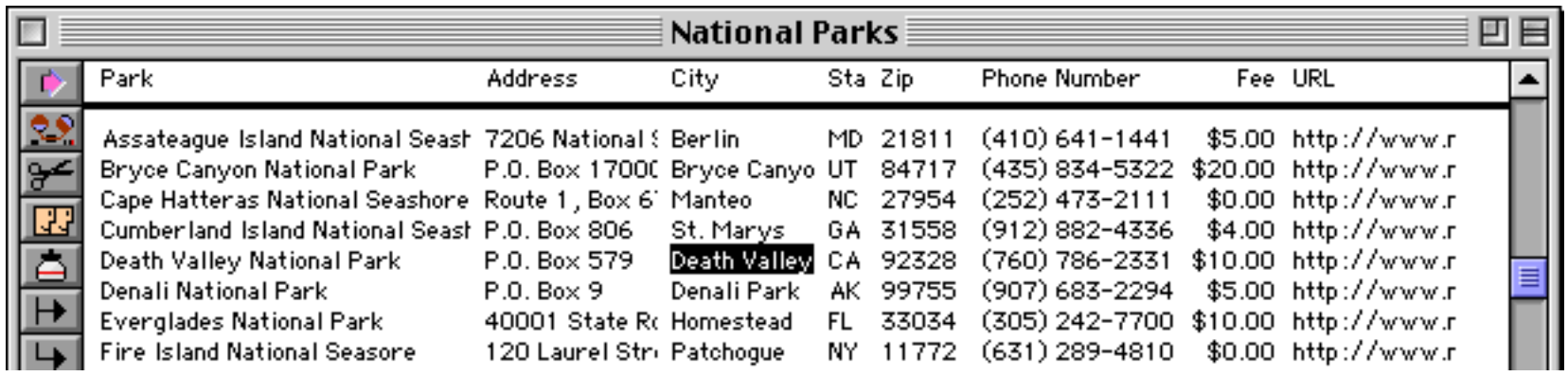
If there aren't any empty line item fields this statement simply leaves the current field wherever it was.

Adding and Deleting Records

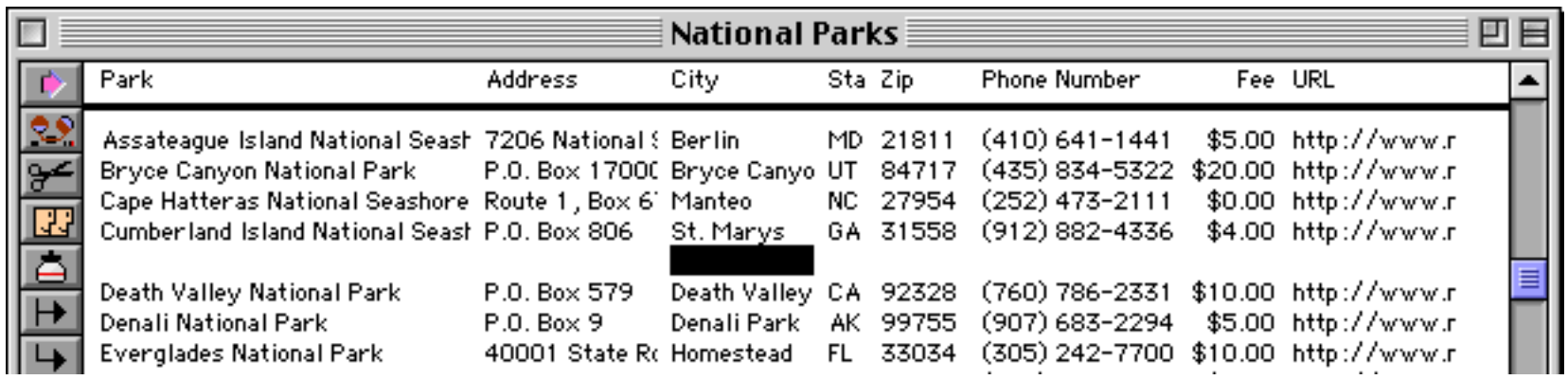
To add a new record at the end of the database use the `addrecord` statement (see "ADDRECORD" on page 5018).



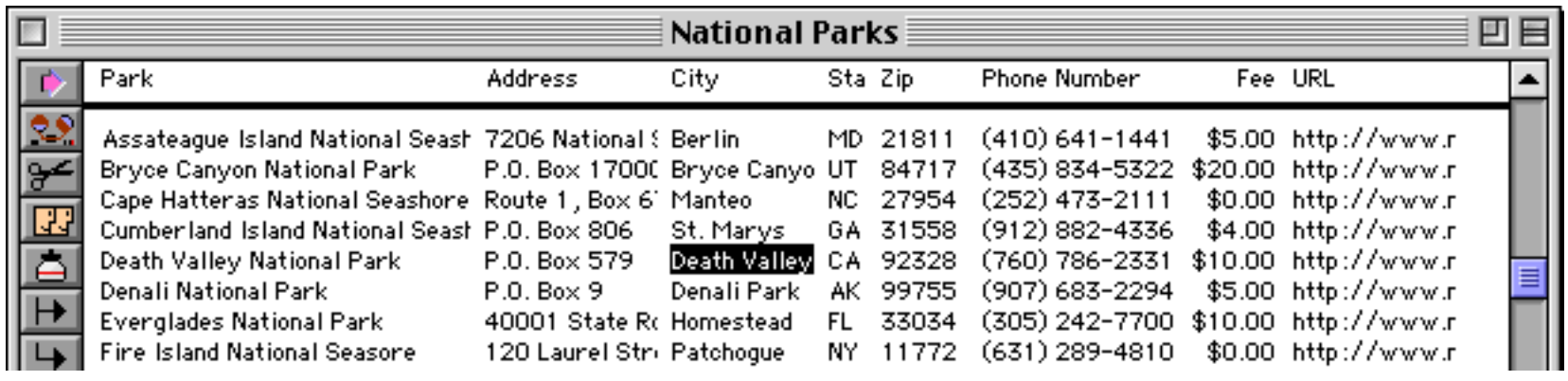
To insert a new record just above the current record use the `insertrecord` statement (see "INSERTRECORD" on page 5457). For example, suppose you start with the database on the Death Valley National Park record, like this.



The `insertrecord` statement inserts a record just above Death Valley.



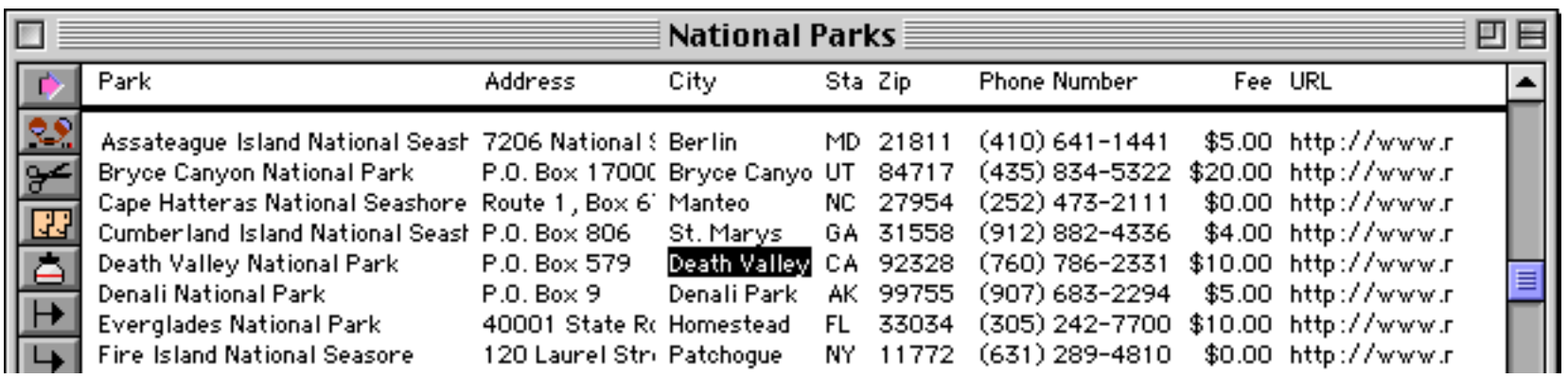
To insert a new record just below the current record use the `insertbelow` statement (see "INSERTBELOW" on page 5455). For example, suppose you start with the database on the Death Valley National Park record, just like the previous example.



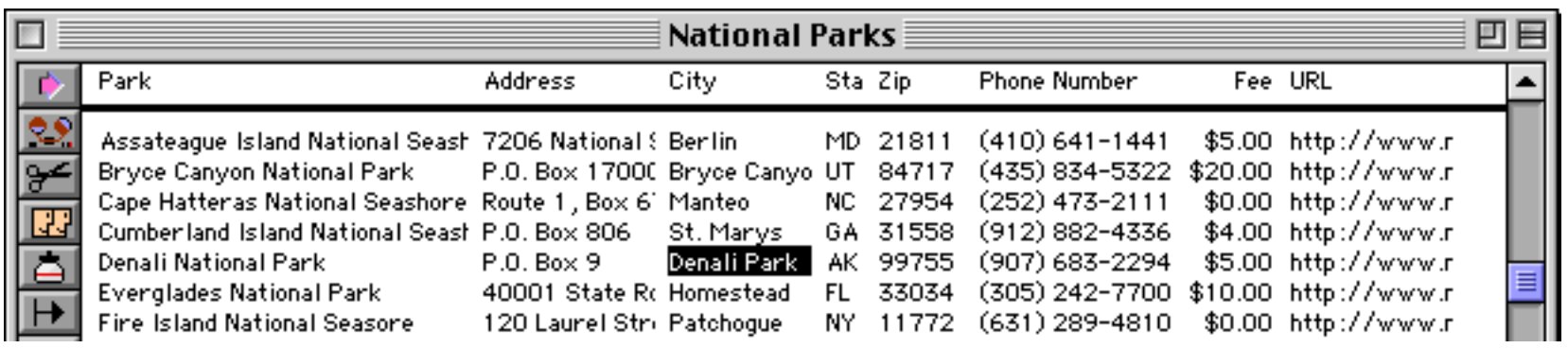
The `insertbelow` statement inserts a record just below Death Valley. Notice that it also moves the current field to the first field in the database.



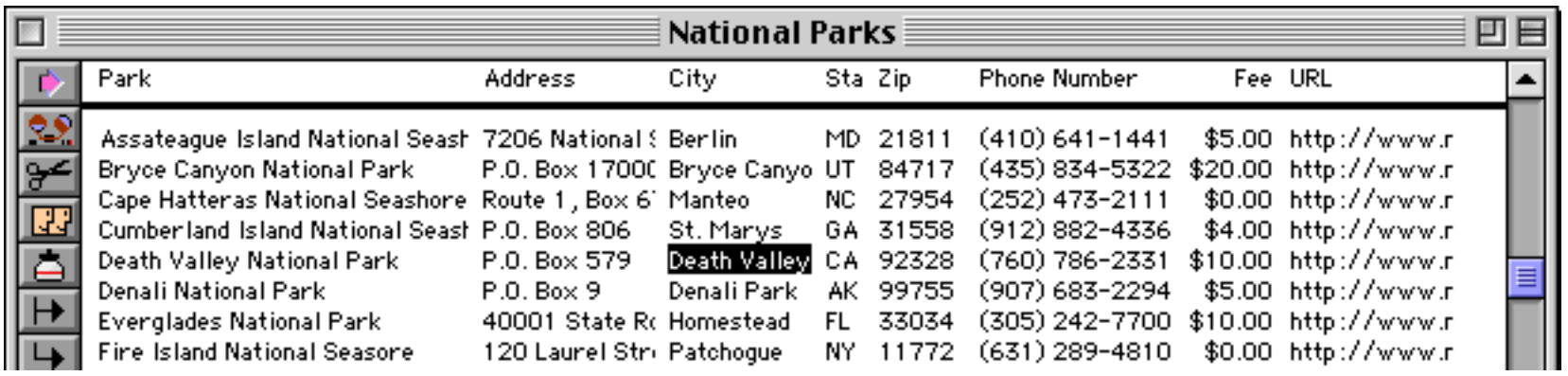
To delete the current record use the `deleterecord` statement (see "DELETERECORD" on page 5161). Once again we'll start with the database on the Death Valley National Park record.



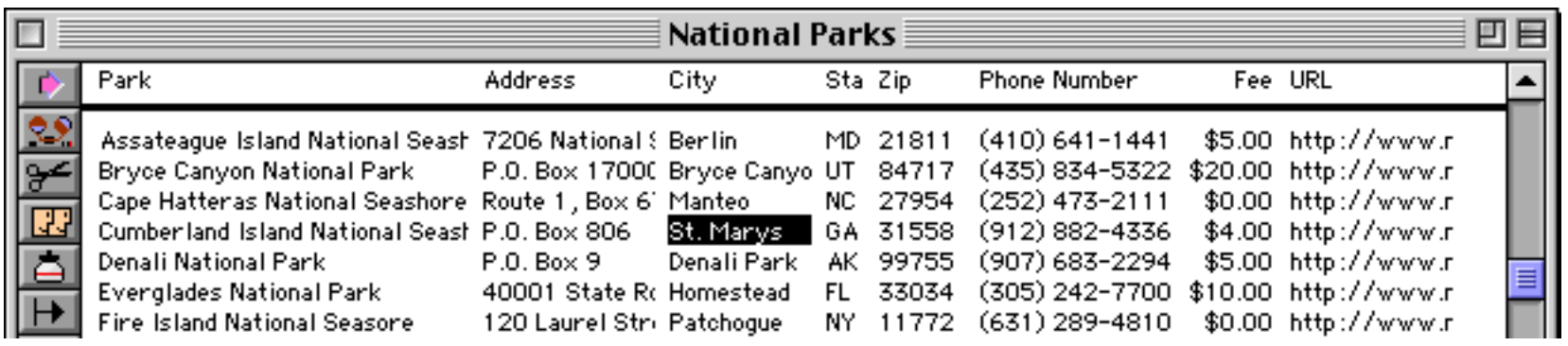
The `deleterecord` statement deletes Death Valley and makes the record below Death Valley (Denali Park) the current record.



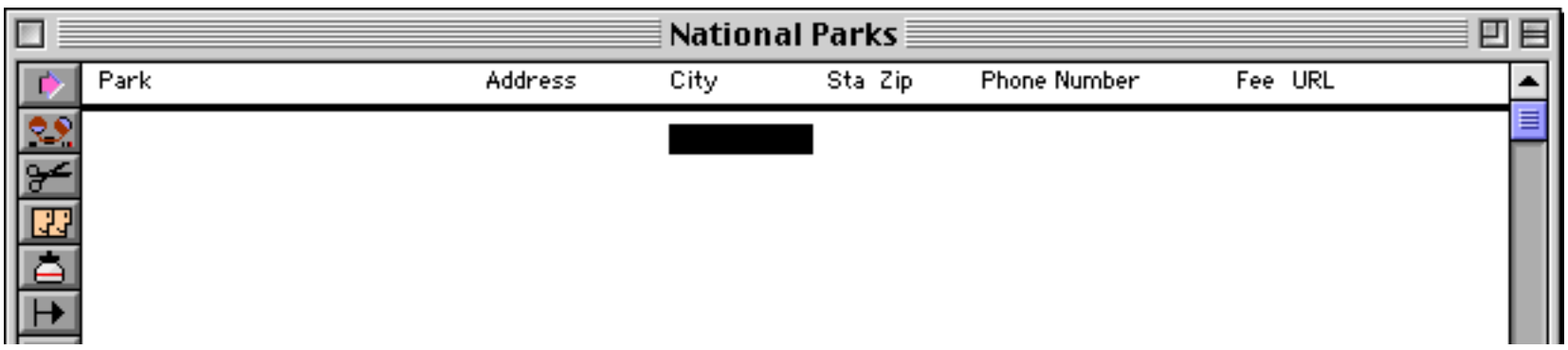
Another way to To delete the current record use the `deleteabove` statement (see "DELETEABOVE" on page 5156). Once again we'll start with the database on the Death Valley National Park record.



The `deleteabove` statement deletes Death Valley and makes the record that was above Death Valley (Cumberland Island national Seashore) the current record.



The `deleteall` statement deletes all the data in the entire database, leaving just one blank record (see “DELETEALL” on page 5157).



Needless to say you need to be very careful with this statement!

Modifying the Database One Cell at a Time

To modify an individual data cell in the current record (see “Moving Up and Down in the Database” on page 1637) you use an assignment statement (see “Assignment Statements” on page 1368). Unlike every other statement, an assignment statement has no specific keyword that identifies the statement. Assignment statements always have the format shown below:

```
<data storage location> = <formula>
```

The first part of the assignment statement is the data storage location. This is the final destination for the data that is being moved. In fact, sometimes the data storage location is simply called the destination of the assignment. The data storage location may be a variable, a field in the currently active record, or the clipboard.

The next part of the assignment statement is the equals symbol. This identifies this statement as an assignment statement.

After the equals symbol is the formula. The formula may simply take a variable or field and pass it along, or it may process, calculate or filter the data before it passes it along to be stored in the data storage location. Here‘s a simple assignment statement that takes the contents of B and moves it into A. After this statement is finished both A and B will contain the same value.

```
A=B
```

More complicated assignment statements may combine multiple fields or variables, and they may process the data in some way. An assignment statement may also take a constant value and store it. Here are some examples:

```
A=B*C

Name=upper(myName)

City="San Francisco"
```

In each case, the process is the same. First Panorama calculates the formula to produce a data value. Then it stores the data value in a data storage location.

Accessing and Modifying the Current Cell

To access or modify the current cell use **«»** (see “Special Characters” on page 1202). For example this statement sets the current cell to Bingo.

```
«»="Bingo"
```

This procedure stores the contents of the current cell in the variable OriginalData.

```
OriginalData=«»
```

To find out which field is the current cell use the `info("fieldname")` function.

Accessing and Modifying the Clipboard

In addition to variables and fields, the operating system itself provides one spot for stashing data…the **clipboard**. This is where data goes when you use the **Copy** or **Cut** commands in the Edit menu. To grab any text that is in the clipboard, use the `clipboard()` function. To put data on the clipboard, use an assignment with `clipboard` on the left hand side of the equals sign.

The example below takes an address, formats it and copies it onto the clipboard.

```
clipboard=Name+¶+Address+¶+sandwich("",City,", ")+State+" "+Zip
```

Here's an example that grabs a name from the clipboard and selects all the records containing that name:

```
local findThis
findThis=clipboard()
select Name contains findThis
```

This example copies the contents of the clipboard into a variable before using it in the `select` statement (see "Selecting Information" on page 1682). This is not absolutely necessary—in fact this procedure could have been written in a single line like this:

```
select Name contains clipboard()
```

However, the original procedure will be much faster. Because of the overhead involved in querying the operating system, accessing the clipboard is much slower than accessing a variable. If you're only going to be accessing the clipboard a few times, by all means use it directly. But if you are going to access the clipboard over and over again (as the `select` statement does) it's much better to copy the value into a variable first.

Triggering Automatic Calculations

A database can be set up so that when a field is modified by the user, one or more formulas is automatically calculated (see "Automatic Calculations" on page 340). When an assignment statement modifies a field, however, these formulas are not automatically calculated. This is to give the procedure programmer the ultimate control over all calculations that occur during the procedure.

If you as the programmer would like the automatic calculations to be performed during an assignment, add an extra equal symbol to the assignment. The two equal symbols must be adjacent with no spaces between them, like this:

```
PriceΩ==19.95
```

In this example, storing the value 19.95 will most likely trigger several additional calculations to compute the total for this line item and the total for the entire invoice.

Triggering Automatic Procedures

A database can be set up so that when a field is modified by the user, a procedure is automatically triggered. This may be a specific procedure for this field (see “Automatically Triggering a Procedure” on page 350), or the generic **.ModifyRecord** procedure (see “.ModifyRecord” on page 1512). These procedures are never triggered automatically when an assignment statement modifies a field, even when the double equal assignment is used (see previous section). If you want a procedure to be triggered after a field is modified, you must call the procedure explicitly with the `call` statement (see “Subroutines” on page 1384). This example modifies several fields, then calls the **.ModifyRecord** procedure:

```
City="San Francisco"
State="CA"
«Area Code»="415"
call .ModifyRecord
```

If the automatically triggered procedure expects that a certain field is active when it is triggered you should make sure that field is active by using the field statement before calling the procedure.

The Set Statement

The `set` statement performs the same job as an assignment statement—moving a data item from one place to another (see “SET” on page 5735). However, unlike the assignment statement, the data storage location is not known in advance. Instead, the data storage location is calculated using a formula.

The set statement has two parameters:

```
set <data storage location formula>,<formula>
```

The first parameter is a formula that calculates the name of the data storage location. Suppose you wanted to store the data in a field named Widget. In an assignment you would simply use this name, but in the `set` statement the name must be calculated, in this case `"Widget"`. (Of course this is a silly example, because if we knew the field name in advance we might as well use a regular assignment statement. We’ll look at a better example in a minute.)

The second parameter is the formula. This formula produces the data that will be stored in the data storage location. It’s exactly the same as the formula used in an assignment statement.

Here’s our example. Suppose we have a database where each record has 10 phone number fields, Phone1, Phone2, Phone3, etc. We want to write a procedure that will automatically store a new phone number in the first empty field. Here’s one way to get this job done using the `set` statement:

```
local newPhone,Counter,tempPhone
Counter=1 newPhone=""
gettext "New phone #:",newPhone
loop
   tempPhone=grabdata("","Phone"+str(Counter))
   if error
      message "No empty phone number slots!"
      stop
   endif
   stoploopif tempPhone=""
   Counter=Counter+1
while forever
set "Phone"+str(Counter),newPhone
```

The procedure starts by initializing the variables, and asking the user to input the new phone number. Then it loops through the phone number fields, starting with Phone1, then Phone2, etc. It checks each field to see if it is empty. If it is, the loop stops and the new phone number is stored with the `set` statement. The first parameter of the `set` statement calculates the field name with the formula `"Phone"+str(Counter)`. For example, if the fourth phone number field was empty, this formula will calculate the field name Phone4.

The FormulaCalc Statement

The `formulacalc` statement is similar to the set statement. It's different from the `set` statement because the data storage location is known in advance, but the formula is not known in advance. Instead, the formula itself is calculated using another formula. The `formulacalc` statement has two parameters:

```
formulacalc <data storage location>,<formula>
```

The first parameter is the data storage location. This should be a variable or field name, just as in the regular assignment statement.

The second parameter is the formula. This formula must itself produce another formula, which is then calculated to produce the data that will be stored in the data storage location. Usually there is a field or variable that contains the formula you want to calculate.

Our example of the `formulacalc` statement lets the user type in a formula, then calculates the formula and displays the result.

```
local xFormula,Answer
xFormula=""
gettext "Enter the formula",xFormula
formulacalc Answer,xFormula
message Answer
```

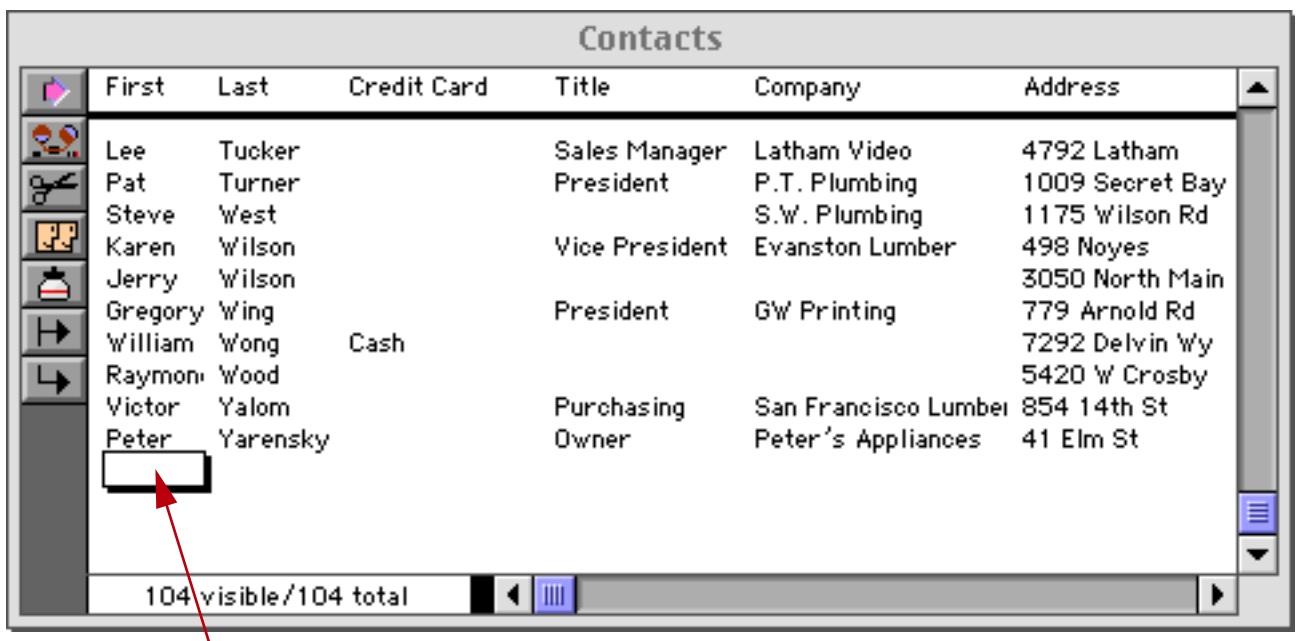
The `formulacalc` statement was primarily designed to make it easy to build a calculator with Panorama. I'm sure some enterprising programmer out there will find some other uses as well.

Opening the Input Box

The data sheet and data cells in a form use a pop-up input box for editing data (see "The Input Box" on page 310). The `editcell` statement automatically opens the input box for the current cell (see "EDITCELL" on page 5184). For example, this procedure adds a new record and automatically opens the input box to get ready for data entry.

```
addrecord
editcell
```

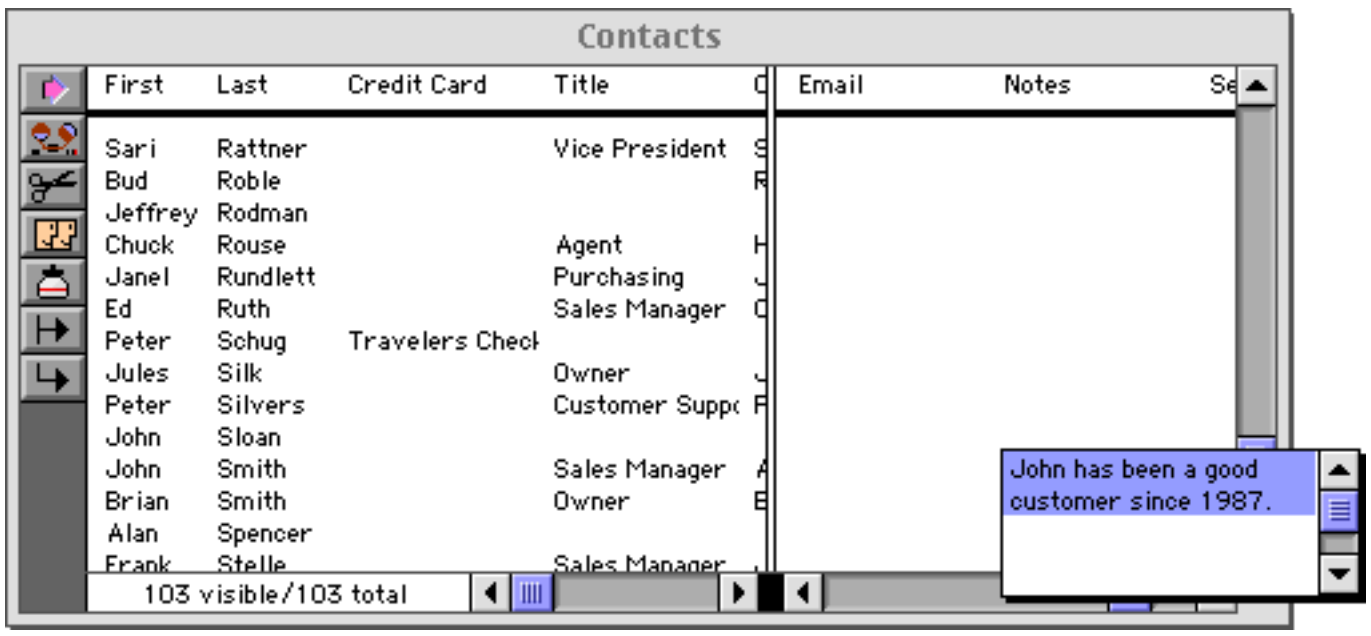
When used on a data sheet window the result of this procedure looks like this.



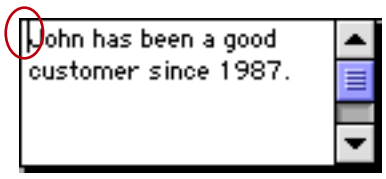
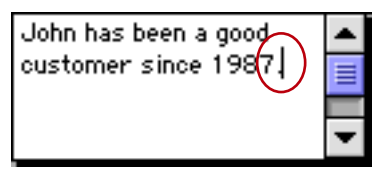
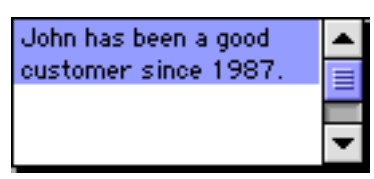
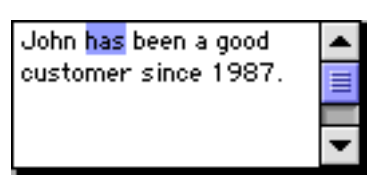
We recommend that the `editcell` statement be the last statement in a procedure. If it is not the last statement in the procedure you should use the `editcellstop` statement, like this.

```
if info("trigger") contains "Add Record"
   addrecord
   editcellstop
endif
```

If the cell contains data the `editcell` statement normally selects all of the data when it opens the input box, like this.



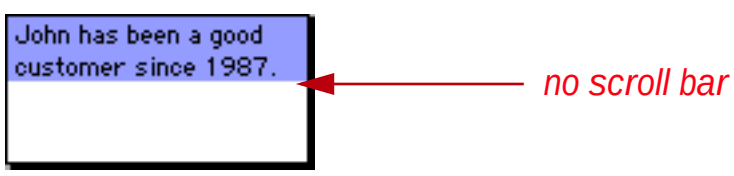
The `editselect` statement allows a procedure to control what text is selected (see "EDITSELECT" on page 5187). It must be used immediately before the `editcell` or `editcellstop` statement. The `editselect` statement has two parameters: start and end. Both parameters are numbers from 0 (first character) to 32768 (last character). The table below shows the effect of different parameters with this statement.

Code	Result
`editselect 0,0` `editcell`	
`editselect 32768,32768` `editcell`	
`editselect 0,32768` `editcell`	
`editselect 5,8` `editcell`	

The input box normally has a scroll bar when it is more than about an inch high. The `noeditscroll` statement suppresses the scroll bar no matter how high the input box is (see “NOEDITSCROLL” on page 5541). This statement is designed to be used as a prefix for the `editcell` and `editcellstop` statements, like this.

```
noeditscroll
editcell
```

The input box appears without a scroll bar.



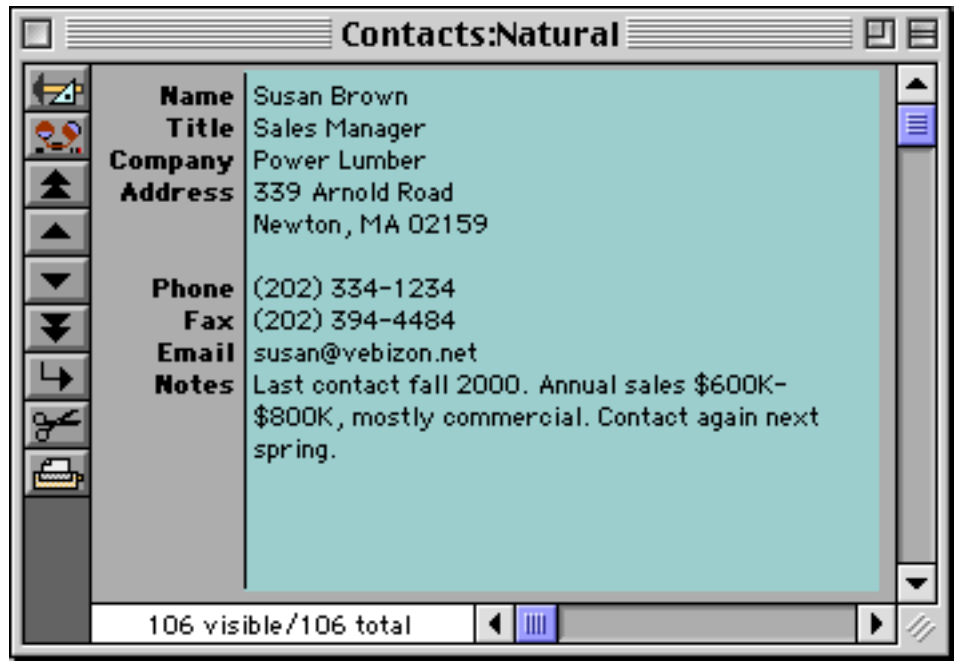
In a form window a procedure can even open an input box in “thin air” in an arbitrary location. No data cell object on the form is required. To learn how to do this see “FLOATINGEDIT” on page 5257.

These statements work only with data cells. To learn how to control editing within a Text Editor SuperObject see “Text Editor SuperObject Commands” on page 1790.

“Natural” Data Entry

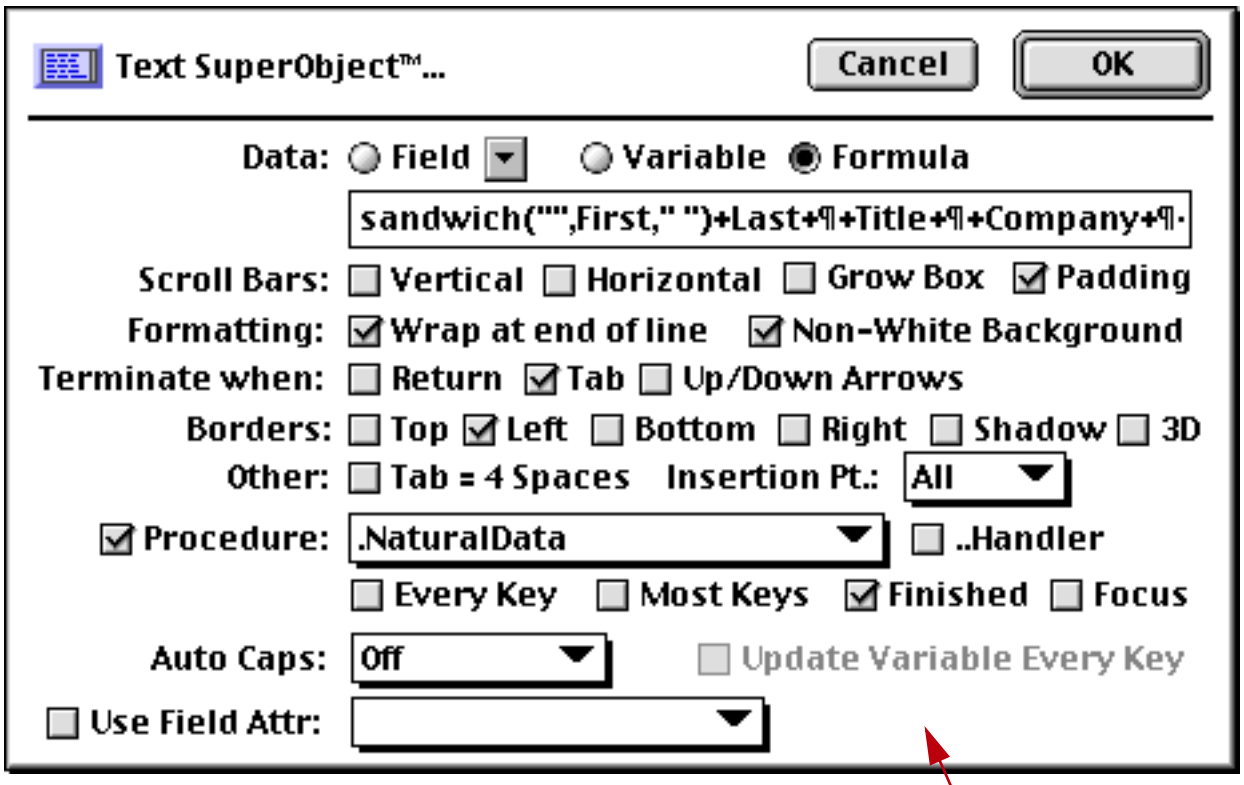
Computers and people often don’t think alike. Computers tend to use rigid formats, while people like to be more free-form. In the case of databases with contact information it’s best to store lots of separate fields for first and last names, street address, city, state, zip etc. (as shown on the left below). This gives the most flexibility in sorting, selecting and reporting data. However, from a data entry point of view it would be much nicer to enter data in a more natural format (as shown on the right).

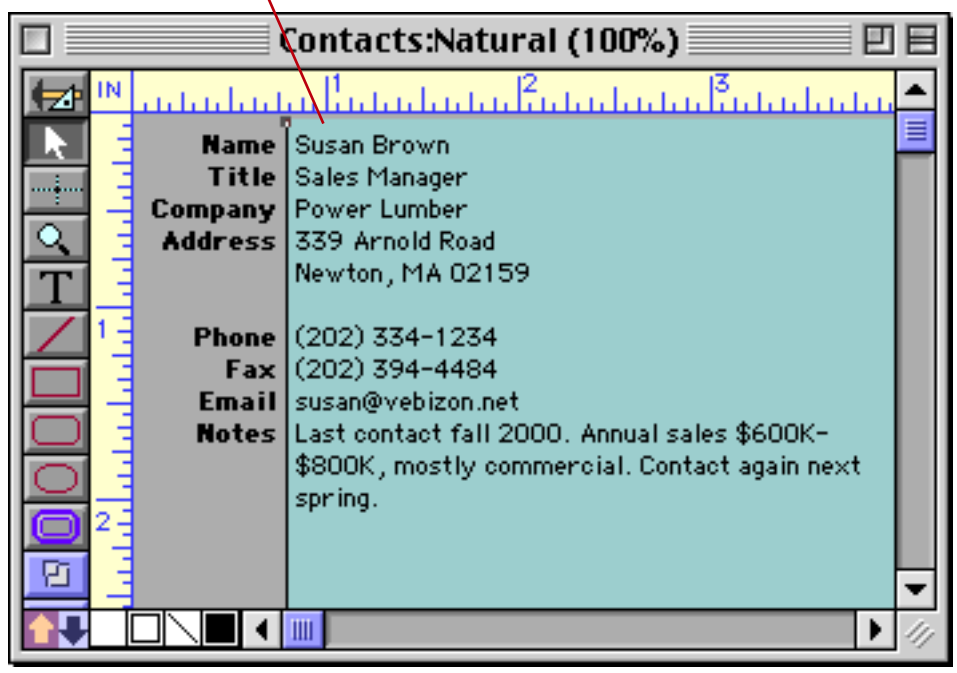




Natural Data Display

To display data in a natural format use a Text Display SuperObject (see "Text Display SuperObjects™" on page 608) or a Text Editor SuperObject (see "Text Editor SuperObject" on page 639) with the Formula option enabled (see "Text Editor Options" on page 643). Of course if you want to be able to edit the data you'll have to use the Text Editor object. Here is the configuration dialog for the Natural format text editor shown in the preceding section. (Note: The blue-green background behind the text was created with a rectangle object placed behind the Text Editor.)

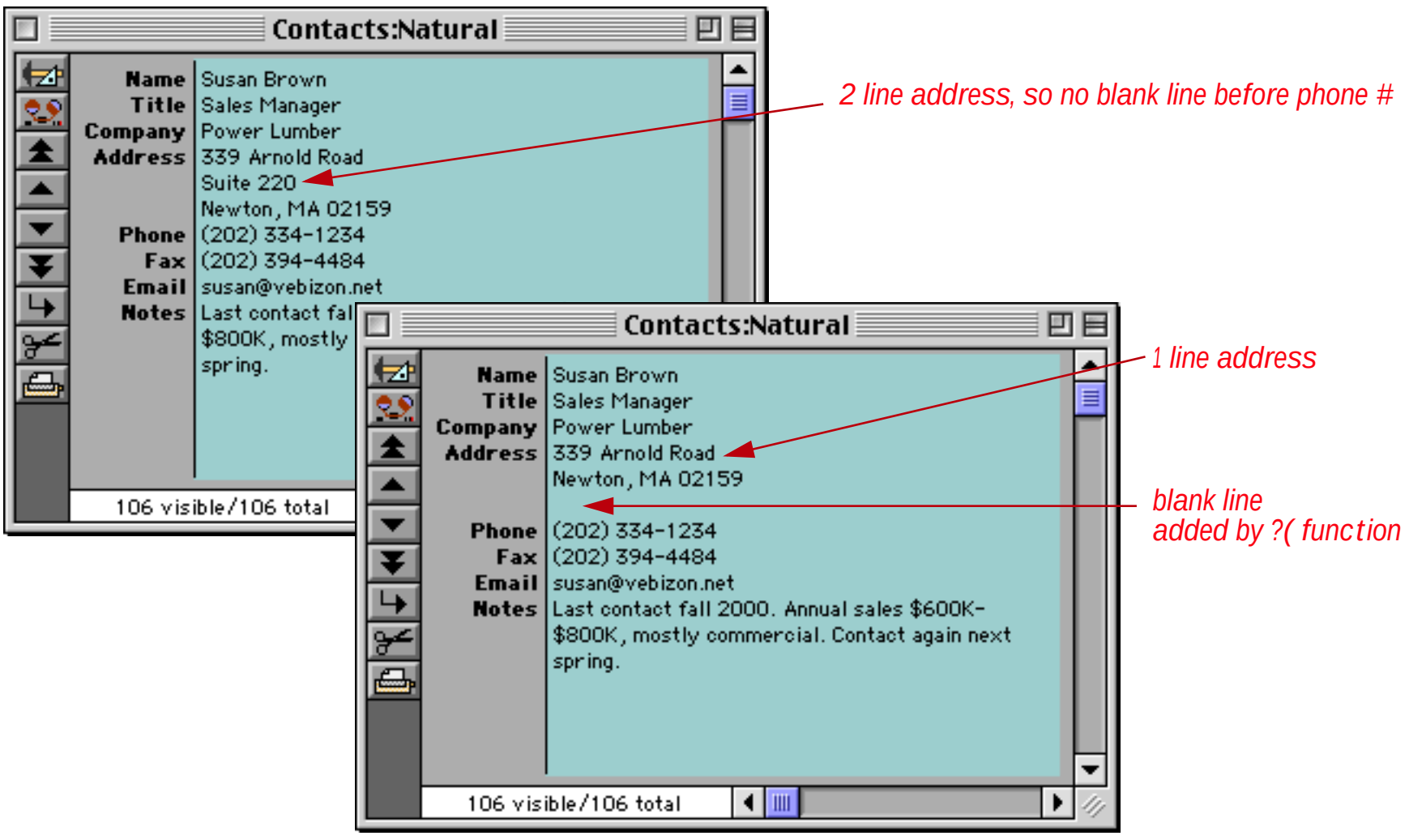




Here is the complete formula for this object.

```
sandwich("",First," ")+Last+¶+
Title+¶+
Company+¶+
arrayrange(Address,1,2,¶)+¶+
sandwich("",City,", ")+sandwich("",State," ")+Zip+sandwich(" ",Country,"")+¶+
?(Address notcontains ¶,¶,"")+
Phone+¶+Fax+¶+Email+¶+Notes
```

The `arrayrange(` function (see "ARRAYRANGE(" on page 5050) is used to extract a maximum of two lines of address (so if an address has 3 or more lines the extras will be removed). The `?(` function (see "The ? Function" on page 1271) checks to see if the Address field contains only one line, and if so, adds an extra blank line between the address and the phone number.



The `sandwich(` functions (see "SANDWICH(" on page 5689) are used to add punctuation (spaces and commas) only if needed.



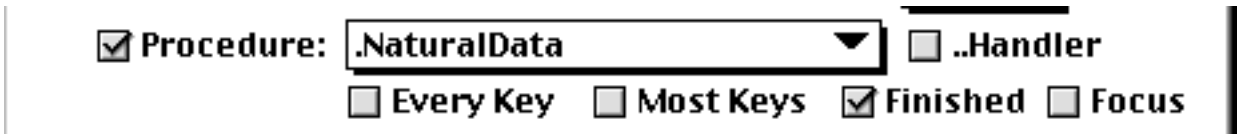
If the city name is empty (for example when entering a new contact) then there is no comma.



Since the box for entering formulas in the Text Editor SuperObject is so small you may want to test out your formula in a Text Display SuperObject first, then copy to a Text Editor once it is working correctly.

Natural Data Entry

Displaying the data in a “natural” format is only half the job. To allow the data to be entered and/or modified in this natural format you’ll need to create a procedure. In our case we’ve configured the Text Editor to trigger a procedure named **.NaturalData** when the **Enter** key is pressed.



Here is the **.NaturalData** procedure itself.

```
local FullContact,FullName,FullAddress,namePrefix,nameMiddle,nameSuffix
local xTitle,xCompany,xPhone,xFax,xEmail,zPhone,zFax,xNotes
FullContact=TextEditingResult
splitlines FullContact,
   "1W",FullName,
   "1W",xTitle,
   "1W",xCompany,
   "3W",FullAddress,
   "1",xPhone,
   "1",xFax,
   "1",xEmail,
   "0",xNotes

Title=xTitle Company=xCompany Email=xEmail Notes=xNotes
getname FullName,namePrefix,First,nameMiddle,Last,nameSuffix
getaddress FullAddress,Address,City,State,Zip,Country
getphone xPhone,"714",Country,zPhone,1,""
Phone=strip(array(zPhone,2,":"))
getphone xFax,"714","",zFax,1,""
Fax=strip(array(zFax,2,":"))
```

The procedure starts by creating the temporary variables it needs. Then it uses the `splitlines` statement (see “SPLITLINES” on page 5786) to split the incoming data into eight separate components. Most of these components are one line high, but the address contains three lines and the notes contains all of the text after the tenth line. In the process of splitting the text the `splitlines` statement also automatically capitalizes the first letter of each word in the name, title, company and address. However, the capitalization only happens if all of the text is entered in lower case, so frank rich will be converted to Frank Rich, but Scott McBride must be typed in like that, not as scott mcbride.

The `getname` statement splits the name into five separate components (see “GETNAME” on page 5306). This database only uses two of the components (first and last names) so the other three components are simply discarded. This table shows a few examples of how a name is split up into its individual components.

Sample	Prefix	First	Middle	Last	Suffix
Frank Rich		Frank		Rich	
Ms. Susan Kay Olson	Ms.	Susan	Kay	Olson	
John Kuttel DVM		John		Kuttel	DVM
General Dwight A. Eisenhower	General	Dwight	A.	Eisenhower	
Mark Jackson Jr.		Mark		Jackson	Jr.

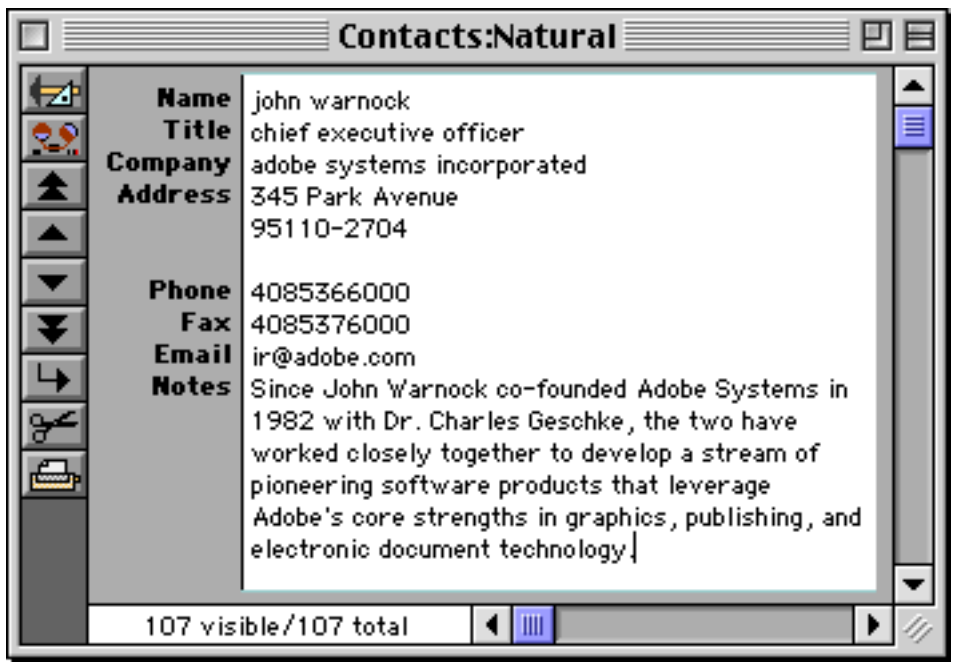
The getaddress statement splits the address into five separate components (see "GETADDRESS" on page 5288). It is primarily designed to handle US and Canadian addresses. If you purchased the optional zip code dictionary (see "Zip Code Lookup" on page 1286) you can enter just the zip code and let Panorama fill in the city and state for you (see the 3rd and 4th examples in the table below). This table shows a few examples of how a name is split up into its individual components.

Sample	Street Address	City	State	Zip	Country
575 Memorial Drive Cambridge, MA 02139	575 Memorial Drive	Cambridge	MA	02139	
445 Hoes Lane Piscataway, NJ 08855-1331	445 Hoes Lane	Piscataway	NJ	08855 -1331	
15180 Transistor Lane 92648	15180 Transistor Lane	Huntington Beach	CA	92648	
400 Seaport Court Suite 100 94063	400 Seaport Court Suite 100	Redwood City	CA	94063	
6733 Missisauga Road Missisauga, ON L5N6J5 canada	6733 Mississauga Road	Missisauga	ON	L5N 6J5	CANADA

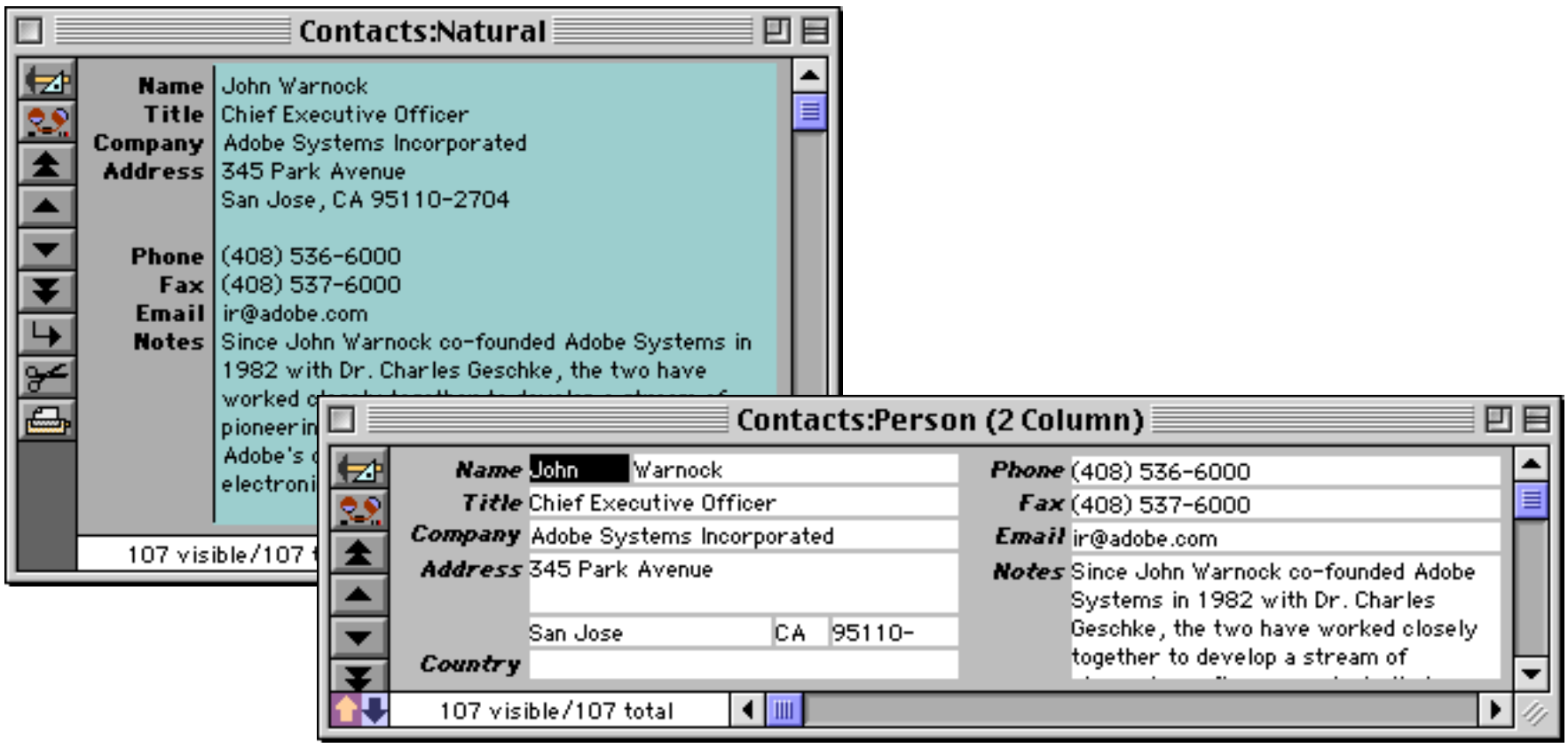
The getphone statement formats the phone number (see "GETPHONE" on page 5308) but only if the country name is blank, USA or CANADA. Our example has been set up to default to the 714 area code, but you can use any default area code you wish.

Sample	Output
5557390	(714) 555-7390
3034491234	(303) 449-1234
(412) 987-3859	(412) 987-3859
7307832x23	(714) 730-7832x23

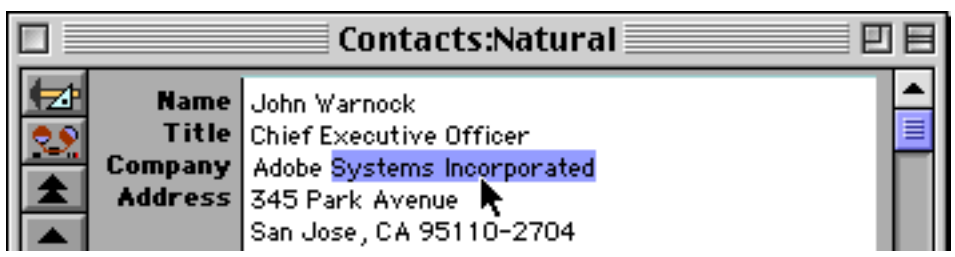
To see how all of this works together let's add a new record and type in the data. We think you'll agree that typing in the data in this natural format is easier than tabbing from field to field.



When the entry is complete press the **Enter** key. The **.NaturalData** procedure will process the entry into the separate fields in the database. Here you see both the natural display format and a form displaying each field separately.



Since the data is stored in separate fields you can easily sort, group or select it any way you want. Nevertheless you still have the convenience of entering and editing it in natural format if you wish. To edit the natural format data simply click on it and edit, then press **Enter** to update the database.



The natural data formats demonstrated in this contacts database can be used with any database you create.

Validating a Credit Card Number

Credit cards have an internal checksum that allows a number to be validated for simple data entry errors (for example missing or transposed digits). The `cardvalidate` statement checks to make sure that a number is a valid credit card number (see “CARDVALIDATE” on page 5090). This statement has two parameters. The first is the card number you want to validate. The second parameter should be a variable. The statement will set this variable to Ok or Error depending on the card number you submit. This example checks the card number in the field CCNumber to see if it is a valid credit card number.

```
local cctemp,ccvalid
cctemp=striptonum(CCNumber)
cardvalidate cctemp,ccvalid
if ccvalid<>"Ok"
   message "This credit card number is not valid!"
endif
```

The `cardvalidate` statement cannot tell whether this card number has actually been issued, what the credit limit is, or any other financial information about the card. It simply provides a basic check for missing or transposed digits within the number. Basically, if this statement says that the number is in error you know for sure that the number is wrong, but if this statement says the number is valid you would still need to check with the issuer to determine if this is a valid card.

By the way, it’s easy to determine the type of card from the first digit of the number.

Card	First Digit
	3
	4
	5
	6

Here’s another handy tip for testing. You can make a “valid” credit card number by taking the digits 1, 2, 3, and 4 and repeating them four times in any order. For example `111122223333444`, `3333111122224444` and `4444111133332222` are all valid credit card numbers (of course there aren’t really any cards with these numbers, but the checksum is ok).

Moving Data Between Files

Many database applications require multiple database files working together. For example, organizing a company's order entry operations usually requires an invoice file, an inventory / price list file, and possibly a customer file. As orders are entered, data will move back and forth between these files to adjust inventory levels, maintain customer credit records, and so forth.

Sometime you may want to display or use information from another file without actually moving it. For example, you may want to display a customer's current credit balance without actually copying it into the invoice. Panorama gives you the option of actually moving data from place to place or simply displaying or printing it.

Panorama's primary method for moving data between databases is the same method used to move data within a record: the assignment statement. Using the lookup(function (along with several variations on this function) you can turn any assignment statement into a cross database transfer. However, assignments are not the only way to transfer data across files…there are also some special statements that are especially useful for multiple-record transfers.

Warning: Panorama can only move data between database files that are currently open. If a database is not open, you must open it before data transfer can take place.

Cross Database Assignment Statements

The primary method for moving data between databases is assignment statements. If you are reading this chapter straight through this may puzzle you, since earlier we stated that assignment statements always worked on the current record on the current database. For example, consider this statement.

```
A=B+C
```

All three of the elements of this assignment (A, B and C) must be either fields in the current record or variables.

To break the limitation to the current record, the assignment statement must use a special function, called a *transfer function*. Panorama has about a half a dozen transfer functions, but at the basic level they all work the same way…you tell them how to find some data and they go get it.

```
A=transferfunction(<how to find the data>)
```

Each transfer function has a series of parameters that tell it how to find the data you want. The transfer function uses these parameters to find the data and bring it back to the formula.

Except for the fact that transfer functions retrieve data from other database files, these functions are just like any other function. You can use them in combination with other operators and functions. For example, suppose you had a transfer function with parameters designed to retrieve a person's name. You could use this function to construct a greeting for a letter like this:

```
Greeting="Dear "+upperword(transferfunction(<how to find the name>))+","
```

You can only transfer one item of data per assignment statement. If you want to transfer more than one item of information, you'll have to transfer one at a time (with some special exceptions explained later). The transfer functions are designed so that transferring several items of data from a single record is a very fast operation.

Identifying Data to Move

Moving data is easy. The hard part is explaining to Panorama exactly what the data is you want it to move. Unfortunately, we usually can't simply point at the data and say "there it is, go get it." Most of the next few sections will deal with how to explain where the data is in an unambiguous way so Panorama will understand and obey.

Panorama has two basic ways for identifying data to move: by position and by value. Data that is identified by position really is identified by pointing at it and saying “go get it.” For example, you might ask the user to find an item in the price list and click on it. Once the user has located the item, Panorama can easily move the data somewhere else with the grabdata(function.

The second way to identify data to move is by value. When this technique is used, we tell Panorama what we are looking for and it goes and finds it for us. For example, we may want to find the current credit balance for Gargantuan Widget Company. In this case, the value is Gargantuan Widget Company. Panorama will scan the database looking for this value, and if it finds a record containing the value, it moves the credit balance from that record to the currently open database.

Identifying data by value is somewhat complicated because the result is uncertain. There are basically three possible outcomes: 1) The value doesn’t exist anywhere in the database, 2) There is one record with this value in the database, or 3) There are many records with this value in the database. To write a successful program for moving data between databases you must decide how your procedure will handle each of these situations.

Transfer Function Parameters

All transfer functions have similar parameters for locating the data to transfer. (Note: The parameters do not necessarily appear in the order presented below.)

The first two parameters listed below, Target Database and Data Field, are used by all transfer functions. The remaining parameters, Key Field, Key Value, Default Value, Summary Level and Separator are used only by transfer functions that identify data by value.

Target Database. This parameter is the name of the database that contains the data you want to retrieve. Usually this is a fixed name, so you would simply enter the name surrounded by quotes, for example "Price List" or "Customer File". If the name of the target file may change you could put the name in a variable. By the way, the target database must be open, or the transfer function will not work.

The target database may even be the currently open database. This allows you to transfer data from other records in the database to the current record. To specify the current database use the info("databasename") function. (You could simply type in the name of the current file surrounded by quotes, but if the file is ever renamed the transfer function will stop working.)

Data Field. This parameter is the field that contains the actual data you want to retrieve. It must be a field within the target database. For example, if you wanted to look up a person’s phone number, this would be a field named something like Phone.

The data field should contain data you don’t already know. For example, if you know a company’s name and want to know their address, the data field would be Address. (The CompanyName field would be the key field, see below.)

Each transfer function can transfer one and only one data field. You can’t use a formula here like

```
City+" "+State+" "+Zip
```

you must specify the name of a single field in the target database. If you want to transfer multiple fields you’ll generally have to use multiple assignment statements and transfer one field at a time.

The type of data the transfer function returns depends on the type of data contained in the data field. If the data field is a text field, the transfer function will return a text value. If the data field is a numeric field, the transfer function will return a numeric value. (Exception: If the transfer function is allowed to return multiple matches, the data is always converted to text.)

Note: Prior to version 3.0, Panorama required that the data field name be surrounded by quotes when used in a transfer function. This is no longer necessary, but is still allowed for compatibility with database files created with older versions. Now you can simply type the field name in without quotes. (However, if the field name contains punctuations or spaces it must, of course, be surrounded by chevrons « and ».)

Key Field. This parameter is the field in the target database that contains the data you already know about. For example, suppose you know someone's name but not their phone number. The Name field would be the key field. The transfer function will scan the key field looking for an exact match with the data supplied as the Key Value parameter.

Only a single field may be used as the key field, and it must match up with the data in the key value. Suppose you want to look up the phone number for Joe Smith, and your target database contains separate fields for first and last name. You've got a problem, because your key value does not match up with any one field in the target database. The possible solutions include: 1) matching up on the last name only, 2) creating a new, redundant field in the target database that contains both first and last names, or 3) using the arraybuild statement instead of an assignment statement to transfer the data. (The `arraybuild` statement allows any formula to be used as a match, so you are not limited to a single key field and a single key value. Using the arraybuild statement to transfer data between files will be described in detail in a later section.)

Note: Prior to version 3.0, Panorama required that the key field name be surrounded by quotes when used in a transfer function. This is no longer necessary, but is still allowed for compatibility with database files created with older versions. Now you can simply type the field name in without quotes. (However, if the field name contains punctuations or spaces it must, of course, be surrounded by chevrons « and ».)

Key Value. This parameter is the value that identifies the data you want to retrieve. For example, if you are looking up data in a price list database, the key value would be values like 180 ohm resistor or 3/8" lock washer. The key value must exactly match a value in the key field. If the key field contains a part called a 250 µf capacitor the key value must be exactly that, not 250 µf Capacitor or 250 µf capacitor (50V).

The key value is usually assembled from one or more fields in the current database. Unlike the key field, the key value may be assembled with any formula you want to use. For example, suppose your target database contains names in a single field (for example Maureen Livingston or Steve Toyota) but the current database has separate fields for first and last names. No problem here, just use a formula like `First+" "+Last` for the key value. Remember, the key value is a value, so any formula may be used to calculate it. The key field is a field, and must be a single field in the target database.

Default Value. What if the transfer function fails to locate the key value anywhere in the target database? There is no data to retrieve, but the transfer function must return some value. This is the job of the default value. The default value steps in when the requested data is not available for any reason.

The default value should have the same data type as the data field. If the data field is text, the default value should be text. If the data field is numeric, the default value should be numeric. The most common default values are "" (empty string) for text, and 0 for numeric. However, you can use any value you want.

Sometimes you may want to create an assignment that moves data from another database to a field in the current database, but leave that field untouched if the data cannot be found. In this case, the default value should be the field itself. That way if the data is not found Panorama will simply move the current field to itself, essentially leaving it untouched. The partial example below shows how this works. This transfer function looks up a phone number, but if it is not found, uses the current phone number as the default.

```
Phone=transferfunction(...,Phone,...)
```

Summary Level. Panorama normally scans all records in the target database looking for a match between the key value and the key field. However, if you set the summary level parameter to a non-zero value, the transfer function will only scan summary records. The summary level parameter should be a number from 0 to 7. If the value is set to 2, only summary levels 2 and above will be scanned, etc. Unless you specifically want to work with summary records, set this parameter to zero.

Separator. This parameter is only used by transfer functions that can return more than one match at a time, like lookupall(. Each returned data item is separated by the text in this parameter. Common separators include commas, slashes, carriage returns (¶) and tabs (¬). For example, suppose you want a transfer function to return a comma separated list of all the invoice numbers for a certain customer. The separator is a comma character (",").

Single Record Transfer Functions

Enough theory, let's get down to specifics. The transfer functions described below locate a single record in a database and retrieve one item of data from that record.

grabdata(target database,data field)

The grabdata(function identifies data by position. This function retrieves data from the current record in the specified database. For example, if the user has selected 7404 hex inverter in the Price List database by clicking on it (or searching for it with the **Find/Select** command), this procedure will copy the information into the current line item in the Invoice database.

```
ItemΩ=grabdata("Price List",Part)
PriceΩ=grabdata("Price List",Price)
```

The drawback of the `grabdata(` function is that the user must manually locate the data, but sometimes this is exactly what you want.

A sometimes useful trick with the `grabdata(` function is to grab data in the current database. This allows you to specify the field name with a variable.

lookup(target database,key field,key value,data field,default value,summary level)

The `lookup(` function identifies data by value. When this function is used, Panorama scans every record in the target database starting from the top (including unselected records). When it finds the first match between the key value and the key field, it stops scanning and returns the value in the data field of that record. There may be 1, 10, 100 or 1,000 possible matches, but the `lookup(` function will only return the first one.

For our example, suppose we have two databases: Invoices and Customers. These two databases have 5 identical fields: Company, Address, City, State and Zip. When the user enters the company name, we want to write a procedure that will automatically move the address, city, state and zip from the customer file into the new invoice.

```
gettext "What company name?",Company
Address=lookup("Customers",Company,Company,Address,"",0)
City=lookup("Customers",Company,Company,City,"",0)
State=lookup("Customers",Company,Company,State,"",0)
Zip=lookup("Customers",Company,Company,Zip,"",0)
```

Since each `lookup(` statement can only transfer one value, four lookups are required. Panorama doesn't actually scan the Customer file four times. When a procedure performs multiple lookups with the same target database, key field, and key value Panorama realizes that it doesn't have to re-scan the database—it already knows where the data is, and it just goes and gets it. (If your database is set up for it, you can use the `speedcopy` statement to move this data even faster. See the description of this statement later in this chapter.)

lookuplast(target database,key field,key value,data field,default value,summary level)

The `lookuplast(` function identifies data by value. When this function is used Panorama scans every record in the target database, but unlike the `lookup(` function, this function starts from the bottom and scans up (including unselected records). When it finds a match between the key value and the key field, it stops scanning and returns the value in the data field of that record. There may be 1, 10, 100 or 1,000 possible matches, but the `lookuplast(` function will only return the last such match in the file. (However, there is an exception. If you are scanning the current database, the `lookuplast(` function will skip the current record. This allows you to find the last match in the file not including the current record.)

When the key field and key value may match many times, we are often more interested in the last match instead of the first match. For example, if a customer has many invoices with us, we are probably more interested in the most recent invoice than in the first invoice from long ago. (Of course this is assuming the database is sorted in chronological order!) The lookuplast(function allows us to locate this most recent information.

Our `lookuplast(` example finds the most recent order for a given customer, and the amount of that order.

```
local theCustomer,lastOrderDate,lastOrderAmount
theCustomer=""
gettext "Customer name:",theCustomer
lastOrderDate=lookuplast("Invoice",Company,theCustomer,Date,0,0)
if lastOrderDate=0
   message "No previous invoices for this customer."
   stop
endif
lastOrderAmount=lookuplast("Invoice",Company,theCustomer,Total,0,0)
message theCustomer+"'s most recent order was for "+
   pattern(lastOrderAmount,"$#,.##")+" on "+
   datepattern(lastOrderDate,"Month ddnth, yyyy")+"."
```

Here's another example that combines `lookup(` and `lookuplast(`. This example first tries to look up a customer in the Customers database. If they are not found there, it checks to see if there is a previous invoice for this customer.

```
Address=lookup("Customers",Company,Company,Address,"",0)
if Address≠""
   City=lookup("Customers",Company,Company,City,"",0)
   State=lookup("Customers",Company,Company,State,"",0)
   Zip=lookup("Customers",Company,Company,Zip,"",0)
else
   Address=lookuplast(info("database"),Company,Company,Address,"",0)
   City=lookuplast(info("database"),Company,Company,City,"",0)
   State=lookuplast(info("database"),Company,Company,State,"",0)
   Zip=lookuplast(info("database"),Company,Company,Zip,"",0)
endif
```

Remember, the `lookuplast(` function locates the matching information that is physically closest to the bottom of the database. What this proximity to the bottom means depends on how the database is sorted.

lookupselected(target database,key field,key value,data field,default value,summary level)

The `lookupselected(` function identifies data by value. When this function is used, Panorama scans every selected record in the target database starting from the top. Unlike a regular `lookup(`, this function skips unselected records.

There are two possible advantages to using the `lookupselected(` function. If the target database contains only a small percentage of selected records compared to unselected records, `lookupselected(` will be faster than `lookup(`. (Of course if the data you want to locate is not selected, `lookupselected(` won't find a match.)

The primary advantage to using `lookupselected(` is the ability to control exactly what records are scanned as part of the lookup. This is especially useful if there is a possibility of more than one match between the key value and key field.

table(target database,key field,key value,data field,default value,summary level)

The `table(` function is the only transfer function that does not require an exact match between the key value and the key field. If it does not find an exact match, the `table(` function will accept a match that is merely "close enough." The table function is designed to be used with rate lookup tables like tax tables, shipping tables, volume discount tables etc. If the `table(` function does not find an exact match between the key value and the key field, it will pick the record in the target database where the key field is closest to, but not

greater than, than the key value. For example, suppose the key field contains the values 5, 25, 100, 250 and 1000. If the key value is 47, the table(function will match with the record containing 25 in the key field. If the key value is 4700, the table(function will match with the record containing 1000 in the key field. If the key value is 4, there is no match, because there is no value in the key field less than 4. In this case the default value will be used.

Suppose you have a database called Shipping Rates that contains the fields and values shown here.

Weight	Zone1	Zone2	Zone3
0	2.50	4.00	5.00
50	2.35	3.80	4.70
100	2.25	3.60	4.50
250	2.12	3.40	4.25
500	2.03	3.00	4.05
1000	1.94	2.85	3.85
2000	1.86	2.70	3.70

The `table(` function interprets this table like this: From 0-49 pounds in Zone 1, the rate is $2.50 per pound. From 50-99 pounds the rate is $2.35 / pound. From 100-249 the rate is $2.25 per pound, and so on. Items 2,000 pounds and over are shipped for $1.86 per pound. The other zones are similar.

The procedure below calculates the shipping charges for a package.

```
local PackageWeight,DestinationZone,ShippingCharge
PackageWeight="" DestinationZone=""
gettext "Package weight:",PackageWeight
PackageWeight=val(PackageWeight)
if PackageWeight<0
   message "Sorry, anti-gravity option not available until 3rd quarter."
   stop
endif
gettext "Zone Number (1-3)",DestinationZone
if length(DestinationZone)≠1 or DestinationZone<"1" or DestinationZone>"3"
   message "Zone must be from 1 to 3"
   stop
endif
ShippingCharge=PackageWeight*
   table("Shipping Rates",Weight,PackageWeight,
        "Zone"+DestinationZone,0,0)
message "Shipping charge is: "+pattern(ShippingCharge,"$#,.##")
```

Notice that this example actually calculates the name of the data field on the fly: either Zone1, Zone2, or Zone3. The data field is still a single field (remember, only one item can be transferred at a time) but we are using a formula to calculate what the name of that field is.

In a real database you probably would not ask the user to enter the zone, but would have another database that would relate zones to zip codes. Here's a simple Zone Chart database that divides the entire USA into three zones based on the first three digits of the zip code.

Zip3	Zone
000	1
300	2
700	3
99:	0

The last value in this table, 99:, is the smallest value that is greater than the last legal zip code (999) according to the ASCII character order. This record can help catch illegal zip codes. For instance, ABC is greater than 99:, so the Zone will be 0 for this illegal zip code.

The assignment below will turn a regular zip code (Zip) into a zone number according to the Zone Chart database.

```
DestinationZone=table("Zone Chart",Zip3,Zip[1,3],Zone,0,0)
```

This assignment can easily be plugged into the previous example to calculate the shipping charges given the weight and zip code.

Clairvoyance and Lookups

When data is transferred between two databases with a lookup, there is usually one field in the current database that contains the key value. For example, when we look up a company name from the Customer database, we usually get the name from the Company field in the current database. For the purposes of this discussion we'll call this field in the current database the key value field. This is different from the key field, which is in the target database. However, both of these fields contain the same data: the same customer names, or the same part descriptions, the same account numbers, etc.

Since the key value field and the key field contain the same data, it makes sense to link them with Panorama's Clairvoyance® feature. This allows you to type in the first few letters of a customer name (or a part description, or an account number, etc.) and then let Clairvoyance complete the entry from the data in the key field in the target database. When it is used this way, Clairvoyance is almost like a pre-lookup. The beauty of this scheme is that when Clairvoyance completes the entry for you, you know that the key value is entered 100% correctly and that the lookup will work perfectly.

To set up Clairvoyance between fields in different databases you'll need to use the design sheet. Open the design sheet for the current database, then click on the name of the field that contains the key value. Now choose Clairvoyance Link from the Special menu. A dialog with a list of databases will appear. Choose the target database. Now a list of fields in the target database will appear. Choose the key field. Press the **OK** button, then use the New Generation tool to update the database itself. Your cross-database Clairvoyance link is now set up. (Of course, this link will only work if both databases are open.)

The SpeedCopy Statement (Multiple Assignments in One Statement)

Some applications require you to transfer many fields from one database to another. Normally this requires a separate assignment for each field you want to move. This isn't so bad for a few fields, but if you need to transfer, say, 20 fields it gets tedious and slow. The `speedcopy` statement can transfer many fields at once, but only if the fields to be copied in the two databases match exactly. The fields to be copied must appear in exactly the same order in both databases, and the fields must have the same data types. With all these restrictions, you may be surprised to find out that the fields do not have to have the same names!

Here's how `speedcopy` works. Before you use `speedcopy`, you must perform an assignment with a `lookup(` function (actually, any single record transfer function will do: `lookuplast(`, `lookupselected(`, etc.). The `lookup(` function locates the record containing the information to be copied.

The `speedcopy` statement has three parameters.

```
speedcopy FirstAssignField,LastAssignField,FirstTargetField
```

The first two parameters are fields in the current database. The last parameter is a field in the target database. All of these field names should be surrounded by quotes (for example "Name", not Name). `Speedcopy` starts by converting these field names into field numbers. For example, if a field would be the third column in the data sheet, it is field #3.

Once speedcopy has converted the field names into numbers, it starts copying data. Suppose the FirstAssignField was field number 3, and the FirstTargetField was field number 8. Speedcopy will start by copying field #8 in the target database into field #3 in the current database. Then it will copy field #9 in the target database into field #4 in the current database. It will continue copying fields until it has copied something into the LastAssignField.

To show a specific example, suppose we have two databases, Organizer and Customers, with the fields listed below:

#	Organizer	Customers
1	Name	Company
2	Title	Address
3	Company	City
4	Address	State
5	City	ZipCode
6	State	Phone#
7	Zip	Fax#
8	Phone	Cust#

The procedure below will quickly copy the Address, City, State, Zip and Phone fields from the Customers database to the Organizer database.

```
Address=lookup("Customers",Company,Company,Address,"",0)
if Address≠""
   speedcopy "City","Phone","City"
endif
```

Let's take a close look at how this procedure works. The first line attempts to lookup the Address from the Customer database. If this lookup fails, the procedure is finished. However, if the lookup succeeds the procedure continues with the speedcopy statement.

The first parameter of the speedcopy statement is City, which is field #5 in the current database (Organizer). The second parameter is Phone, which is field #8 in the current database. The final parameter is City, which is field #3 in the target database (Customers).

In this example speedcopy will copy 4 fields from Customers into Organizer. These four data moves are:

City into City(Customers field #3 into Organizer field #5)

State into State(Customers field #4 into Organizer field #6)

ZipCode into Zip(Customers field #5 into Organizer field #7)

Phone# into Phone(Customers field #6 into Organizer field #8)

As speedcopy moves data from one database to another, it doesn't make any kind of checks on the data. If the fields aren't really in the same order, speedcopy will cheerfully copy them in the wrong order. Even worse, if you try to copy a numeric field into a text field or a text field into a numeric field, speedcopy will not object, but will speedily turn your current database into swiss cheese. The moral of the story is to use the speedcopy statement very carefully. Like any sharp instrument you want to make sure it is pointed in the right direction before you use it.

Multiple Record Transfer Functions

The transfer functions described below locate every matching record in the target database, and return all the data field values in the records that match. The data field values are strung together as a single piece of text, with the separator character (or characters) in between each item. If the separator is a single character, the result is a text array.

lookupall(target database,key field,key data,data field,separator)

The `lookupall(` function builds a text array containing one item for every record in the target database where the data in the key field matches the key value. Each item in the text array contains the value extracted from the data field for that record. If the data field is a numeric or date field, it is converted to text using the default patterns for that field.

Here is an example that looks up all the checks written to a certain person or company. The checks are displayed with a comma in between each check number.

```
local CheckTo,Checks
CheckTo=""
gettext "List checks written to:",CheckTo
Checks=lookupall("Checkbook",Payee,CheckTo,«Check#»,",")
if Checks=""
   message "No checks written to "+CheckTo
else
   message "Checks written to "+CheckTo+":"+Checks
endif
```

The `lookupall(` function will often return a lot of duplicate data. Since the result is an array, you can use the arraydeduplicate statement to sort and eliminate the duplicates. This example produces a sorted list of customers in Arizona.

```
global theCustomers
theCustomers=lookupall("Invoices",State,"AZ",Company,¶)
arraydeduplicate theCustomers,theCustomers,¶
```

The `lookupall(` function is especially useful for displaying or printing lists of items, and for other user interface elements like lists or pop-up menus. The `lookupall(` function can be used directly in auto-wrap text or a Text Display SuperObject to display a list, or as part of the data formula for a Super Matrix object.

lookupcalendar(target database,key field,key data,data field,separator)

The `lookupcalendar(` function is identical to the `lookupall(` function except for the way that the key value and the key field are compared. The `lookupcalendar(` function requires that the key field be a field that contains reminders, and the key value be a date. (To refresh your memory, a reminder is a special data type that holds scheduling information.) The `lookupcalendar(` function will locate all records where the reminder and the date match. For example, if the date specified is 5/23/95, lookupcalendar(will match for reminders that are set for 5/23/95, or are set for 5/23 of every year (annually), or are set for every Tuesday, etc.

As you might guess, this function is very handy for calendars. Using this function you can display all the reminders for a specific date.

lookuprtime(target database,key field,key data,pattern,separator)

Lookuprtime is short for lookup reminder time. The `lookuprtime(` function is similar to the `lookupcalendar(` function, but it returns the actual time of the reminders instead of information in a separate data field. Like `lookupcalendar(`, the key field must be a field that contains reminders, and the key value must be a date. The pattern parameter tells the function how to format the time into text, for example "hh:mm am/pm". You may use any pattern supported by the timepattern(function.

The example below will fill the global variable Agenda with the items on today's schedule.

```
global Agenda
local todayTimes,todayMessages
todayTimes=lookuprtime("Reminders",When,today(),"hh:mm am/pm",¶)
todayMessages=lookupcalendar("Reminders",When,today(),Message,¶)
arrayfilter todayTimes,Agenda,¶
   import()+" - "+array(todayMessages,seq()-1,¶)
```

The list of items in Agenda will be formatted something like this:

```
7:00 am - Breakfast meeting with Bob
9:25 am - Make sure Williams got our quote
1:30 pm - Late lunch with Jennings group
3:45 pm - get prepped for staff meeting
4:00 pm - Weekly staff meeting
5:30 pm - Don't forget flowers for Pat
```

This list can easily be displayed with an auto-wrap text object, a Text Display SuperObject, or a List SuperObject.

After a Lookup…Modifying the Original Data

There is no specific function or statement that allows you to modify data you have looked up. However, it's not difficult to write a procedure that does this. The basic principle is to use the find statement to locate the data, then modify the data.

The example below looks up a price from an Inventory database, then goes to the Inventory database, locates the part with the find statement, decreases the quantity on hand, then goes back to the original window.

```
local originalWindow,theItem,theQty
PriceΩ=lookup("Inventory",Part,ItemΩ,Price,0,0)
if PriceΩ=0
   stop
endif
AmountΩ=QtyΩ*PriceΩ
GrandTotal=sum(AmountΩ)
originalWindow=info("windowname")
theItem=ItemΩ
theQty=QtyΩ
window "Inventory"
if info("records")>info("selected")
   selectall
endif
find Part=theItem
QtyOnHand=QtyOnHand-theQty
window originalWindow
```

(This example is actually a bit unrealistic because the normally you would not want the inventory to be updated immediately as each price was looked up. Instead you would probably wait until the entire invoice was complete and post all the inventory changes at one time. Nevertheless, the basic technique is the same.)

The search technique is a bit awkward to use for modifying data that was retrieved by a `lookupall(` function (or any multiple record transfer function). For example, suppose you want to modify the third item retrieved by the `lookupall(` function. You could use the find statement in a loop to find the third match in the target database, but Panorama has a special statement that helps perform this job: the `allindex` statement. This statement has two parameters:

```
allindex item,transferformula
```

The item parameter must be a variable. Before you use the `allindex` statement you must set this variable to the number of the item you want to locate: 1, 2, 3, etc. Afterwards the variable will contain the line number in the target database that corresponds to this item, or zero if there is no such item.

The second parameter, transferformula, should contain the multiple record transfer function (lookupall(, lookupcalendar(, etc.).

The previous section showed an example that displayed an agenda for a particular day. For this example we'll assume that this agenda has been loaded into a List SuperObject™, and that this SuperObject has been set up to trigger the procedure below whenever the user clicks on an item. This procedure will find out what item the user selected, then use the allindex and find statements to locate the original record in the Reminders database for that item.

```
local AgendaItem
AgendaItem=1
superobject "AgendaList","FindSelected",AgendaItem
if AgendaItem=0
   stop /* nothing clicked! */
endif
allindex AgendaItem,lookupcalendar("Reminders",When,today(),Message,¶)
window "Reminders"
if info("selected") < info("records") selectall endif
find seq()=AgendaItem
/* now we can modify the original data the mouse was pointing to */
editreminder When,Message
```

The `allindex` statement only works if you display the data directly from the `lookupall(` function (or other multiple record lookup.) If the data is filtered, sorted and/or de-duplicated the `allindex` statement will not be able to correctly locate the original data.

Using Lookups for Display/Printing

So far, all our examples have used lookups and other transfer functions to actually transfer data from one database to another via transfer functions. However, since these are functions, they can be used in a formula anywhere. For example, you can use a lookup in an auto-wrap text object or in a Text Display or Text Editor SuperObject.

Let's take an invoice file to show some examples of how lookups can be used for display. If all of your customers are listed in a customer database, you can display the address on the invoice without actually having fields in the database for the address. Use an auto-wrap text object or Text Display SuperObject with a lookup formula to display the address. One advantage of this approach is that if you change the address of the company in the customer database, all the invoices will automatically show the new address. A possible disadvantage of this approach is that there is no way to enter the address of a company that is not in the customer database. You must enter the company in the customer database first, and then create the invoice.

Another application on an invoice would be to display a list of all the previous invoices for this customer. To do this you would use the lookupall(function. As you flip from invoice to invoice the new list is automatically displayed. You might want to use more than one auto-wrap text object or Text Display object; perhaps one to display the invoice numbers, one to display the invoice dates, and one to display the invoice amounts. If these are placed next to each other, they will line up in a neat columnar display.

One place where you probably don't want to simply display the result of a lookup is prices. We've already shown examples where a price is transferred from a price list database into the invoice database. You could simply display the price instead of storing it in the invoice…you can even perform calculations on the price as part of the display formula. But what if the price list changes? You probably don't want your old invoices to change every time a price changes—these old invoices are a historical record of the transactions that actually took place. By transferring the price into the invoice database you are "freezing" it and isolating it from later changes in the price list database.

The point of this section is: Don't always assume that you must transfer data from one database to another to link them. Look at each situation carefully. Do I need to transfer the data to isolate it from further changes? Do I want the data to continue to be linked and to reflect changes in the original database? How much additional processing do I need to do to this data after it has been transferred? Don't always do it the same way, but pick the best solution for each individual situation.

Using ArrayBuild to Transfer Data Between Files

The `arraybuild` statement can scan any open database and extract data from it. Does that sound familiar? The `arraybuild` can act as a super duper lookup with several advantages. The table below summarizes the differences between using regular lookup functions and the arraybuild statement for retrieving data from another database:

	Lookup Functions	ArrayBuild Statement
Locating Data	Only 1 key field	Can combine multiple fields
Match Criteria	Exact match required	Any comparison operator (=, <, >, contains, soundslike, match, etc.)
Retrieving Data	Only 1 data field	Can combine multiple fields
Use in Procedure	Yes	Yes
Use in Form	Yes	No

The primary disadvantage of the `arraybuild` statement is that it is not a function, so it cannot be used as part of a formula. The `arraybuild` statement can only be used as part of a procedure.

Let's start by looking at how the `arraybuild` statement can be used to simulate various transfer functions. Here's an example from earlier in this chapter that uses the `lookup(` function.

```
gettext "What company name?",Company
Address=lookup("Customers",Company,Company,Address,"",0)
City=lookup("Customers",Company,Company,City,"",0)
State=lookup("Customers",Company,Company,State,"",0)
Zip=lookup("Customers",Company,Company,Zip,"",0)
```

Here's another procedure that does the same thing but using the `arraybuild` statement.

```
local theCompany,TransferArray
gettext "What company name?",Company
theCompany=Company
arraybuild TransferArray,¶,"Customers",
   ?(theCompany=Company,Address+¬+City+¬+State+¬+Zip,"")
if TransferArray=""
   stop
endif
Address=array(array(TransferArray,1,¶),1,¬)
City=array(array(TransferArray,1,¶),2,¬)
State=array(array(TransferArray,1,¶),3,¬)
Zip=array(array(TransferArray,1,¶),4,¬)
```

Ok, ok…I know you're not impressed (yet!). This example is longer and more obscure looking than the first one, and it does the same thing. But now take a look at a procedure that is impossible with the `lookup(` function. This procedure looks up a person by their last name and first initial (separate fields in the Customers database).

```
local theFirst,theLast,TransferArray
theFirst=upper(FirstName) theLast=upper(LastName)
arraybuild TransferArray,¶,"Customers",
   ?(theFirst[1,1]=upper(FirstName[1,1]) and theLast=upper(LastName),
   Address+¬+City+¬+State+¬+Zip,"")
if TransferArray=""
   stop
endif
Address=array(array(TransferArray,1,¶),1,¬)
City=array(array(TransferArray,1,¶),2,¬)
State=array(array(TransferArray,1,¶),3,¬)
Zip=array(array(TransferArray,1,¶),4,¬)
```

How does this procedure do its job? First, it transfers the data it wants to look for into temporary variables (theFirst and theLast). In this example, the data is also converted to all upper case, so that it doesn't matter if the names are upper or lower case in the two different databases.

Now the `arraybuild` statement gets to work. The `?` function uses a formula to identify people by their first initial `(theFirst[1,1]=upper(FirstName[1,1])` and also by their last name `(theLast=upper(LastName))`. When it encounters a record that matches the formula, it copies the Address, City, State, and Zip into the TransferArray variable. These four fields are separated by tabs within the array (remember, the ¬ character is a tab). For example, the TransferArray might look like this:

124 W. Olive St	San Jose	CA	95134
2347 N. Riverside	Cambridge	MA	02139
687 E. Dorothy Lane	Bothell	WA	98011
5672 Lakewood Drive	Salinas	CA	93908

In this hypothetical example the `arraybuild` statement has located four records in the target database that match the formula. Which one to choose? The procedure could choose the first match, the last match, or it could display the entire list to the user and ask them to make the choice. The procedure we have created always uses the first matching record. The inner `array(` function, `array(TransferArray,1,f)`, locates the first record that matched from the target database. The outer `array(` function pulls out the individual items: Address, City, etc.

If you wanted to find the last record that matched you would rewrite the end of this procedure like this:

```
…
…
if TransferArray=""
   stop
endif
local LastMatch
LastMatch=arraysize(TransferArray,¶)
Address=array(array(TransferArray,LastMatch,¶),1,¬)
City=array(array(TransferArray,LastMatch,¶),2,¬)
State=array(array(TransferArray,LastMatch,¶),3,¬)
Zip=array(array(TransferArray,LastMatch,¶),4,¬)
```

For some applications the best option may be to let the user choose the appropriate record if there are duplicates. To illustrate this, we'll assume that we have created a form called Select that contains a List SuperObject named Choices. The List SuperObject has been set up to display whatever list is contained in the global variable named ChoiceList. The main procedure looks like this:

```
global TransferArray
local theFirst,theLast,HowMany
theFirst=upper(FirstName) theLast=upper(LastName)
arraybuild TransferArray,¶,"Customers",
   ?(theFirst[1,1]=upper(FirstName[1,1]) and theLast=upper(LastName),
   FirstName+" "+LastName+¬+Address+¬+City+¬+State+¬+Zip,"")
if TransferArray=""
   stop
endif
HowMany=arraysize(TransferArray,¶)
if HowMany=1
   Address=array(array(TransferArray,1,¶),2,¬)
   City=array(array(TransferArray,1,¶),3,¬)
   State=array(array(TransferArray,1,¶),4,¬)
   Zip=array(array(TransferArray,1,¶),5,¬)
else
   /* duplicate records, so let user pick from list */
   arrayfilter TransferArray,ChoiceList,¶,
      array(import(),1,¬)
   opendialog "Select"
endif
```

If TransferArray contains only one record, the procedure simply fills in the Address, City, etc. But if TransferArray contains multiple records, the procedure builds a new array that contains the names of the people it found, and then opens the dialog to display the list.

When the user clicks on one of the names in the List SuperObject we'll need another procedure that actually fills in the Address, City, etc. Here is that procedure:

```
local UserChoice
UserChoice=1
superobject "Choices","FindSelected",UserChoice
if UserChoice=0 stop endif ; nothing clicked!
closewindow ; close the dialog
UserChoice=UserChoice
Address=array(array(TransferArray,UserChoice,¶),2,¬)
City=array(array(TransferArray,UserChoice,¶),3,¬)
State=array(array(TransferArray,UserChoice,¶),4,¬)
Zip=array(array(TransferArray,UserChoice,¶),5,¬)
```

If you've been studying these examples carefully you have probably noticed that in these three examples we've used the `arraybuild` statement in a manner similar to the `lookup(`, `lookuplast(` and `lookupall(` functions—but with much more flexible parameters. The possibilities for linking databases with the `arraybuild` statement are almost limitless.

Posting Data to Other Databases

So far, all of our examples have pulled data from other database files into the current record of the currently open database. What if you need to go the other way? What if you have information in the current database that needs to be moved somewhere else? This is often called “posting” data, and is essentially performing a lookup in reverse. Panorama has two statements for posting data — `post` and `postadjust`.

The Post Statement

The `post` statement assigns values to one or more fields in another database.

```
post mode,database,keyfield,keyvalue,datafield,datavalue,df2,dv2,df3,dv3 …
```

The `mode` parameter controls how the `post` statement decides which record will be updated in the target database. There are three modes available:

Mode	Description
"update"	When this mode is used, the `post` statement will search for a record where the data in the keyfield matches the keyvalue you have supplied. When this record is found, the specified datafields are updated. If a record with matching data does not exist an error will be generated (this error can be trapped with the `if error` statement).
"add"	When this mode is used, the `post` statement will not search for a matching record, in fact the keyfield and keyvalue parameters are ignored. The statement simply adds a brand new record and updates the specified data fields.
"updateadd"	When this mode is used, the `post` statement will search for a record where the data in the keyfield matches the keyvalue you have supplied. If this record is found, the specified datafields are updated. If a record with matching data does not exist a new record is added and the specified datafields are updated.

The next five parameters are very similar to the parameters for the `lookup(` function.

The database parameter is the name of the target database you want to post data to. This database must already be open.

The keyfield parameter is the name of the field in the target database you want to search (update and updateadd modes only).

The keyvalue parameter is a formula that specifies the value you want to search for in the keyfield field.

The datafield parameter is the name of the field that will be modified. The datavalue is the new value to be placed in this field. You may specify additional pairs of fields / values to modify in the target database, in fact, there is no limit to the number of field / value pairs you can include as parameters to this statement.

This example assigns the value "714" to the AreaCode field in the "My Contacts" database and it will assign "555-1212" to the Phone field and "4" to the Extension field. It will look for a matching customer name and create a new record if it can't find one.

```
post "updateadd","My Contacts","CustomerName",CustomerName,
   "AreaCode","714",
   "Phone","555-1212",
   "Extension","4"
```

The PostAdjust Statement

The postadjust statement adjusts the value of a numeric field in another database, by adding (or subtracting) a number. For example, you could use this to subtract from the quantity on hand field of an inventory database as an invoice is processed.

```
postadjust database,keyfield,keyvalue,datafield,deltavalue,minimum,maximum
```

Unlike the post statement, the postadjust statement always runs in update mode. If the postadjust statement cannot find the record specified by the keyfield and keyvalue parameters it will stop and generate an error (you can trap this error with the if error statement).

The first four parameters are very similar to the parameters for the lookup(function.

The database parameter is the name of the target database you want to post data to. This database must already be open.

The keyfield parameter is the name of the field in the target database you want to search.

The keyvalue parameter is a formula that specifies the value you want to search for in the keyfield field.

The datafield parameter is the name of the field that will be modified. This must be a numeric field.

The deltavalue is the adjustment you want to make to the datafield. This should be a positive or negative number. For example, a deltavalue of -6 means “subtract six from the datafield.”

The minimum parameter allows you to specify a minimum value for the datafield. If the adjusted value of this field would be below this value then the adjustment is not made and an error is generated. For example when adjusting an inventory database you would typically set the minimum to zero to prevent the quantity on hand from becoming negative. The error message will look like the one shown below. This error indicates that you tried to subtract 8 items from 5, which would result in -3, which is below the minimum.

```
POSTADJUST failed because adjusted value would be below minimum.
Database Value:5 Delta:8
```

If you trap the error with the if error statement you can extract these values with the info("error") function. An example of this is shown below.

If you don’t want to have any minimum value at all, set the minimum parameter to -1.

The maximum parameter allows you to specify a maximum value for the datafield. If the adjusted value of this field would be above this value then the adjustment is not made and an error is generated. For example if you were adjusting a class size you might set the maximum to the number of seats in the classroom. The error message will look like the one shown below. This error indicates that you tried to add 3 items to 285, which would result in 31, which is above the maximum of 30.

```
POSTADJUST failed because adjusted value would be above maximum.
Database Value:28 Delta:3
```

If you trap the error with the if error statement you can extract these values with the info("error") function.

If you don’t want to have any maximum value at all, set the maximum parameter to 0.

The example below shows how to link an inventory and invoice database so that the inventory quantities on hand are automatically adjusted as invoices are shipped, and back orders are marked as necessary.

The inventory database includes at least two fields for each product.

Field	Description
SKU	Identification code for each product
OnHand	Current quantity on hand

The invoice database includes at least these four line item fields for each item purchased (presumably it would also include fields with pricing information.

Field	Description
SKU1, SKU2, …	Identification code for product
Qty1, Qty2, …	Quantity customer desires to purchase
Shipped1, Shipped2, …	Total quantity shipped to customer
ShipNow1, ShipNow2, …	Quantity to ship now

The procedure below prepares an invoice for shipping. It scans each line item and determines if it needs to be shipped. If one or more items remain unshipped, it checks the inventory database. If there is insufficient inventory on hand to complete the order, it grabs whatever is available.

```
local itemNumber,itemSKU,itemQty,itemShipped,itemNeeded,itemNow,itemAvailable
itemNumber=1
loop
   itemSKU=grabdata("Invoice","SKU"+str(itemNumber))
   if error
      stoploopif true()   /* loop stops when line item field does not exist */
   endif
   if itemSKU<>""   /* ignore blank line items */
      itemQty=grabdata("Invoice","Qty"+str(itemNumber))
      itemShipped=grabdata("Invoice","Shipped"+str(itemNumber))
      if itemQty<itemShipped/* is everything shipped yet? */
         itemNeeded=itemQty-itemShipped
         itemNow=itemNeeded
         postadjust "Inventory","SKU",itemSKU,"OnHand",-itemNeeded,0,0
         if error
            itemNow=0
            itemAvailable=val(tagdata(info("error"),"Database Value:"," ",1)
            if itemAvailable>0
               /* not enough to fill order, just ask for what we can get */
               itemNow=itemAvailable
               postadjust "Inventory","SKU",itemSKU,"OnHand",-itemNow,0,0
            endif
         endif
         set "Shipped"+str(itemNumber)),itemShipped+itemNow
         set “ShipNow"+str(itemNumber)),itemNow
      endif
   endif
   itemNumber=itemNumber+1
while forever
```

When the procedure is done the inventory has been updated and the invoice will contain the quantities to ship now, if any.

Sorting

To sort the database takes two steps. First the procedure must select the field to sort by (see “Moving Left and Right” on page 1640). Then the `sortup` or `sortdown` statement is used to sort the database. The `sortup` statement sorts the database in ascending order — A’s at the top and Z’s at the bottom. The `sortdown` statement does the reverse, Z’s go at the top and A’s at the bottom. This example sorts an address book by last name.

```
field "Last Name"
sortup
```

To sort by two or more fields you must use the `sortupwithin` statement. This statement leaves the data in the original field in order but re-arranges the records within each value. For example the procedure below will sort an address book by state, and then by city within each state.

```
field State
sortup
field City
sortupwithin
```

Note: Because Panorama uses what is called a “stable sort algorithm” there is another way to sort multiple fields. Instead of using `sortupwithin` you can sort the fields in reverse order, like this.

```
field City
sortup
field State
sortup
```

Just like the previous example, this procedure will sort the address by city within state. There really isn’t any advantage to using this technique, but it is available.

To sort the database by the color of the data in a field use the `sortbycolor` statement. See “Sorting By Color” on page 363 to learn more about sorting by color.

Reducing Screen “Flashing”

When a database is sorted more than once in a row the window will redisplay over and over again. This is annoying and wastes time, also. You can eliminate this “flashing” with the `noshow` and `endnoshow` statements. This example shows a revised version of our procedure to sort an address book by city and state.

```
noshow
   field City
   sortup
   field State
   sortup
   showpage
endnoshow
```

This revised procedure will only redisplay the window once (because of the `showpage` statement). To learn more about the `noshow` and `endnoshow` statements see “Suppressing Display of Text and Graphics” on page 1427.

Making Sorts Even Faster

Panorama sorts even large databases very quickly. However, the `noundo` statement can make it sort even faster (see “NOUNDO” on page 5547). This statement disables Panorama’s undo feature. Since the sort doesn’t need to worry about undo it can run slightly faster.

```
noundo
field Company
sortup
```

Locating Information

Panorama has two ways of locating information — **finding** and **selecting** (see "Finding vs. Selecting" on page 367). In a procedure you find information with the `find` statement and select information with the `select` statement.

Finding Information

The `find` statement searches the database, starting from the top. This statement has one parameter, a formula. Starting from the top of the selected records, Panorama scans down the database until it finds a record that makes this formula true (see "True/False Formulas" on page 1265). For example, this procedure will scan down the database until it finds a record where the Park field contains Everglades.

```
find Park contains "Everglades"
```

Notice that the active field stays the same (City) even though the formula searches the Park field.



After the `find` statement you can check to see if Panorama actually found anything with the `info("found")` function (see "INFO("FOUND")" on page 5381). Here is a procedure that uses this statement and function to locate a park and, if found, make it bold (see "Data Style and Color" on page 1713).

```
local choice,wasField
choice=""
gettext "Which park?",choice
find Park contains choice
if info("found")
   wasField=info("fieldname")
   field Park
   Style "cell bold"
   field (wasField)
else
   message "Sorry, park not found."
endif
```

When you run this procedure it starts by asking you what park you want to highlight (see "Basic Text Entry Dialogs" on page 1607).



If the `find` statement locates the requested information it marks the park name in bold.



If the requested information is not found then an error message is displayed, but the database is not modified.

A Handy Universal Find Procedure

Here is a handy procedure that will search every field in the database. It can be used in any database without modifications. (Note: A variation of this procedure is in the **Search All Fields** wizard.)

```
local whatfor
whatfor=""
gettext "Search for?",whatfor
find exportline() contains whatfor
```

The secret of this procedure is the `exportline(` function (see “EXPORTLINE(” on page 5210). This function takes all the fields in a line, converts them to text if necessary, and then appends them together with tabs in between. The procedure uses this handy capability to search all of the fields in the database at once!

Here is a slightly revised version of this procedure that is even cooler. If it finds what you are looking for it automatically moves to the field containing the data it has located.

```
fileglobal whatfor
whatfor=""
gettext "Search for?",whatfor
find exportline() contains whatfor
if info("found")
   field (array(dbinfo("fields",""),
      arrayelement(exportline(),search(upper(exportline()),upper(whatfor)),¬),¶))
else
   beep
endif
```

When you run this procedure it stops and asks you what you want to look for.



If that word or phrase exists anywhere in the database, it will find it.



It can find a phone number like 882-4336.



Or it can even find a numeric value like 10.00.



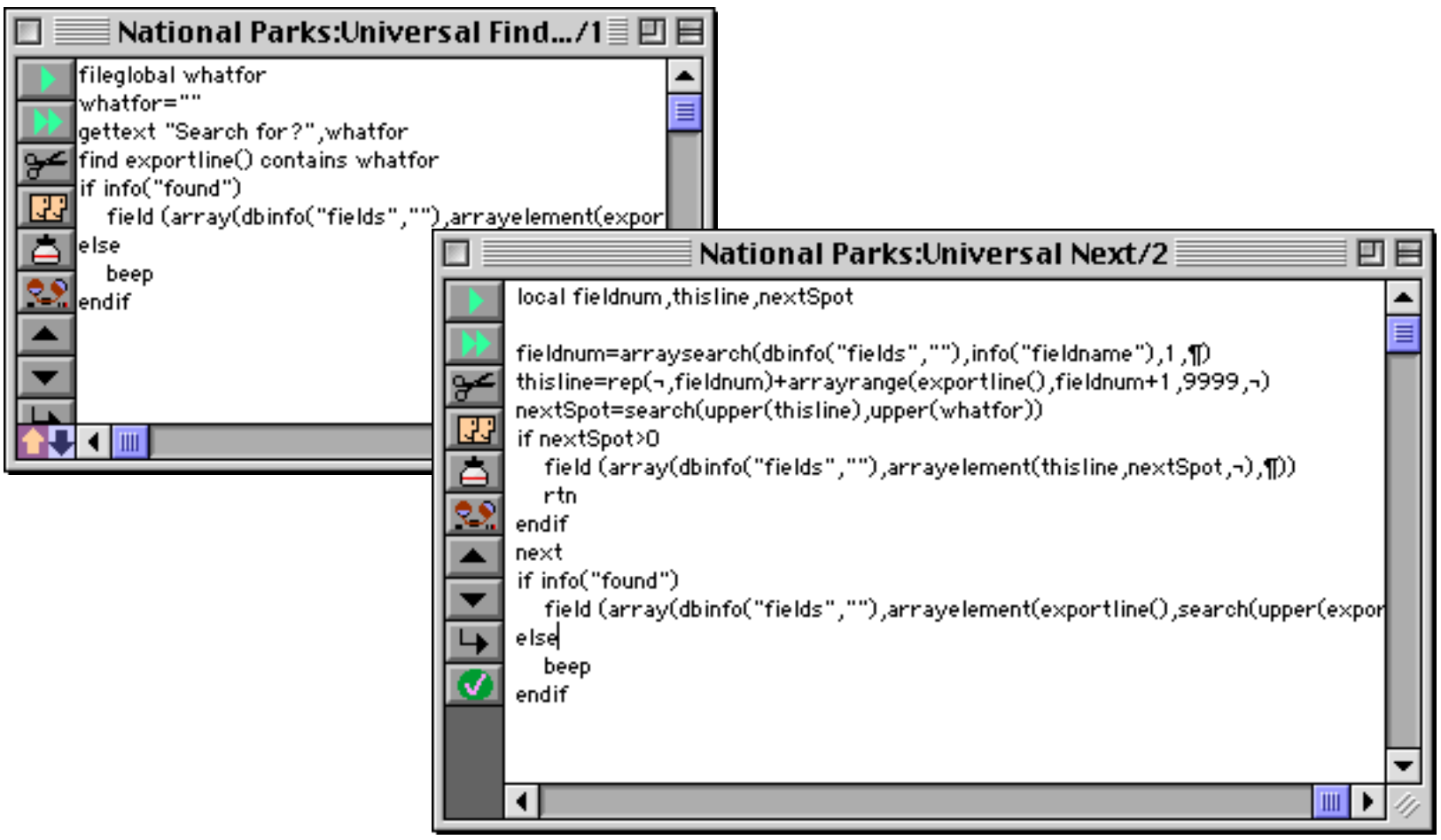
Wherever the data is, this procedure will find it and move right to the spot. See the next section for a “universal find next” procedure to go with this procedure.

Find Next

To find the next match use the `next` statement. This is just like the `find` statement except there is no formula…it re-uses the formula supplied with the `find` statement. You can continue to use the `next` statement over and over again until the `info("found")` function tells you there are no more matches. The procedure below uses the `next` statement to find the next occurrence of the word or phrase — either on the same line or on a different line.

```
local fieldnum,thisline,nextSpot
fieldnum=arraysearch(dbinfo("fields",""),info("fieldname"),1,¶)
thisline=rep(¬,fieldnum)+arrayrange(exportline(),fieldnum+1,9999,¬)
nextSpot=search(upper(thisline),upper(whatfor))
if nextSpot>0
   field (array(dbinfo("fields",""),arrayelement(thisline,nextSpot,¬),¶))
   rtn
endif
next
if info("found")
   field (array(dbinfo("fields",""),
      arrayelement(exportline(),search(upper(exportline()),upper(whatfor)),¬),¶))
else
   beep
endif
```

To illustrate these universal find procedures we've added them to the National Parks database.



To demonstrate these procedures we'll start by running **Universal Find** to search for canyon.



When the **OK** button is pressed the procedure finds the first occurrence of the word canyon.



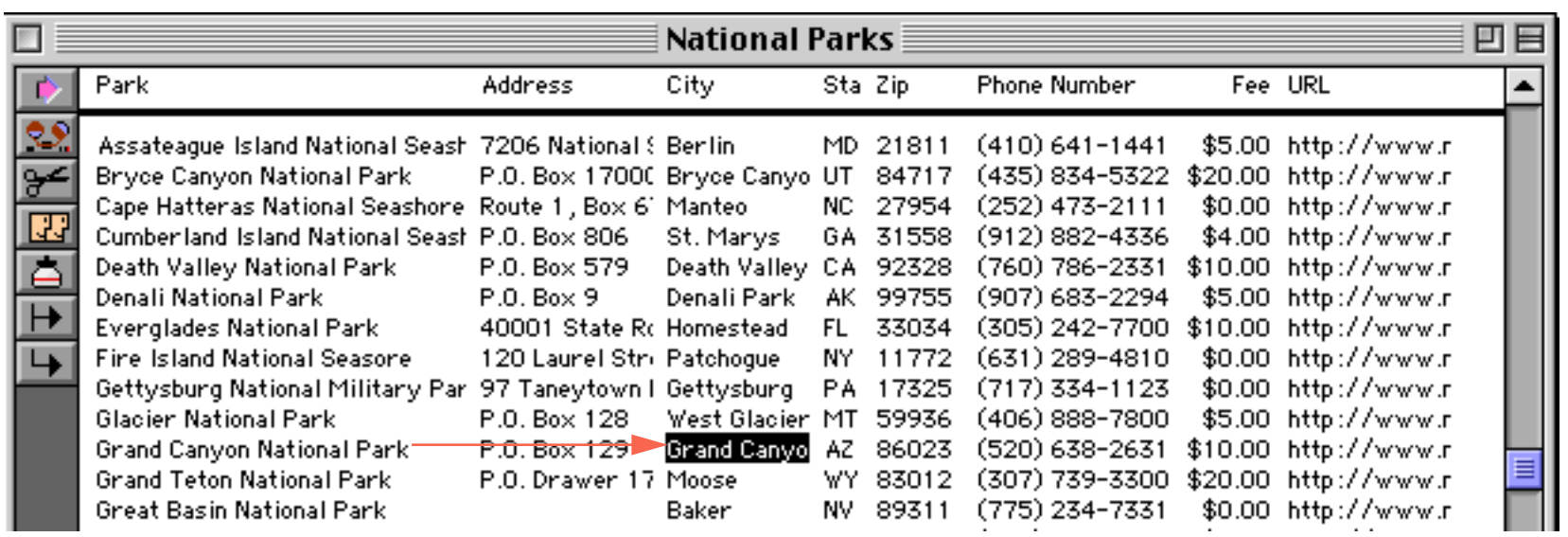
Running the **Universal Next** procedure locates the next occurrence of the word canyon, in the City column of the same record.



There's no more occurrences of the word canyon in this record, so choosing **Universal Next** again jumps down several lines.



Each time you choose **Universal Next** Panorama will jump to the next occurrence of the word canyon, until it finally runs out.



Let's go back and review the original **Universal Find** procedure for a moment.

```
fileglobal whatfor
whatfor=""
gettext "Search for?",whatfor
find exportline() contains whatfor
if info("found")
   field (array(dbinfo("fields",""),
      arrayelement(exportline(),search(upper(exportline()),upper(whatfor)),¬),¶))
else
   beep
endif
```

The first line of this procedure declares the **fileglobal** variable whatfor. It's important that this variable is declared as a **global** or **fileglobal** and not as a **local** variable. Why? Well, remember that Panorama stores the formula used by the `find` statement for use by the `next` statement. In this case the formula is

```
exportline() contains whatfor
```

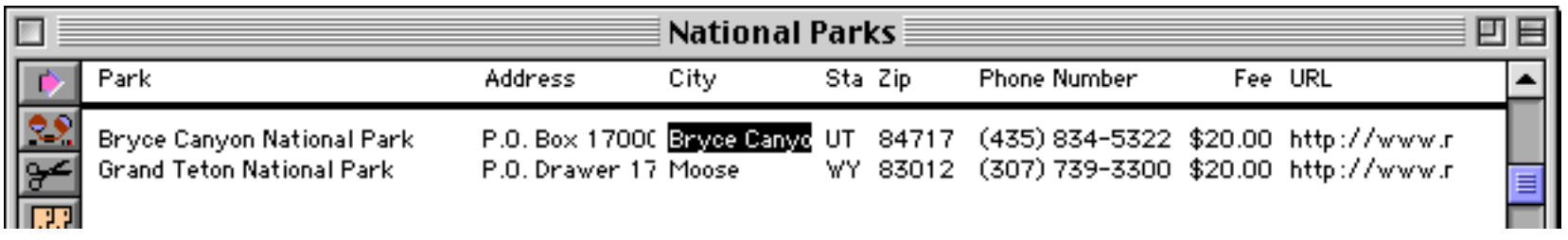
If whatfor is a **local** variable, it will cease to exist as soon as the **Universal Find** procedure is finished (see "The Birth and Death of a Local Variable" on page 1372). Because of this when the `next` statement is executed (or if you manually choose **Next** from the Search menu) an error will occur: field or variable does not exist! The solution is simply to create whatfor as a **global** or **fileglobal** variable so that it will still be hanging around when it becomes time to search for the next occurrence of the word or phrase.

Selecting Information

The `select` statement searches the database, making everything that does not match invisible (see "Finding vs. Selecting" on page 367). This statement has one parameter, a formula. Starting from the top of the selected records, Panorama scans down the database looking for records that makes this formula true (see "True/False Formulas" on page 1265). If the formula is not true the record will be made temporarily invisible. For example, this procedure will select all parks where the admission fee is greater than ten dollars.

```
select Fee>10
```

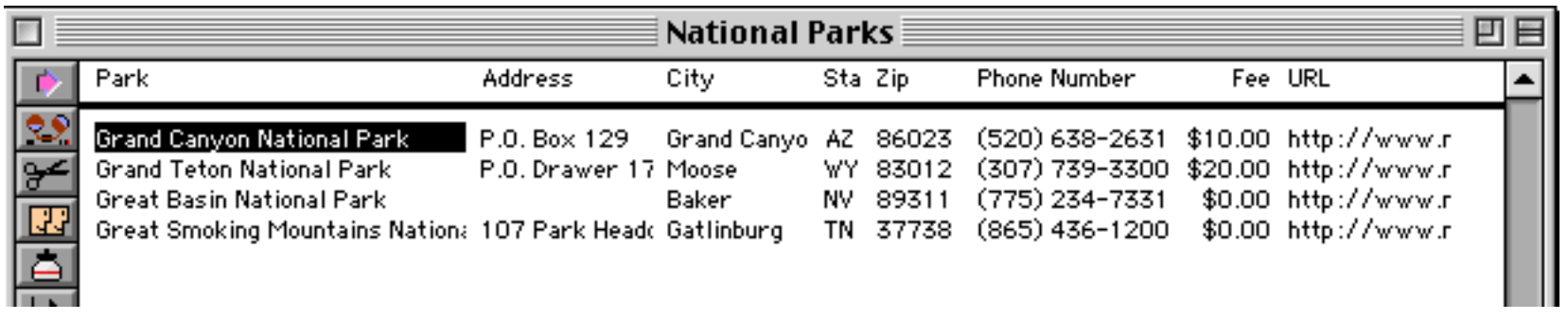
Only two parks in this database fit in this category. All the others are temporarily invisible.



You can construct as complex a formula as you like, combining different elements together with `and` and `or`.

```
select Park contains "Great" or Park contains "Grand"
```

In this case four records match the criteria.



A procedure can find out how many records are selected with the `info("selected")` function. The `info("records")` function returns the total number of records in the database, both visible and invisible. See "Database Information" on page 1319 for more information on these functions.

When a procedure wants to make sure that all records are selected it should use the `selectall` statement. Here is a simple procedure that checks to see if all records are selected, and if not, selects them.

```
if info("selected")<info("records")
   selectall
endif
```

A procedure can use the `selectadditional` statement to add to the current selection (see "SELECTADDITIONAL" on page 5713). The `selectwithin` statement can be used to select a subset of the currently selected subset (see "SELECTWITHIN" on page 5725). The `selectreverse` statement swaps the visible and invisible records (see "SELECTREVERSE" on page 5722).

Handling Empty Selections

What if the `select` statement fails to select any records? Eeek! Panorama always requires that at least one record be selected at all times, it never allows every record in a database to be invisible. If none of the records in the database match the formula, Panorama does nothing. It's as if the `select` never happened. Whatever records were visible before remain visible after. This can be a problem if the following statements are expecting a particular subset of the database to be selected.

Fortunately, Panorama normally handles this situation for you automatically so that your procedures will work correctly. Panorama keeps track of the fact that there should be no records selected, and it will skip any statement that modifies the database, including `formulafill`, `sequence`, `propagate`, `unpropagate`, etc. (basically any statement that corresponds to an item in the Math menu will be skipped). Panorama will continue skipping these statements until it comes to a `selectall` statement or another `select` statement.

Panorama's automatic statement skipping for empty subsets should work fine for most applications. As a procedure programmer, however, you have the choice of overriding this statement skipping and programming your own solution to the empty subset condition.

To test for an empty subset, use the `info("empty")` function. This example calculates the InvoiceAge field only for invoices that have actually been shipped (the ShipDate field is not empty) and that are not paid yet. An error message will appear if there are no outstanding invoices.

```
select sizeof(ShipDate)≠0 and Balance>0
if info("empty")
   message "No outstanding invoices!"
else
   field InvoiceAge
   formulafill today()-ShipDate
   selectall
endif
```

Remember, this logic is only necessary if you want to perform some special handling of empty subsets. Normally, Panorama will handle the empty subset just fine on its own by skipping the statements until the selected subset changes.

Selecting Duplicates

The `selectduplicates` statement may be used to locate and select duplicate entries in a database. The statement has one parameter, a formula that determines what fields to check for duplicates. If this parameter is empty text (`""`) then the current field is assumed. This statement must be combined with the `sortup` statement. For example, to locate duplicate check number entries within a database you would use this procedure.

```
field "Check Number"
sortup
selectduplicates ""
```

If you supply a formula you can check for duplicates across multiple fields, or using only part of a field, or both, as in the example below. This procedure will check for duplicates in the same zip code, with the same last name and first initial. The formula uses a text funnel to extract the initial from the first name. See “Taking Strings Apart (Text Funnels)” on page 1214 if you are not familiar with text funnels.

```
field Zip
sortup
field "Last Name"
sortupwithin
field "First Name"
sortupwithin
selectduplicates Zip+«Last Name»+«First Name»[1,1]
```

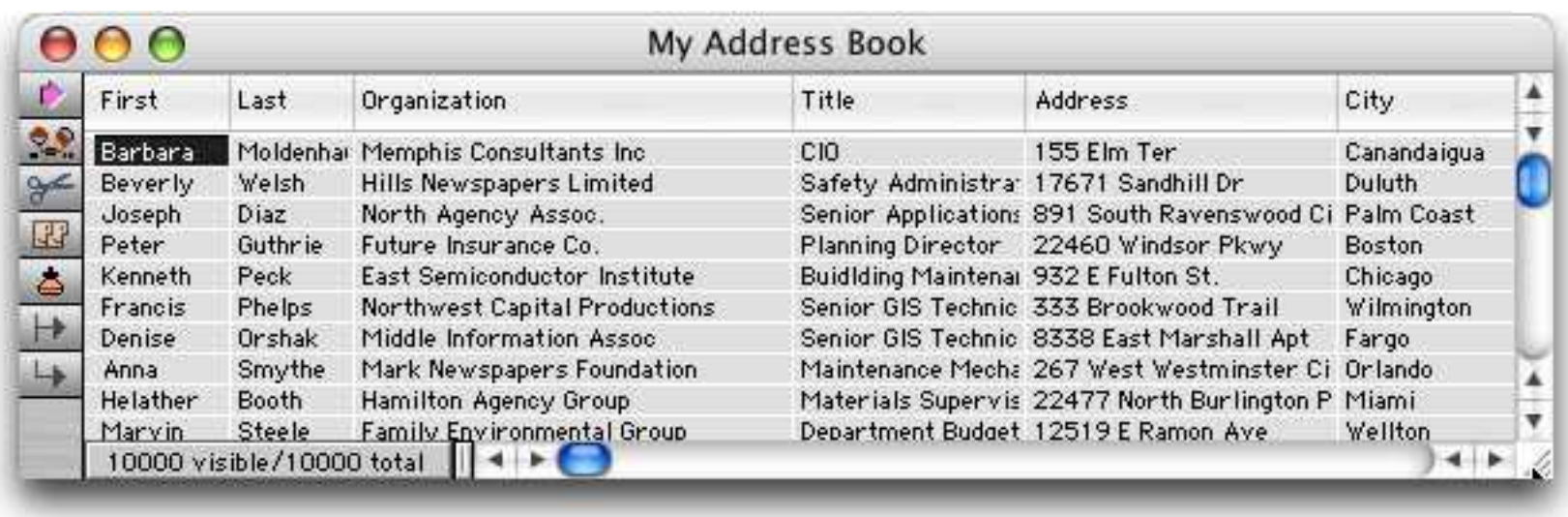
The database must be sorted so that the duplicates you want to find will be consecutively located within the database (see See “Sorting” on page 1676).

Live Clairvoyance™

Panorama V introduced a new search method — **Live Clairvoyance™**. Live Clairvoyance allows you to perform "live" searches on any Panorama database. The search results are updated dynamically as you type, allowing you to "hone in" on just the information you are looking for. The search may include multiple fields or even all fields in the database being searched. (If you've used the search box in iTunes you'll find Live Clairvoyance familiar, although the underlying technology is not related.)

The easiest way to use Live Clairvoyance is with the wizard that comes with Panorama (see “Live Clairvoyance™” on page 4). This wizard performs the neat trick of allowing you to use Live Clairvoyance without doing any programming, and in fact without making any modification at all to your databases. However, for some applications you may want to build Live Clairvoyance into your forms. This allows you to customize it exactly for your needs, and to integrate it with other database operations.

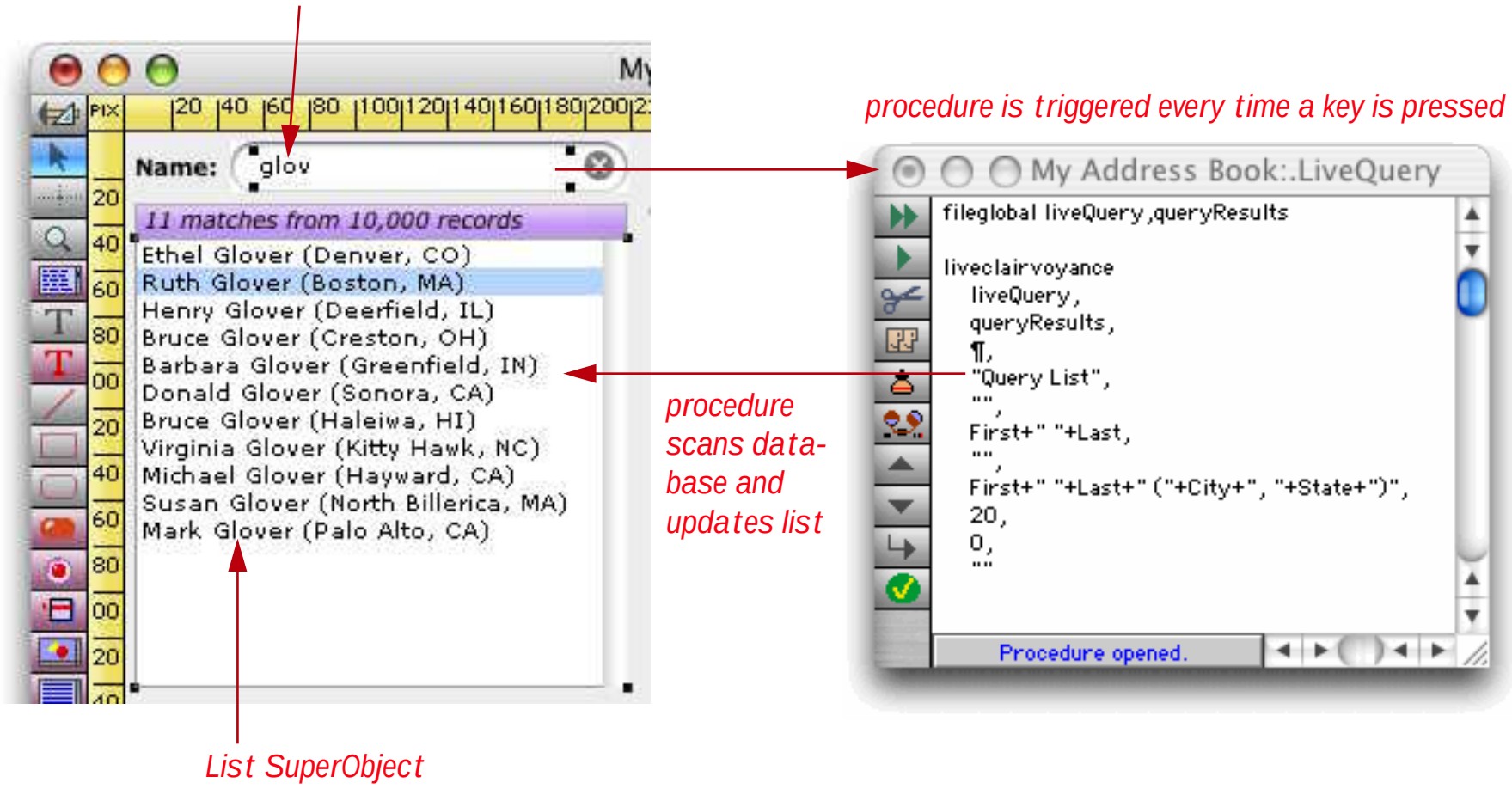
To illustrate how Live Clairvoyance can be used we’ll use a database called My Address Book. You’ll find a copy of this database in the **Guided Tour** submenu of the **Wizard** menu.



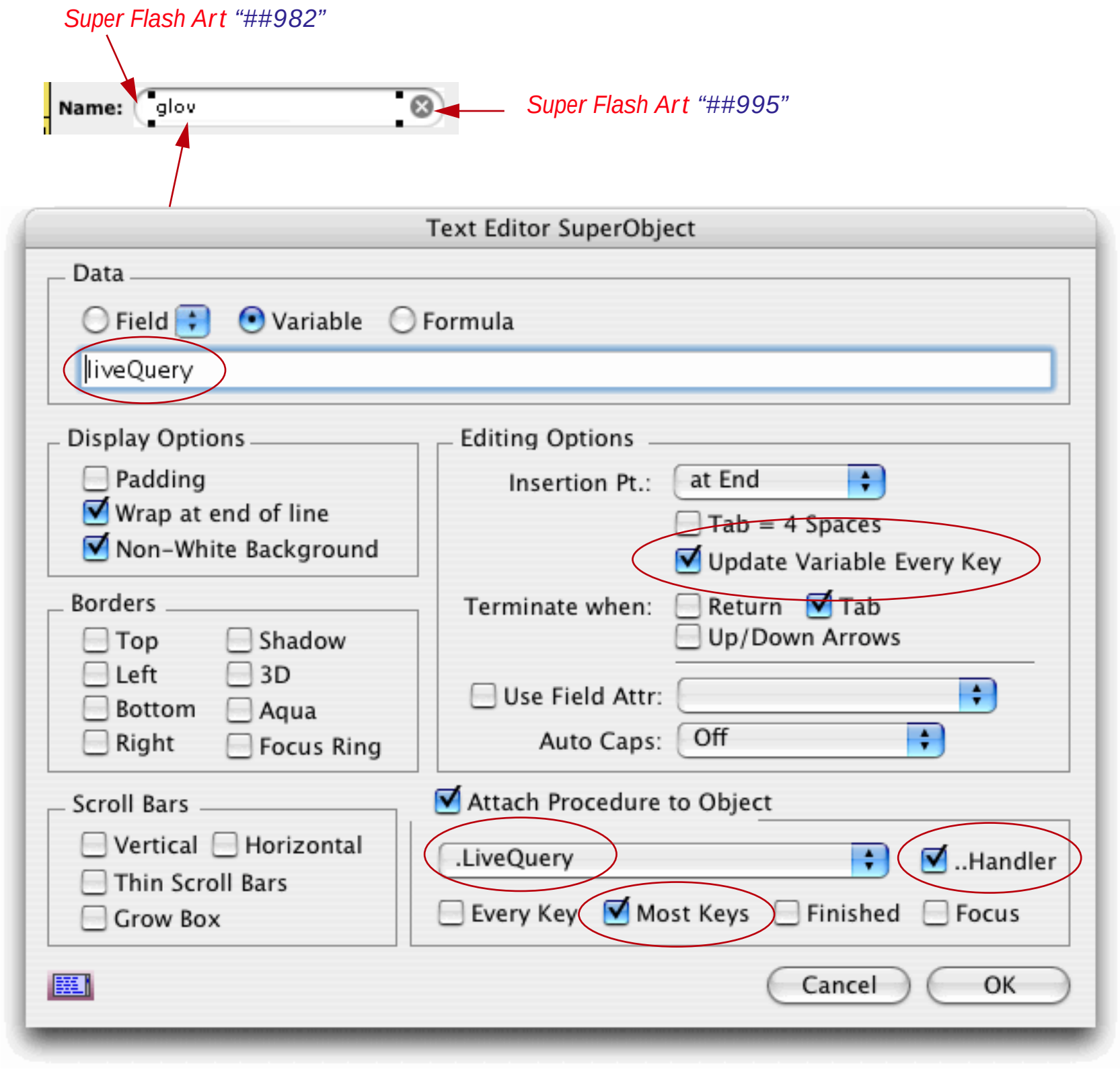
Here is the form that has been set up to use Live Clairvoyance.



This form has many elements, but the Live Clairvoyance portion consists of only three — a Text Editor Super-Object to type in the search word or phrase, a List SuperObject to display the resulting list, and a procedure to perform the actual database scan.



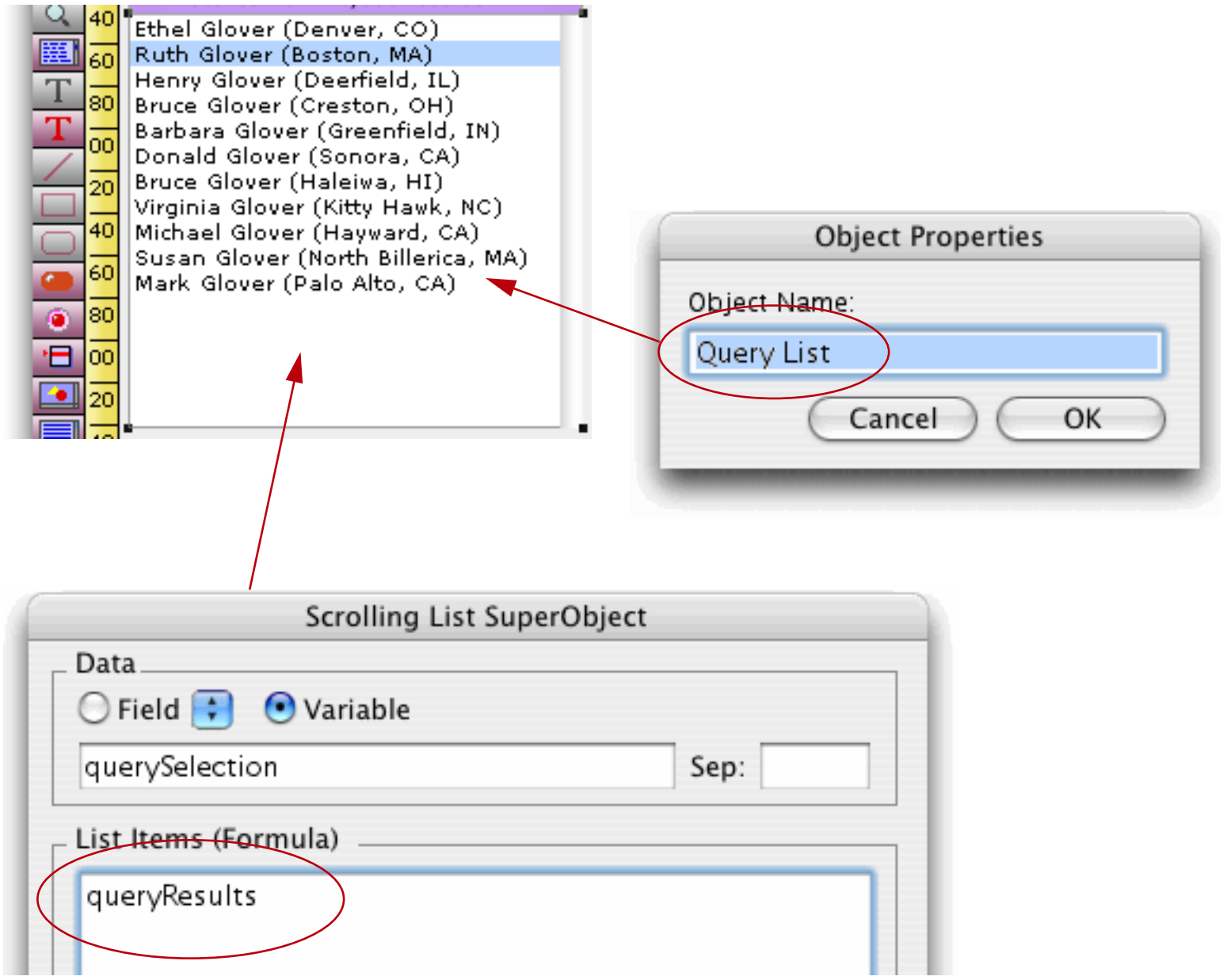
Live Clairvoyance starts with typing in one or more characters to search for. This is done with a Text Editor SuperObject. This illustrations shows the options that must be set for this object. (It also shows the formulas for two Super Flash Art objects to display the background and cancel icon for the search area. See the **Dialogs & Icons** wizard for more information on these images.)



The Data option is set to a variable named liveQuery. You can use any variable name you like, but you'll need to use the same name when you write the procedure that is triggered by this object.

In the Editing Options the Update Variable Every Key option must be enabled. This allows the procedure to see what you have typed. Moving on to the procedure, in this case it is named .LiveQuery. You can use any name you like as long as it is selected in this pop-up menu. You also want to check the Most Keys and ..Handler options.

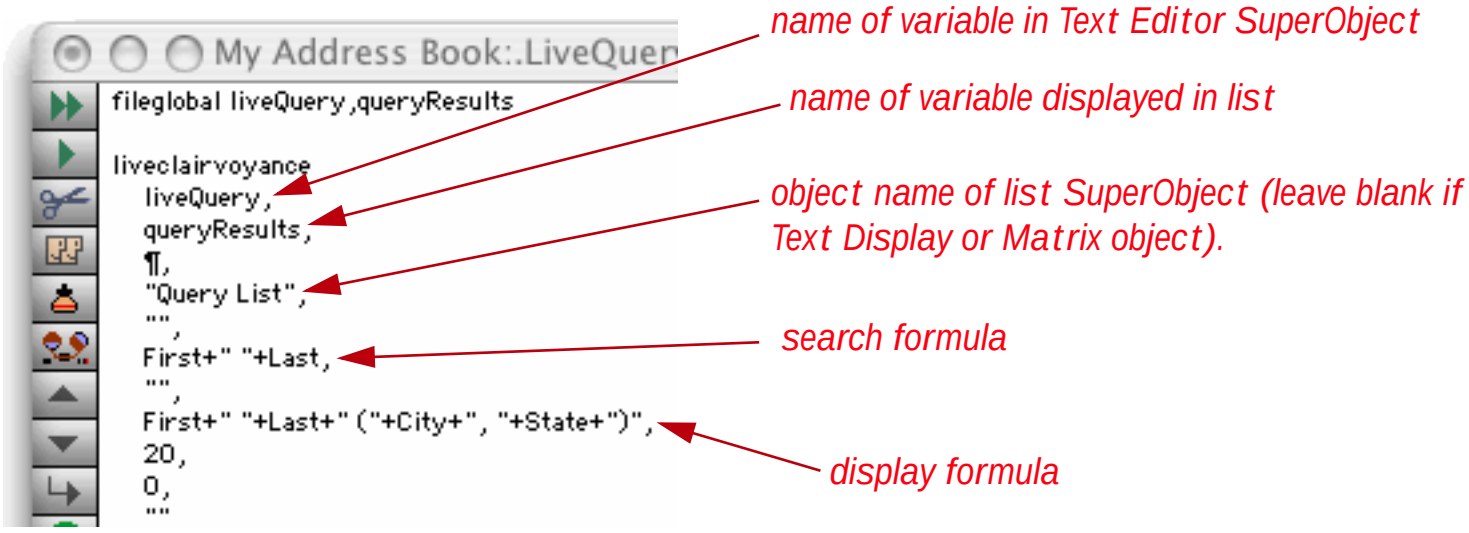
The Live Clairvoyance output can be displayed with a Text Display SuperObject, a Matrix SuperObject, or a List SuperObject, as in this case.



To set up this List object you'll need to set up two dialogs. When first setting up the object you'll need to set the List Items (Formula) option to the name of a variable — in this case we picked queryResults but again you can use any name you want. (In this example the Data option was also set to a variable named querySelection, but this variable is not directly involved in the operation of Live Clairvoyance.)

When using a List Object you must also set the name of the object. In this case we set the name to Query List, but again you can choose any name you want as long as you remember what the name is so that you can use it in the procedure. It's not necessary to set the object name when displaying the result in a Text Display or Matrix Super Object.

Once the form objects are set up there's only one remaining piece - the procedure.



The procedure only has two statements. The first statement defines the two variables used, liveQuery for the search text and queryResults for the output list.

The second statement, liveclairvoyance, does all the heavy lifting. This statement has eleven parameters.

```
LIVECLAIRVOYANCE INPUT,OUTPUTLIST,SEPARATOR,LISTOBJECT,DATABASE,QUERY,COMPARE,TEMPLATE,
   TICKS,MAX,OPTIONS
```

The Input and OutputList parameter must be set to the variables defined in the Text Editor SuperObject and List SuperObject. The ListObject parameter must be set to the name of the list object (if a list object is being used, otherwise set this parameter to "".

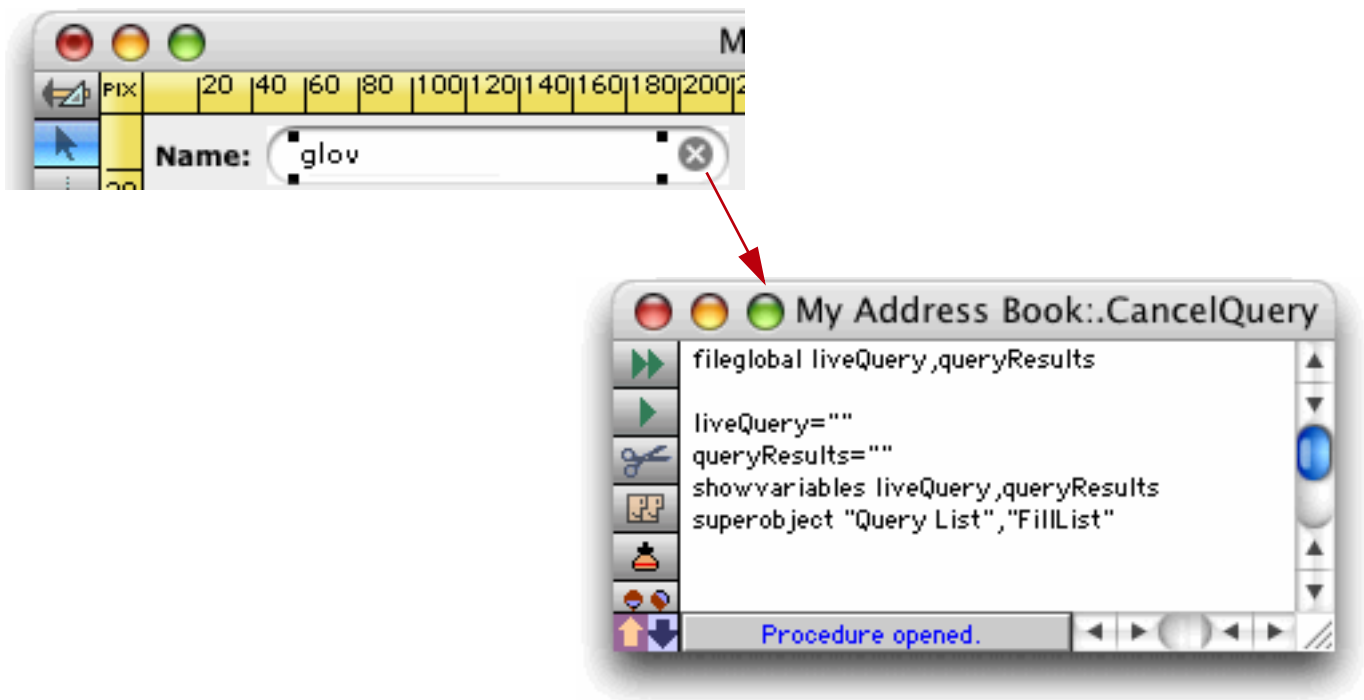
The table below gives detailed explanations for each parameter.

Parameter	Example	Description
Input	liveQuery	This parameter must contain the data to be searched for, usually in a variable.
OutputList	queryResults	This parameter is the name of a variable that will contain the final output array of matching entries.
Separator	¶	The separator character for the output array.
ListObject	"Query List"	If non-blank, this parameter must specify the name of a List SuperObject to be updated each time a search is performed.
Database	""	Name of the database being searched, or blank for the current database.

Parameter	Example	Description
Query	First+" "+Last	Formula used for searching. Each time a search is triggered (usually by pressing a key) the statement will scan the database and check to see if the search text matches this formula. For example, the example formula to the left would find text located in the first and last name fields. This formula must include all of the fields in the database that you want to search (keep in mind, however, that searching more fields may affect performance in large databases).
Compare	""	This parameter specifies the comparison operator to use. The choices are contains, beginswith, endswith, soundslike, match and = (you must put quotes around each of these. If you leave this empty it will default to contains, so in the example database the search performed will be `First+" "+Last contains liveQuery`
Template	First+" "+Last+" ("+City+", "+State+")"	This parameter specifies how the fields in the database will be displayed in the output list. In this example the output will contain the first and last names, followed by the city and state within parentheses.
Ticks	20	This parameter specifies how often the display will be updated during the search. The value is specified in ticks (1 tick is equal to 1/60th of a second). A value of 20 will cause the display to update 3 times per second. Up to a point, lower values will make the search “look” faster because partial results appear sooner. If this value is set to zero then no partial results will be displayed, and nothing will appear until the entire search is complete.
Max	0	This is the maximum number of elements that can appear in the output array. Use 0 for no limit.
Options	" "	This parameter can contain one or more options to modify the search. If it contains "sort" the output array will be sorted alphabetically. If it contains "selected" only selected records will be searched, otherwise all records will be searched (even if some are currently not selected).

Adding a Cancel Search Button

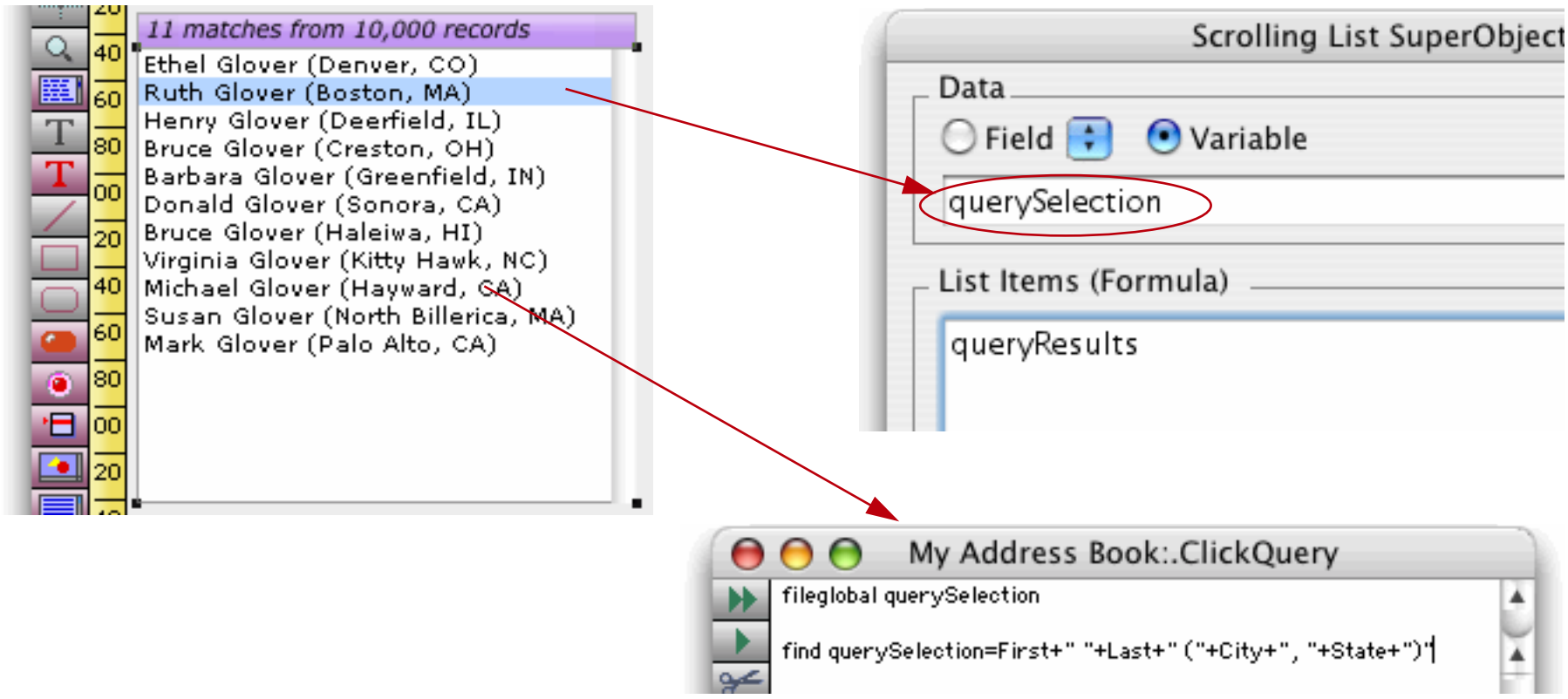
You may want to add a cancel search button. This is easy to do, as shown below.



The procedure simply clears the variables for editing the search text (in this example liveQuery) and displaying the search results (in this example queryResults), and then displays these variables with the `showvariables` statement. The final line is only required if you are displaying the results with a List SuperObject — it forces the list to display the new (blank) information.

Clicking on the Live Clairvoyance List Object

This form has been set up so that when you click on a line in the list the database jumps to the corresponding record. This allows you to display and/or edit the data. When you click on a line in the Scrolling List SuperObject it will place the value of the line you click on into a variable named querySelection and then trigger a procedure named .ClickQuery (the pop-up menu for setting this up is not shown).



The procedure simply uses a `find` statement to locate the corresponding record. If you flip back a couple of pages, you'll see that the right side of the equals sign (`First+" "+Last+" ("+City+", "+State+")"`) is the same as the Template formula parameter of the `LiveClairvoyance` statement. As long as these two formulas match this procedure will be able to find the record that matches the item you clicked on.

Summaries and Outlines

Summarizing a database is a three step process — group, calculate and outline (see "3-Step Summarizing" on page 393). The table below lists the statements that can be used to automate this process. Each statement corresponds to a menu command or tool. In fact, the easiest way to write a procedure to summarize the database is to simply record it (see "Creating a Procedure with the Recorder" on page 1348 and "Adding a Recording to an Existing Procedure" on page 1360).

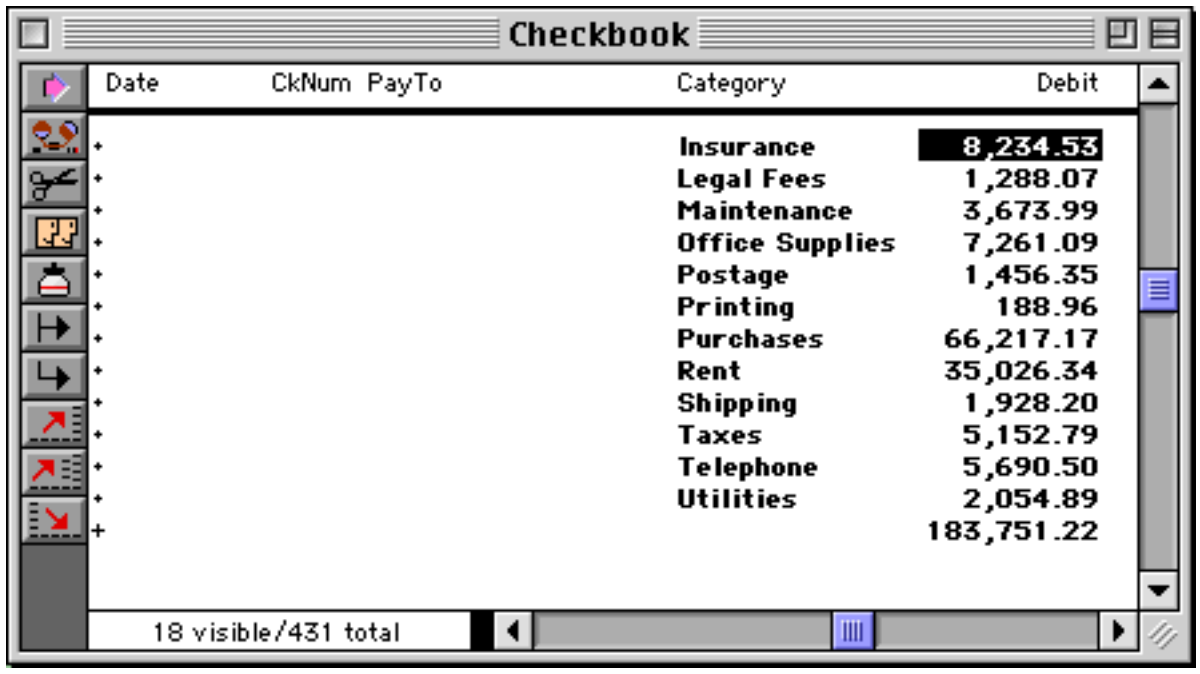
Step	Statement	Ref	Description
Group	**groupup**	Page 5337	This statement groups the database by the current field (see "Moving Left and Right" on page 1640). The database is sorted in ascending order (A's to Z's) and a summary recorded is added at each place where the value in the field changes. If the current field is a date field you must add **by day**, **by week**, **by month**, **by quarter** or **by year** after the **groupup** statement.
	groupdown	Page 5336	This statement works exactly like **groupup**, but sorts the database in descending order (Z's to A's).
	group	Page 5334	This statement groups the database without sorting it.
	groupbycolor	Page 5335	This statement groups the database by color.
Calculate	**total**	Page 5865	This statement calculates totals and subtotals in the current field (see "Moving Left and Right" on page 1640).
	average	Page 5069	This statement calculates averages and subaverages in the current field.
	count	Page 5131	This statement counts non-empty values in the current field.
	minimum	Page 5531	This statement calculates minimum values in the current field.
	maximum	Page 5524	This statement calculates maximum values in the current field.
Outline	**outlinelevel**	Page 5586	This statement expands or collapses the database to show a specific level of detail. The statement has one parameter, which may be "Data" to show all of the detail, or a summary level from "1" to "7" to show a specific outline level.
	removesummaries	Page 5657	This statement removes some or all of the summary records in the database. The statement has one parameter, which specifies the level of summaries to be removed (from "1" to "7"). To make sure that all summary records are removed use "7".
	removedetail	Page 5656	This statement removes the data records from the database, leaving only the summary records. It can also remove lower level summary records. The remaining summary records are dropped in level (the lowest remaining summary records become data records). The statement has one parameter, the lowest level of summary record to be retained (from "1" to "7"). To remove just the data records and leave all summary records use "1".
	collapse	Page 5115	This statement hides any detail records associated with the current summary record.
	expand	Page 5203	This statement expands the next level of detail associated with the current summary record.
	expandall	Page 5205	This statement expands all of the detail associated with the current summary record, right down to the data records.
	info("summary")	Page 5428	This function returns the summary record of the current record, from 0 (data record) to 7 (highest level summary).
	info("expandable")	Page 5374	This function checks to see if the current record is an expandable summary record. It returns false if this is a data record or if this record is already expanded.

Summary/Outline Examples

To summarize a database using a procedure you simply pick one or more statements from each of the three steps — group, calculate and outline. This very basic procedure summaries a checkbook by category and displays just the totals for each category.

```
field Category      /* STEP 1 - GROUP */
groupup
field Debit         /* STEP 2 - CALCULATE */
total
outlinelevel "1"    /* STEP 3 - OUTLINE */
```

The end result of running this procedure will look something like this.



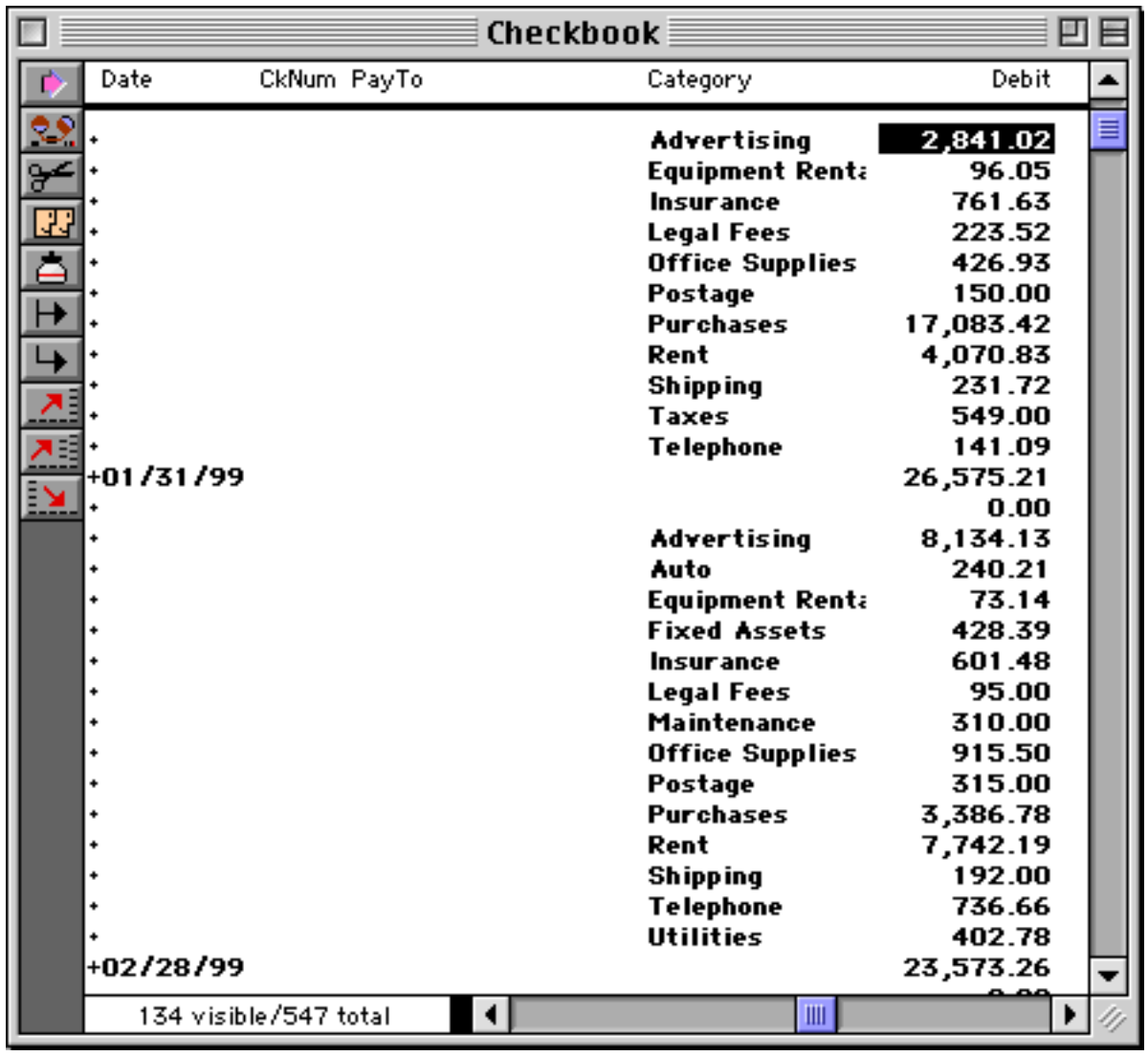
To remove the summaries and get back to the original data we can use a simple one line procedure.

```
removesummaries "7"
```

This slightly more complex procedure will group the database by month and by category within each month.

```
field Date          /* STEP 1 - GROUP */
groupup by month
field Category
groupup
field Debit         /* STEP 2 - CALCULATE */
total
outlinelevel "1"    /* STEP 3 - OUTLINE */
```

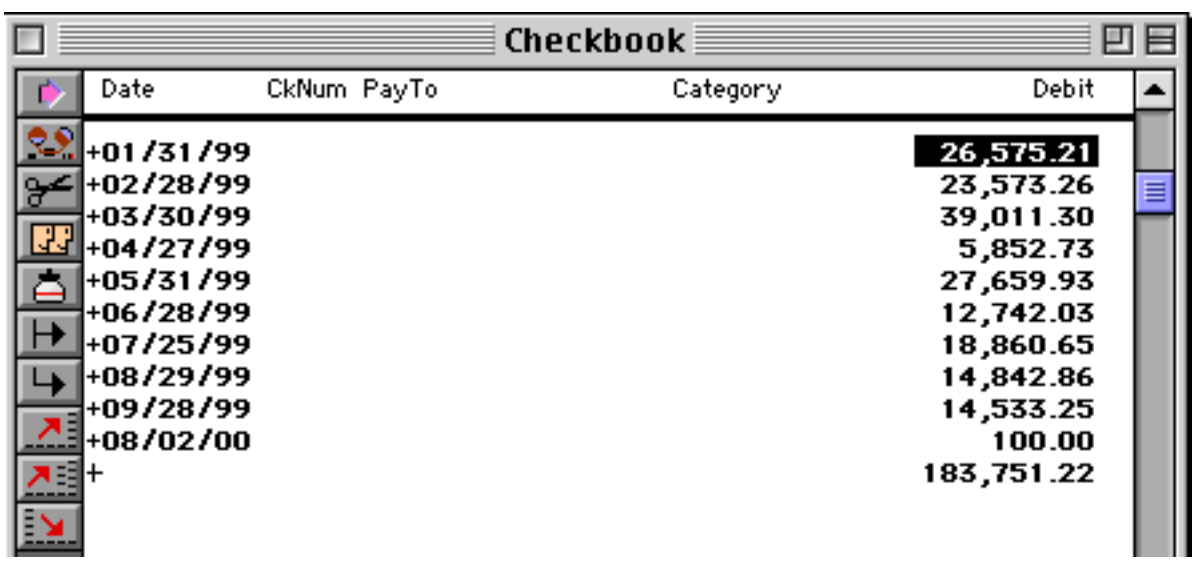
Here is the result of running this procedure.



If the last line of the procedure had been

```
outlinelevel "2"
```

Then the final result would have looked like this.



As the procedure runs it flashes the window over and over again. To eliminate this you can use the noshow and endnoshow statements (see “Suppressing Display of Text and Graphics” on page 1427).

```
noshow
   field Date          /* STEP 1 - GROUP */
   groupup by month
   field Category
   groupup
   field Debit         /* STEP 2 - CALCULATE */
   total
   outlinelevel "1"    /* STEP 3 - OUTLINE */
   showpage
endnoshow
```

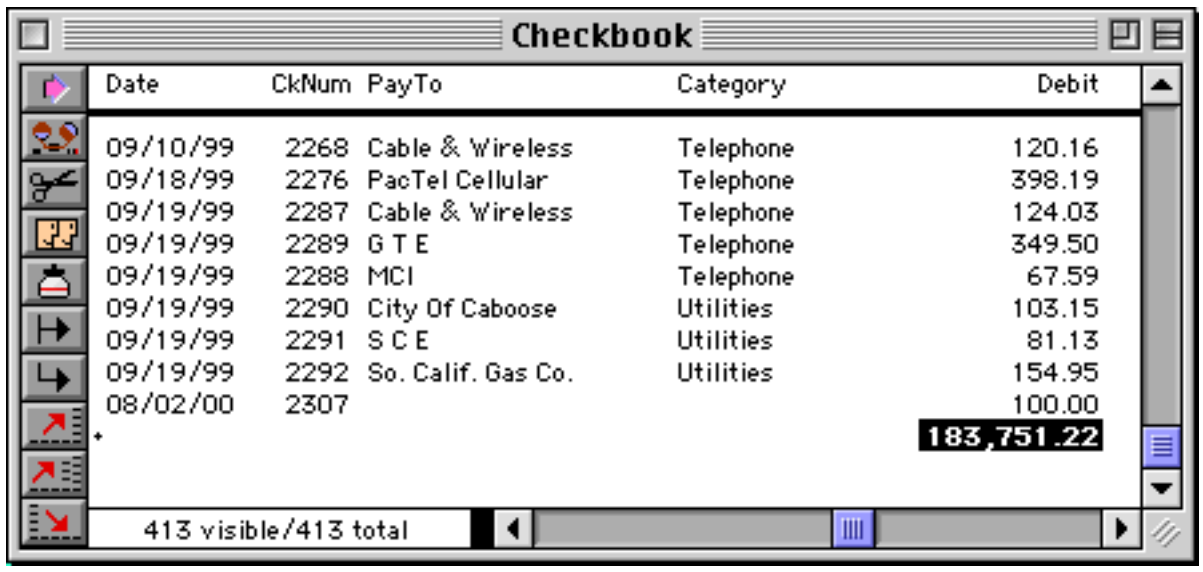
This revised procedure will only re-display the window once, at the very end.

Calculating Grand Totals

There are two methods for calculating a grand total without subtotals. The first is to simply use the total statement. This adds a single summary record at the bottom of the database and calculates the total. (You can also use the average, count, minimum and maximum statements this way.)

```
field Debit
total
```

This procedure produces a single summary record with the total, like this.



Another method for calculating a grand total is to use the formulasum statement (see “FORMULASUM” on page 5278). This statement has two parameters:

```
formulasum result,formula
```

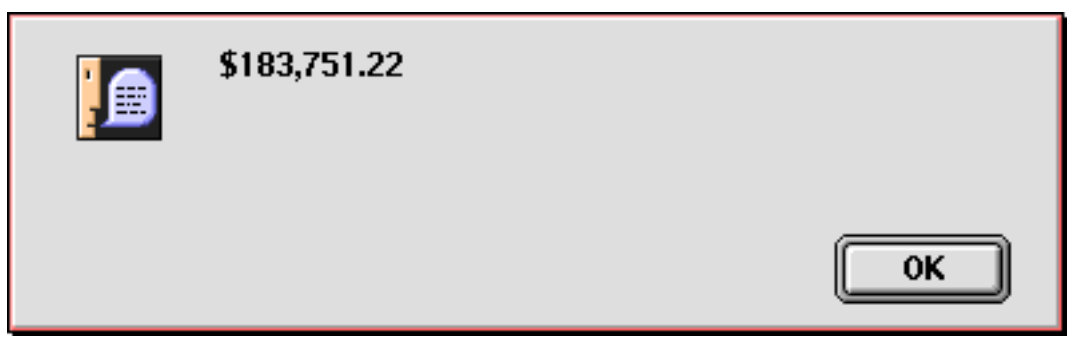
The result parameter must be a field or variable. The final total will be stored in here.

The formula parameter is a formula that will be evaluated for every selected record. Starting from the top of the database, Panorama will visit each record and calculate the result of the formula. As it goes it keeps a running total of the results. The final result is the sum of all of the individual results for each selected record.

The procedure below calculates the grand total for the checkbook database and displays it. The database itself is not modified (no summary record is added).

```
local total
formulasum total,Debit
message pattern(total,"$#,.##")
```

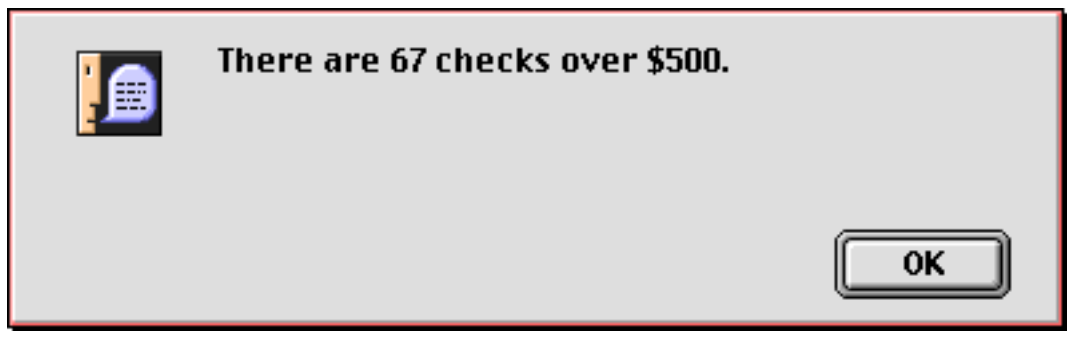
The procedure will display the total like this.



By changing the formula you can calculate different sums. This procedure calculates the number of checks that are over $500.

```
local count
formulasum count,?(Debit>500,1,0)
message "There are "+str(count)+" checks over $500."
```

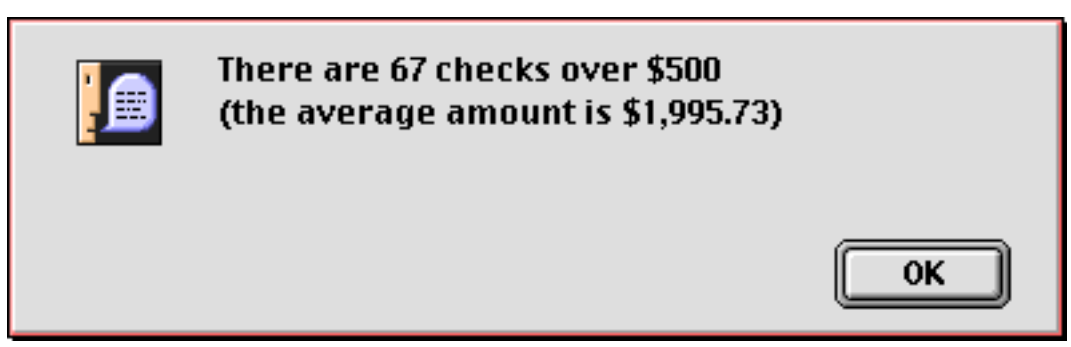
Here is the result.



There's no reason you can't use `formulasum` more than once, like this.

```
local count,total
formulasum count,?(Debit>500,1,0)
formulasum total,?(Debit>500,Debit,0)
message "There are "+str(count)+
   " checks over $500"+¶+"(the average amount is "+
   pattern(total/count,"$#.,##")+")"
```

This procedure displays both the number of checks over $500 and the average value of these checks.



Running Total

The `runningtotal` statement performs a special computation (see “RUNNINGTOTAL” on page 5687). Unlike the other summary calculations, this statement modifies every data cell in the currently active field, not just the summary records. Like the `total` statement, `runningtotal` starts at the top of the database and adds up each data cell as it moves down the column. The `runningtotal` statement, however, replaces each data cell with the current total. The result is a field which contains the cumulative total at each point in the database. Here is a procedure that uses `runningtotal` to calculate a checkbook's balance after each transaction.

```
noshow
   field Balance
   formulafill Credit-Debit
   runningtotal
   showcolumns Balance
endnoshow
```

To see what the result of this procedure looks like go to “Using Running Total to Balance a Checkbook” on page 404. By the way, this procedure uses the `noshow`, `showcolumns` and `endnoshow` statements to make sure that the window only get's redisplayed once. These statements are not necessary for the procedure to operate, but do make it look a little bit cleaner as it runs (“Suppressing Display of Text and Graphics” on page 1427).

Running Difference

The `runningdifference` statement is the opposite of `runningtotal`. This statement fills each data cell with the difference between the cell and the cell above it. Use the `runningdifference` statement when you want to calculate the spread or interval between consecutive values, for example odometer readings or dates. Here is a procedure that uses `runningdifference` to calculate gas mileage per gallon for each fill-up.

```
field Range
formulafill Odometer
runningdifference
field MPG
formulafill Range/Gallons
```

Go to “Using Running Difference to Calculate Gas Mileage” on page 407 to see what the result of this procedure looks like.

Transforming Big Chunks of Data

The **Math** menu contains ten commands for transforming an entire field of data at once (see “Data Processing” on page 451). All of these statements operated on the current field, so you’ll need to position the current field before you use them (see “Moving Left and Right” on page 1640). In addition these commands only operate on selected data, so you must make sure that the proper data is selected before the statement is used (see “Selecting Information” on page 1682).

The workhorse of this group is `formulafill`, which you will probably use more than all the others combined.

Category	Statement	Ref	Description
Fills	**formulafill**	Page 5274	This statement fills all of the selected cells in the current field with a formula. Panorama starts at the top of the database and works it’s way down, calculating a the formula result over and over again for each record. See “Filling a Field with a Formula” on page 453.
	fill	Page 5243	This statement also fills all of the selected cells in the current field with a formula. Unlike **formulafill**, however, this statement calculates the formula only once, before it starts scanning the database. If you are filling the field with a constant value like "US Mail" this statement will be slightly faster than the **formulafill** statement. See “Filling a Field with a Fixed Value” on page 452.
	emptyfill	Page 5192	This statement fills all empty cells in a field with a formula. Like the **fill** statement, the formula is only calculated once before Panorama begins scanning the database. See “Filling Empty Cells” on page 463.
	sequence	Page 5728	This statement fills a numeric field with an increasing or decreasing numeric sequence, for example 1, 2, 3 or 100, 99, 98. See “Automatic Numbering” on page 464.
	change	Page 5095	This statement scans the current field searching for a word or phrase. It replaces every occurrence of the word or phrase it finds with another word or phrase. See “Change (Find and Replace)” on page 472.
Propagates	**propagate**	Page 5619	This statement copies the values in the current field into the empty cells (if any) below. See “Propagate” on page 465.
	unpropagate	Page 5874	This statement is the opposite of **propagate**. It scans the database from top to bottom. If it finds the same value two or more times in a row it erases all but the topmost duplicate value. See “UnPropagate” on page 469.
	propagateup	Page 5620	This statement copies the values in the current field into the empty cells (if any) above. See “Propagate” on page 465.
	unpropagateup	Page 5875	This statement is the opposite of **unpropagate**. It scans the database from bottom to top. If it finds the same value two or more times in a row it erases all but the bottommost duplicate value. See “UnPropagate” on page 469.
Appearance	**stylecolor**	Page 5810	This statement changes the style (bold, italic, etc.) and color (red, green, blue, etc.) of one or more data cells. See “Data Style and Color” on page 474.

Making Transformations Even Faster

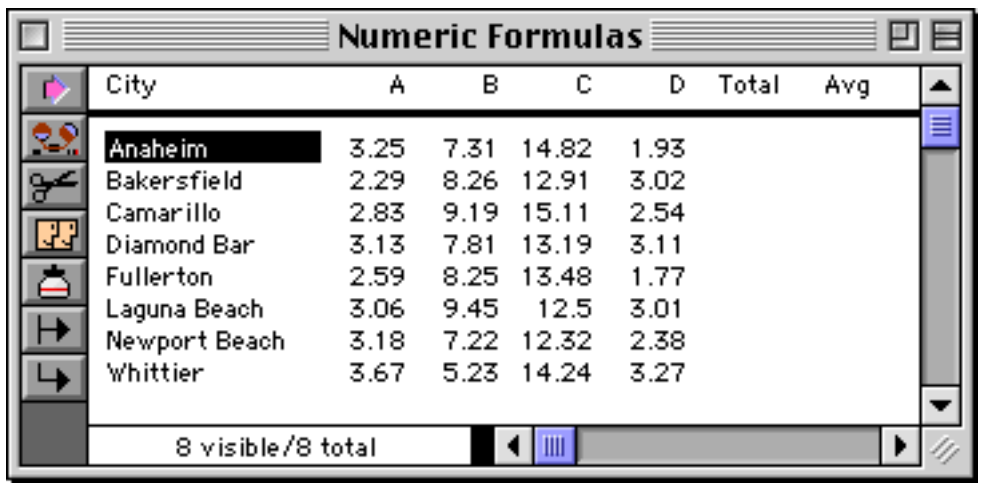
Panorama's transformation (`formulafill`, `propagate`, etc.) statements operate very quickly, even when used with large databases. However, the `noundo` statement can make these operations even faster (see "NOUNDO" on page 5547). This statement disables Panorama's undo feature. Since the transformation doesn't need to worry about undo it can run slightly faster. Here is an example of how to use `noundo` in a procedure with the `formulafill` statement.

```
noundo
field Total
formulafill A+B+C+D
field Avg
formulafill (A+B+C+D)/4
```

The benefit of the `noundo` statement will not be noticeable on smaller databases, but becomes more pronounced the larger the database gets.

Numeric Calculations with FormulaFill

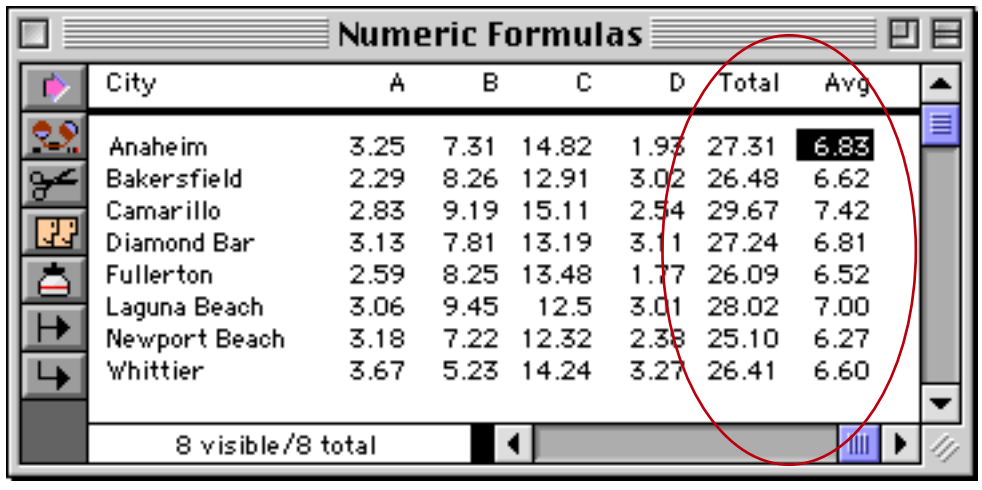
On numeric fields the `formulafill` statement can be used to calculate totals, averages, discounts, percentages, etc. The statement must be followed by a formula to calculate (see "Formulas" on page 1165). To illustrate this statement we'll use this database.



This procedure will calculate all the values in the total and average fields.

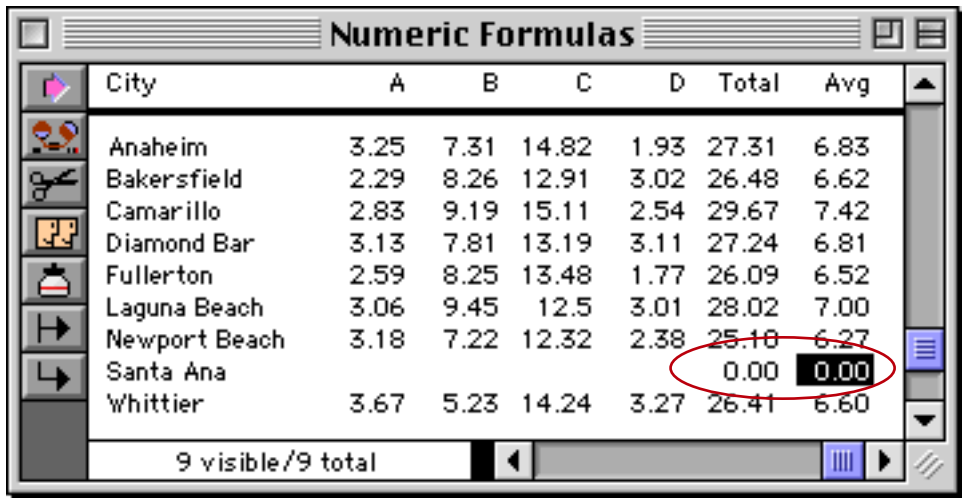
```
field Total
formulafill A+B+C+D
field Avg
formulafill (A+B+C+D)/4
```

Here's the finished result.



Suppressing Zero's

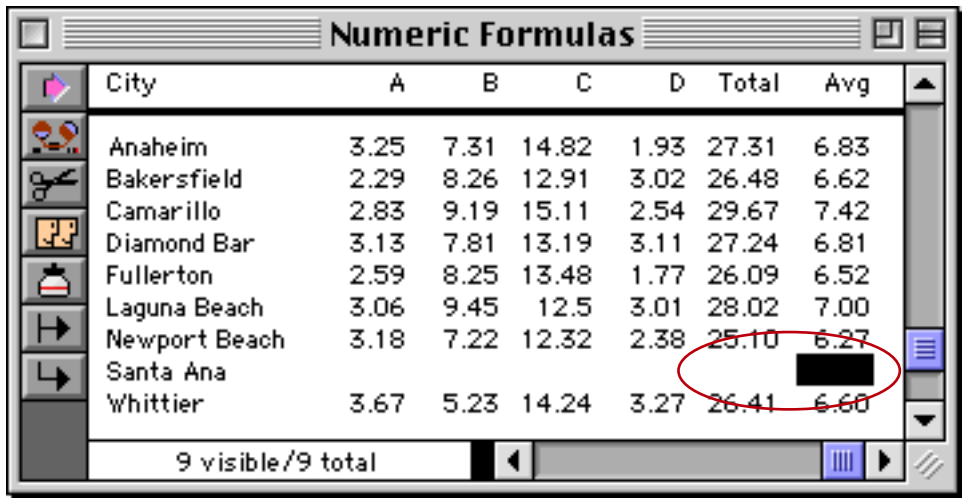
If a new record with incomplete information is added to the database the empty values are treated as zeroes, as shown here.



Sometimes you may want a zero result to be suppressed, leaving the cell blank. To do this use the `zeroblank(` function, like this (see "ZEROBLANK(" on page 5915).

```
field Total
formulafill zeroblank(A+B+C+D)
field Avg
formulafill zeroblank((A+B+C+D)/4)
```

When this procedure is run any zero values are treated as blanks.

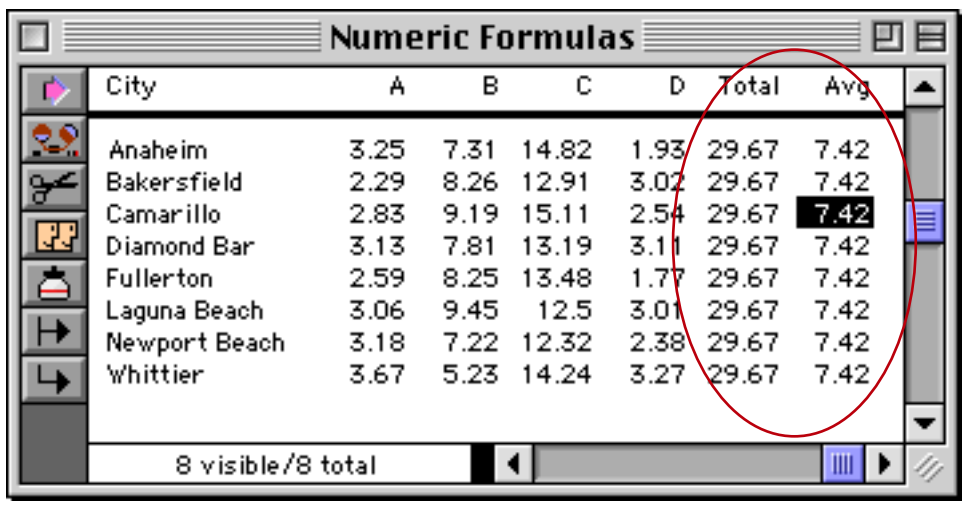


Fill vs. FormulaFill

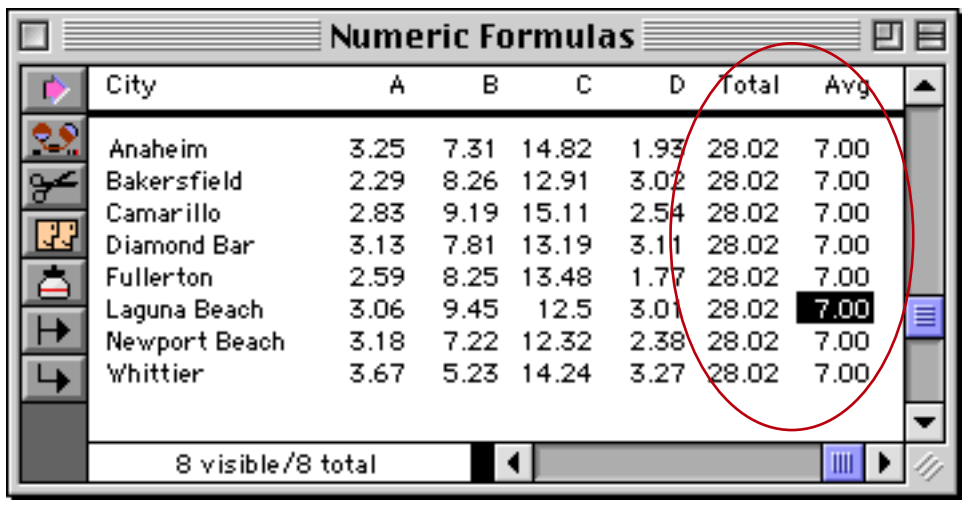
Like the `formulafill` statement, the `fill` statement also takes a formula and fills all the cells in the current column with the result. However, there is a big difference. The `formulafill` statement calculates the formula over and over again, producing a separate result for every record. The `fill` statement only calculates the formula once, before it starts. It then fills all the cells with the same value. To illustrate we'll use a modified version of the procedure from the last section.

```
field Total
fill zeroblank(A+B+C+D)
field Avg
fill zeroblank((A+B+C+D)/4)
```

The result of this procedure depends on what record is active. In this case every cell is filled with the total and average for Camarillo.



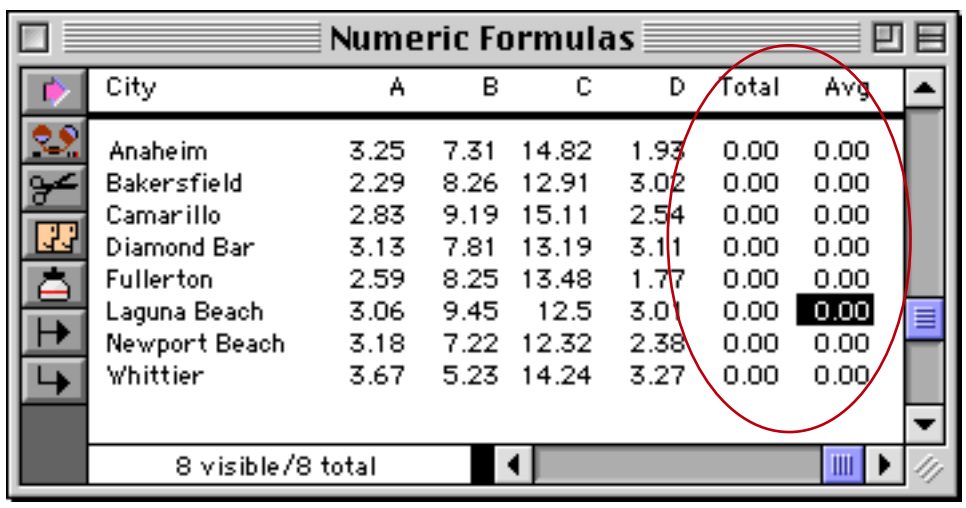
If we click on Laguna Beach and run the procedure again we'll get a different result.



The `fill` statement works fine for constant values, and is slightly faster than `formulafill`.

```
field Total
fill 0
field Avg
fill 0
```

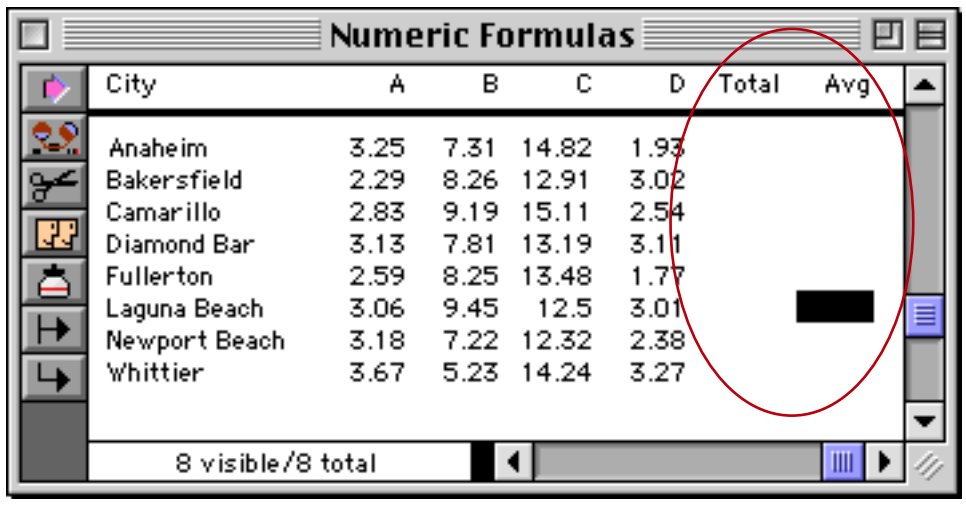
This procedure fills the columns with zeroes.



Use the `zeroblank(` function if you want to fill a numeric field with zeroes (see “Suppressing Zero’s” on page 1699).

```
field Total
fill zeroblank(0)
field Avg
fill zeroblank(0)
```

The result is two completely empty columns.



By the way, you could get the same result with the `formulafill` statement. It would be slightly slower, but the difference would not be measurable unless the database contained tens of thousands of records.

Using FormulaFill to Transform Text

Using the `formulafill` statement you can combine multiple fields, split a field apart, re-arrange words or phrases, and translate characters (for example, converting uppercase to lower case). To illustrate a few examples of this we'll use this contacts database.



This example combines the first and last names into a single field.

```
field Name
formulafill First+" "+Last
```

When triggered the result of this procedure looks like this.



This example combines the first and last names in a different way.

```
field Name
formulafill upper(Last)+", "+First
```

When triggered the result of this procedure looks like this.



Use text funnels to split a field apart or to re-arrange words or phrases. Text funnels allow a formula to extract part of a cell based on a fixed character position within the cell, or based on patterns and context within the cell. See "Taking Strings Apart (Text Funnels)" on page 1214 for a complete explanation of text funnels. This procedure will fill the Name field with the first initial and the last name.

```
field Name
formulafill First[1,1]+". "+Last
```

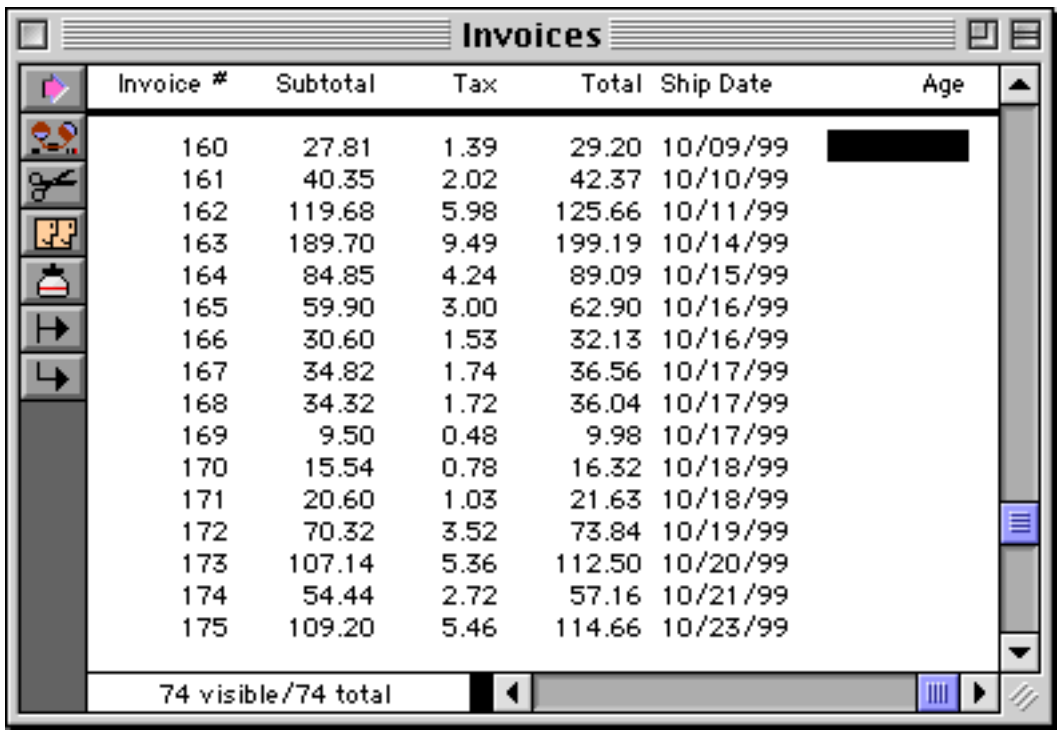
Here is the result.



For more information on formulas that take apart and put together text see "Text Formulas" on page 1212.

Date Calculations with Formula Fill

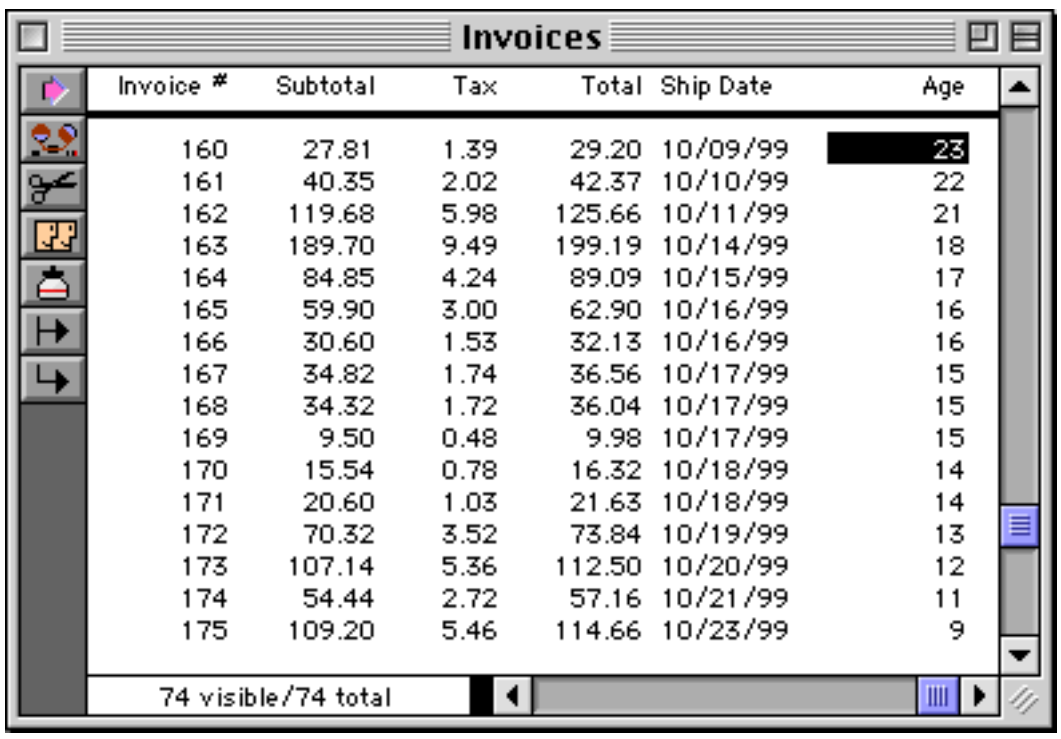
Use the `formulafill` statement to calculate the difference between dates, or to adjust dates. See "HTML Generating Functions" on page 1247 for details on performing calculations with dates. A typical use for date arithmetic is aging of an accounts receivable database.



To calculate the age of an invoice based on the current date, use the **Formula Fill** command with the formula shown here:

```
field Age
formulafill today()-«Ship Date»
```

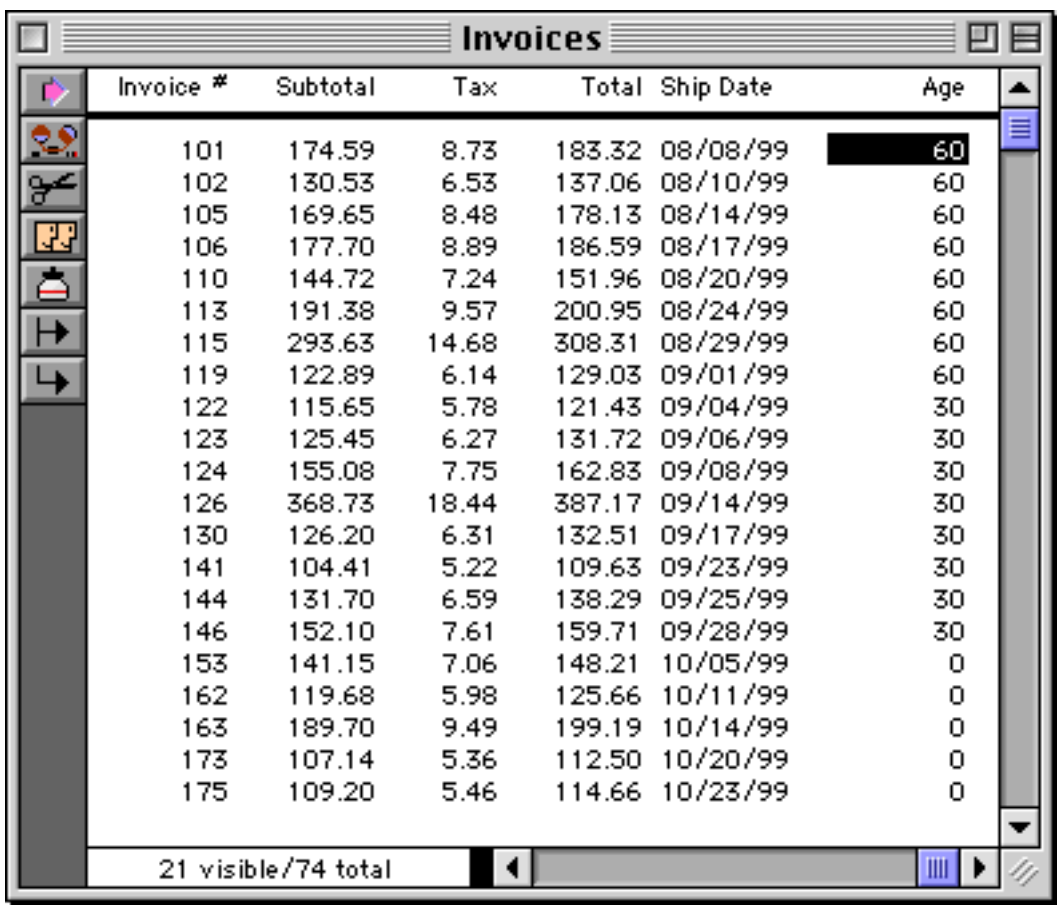
Here is the result of this procedure.



If you want to calculate ages rounded to the nearest 30 day interval use the procedure below instead.

```
field Age
formulafill round((today()-«Ship Date»)-15,30)
```

Here's the result. The age of each invoice rounded to the nearest 30 days.



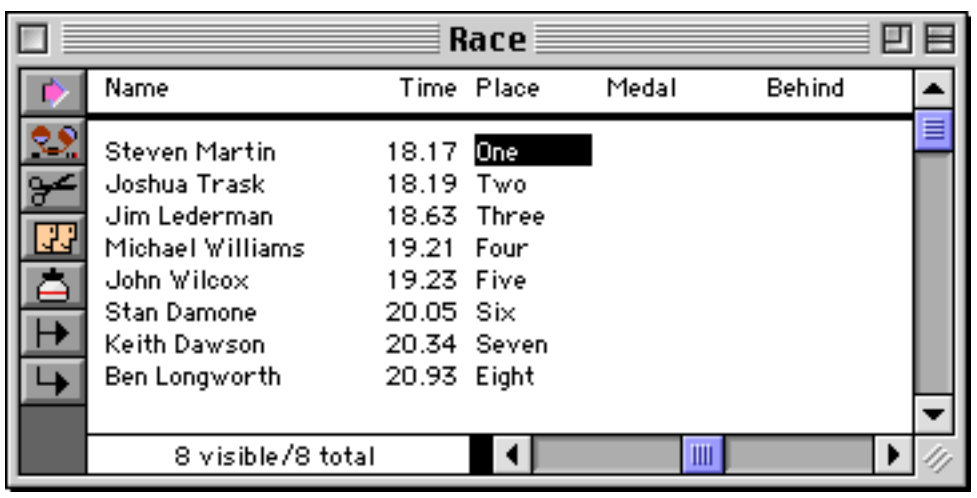
For more information on the `today(` and `round(` functions see "TODAY(" on page 5862 and "ROUND(" on page 5682.

The SEQ Function

The `seq(` function is a special function for use with the `formulafill` statement. This function returns a unique number for each selected record, starting with 1 at the top of the database. Use this function if you need a unique record number in a formula. Here is an example that fills a column with the words One, Two, Three, Four, etc.

```
field Place
formulafill pattern(seq(),"§")
```

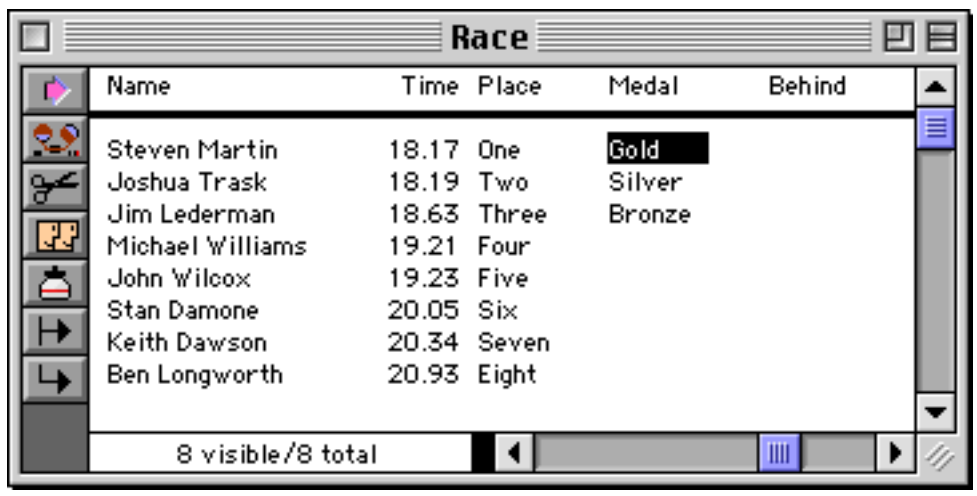
When you press **OK** the field is filled in (see "Displaying Numbers as Words" on page 294 for more information on this output pattern.)



Here is another example that uses the `seq(` function to assign medals to the first three finishers in the race.

```
field Time
sortup
field Place
formulafill array("Gold/Silver/Bronze",seq(),"/")
```

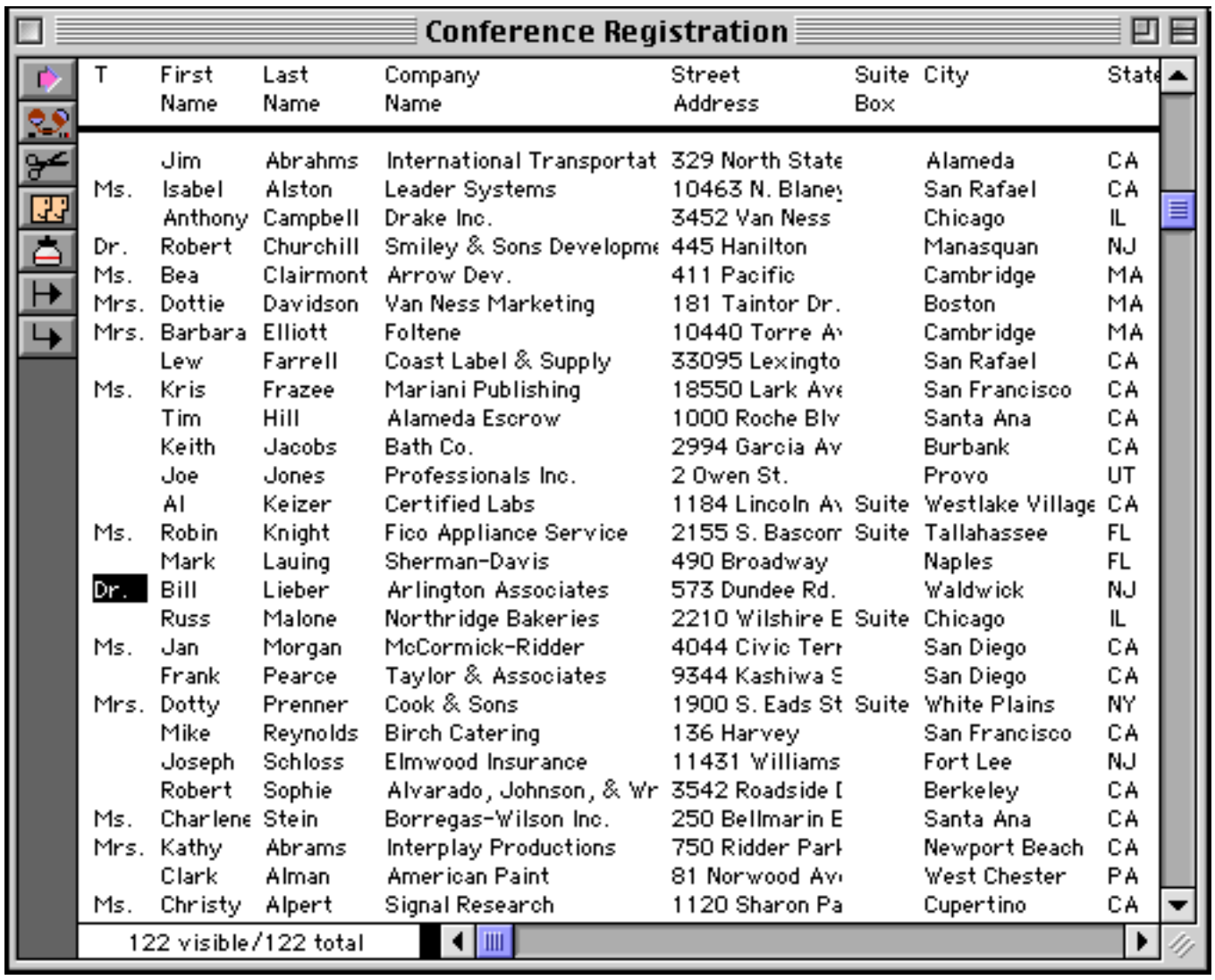
The first three finishers are assigned gold, silver, and bronze medals, with all of the other records left blank.



See "Text Arrays" on page 1236 for more information on the `array(` function used in this example.

Filling Empty Cells

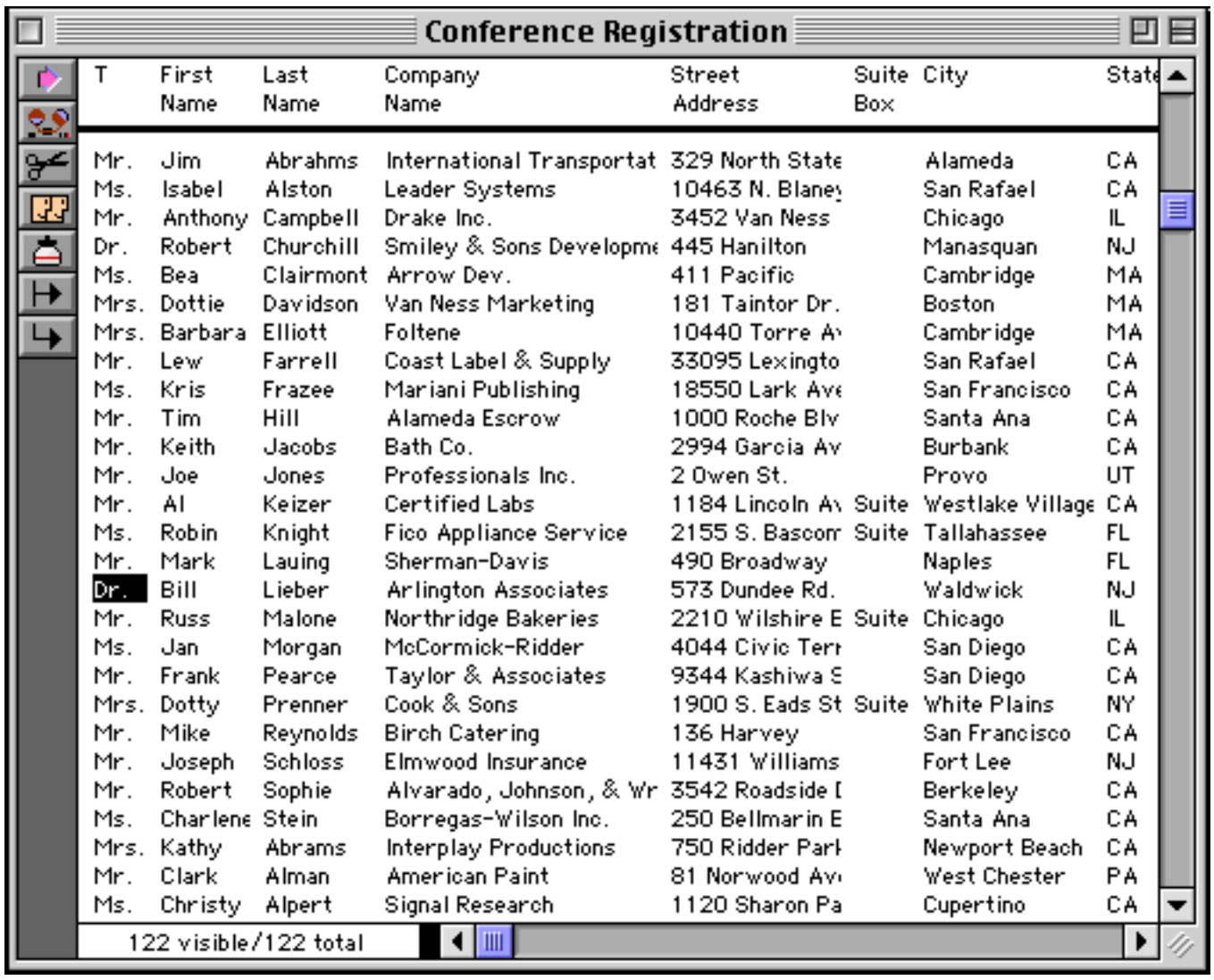
The `emptyfill` statement is very similar to the `fill` statement (see "Fill vs. FormulaFill" on page 1700). However, the `emptyfill` statement will not destroy the data already in the field. In fact, `emptyfill` will only fill cells that are completely empty. Here is a database where some of the name prefixes have been left blank.



Using the `emptyfill` statement these empty cells can quickly be filled with Mr.

```
field T
emptyfill "Mr."
```

Here's the finished result.

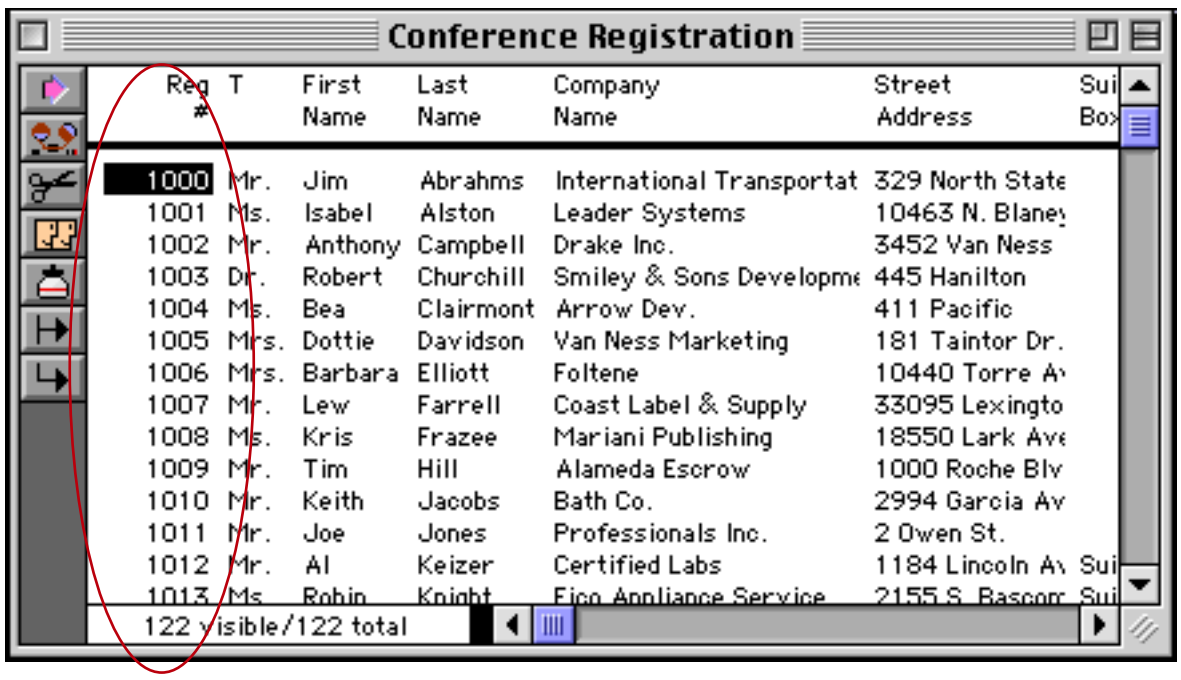


Automatic Numbering

The `sequence` statement fills the current field with a numeric sequence (for example 1, 2, 3 or 100, 110, 120). The `sequence` statement only works with numeric fields, you cannot sequence a text, date, or choice field. The sequence statement has one parameter, a text value that contains two numeric values within it, the starting value and the increment. Here's an example that will number 1000, 1001, 1002, etc.

```
field «Reg #»
sequence "1000 1"
```

Here is the result of this procedure.



The sequence can start with any number and increase by any value, including non-integer values or negative values. The table below shows four examples of starting and increment values.

"1 1"	"5 5"	"1 0.1"	"100 -1"
1	5	1.0	100
2	10	1.1	99
3	15	1.2	98
4	20	1.3	97
5	25	1.4	96

If the database contains summary records, the sequence count will reset to one after each summary record. If you want to sequence the current field without restarting at summary records, use the `formulafill` statement with the formula `seq()`. See "Summaries and Outlines" on page 393 for more information on summary records. See "Making Transformations Even Faster" on page 1698 for more information on the `formulafill` statement.

Propagate and UnPropagate

Like `emptyfill`, the `propagate` statement fills all the empty cells in the current field. However, instead of filling the empty cells with a fixed value, the `propagate` statement propagates filled data cells into the empty data cells (if any) below them. The `propagateup` statement propagates filled data cells into the empty data cells (if any) above them. See “Propagate” on page 465 for examples of these features in action.

The `unpropagate` statement performs the exact inverse of the `propagate` statement. If the same value appears in two or more consecutive data cells, the `unpropagate` statement empties the second and subsequent data cells. The `unpropagateup` statement performs the same operation upside down, leaving the last of several duplicate values while clearing the others. See “UnPropagate” on page 469 for examples of these features in action.

Using UnPropagate to Eliminate Duplicates

The `unpropagate` statement can be used to eliminate duplicate values in a database. To see how to do this manually see “Using UnPropagate to Eliminate Duplicates” on page 470. This procedure will remove all of the duplicate entries in the current field.

```
sortup
unpropagate
select «» <> ""
removeunselected
```

Tip: One possible problem with this technique is that all cells that start out empty will be removed. For example if you are removing duplicate company names but some records don’t contain company names, the records without company names will be removed. To fix this problem, use the `emptyfill` statement to fill the empty names with a unique value like `n/a` before you start, then use the `select` statement to select all values not equal (≠ or) to `n/a`. Then perform the rest of the steps listed above. Here is a revised version of the procedure that takes care of this problem.

```
emptyfill "!empty!"
select «» <> "!empty!"
sortup
unpropagate
select «» <> ""
removeunselected
formulafill ?(«» = "!empty!" , "" , «»
```

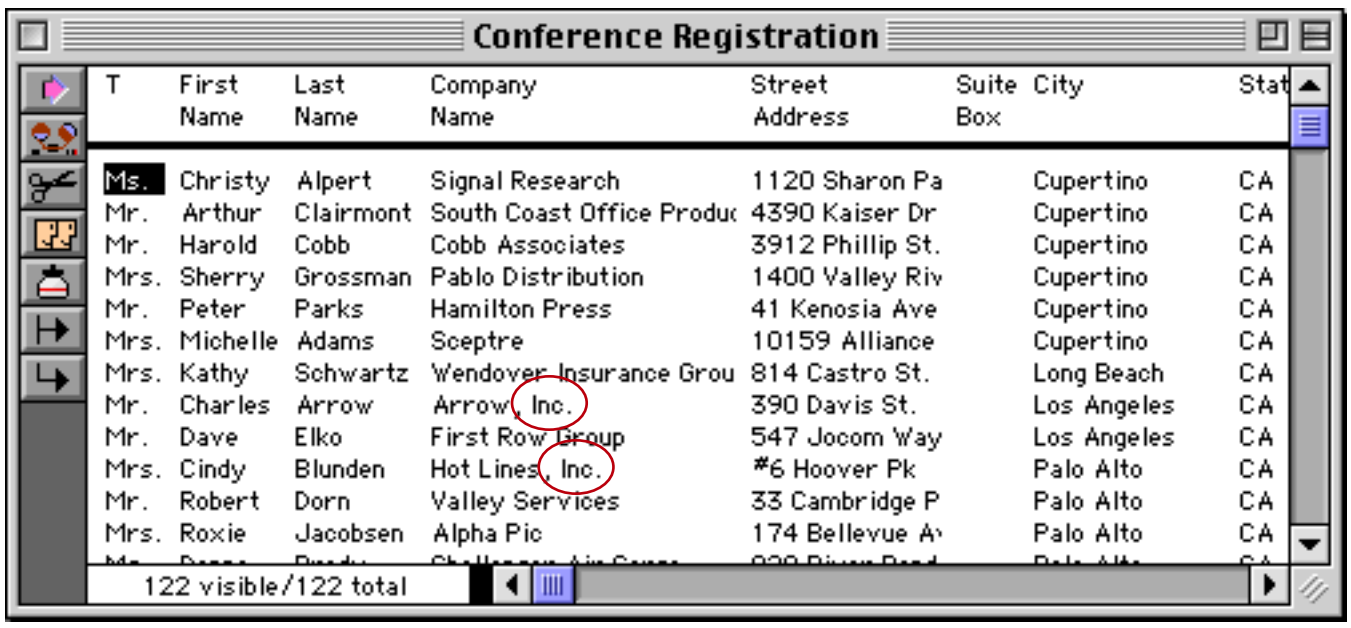
Warning: Keep in mind that all of these techniques will blindly remove all but the first duplicate entry. In this example, there were two entries for Bayshore Typesetting. However, they were probably not really duplicates, since one was in Washington, DC and the other in San Rafael, CA. There is no way for an automatic technique like this to know which of these is really correct, or even if they are really duplicates at all. If you want to manually examine duplicate records instead of blindly deleting them, use the `selectduplicates` statement. See “Selecting Duplicates” on page 1684 for more information on this statement.

Change (Find and Replace)

The `change` statement finds and replaces a word or phrase in the current field (see “CHANGE” on page 5095). For example, you can use the `change` statement to replace every occurrence of Inc. to Incorporated, or every occurrence of Purchase Order to P.O. In it’s most basic form the change statement has two parameters:

```
change <original text>,<new text>
```

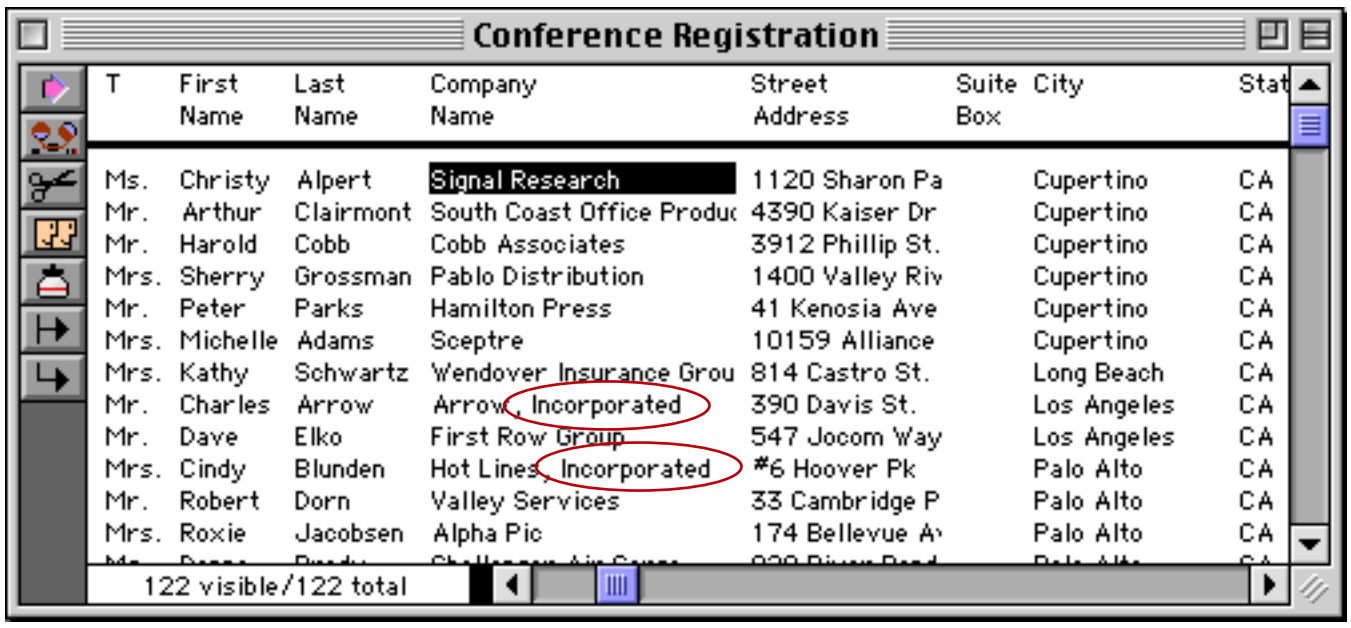
To illustrate this statement we'll use this conference registration database. Notice that it contains the abbreviation Inc. in several places in the company name field.



To change every occurrence of Inc. to Incorporated use this procedure.

```
field «Company Name»
change "Inc.","Incorporated"
```

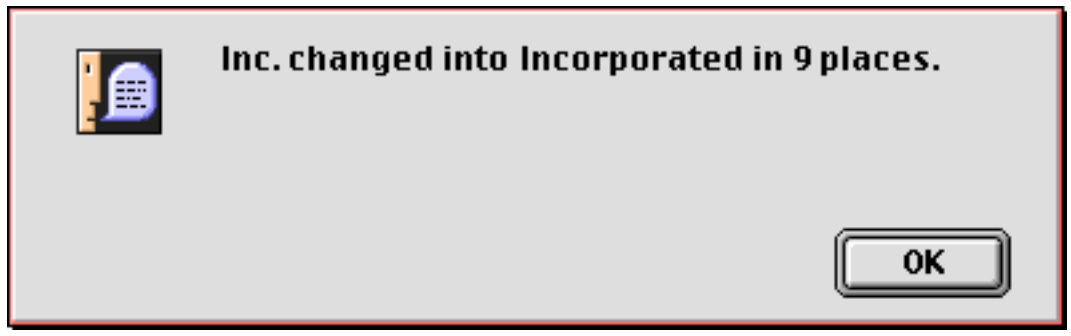
Running this procedure makes the changes.



The procedure can use the `info("changecount")` function to find out how many occurrences of the word or phrase were changed (if any). Here is a modified version of the procedure that simply reports the number of changes.

```
field «Company Name»
change "Inc.","Incorporated"
message "Inc. changed into Incorporated in "+str(info("changecount"))+ " places."
```

Running this revised procedure (on the original data) causes this message to appear.



By adding the `caps` option to the `change` statement you tell Panorama to adjust capitalization as it performs the replacement. The `caps` option should NOT be in quotes, and must be placed after the other parameters, separated by one or more spaces. For example:

```
field «Company Name»
change "Inc.","Incorporated" caps
```

When the `caps` option is added to the statement Panorama will automatically adjust the capitalization of the new word or phrase as it is inserted into the database. If you leave this option off, capitalization is not adjusted. In fact, if the `caps` option is off, only words or phrases that exactly match the capitalization typed into the dialog will be replaced. The table below shows the result of replacing Inc. with Incorporated both with and without the `caps` option.

Original	without **caps** option	with **caps** option
Inc.	Incorporated	Incorporated
INC.	INC.	INCORPORATED
inc.	inc.	incorporated

By adding the `words` option to the `change` statement you tell Panorama to replace only entire words, not sections of words. For example, if you ask Panorama to change is to was, it will also change this to thwas. This is, of course, wrong. To prevent this, add the `words` option after the other parameters, like this.

```
field Body
change "is","was" words
```

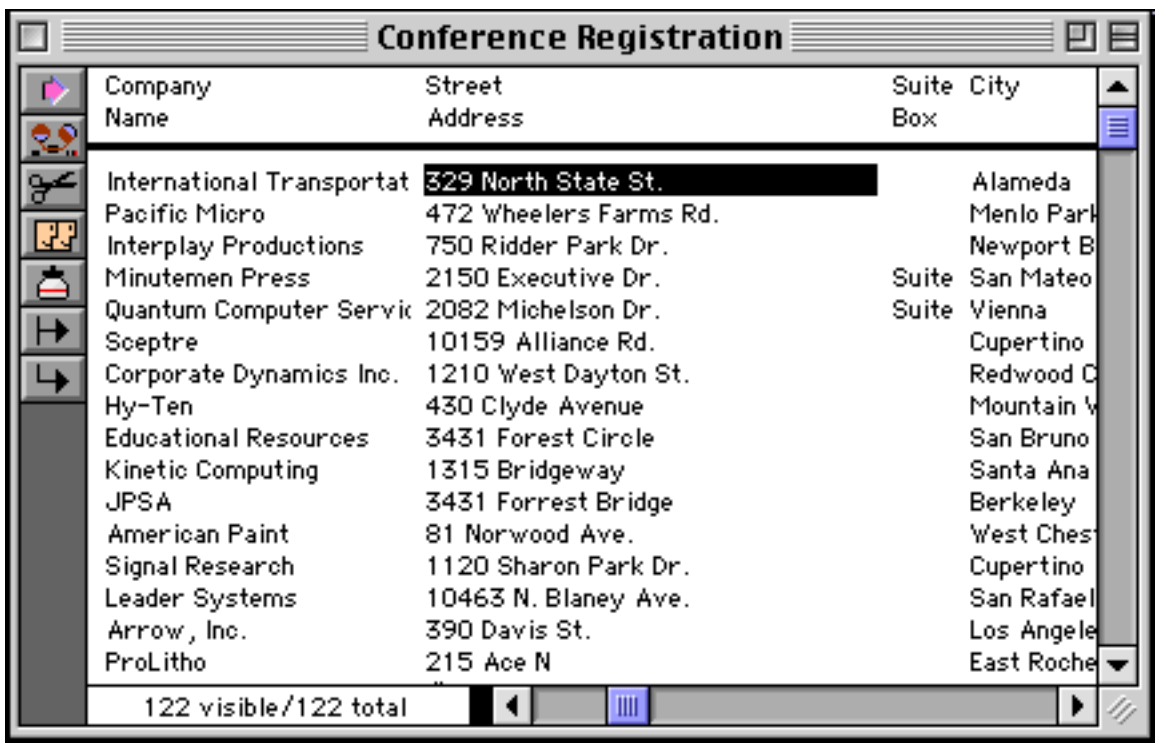
You can combine the `caps` and the `words` options, like this.

```
field Body
change "is","was" caps words
```

The order of the `caps` and `words` options does not matter, as long as both are after the other parameters.

Changing with the Replace(Function

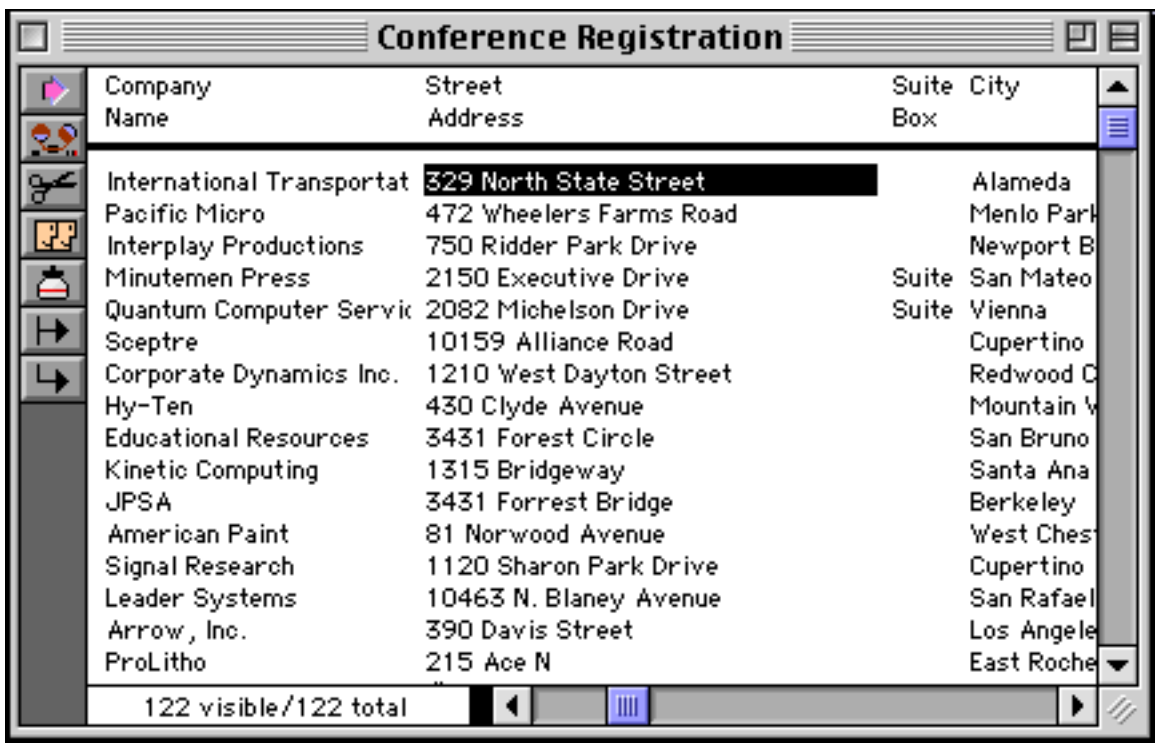
The `change` statement is not the only way to replace words or phrases. You can also use the `formulafill` statement and the `replace(` or `replacemultiple(` functions (see “String Modification Functions” on page 1224). This technique is especially handy if you need to replace several words or phrases at once. For example, consider the addresses in the database below.



Suppose you wanted to expand the abbreviations in these addresses: St. to Street, Dr. to Drive, etc. You could do this by using the `change` statement over and over again. Or you can simply use the `replacemultiple(` function to replace all of the abbreviations in one fell swoop.

```
field «Street Address»
formulafill replacemultiple(«Street Address»,
      "Rd./St./Dr./Ln./Ave.",
      "Road/Street/Drive/Lane/Avenue","/")
```

Running this procedure replaces all of the abbreviations at once.



See “Using FormulaFill to Transform Text” on page 1702 for more information on the `formulafill` statement.

Data Style and Color

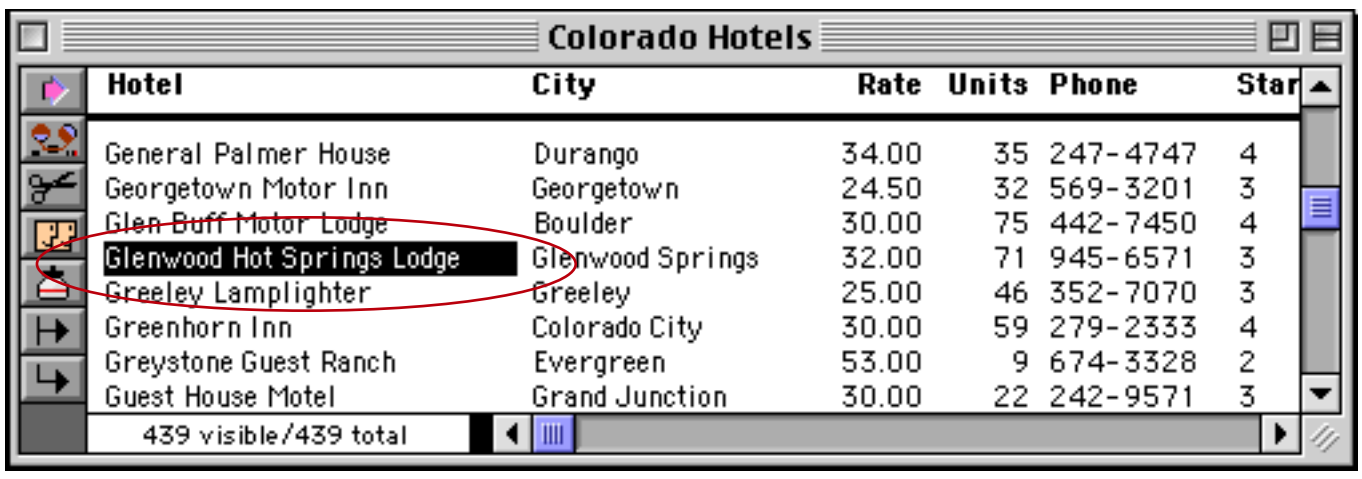
In addition to the data stored in each cell, Panorama also keeps track of the style (plain, bold, italic, etc.) and (to a limited extent) color (red, green, etc.) of each cell (see See "Data Style and Color" on page 474). In a procedure the `stylecolor` statement can be used to change the style or color of one or more data cells. The statement has one parameter which controls what cells get changed (cell, record, field, all), what color the cells should be changed to (black, red, green, blue, cyan, yellow, magenta) and what style (bold, italic, underline, shadow).

If the parameter starts with the word cell, only the current cell will be changed. If the parameter starts with the word record, all the cells in the current record will be changed. If the parameter starts with the word field, all the selected cells in the current field will be changed. If the parameter starts with the word all, every cell in every selected record will be changed.

Here are some examples of different parameter combinations.

```
stylecolor "field blue bold"
stylecolor "all black"
stylecolor "cell red italic"
stylecolor "record bold"
```

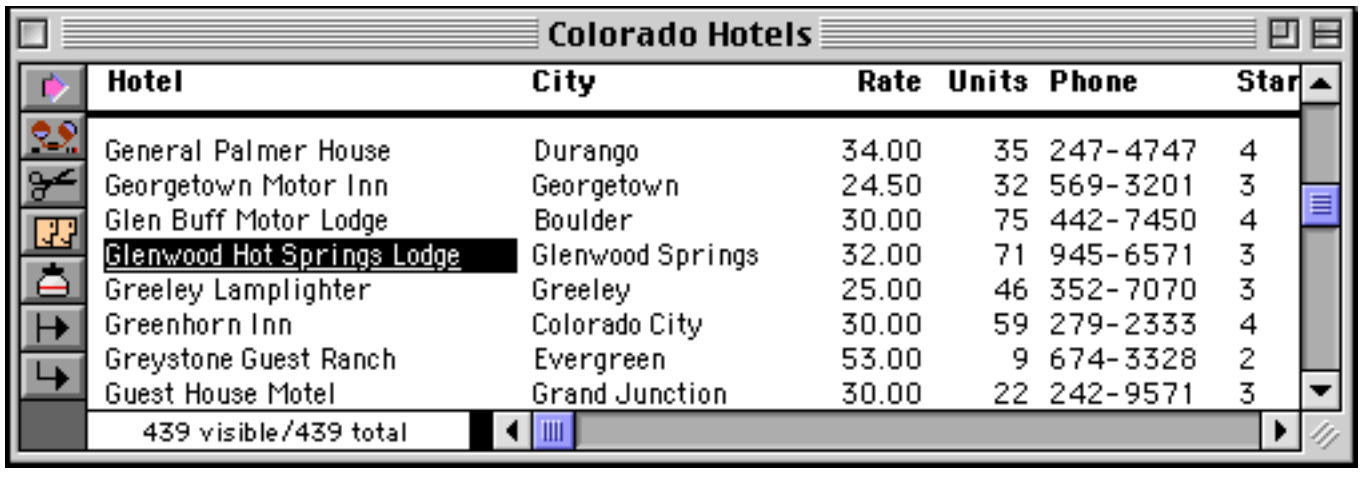
For example, a procedure can underline the current data cell. We'll start with a plain data cell.



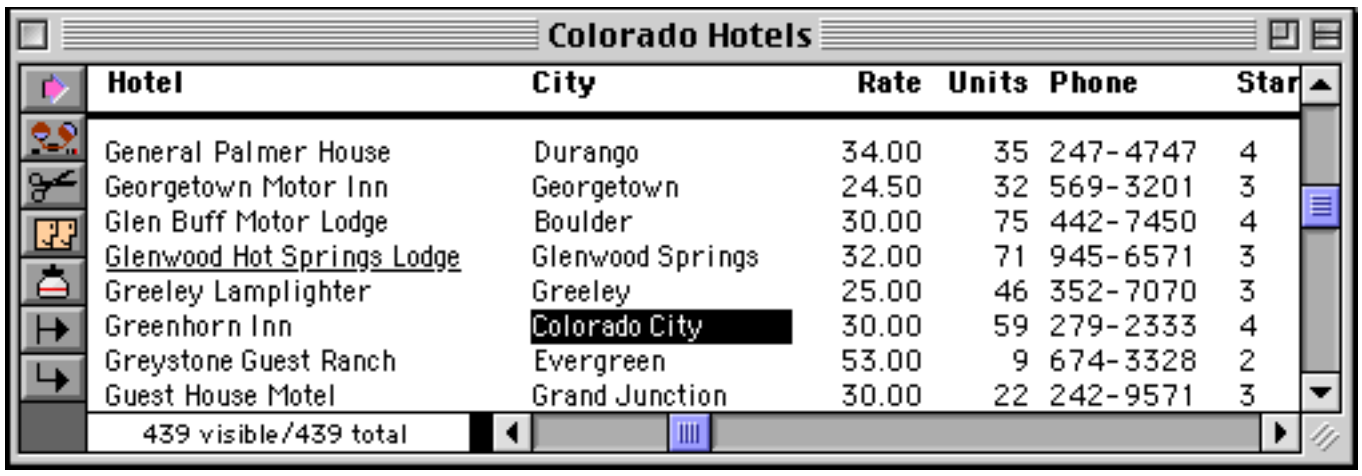
Here is a procedure that changes the style of this cell.

```
stylecolor "cell underline"
```

When you run this procedure the cell will be underlined.



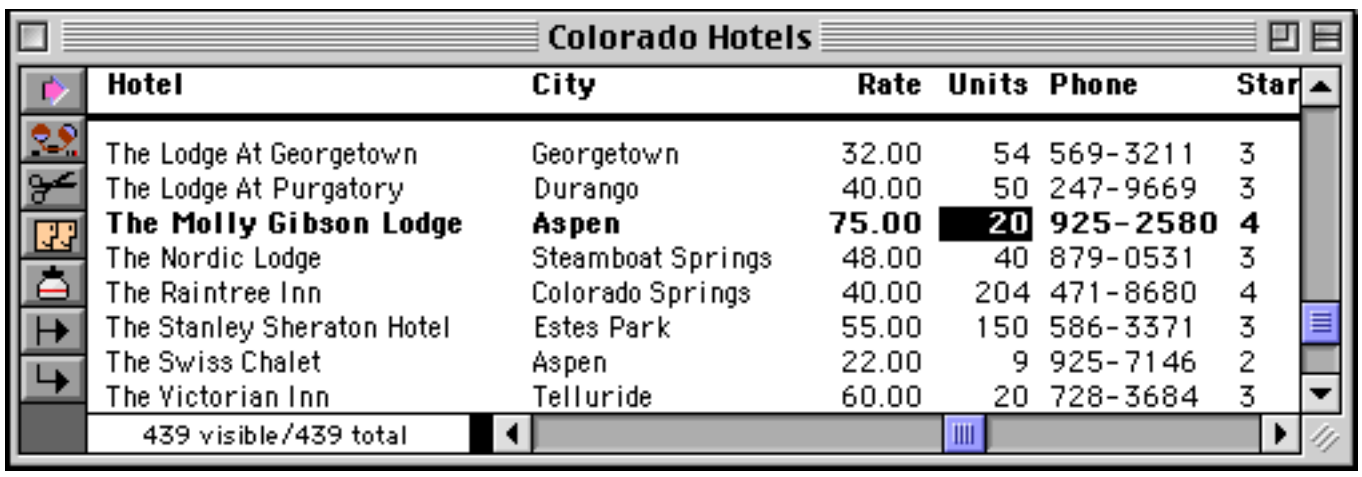
It's easier to see the underline if you click on another cell.



Using a slightly modified procedure we can make an entire line bold.

```
stylecolor "record bold"
```

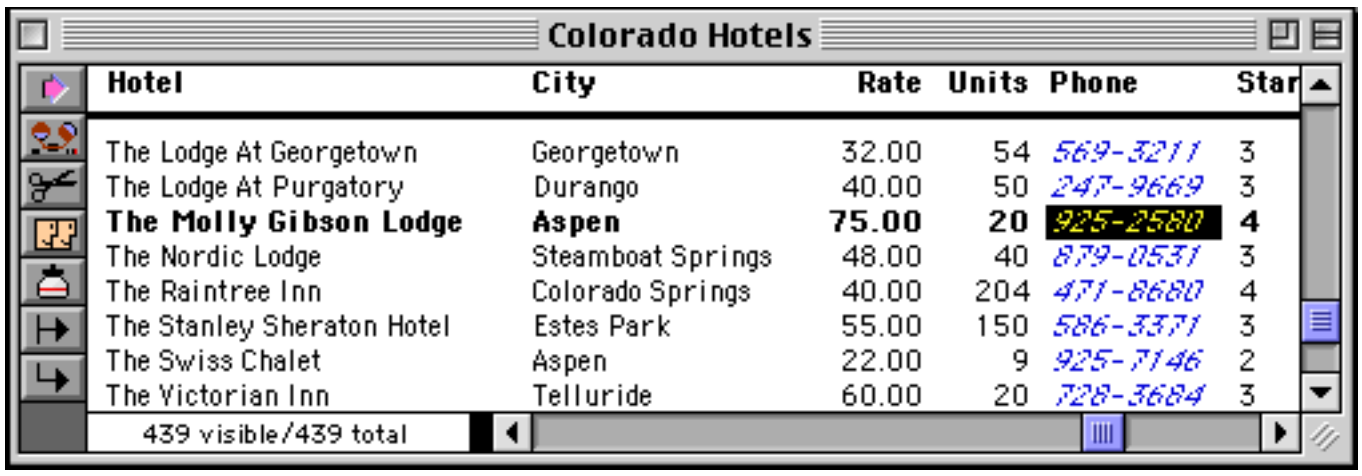
Run the procedure to make the record bold.



This procedure will make all phone numbers appear in italic blue, as shown here.

```
field Phone
stylecolor "field italic blue"
```

Here's the result.

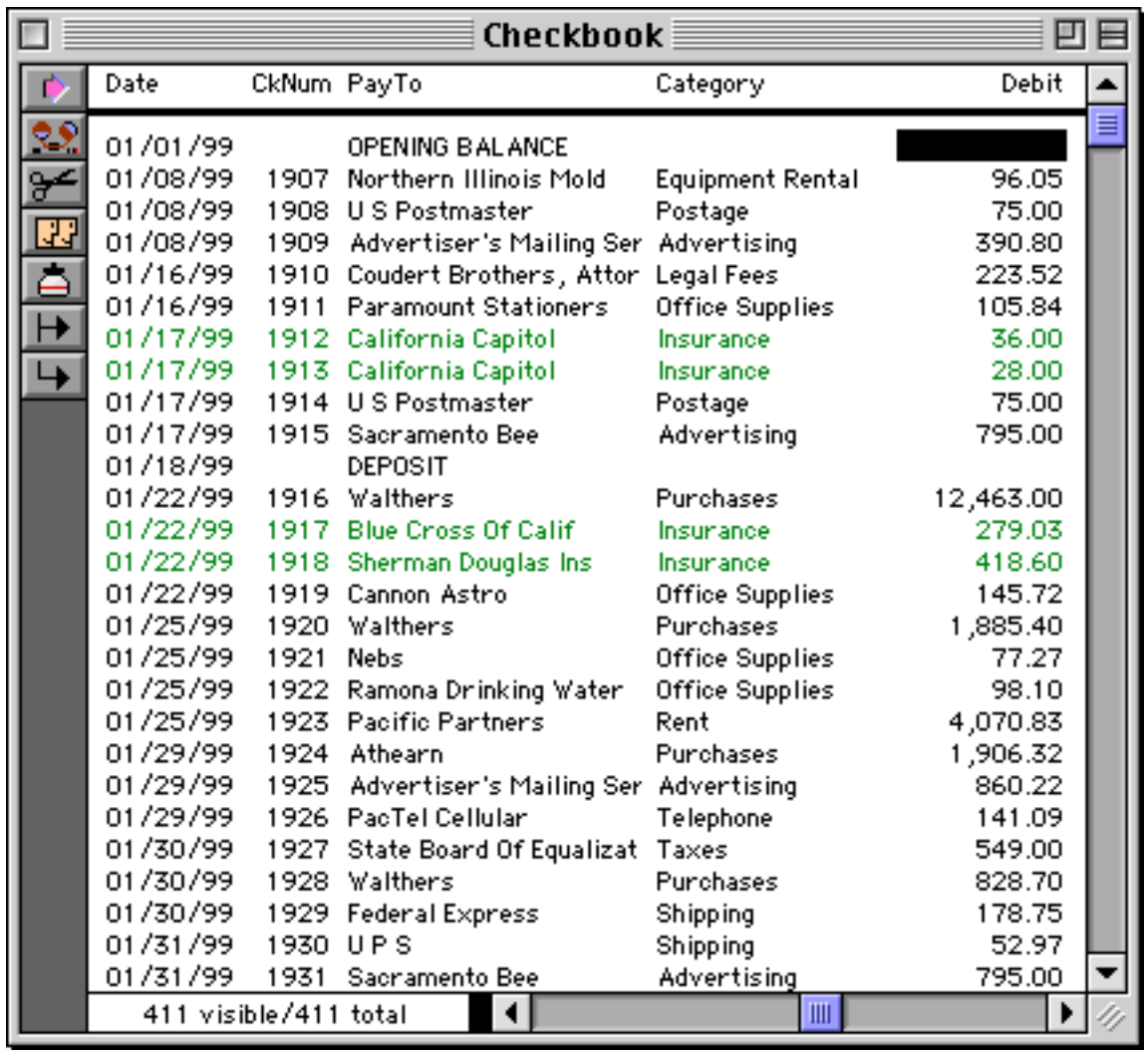


Notice that the italic blue has overridden the bold applied in the previous example.

For our final example we will go to a checkbook database and mark all insurance payments in green.

```
select Category="Insurance"
stylecolor "all green"
selectall
```

When you run this procedure all the insurance records will turn green, like this.



A cell retains its style and color until the data is modified. Any data modification (editing, formula fill, etc.) will cause the cell to revert to plain black.

Every data cell that is not plain black takes up an extra byte of storage. For example a database with 10 fields and 500 records will expand by 5K bytes if you change every data cell to blue or italic (or both).

Accessing Style and Color in a Formula

Panorama formulas can use the `fieldstyle(` function to access both the style and color of individual data cells. When combined with the `select` statement, these functions allows you to select data based on its style or color. (See “Selecting Information” on page 1682 for more information on this command.)

The basic syntax for the `fieldstyle(` function is:

```
fieldstyle(fieldname)
```

This function returns the style and color of a data cell— bold, italic, etc. The fieldname parameter is a string, so it should usually be in quotes—for example `fieldstyle("Price")="bold"`. If the data cell has more than one style or color, this function will return all of them, for example red bold italic. Use the `contains` operator (see “String Testing Functions” on page 1223) to check for a specific style or color, for example

```
select fieldstyle("Name") contains "italic"
```

To check if a cell is plain, use a formula like this

```
fieldstyle("Address")=""
```

For more information on this function see “FIELDSTYLE(” on page 5221.

Processing/Transforming an Entire Array

The previous section described methods for transforming an entire field in a database (See "Transforming Big Chunks of Data" on page 1697). This section describes different methods for quickly processing all the elements in an array. If you are not already familiar with text arrays see "Text Arrays" on page 1236.

"Filtering" an Array

The `arrayfilter` statement allows a procedure to use a formula to process each element of an array (see "ARRAYFILTER" on page 5045). The statement scans the array you specify element by element, and uses the formula you supply to transform each element. It then builds a new array using the transformed elements.

The `arrayfilter` statement has four parameters:

```
arrayfilter oldarray,newarray,separator,formula
```

The first parameter, oldarray, is the original array. This is usually a field or variable, but could be any formula that produces text data.

The second parameter, newarray, is the new array. This must be a field or variable. If you don't mind changing the original array, oldarray and newarray may be the same field or variable. If you want to keep the original array, newarray should be different.

The third parameter is the separator character (see "Picking a Separator Character" on page 1236).

The fourth parameter is the formula that will transform each array element. In addition to the usual functions and operators there are two functions that have special meaning in this function. The `import(` function returns the original data in the array element. The `seq(` function returns the array element number (1, 2, 3, etc.).

Here is a procedure called **.NumberArray** that adds sequence numbers to an array. The procedure that calls **.NumberArray** must pass an array as parameter 1 and a separator character as parameter 2.

```
local tempArray
tempArray=parameter(1)
arrayfilter tempArray,tempArray,parameter(2),"("+str(seq())+") "+import()
setparameter 2,tempArray
```

Here's a procedure that uses **.NumberArray** to produce a numbered list of atomic elements.

```
local Elements
Elements=replace("Hydrogen;Helium;Lithium;Beryllium;Boron;"+
   "Carbon;Nitrogen;Oxygen;Flourine;Neon;"+
   …
   "Mendelevium;Nobelium;Lawrencium",";",¶)
.call .NumberArray,Elements,¶
```

This table shows what the Elements array looks like before and after the **.NumberArray** procedure processes the array.

Before	After
Hydrogen	(1) Hydrogen
Helium	(2) Helium
Lithium	(3) Lithium
Beryllium	(4) Beryllium
Boron	(5) Boron
Carbon	(6) Carbon
Nitrogen	(7) Nitrogen
Oxygen	(8) Oxygen
Fluorine	(9) Fluorine
Neon	(10) Neon
…	…
…	…
Mendelevium	(101) Mendelevium
Nobelium	(102) Nobelium
Lawrencium	(103) Lawrencium

Stripping Blank Elements From An Array

The `arraystrip(` function removes all blank elements from an array (see “ARRAYSTRIP(” on page 5059). This function has two parameters: the original array, and the separator character for that array.

```
arraystrip(array,separator)
```

The `arraystrip(` function can be combined with the `arrayfilter` statement to produce a subset of an array. This example creates a list of recent local phone numbers.

```
global RecentLocalPhone
arrayfilter RecentPhone,RecentLocalPhone,¶,
   ?(length(import())<10,import(),"")
RecentLocalPhone=arraystrip(RecentLocalPhone,¶)
```

The table below shows how this procedure works. The first column shows the original RecentPhone array, with 6 phone numbers, 3 local, 3 in other area codes. The second column shows the RecentLocalPhone array after the `arrayfilter` statement. Using the `?(` function the formula has “blanked out” all the phone numbers with area codes (see “The ? Function” on page 1271). The array elements are still there, but they are empty. The third column shows the RecentLocalPhone array after the `arraystrip(` function. All the empty array elements have been removed, so this array now has only 3 items.

Original	after ArrayFilter	after ArrayStrip(
784-3490	784-3490	784-3490
(213) 454-3309		940-2234
(408) 339-7792		878-2256
940-2234	940-2234	
(303) 452-2284		
878-2256	878-2256	

Reversing the Order of an Array

The `arrayreverse(` function reverses the order of the elements of an array (see "ARRAYREVERSE(" on page 5051). For example, the formula:

```
arrayreverse("1;2;3;4",";")
```

will produce the array 4;3;2;1.

The formula below could be used with an auto-wrap or text display object to display all the checks written to a company, starting with the most recent check (assuming the Checkbook database is sorted by date).

```
arrayreverse(lookupall("Checkbook",«Pay To»,Company,«Check Num»,¶),¶)
```

Using Regular Text Functions with Arrays

Don't forget that you can use regular text processing techniques on arrays (see "Text Formulas" on page 1212). After all, an array is simply a chunk of text that happens to have separator characters in it. For example, to convert our entire atomic element array to upper case in one fell swoop, just use this assignment:

```
Elements=upper(Elements)
```

The assignment below will change all the stainless steel parts in the PartsList array to cheap plastic.

```
PartsList=replace(PartsList,"Stainless Steel","Cheap Plastic")
```

Of course no Panorama customer will ever need a formula like that!

Sorting an Array

Sorting an array in a procedure is easy. The `arraysort` statement does all the work (see "ARRAYSORT" on page 5058). This statement has three parameters:

```
arraysort oldarray,newarray,separator
```

The first parameter, oldarray, is the original array. This is usually a field or variable, but could be any formula that produces text data.

The second parameter, newarray, is the new array. This must be a field or variable. If you don't mind changing the original array, oldarray and newarray may be the same field or variable. If you want to keep the original array, newarray should be different.

The third parameter is the separator character.

The `arraysort` statement always sorts the array in ascending order (A's first, Z's last). Upper case, lower case, and international letters will be sorted correctly (i.e. a comes before B, which may seem obvious but is not the normal ASCII order).

The example below builds a fileglobal variable named FormList that contains a list of the forms in the current database. The list is carriage return delimited and alphabetized. You could use this list with a Pop-Up Menu or a List SuperObject™.

```
fileglobal FormList
FormList=dbinfo("forms","")
arraysort FormList,FormList,¶
```

Removing Duplicate Items from an Array

The `arraydeduplicate` statement also sorts an array. After it sorts the array it eliminates all the elements that are duplicated (see “ARRAYDEDUPLICATE” on page 5042). For example, if an array contains San Francisco three times, this statement will eliminate two and leave only one. The parameters for the `arraydeduplicate` statement are the same as for the `arraysort` statement.

```
arraydeduplicate oldarray,newarray,separator
```

The example below creates a fileglobal variable named Companies, then fills it with a sorted list that contains all the companies in California listed in alphabetical order:

```
fileglobal Companies
Companies=lookupall("Invoices",State,"CA",Company,¶)
arraydeduplicate Companies,Companies,¶
```

Each company will be listed only once, no matter how many times the company appears in the Invoices database.

(Note: There is no automatic way to eliminate the duplicate values in an array without also sorting the array.)

Building an Array from a Database

The previous example showed how an array can be built from the data in a database using the `lookupall(` function. An even more powerful technique is the `arraybuild` statement (see “ARRAYBUILD” on page 5038). This statement scans a database and, using a formula you supply, extracts information from each record to build an array. The statement has four parameters:

```
arraybuild array,separator,database,formula
```

The first parameter, array, is the new array you want to build. This must be a field or variable.

The second parameter is the separator character (see “Picking a Separator Character” on page 1236).

The third parameter, database, is the name of the database that contains the information that will be scanned to build the array. This database must be currently open at the time the `arraybuild` statement is used. If you want to use the current database use the function `info("databasename")` or simply use `""`.

The fourth parameter, `formula`, is the heart of this statement. As the `arraybuild` statement scans the database record by record, it uses this formula to extract data from each record and add it to the array. The formula can also be used to select which records appear in the array. If the formula produces empty text for a particular record, that record will not be included in the array. The formula can reference any field in the database being scanned.

The example below produces a list of past due invoices.

```
fileglobal PastDueAccounts
arraybuild PastDueAccounts,¶,"Invoices",
   ?(Balance>0 and today()-InvoiceDate>30,
   pattern(InvoiceNumber,#####)+" ("+Company+")","")
```

This procedure will produce an array that will look something like this:

```
00436 (Acme Widgets)
02445 (Optimal Resolution Trust)
03689 (Zippy Car Wash)
```

This array could be displayed on a report, or it could be used in a pop-up menu or scrolling list.

Warning: One thing to be careful about when building arrays with the `arraybuild` statement is the size (number of characters) of the array you are building. The array must fit in Panorama's scratch memory allocation. If you are going to build very large arrays you may need to increase this allocation (see "Changing Scratch Memory Size (Macintosh OS 9)" on page 192).

Note: The `arraybuild` statement scans every record in the database, whether it is selected or not. If you would only like to scan selected records, use the `arrayselectedbuild` statement. This statement is identical to the `arraybuild` statement except for the fact that it only scans selected records. If you know that all the records you want to scan are selected, this statement may be much faster than the regular `arraybuild` statement.

Appending an Array to a Database

An array in a variable can be appended to a database almost as if it were a text file on the disk. The array must be carriage return delimited, and if there is more than one field, the fields must be tab delimited. To append a variable to the current database, put `+@` in front of the variable name like this:

```
openfile "+@Array"
```

To replace the entire current database with the array, put `&@` in front of the variable name like this:

```
openfile "&@Array"
```

Here is an example that transfers houses from a Listings database to a Sales database.

```
local TransferArray
arraybuild TransferArray,¶,"Listings",
   ?(Escrow≠"Closed","",
   Date+¬+Address+¬+City+¬+State+¬+Zip+¬+str(SoldPrice) )
select Escrow≠"Closed"
removeunselected
open "Sales"
open "+@TransferArray"
```

The `arraybuild` statement copies all the recently sold listings into the TransferArray variable. The ¬ character (see "Special Characters" on page 1202) separates each field with the required tab. Once the listings are safely copied into the array, they are deleted from the Listings database. The first `open` statement makes sure that the Sales database is open and on top. This database has six fields, SoldDate, Address, City, State, and SoldPrice, in that order. The formula in the `arraybuild` statement has been set up to create the array with the fields in that order. The final statement of the procedure appends the data in TransferArray to the end of the Sales database.

If you are transferring information between two databases with identical fields the `exportline(` function can come in handy (see "EXPORTLINE(" on page 5210). This function outputs all the fields in the current record with tabs in between. The example below appends all of the invoices in the database Paul's Invoices to the current database.

```
local TransferArray
arraybuild TransferArray,¶,"Paul's Invoices",exportline()
openfile "+@TransferArray"
```

This example transferred all of the information across, but you could use the `?(` function to transfer just a subset.

Copying Between Multiple Variables and an Array

Panorama has the ability to combine multiple variables into a single array, or to take an array and split it into many separate variables. This capability can be useful for editing arrays (each array element can be edited in a separate variable) and for saving a collection of variables in a single disk file (for example to store preferences).

The `savevariables` statement takes a list of variables and combines the values of all the variables into a single array. The statement has three parameters:

```
savevariables VariableList,CombinedArray,Separator
```

The VariableList parameter is an array containing the names of the variables to be included in the result. Each item in the array must be separated from the next by the Separator character (see “Picking a Separator Character” on page 1236).

The CombinedArray parameter is a field or variable name. The statement will build the final array of values in this field or variable, using the Separator character to divide each item. Any numbers will be converted to text as the array built. The example below saves all of the `fileglobal` variables for the current file into a disk file.

```
local fileExtraData
savevariables info("filevariables"),fileExtraData,¶
filesave "",info("databasename")+" Variables","",fileExtraData
```

The following example is similar but it saves both the variable name and the data in the format variable=value. The variables will be listed in alphabetical order.

```
local varNames,varData
varNames=info("filevariables")
arraysort varNames,varNames,¶
savevariables info("filevariables"),varData,¶
arrayfilter varData,varData,¶,array(info("filevariables"),seq(),¶)+"="+import()
filesave "",info("databasename")+" Variables","",varData
```

The resulting file will be named something like Contact Variables (the exact name depends on the database name) and will look something like this:

```
ActiveForm=Contacts
LocalAreaCode=714
SearchText=Chicago
```

The `loadvariables` statement takes an array of values and splits the values into individual variables. If the variables don’t exist, the statement will create them. The `loadvariables` statement has three parameters:

```
loadvariables VariableList,VariableValues,Separator
```

The VariableList parameter is an array containing the names of the variables to be loaded. Each item in the array must be separated from the next by the Separator character (see “Picking a Separator Character” on page 1236).

The VariableValues parameter is an array containing the values of the variables. This array must be in the same order as the VariableList parameter. Each value in the array must be separated from the next by the Separator character.

Here is an example that loads three variables from an array.

```
loadvariables "Gold,Silver,Bronze","Johnson,Smith,Fetzl",","
```

This example is exactly the same as:

```
Gold="Johnson"
Silver="Smith"
Bronze="Fetzl"
```

If a variable already contains a numeric value, the `loadvariables` statement keeps it numeric if possible (if all the characters in the new value are numeric). Here is an example where three numbers are loaded into variables.

```
fileglobal Red,Green,Blue
Red=0 Green=0 Blue=0
loadvariables "Red,Green,Blue","24,58,199",","
```

This example is exactly the same as the procedure below. Notice that there are no quotes around the numbers.

```
fileglobal Red,Green,Blue
Red=24
Green=58
Blue=199
```

So far the examples aren't too exciting. Here is an example that is a bit more interesting. Suppose you had an array called ContactInfo that contained name/value pairs like this:

```
Name:Johnson
Email:ajohnson@worldwide.com
url:www.worldwide.com
```

The example below can take this array and separate it into three variables called ContactName, ContactEmail, and Contacturl.

```
global ContactInfo
local contactVariables,contactValues
arrayfilter ContactInfo,contactVariables,¶,"Contact"+array(import(),1,":")
arrayfilter ContactInfo,contactValues,¶,array(import(),2,":")
loadvariables contactVariables,contactValues,¶
```

The procedure starts by splitting the ContactInfo array into two separate arrays for variable names and values, then creates and loads the variables with the values.

The `loadvariables` statement will automatically create fileglobal variables if they do not exist. If you want to create some other kind of variable you can use one of the four statements listed below. (The `loadfileglobalvariables` statement is actually exactly the same as the `loadvariables` statement, but is included for completeness.)

```
LoadGlobalVariables VariableList,VariableValues,Separator

LoadFileGlobalVariables VariableList,VariableValues,Separator

LoadWindowVariables VariableList,VariableValues,Separator

LoadLocalVariables VariableList,VariableValues,Separator
```

If the variables have already been created then these four statements all work exactly the same.

Editing an Array using Separate Variables

The loadvariables and savevariables statements can be used to help edit an array as individual components. Each component has it's own SuperObject for text editing, but the results are all combined into a single variable.

Start by defining the variable names for each individual line item component. You'll also need to build a form with SuperObjects to edit each of these variables.

```
fileglobal LineItems
LineItems=replace("Qty1/Desc1/Price1/Qty2/Desc2/Price2","/",¶)
```

When you open the form to edit the array, this procedure will fill the separate variables with data from the LineItemData array (which could be a field or a variable).

```
loadvariables LineItems,LineItemData,¶
```

When a component is modified you can rebuild the combined array with this procedure.

```
savevariables LineItems,LineItemData,¶
```

Processing/Transforming Binary Data

By now probably everyone who has ever used a computer for more than a week has heard that at their core, computers work with 1's and 0's, on and off, true and false. This is called binary data, because there are only two options. Fortunately, users don't ever have to deal with raw binary data. The programmers take the 1's and 0's and give them structure to create text, numbers, pictures, and other complex elements.

It's not much fun, and it's rarely necessary, but Panorama does allow a procedure programmer to work with raw, unstructured, binary data: 1's and 0's. When you work with raw binary data it will always be in a text field or variable. Panorama normally interprets text as a series of characters, as described earlier in this chapter. The binary functions, however, do not interpret the binary data as characters. Instead, they allow you to directly access and manipulate the 1's and 0's. Panorama uses the text data type to hold raw binary data because text may be of any length and places no restrictions on the binary information that is placed in it. (However, the text may look very strange if you display it in the data sheet or on a form; more on this later.)

Bits

The fundamental unit of computer information is a bit. A bit contains a single 1 or 0. However, a bit is too small to be of much use by itself, so usually several bits are grouped together into a collection called a byte, word, or longword.

Function	Reference Page	Description
bit(number)		This function converts a bit number (1 to 32) into a number (1, 2, 4, 8, 16, etc.)
getbit(number,bitnumber)		This function returns a true or false value by testing a bit. The bit number may be from 1 to 32. If the bit is set, the value will be true, if it is not set, the value will be false.
onescomplement(number)		This function returns the one's complement of a 32 bit number (all bits are reversed)
setbit(number,bitnumber,truefalse)		This function sets one bit within a number, without disturbing any of the other bits. The bit number may be from 1 to 32. The bit will be set based on the true / false parameter - set if true, cleared if false.

Bytes

A byte is a collection of 8 bits. There are 256 possible combinations of 1's and 0's within a byte (2^8). These 256 possible combinations could represent characters, they could represent numbers from 0-255 or they could represent 256 of anything else. The byte(function takes a number from 0 to 255 and converts it into the corresponding pattern of 8 bits.

Function	Reference Page	Description
byte(number)	Page 5079	This function converts a number into a single byte of binary data. (Note: the byte(function is basically the same as the chr(function.) The number parameter must be between 0 and 255. This function converts the number into a single byte of binary data (8 bits).
binaryvalue(data)	Page 5075	This function converts binary data (a byte, word, or longword) into a number. Data is the binary value that you want to convert into a number. This value must be a byte, a word (2 bytes) or a longword (4 bytes). The result is an integer.
bytearray(data,index)		This function extracts a value from an array of bytes. This is not a Panorama style delimited array but a C style array of 8 bit values. The result is an integer.
chararray(data,index)		This function extracts a characters from an array of characters. This is not a Panorama style delimited array but a C style array of ASCII characters. The result is an single characters.

Words

A word is a collection of 16 bits (or 2 bytes). There are 65,536 possible combinations of 1's and 0's within a word (216). These 65,536 possible combinations could represent numbers from 0-65,535 or they could represent 65,536 of anything else. The word(function takes a number from 0 to 65,535 and converts it into the corresponding pattern of 16 bits.

Function	Reference Page	Description
word(number)	Page 5906	This function converts a number into a single word (2 bytes) of binary data. Number is the value that you want to convert into a binary number. This value must be between 0 and 65,535. This function converts the number into a two bytes of binary data (16 bits).
binaryvalue(data)	Page 5075	This function converts binary data (a byte, word, or longword) into a number. Data is the binary value that you want to convert into a number. This value must be a byte, a word (2 bytes) or a longword (4 bytes). The result is an integer.
wordarray(data,index)		This function extracts a value from an array of words. This is not a Panorama style delimited array but a C style array of 16 bit values. The result is an integer.

Long Words

A longword is a collection of 32 bits (or 4 bytes). There are over 4 billion possible combinations of 1's and 0's within a longword (232). The longword(function takes a number from 0 to 4,294,967,295 and converts it into the corresponding pattern of 32 bits.

Function	Reference Page	Description
longword(number)	Page 5499	This function converts a number into a single longword (4 bytes) of binary data. Number is the value that you want to convert into a binary number. This value must be an integer. This function converts the number into a four bytes of binary data (32 bits).

Function	Reference Page	Description
binaryvalue(data)	Page 5075	This function converts binary data (a byte, word, or longword) into a number. Data is the binary value that you want to convert into a number. This value must be a byte, a word (2 bytes) or a longword (4 bytes). The result is an integer.
longwordarray(data,index)		This function extracts a value from an array of longwords. This is not a Panorama style delimited array but a C style array of 32 bit values. The result is an integer.

Creating Binary Values

Binary values are created with the `byte(`, `word(` and `longword(` functions. The example below builds a text data value from a longword, a word, a word and a byte. The resulting text item has a length of 9 (4+2+2+1).

```
global rawData
rawData=longword(96345)+
   word( 1249)+
   word( 9004)+
   byte( 80)
```

Numeric values can be recovered from a text data item with the `binaryvalue(` function. The text input into this function must have a length of 1, 2, or 4. You can use text funnels to control the position and length of the data being converted. The example below will extract four values from a text item that is at least 9 bytes long.

```
global rawData
local myLong,myFirstWord,mySecondWord,myByte
myLong= binaryvalue( rawData[1;4])
myFirstWord= binaryvalue( rawData[5;2])
myFirstWord= binaryvalue( rawData[7;2])
myByte= binaryvalue( rawData[9;1])
```

If rawData contains the information stored in it from the previous example, the myLong variable will contain 96345, myFirstWord will be 1249, mySecondWord will be 9004, and myByte will be 80.

One Dimensional Arrays of Binary Values

Programmers familiar with languages like C and Pascal will be familiar with the concept of an array as a sequence of binary values, for example a sequence of bytes, words, longwords, or some other length of binary data. This is quite different from Panorama's usual concept of arrays as variable length items separated by a special character. Panorama has several functions for extracting a specific element from an array of binary values. The function you pick depends on the type of binary value in the array.

Function	Reference Page	Description
bytearray(data,index)		This function extracts a value from an array of bytes. This is not a Panorama style delimited array but a C style array of 8 bit values. The result is an integer.
chararray(data,index)		This function extracts a characters from an array of characters. This is not a Panorama style delimited array but a C style array of ASCII characters. The result is an single characters.
chunkarray(data,index, chunklength)		This function extracts a binary chunk from an array of chunks. This is not a Panorama style delimited array but a C style array of binary chunks. The result is a binary value (text).
longwordarray(data,index)		This function extracts a value from an array of longwords. This is not a Panorama style delimited array but a C style array of 32 bit values. The result is an integer.

Function	Reference Page	Description
wordarray(data,index)		This function extracts a value from an array of words. This is not a Panorama style delimited array but a C style array of 16 bit values. The result is an integer.

Panorama also has three statements designed for rapidly filtering and/or processing arrays of binary data — characterfilter, chunkfilter, and textfilter.

The CharacterFilter Statement

This statement scans text on a character by character basis. As it scans the input text, it builds a new text field or variable. The contents of the new text is based on the result of the formula which is applied to each character of the original text. The formula can use the `import()` function to retrieve the original character, or the seq() function to retrieve the position of the character within the text.

The `CharacterFilter` statement has three parameters: OriginalText,NewText and Formula.

The OriginalText parameter is the text you want to scan. This may be a field, a variable, or any formula that calculates a text value.

The NewText parameter is the name of a field or variable for holding the output text. If this is a field, it must be a text field.

The Formula parameter is a formula that produces a text result. This formula will be calculated for each character in the OriginalText parameter. The result will be appended to the NewText field or variable. The result of this formula isn't limited to a single character, but may be several characters or zero characters (""). Within the formula you can use the `import()` function to retrieve the original character, or the seq() function to retrieve the position of the character within the text.

Let's review some examples that illustrate the operation of the CharacterFilter statement. This example converts a string of characters into a comma separated array. For example, if A is abcd, then B will become a,b,c,d, .

```
characterfilter A,B,import()+","
```

This example reverses the characters in a string of text. For example, if A is abcd, then B will become dcba .

```
characterfilter A,B,A[-seq();1]
```

This example converts text into a set of hex digits. For example, if A is abcd, then B will become 61 62 63 64 .

```
characterfilter A,B,radixstr("hex",asc(import()))[-2,-1]+" "
```

The ChunkFilter Statement

This statement is almost the same as CharacterFilter, but instead of single characters it allows you to process fixed length chunks. Useful for base64 encryption, hex/ascii conversion, cryptography, etc. As it scans the input text, it builds a new text field or variable. The contents of the new text is based on the result of the formula which is applied to each character of the original text. The formula can use the `import()` function to retrieve the original chunks of characters, or the seq() function to retrieve the chunk number (first chunk is 1, next is 2, etc..

The `ChunkFilter` statement has four parameters: ChunkSize, OriginalText,NewText and Formula.

The ChunkSize parameter is the size of the chunks you want to use. This may be a field, a variable, or any formula that calculates a numeric (integer) value, for example 2, 3, 5 or 10.

The OriginalText parameter is the text you want to scan. This may be a field, a variable, or any formula that calculates a text value.

The NewText parameter is the name of a field or variable for holding the output text. If this is a field, it must be a text field.

The Formula parameter is a formula that produces a text result. This formula will be calculated for each character in the OriginalText parameter. The result will be appended to the NewText field or variable. The result of this formula isn't limited to the same size as the original chunk, but may be several characters or zero characters (""). Within the formula you can use the `import()` function to retrieve the original character, or the `seq()` function to retrieve the position of the character within the text.

To see examples of the chunkfilter statement in a real application, check out the BASESIXTYFOURENCODE and BASESIXTYFOURDECODE procedures in the _InternetLib custom statement library.

The TextFilter Statement

This statement scans text and creates new text based on the original text. Unlike the `CharacterFilter` statement, the `TextFilter` statement doesn't necessarily scan the text character by character. Instead, it allows you to skip over a sequence of characters based on a formula before it starts scanning character by character again.

The `TextFilter` statement starts with the first character in the original text. It then calculates the result of a formula based on this location. This formula must start with the `longword(` function. The number in this function determines the number of characters to skip before continuing the scan. If this skip value is zero, the `TextFilter` statement stops scanning. If there is any text after the `longword(` value, that text is appended to the output text.

The `TextFilter` statement has three parameters: OriginalText,NewText and Formula.

The OriginalText parameter is the text you want to scan. This may be a field, a variable, or any formula that calculates a text value.

The NewText parameter is the name of a field or variable for holding the output text. If this is a field, it must be a text field.

The Formula parameter is a formula that produces a text result. The first four bytes of the result from this formula is treated as a binary value that tells Panorama how many characters to skip before resuming scanning (you can produce this binary value with the longword(function). Any additional text after this binary value will be appended to the NewText field or variable. Within the formula you can use the `import()` function to retrieve the original text starting from the current location, or the `seq()` function to retrieve the position of the character within the text.

Let's review an example that illustrate the operation of the TextFilter statement. This example take some text in A and then place in B a space separated array containing the word lengths. For example, if A contains

```
How long is each word in this sentence?
```

Then the result in B will be

```
3 4 2 4 4 2 4 9
```

Here is the code that performs this job:

```
textfilter A,B,
   longword(1+length(array(import(),1," ")))+
   str(length(array(import(),1," ")))+","
```

There's just one problem with this simple example - there's an even simpler way to do this that doesn't use the `TextFilter` statement:

```
arrayfilter A,B," ",str(length(import()))
```

To see examples of the chunkfilter statement in a real application, check out the FINDALLINTEXT, HARD-WRAP and HEXDUMP procedures in the _TextLib custom statement library.

Data Dictionaries

A conventional dictionary (Webster's, for example) contains a list of entries. Each entry comes as a pair — the word itself, and the definition of the word.

Panorama supports a data structure called a **data dictionary**. Like a conventional dictionary, a data dictionary contains a list of entries, and each entry comes as a pair — the entry **key** (which identifies the entry) and the entry **value** (some data associated with this entry). Sometimes these entries are referred to as **key/value pairs**. If you know the key, you can find out what the value is. A data dictionary allows you to combine any number of these key / value pairs into a single structure that can be stored in a single variable, a single field or a single procedure parameter. For example, in a database you could use a data dictionary to store additional, seldom used data that you don't want to devote an entire field to, or to store information that you didn't anticipate when you created the database. When a procedure (or custom statement) has a large and variable number of parameters, you can combine these into a data dictionary that can be passed back and forth easily (many of Panorama's Internet access statements work this way). Data dictionaries are also an excellent way to store preferences, and they are used that way by many of Panorama's wizards (for example the **Hot Key** wizard and any application that works with Generic fields (see "Generic" Fields" on page 270).)

Key/Value Pairs

Each entry in a dictionary consists of a key / value pair. The key identifies the pair, and must be unique. In other words in a single dictionary you cannot have two pairs with the same key — every pair must have its own unique name. The key can contain any text character you want, including spaces and punctuation. (The only characters that are not allowed are chr(254) and chr(255).)

The value is simply a text value that is associated with they key. The value may contain any character except for chr(254) and chr(255). Here are some examples of typical key / value pairs.

Key	Value
Color	Red
Area Code	951
Greeting	Dear
Home Page	www.myhomepage.com

These examples are for illustration purposes only — you can use just about anything you need as a key or value.

Storing a Key/Value Pair in a Dictionary

To store a key / value pair in a dictionary use the setdictionaryvalue statement.

```
setdictionaryvalue dictionary,key,value
```

The dictionary parameter tells the statement where the dictionary is located. This must be the name of a field or variable. If a variable, the variable must be defined beforehand with the `local`, `fileglobal`, or `global` statements (see "Variables" on page 1370). The variable must also have a defined value, typically "" for a brand new dictionary.

The key parameter is the identifier for this key / value pair. The value parameter is the actual data that will be stored. If this key already exists within the dictionary the original value will be replaced by the new value, otherwise a new key / value pair will be added to the dictionary.

To illustrate, here is an example that creates a new dictionary variable name ColorPalette, then fills it with three key / value pairs.

```
local ColorPalette
ColorPalette=""
setdictionaryvalue ColorPalette,"TextColor","red"
setdictionaryvalue ColorPalette,"ButtonColor","blue"
setdictionaryvalue ColorPalette,"Background","white"
```

Now the ColorPalette variable contains all three key / value pairs — TextColor / red, ButtonColor / blue and Background / white.

You can change the value of each key / value pair individually at any time. This line of code will change the ButtonColor to green.

```
setdictionaryvalue ColorPalette,"ButtonColor","green"
```

If necessary, you can also delete individual key / value pairs with the `deletedictionaryvalue` statement.

```
deletedictionaryvalue ColorPalette,"ButtonColor"
```

This statement completely removes the ButtonColor key / value pair from the ColorPalette dictionary.

Accessing Dictionary Entries

Once values have been stored in a dictionary they can be retrieved with the `getdictionaryvalue(` function, like this.

```
theColor=getdictionaryvalue(Palette,"TextColor")
```

The first parameter is the name of the field or variable that contains the dictionary, while the second parameter is the key value of the item you want to retrieve.

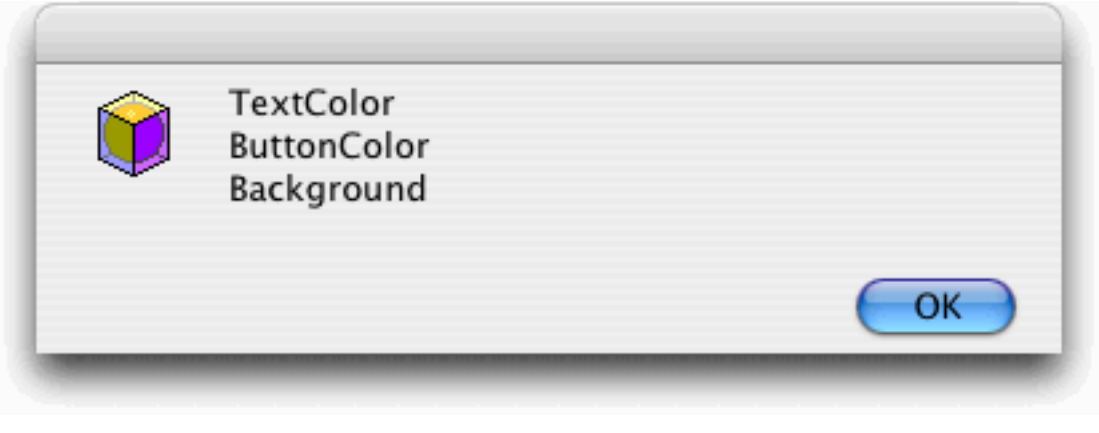
If you are not sure if a dictionary contains a particular key / value pair you can use the `dictionaryvalueexists(` function to find out. This example checks to see if a button color has been defined, and if so calls a subroutine to change the color of all buttons.

```
if dictionaryvalueexsts(Palette,"ButtonColor")
   call ChangeAllButtonColor,getdictionaryvalue(Palette,"ButtonColor")
endif
```

For a complete list of all of the entries in the dictionary use the `listdictionarynames(` function. This function has one parameter, the name of the field or variable that contains the dictionary.

```
message listdictionarynames(ColorPalette)
```

The output is a text array that lists the keys stored in the dictionary, with each item separated by a carriage return.



The dumpdictionary statement outputs a carriage return delimited array that contains both the keys and the values, separated by an equals sign.

```
local ColorPalette,PaletteDump
dumpdictionary ColorPalette,PaletteDump
message PaletteDump
```

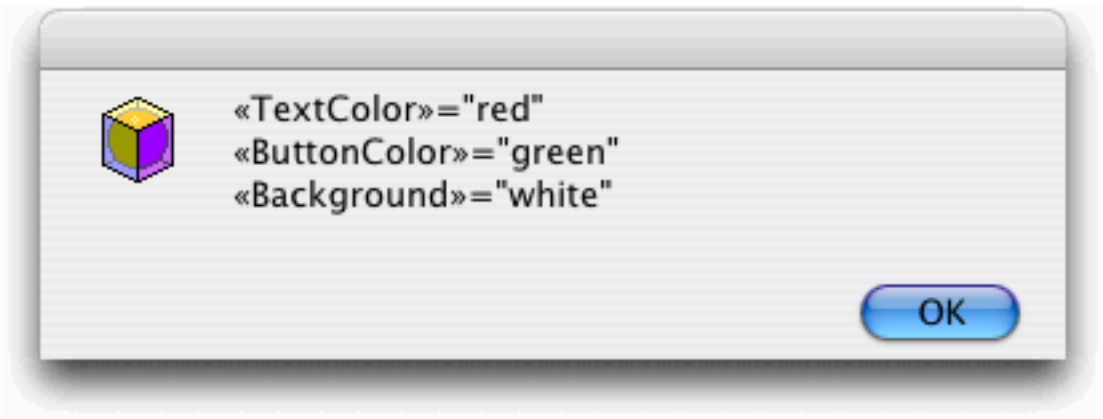
The output displayed by the message statement will look something like this:



The dumpdictionaryquoted statement is similar, but each key is surrounded by chevrons and each value is surrounded by quotes.

```
local ColorPalette,PaletteDump
dumpdictionaryquoted ColorPalette,PaletteDump
message PaletteDump
```

Here's the output:



If you have fields or variables with the same names as the keys in your database, you can use this output with the execute statement to initialize all of the fields or variables at once.

```
local ColorPalette,PaletteDump
dumpdictionaryquoted ColorPalette,PaletteDump
execute PaletteDump
```

Another Technique For Initializing a Dictionary

Earlier we showed you a method for setting up and initializing a dictionary.

```
local ColorPalette
ColorPalette=""
setdictionaryvalue ColorPalette,"TextColor","red"
setdictionaryvalue ColorPalette,"ButtonColor","blue"
setdictionaryvalue ColorPalette,"Background","white"
```

Panorama also includes a special statement for this task, `initializedictionary`. Here is the same task performed with this statement.

```
local ColorPalette
initializedictionary ColorPalette,
   "TextColor","red",
   "ButtonColor","blue",
   "Background","white"
```

You can include as many key / value pairs as you need. Using this statement instead of separate setdictionaryvalue statements has two advantages — it takes less typing, and it runs faster. Of course you can always modify individual key / value pairs later with the `setdictionaryvalue` statement.

Additional Methods for Modifying Dictionary Entries

The `appenddictionaryvalue` statement appends additional text to the value portion of an existing key / value pair. For example, the statement below will add 75% to whatever the ButtonColor is, so Blue becomes Blue 75%. (The third parameter is a separator character, which in this case is a space.)

```
appenddictionaryvalue ColorPalette,"ButtonColor"," ","75%"
```

The `changedictionaryname` statement changes the key while leaving the value the same.

```
changedictionaryname dictionary,oldkey,newkey
```

For example, you could change the `Background` key to BgColor.

```
changedictionaryname ColorPalete,"Background","BgColor"
```

The value (in this case white) will remain the same.

Looking Up a Dictionary Key Given Its Value

In some cases it may be possible to look up a key if all you know is the value. This is the reverse of the normal operation, which is to look up the value given the key. The `getdictionarykey` statement has three parameters:

```
getdictionarykey dictionary,value,key
```

The dictionary parameter is of course the name of the field or variable that contains the dictionary. The value parameter is the value you are looking for. If this value is not unique (if more than one key / value pair have the same value) then it is unpredictable what key will be returned, so you should only use this statement if you know that the value is unique. The key parameter is the name of a field or variable that will be filled with the key name (if any matching value is found, otherwise it will be filled with "").

The example below will find out what key has been set to the value white. If this dictionary is set up as it was in the previous examples, the item variable will be filled with the value `Background`. (This isn't really a great example, since more than one item could be assigned the color white. If you use this statement, be very careful that you know that there are no duplicate values in the dictionary.)

```
local item
getdictionararykey ColorPalette,"white",item
```

One other point to keep in mind about this statement — it's a bit on the slow side. It is much slower than the `getdictionaryvalue(` function.

Accessing the Internet

In this millennium no computer is an island. Panorama allows you to tap into the Internet to automatically retrieve data and graphics from the web, submit information to web servers, and to send e-mail based on database information.

Basic Web Access

In this section we'll show how to automatically receive and send data from anywhere on the web.

Fetching Web Pages

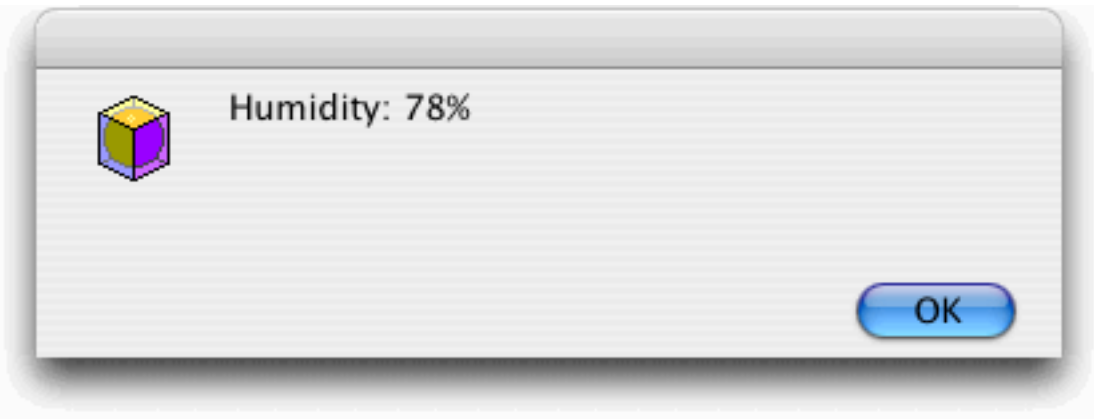
Fetching a web page is simple. The `loadurl` statement has two parameters:

```
loadurl page,url
```

The page parameter is the name of the field or variable to store that you want the downloaded page stored in. The url parameter is the web address of the page you want to load, for example "http:/ / www.weather.com". This example will download the current weather for the 92648 zip code and display the current humidity.

```
local weatherURL,weatherPage,zip
zip="92648"
weatherURL="http://http://www.wunderground.com/cgi-bin/findweather/getForecast?query="
loadurl weatherPage,weatherURL+zip
humidity=tagdata(weatherPage,"Humidity:</td><td valign=top><b>","</b>",1)
message "Humidity: "+humidity
```

When this procedure is triggered the current humidity will be displayed.

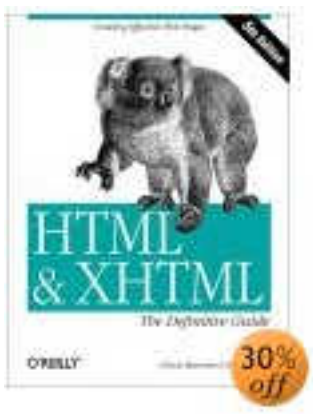
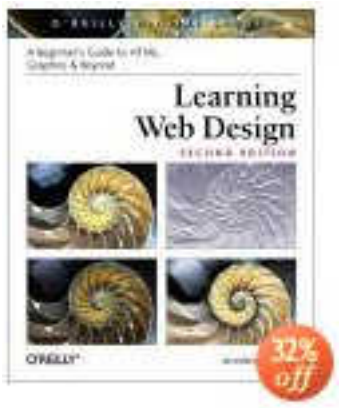


If there is a problem (for example the requested page doesn't exist, or the computer is not currently connected to the internet) the statement will place "" in the page field or variable. This revised example checks to make sure that it got a response from the Weather Underground server.

```
local weatherURL,weatherPage,zip
zip="92648"
weatherURL="http://http://www.wunderground.com/cgi-bin/findweather/getForecast?query="
loadurl weatherPage,weatherURL+zip
if weatherPage<>""
   humidity=tagdata(weatherPage,"Humidity:</td><td valign=top><b>","</b>",1)
   message "Humidity: "+humidity
else
   message "Sorry, weather information is not currently available."
endif
```

Parsing Web Pages

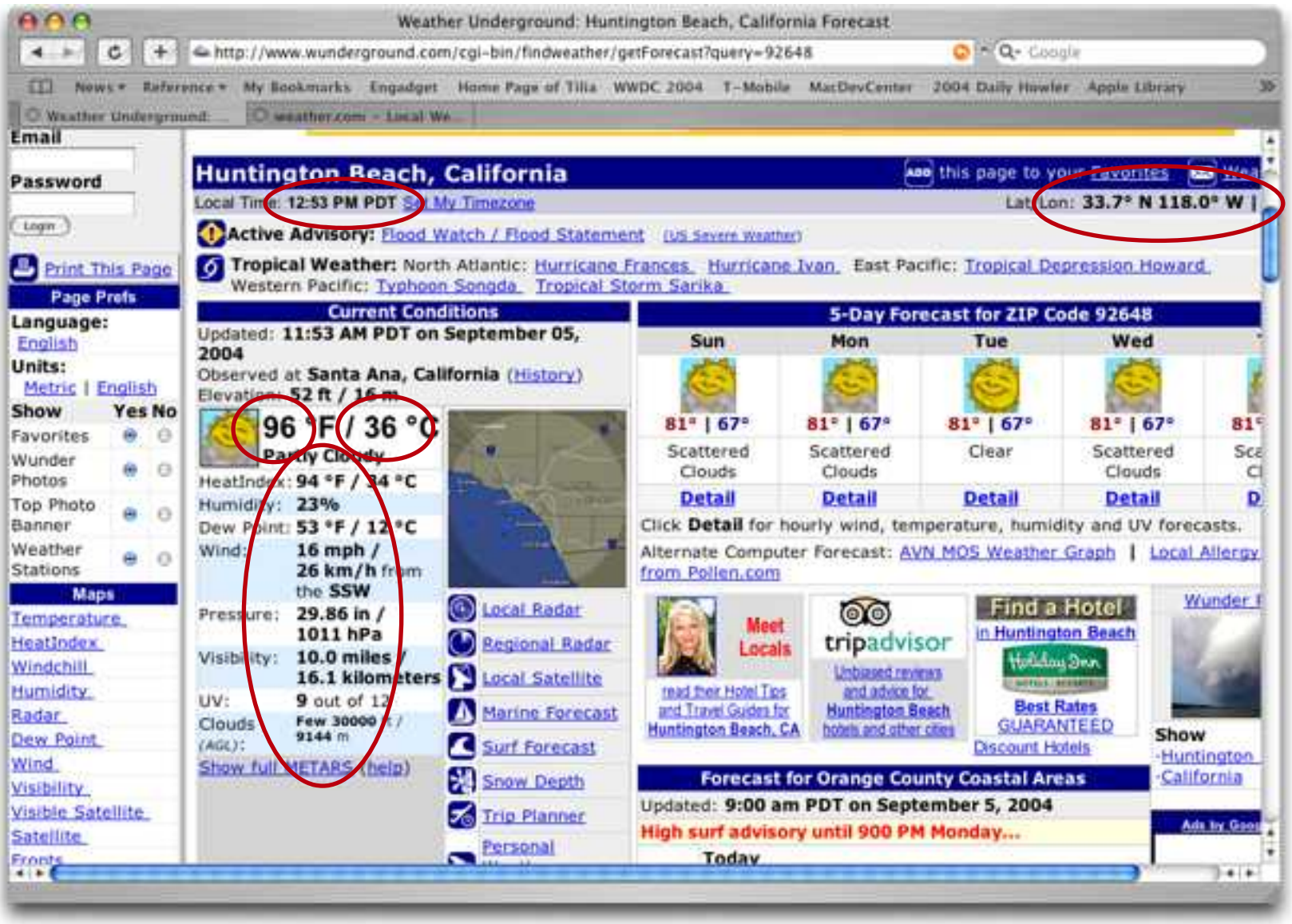
The primary language of the web is HTML (HyperText Markup Language). When you fetch a page from a web server, the page will be constructed with HTML tags like <b>, <a>, <img>, <input>, etc. In order to extract the information you need from downloaded web pages you'll need to learn at least some HTML. The details of HTML syntax and vocabulary are beyond the scope of this book, but here are some books that you may find useful.

	**HTML & XHTML: The Definitive Guide, Fifth Edition** by Chuck Musciano, Bill Kennedy • Paperback: 670 pages • Publisher: O'Reilly; 5 edition (August, 2002) • ISBN: 059600382X • Product Dimensions: 9.3 x 7.0 x 1.3 inches HTML: The Definitive Guide is aimed at beginners as well as those who have more practice in Web-page creation. The authors assume at least a basic knowledge of computers, including how to use a word processor or text editor and how to deal with files. They teach you that learning HTML is like learning any other language and that reading a book of rules can only take you so far. Readers begin writing what may be their first Web page just two pages into the book's second chapter. From there on, they provide a wide range of HTML coding to allow readers to learn from good examples. The book includes a handy "cheat sheet" of HTML codes for quick reference.
	**Learning Web Design, 2nd Edition** by Jennifer Niederst • Paperback: 488 pages • Publisher: O'Reilly; 2 edition (June 27, 2003) • ISBN: 0596004842 • Product Dimensions: 9.8 x 8.1 x 1.1 inches In Learning Web Design, author Jennifer Niederst shares the knowledge she's gained from years of web design experience, both as a designer and a teacher. This book starts from the beginning-- defining the Internet, the Web, browsers, and URLs-- so you don't need to have any previous knowledge about how the Web works. After reading this book, you'll have a solid foundation in HTML, graphics, and design principles that you can immediately put to use in creating effective web pages. In the second edition, Jennifer has updated the book to cover style sheets and reflect current web standards. She has also added exercises that help you to learn various techniques and short quizzes that make sure you're up to speed with key concepts. The companion CD-ROM contains material for all the exercises in the book. Unlike other beginner books, Learning Web Design leaves no holes in your education. It gives you everything you need to create basic web sites and will prepare you for more advanced web work.

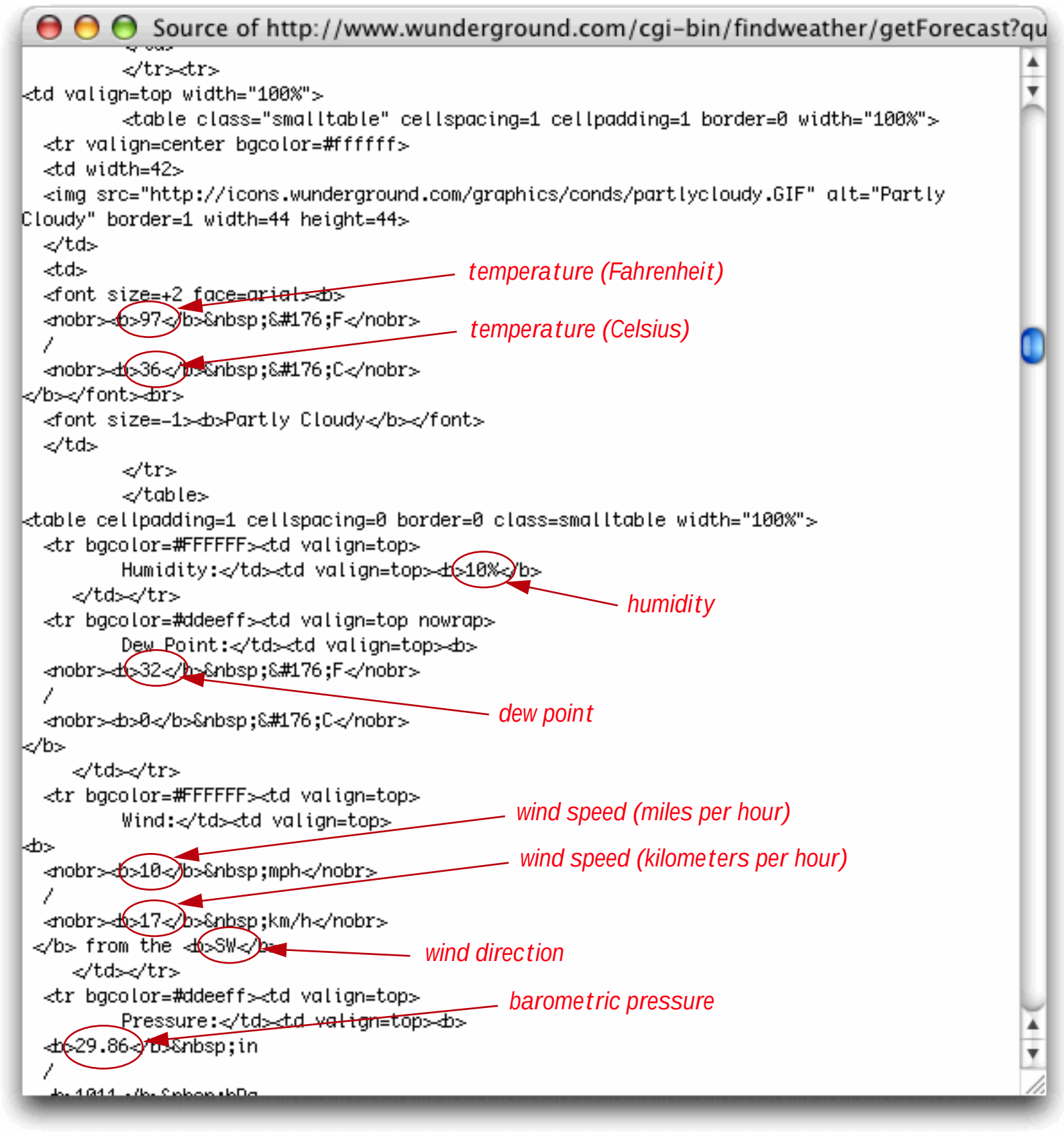
When you fetch a web page with Panorama it's usually because you want to retrieve one or more specific pieces of information within that page. Often the specific information you want is buried like gold nuggets in raw ore. Instead of using a pan or sluice to extract the nuggets, you'll use statements and functions like `search(`, `tagdata(`, `gettaglocation` and `htmltabletoarray`. Extracting information from web pages is called **web scraping**, and you can find various books and articles on the subject in your local bookstore, in programming magazines, and on the web.

Web scraping is kind of like detective work. The trick is to see through the clutter of tags and recognize the patterns that can reliably point to the nuggets of information you are interested in. The detective has tools that help solve the case — DNA, fingerprints, chemical analysis, phone records etc. But it takes imagination and skill to use these tools to connect the dots. Similarly, we can teach you how to use the tools Panorama makes available for analyzing and parsing web pages. But every page is unique and you'll need to develop your own sixth sense for teasing out the patterns in the tags.

In the previous section an example fetched the current weather for the 92648 zip code. In your web browser, this page might look something like this:



As you can see, there is all kinds of interesting data that could be extracted from this page — the current time, the latitude and longitude of this location, the current temperature (in Fahrenheit and Celsius), the humidity, wind, pressure, and many other items. The page layout is designed to make it easy to pick this information out when displayed in a browser. In the actual HTML source, however, it is much more difficult to locate this information. Here we've used the browser's **View Source** command to examine the underlying HTML code for this page:




```
          </tr><tr>
<td valign=top width="100%">
          <table class="smalltable" cellspacing=1 cellpadding=1 border=0 width="100%">
  <tr valign=center bgcolor=#ffffff>
  <td width=42>
  <img src="http://icons.wunderground.com/graphics/conds/partlycloudy.GIF" alt="Partly
Cloudy" border=1 width=44 height=44>
  </td>
  <td>
  <font size=+2 face=arial><b>
  <nobr><b>97</b> &#176;F</nobr>
  /
  <nobr><b>36</b> &#176;C</nobr>
</b></font><br>
  <font size=-1><b>Partly Cloudy</b></font>
  </td>
          </tr>
          </table>
<table cellpadding=1 cellspacing=0 border=0 class=smalltable width="100%">
  <tr bgcolor=#FFFFFF><td valign=top>
          Humidity:</td><td valign=top><b>10%</b>
     </td></tr>
  <tr bgcolor=#ddeeff><td valign=top nowrap>
          Dew Point:</td><td valign=top><b>
  <nobr><b>32</b> &#176;F</nobr>
  /
  <nobr><b>0</b> &#176;C</nobr>
</b>
     </td></tr>
  <tr bgcolor=#FFFFFF><td valign=top>
          Wind:</td><td valign=top>
<b>
  <nobr><b>10</b> mph</nobr>
  /
  <nobr><b>17</b> km/h</nobr>
 </b> from the <b>SW</b>
     </td></tr>
  <tr bgcolor=#ddeeff><td valign=top>
          Pressure:</td><td valign=top><b>
  <b>29.86</b> in
  /
```

You're first task is to visually locate the information you are looking for. In some cases this is fairly easy and you can use your browser or text editor's **Search** command to directly locate the information, for example the Humidity, Dew Point, Wind and Pressure in the text above. In other cases, however, it can be more tricky. There is no clearly identifying text near the temperature readings. In cases like this you'll have to find the information by looking at the original page for other items near the item you are looking for. For example, looking at the original page we can see that the temperature readings are just above the humidity measurement. The humidity is easily found, so looking above we quickly find the temperature.

Once you have managed to find the information visually the real trick begins — devising a means to automatically locate this information with a program. Look around for something unique that can be easily found. For example, the text Pressure: only occurs in one place in this web page, so it can be used with the `tagdata(` function to help locate the barometric pressure measurement.

```
message tagdata(weatherPage,"Pressure:","in",1)
```

As you can see, this formula has removed almost all of the text except for a few tags around the pressure.



There are several methods that could be used to isolate just the pressure itself. Noticing that none of the tags that are left contain any numbers or periods, we can use the `stripchar(` function to extract just the pressure.

```
stripchar(tagdata(weatherPage,"Pressure:","in",1),"09..")
```

There is usually more than one method that could be used to extract a particular text item. For example, here is another method that you could use:

```
tagdata(weatherPage,"Pressure:</td><td valign=top><b>"+lf()+"  <b>","</b> in",1)
```

Usually the simplest formula is the best, but not always. The goal is to create a method that is as reliable as possible, a method that will hopefully continue to work even if there are minor changes made to the page design. Of course it's always possible that the site owners will completely redesign the page or even their entire site at some point in the future. At that time you'll also have to redesign your web scraping method. Keeping your design modular will help minimize the work necessary when a redesign is necessary.

Panorama has several dozen statements and functions that can help you scrape information from web pages. Use the **Programming Reference** wizard to study these tools before you begin your first web scraping project.

Statement / Function	Description
gettaglocation	This statement finds the position of the first matching tag within some text. For example, you could find the position of the first <table> tag, or of the word Pressure.
gettaglocations	This statement finds the positions of all matching tags within some text. For example, you could find the position of every <font> tag (not just the first one.
htmldecode(	This function takes HTML text and converts any special characters in the text into standard ASCII. For example © is converted to © and & is converted to & .
htmlextractlinks	This statement extracts the links from an HTML page. The result is a carriage return separated list of the links (URL's) from this page to other pages.
htmlformitemnames	This statement returns a list of form elements (<INPUT tags, etc.). The result is a carriage return separated list of these elements. This statement is very handy for extracting any data that may be included in an HTML form.
htmltablecell(	This function extracts the data from a cell in an HTML table. Any HTML tags in the cell are removed, leaving only the actual text.
htmltablecellexists(	This function checks to see whether a cell in an HTML table exists or not. The result will be true if the cell exists, or false if it doesn't.

Statement / Function	Description
htmltablecellraw(	This function extracts the data from a cell in an HTML table. Unlike the htmltablecell(function, any HTML tags in the cell are retained.
htmltableheight(	This function calculates the height (number of rows) in an HTML table. It assumes that the table is a regular matrix (no rowspan tags).
htmltablerowraw(	This function extracts the data from a row in an HTML table. Any HTML tags in the row are retained.
htmltabletoarray	This statement converts an HTML table into a text array. The HTML table must be a simple array with no nested table inside it. You can include all of the columns in the final array, or just selected elements.
htmltablewidth(	This function calculates the height (number of rows) in an HTML table. It assumes that the table is a regular matrix (no colspan tags) and that all of the rows have the same number of columns as the top row in the table.
onespace(	This function removes any extra spaces between words, so that there is exactly one and only one space between each word.
onewhitespace(	This function removes any extra whitespace between words, making sure that there is one and only one space between each word. Other whitespace characters (carriage returns, tabs) are converted to spaces and removed if there is more than one between words.
stripchar(	This function removes characters you don’t want from a text item. You specify exactly what kinds of characters you want and don’t want included in the final output.
tagarray(	This function builds an array containing the body of all the specified tags (usually HTML tags) in the text. Each element in the array is separated from the next with the separator character (usually ¶ or ,).
tagcount(	This function counts the number of times a specified tag (usually an HTML tag) appears in the text.
tagdata(	This function extracts the body of the specified tag (usually an HTML tag) in the text.
tagend(	This function returns the ending position of the specified tag (usually an HTML tag) in the text.
tagnumber(	This function checks to see if a specified position is inside of a tag (usually an HTML tag).
tagparameter(	This function extracts the value of a tag parameter embedded in some text, where the tag parameter takes the form name=value.
tagparameterarray(	This function extracts the value of multiple tag parameter embedded in some text, where each tag parameter takes the form name=value.
tagstart(	This function returns the starting position of the specified tag (usually an HTML tag) in the text.
tagstrip(	This function removes tags (usually an HTML tag) from within a piece of text.
textafter(	This function extracts the text after the tag. The tag many be one or more characters long. If the tag doesn't occur in the text then the entire original string is returned.
textbefore(	This function extracts the text before the tag. The tag many be one or more characters long. If the tag doesn't occur in the text then the entire original string is returned.
urldecode(	This function takes standard ASCII text and converts into a format guaranteed to be legal in an internet URL (Universal Resource Locator). For example the url my%20web%20page is converted to my web page.

In addition to these statements and function that are specifically designed for HTML parsing you may also need Panorama’s general text handling tools. See “Text Formulas” on page 1212.

Fetching Images

Fetching an image is simple. The `saveurl` statement has two parameters:

```
saveurl path,url
```

The path parameter is the path and filename of the file that you want the downloaded image stored in. The url parameter is the web address of the image you want to load, for example "http://icons.wunderground.com/graphics/conds/clear.GIF". (This statement isn't limited to images, you can also save a regular HTML web page directly into a file.) This example will download the icon for "sunny and clear" from the weather underground web site.

```
saveurl "SunnyAndClear.gif","http://icons.wunderground.com/graphics/conds/clear.GIF"
```

Since the path includes only a filename (no disk or folders) the file will be stored in the same folder as the database. If you have the Enhanced Image Pack (see "Displaying Non PICT Images (Enhanced Image Pack)" on page 773) you can display this GIF image in a form.



If you want to save the image in a subfolder of the current database folder then the path must begin with a colon. This example will save the image in the Icons folder which is in the same folder as the current database.

```
saveurl ":Icons:SunnyAndClear.gif",
   "http://icons.wunderground.com/graphics/conds/clear.GIF"
```

You can also specify a complete path to store the image anywhere on your hard drive.

```
saveurl "My Disk:Documents:Web Icons:SunnyAndClear.gif",
   "http://icons.wunderground.com/graphics/conds/clear.GIF"
```

No matter where the path points to, the folder must already exist. If it doesn't you can use the `makenewfolder` statement to create it.

Relative URLs

The `loadurl` and `saveurl` statements normally require a complete URL, like this:

```
loadurl thePage,"http://www.someserver.com/somefolder/somepage.html"
```

However, once you have loaded or saved a URL you can then access other url's relative to that URL. For example, suppose that there is an image in the same folder as somepage.html. You could fetch this image with an absolute URL like this:

```
saveurl "someimage.jpg","http://www.someserver.com/somefolder/someimage.jpg"
```

However, if you have just loaded the somepage.html page with loadurl, you can then save the image with a relative URL like this:

```
saveurl "someimage.jpg","someimage.jpg"
```

You don't need to specify the complete URL because Panorama assumes that the missing part is the same as the last URL. You can also use the common ../ prefix notation to move up the server's directory structure. Again assuming that you have just loaded the somepage.html page, then these two URL's are the same.

```
"http://www.someserver.com/differentfolder/bigimage.jpg"
"../differentfolder/bigimage.jpg"
```

Remember, relative URL's are relative to the last URL saved or loaded.

Submitting Forms

Many applications call for submitting information to a web site. This is called a "form." An HTML form contains one or more fields that are submitted to the web server. When using a web browser, you fill out a form by typing and/or by selecting from pop-up's, checkboxes and radio buttons. When you have entered all of the information you press the Submit button.

When using Panorama you can submit information to a web site using the `posturl` statement.

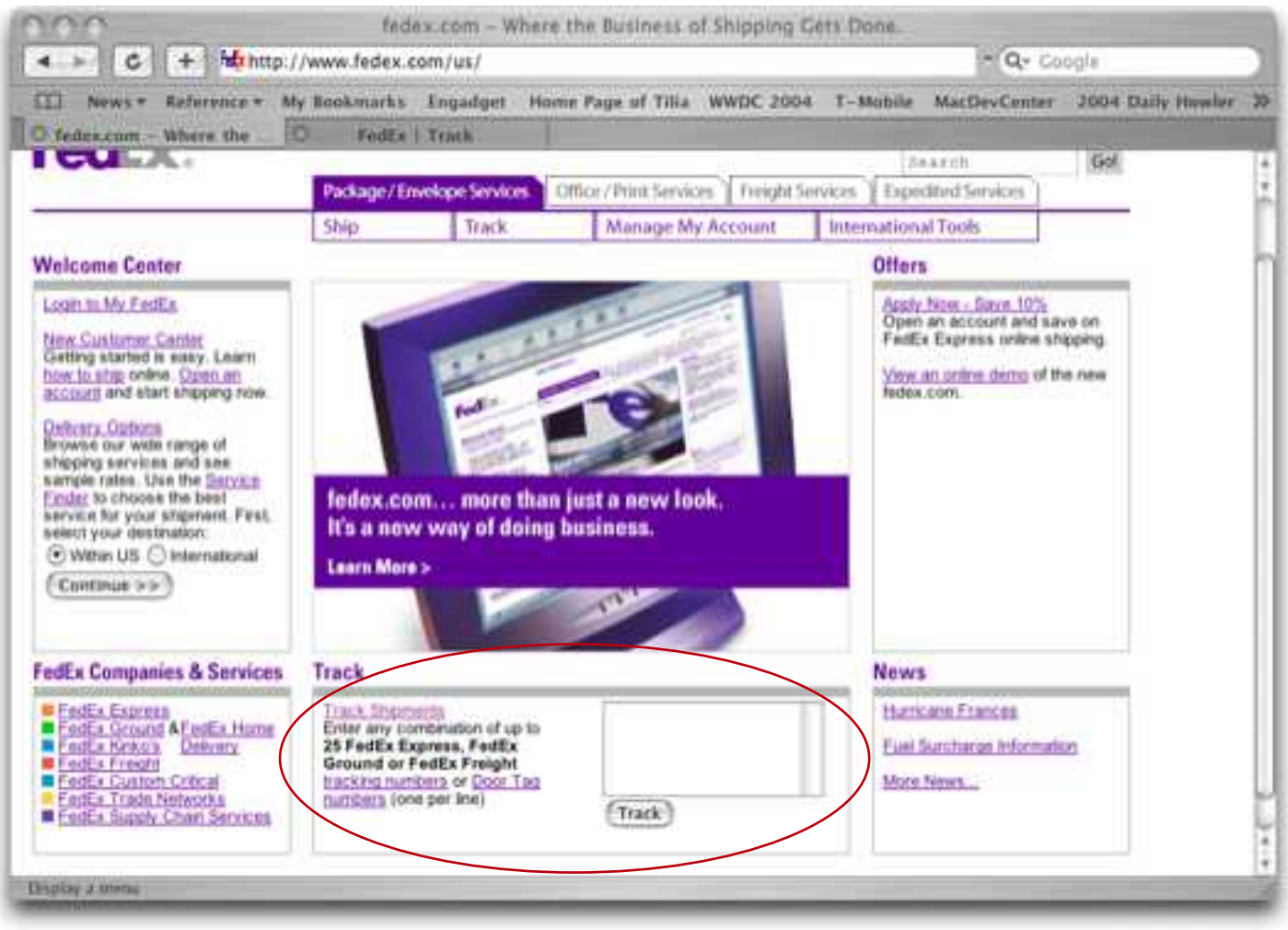
```
loadurl page,url,field1,value1,field2,value2,field3,value3 ...
```

The page parameter is the name of the field or variable to store that you want the downloaded page stored in. This is the server's response to the data you have submitted. The url parameter is the web address of the page you want to load, for example "http://www.fedex.com/cgi-bin/tracking". The field parameters are the names of the data items being submitted to the server, while the value parameters supply the actual data for each corresponding field. The statement can have as many field/value pairs as you need, but they must come in pairs. This example will look up shipment information for a FedEx tracking number. The tracking number must be in a field or variable named `trackingNumber`.

```
posturl webPage,"http://www.fedex.com/us/tracking/",
   "tracknumbers",trackingNumber,
   "action","track",
   "language","english",
   "cntry_code","us",
   "mps","y",
   "ascend_header","1"
```

In this case we are submitting six fields to the FedEx server. If we don't submit the exact fields the FedEx server is expecting to the right URL, the FedEx server will not respond. How can we find out what fields we need to submit?

To find out, first use your browser to go to the web server you want to access, and find the form you want to submit using Panorama.



Now use **View Source** to look at the actual HTMl source code and find the HTML code for the form you want to submit.

Source of http://www.fedex.com/us/

```
                                    <td class="small">
                                        <form id="tracking" method="post" action="/us/tracking/">
                                        <textarea style="font-family: monospace; font-size: 11px;
width: 150px; height:65px;" name="tracknumbers" cols="18" rows="3"></textarea>
                                        <input type="hidden" name="action" value="track">
                                        <input type="hidden" name="language" value="english">
                                        <input type="hidden" name="cntry_code" value="us">
                                        <input type="hidden" name="mps" value="y">
                                        <input type="hidden" name="ascend_header" value="1">
                                        <input type="submit" name="imageField" value="Track"
class="buttonpurple">
                                        </form>
                                    </td>
```

We've underlined the important parts of this form as far as Panorama is concerned. The action=/us/tracking parameter of the <form> tag is added to the base URL for this page to form the URL for submitting the form

```
http://www.fedex.com/us/tracking/
```

In this form the <textarea> tag contains the name=tracknumbers of the field that is used for submitting the tracking numbers. This becomes the first field parameter to our `posturl` statement (see above). All of the other five <input> tags are hidden and have fixed tags, so these make up the additional `posturl` parameters.

In this case the FedEx web authors created a nice compact form that is easy to decipher. Often forms will not be this compact - you'll have to wade through tons of additional tags and whitespace. Just let the tags guide you — start by finding the opening and closing <form> tags (and remember that some pages have more than one form on the page, so you have to make sure you have the right form). The opening <form> tag will contain the action parameter, which will have a relative or absolute URL. Then look for <input>, <textarea> and <select> tags to find out the data that needs to be submitted.

Once you've got the `posturl` statement working you are still not quite done. The server will return an HTML page that contains the information you want to retrieve (if any - sometimes you may just want to submit information, for example to add information to an on-line database). If you need to retrieve information from this response you'll have to use the web scraping techniques discussed earlier in this chapter. (However, for FedEx tracking numbers we've already done this work for you as you'll see later in this chapter.)

Cookies

A "cookie" is a small piece of information sent by a web server to store on a web client so it can later be read back from that browser. This is useful for having the browser remember some specific information, for example user id's, shopping baskets, and site preferences. By default Panorama ignores cookies, but some web pages cannot be accessed unless the web client handles cookies. If you are accessing a site that requires cookies you can turn them on with the `cookies` statement. (Note: Currently only the OS X version of Panorama supports this statement.) For most applications you can simply place the `cookies` statement with no parameters near the top of your procedure, before any `loadurl` or `posturl` statements that would require cookies to be saved.

```
cookies
loadurl …
posturl …
```

The `cookies` statement also has three optional parameters that give you more control over how cookies are processed and stored.

```
cookies folder,file,options
```

The `cookies` statement keeps the information it gets from the web server in a file on your hard disk. The folder and file parameters specify where this file should be stored and what name should be used. If the folder parameter is empty, Panorama will keep the cookie file in the same folder as the current database. If the file parameter is empty Panorama will automatically assign a name for you.

The options parameter allows you to specify one or more options. Each option is specified by a keyword from the table below.

Keyword	Description
global	By default cookies are only enabled for one database at a time. If you enable cookies for more than one database, each will keep a separate cookie file. However, if you include the global keyword in the options parameter Panorama will create a global cookie file that will be used no matter what database is currently active. If you don't specify the file and folder options this file will be called Panorama.cookie and will be stored in the Extensions:Work Files folder.
suspend	The cookies statement normally turns cookies on, but if this keyword is included cookies will be suspended. After this option is used any cookies information sent from the server will be ignored. You can re-enable cookies later, any cookie information that was already saved when cookies were suspended will be retained.
reset	The reset option completely erases any cookie information that has been saved. If you want to erase this information and turn cookies off, use this option in combination with the suspend keyword. If you want to erase global cookie information you must include the global keyword as well.

These keywords can be combined, for example "global suspend reset" to stop and erase any global cookies that have been stored.

Accessing Web Content

The previous section described how to perform general access to any web server. Panorama also includes pre-built tools designed for accessing specific types of information, for example maps, addresses, shipping information, etc. We will be adding new tools for accessing web content on a regular basis, so check our web site for updates.

Maps

The savewebmap statement makes it easy to download a map for any US address. If you have purchased the Enhanced Image Pack you can display this image on a form. This statement has eight parameters.

```
savewebmap path,zoom,street,city,state,zip,country,mapurl
```

The path parameter specifies the name and location of the image file to save. For example, if you wanted to save the file in the folder that contains the current database this parameter only needs the file name:

```
savewebmap "Acme Widgets Map.jpg", …
```

If you want to store the image file in a subfolder of the current database folder, start the path with a colon (:).

```
savewebmap ":Maps:Acme Widgets.jpg", …
```

You can also specify a full path, including the disk name, to place the saved image anywhere on your hard drive.

The zoom parameter is the map zoom level, from "1" (entire country) to "9" (a few blocks). Note: This parameter is currently ignored.

The street, city, state, zip and country parameters specify the address you want to map. Do not include the suite or apartment number in the street address. The country parameter is currently ignored, this statement only works for United States. Future versions may be more global.

The mapurl parameter must contain the name of a field or variable. When the statement is complete this field or variable will contain the url of the map that was saved. If the address was not a valid address the field or variable will be empty ("").

This example creates a .jpg file that contains a map showing the location of Disneyland.

```
local mapurl
savewebmap "Disneyland Map.jpg","1313 S. Harbor Blvd.","Anaheim","CA","92803","",mapurl
```

General Zip Code Information

The zipinfo statement queries the web to get general information about a zip code: city, state, area code, etc.

```
zipinfo zipcode,info,channel
```

The zipcode parameter must contain a 5 digit zip code. This must be a text value, not a number (e.g. "95001" not 95001).

The info parameter must be the name of a field or variable. When the statement is finished this field or variable will contain a data dictionary that contains the information loaded from the web (see “Data Dictionaries” on page 1728). Depending on the channel you have selected (see below) this data dictionary will contain one or more of the items listed below. You can retrieve individual items from this dictionary with the `getdictionaryvalue(` function.

Item	Description	Channel	
		USPS	ZipInfo.com
CITY	Primary name of the city associated with this zip code.	√	√
STATE	Two letter abbreviation of the state this zip code is in.	√	√
COUNTY	The name of the county this zip code is in.	√	√
FIPS	FIPS code.		√
AREACODE	Name or the primary area code associated with this zip code.		√
TIMEZONE	Time zone associated with this zip code.		√
DAYLIGHTSAVINGS	Yes / No		√
LATITUDE	Geographic co-ordinates of center of zip code.		√
LONGITUDE	Geographic co-ordinates of center of zip code.		√
MSA	Metropolitan statistical area		√

This statement uses a channel to retrieve the zip code information from the web. You can set the default channel using the **Channels** wizard. You can use the `channel` parameter to override the default. At the time of this writing the options for this parameter are "USPS" and "ZipInfo.com", but additional options may be available in the future. Check the Channels wizard for the latest information.

This example displays the city and state associated with a zip code.

```
local z5,zinfo
z5=""
gettext "Zip Code:",z5
zipinfo z5,zinfo
message z5+": "+getdictionaryvalue(zinfo,"CITY")+", "+getdictionaryvalue(zinfo,"STATE")
```

Street Address Information

Given a US street address, the `zipinfoplus` statement uses the US Post Office web site to find out the zip+4, carrier route, and other information.

```
zipinfoplus address1,address2,city,state,zip5,info
```

The address1 parameter is the first line of the address, address2 is the second line (optional). The city parameter is the name of the city, state is the two letter state abbreviation. The zip5 parameter is the five digit zip code, but you can leave this blank.

The info parameter must be the name of a field or variable. When the statement is finished this field or variable will contain a data dictionary that contains the information loaded from the web (see “Data Dictionaries” on page 1728). This data dictionary will contain items listed below. You can retrieve individual items from this dictionary with the `getdictionaryvalue(` function.

Item	Description
ADDRESS	Corrected street address.
CITY	Primary name of the city associated with this address.
STATE	Two letter abbreviation of the state this zip code is in.
COUNTY	The name of the county this zip code is in.
ZIP9	Nine digit zip code.
CARRIERROUTE	Postal carrier route.

This example checks an address in the database to see if it is correct. If it is completely invalid, an error message is displayed. If it is not formatted correctly by USPS rules, the database is corrected.

```
local zinfo,correctAddress
zipinfoplus Address,"",City,State,Zip,zinfo
correctAddress=getdictionaryvalue(zinfo,"ADDRESS")
if correctAddress=""
   message "Address is not valid!"
   rtn
endif
if correctAddress=Address rtn endif
Address=correctAddress
```

White Pages

The `querywhitepages` statement queries the web to look up white page information.

```
querywhitepages first,last,city,state,zip,url,listings,info,channel
```

The first parameter is the first name of the person you are looking for. You can leave this blank if you don’t know the first name, or use a partial first name (for example the first initial). The more detail you can supply, the more likely it is that you will be able to find the person you are looking for.

The last parameter is the last name of the person you are looking for. You cannot leave this blank, but you can use a partial name. Again, however, the more you supply the more likely you’ll get the information you’re looking for.

The city is the city you are looking for. You can leave this blank if you don’t know, or if you are supplying the zip code.

The state is the two letter state abbreviation. You can leave this blank, but unless the name you are looking for is extremely unusual this is generally not a good idea (unless you supply a zip code).

The zip code can be supplied instead of the city and state.

The url parameter is normally set to "". If it is not blank, it must be a URL received from a previous invocation of the `querywhitepages` statement. This allows you to download additional listings that match your search criteria, and will be described in more detail below.

The listings parameter must be the name of a field or variable. When the statement is complete, this field or variable will contain a carriage return separated list of names, addresses and phone numbers. Each line contains information for a single person. Within in each line are seven tab separated fields:

First	Last	Address	City	State	Zip	Phone

We've found that sometimes entries earlier in the list contain more information that entries farther down. To get the most detail about a specific person, narrow the search as much as possible.

The info parameter must be the name of a field or variable. When the statement is complete, this field or variable will contain a data dictionary with the following items, which you can retrieve with the `getdictionaryvalue(` function:

Item	Description
FOUND	This is the total number of people found. This number may be more than the actual number in the list that is returned, this list normally contains no more than 10 lines no matter how many people were found. If this number is -1 then the total number of people found is a very large, but unknown number.
INDEX	This is the index number of the first person in the list of returned names. This value is always 1 unless you have used the url parameter to request additional names (beyond the first ten).
MOREPEOPLEURL	If more than ten people are found, only the first ten are returned in the list. In that case, this value will contain a URL that can be used to request additional names that match this search. You can pass this URL back to this statement using the url parameter (in that case, leave the first, last, city, state and zip parameters blank). Each time you request additional names you can check this data dictionary entry to see if more names are available, and if so, you can request them.

This statement uses a channel to retrieve the white page listings from the web. You can set the default channel using the **Channels** wizard. You can use the `channel` parameter to override the default. At the time of this writing the only option for this parameter is "Switchboard", but additional options may be available in the future. Check the **Channels** wizard for the latest information.

To see examples of how the `querywhitepages` statement can be used see the **White Pages** wizard that comes with Panorama.

FedEx Shipment Tracking

The `fedextracking` statement queries the FedEx web site to determine the status of a shipment.

```
fedextracking trackingnumber,shipinfo
```

The trackingnumber parameter must contain a FedEx tracking number.

The shipinfo parameter must be the name of a field or variable. When the statement is complete, this field or variable will contain a data dictionary with some of the following items (which items are available depends on the current package status), which you can retrieve with the `getdictionaryvalue(` function:

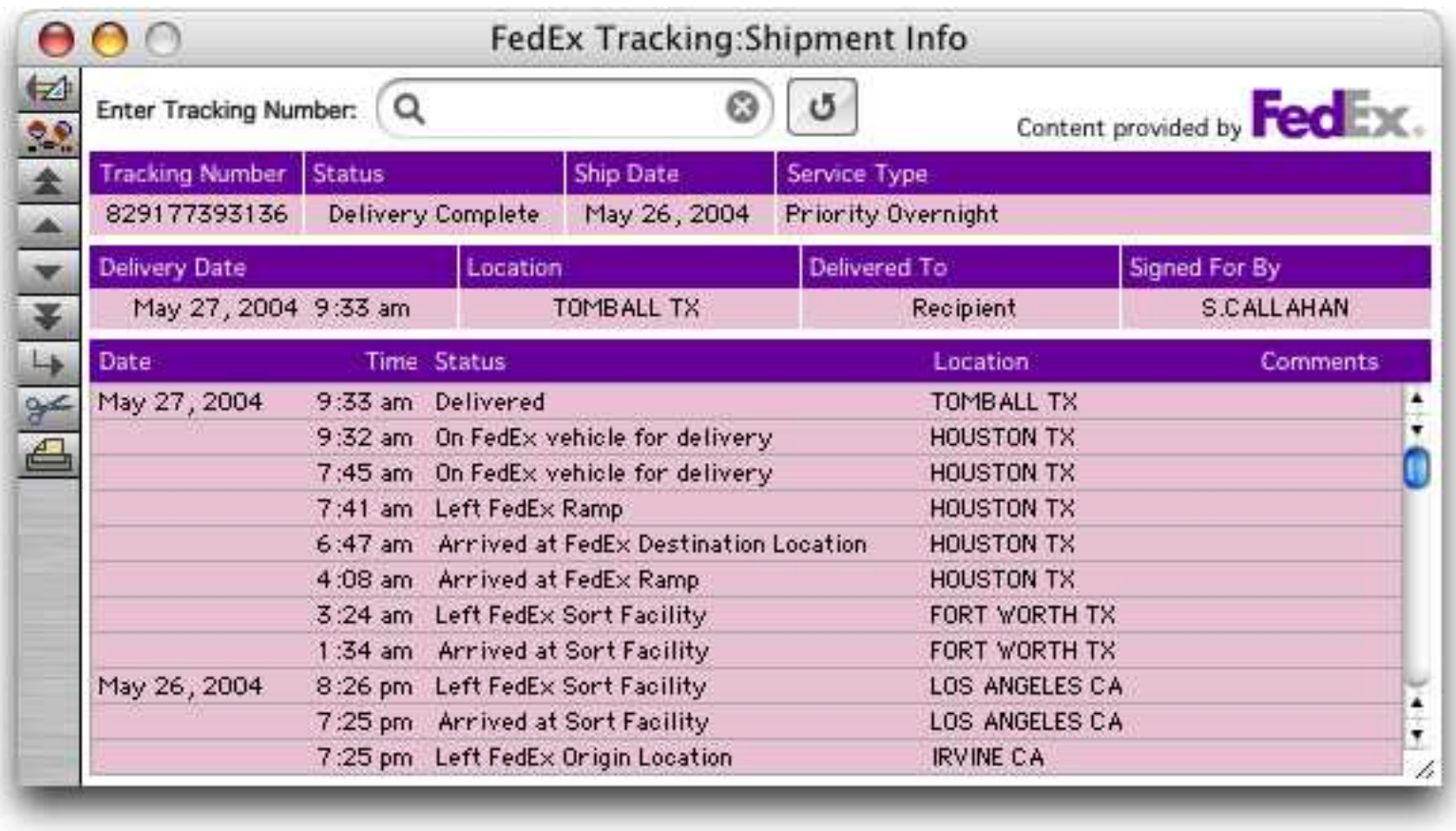
Item	Description
TRACKING NUMBER	This is the tracking number, a duplicate of the first parameter
STATUS	Status of this shipment. Possible values are "Invalid", "Not Shipped", "In Transit" and "Delivery Complete".
SHIP DATE	This is the date the package was picked up by FedEx.
ESTIMATED DELIVERY DATE	If the package is in transit, this is the estimated delivery date and time.

Item	Description
DELIVERY LOCATION	The package destination (city and state).
DELIVERED TO	If the package has been delivered, the location or person it was delivered to (for example "Recipient", "Front Desk" or "Receiving Dock").
SERVICE TYPE	"Priority Overnight", etc.
SIGNED FOR BY	If the package has been delivered, the name of the person that signed for it.
DELIVERY DATE/TIME	If the package has been delivered, the date and time that occurred.
SHIPMENT HISTORY	This is a carriage return separated array showing the progress of the shipment. Each line contains 5 columns separated by tabs: Date, Time, Action/Status, Location and Comments.

This example displays the status of a FedEx shipment.

```
local tracknum,trackinfo
tracknum=""
gettext "Tracking number:",tracknum
fedextracking tracknum,trackinfo
message "Shipment status: "+getdictionaryvalue(trackinfo,"STATUS")
```

For more a more detailed example of this statement see the **FedEx Tracking** wizard.



Controlling Web and E-Mail Clients

In addition to directly accessing web pages (see above) and sending e-mail (see below) Panorama can control your default web and e-mail client to view pages and set up outgoing e-mail.

Displaying a Web Page

To display a web page use the `shellopendocument` statement.

```
shellopendocument url
```

The url must be a complete url, beginning with http://.

```
shellopendocument "http://www.crazyapplerumors.com"
```

The HTML document will open in your default browser. On Windows this is normally Internet Explorer, on MacOS X it is normally Safari.



Displaying a Web Page on a Local Hard Drive

To display a web page that is on your local hard drive (not on the web) use the `openanything` statement.

```
openanything folder,file
```

This statement will display an HTML file named MyLife.html. This file is located on My Disk in the My Stuff folder.

```
openanything folder("My Disk:My Stuff:"),"MyLife.html"
```

Displaying a Map

Use the `openwebmap` statement to display the map for any US address.

```
openwebmap street,city,state,zip,country
```

The street, city, state, zip and country parameters specify the address you want to map. Do not include the suite or apartment number in the street address. The country parameter is currently ignored, this statement only works for United States. Future versions may be more global.

This example opens the default web browser and displays a map showing the location of Disneyland.

```
openwebmap "1313 S. Harbor Blvd.","Anaheim","CA","92803",""
```

Here's the resulting map:



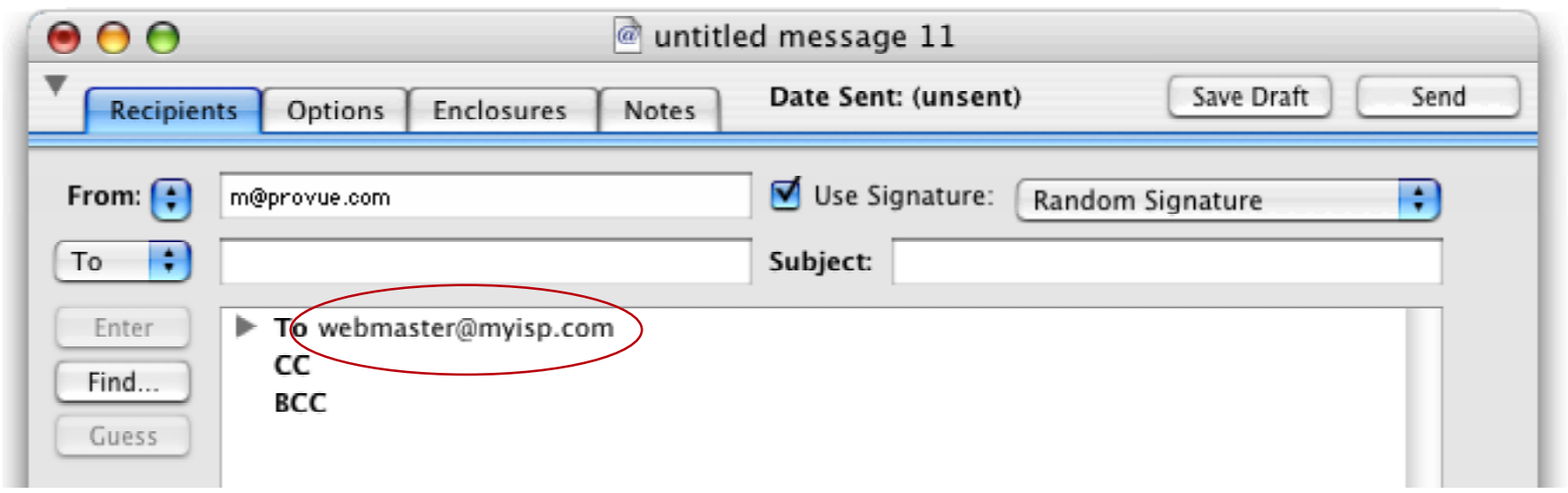
Remember that you can also automatically save this map to a disk file with the `savewebmap` statement (see "Maps" on page 1742).

Sending an E-Mail

In addition to displaying web pages, the `shellopendocument` statement can also be used to initiate the process of creating a new e-mail. Instead of starting the url with http://, start it with mailto:, followed by the e-mail address. This example creates a new message addressed to webmaster@myisp.com.

```
shellopendocument "mailto:webmaster@myisp.com"
```

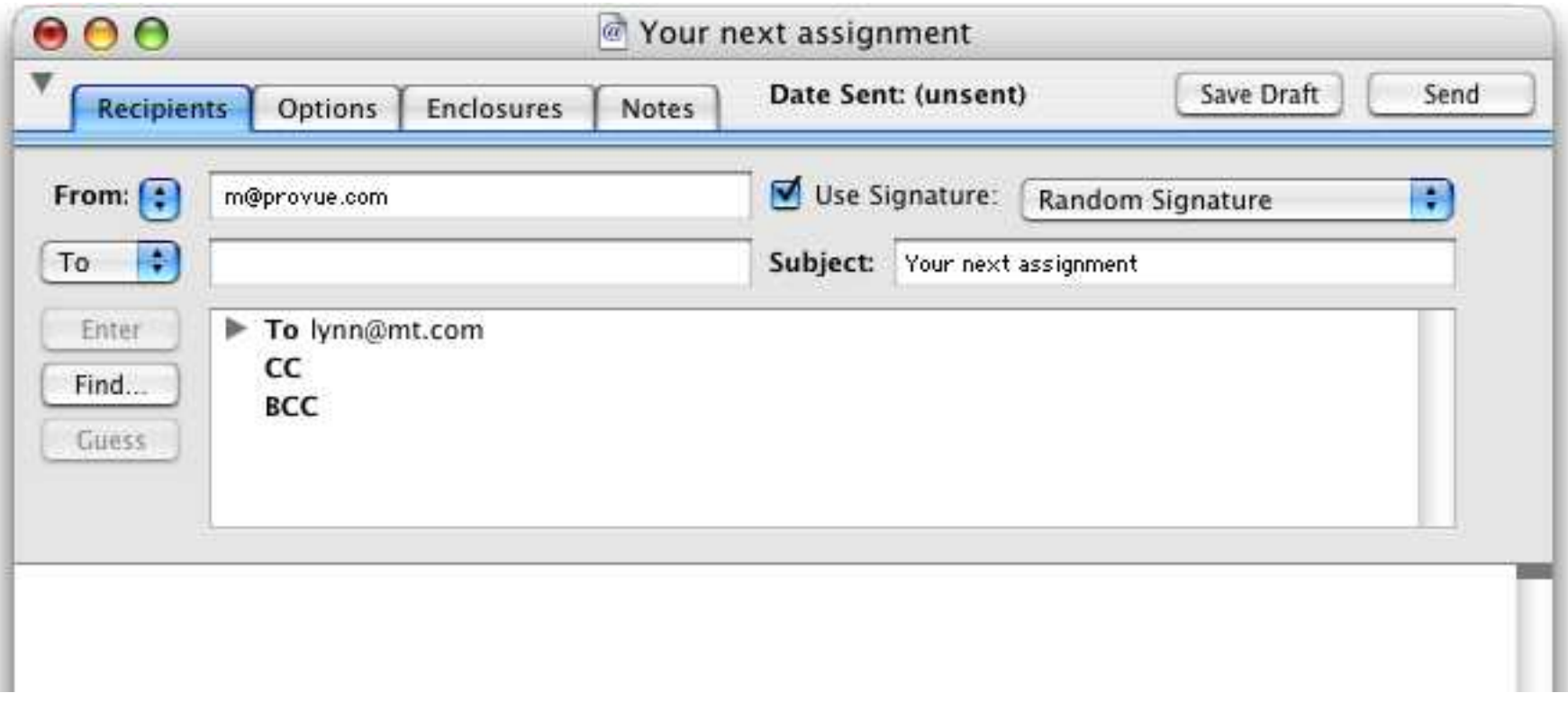
Running a procedure that contains this statement will open your default e-mail client and create a new e-mail message addressed to the specified person.



To specify a subject for the new e-mail, add a suffix beginning with ?. (Note: Not all e-mail clients support this suffix.) This example creates a new message addressed to lynn@mt.com with a subject of Your next assignment.

```
shellopendocument "mailto:lynn@mt.com?Your next assignment"
```

Running a procedure that contains this statement will open your default e-mail client and create a new e-mail message addressed to the specified person and with the subject already typed in.



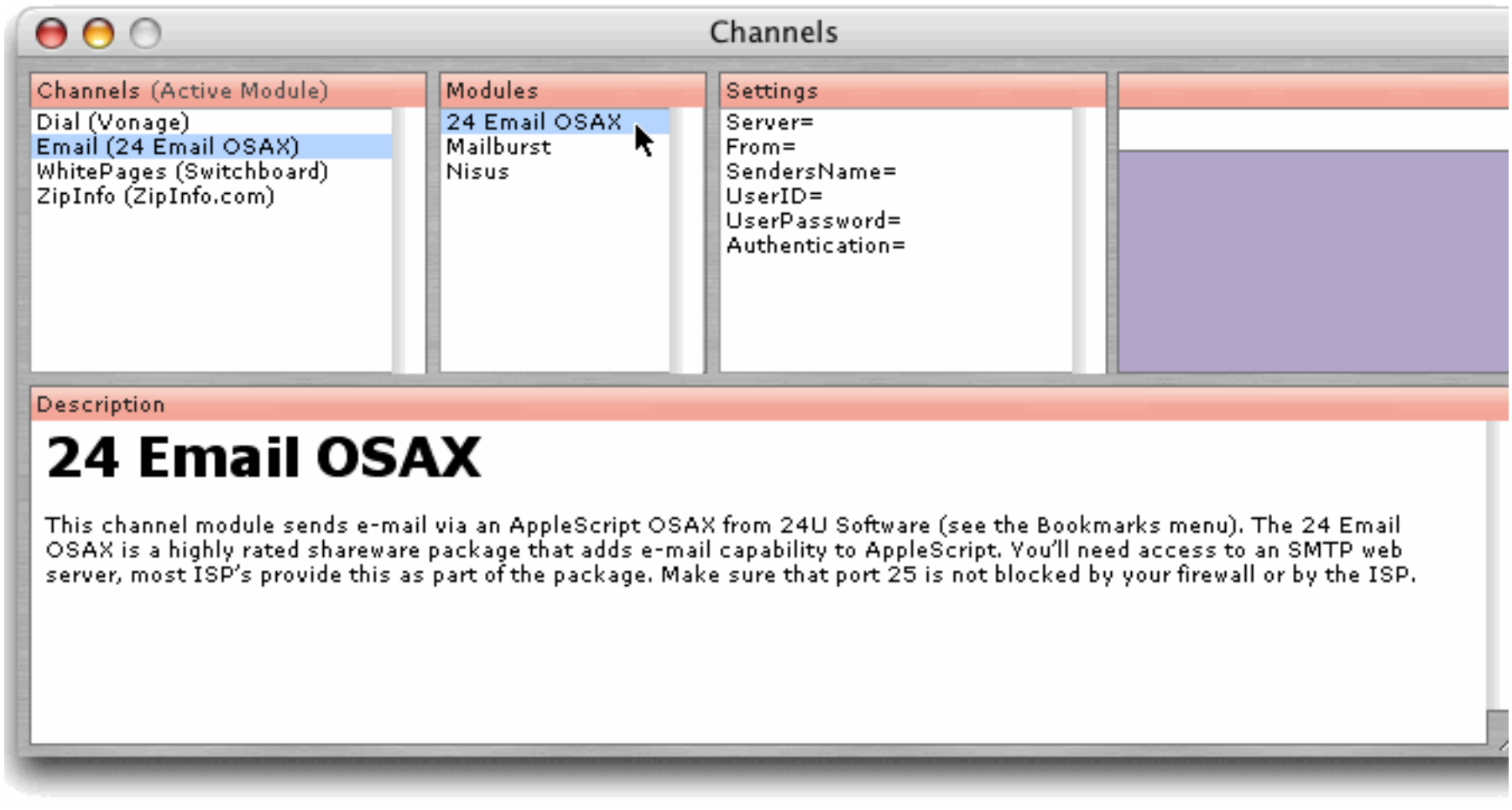
All you need to do is type in the body of the message and press the **Send** button.

Sending E-Mail

Panorama cannot send e-mail by itself. However, through the use of a channel Panorama can interface with external software to send e-mails automatically. Panorama comes with several ready to use channels, and it is also possible to write your own channels.

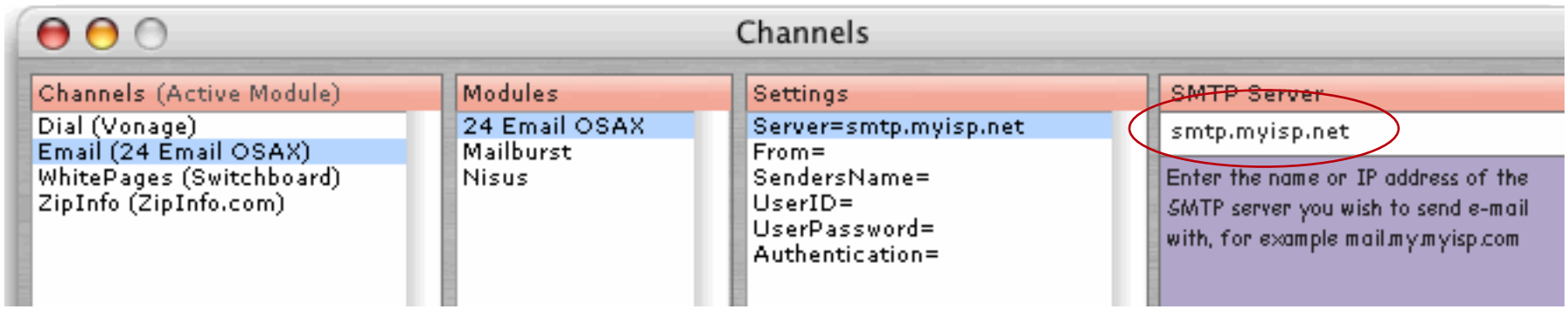
Channel Configuration

Before sending e-mail you'll need to select and configure the e-mail channel you are planning to use. This is done with the **Channels** wizard, which is in the **Preferences** submenu.

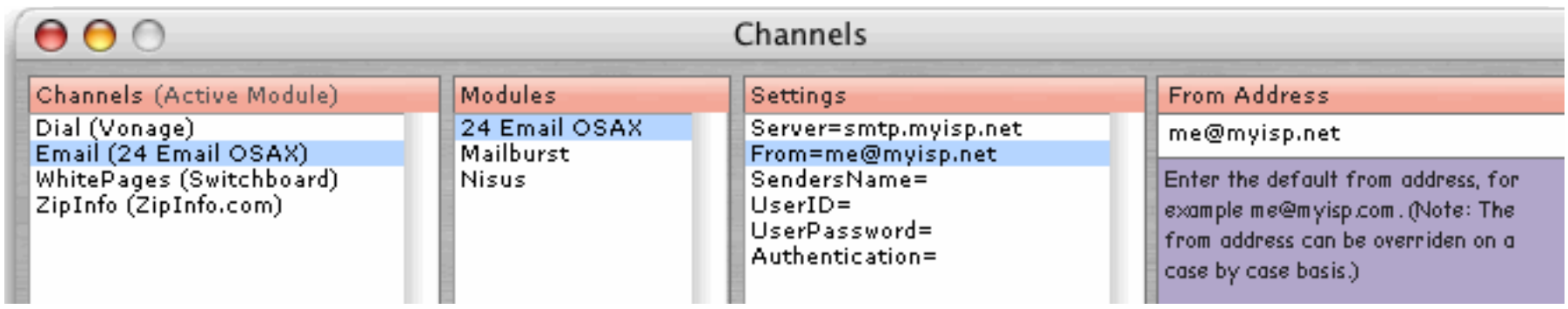


The first step is to click on E-mail in the left column, then select the module you want to use. In this case we've picked 24 Email OSAX, which uses a software package, called, surprisingly enough *24 Email OSAX* (this must be purchased separately from another company). You'll want to pick the module for whatever external software you already have or plan to acquire.

The exact setup details depend on the module you choose. We'll show how to set up for *24 Email OSAX* as an example. Since *24 Email OSAX* communicates directly with an SMTP server, you'll need the same SMTP account information you used when setting up your e-mail client. To start, click on Server= and type in the URL for the SMTP server your ISP provides.



Now click on From= and type in your e-mail address.



Repeat to type in any additional information required by your ISP, then close the **Channels** wizard when you are done.

Sending a single e-mail

Once you have set up a channel you are ready to send e-mails. To send a single e-mail to a single recipient use the `sendoneemail` statement.

```
sendoneemail from,to,subject,body
```

The from parameter is the sender's email address. If blank, the default from address will be used. However, some channels may ignore the from address specified by this channel and always use the default from address.

In addition to the actual email address you can also specify the senders actual name after a comma, for example joe@acme.net,Joe Smith. However, not all channels will use this information. It will be ignored if the channel doesn't support this option.

The to parameter is the recipient's e-mail address, the address of the person you are sending the e-mail to.

The subject parameter is a line of text that will become the subject of the e-mail.

The body parameter is the main body of the e-mail message. Panorama only supports text messages, it doesn't support attachments.

This example sends an e-mail to joe@aol.com.

```
sendonemail "","joe@aol.com","Next Week's meeting",
   "Dear Joe"+¶+¶+"Please bring the new XRF-89 specs with you to the meeting."
```

Sending multiple e-mails

Panorama has two statements for sending multiple copies of the same e-mail to different recipients — `sendbulkemail` and `sendarrayemail`. Use `sendbulkemail` if the recipient e-mail addresses are in a database field. Use `sendarrayemail` if the recipient e-mail addresses are in a text array.

The `sendbulkemail` statement has five parameters.

```
sendbulkemail from,database,recipient,subject,body
```

The from parameter is the sender's email address. If blank, the default from address will be used. However, some channels may ignore the from address specified by this channel and always use the default from address.

In addition to the actual email address you can also specify the senders actual name after a comma, for example joe@acme.net,Joe Smith. However, not all channels will use this information. It will be ignored if the channel doesn't support this option.

The database parameter is the name of the database to extract the e-mail addresses from. Use "" for the current database.

The recipient parameter is the name of the field that contains the e-mail addresses you want to send this message to.

The subject parameter is a line of text that will become the subject of the e-mail.

The body parameter is the main body of the e-mail message. Panorama only supports text messages, it doesn't support attachments.

This example sends an e-mail to every selected person in the Contacts database.

```
sendbulkemail "","Contacts",Email,"Next Week's Meeting",
   "Don't forget, the XRF-89 specification meeting is next Tuesday at 10:30 AM. "+
   "Joe Wilson will be presenting the new specifications."
```

The `sendarrayemail` statement sends multiple e-mails to recipients listed in a comma separated array.

```
sendarrayemail from,to,subject,body
```

The from parameter is the sender's email address. If blank, the default from address will be used. However, some channels may ignore the from address specified by this channel and always use the default from address.

In addition to the actual email address you can also specify the senders actual name after a comma, for example joe@acme.net,Joe Smith. However, not all channels will use this information. It will be ignored if the channel doesn't support this option.

The to parameter is a comma separated array containing recipient's e-mail addresses.

The subject parameter is a line of text that will become the subject of the e-mail.

The body parameter is the main body of the e-mail message. Panorama only supports text messages, it doesn't support attachments.

This example sends an e-mail to three recipients: joe, jack and sue.

```
sendarraymail "","joe@aol.com,jack@earthlink.com,sue@hotmail.net",
   "Next Week's Meeting",
   "Don't forget, the XRF-89 specification meeting is next Tuesday at 10:30 AM. "+
   "Joe Wilson will be presenting the new specifications."
```

Panorama also has one additional statement for sending e-mail to multiple recipients — `sendemail`. This statement is a bit more complicated but gives you more precise control over your e-mail, for example you can send e-mail to some recipients as CC and some as BCC. For more information on using this statement see the **Programming Reference** wizard.

Programming Graphic Objects on the Fly

Graphic objects are usually manipulated manually in Graphics Mode. A procedure can also be programmed to perform manipulations on graphic objects. For example a procedure can move or change the size of objects, change the color of objects, change the font of text objects, etc. When a procedure manipulates graphic objects it does so directly in Data Mode (not in Graphic Mode).

Although procedures can manipulate graphics, they cannot do everything that you can do manually in Graphics Mode. A procedure cannot create new objects or delete existing objects, and it cannot make any change that would change the amount of memory used by the graphic object. (For example a procedure cannot change the text in an auto-wrap text object.)

Basics of Graphic Object Programming

Working with graphic objects is a two step process. First, the program must identify the object or objects that need to be modified. This is called selecting the objects. The process is similar to manually selecting a graphic object by clicking on it or dragging around it. (However, unlike objects that are selected manually in graphics mode, no handles appear at the corners of objects that are selected by a procedure.) Of course a procedure cannot click on an object, so it has to use one or more properties of the object to identify it. For example, you can select an object based on its name, based on its position, based on its color, or based on a number of other attributes (or combinations of attributes).

Once at least one object is selected the procedure can use the `changeobjects` statement to change the object or objects. The `changeobjects` statement can change one property of an object (or objects) at time. If you need to change more than one property (for example position and color) you'll need to use more than one `changeobjects` statement.

Selecting an Object by Name

If an object has a unique name within a form, the simplest way to select the object is using that name. Any graphic object can have a name that can be used to identify that object. To give an object a name first select the object (in Graphics Mode), then use the Object Name command in the Edit menu or click on the object name in the Graphic Control Strip (see "Object Type / Object Name" on page 533). (The Graphic Control Strip can also display the name of the object when you click on the object.)

To select an object by name use the `object` statement. This statement has one parameter—the name of the object to select. For example, to select an object named Swiss Cheese use this procedure:

```
object "Swiss Cheese"
```

The parameter must match the object name exactly, including upper and lower case. If there is more than one object named Swiss Cheese this statement will select the one farthest to the back. (To select multiple objects at a time use the `selectobjects` statement, described in the next section.)

If the user is currently editing using a SuperObject (text editor or word processor) the procedure can find out the name of the object being edited with the `info("editing")` function. You can use this function with the `object` statement to select the object, and possibly change one or more of its attributes. (Note: This function scans all the objects in the current form, so if you are going to use it over and over again it might be faster to use it once and copy the name into a local variable, then use the local variable.)

Selecting Multiple Objects

To select multiple objects at once use the `selectobjects` statement. This statement scans the objects in the current form and selects some of them based on a formula. The formula can use the `objectinfo(` function (see the next section) to examine each object as it is scanned and decide whether or not the object should be selected. For example, the statement below scans the current form and selects all objects that are pure blue.

```
selectobjects objectinfo("color")=rgb(0,0,65535)
```

To quickly select all of the objects in the current form, use the `selectallobjects` statement. To quickly unselect all of the objects in the current form, use the `selectnoobjects` statement.

No matter how the objects are selected, they will remain selected until you close the form, switch the form into graphics mode, or perform another statement that selects objects. Once one or more objects are selected you can use the `changeobjects` statement to change many of the attributes of the object (more on this later in this chapter).

Getting Information About Individual Objects

A procedure does not have eyes to see the graphic objects in a form. Instead of eyes or a camera, the procedure uses the `objectinfo(` function to gather information about graphic objects. The `objectinfo(` function has one parameter—the type of information you want to collect (object location, size, color, font, etc.).

Like a camera, the `objectinfo(` function must be "pointed" at a specific object. There are several statements that can "point" at a specific object, including the `object` statement and the `selectobjects` statement (see previous sections).

Here is an example of the object statement and `objectinfo(` function in action. This example finds out the font and text size of the object named MySpecialButton.

```
local myFont, mySize
object "MySpecialButton"
if info("found")
   myFont=objectinfo("font")
   mySize=objectinfo("textsize")
endif
```

There are about a dozen types of information the `objectinfo(` function can extract from an object.

objectinfo(function	Description

objectinfo("rectangle")	This option returns the dimensions (location and size) of the object. The dimensions are returned using the rectangle data type (see "Rectangles" on page 1290). The rectangle is returned in form relative co-ordinates (see "XYTOXY(" on page 5910).
	
	The example below selects the data cell(s) the user clicked on. The procedure uses the **inrectangle(** function to determine which object (if any) was clicked on. (Note: Presumably this procedure would be triggered by a push button which covers the data cell objects.)
	
	`local hitPt, hitField`
	`hitPt=xytoxy(info("click"),"Screen","Form")`
	`selectobjects`
	`inrectangle(hitPt,objectinfo("rectangle")) and`
	`  objectinfo("type") beginswith "Data Cell:"`
	`objectnumber 1`
	`hitField=objectinfo("type")[":",-1][-2,-1]`
	`if hitField="" stop endif`
	`field hitField`
	`editcell`
	
	If the user did click on a data cell, the procedure activates the cell.

<table>
<tr><th>objectinfo(function</th><th>Description</th></tr>
<tr><td>objectinfo("name")</td><td>
This option returns the name of the object. This is the name that is assigned by the Object Name dialog (in the Edit menu, or Graphic Control Strip). The two lines shown below are basically equivalent.
<pre>object "Swiss Cheese"
selectobjects objectinfo("name")="Swiss Cheese"</pre>
These two statements are not completely equivalent. If there is more than one object named Swiss Cheese the selectobjects statement will select all of them. The object statement will select only the one closest to the back.
The objectinfo("name") function can be used in a formula to decode object names. For example, if a form contains rows and columns you can give each cell a name like c1r1, c1r2, … c4r12. Using the objectinfo("name") function a procedure could decode these names and select a specific column or row. For example, here is a procedure that selects the third column:
<pre>selectobjects objectinfo("name")[1,2]="c3"</pre>
Here is another procedure that selects the seventh row:
<pre>selectobjects objectinfo("name")[3,4]="r7"</pre>
By carefully assigning object names you can often simplify the design of your procedures tremendously. Look for patterns that you can take advantage of.
</td></tr>
<tr><td>objectinfo("type")</td><td>
This option returns the type of the object. The object types are:
<pre>Rectangle
Rounded Rectangle
Oval
Line
Picture
Auto-Wrap Text
Click Text
Data Cell:<field>
Button
Chart
Flash Art
Flash Sound
Balloon Help
SuperObject:<type of SuperObject>
Tile:<type of tile>
Group</pre>
To see a complete list of SuperObject types see the objectinfo("custom") function (Page 1759). To see a complete list of tile types see the objectinfo("tile") function (Page 1758).
Here is a procedure that uses this function to select all of the rectangles in the current form.
<pre>selectobjects objectinfo("type")="Rectangle"</pre>
</td></tr>
<tr><td>objectinfo("font")</td><td>
This option returns the font for this object. If the option does not have a font (an oval, for example) this option will return empty text.
This procedure converts all Courier text to American Typewriter.
<pre>selectobjects objectinfo("font")="Courier"
changeobjects "font","American Typewriter"</pre>
</td></tr>
</table>

objectinfo(function	Description

objectinfo("textsize")	This option returns the size of the text displayed by this object. If the object does not have a text size (an oval, for example) this option will return zero. Here is a procedure that selects all objects with a text size greater than 18 points (1/4 inch) and changes them to American Typewriter.
	
	`selectobjects objectinfo("textsize")>18`
	`changeobjects "font","American Typewriter"`

objectinfo("textstyle")	This option returns the text style of text displayed by the object. The text style is a number that is created by adding up the numbers for each individual style from the table below. For example, for bold italic text the style will be 3.
	
	`0 Plain`
	`1 Bold`
	`2 Italic`
	`4 Underline`
	`8 Outline`
	`16 Shadow`
	
	The example below selects all italic objects and then changes the color of the italic objects to blue.
	
	`selectobjects objectinfo("textstyle") and 2`
	`changeobjects "color",rgb(0,0,65535)`

objectinfo("color")	This option returns the color of the object (see “Colors” on page 1294). For example, this procedure selects all objects with brightness below 50%, then changes it to a minimum brightness of 50%.
	
	`selectobjects brightness(objectinfo("color"))<32768`
	`changeobjects "color",`
	`  hsb(`
	`  hue(objectinfo("color")),`
	`  saturation(objectinfo("color")),`
	`  32768`
	`  )`

objectinfo("selected")	This option returns whether or not the object is already selected (by a previous **selectobjects** statement).

objectinfo("locked")	This option returns true or false depending on whether or not the object is locked. (A locked object cannot be modified when in graphic editing mode, see “Locked Objects” on page 575.) The example below selects all rectangles that are not locked.
	
	`selectobjects objectinfo("type")="Rectangle"`
	`  and not objectinfo("locked")`

objectinfo("expandable")	This option returns true or false depending on whether or not the object can expand depending on the amount of data to be printed (see “Variable Height Records” on page 1111).
objectinfo("expandshrink")	This option returns true or false depending on whether or not the object can expand or shrink depending on the amount of data to be printed (see “The Expand/Shrink Option” on page 1118).

<table>
<tr><th>objectinfo(function</th><th>Description</th></tr>
<tr><td>objectinfo("text")</td><td>This option returns the text in auto-wrap text objects or click text objects (see “Fixed Text Objects” on page 587). When used with any other type of object it returns empty text.

This example changes all text objects that contain the word Phone to italic.

<pre>selectobjects objectinfo("text") contains "Phone"
changeobjects "textstyle",
        objectinfo("textstyle") or 2</pre></td></tr>
<tr><td>objectinfo("fillpattern")</td><td>This option returns the fill pattern of the object (if any, see “Fill Pattern” on page 521). Patterns are 8 bytes of raw data (see “Raw Binary Data” on page 1296). Here are some formulas for typical patterns.

<pre>Formula   Pattern

radix(16,"FFFFFFFFFFFFFFFF") black
radix(16,"00000000000000")   white
""   none (transparent)
radix(16,"AA55AA55AA55AA55") 50% gray pattern
radix(16,"8822882288228822") light gray
radix(16,"DD77DD77DD77DD77") dark gray
radix(16,"8888888888888888") vertical lines
radix(16,"FF000000FF000000") horizontal lines
radix(16,"FF888888FF888888") cross-hatch</pre>
This list shows only a few of the possible patterns—there are literally millions of patterns that can be created.</td></tr>
<tr><td>objectinfo("linepattern")</td><td>This option returns the line pattern of the object (if any, see “Line Pattern” on page 523). Patterns are 8 bytes of raw data (see “Raw Binary Data” on page 1296). Here are some formulas for typical patterns.

<pre>Formula   Pattern

radix(16,"FFFFFFFFFFFFFFFF") black
radix(16,"00000000000000")   white
""   none (transparent)
radix(16,"AA55AA55AA55AA55") 50% gray pattern
radix(16,"8822882288228822") light gray
radix(16,"DD77DD77DD77DD77") dark gray
radix(16,"8888888888888888") vertical lines
radix(16,"FF000000FF000000") horizontal lines
radix(16,"FF888888FF888888") cross-hatch</pre>
This list shows only a few of the possible patterns—there are literally millions of patterns that can be created</td></tr>
<tr><td>objectinfo("linewidth")</td><td>This option returns the line width of the object (if any, see “Line Width” on page 525). The line width is a number from 1 to 8, or zero if this object does not support a line width.</td></tr>
</table>

objectinfo(function	Description
objectinfo("tile")	This option returns the type of tile (if the object is a tile, otherwise it returns ""). You can also get this information using the objectinfo("type") function. The tile type will be one of the names in this list. "1st page Header" "1st page Header (Center) " "1st page Header (Right) " "Header" "Header (Center) " "Header (Right) " "Table Header " "Group Header (1) " "Group Header (2) " "Group Header (3) " "Group Header (4) " "Group Header (5) " "Group Header (6) " "Group Header (7) " "Group Sidebar (1) " "Group Sidebar (2) " "Group Sidebar (3) " "Group Sidebar (4) " "Group Sidebar (5) " "Group Sidebar (6) " "Group Sidebar (7) " "Data " "Summary (1) " "Summary (2) " "Summary (3) " "Summary (4) " "Summary (5) " "Summary (6) " "Summary (7) " "Table Footer " "Footer" "Footer (Center) " "Footer (Right) " "Left Margin " "Right Margin " "BackDrop " "Spacer" "Data (Page 2) " "Data (Page 3) " "Data (Page 4) " "Data (Page 5) " "Data (Page 6) " "Data (Page 7) " "Data (Page 8) " "Data (Page 9) " "Top Margin" "Data Overflow" "Last page Header" "Last page Header (Center)" "Last page Header (Right)" For more information on report tiles see “Working with Tiles” on page 1050.

objectinfo(function	Description
objectinfo("custom")	This option only works with SuperObjects. It returns the type of SuperObject (see list below). This information can also be obtained by using the objectinfo("type") function. "Text Display" "Text Editor" "PgCell" (word processor) "Super Flash Art" "Push Button" "Flash Art Push Button" "Data Button" "Sticky Push Button" "Flash Art Data Button" "PopUp Menu" "Text List" (scrollable list) "Scroll Bar" "Super Matrix" "Auto Grow" (elastic form)
objectinfo("ID")	This option returns a unique number that can be used to identify this object later. The number is valid as long as the form is not edited in graphics mode. The **objectid** statement can use this unique ID number to re-locate this object later (see “Object ID Values” on page 1763).
objectinfo("count")	This option applies not to a specific object, but to the entire form. It counts the number of currently selected objects. For example, this example displays the number of rectangles in the current form. selectobjects objectinfo("type") contains "rectangle" message "This form contains "+ str(objectinfo("count"))+ " rectangles."
objectinfo("boundary")	This option applies not to a specific object, but to the entire form. It calculates the minimum rectangle that encloses all of the selected objects.

Modifying Selected Objects

A program can use the `changeobjects` statement to modify certain attributes of selected objects (see “CHANGEOBJECTS” on page 5097). The `changeobjects` statement has two parameters:

```
changeobjects how,data
```

The how parameter specifies how the objects should be adjusted—a new font, a new color, new position, etc. The data parameter specifies the new object attributes—"Palatino", rgb(5000,12000,48000), rectangle(100,120,410,240), etc. The following table describes each of the options available.

Option	Description
rectangle	This option changes the rectangle of all selected objects. This example moves all selected objects down and to the right by 36 pixels (1/2 inch). changeobjects "rectangle", rectangleadjust(objectinfo("rectangle"), 36,36,36,36)

Option	Description

fieldname	This option applies only to data cells. It changes the field associated with the any selected data cells. The example below changes all Qty cells to Price cells (Qty1 to Price1, Qty2 to Price2, etc.)
	
	`selectobjects objectinfo("fieldname") match "Qty?"`
	`changeobjects "fieldname",`
	`  "Price"+objectinfo("fieldname")[4,-1]`

font	This option changes the font of selected objects. Non text objects will not be affected. The example below sets the font of all data cells to Times Roman.
	
	`selectobjects objectinfo("type")="Data Cell"`
	`changeobjects "font","Times Roman"`

textsize	This option changes the text size of selected objects. Non text objects will not be affected. The example below reduces the text size of all data cells by 3 points, down to a minimum of 9 points.
	
	`selectobjects objectinfo("type")="Data Cell"`
	`changeobjects "textsize",`
	`  maximum(9,objectinfo("textsize")-3)`

textstyle	This option changes the text size of selected objects. Non text objects will not be affected. The text style is a number that is created by adding up the numbers for each individual style from the table below. For example, for bold italic text the style will be 3.
	
	`0 Plain`
	`1 Bold`
	`2 Italic`
	`4 Underline`
	`8 Outline`
	`16 Shadow`
	
	The example below sets the style of all data cells to bold italic.
	
	`selectobjects objectinfo("type")="Data Cell"`
	`changeobjects "textstyle",3`

color	This option changes the color of the selected objects (see “Colors” on page 1294). The example procedure below changes any pure red objects on the current form into blue objects.
	
	`selectobjects objectinfo("color")=rgb(65535,0,0)`
	`changeobjects "color",rgb(0,0,65535)`

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>fillpattern</td><td>
This option changes the fill pattern of the selected objects (see “Fill Pattern” on page 521). Patterns are 8 bytes of raw data (see “Raw Binary Data” on page 1296). Here are some formulas for typical patterns.
<pre>
Formula   Pattern

radix(16,"FFFFFFFFFFFFFFFF") black
radix(16,"00000000000000")   white
""   none (transparent)
radix(16,"AA55AA55AA55AA55") 50% gray pattern
radix(16,"8822882288228822") light gray
radix(16,"DD77DD77DD77DD77") dark gray
radix(16,"8888888888888888") vertical lines
radix(16,"FF000000FF000000") horizontal lines
radix(16,"FF888888FF888888") cross-hatch
</pre>
This list shows only a few of the possible patterns—there are literally millions of patterns that can be created. The example procedure below sets the Check Background object to a dark gray pattern.
<pre>
selectobjects objectinfo("name")="Check Background"
changeobjects "fillpattern",radix(16,"DD77DD77DD77DD77")
</pre>
</td></tr>
<tr><td>linepattern</td><td>
This option changes the line pattern of the selected objects (see “Line Pattern” on page 523). Patterns are 8 bytes of raw data (see “Raw Binary Data” on page 1296). Here are some formulas for typical patterns.
<pre>
Formula   Pattern

radix(16,"FFFFFFFFFFFFFFFF") black
radix(16,"00000000000000")   white
""   none (transparent)
radix(16,"AA55AA55AA55AA55") 50% gray pattern
radix(16,"8822882288228822") light gray
radix(16,"DD77DD77DD77DD77") dark gray
radix(16,"8888888888888888") vertical lines
radix(16,"FF000000FF000000") horizontal lines
radix(16,"FF888888FF888888") cross-hatch
</pre>
This list shows only a few of the possible patterns—there are literally millions of patterns that can be created. The example procedure below sets the Check Background object to a 50% gray pattern (which will display a dotted line).
<pre>
selectobjects objectinfo("name")="Check Border"
changeobjects "linepattern",radix(16,"AA55AA55AA55AA55")
</pre>
</td></tr>
<tr><td>linewidth</td><td>
This option changes the line width of the selected objects (see “Line Width” on page 525). The example below sets the line width of the Check Background object to 4 pixels.
<pre>
selectobjects objectinfo("name")="Check Border"
changeobjects "linewidth",4
</pre>
</td></tr>
<tr><td>expandable</td><td>
This option allows a procedure to make an object expandable (so that it will expand when printed in a custom report, see “Variable Height Records” on page 1111). Use -1 to make the selected objects expandable and 0 to make the objects fixed height. This example makes every auto-wrap text object on the current form expandable.
<pre>
selectobjects objectinfo("type")="Auto-Wrap Text"
changeobjects "expandable",-1
</pre>
</td></tr>
</table>

Option	Description

expandshrink	This option allows a procedure to make an object expandable/shrinkable (so that it will expand or shrink as necessary when printed in a custom report, see "The Expand/Shrink Option" on page 1118). Use -1 to make the selected objects expand/shrinkable and 0 to make the objects fixed height. This example makes every auto-wrap text object on the current form expand/shrinkable.
	`selectobjects objectinfo("type")="Auto-Wrap Text"`
	`changeobjects "expandshrink",-1`

lock	This option can lock or unlock a graphic object (see "Locked Objects" on page 575). Use -1 to lock the selected objects and 0 to unlock the objects. This example locks every object on the form.
	`selectallobjects`
	`changeobjects "lock",-1`

The `changeobjects` statement is designed to work closely with the `selectobjects` statement and the `objectinfo(` function. See the previous section ("Modifying Selected Objects" on page 1759) for several additional examples of how these statements can work together.

Getting Information About Selected Objects

The `selectobjects` statement can select dozens or even hundreds of graphic objects. To get information about one of these objects use the `objectnumber` statement. This statement has one parameter, a number which specifies which selected object you want to get information about. After the `objectnumber` statement the procedure should have one or more assignment statements that use the `objectinfo(` function to get information about the object.

Suppose there are 5 objects selected. To find out the name of the first selected object (closest to the back) use the procedure:

```
local objName
objectnumber 1
objName=objectinfo("name")
```

To find out the name of the last selected object (closest to the front) use the procedure:

```
local objName
objectnumber 5
objName=objectinfo("name")
```

If there are not enough selected objects to fulfill the request, the `info("found")` function will return false. In other words, if there are only 3 objects selected and you try to get information about number 7, `info("found")` will be set to false. The procedure below takes advantage of this feature to build a list of all the names of all the SuperObjects in the current form.

```
local objectNames,X
X=1
objectselect objectinfo("type") beginswith "SuperObject"
loop
   objectnumber X
   stoploopif (not info("found"))
   objectNames=sandwich("",objectNames,¶)+objectinfo("name")
   X=X+1
while forever
```

You can also use the `objectinfo("count")` function to find out how many objects are selected. Here is another procedure that does the same job as the last example but in a slightly different way.

```
local objectNames,maxObject,X
X=1
objectselect objectinfo("type") beginswith "SuperObject"
maxObject=objectinfo("count")
loop
   stoploopif X>maxObject
   objectnumber X
   objectNames=sandwich("",objectNames,¶)+objectinfo("name")
   X=X+1
while forever
```

The example below finds the name of the top object the user clicked on. The procedure uses the `inrectangle(` function to determine which object (if any) was clicked on. (Note: Presumably this procedure would be triggered by a transparent push button which covers all the other objects. This button is not counted as the object the user clicked on.)

```
local hitPt, hitObject
hitPt=xytoxy(info("click"),"Screen","Form")
selectobjects inrectangle(hitPt,objectinfo("rectangle")) and
   objectinfo("type") ≠ "Button"
objectnumber objectinfo("count
hitObject=objectinfo("name")
```

Object ID Values

Each graphic object has a unique ID value that can be used to identify that object. The ID value is a number that is guaranteed to be unique for that object only. (However, if you edit the form in graphic editing mode the ID value may change.)

A procedure can use the `objectinfo("ID")` function to find out the ID of an object. The procedure can store the ID value and later use it with the `objectid` statement to re-select the object. For example, here is a procedure that finds and stores the ID of an object the user clicks on (see previous section for more details on this example.)

```
global hitObject
local hitPt
hitPt=xytoxy(info("click"),"Screen","Form")
selectobjects inrectangle(hitPt,objectinfo("rectangle")) and
   objectinfo("type") ≠ "Button"
objectnumber objectinfo("count")
hitObject=objectinfo("ID")
```

Later another procedure can re-select this object with a single statement.

```
objectid hitObject
```

You can also use the object ID value to determine the relative front-to-back order of two or more objects. Objects that are closer to the front will have higher ID values, while objects that are closer to the back will have lower ID values.

Redrawing an Object

It's usually not necessary to explicitly redraw an object (or objects), but if it is necessary you can do so with the `drawobjects` statement. This statement has no parameters, and must be preceded by the `object`, `selectobjects`, or `objectid` statements. This example redraws the object called Swiss Cheese (see "Selecting an Object by Name" on page 1753).

```
object "Swiss Cheese"
drawobjects
```

This example redraws all of the objects in the current form that are displayed in the font Courier (see “Selecting Multiple Objects” on page 1753).

```
selectobjects objectinfo("font")="Courier"
drawobjects
```

The `drawobjects` statement normally redraws objects in the current window, but it may be used with “magic windows” to redraw objects in other open windows (see “‘Magic” Windows” on page 1583).

Dragging a Rectangle

Dragging is the standard interface technique for moving items from one place to another. A Panorama procedure can allow a user to drag a gray rectangle from one spot to another spot. When the user releases the mouse, the procedure can be programmed to move an item or to copy data to another spot or another database (drag and drop).

The key to dragging is a special statement called `draggraybox`. This statement is designed to be used in a procedure that is triggered by a transparent button with the click/release option turned off. When the user presses on the button, the procedure is triggered immediately. The procedure calculates size and location of the original rectangle to drag around, as well as the limits to where this rectangle can be dragged. Then the `draggraybox` statement takes over. As long as the user continues to hold down the mouse a gray box will follow the mouse around. (Note: Mac OS X only allows you to drag within the current window.) When the user lets up on the mouse button the `draggraybox` statement tells the procedure the final position of the box. The procedure can then take whatever action is appropriate (moving a graphic object, copying data, etc.)

The `draggraybox` statement has four parameters. The first three of these parameters are rectangles, the fourth is a number.

```
draggraybox dragrectangle,limits,slop,axis
```

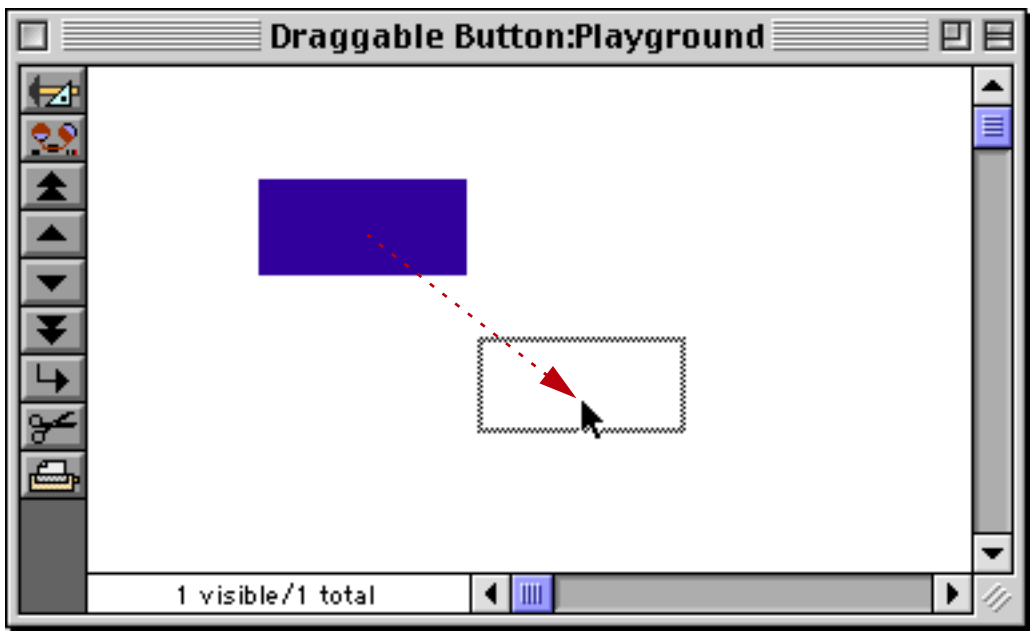
The dragrectangle parameter is the original co-ordinates of the rectangle the user will drag around. Often these co-ordinates are the same as the co-ordinates for the button the user pressed on. (Note: the co-ordinates for this rectangle, along with the next two, are relative to the upper left hand corner of the screen.) This parameter should be a field or variable (not a more complex formula) because after the user has released the mouse Panorama will copy the final co-ordinates into this parameter.

The limits parameter is the co-ordinates of a boundary rectangle that defines how far the dragrectangle can be dragged in each direction. For example if you don’t want the user to be able to drag the box outside of the current window you should supply the co-ordinates of the current window for limits. If the limits parameter is empty (`""`) there will be no limit on how far the rectangle can be dragged. (Note: When using Mac OS X you cannot drag outside of the current window, even if the limit parameter is empty.)

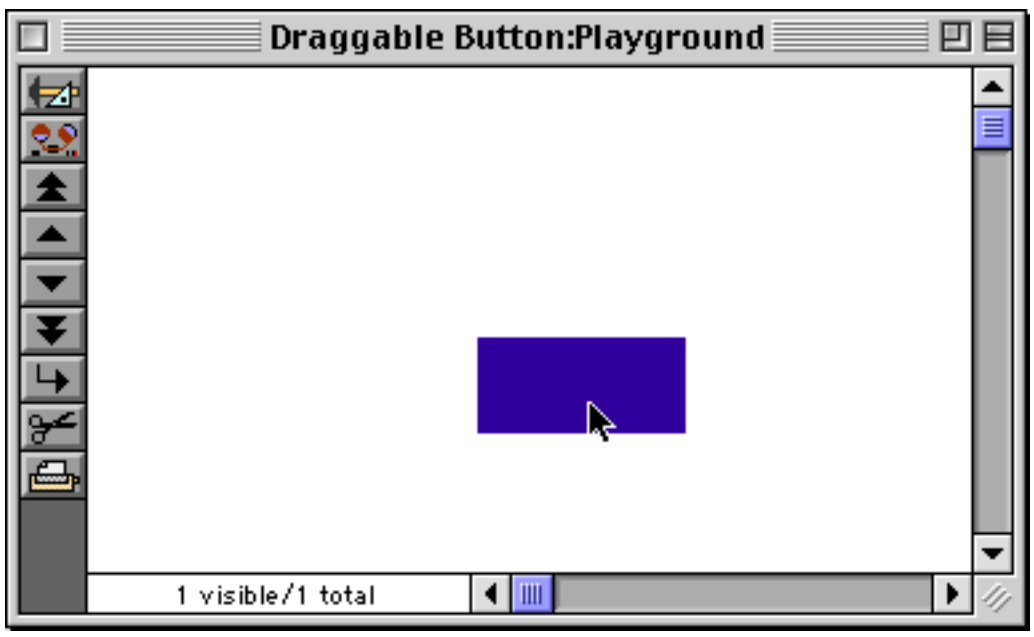
The slop parameter is the co-ordinates of a boundary rectangle past the limits boundary. If the user drags the mouse beyond the slop rectangle the gray rectangle will disappear completely (until the user drags back inside the slop rectangle). If the slop parameter is empty (`""`) it will be the same as the limits boundary rectangle.

The axis parameter allows the procedure to restrict the direction the rectangle can be dragged to either horizontal or vertical. If the axis parameter is 0 the rectangle can be dragged in any direction. If the axis is 1 the rectangle can only be dragged horizontally. If the axis is 2 the rectangle can only be dragged vertically.

Here is a procedure that allows the user to drag a button around the window.



When the user releases the mouse, the procedure moves the button to the new location.



(Remember, this procedure should be triggered by a button with the click/release option turned off.)

```
❶ local drag,insidewindow
❷ drag=info("buttonrectangle")
❸ selectobjects xytoxy(drag,"s","f")=objectinfo("rectangle")
❹ insidewindow=rectangle(
   rtop(info("windowrectangle"))+20,
   rleft(info("windowrectangle"))+26,
   rbottom(info("windowrectangle"))-16,
   rright(info("windowrectangle"))-16)
❺ draggraybox drag,insidewindow,info("windowrectangle"),0
❻ if drag="" stop endif
❼ drag=xytoxy(drag,"s","f")
❽ changeobjects "rectangle",drag
```

This example is a bit complicated, so let's take a look at it statement by statement.

❶ We start by allocating the variables we need: drag and insidewindow.

❷ This statement finds the original location of the button, relative to the upper left hand corner of the screen.

❸ The selectobjects statement selects the button the user clicked on. It identifies the button by it's location on the form. If there are any other objects with the exact same dimensions, they will be selected (and moved) also.

❹ This assignment calculates the inside dimensions of the window. It takes the raw window dimensions and moves the top down by 20 pixels (for the drag bar), the left side over by 26 pixels (for the tool palette), and the bottom and right sides in by 16 pixels (for the scroll bars). This will define the limits beyond which the button cannot be dragged.

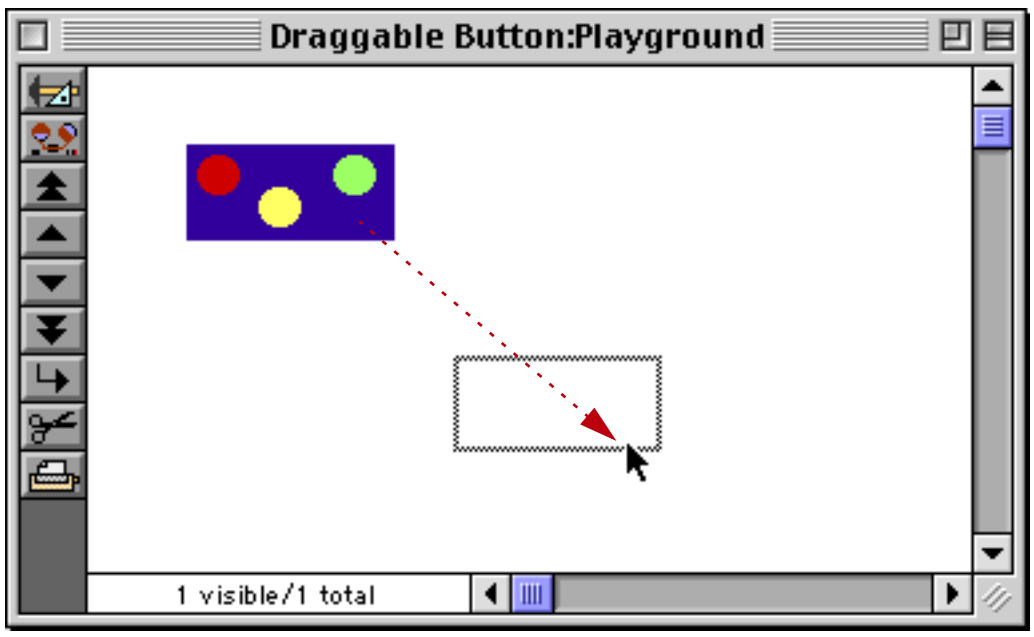
❺ Here's where dragging actually takes place. The parameters define the starting point for the drag (the original button location), the limits of dragging (the inside boundary of the window) and the limits beyond which the gray box completely disappears (the outside boundary of the window). The final parameter indicates that the button may be dragged in any direction.

❻ If the user dragged the button completely out of the window the drag variable will be set to "". In that case the procedure simply stops without moving anything.

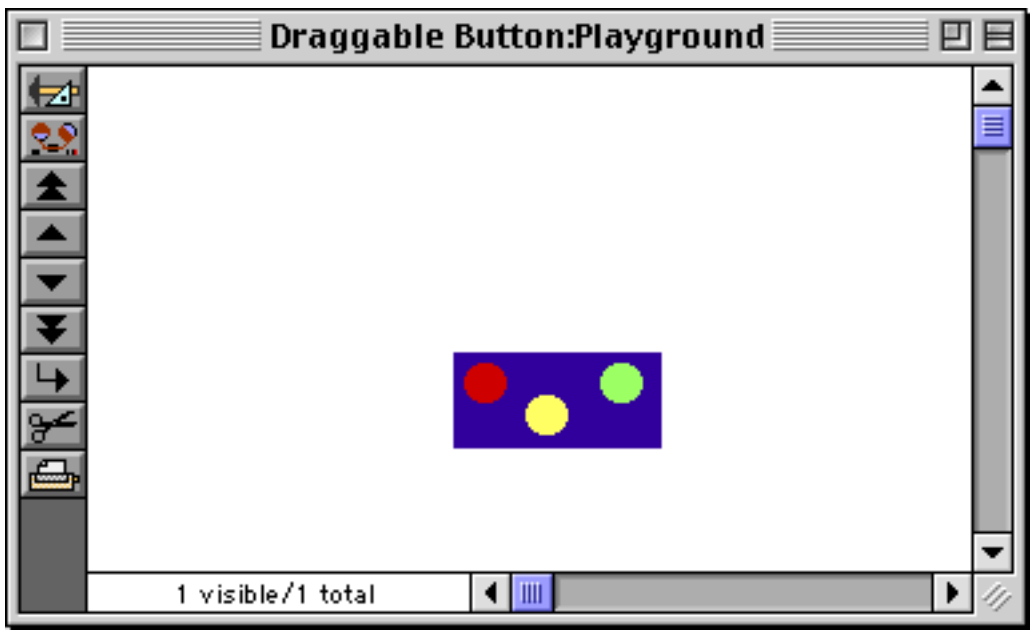
❼ The new co-ordinates for the button are in drag. However, these co-ordinates are relative to the upper left hand corner of the screen, and the changeobjects statement needs them relative to the upper left hand corner of the form. The xytoxy(function will convert the co-ordinates.

❽ The changeobjects statement moves the button (and any other objects with the same co-ordinates to the new position.

With a few changes the procedure can be modified to move multiple objects at once.



With this new procedure when the mouse is released all of the objects inside the boundaries of the button will move also.



This procedure moves all the objects inside the boundaries of the button.

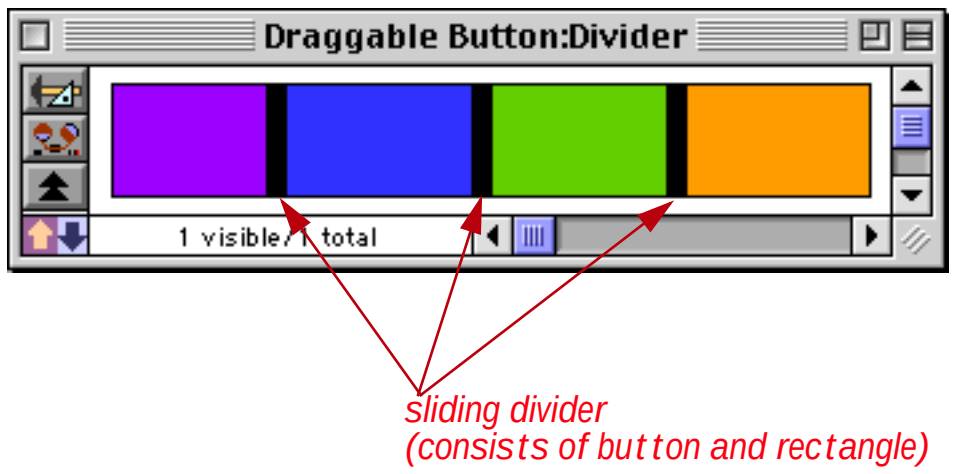
```
❶ local drag,dragstart,insidewindow,deltaV,deltaH
❷ drag=info("buttonrectangle")
❸ dragstart=drag
❹ selectobjects
   unionrectangle(xytoxy(drag,"s","f"),objectinfo("rectangle"))
   =xytoxy(drag,"s","f")
❺ insidewindow=rectangle(
   rtop(info("windowrectangle"))+20,
   rleft(info("windowrectangle"))+26,
   rbottom(info("windowrectangle"))-16,
   rright(info("windowrectangle"))-16)
❻ draggraybox drag,insidewindow,info("windowrectangle"),0
❼ if drag ="" stop endif
❽ deltaV=rtop(drag)-rtop(dragstart)
❾ deltaH=rleft(drag)-rleft(dragstart)
❿ changeobjects "rectangle",rectangle(
   rtop(objectinfo("rectangle"))+deltaV,
   rleft(objectinfo("rectangle"))+deltaH,
   rbottom(objectinfo("rectangle"))+deltaV,
   rright(objectinfo("rectangle"))+deltaH)
```

This procedure is similar to the last example, but with a couple of twists. Statement ❹, `selectobjects`, uses a trick with the `unionrectangle(` function (see “Rectangles” on page 1290) to select all the objects inside the button. If the union of the button rectangle and object X‘s rectangle is equal to the button’s rectangle then object X is completely inside the button rectangle.

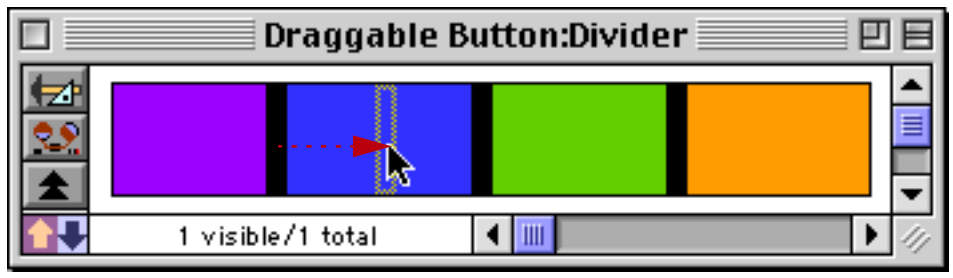
The other twist is in how the objects are moved after the drag is completed. The procedure can’t simply change all the objects to the new drag rectangle, because each object has a different position within the button. Instead, the procedure calculates the vertical and horizontal offsets between the old position and the new position (statements ❽ and ❾) and then adds this offset to each of the selected objects (statement ❿).

Movable Dividers

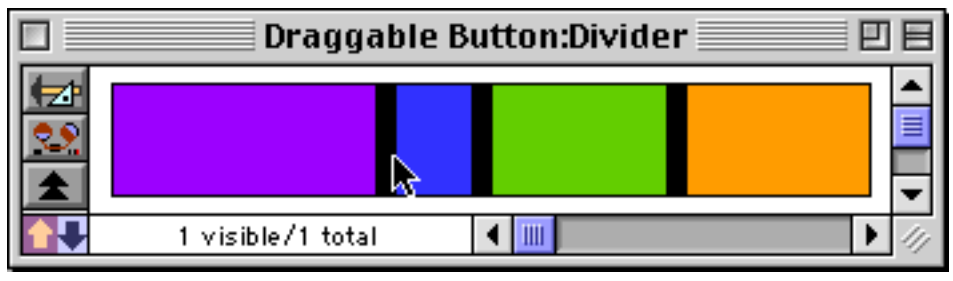
Using the `draggraybox` statement you can create a movable divider between two elements on a form. The user can slide this divider to change the division point between the two form elements. To illustrate this consider the form shown below. The form has three sliding dividers that divide the four different sections.



Each sliding divider consists of a button and a rectangle. When you press on the button a procedure is triggered (see below). That procedure allows the divider to move left or right. For example you could slide the divider between the purple and blue sections to the right.



When the mouse is released the purple section expands and the blue section gets smaller.



The dividers can be moved at any time to adjust the form as his or her needs change.

Building a sliding divider like this requires three (optionally four) graphic objects. First are the two primary elements being divided. For our example we're assuming that these two elements are side-by-side and are the same height. Between the two main elements is a small gap. This gap should be filled with a regular pushbutton (see "Push Buttons" on page 813). The pushbutton must exactly match the gap between the two objects, so that the edges of the pushbutton are exactly on top of the edges of the primary elements. The pushbutton must have the click/release option turned off (see "Click/Release" on page 819). You can optionally include another graphic element (for example a black rectangle or a flash art object) with the same dimensions as the pushbutton.

Here is the procedure that allows the user to slide the divider back and forth.

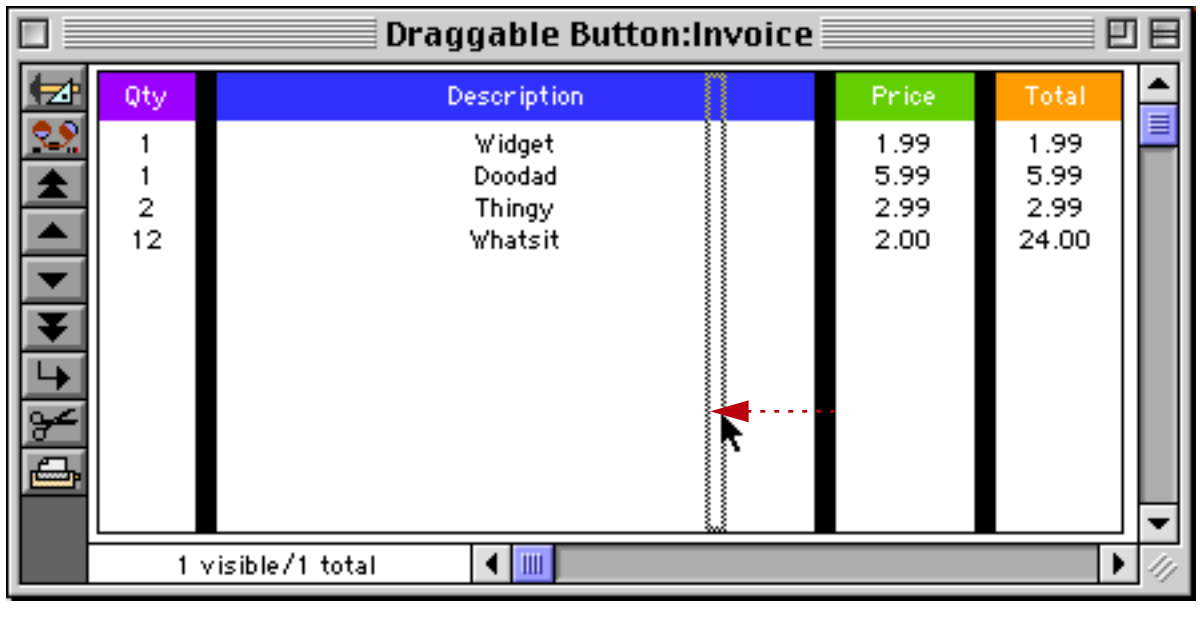
```
local drag,dragstart,deltaV,deltaH,slider,slidebox
drag=info("buttonrectangle")
dragstart=drag
slider=xytoxy(drag,"s","f")
slider=rectangle(
   rtop(slider),
   rleft(slider),
   rbottom(slider)+1,
   rright(slider)+1)
selectobjects
   intersectionrectangle(xytoxy(drag,"s","f"),objectinfo("rectangle"))
      ≠rectangle(0,0,0,0)
slidebox=xytoxy(objectinfo("boundary"),"f","s")
slidebox=rectangleadjust(slidebox,0,16,0,-16)
draggraybox drag,slidebox,info("windowrectangle"),1
if drag="" stop endif
deltaV=rtop(drag)-rtop(dragstart)
deltaH=rleft(drag)-rleft(dragstart)
changeobjects "rectangle",
   adjustxy(objectinfo("rectangle"),slider,deltaV,deltaH)
```

In this procedure, the variable slider is the dimensions of the button in the gap. The variable slidebox is the area the slider can slide back and forth in. This area includes most of the two primary elements, with a 16 pixel buffer on each end.

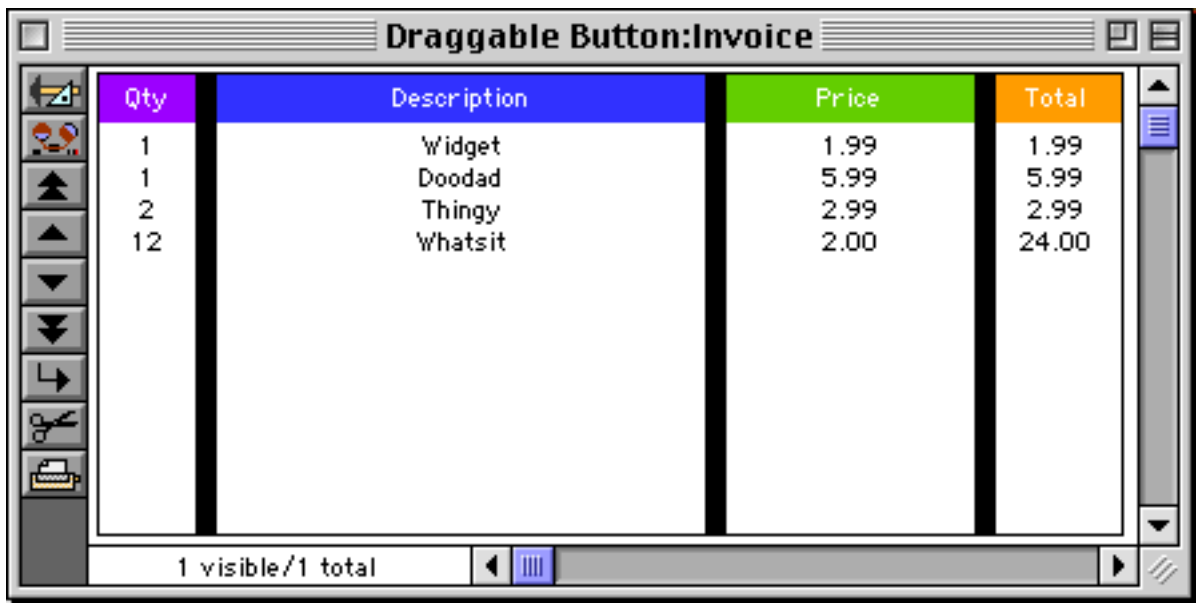
The last statement of this procedure uses the `adjustxy(` function to actually adjust the slider and the primary elements (see “Rectangles” on page 1290). This function has four parameters: the original rectangle, a boundary rectangle, the vertical offset and the horizontal offset. The function takes the original rectangle and adjusts each corner of the rectangle by the offsets, but only if the corner is inside the boundary rectangle. If a corner is outside the boundary rectangle, it is not adjusted. Using this function it is easy for the procedure to shift the slider and gap between the two primary elements without shifting the outside edges of the primary elements.

You may have noticed that the procedure does not directly refer to either the primary objects or the slider objects. Instead it refers to everything by position. You can use this same procedure to drive several sliders in your form, or even in several forms. You can also stack several primary elements end-to-end with sliders in between each. The user can move the sliders back and forth any way they want to adjust the size of each primary element.

Here is an example of a more practical use for this procedure. The invoice contains four columns.



The width of each column may be adjusted at any time simply by dragging on a divider (without going into graphics mode).



Drag and Drop

Panorama supports the ability to drag data from one location to another (usually called drag-and-drop). On MacOS computers you can drag data within Panorama, and also drag data from other applications to Panorama or drag data from Panorama to other applications. On Windows systems you can only drag within Panorama, drag and drop between Panorama and other applications is not supported.

Drag Items and Flavors

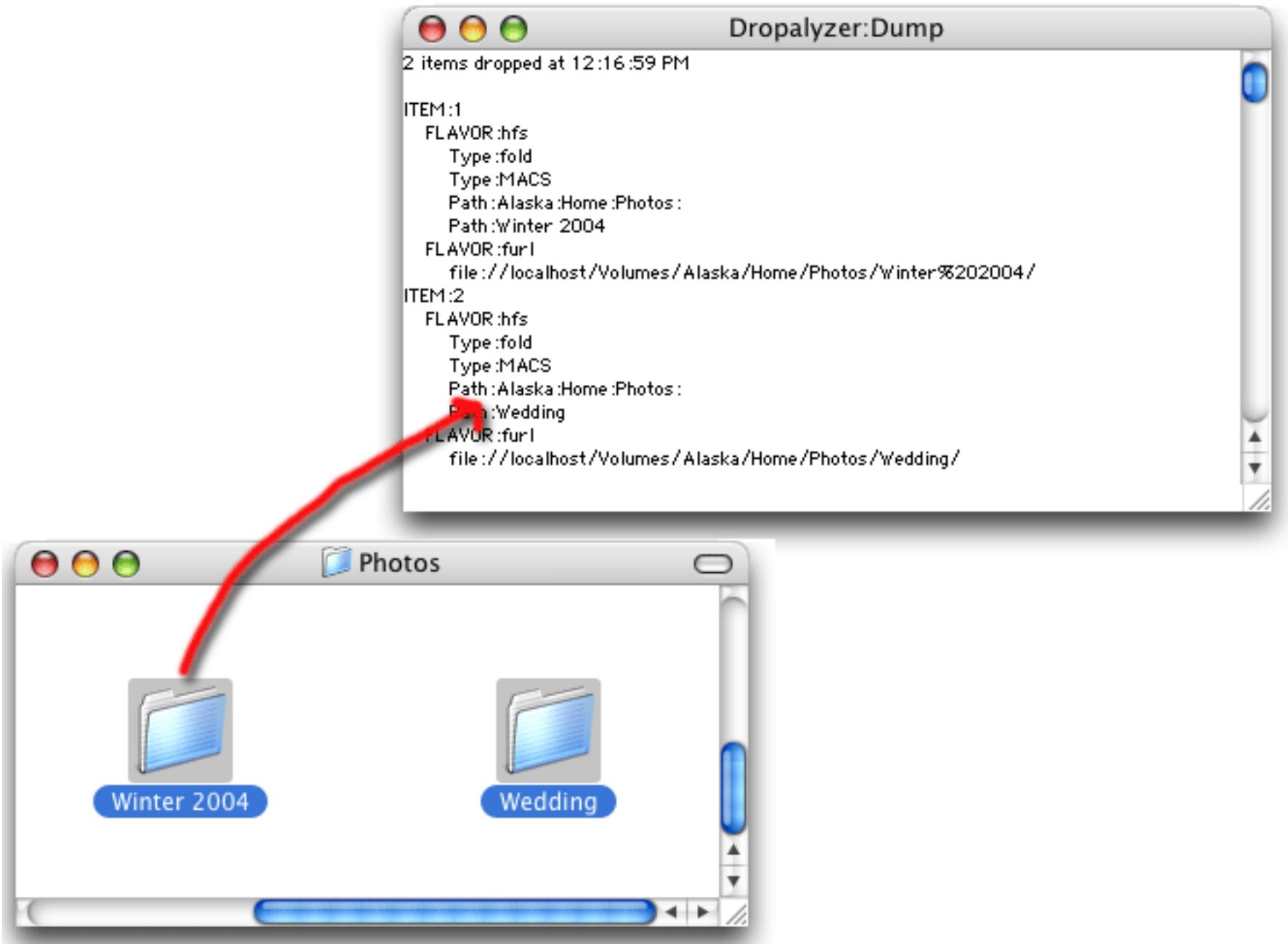
It's possible to drag all sorts of data: text, images, files, sounds, etc. It's possible to drag more than one item at a time (for example several files from the desktop). It's also possible to drag more than one type of data at a time per item, for example an image along with text describing the image. Each type of data being dragged is called a flavor. Flavors are identified by a four character code. The table below lists some of the common flavors you may encounter.

Flavor	Description
TEXT	Normal text
PICT	Picture
hfs	File or folder
furl	UNIX file path
url	Web URL
vCrd	VCard (contact info)

You can also invent your own drag flavors. Just give them a unique four letter code. Of course only your application will understand your custom flavor — you won't be able to drag items using your custom flavor to any other application. (Since you can drag more than one flavor at once you can get around this by also including one of the standard flavors, if necessary.)

The Dropalyzer Wizard

The **Dropalyzer** wizard is a handy tool for analyzing, writing and testing drag and drop procedures. You'll find this wizard in the **Developer Tools** submenu of the Wizard menu. When you first open this wizard it is completely blank, but you can drag anything you want onto this wizard and it will display some information about what was dropped. The illustration below shows the display if you drop two folders from the Finder onto the **Dropalyzer** wizard.

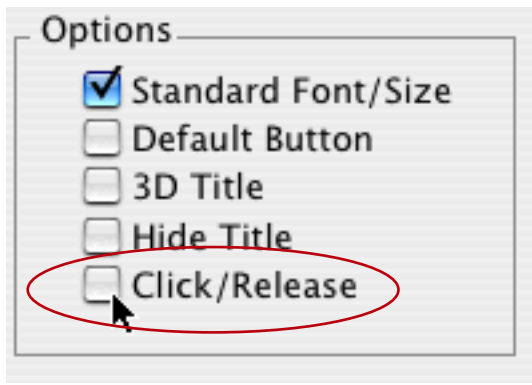


The wizard shows you that two items were dropped on it. Each item has two flavors, hfs and furl.

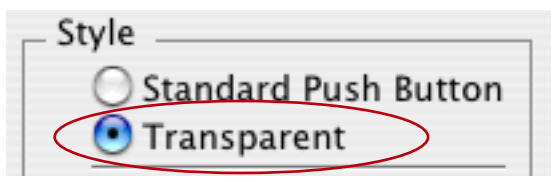
Any time you are wondering what is really going on with a drag and drop problem the Dropalyzer wizard is a lifesaver. If you are trying to figure out what an application is giving you just drop the items on the wizard. If you are trying to debug a procedure that drags from Panorama just drag to the Dropalyzer wizard and you'll see exactly what your procedure is generating.

Dragging Items from Panorama

Dragging data from Panorama always starts with clicking on a button. To allow dragging, the button's click release option must be turned off.



You can use a regular button, but usually you'll want to use a graphical button. You can either use a Flash Push Button (see "Flash Art™ Push Button SuperObjects™" on page 823) or a regular push button with the Transparent option turned on. If you are using a transparent button just lay it on top of the graphic image or icon.



Like any push button you must set up a procedure that is triggered by the button. This procedure will collect the information to be dragged and start the actual drag operation. The actual code in this procedure depends on whether you are dragging one or more flavors.

Dragging a Single Flavor

If you are only dragging a single flavor the procedure only needs one statement to start the drag: `DragDrop`.

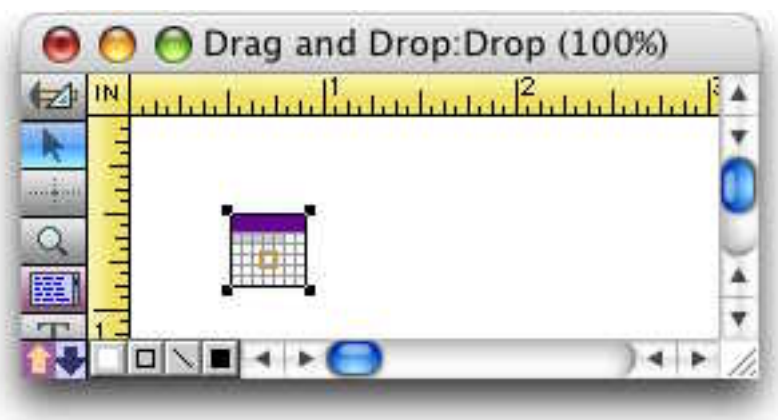
```
dragdrop rectangle,flavor,data
```

Rectangle is the dimensions of the dragged area in global co-ordinates. If you want to drag an area that is the same size as the button just use the `info("buttonrectangle")` function.

Flavor is the four letter flavor code for the data being dragged, for example "TEXT" or "PICT".

Data is the actual data to be dragged. If this is text you can simply supply a formula to calculate the text.

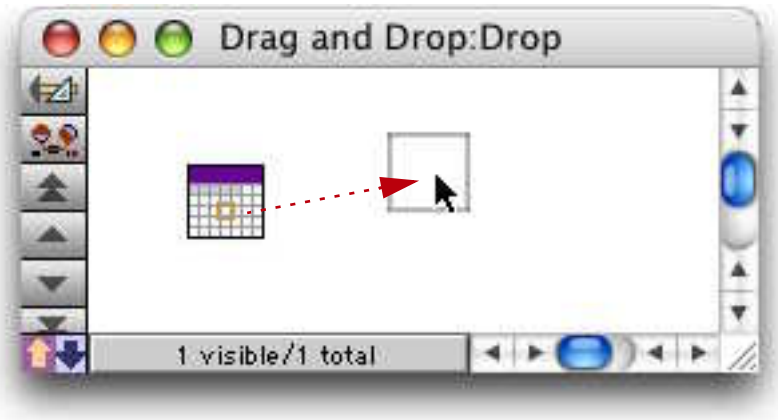
Here is an example that drags the current date. The form contains a graphic covered by a transparent push button.



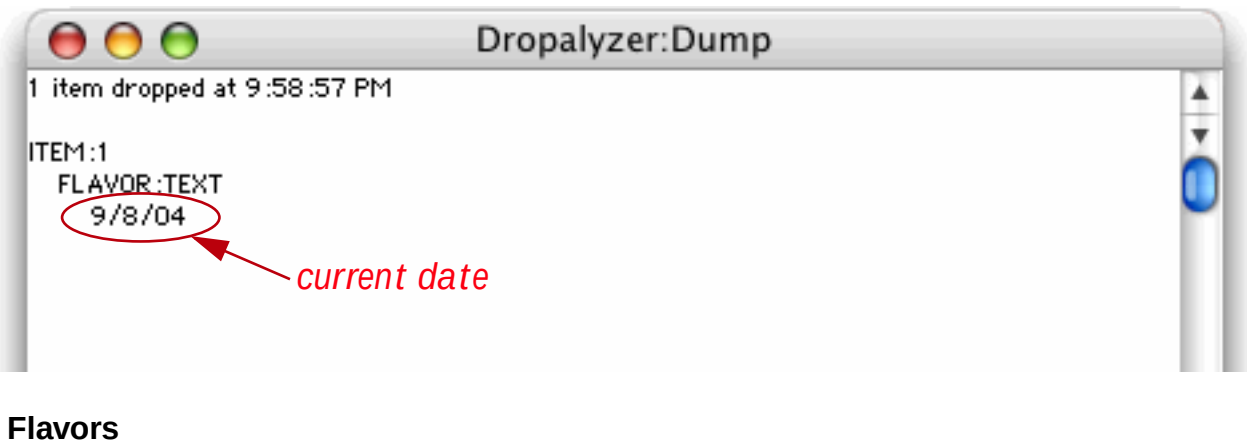
The button is designed to trigger a procedure named .DragDate. This procedure has a single statement:

```
DragDrop info("buttonrectangle"),"TEXT",datepattern(today(),"mm/dd/yy")
```

To copy the date to another location, click on the calendar and begin to drag.



When you drop this item on any application that accepts drag and drop text the current date will be transferred in the format mm/dd/yy, for example 5/30/06.



Dragging Multiple Flavors

If you want to include more than one flavor a multi-step procedure is required. The first step is the DragData statement (see above), but the Flavor and Data parameters must be left blank (""). This tells Panorama that there are additional flavors to follow.

The next step is to add flavors (with data) to the drag. This is done with the DragDropData statement, which has two parameters: Flavor and Data. You can add as many flavors as you like by simply repeating the statement over and over.

Flavor is the four letter flavor code.

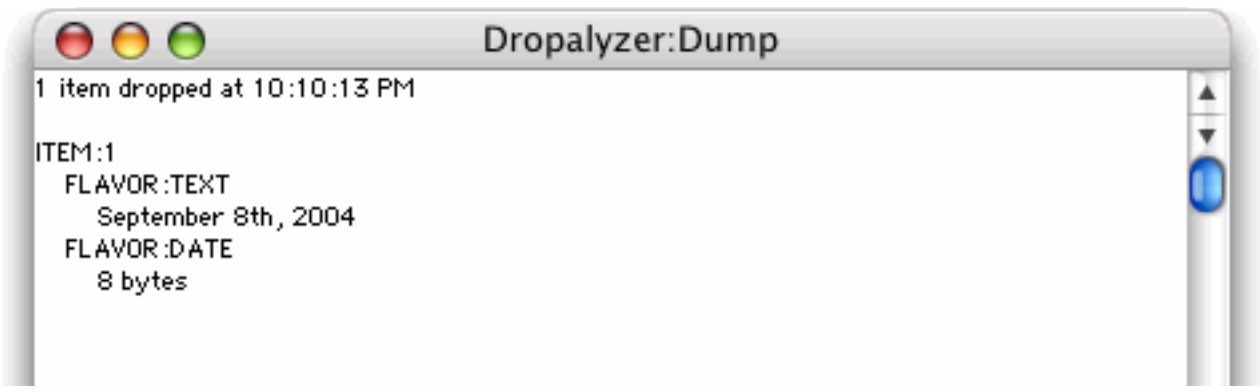
Data is the actual data to be included in the flavor.

Once all of the flavors have been added the final step is to use the StartDragDrop statement. This tells Panorama that all of the flavors have been specified and it should start the drag operation. This statement has no parameters.

Here is an example that drags today's date in two formats:

```
DragDrop info("buttonrectangle"),"",""
DragDropData "TEXT",datepattern(today(),"Month ddnth, yyyy")
DragDropData "DATE",datepattern(today(),"MM/DD/YY")
StartDragDrop
```

Dragging from this button to the Dropalyzer now shows that both flavors have been included in the drag.



Note: The Dropalyzer wizard doesn't know how to display the contents of the DATE flavor, but it does show that it exists and contains 8 characters.

Receiving Dragged Data

What happens when an item is dragged and dropped onto a Panorama window? If the window is not a standard form window nothing will happen. Panorama does not support dropping on a data sheet or view-as-list window, only a standard single record form window. Any data dropped on other types of windows will be ignored.

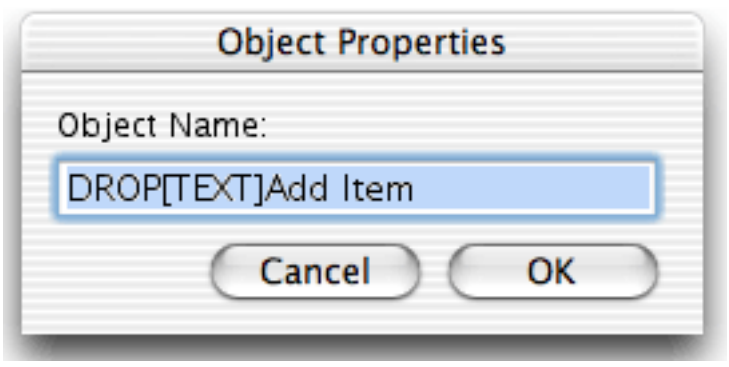
If the window dropped on is a standard form window, Panorama checks to see if the drop was on top of an object. If not, the drop is ignored.

If the data was dropped on an object, Panorama checks the name of the object. If the object name does not begin with DROP[, the dropped data is ignored.

If the object name contains one or more flavors followed by] after DROP (for example DROP[TEXT]), Panorama checks to see if data being dropped is in the list of flavors. If not, the dropped data is ignored. If the object name begins with DROP[], then any kind of flavor will be accepted.

If the all these criteria are met then Panorama triggers the .DropProcedure procedure. This procedure processes the incoming dropped data, we'll discuss this procedure in more detail in a moment.

As you can see, the object name is an important component of drag and drop. Only objects with the proper name can receive dropped data. To review, you can set an objects name by selecting the object and choosing **Object Name** from the Edit menu, or by clicking on the object name in the Graphic Control Strip.



Any text in the object name after the] can be retrieved with the `info("droptrigger")` function. This allows the .DropProcedure procedure to figure out where on the form the data was dropped. The procedure can also find out the entire object name with the `info("dropobject")` function.

The .DropProcedure

The .DropProcedure procedure can use several statements and functions to find out information about the drag:

`info("droptrigger")` returns the trigger of the object that received the drag.

`info("dropobject")` returns the name of the object that received the drag (not just the trigger). This is handy if you use a Text Editor SuperObject as a drop object, allowing you to easily write a single procedure that allows you to drop on multiple objects.

`info("dropdatabase")` returns the name of the database that was dropped on.

`info("dropwindow")` function returns the name of the window that was dropped on. Since this may not be the current window, the first step usually taken by the .DropProcedure is to make this window current with this statement:

```
window info("dropwindow")
```

`info("dropform")` returns the name of the form that was dropped on

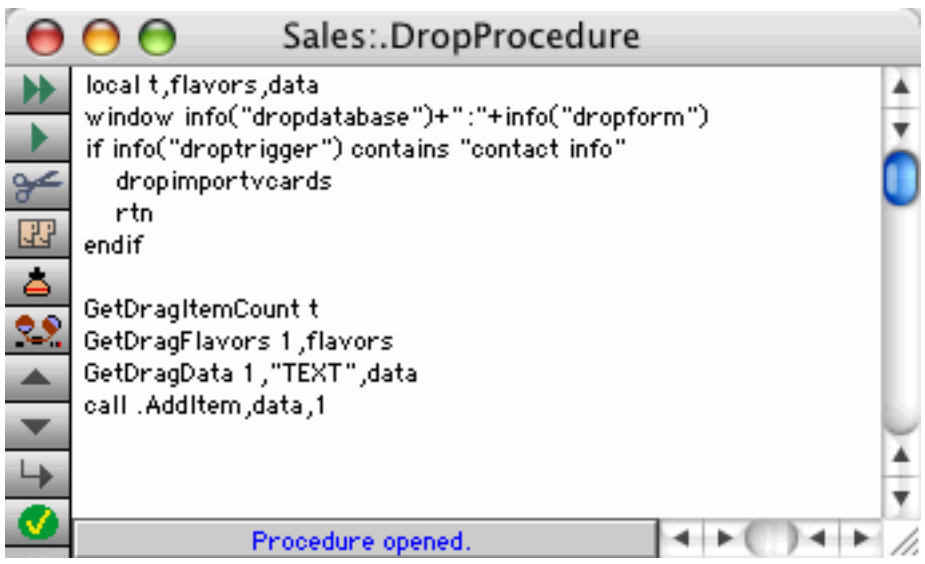
`GetDragItemCount variable` This statement returns the number of items in the drag. For example, if you select 5 items in the Finder and then drag them this statement will return 5. Items are numbered starting with 1.

`GetDragFlavors itemnum,array` This statement returns a carriage return delimited array of flavors for a particular item. Each array element is 4 characters long. The item # must be between 1 and the number of items that were dragged.

`GetDragData num,flavor,variable` This statement returns the data in a flavor.

`DragComplete` This statement frees the scratch memory used when receiving a drag. After you use this statement you can no longer use the statements and functions above (until the next drag is received).

The Sales database example included with Panorama (in the Guided Tour submenu of the Wizard menu) allows you to drag items from the Catalog to the invoice. Here is the .DropProcedure for this database.



The procedure starts by making sure the invoice window is the top window (the `window` statement). It then checks the drop trigger to see if the data was dropped on the contact info section of the form. If so, it uses the dropimportvcards statement (described later in this chapter) to import the Vcard info that was dropped (if any). Otherwise it gets the text that was dropped into a variable named data, and passes that to the .AddItem procedure. This procedure adds the item to the invoice.

Dropping Files and Folders on Panorama

Want your Panorama application to process files and/or folders that are dropped onto one of your forms? The `dropfromfinder` statement will do much of the work for you!

```
dropfromfinder filter,filelist
```

The filter parameter specifies types of files that you want to accept. The parameter must be 1 or more four character type codes, for example TEXT, PICT, etc. If no type codes are supplied then all files will be accepted. Any folders dropped are normally expanded (including subfolders) into a list of files contained in the folders. However, if the filter begins with *f* folders will not be expanded. In that case you would be responsible for checking for and processing folders yourself.

The files parameter is the name of a field or variable. When the statement is complete this field or variable will contain a carriage return separated list of every file that was dropped, including the path of the file. If any folders were dropped the statement will automatically list every file within the folder, including within subfolders (unless the filter parameter begins with *f*).

The example .DropProcedure below will accept files dropped from the finder. Any Picture files (PICT) will be added to the database in the Photo field.

```
local pictFiles,pf,n
dropfromfinder "PICT",pictFiles
n=1
loop
   pf=array(pictFiles,n,¶)
   stoploopif pf=""
   addrecord
   Photo=pf
   n=n+1
while forever
```

The photos added to the database could be displayed with a Flash Art SuperObject.

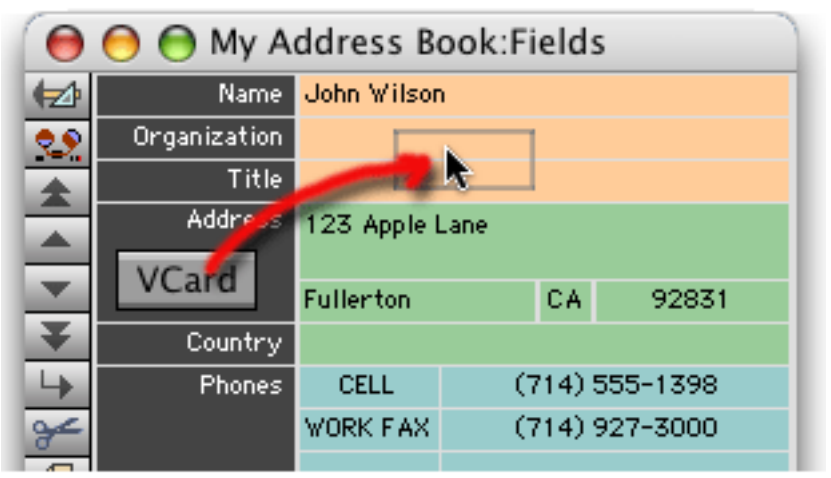
VCard Drag and Drop

Once generic fields have been set up for a database (see "“Generic” Fields" on page 270) it is very easy to add drag and drop support so that you can exchange data with Apple's Address Book (or other databases or VCard enabled applications) without using a wizard.

Dragging VCard information is just like anything else, you set up a button and a procedure. The procedure only needs a single statement:

```
dragvcard
```

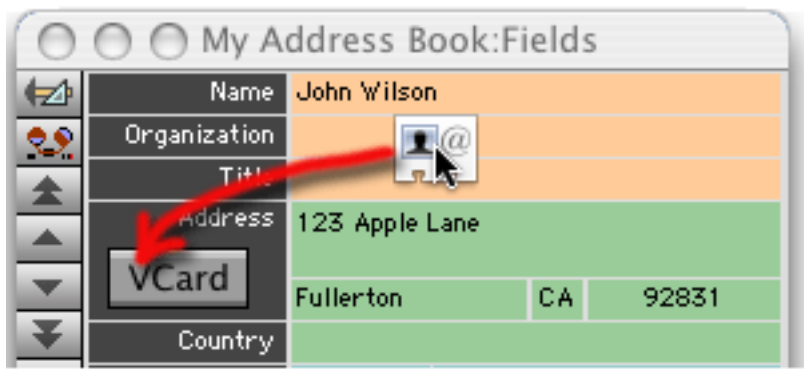
That's all there is to it! Now you can drag from your button to any VCard enabled application.



Adding the ability to drop VCards on a form is not much more difficult. The object for dropping must be set up with the proper object name, as always. This procedure only also needs only one statement

```
dropimportvcards
```

(If your database can handle other types of data dropped on it you'll need a more complex procedure that decides what has been dropped. If you detect that a VCard has been dropped, use the `dropimportvcards` statement.) Now you can drop VCards on this object to import them into the database.

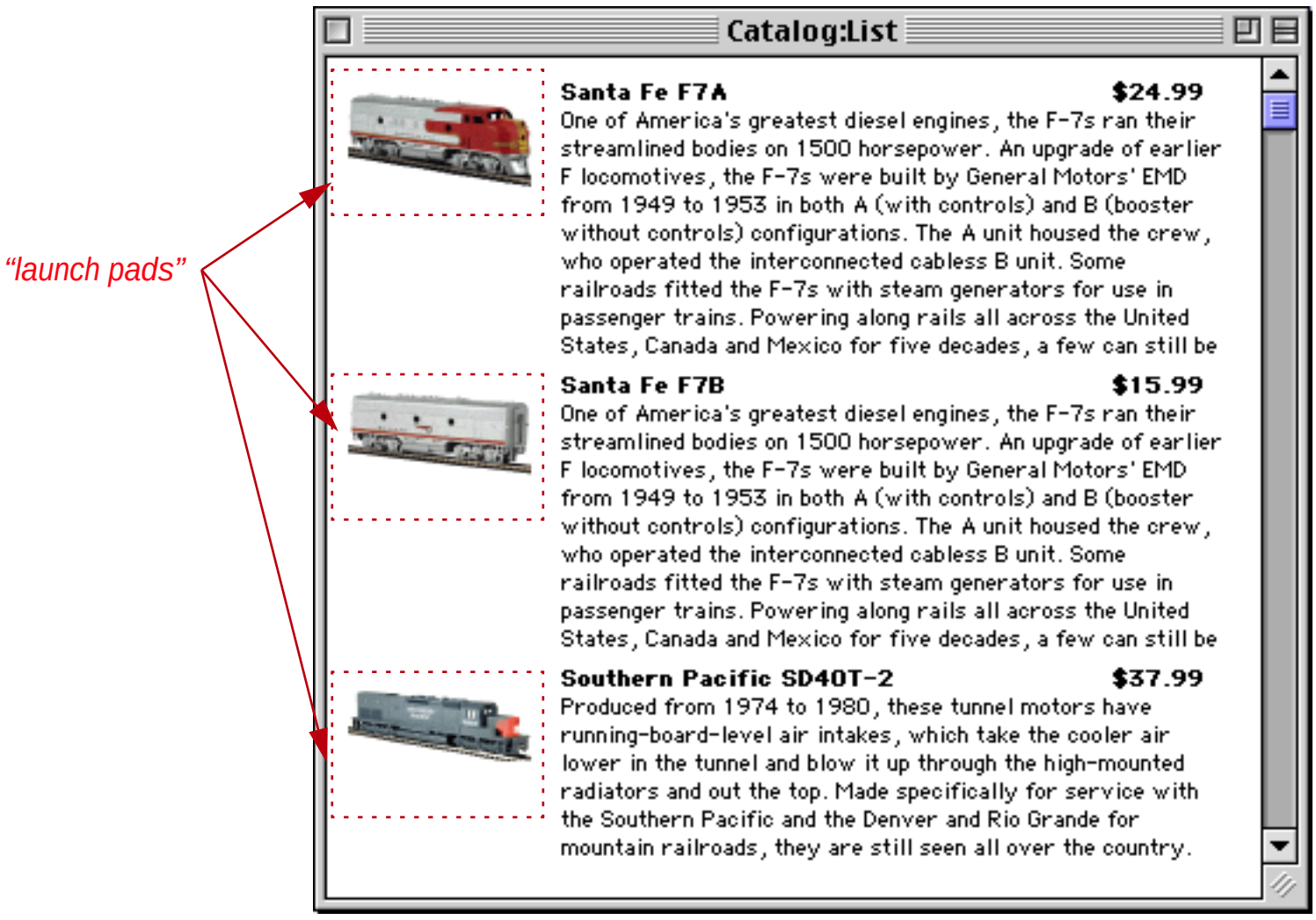


Remember, you **must** set up Generic Fields for this database before you use these statements. You'll get all kinds of error messages if you don't set Generic Fields first.

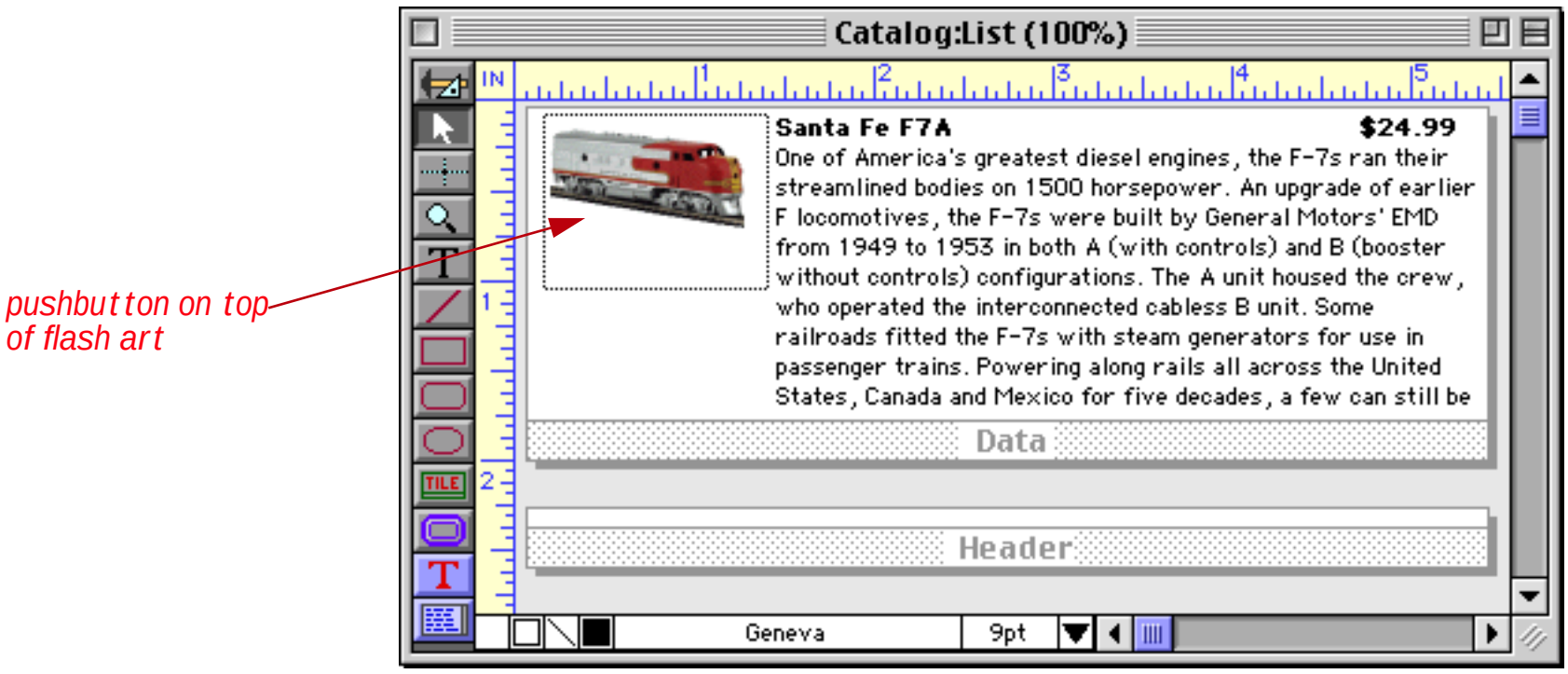
Drag and Drop (Obsolete Method)

This section describes an older technique for drag and drop that uses draggraybox. Since this statement can no longer be used to drag outside of the current window we no longer recommend this technique. We are retaining this section of the manual to help you understand any old databases that you have that may use this method. The technique will continue to work on OS 9 and Windows, but we recommend that you convert to the new method as soon as possible.

Dragging and dropping involves two active areas: a launching pad and a landing zone. Dragging starts when the user presses the mouse on an active launching pad, which is usually a pushbutton with the click/release option turned off. The user drags from the launching pad to a landing zone, an area that can receive the data from the launching pad. The landing zone may be on the same form as the launching pad, or it may be on a different form. A single launching pad can have several possible landing zones. Here is a form with three "launching pads," one per record.



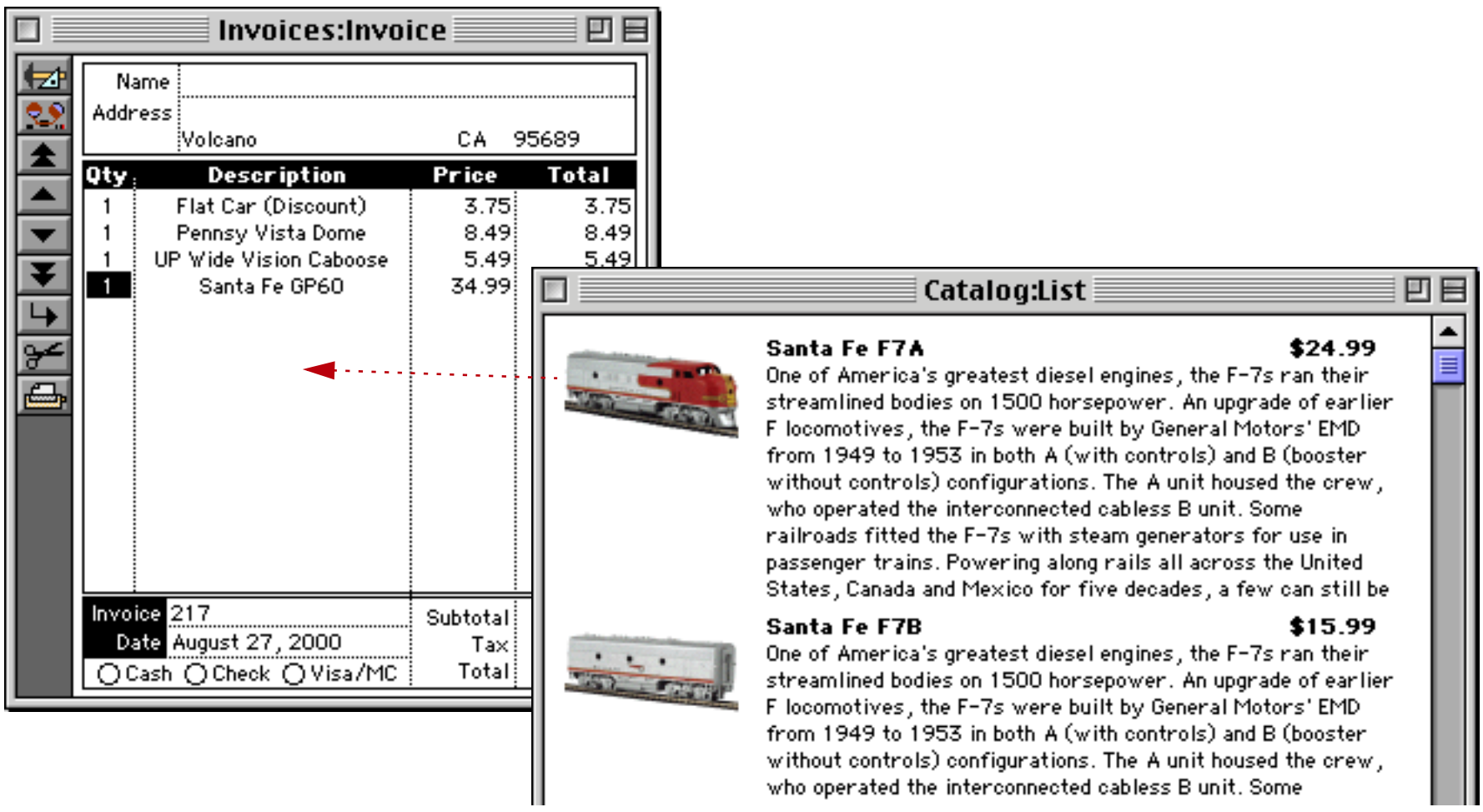
The launch pads all come from a single pushbutton on the data tile in this view-as-list form (see "View-As-List Forms" on page 889).



The launching pad is a pushbutton. What defines a landing zone? A landing zone is whatever your procedure defines it to be. It could be an object, a collection of objects, or even an entire window.

The launch pad pushbutton triggers a procedure. This procedure uses the `draggraybox` statement to let the user drag to another location. When the user releases the mouse, the procedure must decide whether or not the mouse is over a suitable landing zone. The procedure can use the `findwindow(` function to find out what window the mouse is on top of (see "FINDWINDOW(" on page 5250). If this is a window that can contain a landing zone the procedure can bring that window to the front and then use the `selectobjects` statement to find out if the mouse is over an object that is a suitable landing zone (this step is unnecessary if the whole window can be a landing zone). If the mouse is over a landing zone, the procedure then copies the data appropriately.

Now that you are familiar with the theory of drag and drop, let's take a look at some practical examples. We'll start with a catalog and invoice database, like the one's shown below. The goal is to be able to drag an item from the catalog onto the invoice and have that item added to the invoice, as shown in this illustration.



Here is the procedure that is triggered by the pushbutton. Remember, the pushbutton must have the click/release option turned off.

```
local drag,landingWindow,landingDatabase,landingFields
local dragItem
dragItem=Item                    /* copy the data for later */
drag=info("buttonrectangle")     /* initial co-ordinates of box */

/* drag the box around */
draggraybox drag,"","",0
if drag="" stop endif
landingWindow=findwindow(info("mouse"))

/* if we landed (mouse up)outside a window then stop */
if landingWindow="" stop endif

/* what database did we land on? */
landingDatabase=stripchar(landingWindow[1,":"],"!9;ÿ")

/* if landed in catalog then stop */
if landingDatabase=info("databasename") stop endif

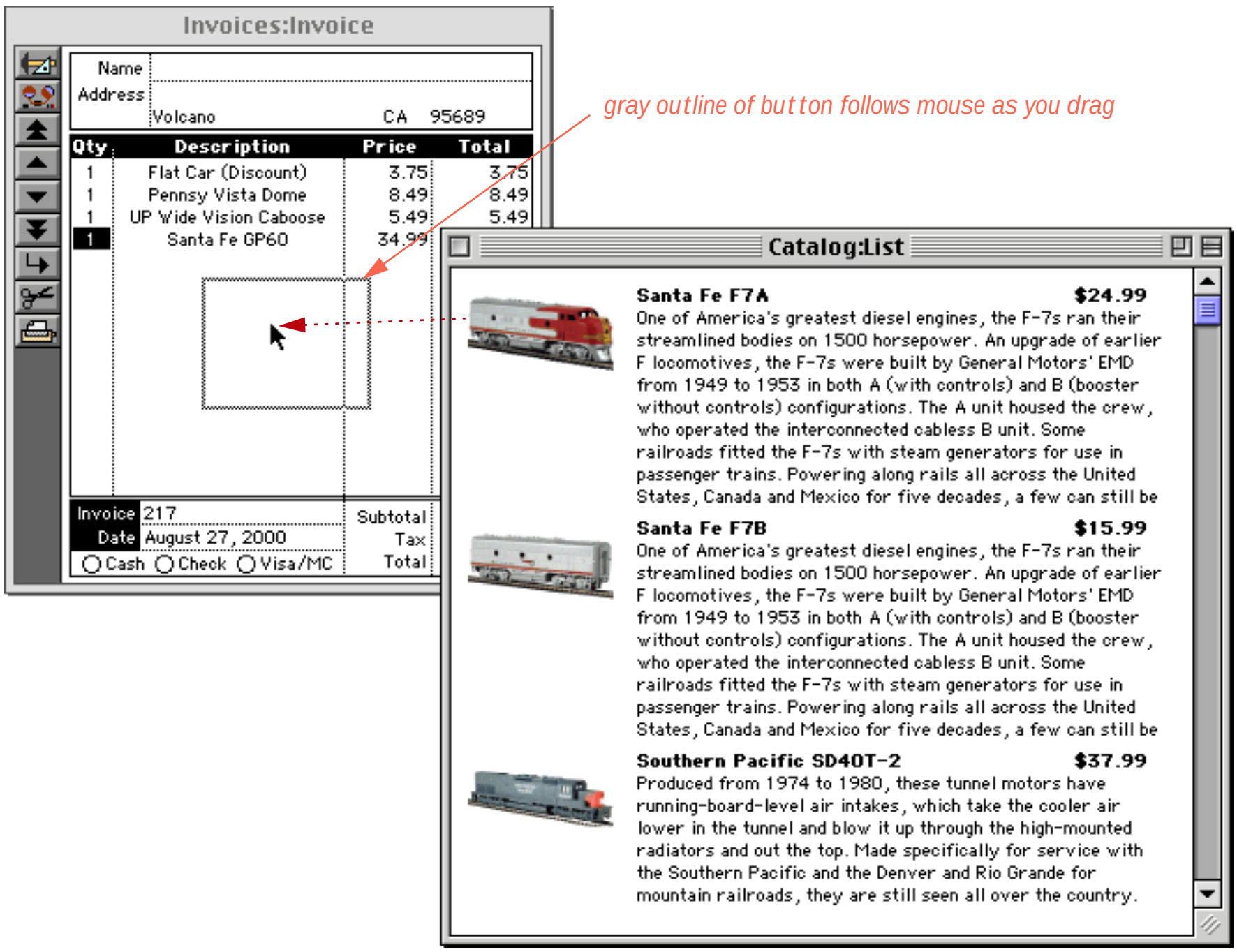
/* does the database we landed on contain the right fields? */
landingFields=dbinfo("fields",landingDatabase)
if (not (landingFields contains "Description1" and landingFields contains "Price1" and
   landingFields contains "Quantity1"))
      message "Cannot drag item to this database" stop endif

/* copy the data from the catalog into the invoice */
window landingWindow
emptyfield "QuantityΩ"
QuantityΩ=1
DescriptionΩ==dragItem
```

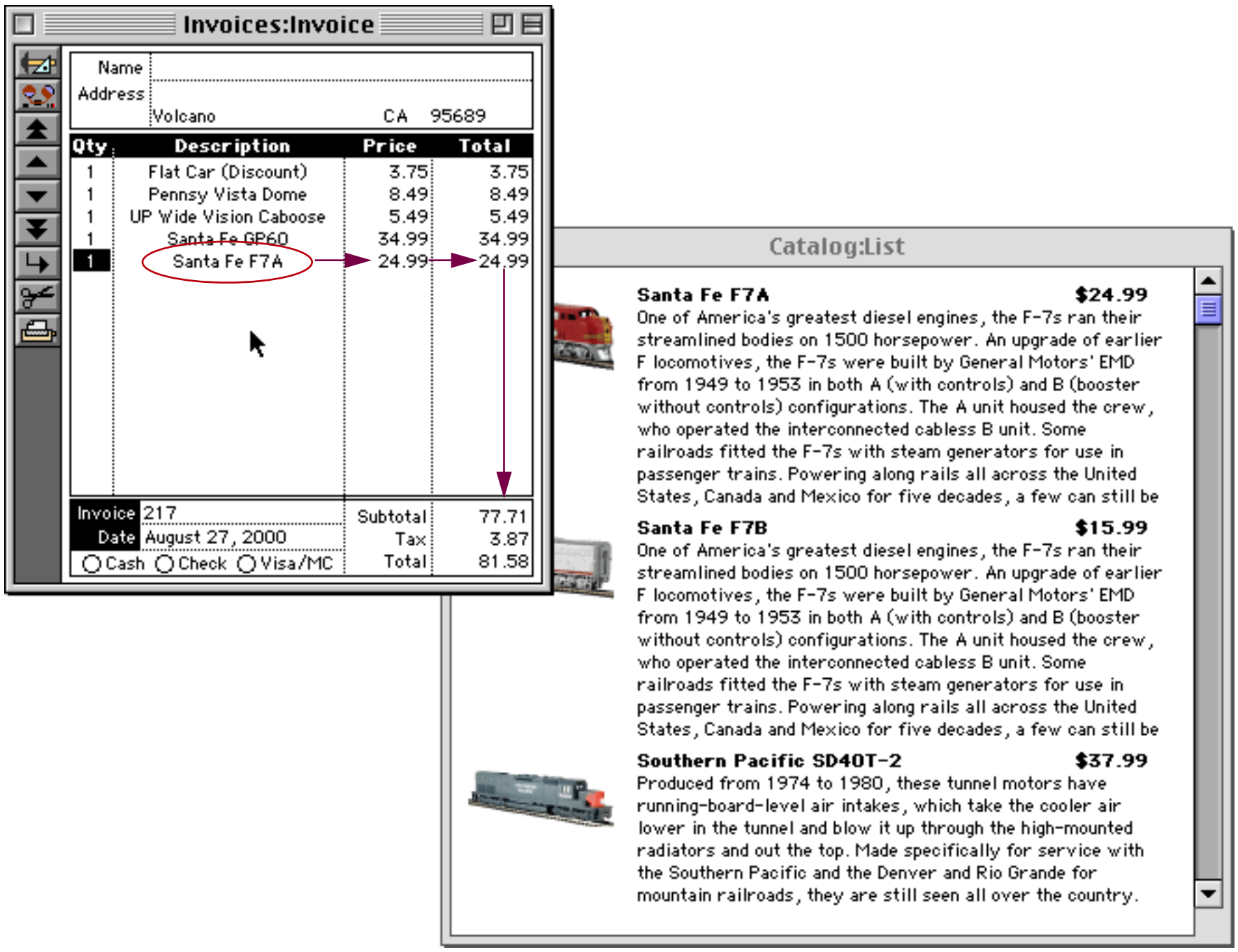
The procedure starts by copying the data that may be dragged into local variables (`dragItem`). Then it allows the user to drag. The drag limits are set to `""`, so the user can drag anywhere on the screen.

After the user releases the mouse, the procedure continues. First, it checks to see what window (if any) the user dragged to. If the user did drag to a window, the procedure strips off any extra information to figure out the name of the database. If the user released the mouse over the original database (the catalog) then the drag and drop is aborted. Otherwise, the procedure checks to see if the database has `Description`, `Price` and `Quantity` line item fields. If not, the drag and drop is aborted. If it does, the procedure brings the new window to the front and copies the data into the appropriate fields. (In this case, the landing zone is the entire window, so no further checking is required once the procedure has determined that the database the user dragged to can accept the data.

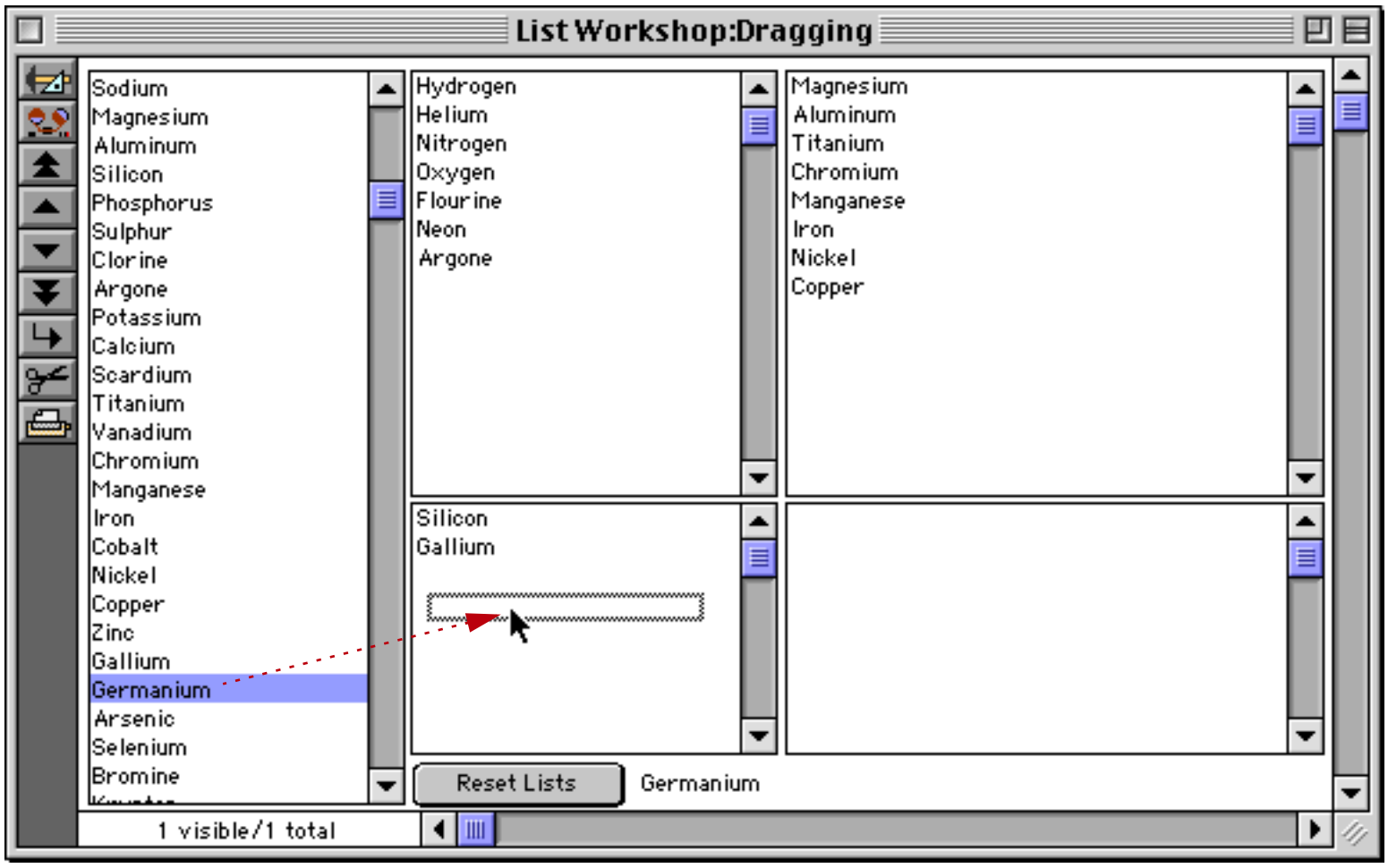
The following illustrations show the final result. When you press on one of the buttons a gray rectangle appears. This gray rectangle can be dragged over the Invoice database.



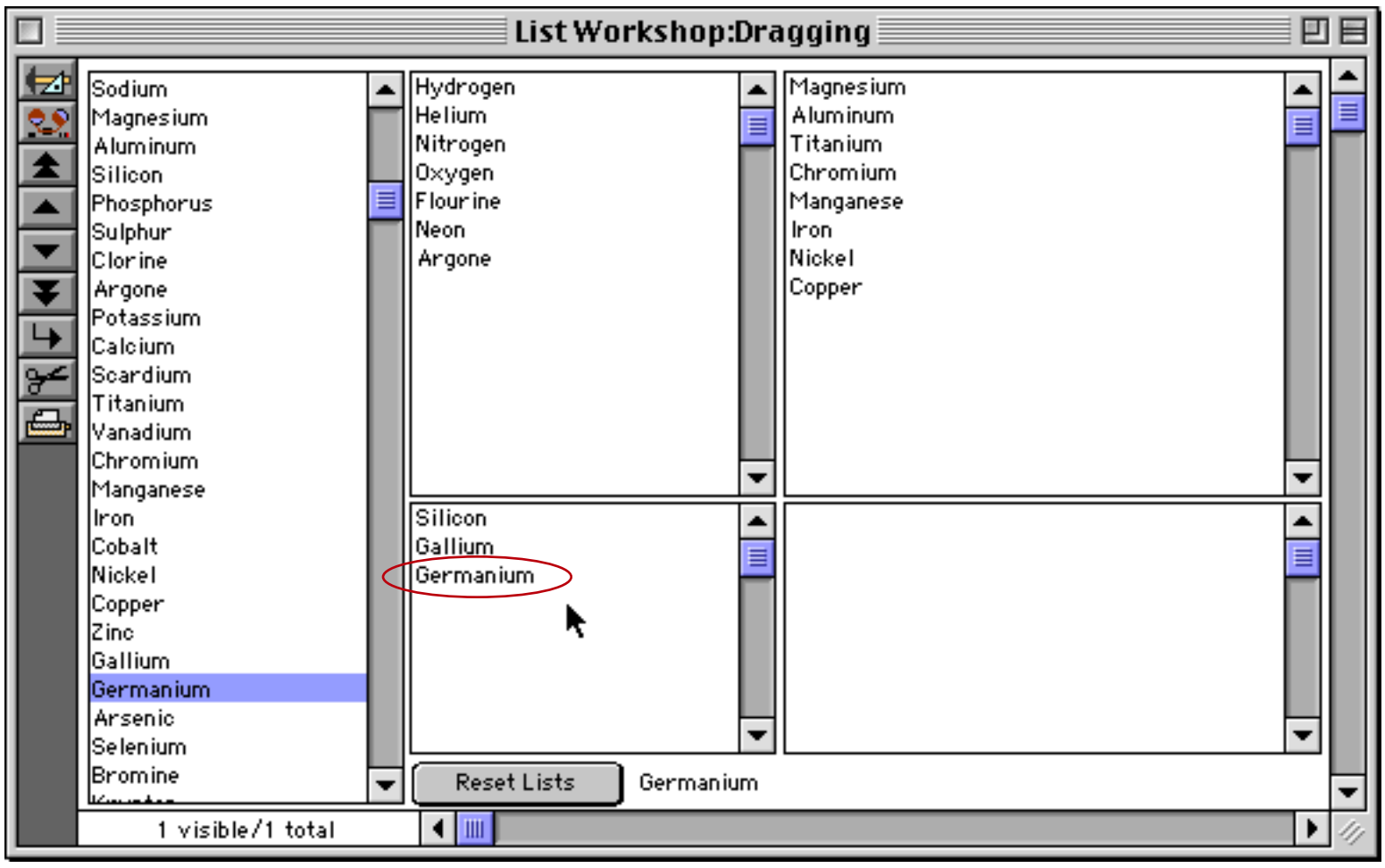
It doesn't matter where you release the mouse, as long as it is somewhere over the Invoice database. When the mouse is released the Invoice window comes to the front and the new item is added to the invoice. By using the double equals sign (see "Triggering Automatic Calculations" on page 1648) the procedure triggers the automatic calculations built into the Invoice database to lookup the price and calculate the line total, sub-total, tax and grand total (as shown by the arrows).



Our second drag and drop example has one launching pad, a List SuperObject, and four landing zones, each a text display SuperObject. The finished example will allow items from the list to be dragged into one of the four landing zones.



When the mouse is released the item is dropped onto the list.



Here is the procedure that performs this drag and drop operation. It assumes that the four Text Display Objects that are acting as landing zones are named DragList1, DragList2, DragList3 and DragList4 (see “Object Type / Object Name” on page 533) and that they are configured to display fileglobal variables with these same four names (the variables must be created in the **.Initialize** procedure).

```
/* the Click/Release option must be turned OFF!!! */

local cell,cellbox,newcell,mouse,mouseStart,landingObject,newWorkList
mouseStart=info("click")
cell=1

/* what cell did user click on */
superobject "Work List","FindCell",cell,dragItem

/* what are the dimensions of this cell */
superobject "Work List","cellrectangle",cell,cellbox

/* we need screen relative dimensions, not window relative */«
cellbox=xytoxy(cellbox,"w","s")

/* drag the box around */
draggraybox cellbox,info("windowrectangle"),info("windowrectangle"),0

/* if dragged outside of window, stop */
if cellbox="" rtn endif

/* where did we end up? */
mouse=xytoxy(info("mouse"),"s","w")

/* did we land on an object? */
selectobjects inrectangle(mouse,objectinfo("rectangle"))
objectnumber 1
landingObject=objectinfo("name")
selectnoobjects

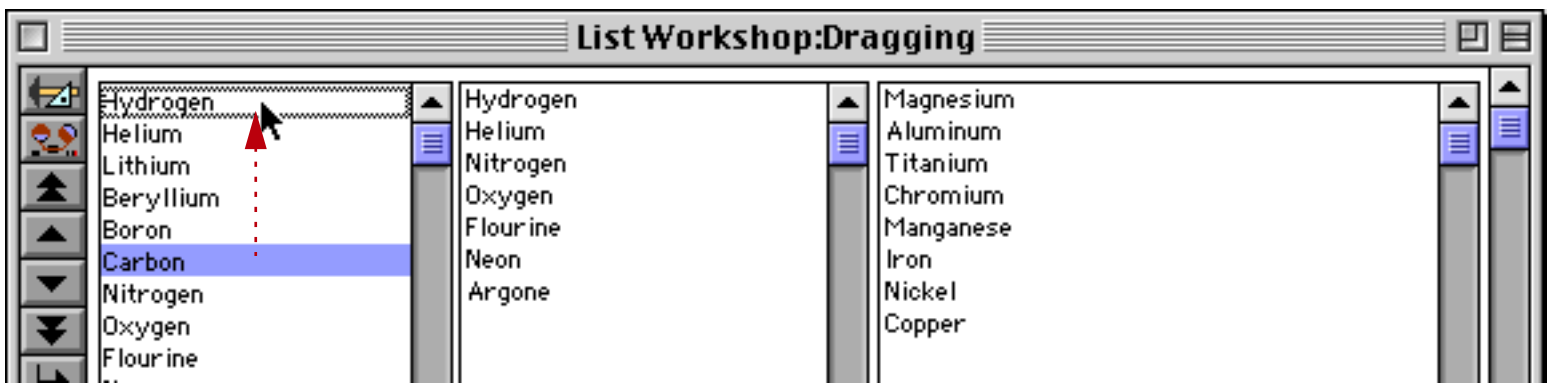
/* if landed on one of the lists, add item to the list */
if landingObject beginswith "DragList"
   /* isn't execute cool?
       this will generate something like this:

         DragList1=sandwich("",DragList1,¶)+"Carbon" showvariables DragList1

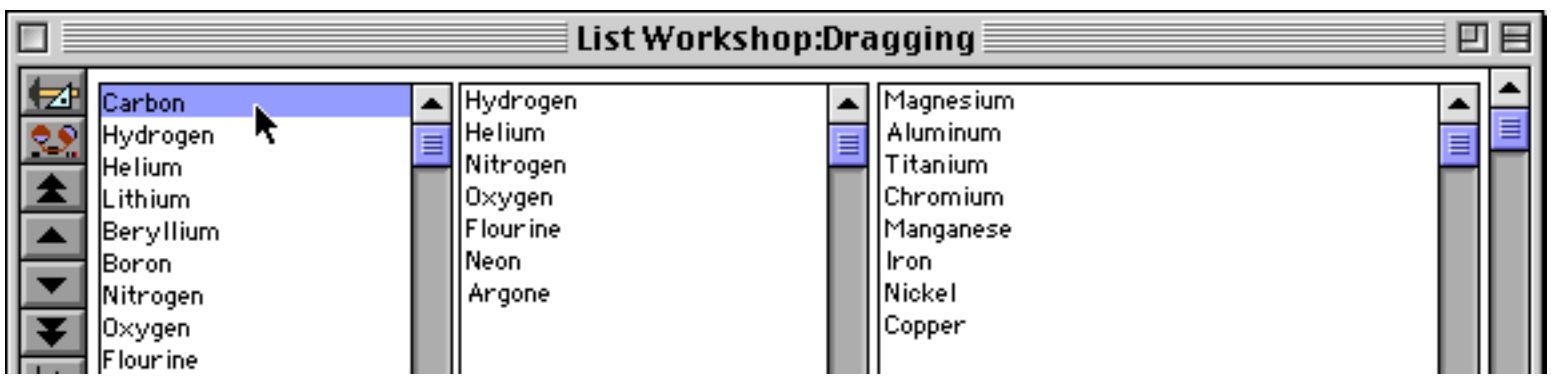
   */
   execute landingObject+{=sandwich("",}+landingObject+{,¶)+"}+
      dragItem+{" showvariables}+landingObject

endif
```

With a slight addition this procedure can also allow the main list itself to be re-arranged by dragging around the items. For example, the Carbon could be dragged up to the top of the list.



When the mouse is released, Carbon moves to the top spot and all of the other items move down.



This capability can be added by appending the steps below to the previous procedure.

```
   /* if we landed on the list itself, re-arrange the order of the list */
   if landingObject = "Work List"

      /* what cell did we land on? */
      superobject "Work List","pointtocell",mouse,newcell

      /* if didn't actually drag (just stayed in the same place) then stop */
      if cell=newcell stop endif

      /* check if we are off the end of the list */
      if newcell≥arraysize(workList,¶)
         newcell=newcell-1 /* make adjustment to stay in the list */
         newcell=0 /* add to end of list */
      endif

      /* delete dragged item from list */
      newWorkList=arraydelete(workList,cell,1,¶)

      /* add dragged item back into list in the new position */
      if newcell>0
         newWorkList=arrayinsert(newWorkList,newcell,1,¶)
         newWorkList=arraychange(newWorkList,dragItem,newcell,¶)
      else
         newWorkList=newWorkList+¶+dragItem
      endif

      /* update and display the list in the new order */
      workList=newWorkList
      superobject "Work List","FillList"
      showvariables dragItem
   endif
```

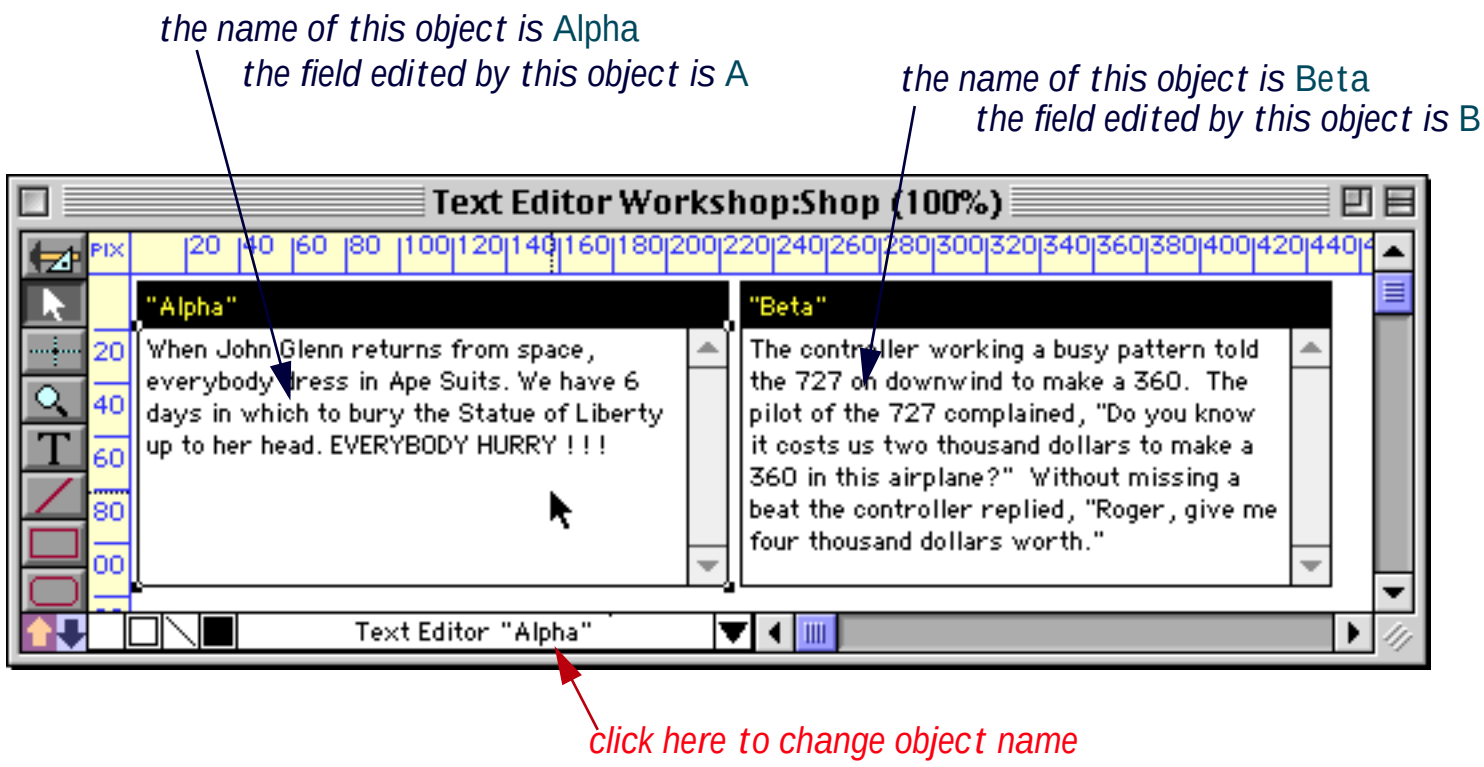
Because Panorama uses a procedure to implement drag and drop, the possibilities are endless.

Program Control of SuperObjects™

In addition to the general graphic program techniques described in the previous sections (changing position and size of objects, font, color, etc.) most types of SuperObjects™ have an additional set of specific commands that it can respond to. For example, a Text Editor SuperObject can be commanded to select a particular section of text, while a List SuperObject can be commanded to add or remove items from the list it displays. To send a command to a specific SuperObject a procedure must use the `superobject` statement (see "SUPEROBJECT" on page 5823).

```
superobject <name of object>,<command>,<additional parameters>
```

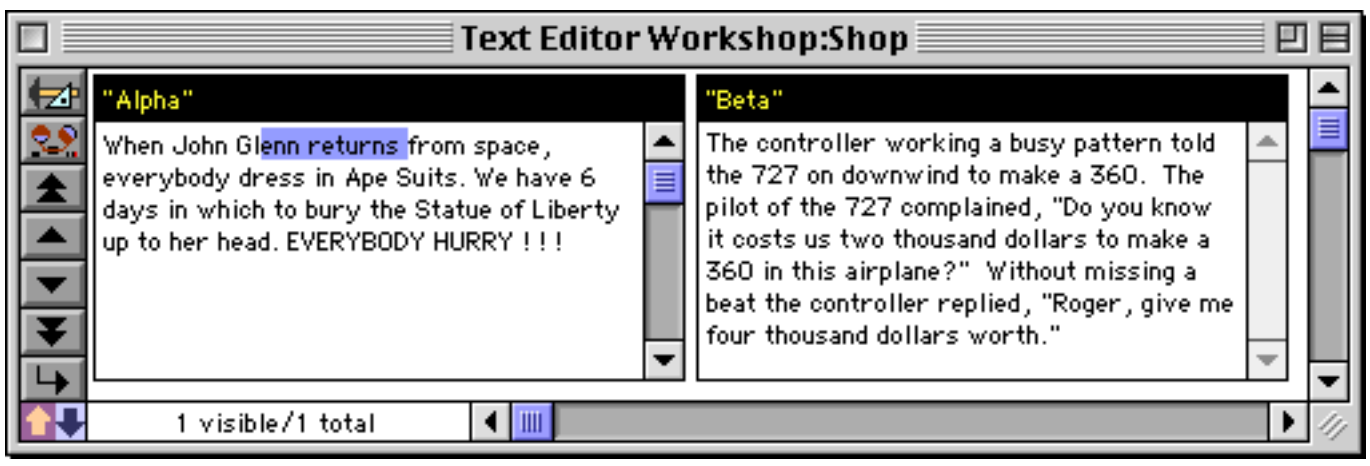
To send a command to a SuperObject the object must have a name. See "Object Type / Object Name" on page 533 to learn how to set or change the name of an object. The form shown below contains two Text Editor SuperObjects, one named Alpha and the other named Beta (these names are completely arbitrary, you can use whatever names you like). In this case the objects have been configured to edit database fields A and B respectively.



This short procedure sends two commands to the Alpha object (the left object). The first command tells the object to open itself for editing (the same as clicking on it). The second command tells the object to select characters 12 through 24 (the same as dragging to select these characters.)

```
superobject "Alpha","Open"
superobject "Alpha","SetSelection",12,24
```

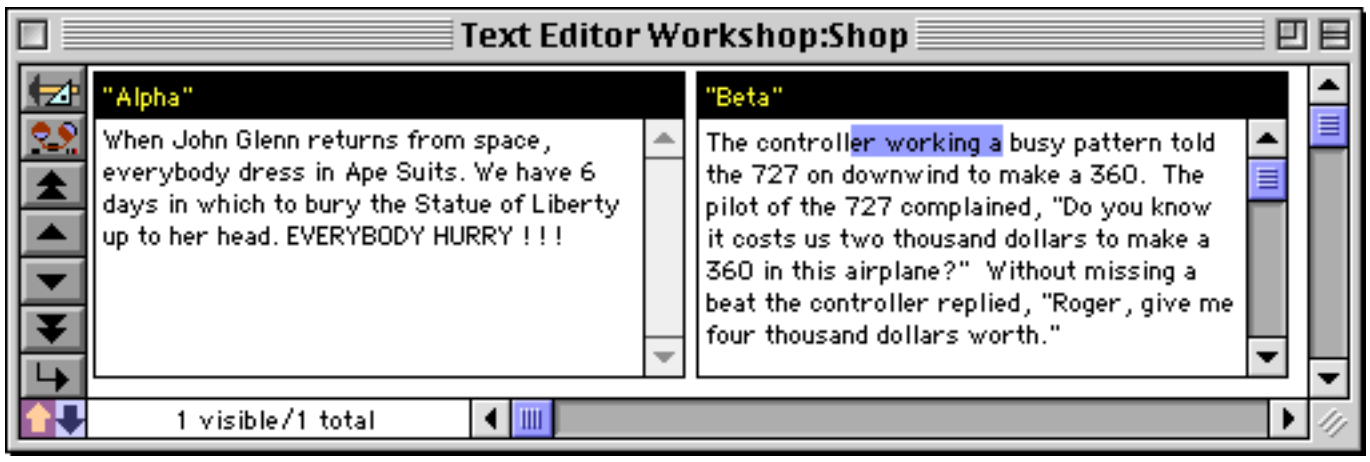
Here's the result of running this procedure.



By changing the first parameter of the `superobject` statement the procedure can control which object the commands are sent to.

```
superobject "Beta","Open"
superobject "Beta","SetSelection",12,24
```

Here's the result of running this revised procedure.



Another option is to specify the object to be manipulated in a separate `object`, `selectobjects` or `objectid` statement (see "Selecting an Object by Name" on page 1753, "Selecting Multiple Objects" on page 1753 and "Object ID Values" on page 1763). The command will be sent to every selected object in the current form.

```
object "Beta"
superobject "","Open"
superobject "","SetSelection"
```

The advantage of this technique is that it makes it possible to control what objects are affected on the fly. For example you could send a command to all blue objects, or all text editor objects that appear in 12 point Times-Roman.

The `superobject` statement normally sends a command to an object(s) in the current window. If you want to send a command to an object in a different window use the `magicwindow` statement (see ""Magic" Windows" on page 1583).

The Active SuperObject

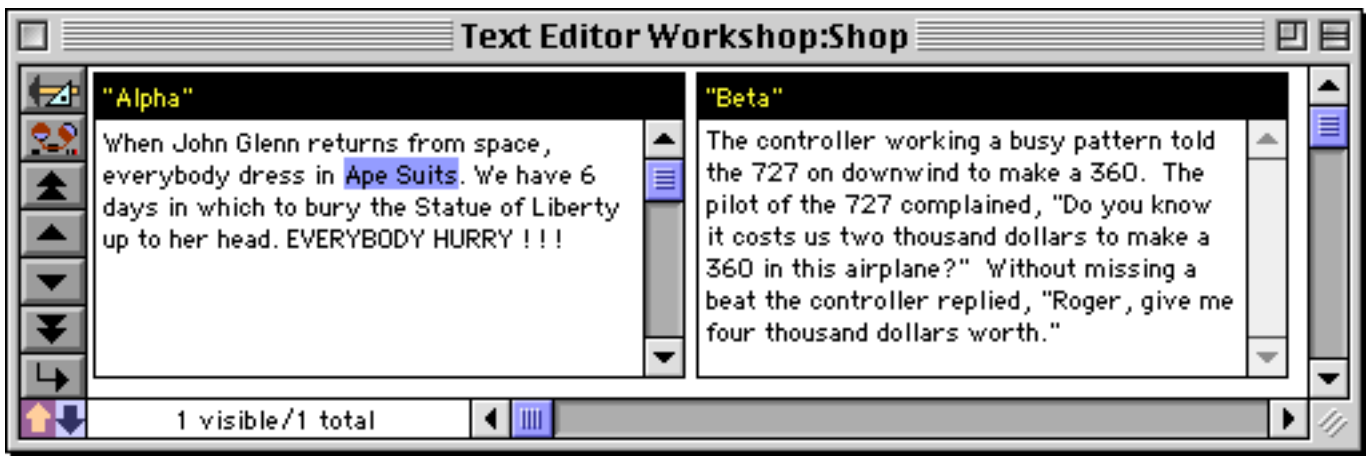
In the case of SuperObjects that edit text (Text Editor, Word Processor) only one object can be "active" at a time. The active object is the object that is currently being edited. If your procedure attempts to send a command to an editor SuperObject that is not active the procedure will stop with an error. For example the following procedure will not work.

```
superobject "Alpha","Open"
superobject "Beta","SetSelection",12,24
```

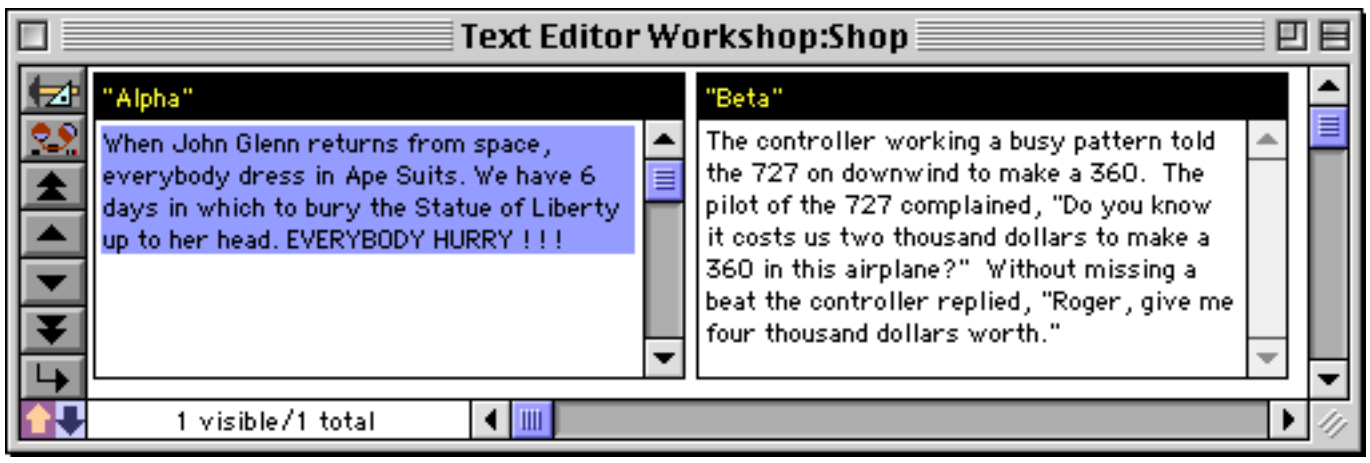
Sometimes you may want a procedure to work with whatever SuperObject happens to be open. This can be done with the `activesuperobject` statement, which always sends commands to the currently active SuperObject. Here is a procedure that will select all of the text that is currently being edited.

```
activesuperobject "SetSelection",0,-1
```

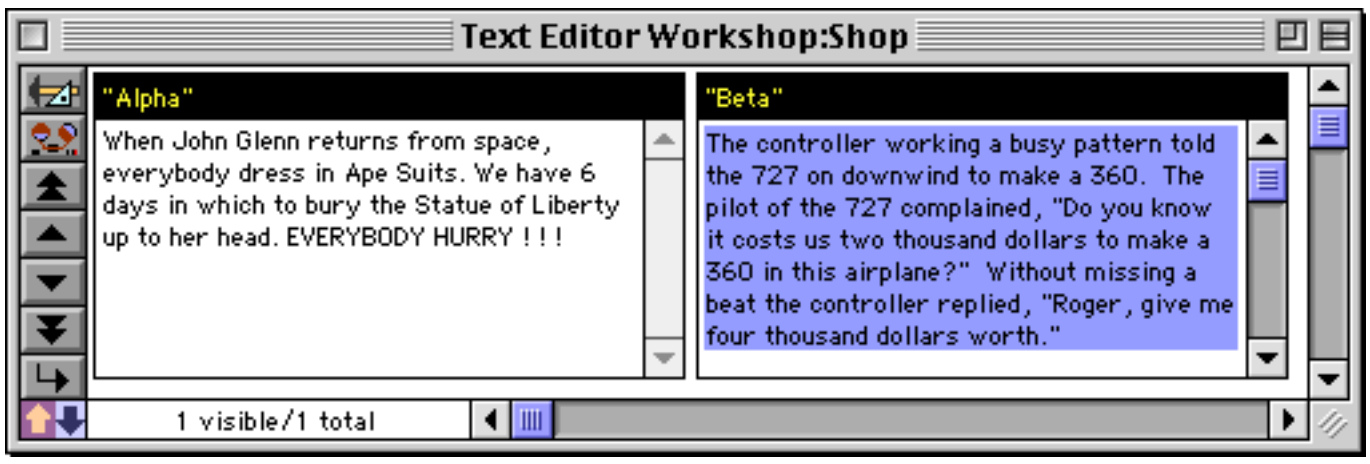
If the text on the left is being edited…



then running this procedure selects all of the text on the left.



But if the text on the right is being edited, then running this procedure selects all of the text on the right.



A procedure can find out which text editor object is active (if any) with the `info("activesuperobject")` function. This function will return the object name of the active object, or `""` if no text is currently being edited.

A procedure can close the currently open text editor object with the `superobjectclose` statement. This statement checks to see if any text editing object (Text Editor or Word Processor) is currently open, and if so, closes it. If none is open, the procedure simply continues. This statement is often useful at the beginning of a procedure where you need to make sure that no text editing is happening before continuing with the procedure.

Accessing and Modifying a SuperObject's Internal Data

Most SuperObject's contain internal data for options and object status. For example a matrix object contains data that specifies the number of rows and columns, while a Super Flash Art object has internal data controlling how the image is aligned within the object. The `objectinfo(` function and `changeobjects` statement each have a "back door" that allows you to access, and in some cases modify this internal object data. Each internal data item is identified with a special identifier that may be used to access the data item. This identifier always begins with a # symbol, for example —

```
#SUPER MATRIX COLUMNS
#SUPER MATRIX ROWS
#SUPER FLASH ART ALIGNMENT
```

A procedure can find out what the current value of an internal data item is by using the `objectinfo(` function with the identifier for that data item. Here is an example procedure that finds out the number of rows and columns in the SuperObject Matrix named Photo Matrix.

```
local mCols,mRows
object "Photo Matrix"
mCols=objectinfo("#SUPER MATRIX COLUMNS")
mRows=objectinfo("#SUPER MATRIX ROWS")
```

Some (but not all) internal data items can be modified by using the `changeobjects` statement with the identifier for that data item. Here is an example that sets the Photo Matrix object to 3 rows by 4 columns.

```
selectobjects objectinfo("name")="Photo Matrix"
changeobjects "#SUPER MATRIX ROWS",4
changeobjects "#SUPER MATRIX COLUMNS",3
```

This mechanism is truly a "back door" — it changes the internal data but it does not cause the object to redraw if necessary. It's up to you as the procedure writer to force the object to redraw somehow, perhaps using the `showpage` statement or by overlaying a Text Display SuperObject with a variable so that a `showvariables` statement forces both objects to redraw.

This mechanism is a "back door" in another sense as well — it doesn't do any error checking. For some internal data items you may be able to change the value to something that doesn't make sense, does not work, or even causes a crash. Be careful, and save your work often.

Internal Data Types

Internal data items come in several flavors, as shown in this table.

Type	Description
Bit	This is a single binary digit, either 0 (off) or -1(on)
Byte	This is a number from 0 to 255 (8 bits)
Word	This is a number from 0 to 65,535 (16 bits)
Long Word	This is a number from 0 to 2,100,000,000 (32 bits)
Text	This is a string of characters

When changing an internal data type you must be careful to supply the correct type of data.

Text Editor SuperObject Commands

The Text Editor SuperObject understands about a dozen commands that can be sent to it with the `superobject` or `activesuperobject` statements in a procedure (see “Program Control of SuperObjects™” on page 1786). This table describes each of these commands in detail.

Command	Parameters	Description

"Open"		This command opens the SuperObject for editing, if it is not already active. This command is the equivalent of clicking on the object to start editing it. Since the object isn't active yet, you can't use the **activesuperobject** statement. The example below opens the Memo SuperObject.
		
		`SuperObject "Memo","Open"`
		`if info("ActiveSuperObject")≠"Memo"`
		`  beep`
		`  stop`
		`endif`
		
		If another data cell or SuperObject is currently active, it's possible that Panorama won't be able to open the SuperObject. If there is an error while attempting to close the currently active item (for example, incorrect date format or an illegal character in a number), the user may choose to cancel and re-edit the incorrect data. The example above checks to make sure that the SuperObject has really been opened for editing—if not, the procedure beeps and stops.

"Close"		This command closes the SuperObject. This is equivalent to pressing the **Enter** key.
		
		If there is an error in the data that was being edited (for example, incorrect date format or an illegal character in a number), the user may choose to cancel and re-edit the incorrect data. The procedure below checks for this and stops if this happens.
		
		`ActiveSuperObject "Close"`
		`if info("ActiveSuperObject")≠""`
		`  stop`
		`endif`
		
		An alternate method for closing the currently active SuperObject is to use the **SuperObjectClose** statement. This statement, which has no parameters, simply closes the currently active SuperObject, if any. Unlike the "**Close**" command, the **SuperObjectClose** statement will not cause an error if there is no SuperObject currently open for editing.

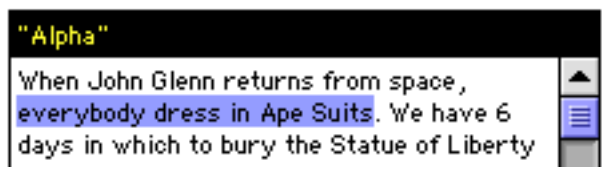
"Cut"		This command copies the currently selected text to the clipboard, then deletes the selected text. This is the same as choosing **Cut** from the Edit Menu. (Technical factoid: The Edit menu actually works by sending this command to the currently active SuperObject.)
"Copy"		This command copies the currently selected text to the clipboard, but does not delete the text. This is the same as choosing **Copy** from the Edit Menu. (Technical factoid: The Edit menu actually works by sending this command to the currently active SuperObject.)
"Paste"		This command pastes the text in the clipboard into the text being edited. The new text will replace any currently selected text, or the text will be inserted at the current insertion point if no text is currently selected. This is the same as choosing **Paste** from the Edit Menu. (Technical factoid: The Edit menu actually works by sending this command to the currently active SuperObject.)
"Clear"		This command deletes the selected text (without copying it to the clipboard). This is the same as choosing **Clear** from the Edit Menu. (Technical factoid: The Edit menu actually works by sending this command to the currently active SuperObject.)

<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"GetSelection"</td><td>Start,End</td><td>
This command gets the start and end points of the currently selected text. For the purpose of the GetSelection command (and the SetSelection command) each character is numbered, starting with zero in front of the first character. For example, if the first character was currently selected, GetSelection will return 0 and 1. If the 3rd through 8th characters are currently selected, GetSelection will return 2 and 8. If there is currently an insertion point, the starting and ending point will be the same. This command only returns the position of the selected text; if you want to get the text itself, use the "GetSelectedText" command.

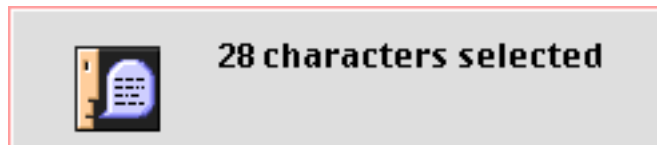
The example procedure below counts and displays the number of characters selected.

<pre>
local SelStartPoint,SelEndPoint
SelStartPoint=0 SelEndPoint=0
if info("activesuperobject")≠""
   ActiveSuperObject "GetSelection",
      SelStartPoint,SelEndPoint
endif
message str(SelEndPoint-SelStartPoint)+
   " characters selected"
</pre>

This procedure checks to make sure that a SuperObject is active. If there is no SuperObject active, it will display the message 0 characters selected. If the procedure did not check, Panorama would stop the procedure and display an error message if there was no active SuperObject. For example, suppose this text was selected.

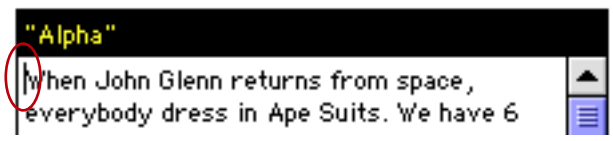
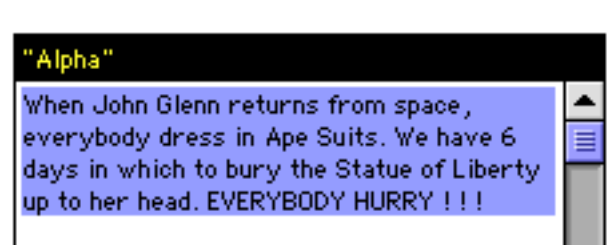
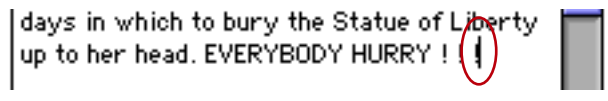


Running this procedure will display this message.



</td></tr>
</table>

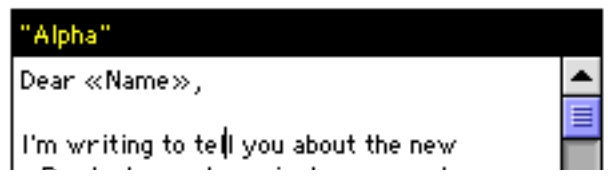
Command	Parameters	Description

"SetSelection"	Start,End	This command allows a procedure to change the selection area. It is equivalent to clicking or dragging on the text to select it. For the purpose of the "**SetSelection**" command (and the "**GetSelection**" command), each character is numbered, starting with zero in front of the first character. For example, the procedure below would put the insertion point in front of the first character in the text.
		
		`if info("activesuperobject")≠""`
		`  ActiveSuperObject "SetSelection",0,0`
		`endif`
		
		Here is the location of the insertion point after running this procedure.
		
		 
		
		The next example will select all of the text. Notice that the end position may be past the end of the text…Panorama will automatically adjust this for you.
		
		`if info("activesuperobject")≠""`
		`  ActiveSuperObject "SetSelection",0,32768`
		`endif`
		
		After this procedure is run all of the text is selected.
		
		 
		
		Here's a similar example that places the insertion point at the end of the text.
		
		`if info("activesuperobject")≠""`
		`  ActiveSuperObject "SetSelection",32768,32768`
		`endif`
		
		Here is the location of the insertion point after running this procedure.
		
		 
		
		This final example will increase the length of the current selection by one character.
		
		`Local SelStartPoint,SelEndPoint`
		`SelStartPoint=0 SelEndPoint=0`
		`if info("activesuperobject")≠""`
		`  ActiveSuperObject "GetSelection",`
		`  SelStartPoint,SelEndPoint`
		`  SelEndPoint=SelEndPoint+1`
		`  ActiveSuperObject "SetSelection",`
		`  SelStartPoint,SelEndPoint`
		`endif`
		
		If the selection was an insertion point it will now be one character, if it was one character it will be two, if two then now three, etc.

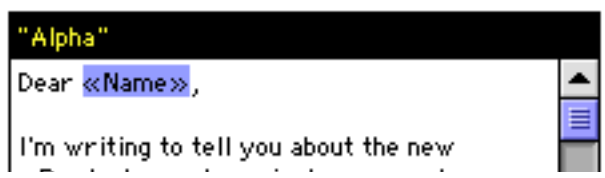
<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"GetText"</td><td>Text</td><td>
This command gets all of the text being edited and puts it in a variable you specify. (Note: If you want only the selected text, use the "GetSelectedText" command.)

The example procedure below searches for text in chevrons («») and if found, selects it. Using this procedure you could create templates with blanks to be filled in, for example ...«Gallery»...«Artist»...«Title». (Of course it might be better to store this information in fields and merge it into the text with a formula.)

<pre>local someText,selStart,selEnd
if info("activesuperobject") = "" stop endif
ActiveSuperObject "GetText",someText
selStart=search(someText,"«")
selEnd=search(someText,"»")
if selStart>0 selStart=selStart-1 endif
if selEnd<selStart selEnd=selStart+1 endif
ActiveSuperObject "SetSelection",selStart,selEnd</pre>
To illustrate this, consider the text being edited below.



After running the procedure, «Name» will be highlighted, like this.

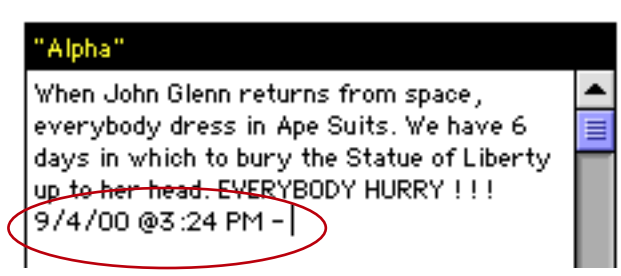


</td></tr>
<tr><td>"SetText"</td><td>Text</td><td>
This command replaces the text currently being edited with completely new text! This is a very powerful command.

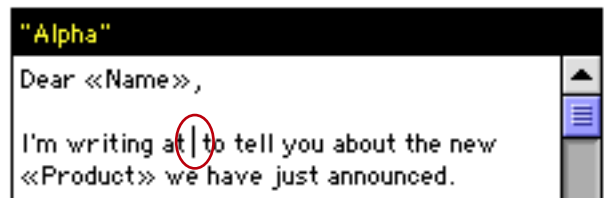
Here is a very simple example that simply erases all of the text. This is similar to Clear, except that all the text is erased, not just the selected text.

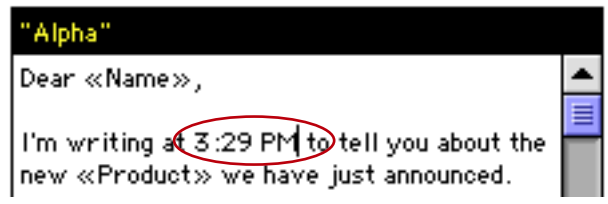
<pre>if info("activesuperobject") = "" stop endif
ActiveSuperObject "SetText",""</pre>
The next example adds a new line with a date and time stamp to the currently edited text. It also moves the insertion point to the end of the new time and date stamp, so the user can immediately type in a note.

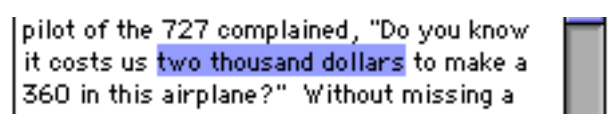
<pre>local someText
if info("activesuperobject") = "" stop endif
ActiveSuperObject "GetText",someText
ActiveSuperObject "SetText",someText+¶+
   datepattern(today(),"mm/dd/yy")+" @"+
   timepattern(now(),"hh:mm am/pm")+" - "
ActiveSuperObject "SetSelection",32768,32768</pre>
Here is the result of running this procedure.

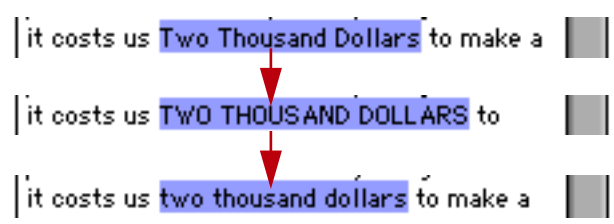


</td></tr>
</table>

<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"InsertText"</td><td>Text</td><td>
This command inserts text. The new text replaces the currently selected text, or is inserted at the insertion point if no text is selected. The example below inserts the current time into the text.
<pre>
if info("activesuperobject") = "" stop endif
ActiveSuperObject "InsertText",
   timepattern(now(),"hh:mm am/pm")
</pre>
To use this procedure start by clicking to set the insertion point where you want the time to be inserted.


Running the procedure inserts the current time.


</td></tr>
<tr><td>"GetSelectedText"</td><td>Text</td><td>
This command gets the selected text and puts it into a variable. The example below uses this command to change the case of the selected text. Each time the procedure is used the case will toggle: if the text is all lower case, it will be converted to initial caps; if it is initial caps, it will be converted to all upper case; otherwise it will be converted to all lower case.
<pre>
local someText,editStart,editEnd
if info("activesuperobject") = "" stop endif
ActiveSuperObject "GetSelection",editStart,editEnd
ActiveSuperObject "GetSelectedText",someText
case someText=lower(someText)
   someText=upperword(someText)
case someText=upperword(someText)
   someText=upper(someText)
defaultcase
   someText=lower(someText)
endcase
ActiveSuperObject "InsertText",someText
ActiveSuperObject "Clear"
ActiveSuperObject "SetSelection",editStart,editEnd
</pre>
To use this procedure, start by selecting some text.


Each time you run the procedure the text is converted to a different upper/lower case combination.


</td></tr>
</table>

Command	Parameters	Description
"Find"		This command displays a dialog asking the user what they would like to find, then locates the word or phrase within the text being edited. This is the same as using the Find in Cell command in the Edit Menu (see “Searching for Text Within the Input Box” on page 355). Another way to find is to use the **search(** function. For an example of this, see the "**GetText**" command earlier in this section.
"FindNext"		This command locates the next occurrence of the word or phrase searched for with the "**Find**" command. This is the same as using the **Find Next in Cell** command in the Edit Menu (see “Searching for Text Within the Input Box” on page 355).

"Change"		This command displays a dialog asking the user what they would like to change, then changes every occurrence it finds in the text being edited. This is the same as using the **Change in Cell** command in the Edit Menu (see “Replacing Words or Phrases Within a Cell” on page 356).
		
		Another way to change is to use the "**GetText**" command and the **replace(** function. The example below replaces the initials rdb with Robert D. Bryce, then moves the insertion point to the end of the text.
		
		`local someText`
		`if info("activesuperobject") = "" stop endif`
		`ActiveSuperObject "GetText",someText`
		`ActiveSuperObject "SetText",`
		`  replace(someText,"rdb","Robert D. Bryce")`
		`ActiveSuperObject "SetSelection",32768,32768`

"Spell"		This command locates the next misspelled word in the text being edited. This is the same as using the **Spelling** command in the Edit Menu (see “Using the Spelling Checker within a Cell” on page 357). Note: This command does not work if the optional Panorama spelling dictionary has not been installed.
"GetScroll"	Vertical,Horizontal	This command returns the status of the scroll bars and returns the amount scrolled (in pixels). If the text is not scrolled this will return 0,0.

"SetScroll"	Vertical,Horizontal	This command scrolls the text. It is the automatic equivalent to dragging on the scroll bars. To scroll to the top, use
		
		`activesuperobject "setscroll",0,0`
		
		To scroll down four inches, use
		
		`activesuperobject "setscroll",72*4,0`

"GetLineCount"	Count	This command querys the object to find out how many lines of text are currently being displayed. Count must be the name of a field or variable to receive the count.

Text Editor Internal Data

This table describes the internal data in a Text Editor SuperObject that can be accessed and modified using the “back door” described in “Internal Data Types” on page 1789. To learn more about how these options work see “Text Editor Options” on page 643.

Identifier	Data Type	Changeable?	Description
"#TEXT EDITOR FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — scroll bars, borders, padding, grow box, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#VERTICAL SCROLL BAR"	Bit	Yes	-1 if vertical scroll bar is enabled, 0 if disabled.
"#HORIZONTAL SCROLL BAR"	Bit	Yes	-1 if horizontal scroll bar is enabled, 0 if disabled.
"#THIN SCROLL"	Bit	Yes	-1 if thin scroll bars are enabled, 0 if disabled.
"#GROW BOX"	Bit	Yes	-1 if grow box is enabled, 0 if disabled.
"#TEXT WRAP"	Bit	Yes	-1 if wrap at end of line is enabled, 0 if disabled.
"#PROCEDURE EVERY KEYSTROKE"	Bit	Yes	-1 if procedure every key is enabled, 0 if disabled.
"#PROCEDURE MOST KEYSTROKES"	Bit	Yes	-1 if procedure most keys is enabled, 0 if disabled.
"#TOP BORDER"	Bit	Yes	-1 if top border is enabled, 0 if disabled.
"#LEFT BORDER"	Bit	Yes	-1 if left border is enabled, 0 if disabled.
"#BOTTOM BORDER"	Bit	Yes	-1 if bottom border is enabled, 0 if disabled.
"#RIGHT BORDER"	Bit	Yes	-1 if right border is enabled, 0 if disabled.
"#AQUA BORDER"	Bit	Yes	-1 if aqua borders (os x style) are enabled, 0 if disabled.
"#3D BORDER"	Bit	Yes	-1 if 3D border is enabled, 0 if disabled.
"#FOCUS RING"	Bit	Yes	-1 if blue editing focus ring is enabled, 0 if disabled.
"#TERMINATE RETURN"	Bit	Yes	-1 if terminate when return is enabled, 0 if disabled.
"#TERMINATE TAB"	Bit	Yes	-1 if terminate when tab is enabled, 0 if disabled.
"#TERMINATE UP/DOWN"	Bit	Yes	-1 if terminate when up/down arrows is enabled, 0 if disabled.
"#NON-WHITE BACKGROUND"	Bit	Yes	-1 if non-white background is enabled, 0 if disabled.
"#UPDATE VARIABLE EVERY KEY"	Bit	Yes	-1 if update variable every key is enabled, 0 if disabled.
"#FOUR SPACE TAB"	Bit	Yes	-1 if tab = 4 spaces is enabled, 0 if disabled.
"#PADDING"	Bit	Yes	-1 if padding is enabled, 0 if disabled.
"#NO NEOTEXT"	Bit	Yes	-1 if alternate editing style is enabled, 0 if disabled.
"#DROP SHADOW DEPTH"	Byte	Yes	0 if drop shadow is disabled. Non zero values specify the drop shadow offset (standard depth is 2 pixels).
"#SELECT STARTUP"	Byte	Yes	Insertion point option. 0 = end of text, 1 = beginning of text, 2 = all text selected.
"#TEXT EDITOR AUTO CAPITALIZATION"	Byte	Yes	Auto caps option. 0 = off, 1= all, 2 = word, 3 = sentence.
"#PROCEDURE"	Text	Yes	Procedure to be triggered automatically.
"#FORMULA"	Text	No	Field name, variable name, or formula.

Text Display SuperObject Internal Data

This table describes the internal data in a Text Display SuperObject that can be accessed and modified using the “back door” described in “Internal Data Types” on page 1789. To learn more about how these options work see “Text Display Options” on page 611.

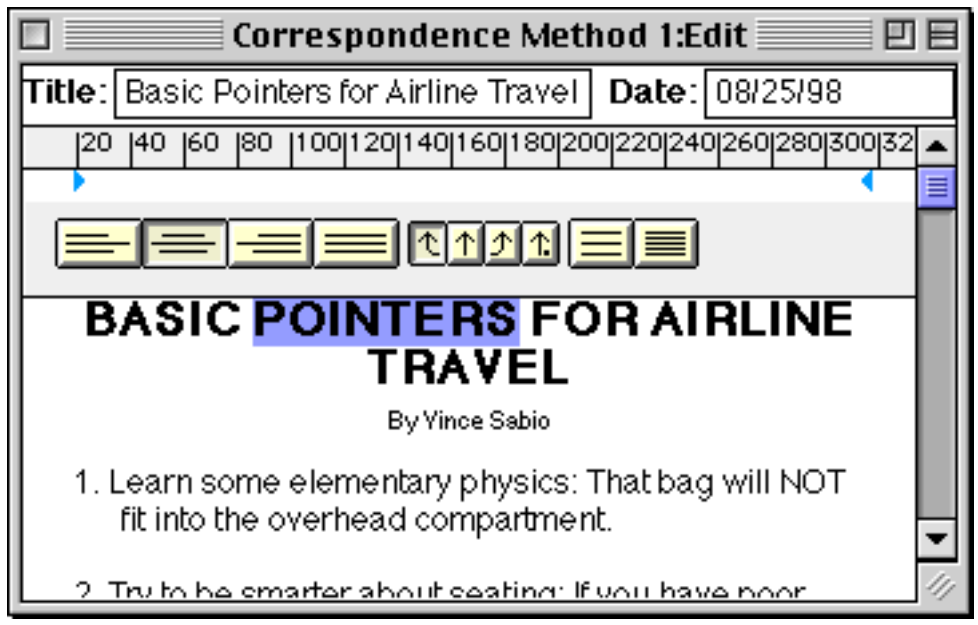
Identifier	Data Type	Changeable?	Description
"#TEXT DISPLAY FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — scroll bars, borders, alignment, grow box, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#VERTICAL SCROLL BAR"	Bit	Yes	-1 if vertical scroll bar is enabled, 0 if disabled.
"#HORIZONTAL SCROLL BAR"	Bit	Yes	-1 if horizontal scroll bar is enabled, 0 if disabled.
"#THIN SCROLL"	Bit	Yes	-1 if thin scroll bars are enabled, 0 if disabled.
"#EVALUATE FORMULA TWICE"	Bit	Yes	-1 if evaluate formula twice is enabled, 0 if disabled.
"#NO TEXT WRAP"	Bit	Yes	-1 if don’t wrap text is enabled, 0 if disabled.
"#AQUA TEXT"	Bit	Yes	-1 if aqua text (anti-aliased) is enabled, 0 if disabled.
"#3D TEXT"	Bit	Yes	-1 if 3D text is enabled, 0 if disabled.
"#AUTO SIZE TEXT"	Bit	Yes	-1 if scale text size is enabled, 0 if disabled.
"#GROW BOX"	Bit	Yes	-1 if grow box is enabled, 0 if disabled.
"#TOP BORDER"	Bit	Yes	-1 if top border is enabled, 0 if disabled.
"#LEFT BORDER"	Bit	Yes	-1 if left border is enabled, 0 if disabled.
"#BOTTOM BORDER"	Bit	Yes	-1 if bottom border is enabled, 0 if disabled.
"#RIGHT BORDER"	Bit	Yes	-1 if right border is enabled, 0 if disabled.
"#DROP SHADOW DEPTH"	Byte	Yes	0 if drop shadow is disabled. Non zero values specify the drop shadow offset (standard depth is 2 pixels).
"#TEXT DISPLAY ALIGNMENT"	Byte	Yes	Align option. 0 = upper left 1 = upper center 2 = upper right 3 = middle left 4 = middle center 5 = middle right 6 = bottom left 7 = bottom center 8 = bottom right
"#TEXT DISPLAY SCALE FACTOR"	Long Word	Yes	If the #AUTO SIZE TEXT option is enabled this value controls the number of lines that will be displayed. The value is an integer that is 100 times the actual value. For example, if you want to display 2.5 lines in the text display object this value must be set to 250.
"#FORMULA"	Text	No	Formula used to display text.

Word Processor SuperObject Commands

The Word Processor SuperObject understands about two dozen commands that can be sent to it with the `superobject` or `activesuperobject` statements in a procedure (see “Program Control of SuperObjects™” on page 1786). Many of these commands are identical or nearly identical to the same commands for the Text Editor SuperObject (see previous section). This table describes each of these commands in detail.

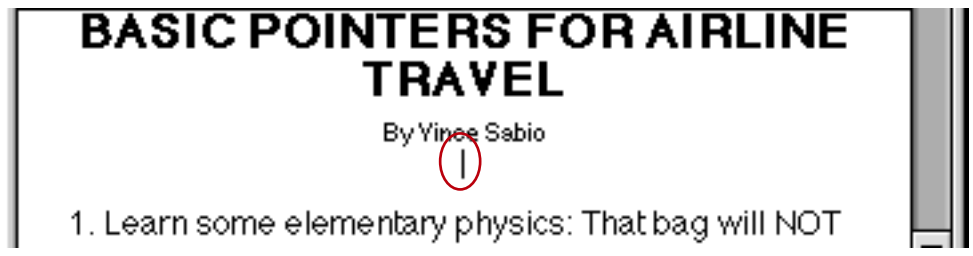
Command	Parameters	Description
"Open"		Identical to Text Editor SuperObject (see Page 1790).
"Close"		Identical to Text Editor SuperObject (see Page 1790).
"Undo"		Undo’s the most recent editing operation.
"Cut"		Identical to Text Editor SuperObject (see Page 1790).
"Copy"		Identical to Text Editor SuperObject (see Page 1790).
"Paste"		Identical to Text Editor SuperObject (see Page 1790).
"Clear"		Identical to Text Editor SuperObject (see Page 1790).
"GetSelection"	Start,End	Identical to Text Editor SuperObject (see Page 1791).
"SetSelection"	Start,End	Identical to Text Editor SuperObject (see Page 1792).
"GetText"	Text	Almost identical to Text Editor SuperObject (see Page 1793). However, only the text itself is copied into the variable. Style information (font, size, bold, italic, etc.) is not copied into the variable.
"SetText"	Text	Almost identical to Text Editor SuperObject (see Page 1793). However, the new text is inserted into the document using the default font and style. All pre-existing style information (font, size, bold, italic, etc.) in the document is removed. To insert text without disturbing the style of existing text use the "**InsertText**" command.
"InsertText"	Text	Almost identical to Text Editor SuperObject (see Page 1794). The new text is inserted into the document using the font and style of the text at the current insertion point.
"GetSelectedText"	Text	Almost identical to Text Editor SuperObject (see Page 1794). However, only the text itself is copied into the variable. Style information (font, size, bold, italic, etc.) is not copied into the variable.
"Find"		Identical to Text Editor SuperObject (see Page 1795).
"FindNext"		Identical to Text Editor SuperObject (see Page 1795).
"Change"		Identical to Text Editor SuperObject (see Page 1795).
"Spell"		Identical to Text Editor SuperObject (see Page 1795).

Command	Parameters	Description

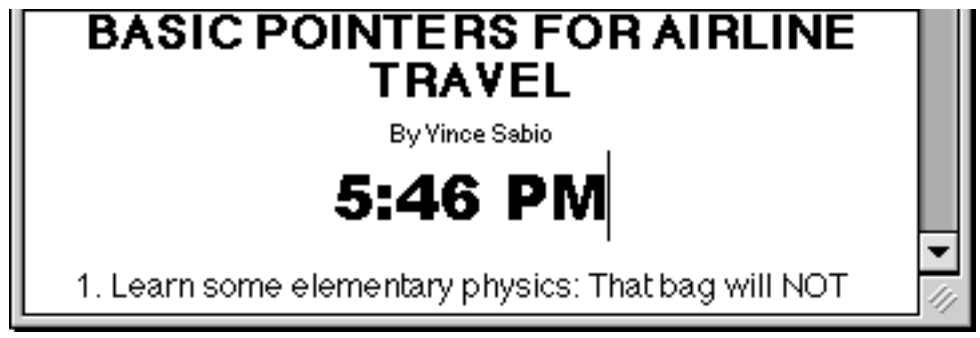
"GetFont"	FontName	This command gets the name of the font used for the selected text. If the selected text contains more than one font, only the first font is listed. The example below displays the font of the selected text.
		
		`local MyFont,MyFontSize`
		`ActiveSuperObject "GetFont",MyFont`
		`ActiveSuperObject "GetFontSize",MyFontSize`
		`message "The current font is: "+`
		`  str(MyFontSize)+"pt "+MyFont`
		
		For example, suppose the word Pointers is selected as shown here.
		
		 
		
		Running this procedure displays the current font and size of this word.
		
		 

<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"SetFont"</td><td>FontName</td><td>This command changes the font of the selected text. All the selected text is changed to the same font. The example below inserts the current time into the text using 24 point Arial Black.

<pre>
if info("activesuperobject") = "" stop endif
ActiveSuperObject "SetFont","Arial Black"
ActiveSuperObject "SetFontSize",24
ActiveSuperObject "InsertText",
   timepattern(now(),"hh:mm am/pm")
</pre>
To use this procedure start by selecting an insertion point.



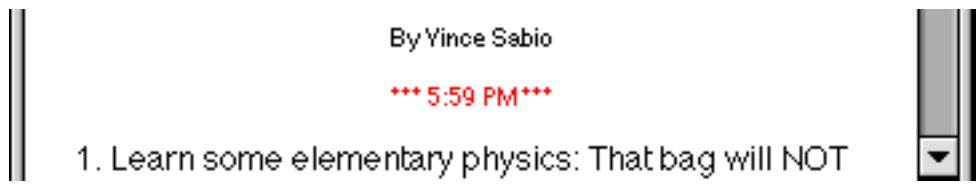
Then run the procedure to insert the time.



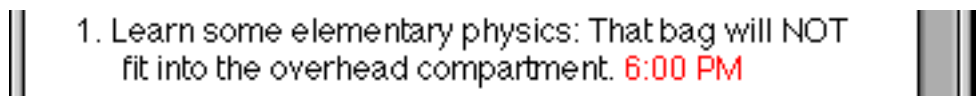
Any additional text inserted at this point will also use 24 pt Arial Black.</td></tr>
<tr><td>"GetFontSize"</td><td>Size</td><td>This command gets the font size of the selected text. If the selected text contains more than one size, only the first size is listed. See "GetFont" (Page 1799) for an example illustrating this command.</td></tr>
<tr><td>"SetFontSize"</td><td>Size</td><td>This command changes the size of the selected text. All the selected text is changed to the same size. See "SetFont" (Page 1800) for an example using this command.</td></tr>
</table>

<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"GetJustification"</td><td>Alignment</td><td>This command gets the text justification status of the selected text. The result may be one of these values: Left, Center, Right or Full. If the selected text contains more than one justification, only the first justification is listed. Here is another procedure that inserts a time stamp into the file.

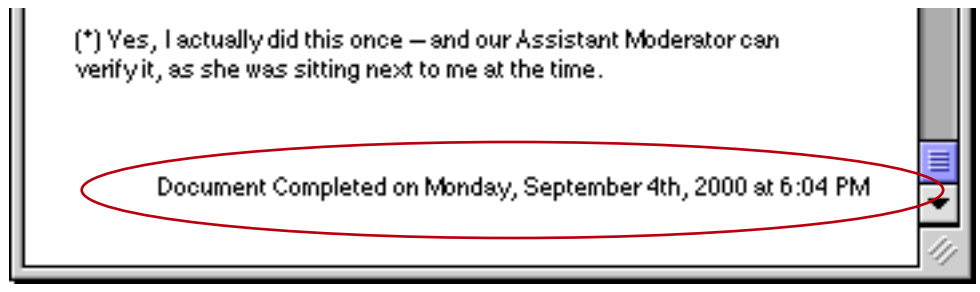
<pre>local theStamp,textAlign
theStamp=timepattern(now(),"hh:mm am/pm")
ActiveSuperObject "GetJustification",textAlign
if textAlign="Center"
   theStamp="*** "+theStamp+" ***"
endif
ActiveSuperObject "SetTextColor",rgb(65535,0,0)
ActiveSuperObject "InsertText",theStamp</pre>
If the text is inserted in centered text three asterisks are added on each side of the time.



If the text is inserted in left or right justified text only the time is inserted.


</td></tr>
<tr><td>"SetJustification"</td><td>Alignment</td><td>This command changes the justification of the selected text. The new justification may be one of these values: Left, Center, Right or Full. All the selected text is changed to the same justification. The example below adds a new, right justified line to the end of the document.

<pre>if info("activesuperobject") = "" stop endif
ActiveSuperObject "SetSelection",999999,999999
ActiveSuperObject "InsertText",¶+¶
ActiveSuperObject "SetJustification","Right"
ActiveSuperObject "InsertText",
   "Document Completed on "+
   datepattern(today(),
      "DayOfWeek, Month ddnth, yyyy")+" at "+
   timepattern(now(),"hh:mm am/pm")</pre>
Here is the result of running this procedure.


</td></tr>
</table>

<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"GetLeading"</td><td>Spacing</td><td>This command gets the leading of the selected text (an integer). If the selected text contains more than one leading, only the first leading is returned. For normal single spaced text the leading value is zero.

Michael Jordan will make over $300,000 a game: $10,000 a minute, assuming he averages about 30 minutes per game.

For 12 point text a leading value of 6 is about 1 1/2 spacing.

Michael Jordan will make over $300,000 a game:
$10,000 a minute, assuming he averages about 30
minutes per game.

This text has a leading value of 12, which is double spaced for this font size.

Michael Jordan will make over $300,000 a game:

$10,000 a minute, assuming he averages about 30

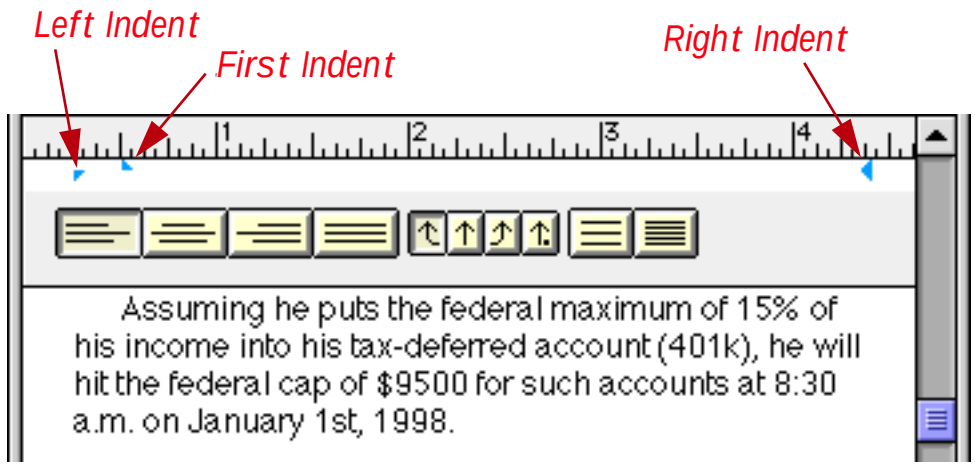
minutes per game.

You can set the leading of selected text with the ruler or the Paragraph Settings dialog (see “Line Spacing” on page 691).</td></tr>
<tr><td>"SetLeading"</td><td>Spacing</td><td>This command changes the leading of the selected text. All the selected text is changed to the same leading. For normal single spaced text the leading value should be zero.

Here is a procedure that makes the currently selected text double spaced.

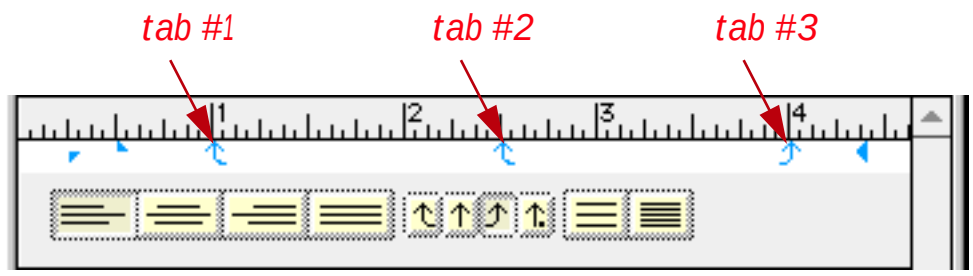
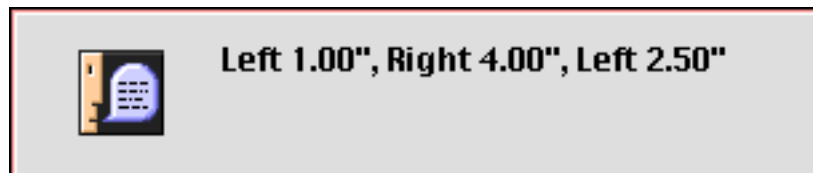
<pre>local fontSize
activesuperobject "GetFontSize",fontSize
activesuperobject "SetLeading",fontSize</pre>
This procedure makes the currently selected text single spaced.

<pre>activesuperobject "SetLeading",0</pre></td></tr>
</table>

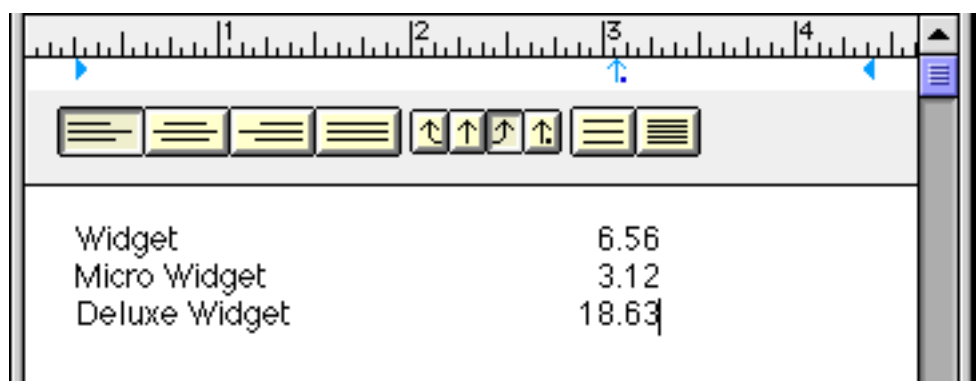
<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"GetLeftIndent"
"GetRightIndent"
"GetFirstIndent"</td><td>Indent</td><td>These three commands get the indents distances of the selected text. If the selected text contains more than one indent value, only the first indent value is returned. The "GetLeftIndent" and "GetRightIndent" commands return the left and right indent values, respectively. The "GetFirstIndent" command returns the indent of the first line of the paragraph.
 

All indent values are specified in points (72 points per inch). See “Margins (Indents)” on page 682 to learn how to set indents with the ruler or Paragraph Settings dialog.</td></tr>
<tr><td>"SetLeftIndent"
"SetRightIndent"
"SetFirstIndent"</td><td>Indent</td><td>These three commands change the indents of the selected text. All the selected text is changed to the same indents. All indent values are specified in points (72 points per inch). The example below sets the margins for the currently selected text at 1/2 inch (36 points).
<pre>ActiveSuperObject "SetLeftIndent",36
ActiveSuperObject "SetFirstIndent",36
ActiveSuperObject "SetRightIndent",36</pre></td></tr>
<tr><td>"ClearTabs"</td><td></td><td>This command clears all tabs from the selected text.</td></tr>
</table>

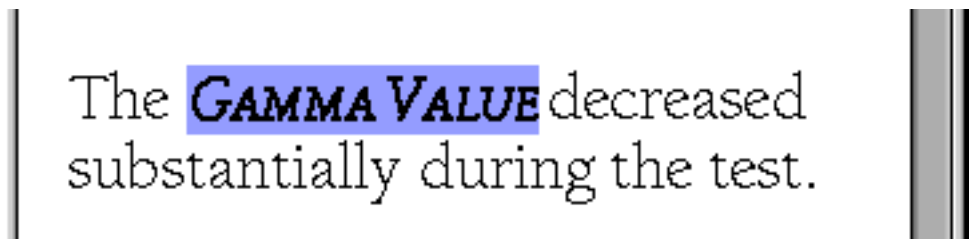
Command	Parameters	Description

"GetTab"	Tab,Position,Type,Leader	This command gets information about a tab stop active with the currently selected text (see "Tab Stops" on page 685). The first parameter, Tab, is the number of the tab you want to get information about (starting with 1).
		
		 
		
		The remaining three parameters are filled in by the command. The Position is the position of the tab, in points. The Type is the type of tab. The possible types are Left, Center, Right, Decimal and None. A Type of None indicates that the requested tab does not exist. In that case values of the Position and Leader characters are not defined. The Leader parameter is the tab leader character, if any. The example below will display a list of the current tab stops.
		
		`local tabList,theTab,tabSpot,tabType,tabLeader`
		`theTab=1`
		`tabList=""`
		`tabType="None"`
		`loop`
		`  ActiveSuperObject "GetTab",theTab,`
		`  tabSpot,tabType,tabLeader`
		`  stoploopif tabType = "None"`
		`  tabList=sandwich("",tabList,", ")+`
		`  tabType+" "+pattern(tabSpot/72,"#.##")+{"}`
		`  theTab=theTab+1`
		`while forever`
		`message tabList`
		
		When you run this procedure it will display a message something like this.
		
		 
		
		There is no "**SetTab**" command. To change a tab setting you must clear all tabs and then use the "**AddTab**" command.

Command	Parameters	Description

"AddTab"	Position,Type,Leader	This command adds a new tab stop. The Position is the position of the tab, in points. The Type is the type of tab. The possible types are Left, Center, Right and Decimal. The Leader parameter is the tab leader character, if any. The example below will add a tab stop and then add several lines of pricing information.
		
		`if info("activesuperobject") = "" stop endif`
		`ActiveSuperObject "SetSelection",999999,999999`
		`ActiveSuperObject "InsertText",¶`
		`ActiveSuperObject "ClearTabs"`
		`ActiveSuperObject "AddTab",220,"Decimal",""`
		`ActiveSuperObject "InsertText",`
		`"Widget"+¬+"6.56"+¶+`
		`"Micro Widget"+¬+"3.12"+¶+`
		`"Deluxe Widget"+¬+"18.63"`
		
		Here is the finished result of this procedure.
		
		 

Command	Parameters	Description

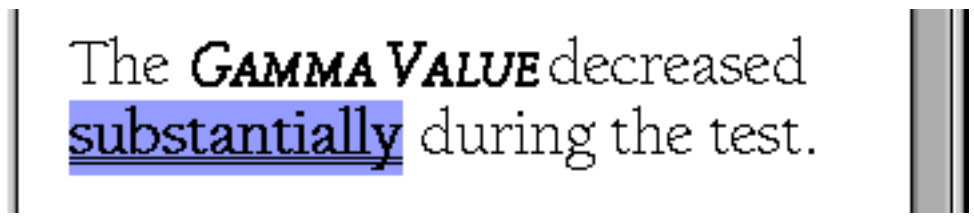
"GetStyle"	StyleName,Status	This command will check the selected text to see if it is a certain style. If there is more than one style in the selected text, the style of the first character will be returned. If the selected text matches the specified cell the result is -1, if it does not match, the result is 0. The style names are:
		
		`Plain Bold Italic Outline`
		`Shadow Condensed Extended Hidden Text`
		`Strikeout SuperScript SubScript SmallCaps`
		`AllCaps AllLowerCase FormulaMerge UnderLine`
		`DoubleUnderLine WordUnderLine`
		`DottedUnderLine OverLine`
		
		A text selection may contain more than one of these styles. You must test for each style separately. Here is procedure that makes a list of all the styles enabled for the first character of the currently selected text.
		
		`local allstyles,n,checkstyle,stylelist,styletrue`
		`allstyles="Plain,Bold,Italic,Outline,Shadow,"+`
		`"Condensed,Extended,Hidden Text,"+`
		`"Strikeout,SuperScript,SubScript,"+`
		`"SmallCaps,AllCaps,AllLowerCase,"+`
		`"FormulaMerge,UnderLine,"+`
		`"DoubleUnderLine,WordUnderLine,"+`
		`"DottedUnderLine,OverLine"`
		
		`stylelist=""`
		`n=1`
		`loop`
		`checkstyle=array(allstyles,n,",")`
		`stoploopif checkstyle=""`
		`activesuperobject "GetStyle",`
		`checkstyle,styletrue`
		`if styletrue`
		`stylelist=`
		`sandwich("",stylelist,",")+checkstyle`
		`endif`
		`n=n+1`
		`while forever`
		`message stylelist`
		
		This text illustrates the operation of the procedure.
		
		 
		
		This text has three styles — bold, italic, and small caps.
		
		 

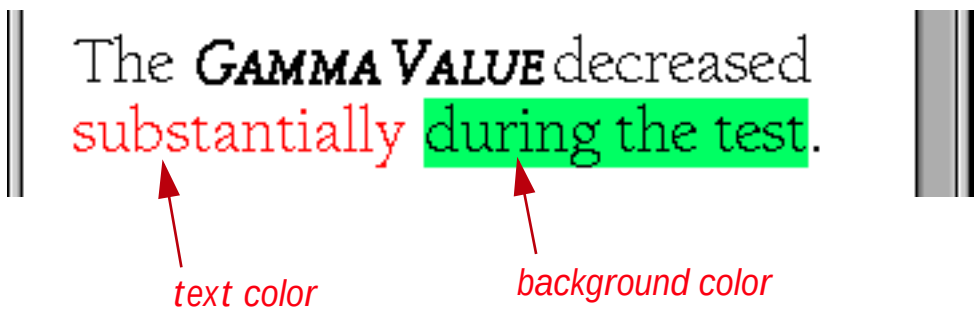
<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"SetStyle"</td><td>StyleName,Status</td><td>This command will change the style of the selected text. A procedure can only set one style at a time, from this list.

<pre>Plain   Bold   Italic   Outline
Shadow   Condensed   Extended   Hidden Text
Strikeout SuperScript   SubScript   SmallCaps
AllCaps   AllLowerCase FormulaMerge UnderLine
DoubleUnderLine   WordUnderLine
DottedUnderLine   OverLine</pre>
If the Status is -1, the specified style is turned on. If the Status is 0, the specified style is turned off. The style names are listed in the previous section.

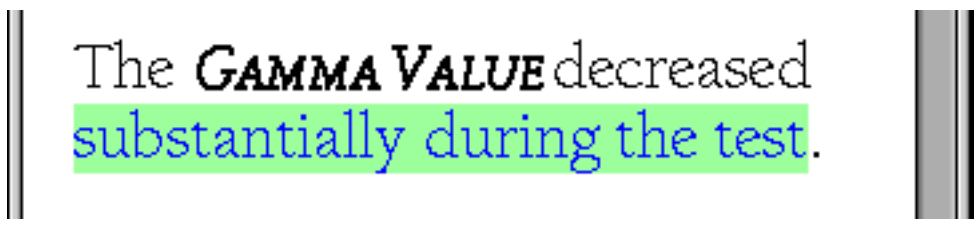
The "SetStyle" command adds or subtracts the specified style from the styles the selected text already has. If you want to make sure the selected text has only the styles you specify, start by making the text plain. The example below sets the selected text to bold double underline.

<pre>ActiveSuperObject "SetStyle","Plain",-1
ActiveSuperObject "SetStyle","Bold",-1
ActiveSuperObject "SetStyle","DoubleUnderLine",-1</pre>
Whatever text is selected when this procedure is run will be made bold with a double underline.


</td></tr>
<tr><td>"GetTextColor"
"GetTextBack-groundColor"</td><td>Color</td><td>These two commands will return the color of the selected text. See “Colors” on page 1294 for a complete discussion of colors.

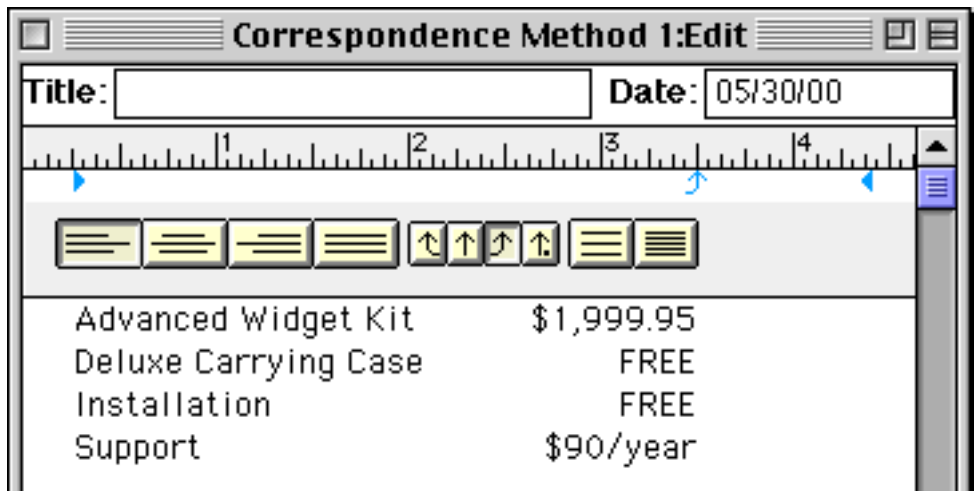

</td></tr>
<tr><td>"SetTextColor"
"SetTextBack-groundColor"</td><td>Color</td><td>This command will set the color of the selected text. See “Colors” on page 1294 for a complete discussion of colors. The example below sets the selected text to a pure blue on a light green background.

<pre>ActiveSuperObject "SetTextColor",rgb(0,0,65535)
ActiveSuperObject "SetTextBackGroundColor",
   rgb(40000,65535,40000)</pre>
Here is the result of selecting text and running this procedure.


</td></tr>
</table>

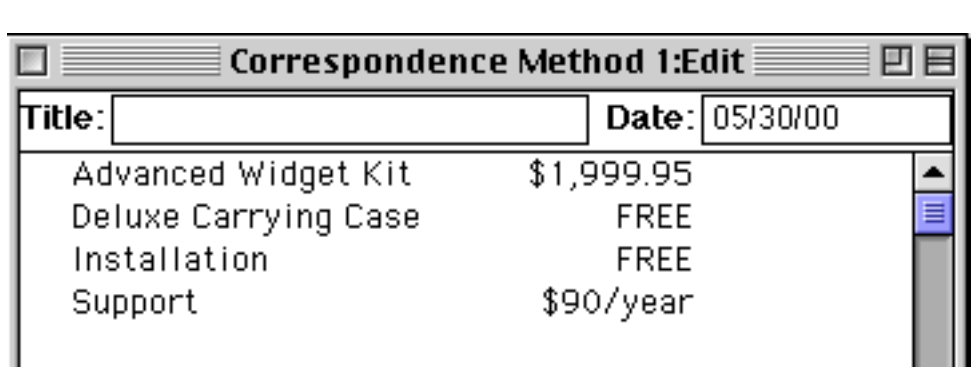
<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"ShowRuler"</td><td>Status</td><td>This command will turn the display of the ruler on and off. If the Status is -1, the ruler will be visible; if the Status is 0, the ruler will not be visible. The example below makes sure the ruler is visible.

<pre>ActiveSuperObject "ShowRuler",-1</pre>
The ruler allows you to manually set indents, alignment and tab stops.



This procedure makes the ruler invisible.

<pre>ActiveSuperObject "ShowRuler",0</pre>
With the ruler turned off you can still edit text, but changing indents, alignment and tab stops can only be done through dialogs.


</td></tr>
<tr><td>"LockDocument"</td><td>Status</td><td>This command allows the document to be locked. If the Status is -1, the document will be locked and cannot be edited. If the Status is 0, the document will be unlocked and may be edited again. The example below locks the current document.

<pre>ActiveSuperObject "LockDocument",-1</pre></td></tr>
</table>

Word Processor Internal Data

This table describes the internal data in a Word Processor SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "Configuring the Word Processor" on page 696.

Identifier	Data Type	Changeable?	Description
"#WORD PROCESSOR FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — scroll bars, borders, padding, grow box, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#VERTICAL SCROLL BAR"	Bit	Yes	-1 if vertical scroll bar is enabled, 0 if disabled.
"#HORIZONTAL SCROLL BAR"	Bit	Yes	-1 if horizontal scroll bar is enabled, 0 if disabled.
"#TEXT WRAP"	Bit	Yes	-1 if wrap at end of line is enabled, 0 if disabled.
"#PROCEDURE EVERY KEYSTROKE"	Bit	Yes	-1 if procedure every key is enabled, 0 if disabled.
"#PROCEDURE MOST KEYSTROKES"	Bit	Yes	-1 if procedure most keys is enabled, 0 if disabled.
"#PROCEDURE ON DEACTIVE"	Bit	Yes	-1 if procedure termination is enabled, 0 if disabled.
"#TOP BORDER"	Bit	Yes	-1 if top border is enabled, 0 if disabled.
"#LEFT BORDER"	Bit	Yes	-1 if left border is enabled, 0 if disabled.
"#BOTTOM BORDER"	Bit	Yes	-1 if bottom border is enabled, 0 if disabled.
"#RIGHT BORDER"	Bit	Yes	-1 if right border is enabled, 0 if disabled.
"#TERMINATE RETURN"	Bit	Yes	-1 if terminate when return is enabled, 0 if disabled.
"#TERMINATE TAB"	Bit	Yes	-1 if terminate when tab is enabled, 0 if disabled.
"#TERMINATE UP/DOWN"	Bit	Yes	-1 if terminate when up/down arrows is enabled, 0 if disabled.
"#NON-WHITE BACKGROUND"	Bit	Yes	-1 if non-white background is enabled, 0 if disabled.
"#UPDATE VARIABLE EVERY KEY"	Bit	Yes	-1 if update variable every key is enabled, 0 if disabled.
"#3D BORDER"	Bit	Yes	-1 if 3D border is enabled, 0 if disabled.
"#GROW BOX"	Bit	Yes	-1 if grow box is enabled, 0 if disabled.
"#OVERFLOW PRINTING"	Bit	Yes	-1 if handle overflow is enabled, 0 if disabled.
"#FILE ON DISK"	Bit	Yes	-1 if file on disk is enabled, 0 if disabled.
"#DROP SHADOW DEPTH"	Byte	Yes	0 if drop shadow is disabled. Non zero values specify the drop shadow offset (standard depth is 2 pixels).
"#TEXT EDITOR AUTO CAPITALIZATION"	Byte	Yes	Auto caps option. 0 = off, 1= all, 2 = word, 3 = sentence.
"#PROCEDURE"	Text	Yes	Procedure to be triggered automatically.
"#FORMULA"	Text	No	Field name, variable name, or formula.

Super Flash Art Commands (Including Movie Control)

The Super Flash Art SuperObject understands about a dozen commands that can be sent to it with the `superobject` statement in a procedure (see “Program Control of SuperObjects™” on page 1786). Many of these commands work with QuickTime movies (see “Displaying Movies in a Form” on page 810). This table describes each of these commands in detail.

Command	Parameters	Description

"Dimensions"	Rectangle	This command obtains the original height and width of the currently displayed image and places it in the rectangle you supply (usually a variable, see “Rectangles” on page 1290). The dimensions are in pixels (1/72 inch). You should use the **rheight(** and **rwidth(** functions to extract the height and width of the rectangle.
		
		Here is a procedure that examines the photo being displayed in the object named Photo and decides whether it is portrait, landscape, or panoramic.
		
		`local photoRect,photoHeight,photoWidth`
		`superobject "Photo","Dimensions",photoRect`
		`photoHeight=rheight(photoRect)`
		`photoWidth=rwidth(photoRect)`
		`case photoWidth>photoHeight*2`
		`  message "Panoramic Photo"`
		`case photoWidth>photoHeight`
		`  message "Landscape Photo"`
		`default`
		`  message "Portrait Photo"`
		`endcase`

"FindText"	Point,Text	This command only works when displaying an Apple PICT format image that contains vector text (not bitmap). (See “Preparing Pictures with Extractable Text” on page 1816.) When the procedure issues this command it must supply an x,y position (Point) within the PICT image. The command will scan the image and see if there is any text at this point. If so, the command will fill in the Text parameter (usually a variable) with the text. This command is usually used to create a web like hypertext system within Panorama — see “Building Web Like HyperText Systems with Super Flash Art” on page 1816.
"ExtractText"	Font,Size,Style,Sep,Text	This command only works when displaying an Apple PICT format image that contains vector text (not bitmap). (See “Preparing Pictures with Extractable Text” on page 1816.) The procedure specifies what Font, Size, and Style it wants to extract (for example "Helvetica",12,0) and the command scans the image looking for text that matches. If it finds any it is placed in the Text parameter. If there is more than one section of text that matches the sections are appended together with the Sep character in between. For more detail about this command see “Extracting All Text of a Specific Style” on page 1820.

"GetDuration"	Time	This command allows you to get the duration of a movie. The result is in 600ths of a second. This example displays the length of the currently playing movie.
		
		`local mTime`
		`superobject "MyMovie","GetDuration",mTime`
		`message "Movie Length: "+str(mTime/600))+" seconds"`

<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"GetRate"
"SetRate"</td><td>Rate</td><td>These commands allows you to get and set the current movie playback speed. If a movie is stopped, the rate is zero. Normal speed is 65536, 1/2 speed (slow motion) is 32768, double speed is 131072. The example below cuts the current playback speed in half.

<pre>local mSpeed
superobject "MyMovie","GetRate",mSpeed
superobject "MyMovie","SetRate",mSpeed/2</pre></td></tr>
<tr><td>"GetPreferred Rate"
"SetPreferred Rate"</td><td>Rate</td><td>These commands allows you to get and set the default movie playback speed. Normal speed is 65536, 1/2 speed (slow motion) is 32768, double speed is 131072. The example below restores the default playback speed (for example, after you have set the speed to slow motion. (Note: Pressing the Play button on the movie controller automatically sets the playback speed to the preferred speed).

<pre>local mSpeed
superobject "MyMovie","GetPreferredRate",mSpeed
superobject "MyMovie","SetRate",mSpeed</pre></td></tr>
<tr><td>"GetTime"
"SetTime"</td><td>Position</td><td>These commands allows you to get and set the current movie playback location. You can use these commands to implement "bookmarks" within a movie. The value returned by the GetTime command is a binary value that represents the location within the movie. It is not a number (# of seconds, etc.) and cannot be used in calculations within Panorama. However, you can pass this value to the SetTime command to move the movie back to this location later. The example below shows two procedures for creating and using bookmarks within a movie. The first procedure adds a “bookmark” recording the current spot within the movie.

<pre>/* Procedure 1: Add a bookmark */
global movieMarks
define movieMarks,""
local markName,markSpot
markName=""
gettext "Bookmark Name",markName
superobject "MyMovie","GetTime",markSpot
movieMarks=sandwich("",movieMarks,¶)+
   markName+chr(9)+radixstr("hex",markSpot)</pre>
The second procedure is designed to be used with a pop-up menu or list superobject. When the user selects a bookmark in the menu or list the movie jumps to that spot.

<pre>/* Procedure 2: Goto a bookmark */
global movieMarks,movieSpot
/* assume movieSpot contains name of bookmark,
perhaps from pop-up menu */
local movieMark,markSpot,mNum
mNum=arraysearch(
   movieMarks,movieSpot+chr(9)+"*",1,¶)
movieMark=array(movieMarks,mNum,¶)
movieMark=array(movieMark,2,chr(9))
markSpot=radix("hex",movieMark)
superobject "MyMovie","SetTime",markSpot</pre></td></tr>
</table>

Command	Parameters	Description

"GetVolume"
"SetVolume"
	Level	These commands allows you to get and set the current movie playback volume. Full volume is 255, zero volume is 0. The example below cuts the current playback volume in half.
		
		`local mVol`
		`superobject "MyMovie","GetVolume",mVol`
		`superobject "MyMovie","SetVolume",mVol/2`

"GetPreferred Volume"	Level	This command allows you to get the default movie playback volume. Full volume is 255, zero volume is 0. The example below sets the current playback volume to 1/3 of the default.
		
		`local mVol`
		`superobject "MyMovie","GetPreferredVolume",mVol`
		`superobject "MyMovie","SetVolume",mVol/3`

"Play"		This command starts the movie playing from the current location. `superobject "MyMovie","Play"`
"Stop"		This command stops the movie playing. `superobject "MyMovie","Stop"`
"GoToBeginning"		This command resets the current location to the beginning of the movie. `superobject "MyMovie","GoToBeginning"`
"GoToEnd"		This command resets the current location to the end of the movie. `superobject "MyMovie","GoToEnd"`

"PlayFinished"		This command checks to see if the movie has reached the end. The example starts the movie playback. If the movie is already at the end, the procedure resets the movie to the beginning before beginning playback.
		
		`local mStatus`
		`superobject "MyMovie","PlayFinished",mStatus`
		`if mStatus=1`
		`  superobject "MyMovie","GoToBeginning"`
		`endif`
		`superobject "MyMovie","Play"`

Super Flash Art Internal Data

This table describes the internal data in a Super Flash Art SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "Super Flash Art™ Options" on page 777.

Identifier	Data Type	Changeable?	Description
"#SUPER FLASH ART FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — scroll bars, borders, grow box, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#VERTICAL SCROLL BAR"	Bit	Yes	-1 if vertical scroll bar is enabled, 0 if disabled.
"#HORIZONTAL SCROLL BAR"	Bit	Yes	-1 if horizontal scroll bar is enabled, 0 if disabled.
"#INCLUDE PICTURES ON DISK"	Bit	Yes	-1 if include pictures on disk is enabled, 0 if disabled.
"#DISPLAY GROUP OF PICTURES"	Bit	Yes	-1 if display group of pictures is enabled, 0 if disabled.
"#TOP BORDER"	Bit	Yes	-1 if top border is enabled, 0 if disabled.
"#LEFT BORDER"	Bit	Yes	-1 if left border is enabled, 0 if disabled.
"#BOTTOM BORDER"	Bit	Yes	-1 if bottom border is enabled, 0 if disabled.
"#RIGHT BORDER"	Bit	Yes	-1 if right border is enabled, 0 if disabled.
"#GROW BOX"	Bit	Yes	-1 if grow box is enabled, 0 if disabled.
"#OVERFLOW PRINTING"	Bit	Yes	-1 if overflow is enabled, 0 if disabled (see "Printing Multiple Page Records" on page 1102).
"#DROP SHADOW DEPTH"	Byte	Yes	0 if drop shadow is disabled. Non zero values specify the drop shadow offset (standard depth is 2 pixels).
"#SUPER FLASH ART ALIGNMENT"	Byte	Yes	Align option (see "Align" on page 788). 0 = upper left 1 = upper center 2 = upper right 3 = middle left 4 = middle center 5 = middle right 6 = bottom left 7 = bottom center 8 = bottom right 9 = scale to fit 10 = scale to fit (proportional) 11 = tile
"#FLASH ART FILE"	Text	Yes	Alt File (tells Panorama to look in another database for Flash Art scrapbook, see "Alt File" on page 779).
"#FLASH ART DEFAULT CAPTION"	Text	Yes	Default image name (see "Default" on page 778).
"#FORMULA"	Text	No	Field name, variable name, or formula. If you want to be able to change the formula on the fly see "Formula" on page 777.

Converting Between Image Formats

If the optional **Enhanced Image Pack** is installed a procedure can convert an image file from one format into another (for example from PICT into JPEG or JPEG into TIFF). (The **Enhanced Image Pack** requires that Apple Quicktime 4.0 or later be installed on your computer. If Quicktime is not already installed on your system you can download it from `www.apple.com`. It is also included on the Panorama CD.) Image conversions are performed with the `convertimage` statement.

```
convertimage input,output,type,height,width
```

The input parameter specifies the original image file. If the image file is in the same folder as the currently active database then only the file name is required (for example "Cool Sunset.jpg"). If the image file is in a different folder then both the folder and file name must be included (for example "D:\Photography\Cool Sunset.jpg"). (Note: The original image file may be a GIF file, but convertimage cannot produce a GIF output file.)

The output parameter specifies the new, converted image file. If you want to put this new image file in the same folder as the current database then only the file name is required, if you want to put it in a different folder then both the folder and file name must be included. If a file with this name already exists in this location it will be erased.

The type parameter specifies the type of image that will be created. If the output file has an extension (for example .jpg, .pct, .tif) you should leave this parameter blank ("") and let Panorama automatically figure out the type. If the output file does not have an extension you must specify the type from the list below.

Image Type	PC Extensions	Notes
PICT	.pct	Apple PICT bitmap
BMP	.bmp	Windows and OS/2 bitmap
JPEG	.jpg .jpeg	JPEG compressed image
PNG	.png	Portable Network Graphics bitmap
TIFF	.tif .tiff	Tagged Image Format
PHOTOSHOP	.psb	Adobe Photoshop
FLASHPIX	.fpx	FlashPix bitmap
TARGA	.targa	

The height and width parameters are the height and width of the new image (in pixels). If either (or both) of these parameters is zero then the height and/or width of the original image will be used.

Here is an example that converts a BMP image into a TIFF image. Since the output file has an extension (.tif) the output image type does not need to be specified. The TIFF image will have the same dimensions as the original PICT image.

```
convertimage "my picture.bmp","my picture.tif","",0,0
```

This example converts an image into a 32 by 32 pixel icon. Since the files do not have any extensions this example can only work on a Macintosh, not on a Windows PC.

```
convertimage "my picture","my icon","PICT",32,32
```

Here is a similar example that can work on a Windows PC (it can work on a Macintosh also, if the file names have extensions).

```
convertimage "my picture.jpg","my icon.jpg","",32,32
```

When the output file is in JPEG format you can use the `imagequality` statement to control the compression level of the JPEG conversion. This statement has one parameter, a number from 0 (very low quality, high compression) to 100 (high quality, least compression).

```
imagequality level
```

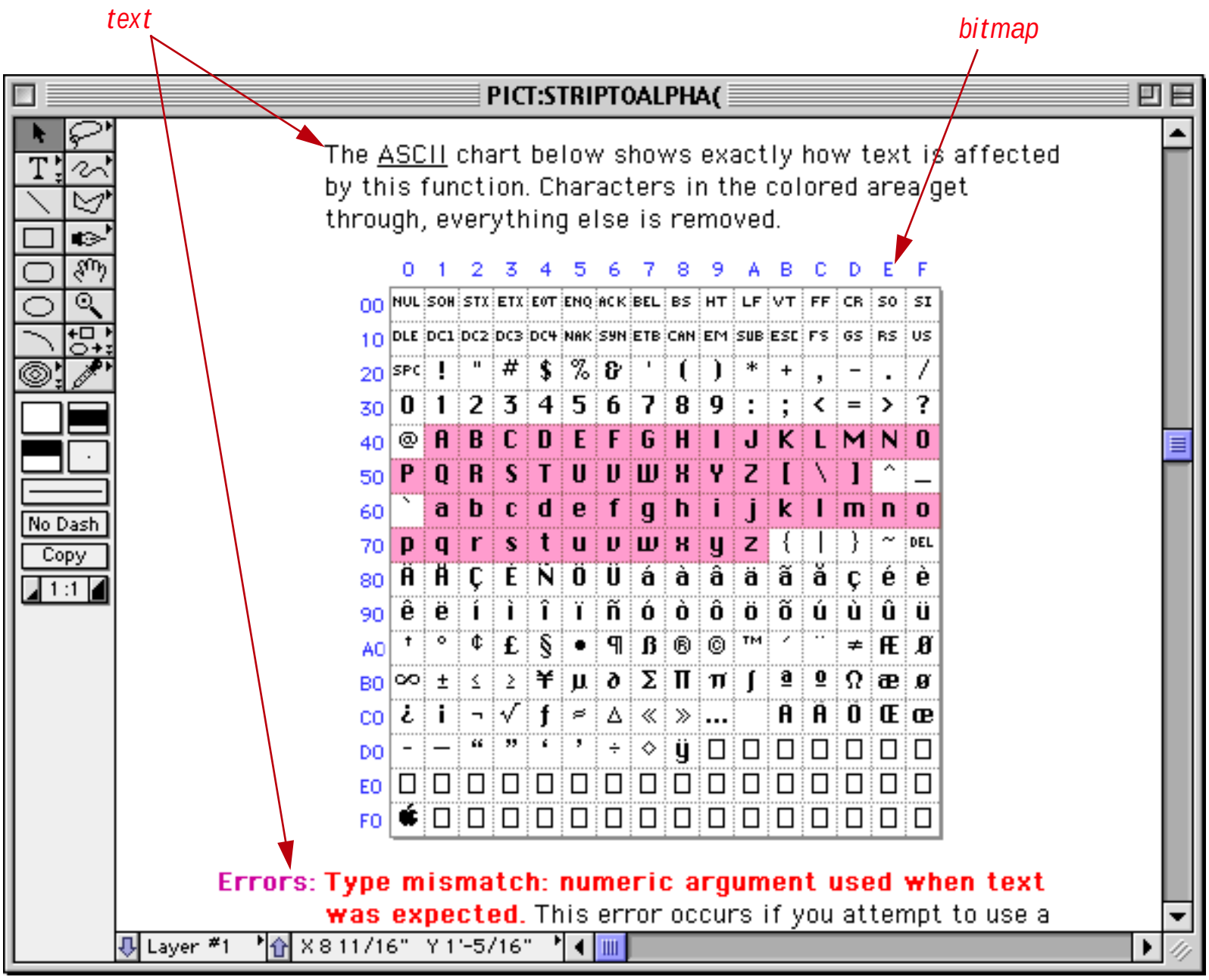
The `imagequality` statement must be used just before the `convertimage` statement. Here is an example that creates two JPEG images from a TIFF original, one low quality and one high quality. Each is placed in a different subfolder of the current database folder.

```
imagequality 80
convertimage "Sunset.tif",":Hi Quality:Sunset.jpg","",0,0
imagequality 20
convertimage "Sunset.tif",":Lo Quality:Sunset.jpg","",0,0
```

For more information on ordering the **Enhanced Image Pack** visit our website at **http://www.provue.com**.

Building Web Like HyperText Systems with Super Flash Art

Apple's PICT image format allows an image to contain text as well as bitmap information. For example the image below (shown in Deneba's Canvas 3.5) contains both text and bitmap graphics.



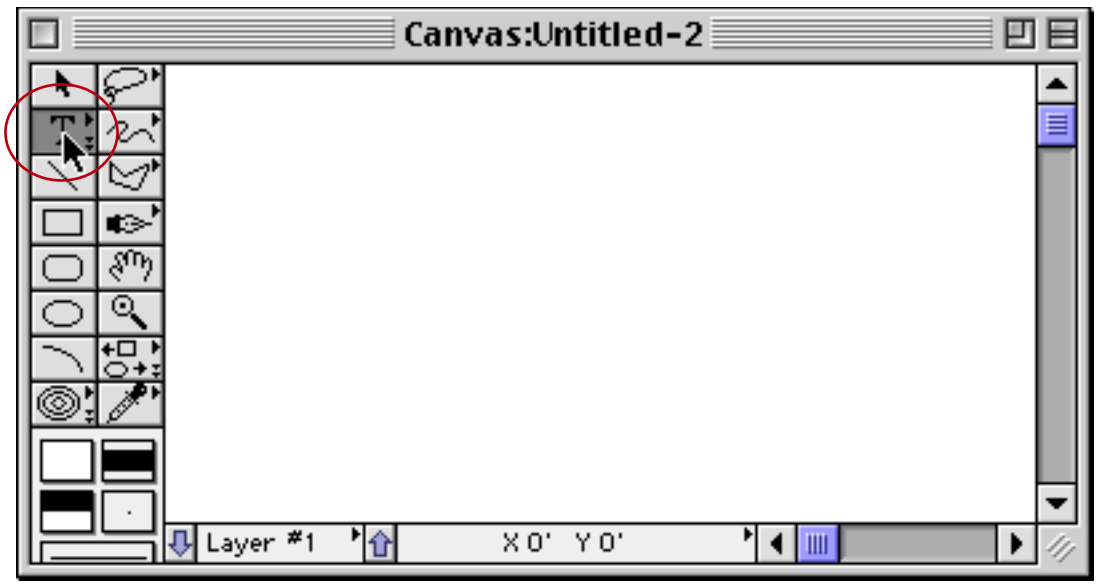
Depending on how the picture was created, it can be possible to extract the text in a picture based on certain specifications: location, style, color, font, etc. This feature gives Panorama the power to turn a collection of pictures into a linked hypertext system. The **Panorama On-Line Reference** is an example of such a system. This system is basically just a collection of PICT images. When the user clicks on a word or phrase within an image (for example the word ASCII in the image above) a simple procedure decodes what word or phrase they clicked on and switches to the new page (a new picture) based on that information.

Preparing Pictures with Extractable Text

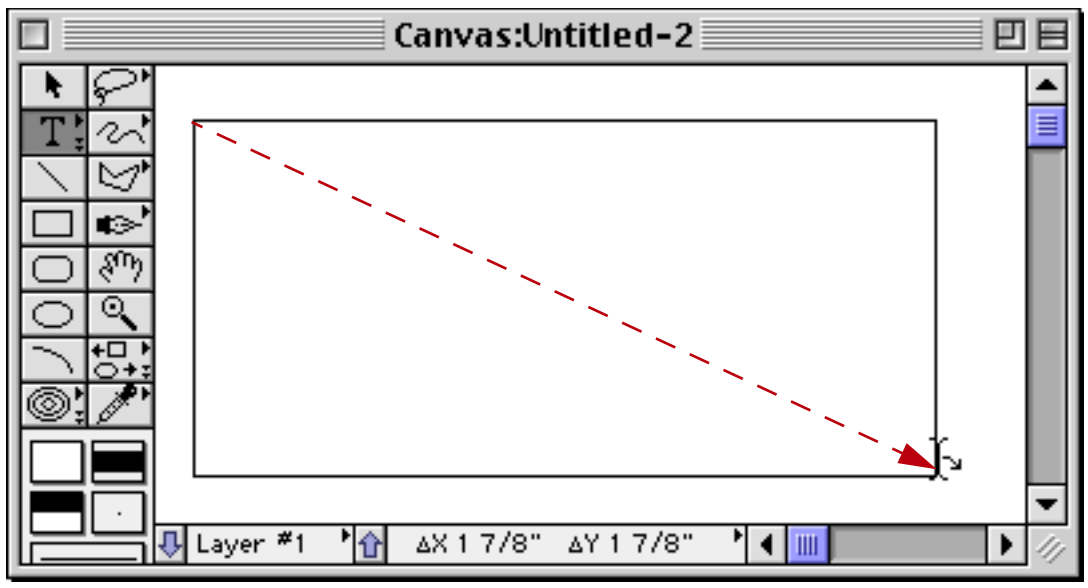
Not all text in every picture can be extracted. Text that has been converted to a pixel (bitmap) image cannot be extracted. As a general rule, if the text can be edited as text in your drawing program, the text can be extracted. For example, Photoshop does not allow text to be edited after it has been created, so text in an image created by Photoshop cannot be extracted. (Of course, Photoshop does allow the text to be manipulated with graphic tools, but that doesn't count. You must be able to insert and delete text, type in new text, etc.)

Some programs that work well for creating extractable text include Canvas and Freehand. (In Canvas, you must make sure that the text is in a text object, not a paint object.) For other programs we recommend that you try a small picture before you do a lot of work. Our favorite program for creating images with extractable text is **Deneba Canvas** (see **http://www.deneba.com**).

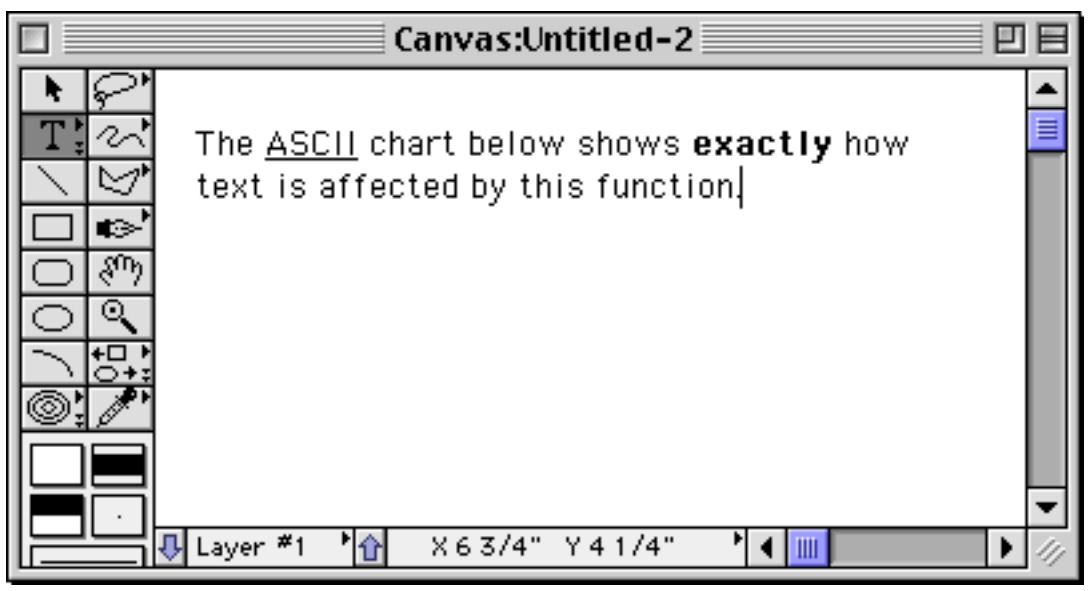
To create extractable text with Canvas you start by selecting the Text tool.



Next, you drag the mouse over the location where you want to create the text, just like creating auto-wrap text in Panorama.



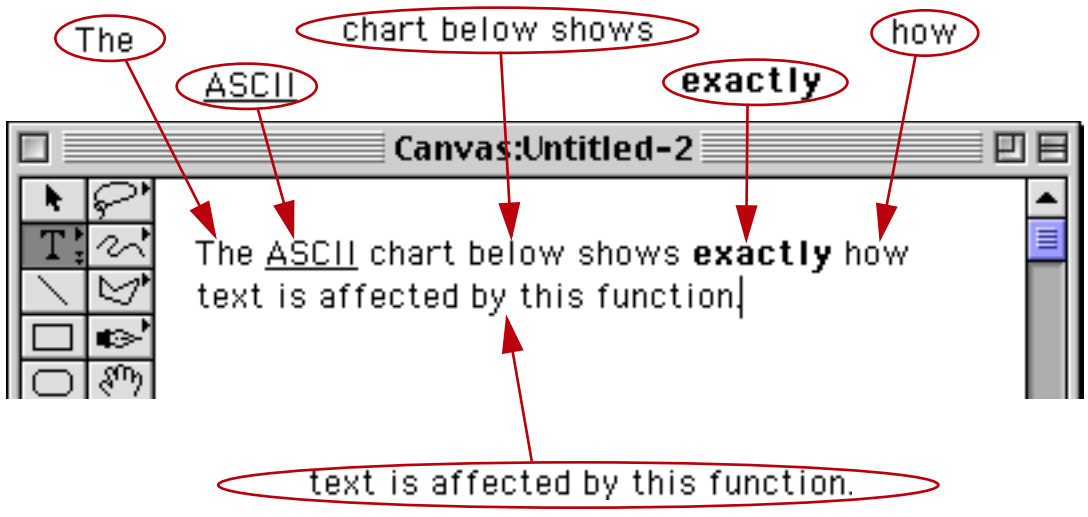
Now type in the text.



When the image is complete, be sure to save it using PICT format.

Programming a HyperText Engine

When text is saved as part of an image in PICT format the text is split up internally into chunks. A new chunk starts: 1) whenever there is any change in the text font, size, style or color, or 2) whenever a new line begins. The image created in the previous section actually consists of six separate text chunks.



The Super Flash Art "**FindText**" command takes a point within the image (x,y co-ordinates) and checks to see if a chunk of text is at that location. If there is, it extracts the text chunk and returns it to the procedure for further processing. Using this command a procedure can find out what chunk of text has been clicked on (if any) and take appropriate action.

Finding out what chunk of text has been clicked on takes three components: 1) a Super Flash Art object (see "Creating Super Flash Art Objects" on page 749), 2) a transparent "Classic" Pushbutton with the click / release option turned off (see "Transparent Push Buttons" on page 822), and 3) a procedure triggered by the "Classic" pushbutton. The transparent button should be overlaid exactly on top of the Super Flash Art object…use the **Align** command to get exact alignment (see "Aligning Objects" on page 553). If the Super Flash Art object has scroll bars, however, they should not be covered by the button. Only cover the area where the actual image is displayed.

The example below shows a procedure that will figure out what text was clicked on, and what font, size, and style the text is. The example assumes that the Super Flash Art object is named HyperFlash. (To give an object a name, first select the object, then use the **Object Name** command in the Edit menu or click on the object name in the Graphic Control Strip, see "Object Type / Object Name" on page 533.)

```
local v,h,clickPoint
local clickText, clickFont, clickSize, clickStyle
v=v(info("mouse"))-rtop(info("buttonrectangle"))
h=h(info("mouse"))-rleft(info("buttonrectangle"))
clickPoint=point(v,h)
superobject "HyperFlash","FindText",clickPoint,clickText
clickFont=objectinfo("font")
clickSize=objectinfo("textsize")
clickStyle=objectinfo("textstyle")
```

This example uses the special `superobject` "**FindText**" command. This command only works with Super Flash Art objects. The command has two additional parameters: 1) the location within the picture object (in this case clickPoint) and the field or variable the extracted text should be placed into (in this case clickText).

If the click was over a chunk of text, the "FindText" command will extract that chunk. After the chunk has been extracted the procedure can use the `objectinfo(` function to find out the font, text size, and style of the chunk (as shown in the program above). The style is a number that is calculated by adding up the following numbers for each possible style:

0	Plain
1	Bold
2	Italic
4	Underline
8	Outline
16	Shadow

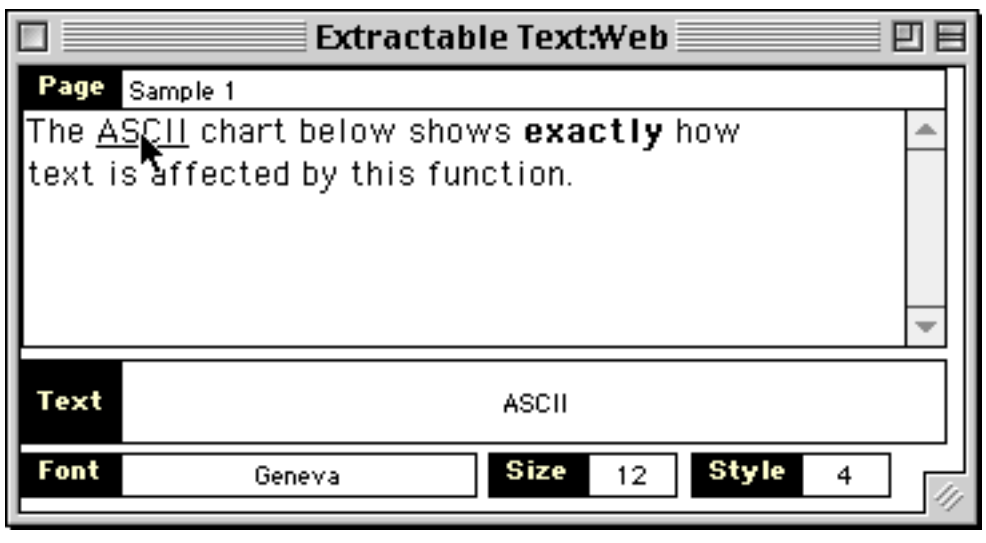
For example, if the chunk is bold-italic, the `objectinfo("textstyle")` function will return the value 3.

The statements shown below could be added to the end of the previous program to ignore all text that is not underlined.

```
if (clickStyle and 4) <> 4
   stop
endif
```

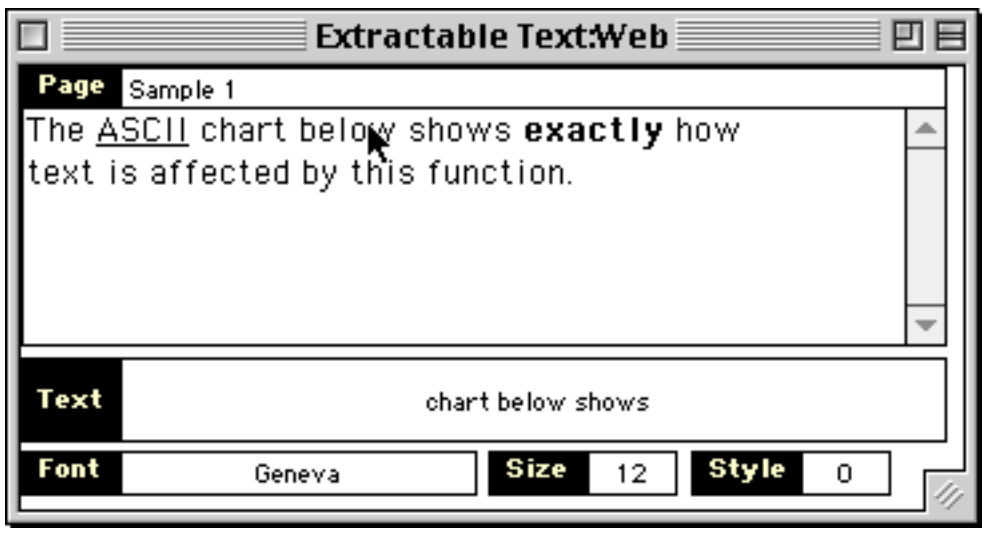
The `and` operator isolates only the underline attribute. If the statement was simply `if clickStyle<>4` the procedure would stop if the chunk was a combination style like bold-underline or italic-underline.

The sample database **Extractable Text** demonstrates Panorama's ability to extract chunks of text. When you click on the image it uses the procedure listed above to extract the text, along with it's font, size, and style. For example, if you click on the word ASCII as shown here it extracts and displays the chunk of text (ASCII) along with all of the attributes of the chunk.



As you can see the chunk of text is ASCII, the font is Geneva 12, and the style is bold (4).

You can click on any location to see what chunk of text is there.



If there is no chunk of text at the spot that was clicked the clickText variable will contain empty text ("").

Extracting All Text of a Specific Style

The `superobject` "**ExtractText**" command allows all the text that matches specified criteria to be extracted from the picture currently displayed in a Super Flash Art object. This command has several parameters as shown here:

```
SuperObject "Object Name","ExtractText",Font,Size,Style,Sep,Result
```

The first parameter, Font, specifies the font of the text you want to extract. If you don't care what the font is, leave this parameter empty ("").

The second parameter, Size, specifies the size of the text you want to extract. If you don't care, this parameter should be zero.

The third parameter, Style, specifies the style of the text you want to extract. This parameter allows you extreme flexibility in selecting what styles you want to extract.

If the Style parameter is zero, any style is ok. For example, if you want to extract all Monaco 9 point text of any style into a variable named Samples, here's how you would do it:

```
local Samples
superobject "HyperFlash","ExtractText","Monaco",9,0,";",Samples
```

If the Style parameter is 1-255, it specifies the exact style you want. Add up the numbers for each individual style. For example, for underlined text you would specify 4, for bold text, 1. The example below will extract all bold text, but not bolditalic or bold underlined.

```
local Samples
superobject "HyperFlash","ExtractText","",0,1,¶,Samples
```

If the Style parameter is 256 or greater, it specifies both the style and a mask for the style. The mask allows you to isolate individual styles. The mask uses the same style numbers as the individual styles, but multiplied by 256. (Why 256? 256 is 2 the 8th power (2^8), an even number in the computer's binary numbering system.) For example, suppose you wanted to extract all bold text, even bold text that is combined with other styles. By using a mask of 4*256 you tell the "**ExtractText**" command that you only care about the underlined style. The example below will extract all underlined, underlined-italic, underlined-bold; any text that is underlined no matter what other attributes it may have.

```
local Samples
superobject "HyperFlash","ExtractText","",0,(4*256)+4,¶,Samples
```

The fourth parameter, Sep, specifies what separator character(s) should be used between text chunks as they are extracted. Usually this is a carriage return (¶), comma, space, slash, etc. (see “Picking a Separator Character” on page 1236).

The character `"t"` is a special separator. When this separator is used, Panorama checks each piece of extracted text to see if it is on the same line as the previously extracted piece of text. If it is on the same line, Panorama will connect the pieces with a space. If the two pieces are on different lines, they will be connected with a carriage return. This allows the extracted text to be more or less reconstructed in its original form. (Note: Either `"t"` or `"T"` will trigger this special operation.)

The final parameter, Result, is the field or variable that you want the extracted text placed into.

The procedure below displays the number of underlined segments in the current picture.

```
local KeywordList
superobject "HyperFlash","ExtractText","",0,4,¶,KeywordList
message arraysize(KeywordList,¶)
```

The next example takes all the Monaco 9 pt text in the current picture and combines it. It then copies the text onto the clipboard.

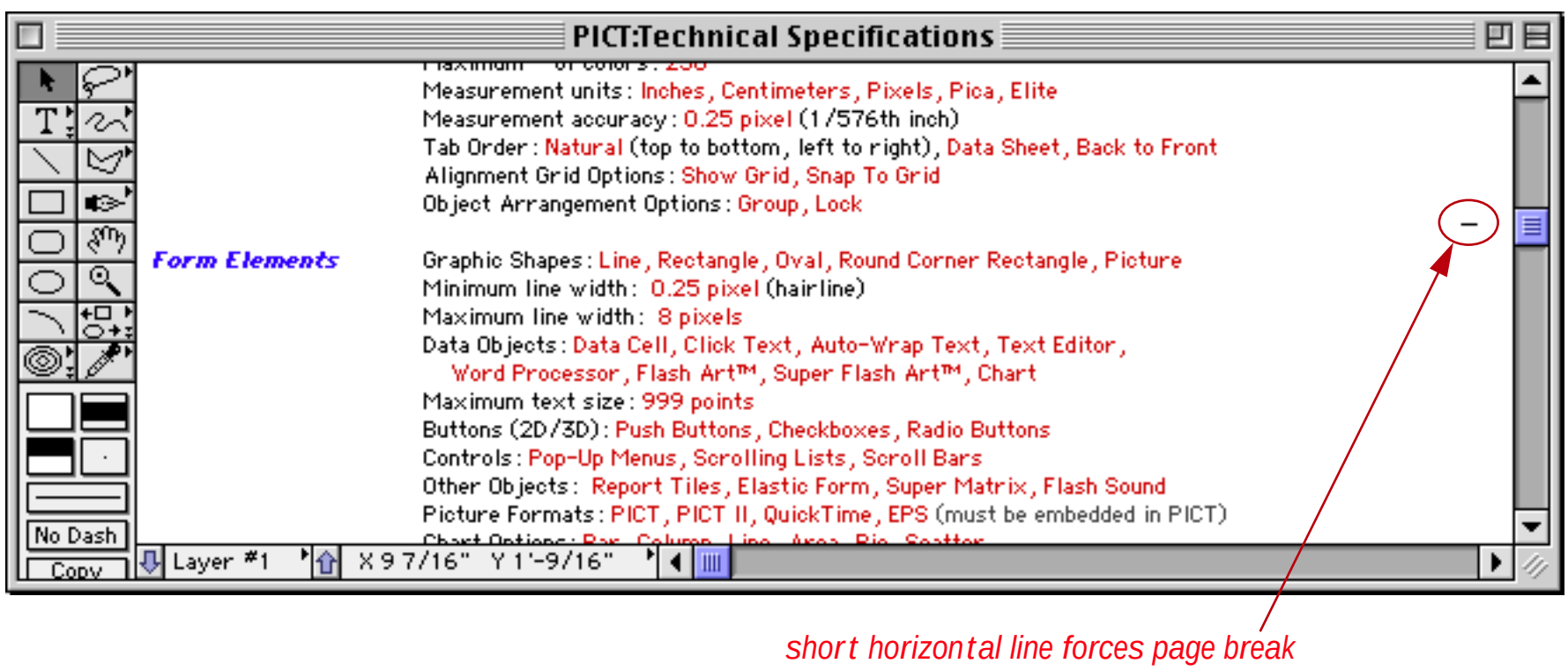
```
local SampleText
superobject "HyperFlash","ExtractText","Monaco",9,0,"t",SampleText
clipboard=SampleText
```

Creating Multi-Page Pictures

Most pictures fit on a single page. However, if you are creating a HyperText system, you may want to build pictures that are several pages long. On the screen, the user can use scroll bars to see the entire picture. But what about printing?

To allow printing of multi-page text or pictures, Panorama includes a feature called an “overflow” tile. The “overflow” tile works in conjunction with a regular data tile to print any leftover data that would not fit on the data tile. See “Printing Data that Overflows a Page” on page 1104 to learn how to set up multi-page printing.

If you use this picture overflow system, you can build special objects into your pictures that will tell Panorama where to split the picture into separate pages when printing. This will prevent the printout from splitting the page in the middle of a graphic or in the middle of a line of text. To specify a page break, you must draw a short horizontal line (a few pixels wide) near the right edge of the picture (within 16 pixels of the right edge) as shown in this example.



Push Button Internal Data

This table describes the internal data in a Push Button SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "Super Object Push Button" on page 813.

Identifier	Data Type	Changeable?	Description
"#PUSH BUTTON FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — 3D text, click/release, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#CLICK/RELEASE"	Bit	Yes	-1 if click/release is enabled, 0 if disabled.
"#3D TEXT"	Bit	Yes	-1 if 3D text is enabled, 0 if disabled.
"#HIDE BUTTON TITLE"	Bit	Yes	-1 if hide title is enabled, 0 if disabled.
"#COLOR TITLE"	Bit	Yes	-1 if color:title is enabled, 0 if disabled.
"#COLOR BORDER"	Bit	Yes	-1 if color:border is enabled, 0 if disabled.
"#COLOR FILL"	Bit	Yes	-1 if color:fill is enabled, 0 if disabled.
"#COLOR HIGHLIGHT"	Bit	Yes	-1 if color:highlight is enabled, 0 if disabled.
"#PUSH BUTTON STYLE"	Byte	Yes	Style option (see "Push Button Styles" on page 816). -1 = transparent 0 = plain rectangle 1 = plain rounded rectangle 2 = plain circle 3 = Standard Push Button (default) 4 = 3D rectangle 5 = 3D rounded rectangle 6 = 3D circle 7 = Beveled Rectangle
"#BUTTON TITLE OFFSET"	Byte	Yes	+/- vertical title offset (see "Title Positioning" on page 818).

"#BUTTON TITLE"	Text	Yes	Title of button (maximum 31 characters). The example procedure below changes a button's title from **Go** to **Stop** to **Go** to **Stop** each time the procedure runs.
			`local bTitle`
			`selectobjects`
			`  objectinfo("name")="Signal"`
			`bTitle=objectinfo("#BUTTON TITLE")`
			`if bTitle="Go"`
			`  bTitle="Stop"`
			`else`
			`  bTitle="Go"`
			`endif`
			`changeobjects "#BUTTON TITLE",bTitle`
			`selectnoobjects`

"#PROCEDURE"	Text	Yes	Name of the procedure that is triggered when this button is pressed.

Flash Art Push Button Internal Data

This table describes the internal data in a Flash Art Push Button SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "Flash Art™ Push Button SuperObjects™" on page 823.

Identifier	Data Type	Changeable?	Description
"#PUSH BUTTON FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — click/release, include pictures on disk, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#CLICK/RELEASE"	Bit	Yes	-1 if click/release is enabled, 0 if disabled.
"#INCLUDE PICTURES ON DISK"	Bit	Yes	-1 if include pictures on disk is enabled, 0 if disabled.
"#BUTTON TITLE"	Text	Yes	Title of button (maximum 31 characters).
"#PROCEDURE"	Text	Yes	Name of the procedure that is triggered when this button is pressed.
"#FLASH ART FILE"	Text	Yes	Alt File (tells Panorama to look in another database for Flash Art scrapbook, see "Alt File" on page 779).
"#FORMULA"	Text	No	Formula used to select which flash art image to display.

Data Button SuperObject Internal Data

This table describes the internal data in a Data Button SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "Super Data Button Options" on page 839.

Identifier	Data Type	Changeable?	Description
"#DATA BUTTON FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — allow multiple values, "radio" button, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#MULTIPLE VALUES"	Bit	Yes	-1 if allow multiple values is enabled, 0 if disabled.
"#RADIO BUTTON"	Bit	Yes	-1 if "radio" button is enabled, 0 if disabled.
"#BUTTON TITLE"	Text	Yes	Title of button (maximum 49 characters).
"#SEPARATOR"	Text	Yes	Value separator for multiple values (maximum 5 characters).
"#BUTTON ON VALUE"	Text	Yes	Value of button (maximum 49 characters).
"#PROCEDURE"	Text	Yes	Name of the procedure that is triggered when this button is pressed.
"#FORMULA"	Text	No	Name of field or variable that will contain data value.

Flash Art Data Button SuperObject Internal Data

This table describes the internal data in a Flash Art Data Button SuperObject that can be accessed and modified using the “back door” described in “Internal Data Types” on page 1789. To learn more about how these options work see “Flash Art Data Button SuperObjects™” on page 842.

Identifier	Data Type	Changeable?	Description
"#FLASH STICKY BUTTON FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — allow multiple values, “radio” button, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#CLICK/RELEASE"	Bit	Yes	-1 if click/release is enabled, 0 if disabled.
"#BUTTON ANIMATION"	Bit	Yes	-1 if f/x is enabled, 0 if disabled.
"#HIDE BUTTON TITLE"	Bit	Yes	-1 if hide title is enabled, 0 if disabled.
"#INCLUDE PICTURES ON DISK"	Bit	Yes	-1 if include pics on disk is enabled, 0 if disabled.
"#MULTIPLE VALUES"	Bit	Yes	-1 if allow multiple values is enabled, 0 if disabled.
"#RADIO BUTTON"	Bit	Yes	-1 if “radio” button is enabled, 0 if disabled.
"#BUTTON TITLE OFFSET"	Byte	Yes	+/- vertical title offset (see “Title Positioning” on page 818).
"#BUTTON TITLE"	Text	Yes	Title of button (maximum 49 characters).
"#SEPARATOR"	Text	Yes	Value separator for multiple values (maximum 5 characters).
"#BUTTON ON VALUE"	Text	Yes	Value of button (maximum 49 characters).
"#FLASH ART FILE"	Text	Yes	Alt File (tells Panorama to look in another database for Flash Art scrapbook, see “Alt File” on page 779).
"#FLASH ART FORMULA"	Text	No	Formula used to select which flash art image to display.
"#PROCEDURE"	Text	Yes	Name of the procedure that is triggered when this button is pressed.
"#FORMULA"	Text	No	Name of field or variable that will contain data value.

Sticky Push Button SuperObject Internal Data

This table describes the internal data in a Sticky Push Button SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "Sticky Push Button SuperObjects™" on page 845.

Identifier	Data Type	Changeable?	Description
"#STICKY PUSH BUTTON FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — allow multiple values, "radio" button, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#CLICK/RELEASE"	Bit	Yes	-1 if click/release is enabled, 0 if disabled.
"#3D TEXT"	Bit	Yes	-1 if 3D text is enabled, 0 if disabled.
"#HIDE BUTTON TITLE"	Bit	Yes	-1 if hide title is enabled, 0 if disabled.
"#MULTIPLE VALUES"	Bit	Yes	-1 if allow multiple values is enabled, 0 if disabled.
"#RADIO BUTTON"	Bit	Yes	-1 if "radio" button is enabled, 0 if disabled.
"#COLOR TITLE"	Bit	Yes	-1 if color:title is enabled, 0 if disabled.
"#COLOR BORDER"	Bit	Yes	-1 if color:border is enabled, 0 if disabled.
"#COLOR FILL"	Bit	Yes	-1 if color:fill is enabled, 0 if disabled.
"#COLOR HIGHLIGHT"	Bit	Yes	-1 if color:highlight is enabled, 0 if disabled.
"#PUSH BUTTON STYLE"	Byte	Yes	Style option (see "Push Button Styles" on page 816). 0 = rectangle 1 = rounded rectangle 2 = circle 3 = 3D rectangle 4 = 3D rounded rectangle 5 = 3D circle 6 = Beveled Rectangle
"#BUTTON TITLE OFFSET"	Byte	Yes	+/- vertical title offset (see "Title Positioning" on page 818).
"#BUTTON TITLE"	Text	Yes	Title of button (maximum 49 characters).
"#SEPARATOR"	Text	Yes	Value separator for multiple values (maximum 5 characters).
"#BUTTON ON VALUE"	Text	Yes	Value of button (maximum 49 characters).
"#PROCEDURE"	Text	Yes	Name of the procedure that is triggered when this button is pressed.
"#FORMULA"	Text	No	Name of field or variable that will contain data value.

Pop-Up Menu SuperObject Internal Data

This table describes the internal data in a Pop-Up Menu SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "Pop-Up Menu Options" on page 857.

Identifier	Data Type	Changeable?	Description
"#POP-UP MENU FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — multi-column menus, combo box, show value, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#POP-UP 2 PIXEL DROP SHADOW"	Bit	Yes	-1 if drop shadow:2 pixels is enabled, 0 if disabled.
"#POP-UP 1 PIXEL DROP SHADOW"	Bit	Yes	-1 if drop shadow:1 pixel is enabled, 0 if disabled.
"#POP-UP SHOW VALUE"	Bit	Yes	-1 if show value is enabled, 0 if disabled.
"#POP-UP CHICAGO"	Bit	Yes	-1 if Chicago 12 is enabled, 0 if disabled.
"#POP-UP MENU WRAPPING"	Bit	Yes	-1 if multi-column is enabled, 0 if disabled.
"#POP-UP TRIANGLE"	Bit	Yes	-1 if show triangle is enabled, 0 if disabled.
"#COMBO BOX"	Bit	Yes	-1 if combo box is enabled, 0 if disabled.
"#SMART POP UP COMBO BOX"	Bit	Yes	-1 if mac popup/windows combo box is enabled, 0 if disabled.
"#POP-UP FILL COLOR"	Text	Yes	Fill color for pop-up menu (see "Colors" on page 1294).
"#POP-UP LAST ITEM"	Word	No	Last menu item selected (number from 1 to maximum number of items in menu). For example if the menu contains four items (Red, Green, Blue, Orange) and the user picks Blue this value will be 3.
"#PROCEDURE"	Text	Yes	Name of the procedure that is triggered when this button is pressed.
"#FIELD"	Text	Yes	Name of field or variable that contains value of pop-up menu (maximum 31 characters).
"#FORMULA"	Text	No	Formula for calculating contents of menu.

List SuperObject™ Commands

The List SuperObject understands about a dozen commands that can be sent to it with the `superobject` statement in a procedure (see “Program Control of SuperObjects™” on page 1786). This table describes each of these commands in detail.

Command	Parameters	Description

"FillList"	Formula,Database	This command re-fills the specified list. You can use this command to update the list, or to fill it with completely new information.
		
		To update the list using the settings in the List Configuration Dialog (see “List Options” on page 873) leave off the Formula and Database. For example, if the pizza toppings list was derived from a pizza toppings database, you would want to use this procedure when the pizza toppings database had changed:
		
		`superobject "Toppings","FillList"`
		
		You can also use the "**FillList**" command to fill the list with entirely new information, completely ignoring the formula and database originally specified in the List dialog. The same list can be filled and refilled again and again with different items as conditions change. Below are three samples that could be used to fill a list from the Pizza Toppings database. The first sample lists all toppings, the next veggie only, the final meat only.
		
		`superobject "Toppings","FillList",`
		`  Topping,"Pizza Toppings"`
		
		`superobject "Toppings","FillList",`
		`  ?(Category="Veggie",Topping,""),"Pizza Toppings"`
		
		`superobject "Toppings","FillList",`
		`  ?(Category="Meat",Topping,""),"Pizza Toppings"`
		
		Of course you can also use the "**FillList**" command to directly specify the contents of the list. In this case the database should be set to "".
		
		`superobject "Toppings","FillList",`
		`  "Pepperoni"+¶+"Sausage"+¶+"Meatballs"+¶+`
		`  "Mushrooms"+¶+"Olives"+¶+"Onions"`
		`  ,""`
		
		Off course the topping list could also be created with variables. Here’s an example:
		
		`local MeatToppings,VeggieToppings,SpecialtyToppings`
		`MeatToppings="Pepperoni"+¶+"Sausage"+¶+"Meatballs"`
		`VeggieToppings="Mushrooms"+¶+"Olives"+¶+"Onions"`
		`SpecialtyToppings="Anchovies"+¶+"Garlic"`
		`superobject "Toppings","FillList",VeggieToppings,""`

"AutoScroll"		This command scrolls the list so that the first selected item is visible. For example, this procedure selects Pineapple and scrolls the list to make sure that the Pineapple item is visible. (The procedure assumes that the selected list value is stored in a field named ListCell — see “Data” on page 873).
		
		`ListCell="Pineapple"`
		`SuperObject "Toppings","AutoScroll"`

Command	Parameters	Description

"CellRectangle"	Item,Rectangle	This command allows a procedure to determine the physical location and size (i.e. rectangle) of any item in the list. This command has two parameters as shown below: the Item number (from 1 to the maximum number of items in the list) and the Rectangle. The Rectangle should be a variable that will contain the final result.
		
		Here is an example that fills in the variable dragRectangle with the dimensions of the third item in the list.
		
		`superobject "My List",`
		`  "CellRectangle",3,dragRectangle`
		
		Note: The rectangle that is returned by this command is in window relative co-ordinates (see “Rectangles” on page 1290). You can change this to screen or form relative co-ordinates using the **xytoxy(** function (see “XYTOXY(” on page 5910).

"PointToCell"	Point,Cell	This command allows a procedure to determine what list item (if any) corresponds to any point on the screen. For example, if someone drags something onto the list, this command allows the procedure to determine where in the list the item should be placed. This command has two parameters as shown below: the Point and the Cell. The Cell parameter should be a variable that will contain the final result. `superobject "object name","PointToCell",Point,Cell`
"GetList"	List	This command produces a list of all the items in the list, with each item separated from the next by a carriage return. The list is placed into the field or variable specified by List.
"GetSelected"	List	This command produces a list of all the selected items in the list, with each item separated from the next by a carriage return. The list is placed into the field or variable specified by List. (Note: This command is redundant if the list is already associated with a field or variable. Instead of using the command the procedure can simply examine (or change!) the value of the field or variable.)
"GetCount"	Number	This command returns a count of the total number of items currently in the list into the field or variable specified by Number.

"GetCell"	Item,Value	This command extracts the contents of a particular item in the list. The command treats the list as a series of numbered items, starting from 1 at the top of the list. This example will copy the first item in the list PartsList into the variable NextPart.
		
		`local NextPart`
		`superobject "PartsList","GetCell",1,NextPart`
		
		This example will copy the last item in the list PartsList into the variable NextPart.
		
		`local NextPart,ListCount`
		`superobject "PartsList","GetCount",ListCount`
		`superobject "PartsList",`
		`  "GetCell",ListCount,NextPart`

<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"FindCell"</td><td>Cell,Text</td><td>This command searches the list to find a specified value. The list item must match exactly, or the search will be unsuccessful. The search starts with the item specified by Cell. If successful, the number of the item containing the searched for value will be placed in Cell, otherwise Cell will be set to zero. The example below will locate Garlic in the list of pizza toppings and select it (tasty!).

<pre>local ListCell
ListCell=1
superobject "Toppings","FindCell",ListCell,"Garlic"
if ListCell≠0
   superobject "Toppings","SelectCell",ListCell
endif</pre>
Keep in mind that the word or phrase must match exactly. In this case only Garlic will be located; garlic or Roasted Garlic will not.

Note: This command is redundant if the list is already associated with a field or variable. Instead of using the "FindCell" command the procedure can simply set the value of the field or variable. Here is a much simpler procedure that performs the same function as the procedure above. (The procedure assumes that the selected list value is stored in a field named ListCell — see “Data” on page 873).

<pre>ListCell="Garlic"
showvariables ListCell</pre></td></tr>
<tr><td>"SelectCell"</td><td>Cell</td><td>This command selects a specified item in the list. The item is specified by Cell, which should be a number from 1 to the maximum number of items in the list.</td></tr>
<tr><td>"UnSelectCell"</td><td>Cell</td><td>This command unselects a specified item in the list. The item is specified by Cell, which should be a number from 1 to the maximum number of items in the list. The example below makes sure that there are no anchovies on the pizza!

<pre>local ListCell
ListCell=1
SuperObject
"Toppings","FindCell",ListCell,"Anchovies"
if ListCell≠0
   superobject "Toppings","UnSelectCell",ListCell
endif</pre>
Note: This command is usually redundant if the list is already associated with a field or variable. Instead of using the "UnSelectCell" command the procedure can simply set the value of the field or variable. Here is a much simpler procedure that performs the same function as the procedure above. (The procedure assumes that the selected list value is stored in a field named ListCell that is a carriage return separated array — see “Data” on page 873).

<pre>ListCell=replace(ListCell,"Garlic","")
ListCell=arraystrip(ListCell,¶)
showvariables ListCell</pre></td></tr>
</table>

<table>
<tr><th>Command</th><th>Parameters</th><th>Description</th></tr>
<tr><td>"SetCell"</td><td>Cell,Value</td><td>This command changes the contents of a specified item in the list. The item is specified by Cell, and should be from 1 to the maximum number of items in the list. The example below changes the Cheese item to Extra Cheese.

<pre>local ListCell
ListCell=1
superobject "Toppings","FindCell",ListCell,"Cheese"
if ListCell<>0
   superobject "Toppings","SetCell",ListCell,
      "Extra Cheese"
endif</pre></td></tr>
<tr><td>"AddCell"</td><td>Value</td><td>This command adds a new item to the end of the list. This example adds the item Sun Dried Tomatoes to the end of the list of pizza toppings.

<pre>superobject "Toppings","AddCell",
   "Sun Dried Tomatoes"</pre></td></tr>
<tr><td>"InsertCell"</td><td>Cell,Value</td><td>This command inserts a new item into the middle of the list. The Cell parameter specifies where the new item should be inserted. This parameter must be a number from 1 up to the number of items in the list. The new item will go above the item specified. For example, you could insert the item Extra Cheese at the very top of the pizza topping list:

<pre>superobject "Toppings",
   "InsertCell",1,"Extra Cheese"</pre>
This more complex example inserts Grilled Onions after Onions.

<pre>local ListCell
ListCell=1
superobject "Toppings","FindCell",ListCell,"Onions"
if ListCell≠0
      ListCell=ListCell+1
      superobject "Toppings","InsertCell",
            ListCell,"Grilled Onions"
endif</pre>
Notice that the example adds one to ListCell before inserting the new item. This is so the new item (Grilled Onions) will be inserted after the original item (Onions) instead of before it.</td></tr>
<tr><td>"DeleteCell"</td><td>Start,End</td><td>This command deletes one or more items from the list. If you just want to delete a single cell, then only one number is needed. This example deletes the first item in the list.

<pre>superobject "Toppings","DeleteCell",1</pre>
To delete a bunch of cells at once, specify two numbers — the first and last cell to delete. This example deletes the first 5 items in the list.

<pre>superobject "Toppings","DeleteCell",1,5</pre>
This example will delete the entire list in a big hurry!

<pre>superobject "Toppings","DeleteCell",1,10000</pre></td></tr>
</table>

Command	Parameters	Description

"FindSelected"	Cell	This command finds the next selected cell, starting with Cell. The result is placed in Cell, or zero if there are no selected cells below the starting spot. The example below deletes all the selected items from the list.
		
		`Local Spot`
		`Spot=1`
		`loop`
		`  SuperObject "Toppings","FindSelected",Spot`
		`  if Spot=0`
		`  stop`
		`  endif`
		`  SuperObject "Toppings",DeleteCell,Spot`
		`next`

Using Drag and Drop to Change the Order of Items in a List

This example shows how to set up a procedure that allows the user to drag items up or down in a list to change the order of a list. This example assumes that there is a list of names in a field called Names, with each name separated from the next by a carriage return. The current form must contain a List SuperObject named Names List (see “Object Type / Object Name” on page 533). The Names List object displays the Names field, it is also linked to a global variable named theName (see “Data” on page 873).

```
global theName
local cell,cellbox,listbox,mouse,newcell,newNames
cell=1
superobject "Names List","findselected",cell
superobject "Names List","cellrectangle",cell,cellbox
cellbox=xytoxy(cellbox,"w","s")
object "Names List"
listbox=xytoxy( objectinfo("rectangle"),"f","s")
draggraybox cellbox,listbox,listbox,0
if cellbox="" stop endif /* user dragged out of the list */
mouse=xytoxy( info("mouse"),"s","w")
/* where is the new location for this item? */
superobject "Names List","pointtocell",mouse,newcell
if cell=newcell stop endif /* item did not move */
if newcell>cell
   newcell=newcell-1 /* adjust for deleting item in old spot */
endif
/* delete item from old spot */
newNames=arraydelete(Names,cell,1,¶)
if newcell>0
   /* insert item in new spot */
   newNames=arrayinsert(newNames,newcell,1,¶)
   newNames=arraychange(newNames,theName,newcell,¶)
else
   /* add item to end of list */
   newNames=newNames+¶+theName
endif
Names=newNames /* update original field */
superobject "Names List","filllist" /* re-display list */
showvariables theName /* and select correct item */
```

The illustration below shows this procedure in action. You can click on any item in the list and drag it into a new position on the list.



List SuperObject Internal Data

This table describes the internal data in a List SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "List Options" on page 873.

Identifier	Data Type	Changeable?	Description
"#LIST FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — sort up, no duplicates, click/release, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#LIST SORT"	Bit	Yes	-1 if sort up is enabled, 0 if disabled.
"#LIST NO DUPLICATES"	Bit	Yes	-1 if no duplicates is enabled, 0 if disabled.
"#GROW BOX"	Bit	Yes	-1 if grow box is enabled, 0 if disabled.
"#CLICK/RELEASE"	Bit	Yes	-1 if click/release is enabled, 0 if disabled.
"#THIN SCROLL"	Bit	Yes	-1 if thin scroll bars are enabled, 0 if disabled.
"#LIST CLICK FLAGS"	Byte	Yes	This value controls the click action configuration (see "Click Action" on page 882). 0 = Normal 128 = One Cell Only 32 = Contiguous Cells Only 118 = Extend w/o Shift
"#LIST DATABASE"	Text	Yes	Name of database to scan ("" if formula builds list directly).
"#SEPARATOR"	Text	Yes	Value separator for multiple values (maximum 5 characters).
"#PROCEDURE"	Text	Yes	Name of the procedure that is triggered when this list is pressed.
"#FIELD"	Text	Yes	Name of field or variable that contains value of pop-up menu (maximum 31 characters).
"#FORMULA"	Text	No	Formula for calculating contents of list.

Auto Grow SuperObject™ Commands (Elastic Forms)

The Auto Grow SuperObject (see “Elastic Forms” on page 912) understands a small set of commands that can be sent to it with the `superobject` statement in a procedure (see “Program Control of SuperObjects™” on page 1786). This table describes each of these commands in detail.

Command	Parameters	Description
"GetMinSize"	Height,Width	This command gets the minimum window size (see “Building an Elastic Form” on page 915).
"SetMinSize"	Height,Width	This command sets the minimum window size (see “Building an Elastic Form” on page 915).
"GetMaxSize"	Height,Width	This command gets the maximum window size (see “Maximum Window Size” on page 919).
"SetMaxSize"	Height,Width	This command sets the maximum window size (see “Maximum Window Size” on page 919).

Auto Grow SuperObject Internal Data

This table describes the internal data in an Auto Grow SuperObject that can be accessed and modified using the “back door” described in “Internal Data Types” on page 1789. To learn more about how these options work see “Building an Elastic Form” on page 915.

Identifier	Data Type	Changeable?	Description
"#AUTO GROW FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — sort up, no duplicates, click/release, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#AUTO GROW HORIZONTAL"	Bit	Yes	-1 if slave (horizontal) is enabled, 0 if disabled.
"#AUTO GROW VERTICAL"	Bit	Yes	-1 if slave (vertical) is enabled, 0 if disabled.
"#NO AUTO GROW"	Bit	Yes	-1 if don’t adjust form is enabled, 0 if disabled.
"#AUTO GROW ICON"	Bit	Yes	-1 if draw grow icon is enabled, 0 if disabled.
"#AUTOGROW PROCEDURE"	Bit	Yes	-1 if .Autogrow proc is enabled, 0 if disabled.

Super Matrix SuperObject™ Commands

The Super Matrix SuperObject (see “Super Matrix Objects” on page 929) understands a small set of commands that can be sent to it with the `superobject` statement in a procedure (see “Program Control of SuperObjects™” on page 1786). This table describes each of these commands in detail.

Command	Parameters	Description

"ReDraw"	Area,Start,End	This command redraws some or all of the cells in a super matrix. The first parameter, Area, defines the area that will be redrawn. Legal options for this parameter are: "all", "column", "row", and "cell".
		
		The Start and End parameters define the start and end of the area to be redrawn. For example, if the Area parameter was "column" and the last two parameters were 3 and 5, then columns 3 thru 5 would be redrawn. (Note: The start and end values are ignored if the "all" area is chosen.)
		
		The following examples illustrate different ways a matrix might be updated. This calendar example redisplays the entire month.
		
		`superobject "Month","redraw","all",0,0`
		
		This example redisplays only weekdays.
		
		`superobject "Month","redraw","column",2,6`
		
		This example works with a matrix of photographs. The procedure redisplays photo 7 only.
		
		`superobject "Photographs","redraw","cell",7,7`
		
		This example redisplays all photos after photo 12. This procedure would be used if someone inserts or deletes a photograph at position 12.
		
		`superobject "Thumbnails","redraw","cell",12,9999`

"CellRectangle"	Cell,Rectangle	This command returns the dimensions of an individual cell in the matrix. The dimensions are in window co-ordinates. The Cell parameter should be a number from 1 to the maximum number of cells in the matrix. The Rectangle parameter should be a field or variable where the rectangle will be stored.
		
		This example uses the "**CellRectangle**" command to open a new window over the current matrix cell (the matrix cell that was clicked on).
		
		`local subWindowRectangle`
		`superobject "Calendar","CellRectangle",`
		`  info("matrixcell"),subWindowRectangle`
		`setwindowrectangle`
		`  xytoxy(subWindowRectangle,"w","g"),""`
		`openform "Day"`

"CellToXY"	Cell,Row,Col	This command converts a matrix cell number into a row and column. The example below displays what row and column were clicked on.
		
		`local mRow,mCol`
		`superobject "Thumbnail",`
		`  "CellToXY",info("matrixcell"),mRow,mCol`
		`message "You clicked on row "+str(mRow)+`
		`  " and column "+str(mCol)`

"MatrixShape"	Row,Column	This command changes the shape of the matrix. The parameters specify the visible rows and columns. For example, if you want to display 3 by 5 matrix the parameters should be 3,5. If negative, these are fixed heights/widths (in pixels). For example, to display 64 by 64 icons use -64,-64.

Command	Parameters	Description
"MatrixGetShape"	Row,Column	This command returns the current shape of the matrix. Positive numbers indicate the number of rows or columns, negative numbers indicate a fixed height or width in pixels. For example -20,1 indicates that there is one column, with a variable number of rows that are each 20 pixels high.
"MatrixScroll"	Row,Column	This command moves the upper left hand corner of the matrix to the specified coordinates, adjusting the scroll bars as necessary. The co-ordinates are number from 1..N. If there is only one scroll bar then the other parameter is ignored.
"MatrixGetScroll"	Row,Column	Get the coordinates of the upper left hand cell, starting from 1. As you scroll down and to the right, the numbers will increase.
"PointToCell"	Point,Cell	Given a point in local co-ordinates, it returns the corresponding cell # (or zero if not in a cell).
"MatrixBounds"	Cells/Rows,Columns	This command sets the bounds of the scrollable area. If there is only one scroll bar, then only the first parameter is used and it is treated as the number of CELLS in the matrix. For example, if your matrix is displaying 97 items then you would set the number of cells to 97. This means that when you drag the scroll thumb to the bottom you'll see the last row of items, up to number 97. If there are two scroll bars then the parameters are rows and columns. Until this statement is used the default is 100 cells or rows/columns. (Note: If you have specified the matrix data formula and separator then the matrix bounds will be calculated automatically, and you should not use this command.)

"MatrixSize"	Cells	This command calculates the current number of cells in the matrix. Here is a procedure that displays all the vital statistics for a matrix.
		`local mCells,mRows,mCols`
		`superobject "Images","MatrixSize",mCells`
		`superobject "Images","CellToXY",mCells,mRows,mCols`
		`message "This matrix contains "+`
		`str(mCells)+" cells ("+`
		`str(mRows)+" rows by "+`
		`str(mCols)+ "columns).`

"Scroll"	Rows,Cols	This command slides the display of the matrix up or down and/or right or left. This command simply slides the matrix display — it's up to you to adjust the underlying data structure. If the Rows parameter is positive the matrix display will slide up by the specified number of rows, if negative it will slide down. If the Cols parameter is positive the matrix display will slide right by the specified number of columns, if negative it will slide left. For either parameter a value of 0 may be used to maintain the same position on that axis. Note: This command is obsolete, it was designed before matrix objects had scroll bars.

Super Matrix SuperObject Internal Data

This table describes the internal data in a Super Matrix SuperObject that can be accessed and modified using the "back door" described in "Internal Data Types" on page 1789. To learn more about how these options work see "Designing a Matrix Template" on page 939.

Identifier	Data Type	Changeable?	Description
"#SUPER MATRIX FLAGS"	Long Word	Yes	This internal data item contains all of the on/off options for the object — matrix order, fixed width, fixed height, click/release, etc. You can also access each of these options separately (see following entries). Being able to access all of these values at once makes it easy to save all the flags, modify selected flags, and then restore all of the original settings.
"#VERTICAL SCROLL BAR"	Bit	Yes	-1 if vertical scroll bar is enabled, 0 if disabled.

Identifier	Data Type	Changeable?	Description
"#HORIZONTAL SCROLL BAR"	Bit	Yes	-1 if horizontal scroll bar is enabled, 0 if disabled.
"#THIN SCROLL BARS"	Bit	Yes	-1 if thin scroll bars are enabled, 0 if disabled.
"#GROW BOX"	Bit	Yes	-1 if grow box is enabled, 0 if disabled.
"#SUPER MATRIX SHOW FRAME"	Bit	Yes	-1 if display:frame object is enabled, 0 if disabled.
"#SUPER MATRIX ORDER"	Bit	Yes	-1 if horizontal is enabled, 0 if vertical.
"#SUPER MATRIX FIXED WIDTH"	Bit	Yes	-1 if fixed width (pixels) is enabled, 0 if fixed # of columns.
"#SUPER MATRIX FIXED HEIGHT"	Bit	Yes	-1 if fixed height (pixels) is enabled, 0 if fixed # of rows.
"#CLICK/RELEASE"	Bit	Yes	-1 if click/release is enabled, 0 if disabled.
"#SUPER MATRIX GROW METHOD"	Bit	Yes	-1 if slide is enabled, 0 if proportional.
"#SUPER MATRIX BORDERS"	Bit	Yes	-1 if cell borders is enabled, 0 if disabled.
"#OVERFLOW"	Bit	Yes	-1 if overflow is enabled, 0 if disabled.
"#SUPER MATRIX COLUMNS"	Long Word	Yes	Number of columns if fixed # of columns is enabled, or width of each column (in pixels) if fixed width is enabled.
"#SUPER MATRIX ROWS"	Long Word	Yes	Number of rows if fixed # of rows is enabled, or width of each column (in pixels) if fixed height is enabled.
"#SUPER MATRIX GROW BOUNDARY"	Long Word	Yes	Point specifying slide boundaries (used if slide is enabled). Use **v(** and **h(** functions to extract individual dimensions (see “Points” on page 1288).
"#SUPER MATRIX FRAME"	Text	Yes	Name of matrix frame object (maximum 31 characters).
"#PROCEDURE"	Text	Yes	Name of procedure triggered when matrix is clicked on, if any.
"#SEPARATOR"	Byte	Yes	Array separator character, if any.
"#FORMULA"	Text	Yes	Formula for data array, if any.

Scroll Bar SuperObject™ Commands

The Scroll Bar SuperObject (see “Scroll Bars” on page 968) understands a small set of commands that can be sent to it with the `superobject` statement in a procedure (see “Program Control of SuperObjects™” on page 1786). This table describes each of these commands in detail.

Command	Parameters	Description
"GetScrollMin"	Value	This command gets the minimum scroll bar value and places into the field or variable specified by Value.
"SetScrollMin"	Value	This command sets the minimum scroll bar value to any numeric value (must be integer) between 1 and 65535. (This value is normally set by the Min value in the Scroll Bar dialog.)
"GetScrollMax"	Value	This command gets the maximum scroll bar value and places into the field or variable specified by Value.

Command	Parameters	Description

"SetScrollMax"	Value	This command sets the maximum scroll bar value to any numeric value (must be integer) between 1 and 65535. (This value is normally set by the Max value in the Scroll Bar dialog.) Here is an example that sets the maximum value of the scroll bar named Slider to the number of elements in the array People:
		
		`superobject "Slider",`
		`"SetScrollMax",arraysize(People,¶)`

"GetScrollPage"	Value	This command gets the scroll bar page amount and places into the field or variable specified by Value. This value is the amount the scroll bar value will increase or decrease if the user clicks on the gray area above or below the thumb of the scroll bar.
"SetScrollPage"	Value	This command sets the scroll bar page amount to any numeric value (must be integer) between 1 and 65535. This value is the amount the scroll bar value will increase or decrease if the user clicks on the gray area above or below the thumb of the scroll bar. (This value is normally set by the Page Up/Down value in the Scroll Bar dialog.)
"GetScrollValue"	Value	This command gets the current position of the scroll bar. This command is redundant because you can always get the position by examining the field or variable linked to the scroll bar.
"DisableScroll"		This command disables the scroll bar. The scroll bar is still visible, but it turns white and the thumb disappears.
"EnableScroll"		This command enables the scroll bar (see **DisableScroll** above).
"GetScrollEnable"	Value	This command checks to see if a scroll bar is enabled or disabled, and sets the field or variable specified by Value with a true or false result accordingly.

Speech Synthesis

Speech synthesis is an exciting new feature introduced in Panorama V. (Note: Speech synthesis is available only on Macintosh systems. It is not available on Windows systems.) There are several new statements and functions that support speech synthesis. In it's simplest form you can cause Panorama to speak a word or phrase simply by using the speak statement followed by the text to speak.

The Speak Statement

This statement speaks a word or phrase using the current voice. If the voice is already talking, it waits until the current text has been spoken before speaking the new text. The program continues running immediately - in other words, the next statement will be executed immediately, without waiting for the text to be spoken. See the `info("speaking")` function if you need to wait until the text has been spoken to proceed.

The `speak` statement has one parameter, the text to be spoken. Here is an example that speaks a short sentence.

```
speak "Welcome to the order entry system!"
```

Embedded Speech Commands

The system does it's best to parse the text you supply and to speak it properly. For example, it will automatically inflect the end of a sentence depending on whether the sentence ends with a period, question mark, or exclamation mark. You can also tweak the way the text is pronounced with embedded speech commands. These commands are text tags that are embedded in the text to change the rate, pitch and emphasis of the text being spoken. To learn about how to use these tags see *Embedded Speech Commands* on Apple's developer website.

```
http://developer.apple.com/techpubs/mac/Sound/Sound-200.html#HEADING200-0
```

The StopSpeaking Statement

This statement tells Panorama to shut up now! Any text that is being spoken will stop immediately in mid-sentence. There are no parameters.

The info("speaking") Function

This function returns true if talking is occurring right now. You can use this function to wait for speech to finish before continuing to the next step in a procedure. For example, this procedure times now long it takes to speak a phrase.

```
local start
start=now()
speak "How long does it take to say this?"
loop
   nop
while info("speaking")
message now()-start
```

Buffered Speech

The buffered speech statements allow you to build up the text you want to speak item by item. For example, you might want to include data from different records in the text to be spoken. The buffered speech statements allow you to build up the text to be spoken over several statements, then smoothly speak the combined text when you are done assembling the items. The two basic buffered speech statements are speakwords and speaknow. Here's a silly example of how these could be used to count from one to five.

```
speakwords "One"
speakwords "Two"
speakwords "Three"
speakwords "Four"
speakwords "Five"
speaknow
```

Before moving to a less silly example, the table below describes each of the statements for buffering speech. As the table shows, each of these statements can speak text, numbers, dates, or some combination.

Data Type	Statement	Description
TEXT	speakwords	The text is spoken using normal English.
	speakdigits	The data is spoken in English, but if the data contains any numbers they will be spoken as individual digits. For example, 4892 will be spoken as four eight nine two.
	speakdigitpairs	The data is spoken in English, but if there are any numbers they will be spoken as digit pairs. For example, 4892 will be spoken as forty-eight ninety-two.
	speakletters	The data is spoken letter by letter, including upper and lower case. For example Jim will be spoken as upper case J, lower case I, lower case M.
	speakcharacters	The data is spoken letter by letter. For example April will be spoken as A P R I L.
	speakcharactersslowly	The data is spoken letter by letter, with a delay between each letter. For example April will be spoken as A; P; R; I; L.
	speakaddress	The data is spoken in English, but any official USPS abbreviations are expanded as the text is spoken. For example S or S. is expanded to South, while Bl. is expanded to Boulevard. Any numbers in the text are spoken as digit pairs.
	speakstate	If the text is a two letter abbreviation of a U.S. state it will be spoken as the full name. For example CA will be spoken as California, while WV will be spoken as West Virginia.
	speakphonenumber	The first ten digits in the text are spoken as a phone number, along with any extension after that.
NUMBER	speakwords	The number is spelled out. For example, 4892 will be spoken as four-thousand eight-hundred and ninety-two.
	speakdigits	The number is spoken as individual digits. For example, 4892 will be spoken as four eight nine two.
	speakdigitpairs	The number is spoken as digit pairs. For example, 4892 will be spoken as forty-eight ninety-two.
	speakdollars	The integer portion of the number is spelled out, followed by the word Dollars. For example 378.93 will be spoken as Three-hundred seventy-eight dollars.
	speakdollarsandcents	The integer portion of the number is spelled out, followed by the word Dollars, followed by the two digits after the decimal point, followed by the word Cents. For example 378.93 will be spoken as Three-hundred seventy-eight dollars and ninety-three cents.
DATE	speakdate	The date is spelled out. For example 4/18/04 is spoken as April eighteenth, 2004.

Here is an example that reads a check from a checkbook database:

```
speakwords "Check number"
speakdigitpairs Check
speakwords "to"
speakwords «Pay To»
speakwords "for”
speakdollarsandcents Debit
speaknow
```

This example reads an address from a contact database.

```
speakwords First+" "+Last
speakaddress Address
speakwords City+","
speakstate State
speakdigits Zip
speaknow
```

Speaking Using the Speech Wizard

The **Speech Wizard** (see “Speech Wizard” on page 62) allows you to define templates for speaking the data in each database. To speak the data in the current record using the currently active template use the `speakthisrecord` command (no parameters are required.)

To speak the data in the current record using a specific template use the `speakscript` statement.

```
speakscript database,template
```

If the database is "" the current database will be assumed, otherwise you can specify any open database.

The template you specify must have been previously defined in the speech wizard. You can use the `speechscripts` command to find out what templates are available.

```
speechscripts database,templatelist
```

If the database is "" the current database will be assumed, otherwise you can specify any open database.

The templatelist must be the name of a field or variable. When the statement is complete, this field or variable will contain the names of the templates that have been created in the Speech Wizard, one template per line.

The example below will speak the current record over and over again using each template that has been set up in the Speech Wizard.

```
local tlist,tp,n
speechscripts "",tlist
n=1
loop
   tp=array(tlist,n,¶)
   stoploopif tp=""
   speakscript "",tp
   n=n+1
while forever
```

Printing

Even in this e-commerce age many jobs still require paper output. Printing can be done manually (see "Printing Basics" on page 1037) or via a procedure.

Selecting a View for Printing

In Panorama printing is always done through a specific view. You can always print the data sheet, but usually when printing with a procedure you'll be using a form set up with a custom report (see "Custom Reports" on page 1049). Before printing begins the procedure must select the appropriate form, either in a new window with the `openform` statement (see "Opening a Window" on page 1572) or within the current window with the `goform` statement (see "Changing a Window's View" on page 1578). An alternate way to select a view for printing is to use the `printusingform` statement. This statement allows a procedure to print using a form without actually opening the form. See "Printing Using an Alternate Form" on page 1843 to learn how to use this statement.

Selecting a Printer

A procedure always prints to the currently selected printer. Unfortunately, there is no way to change which printer is currently selected, this must be done manually using your system software.

Adjusting Page Setup

The **Page Setup** dialog allows you to configure various printing options (see "The Page Setup Dialog" on page 1043). A procedure cannot control these options directly, but it can open the **Page Setup** dialog automatically using the `pagesetup` statement (see "PAGESETUP" on page 5588). This statement does not have any parameters, it is simply used by itself. Here is a simple procedure that opens a form and allows the page setup to be adjusted, then closes the form.

```
openform "My Report"
pagesetup
closewindow
```

Panorama keeps a separate **Page Setup** configuration for each form. This allows different forms to have different configurations (for example portrait vs. landscape orientation).

Preparing Data For Printing

Most procedures that print the database also prepare the database in some way. Typically, a procedure may sort, select a subset of the database and/or prepare summaries. See "Sorting" on page 1676, "Locating Information" on page 1677 and "Summaries and Outlines" on page 1691 to learn how to perform these tasks with a procedure.

Printing the Database

A procedure can print all selected records in the current database using the `print` statement (see "PRINT" on page 5613). This statement may be used one of two ways. The first method is to follow the statement with the parameter `dialog`, like this (there must be a space between `print` and `dialog`, as shown below).

```
print dialog
```

When used this way the procedure will pause and display the standard **Print** dialog, the same dialog that normally appears when you choose the **Print** command. This allows you to choose the print options for this print run (number of copies, paper source, etc.) The exact options depend on the printer you have selected. When the **Print** button is pressed the procedure will go ahead and print all the selected records in the current database.

The second method is to follow the `print` statement with the parameter `""`, as shown here.

```
print ""
```

When the print statement is used this way it will not display the **Print** dialog. Instead, it simply prints the database using the same options that were used the last time this form was printed. (Note: Depending on the printer driver software your printer uses, some options may not be saved from print to print. These options will use default settings. For absolute control over all print options we recommend that you use the dialog option.)

Here is an example of a complete procedure to print a report. The procedure opens the form that is designed for printing this report and then selects the appropriate data. After printing it selects all of the data again and then closes the form.

```
openform "90 Day Report"
select Date>today()-90
print dialog
selectall
closewindow
```

Printing a Single Record

To print just the current record use the `printonerecord` statement (see “PRINTONERECORD” on page 5616). For example this statement could be used to print a single letter or a single invoice. Like the `print` statement the `printonerecord` statement may be used with a parameter of either `dialog` or `""`. Here is an example that prints the current invoice.

```
openform "Paper Invoice"
printonerecord dialog
closewindow
```

Print Preview

The `printpreview` statement opens a special preview window. This window displays a preview of the printed results for the current view. This is the same as choosing **Preview** from the File menu (see “Print Preview” on page 1046). The user can flip forward to see additional pages of the previewed report. When the user closes the preview window, the procedure continues with the statement after the `printpreview` statement. Usually this should be either the end of the procedure or the `stop` statement (see “Stopping the Program” on page 1397).

The most common reason to use the `printpreview` statement in a procedure is to simulate the **Print Preview** command in your own custom File menu. Here are the statements to use in your **.CustomMenu** procedure (see “The .CustomMenu Procedure” on page 1466). (You could also trigger print preview with a button.)

```
if info("trigger") beginswith "Menu.File.Print Preview"
   printpreview
   stop
endif
```

Printing Using an Alternate Form

The `printusingform` statement allows the current database to be printed using a different form than the one currently being displayed (see “PRINTUSINGFORM” on page 5618). It is designed to be used in combination with the `print`, `printonerecord`, or `printpreview` statements (see previous sections). This statement has two parameters: file and form.

```
printusingform file,form
```

File is the name of the database file that contains the form to be printed. The database file must be open. Usually the form will be in the current database, and in that case you can simply use an empty string (`""`) for the file name. Form is the name of the form to be printed.

The `print` statement normally prints whatever window is currently active. If you want to print a different window, you must first open that window and then print (see “Selecting a View for Printing” on page 1841). The `printusingform` statement is another way to print an alternate form.

Warning: The `printusingform` statement may only be used when a form window is currently on top. It will not work when a data sheet window is the current window.

The procedure below will print My Report, even if another form is currently visible.

```
printusingform "","My Report"
print dialog
```

The procedure below will print Standard Report #4 from the Reports database. Although the form is from the Reports database, the data will be from the current database. This usually only makes sense if the two databases have the same fields.

```
printusingform "Reports","Standard Report #4"
print dialog
```

Printing Data in an Array

The printonemultiple statement prints a form over and over again without advancing from record to record (see “PRINTONEMULTIPLE” on page 5614). Instead of advancing from record to record, a variable is incremented each time the form is printed. This statement is designed for printing information in an array (see “Text Arrays” on page 1236) using a Super Matrix (see “Super Matrix Objects” on page 929). Typical examples include calendars and photo thumbnails. The printonemultiple statement has five parameters.

```
printonemultiple variable,start,end,bump,copies
```

The variable parameter is the name of the variable you wish to increment as each page is printed.

The start parameter is the beginning sequence number or date value. Start can be an integer number, a variable containing a numeric integer or date value, or a formula or function which results in a numeric integer or date value. The start parameter must be less than or equal to the end parameter.

The end parameter is the ending sequence number or date value. End can be an integer number, a variable containing a numeric integer or date value, or a formula or function which results in a numeric integer or date value. End must be greater than or equal to start.

The bump parameter is the increment value for your sequence. Bump may be a number, a numeric variable, or a formula which results in a positive numeric integer. The bump value must be a positive integer for a numeric field. For a date field bump may also be one of the following:

Bump	Description
"M"	bump one month per page
"Y"	bump one year per page
1	bump one day per page
7	bump one week per page

The copies parameter is the number of times the form is printed for each sequence number. Copies may be a number, a numeric variable, or a formula which results in a positive numeric integer (usually 1).

The printonemultiple statement will print a form a predetermined number of times. Each printing of the form may be sequenced with incrementing integer or date values in a specified variable. Note: the printonemultiple statement does not actually print itself, but must be followed by a printonerecord statement (see “Printing a Single Record” on page 1842).

This example prints the next 3 months of a monthly calendar. The example assumes that the form Monthly Calendar will display the month specified by the variable CalendarDate.

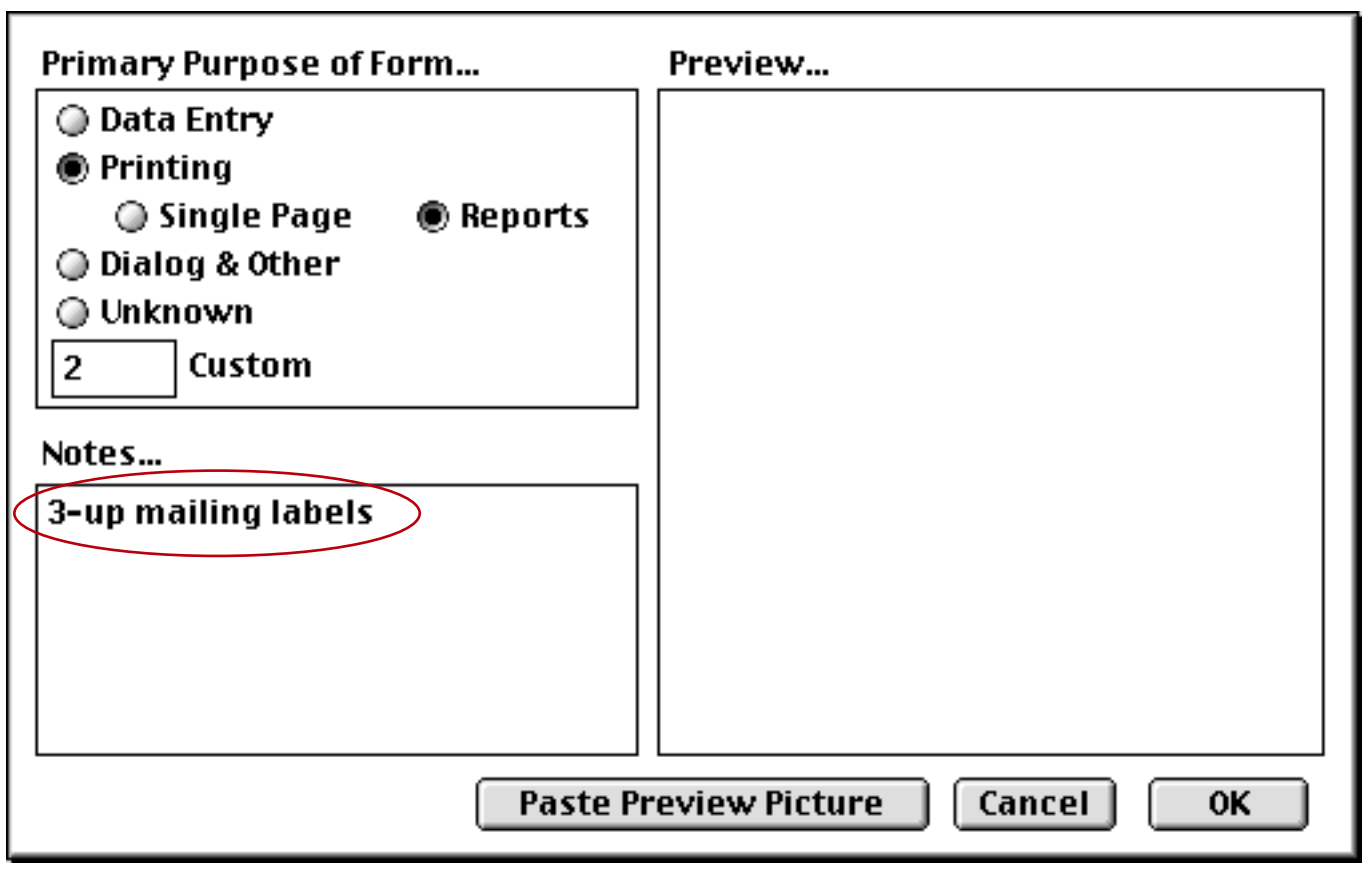
```
fileglobal CalendarDate
openform "Monthly Calendar"
printonemultiple CalendarDate,today(),today()+90,"m",1
printonerecord dialog
closewindow
```

This example prints all the photograph files in the current folder. The example assumes that the form Picture Matrix will display 20 pictures per page, probably using a SuperMatrix object. (It’s possible for a procedure to find out how many pictures are on each page, see “Super Matrix SuperObject™ Commands” on page 1834). The picture in the top left corner of each is controlled by the global variable PicNumber.

```
fileglobal PicNumber,PicMax
PicMax=arraysize( listfiles("","PICT"),¶)
openform "Picture Matrix"
printonemultiple PicNumber,1,PicMax,20,1
printonerecord dialog
closewindow
```

Form Comments

Panorama allows you to create extra explanation comments for each form in a database. These comments let you keep some notes to yourself about each form-what its purpose is, what kind of paper it is printed on, whatever you want to remember. There's a limited amount of space, however, so don't go into great detail about each field in the form. To create these comments, use the **Form Comments** command in the Setup Menu (graphics mode only). Enter the comments in the box in the lower left hand corner of the dialog.

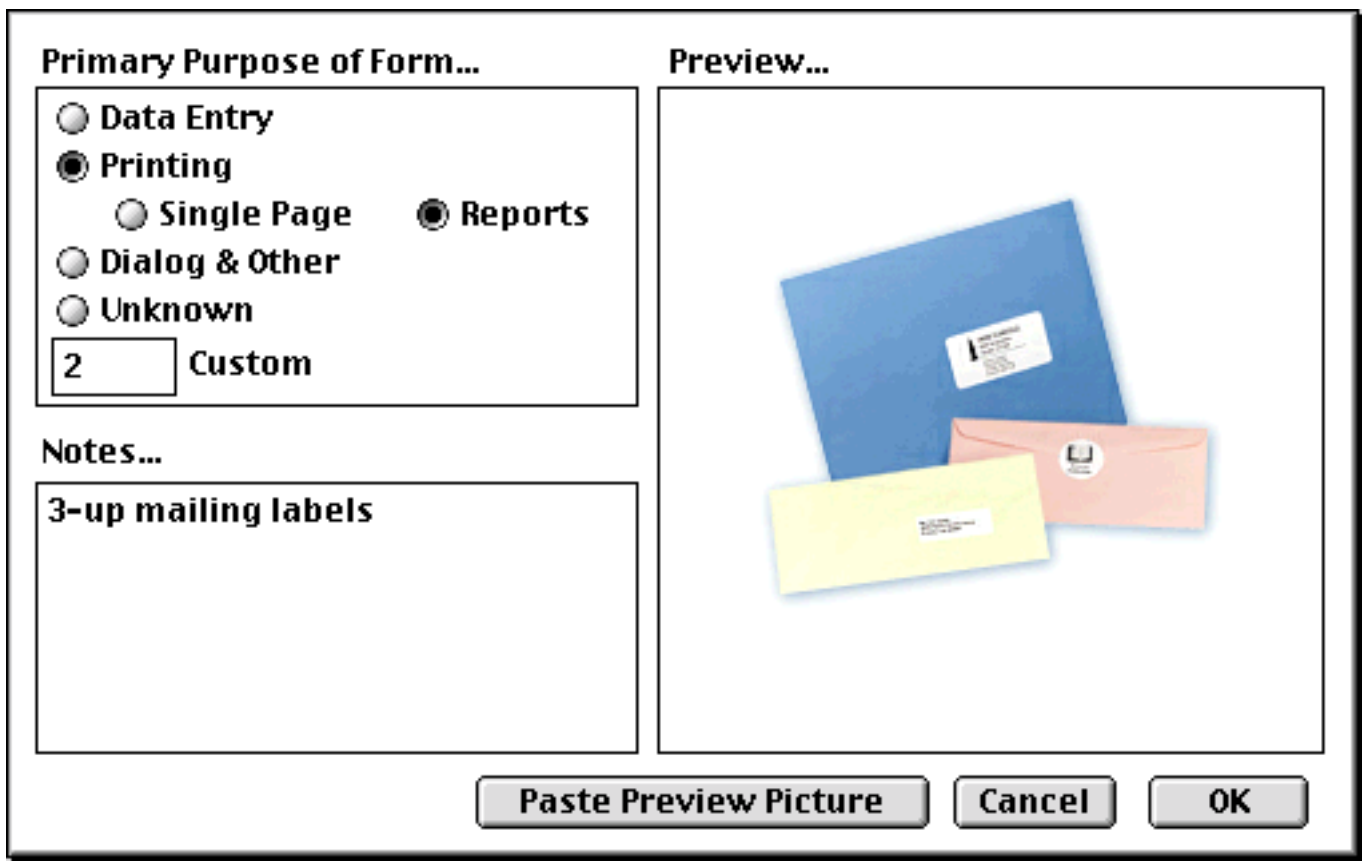


You can also assign a form type using the radio buttons in the upper left hand corner of the dialog. This type is for your information only and may also be used to select classes of forms using the `formselect` procedure statement. The pre-defined form types are:

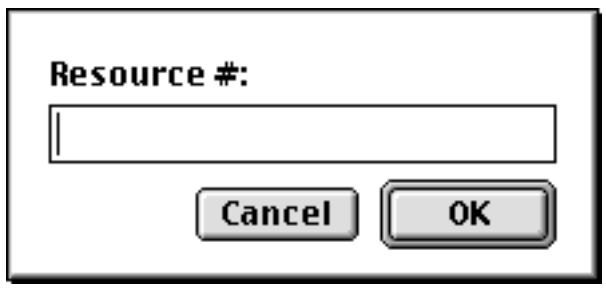
Option	Description
Data Entry	This is for forms that are primarily for data entry.
Printing	This is for forms that are primarily designed for printing. This option is further subdivided into **single page forms** that are designed for printing individual records, for example checks or tax forms, and **reports** that are designed for printing many records at a time.
Dialog & Other	This is for forms that are designed to be displayed as dialog boxes.
Unknown	This is the default setting before a purpose has been assigned.
Custom	Allows you to create your own form classifications (for use with the **formselect** statement - see "The FormSelect Statement" on page 1847). Custom classes may be numbers from 10 to 255.

Remember, these form types are for your information only—Panorama will not stop you from printing a form that is designated as a dialog or from editing data in a form that is designated as a report. However, form types can be very useful to help the database designer (this means you) keep track of what forms are for what.

In addition to the text comments, you can also assign a preview picture to the form. Before opening the **Form Comments** dialog, copy the picture into the clipboard. Once the dialog is open, you can use the **Paste Preview Picture** button to paste the picture into the comment window.



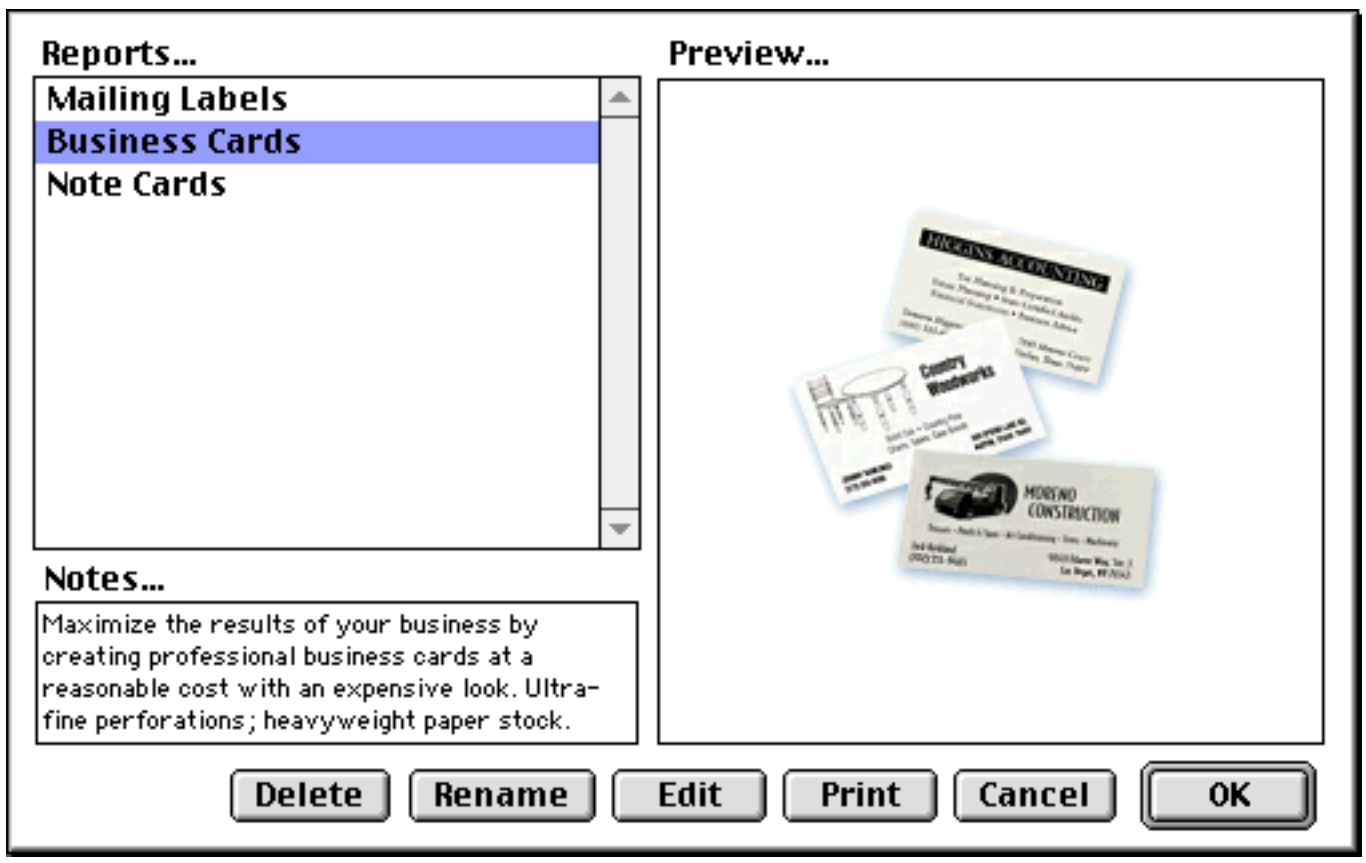
Another way to paste pictures into a form comment is to put the pictures in a resource file. The advantage of this approach is that the picture doesn't waste any of your valuable memory. However, the resource file must be opened in the **.Initialize** procedure for this to work (see "Opening and Closing Resource Files" on page 1562). Once the picture is stored in the resource file (see "Working with Resources" on page 1560), you can open the **Form Comments** dialog, then hold down the **Option** key and press the **Paste Preview Picture** button. Panorama will request the number of the resource containing the picture. Enter the number and press **Enter**.



No matter how you get the picture into Panorama, the picture itself should be no more than 256 pixels high by 256 pixels wide.

The FormSelect Statement

The `formselect` statement pauses a procedure and displays a dialog through which the user can choose a form from the active database (see “FORMSELECT” on page 5267). The dialog may also show the Form Comments information (see “Form Comments” on page 1845.) The dialog will look something like this.



The `formselect` statement has four required parameters.

```
formselect dialog,filter,button,form
```

Dialog is the resource number that identifies the dialog you wish to display. If you do not wish to create your own dialog, with ResEdit for example, you may use Panorama's built in dialog number 2086. Filter is a numeric value used to determine which type of forms will be displayed in the dialog (see “Form Comments” on page 1845). The following table shows the possible filter values.

Value	Selected Forms
0	All forms
1	Data Entry forms
2	Printing forms
3	Dialog & related forms
4 or greater	Custom forms

Button is the name of a variable that will contain the name of the button that was pressed inside the `formselect` dialog. Clicking on any button in the dialog closes the dialog and allows the procedure to continue.

Form is the name of a variable that will contain the name of the form selected in the dialog. If the variable is pre-set to the form name before the `formselect` statement is reached this form will be selected when the dialog opens. If no form is selected this variable will equal `""` .

This example opens the built-in Panorama Form Selection dialog, displaying all forms. It will store the button selection and form selection in the global variables defined.

```
global buttonname,formname
formselect 2086,0,buttonname,formname
```

This procedure opens a custom Form Selection dialog (# 3000) displaying Print forms only and pre-selects the form called Sheet. The procedure makes a decision based on one of three buttons pressed: Cancel, Print, or Edit.

```
local PrintButton,PrintForm
PrintForm = "Sheet"
openresource "Dialogs"
formselect 3000,2,PrintButton,PrintForm
if PrintButton = "Cancel"
   stop
endif
if PrintButton = "Print"
   openform PrintForm
   print dialog
   closewindow
endif
if PrintButton = "Edit"
   openform PrintForm
   graphicsmode
   stop
endif
```

Reading and Modifying Form Comments in a Procedure

Using the `formcomments(` function a procedure can read the form comments from any procedure in any open database (see “FORMCOMMENT(” on page 5263). This function has two parameters, database and form. Here is an example that checks for the word printable in the comments for the current form, and only prints the form if the comments contain this word.

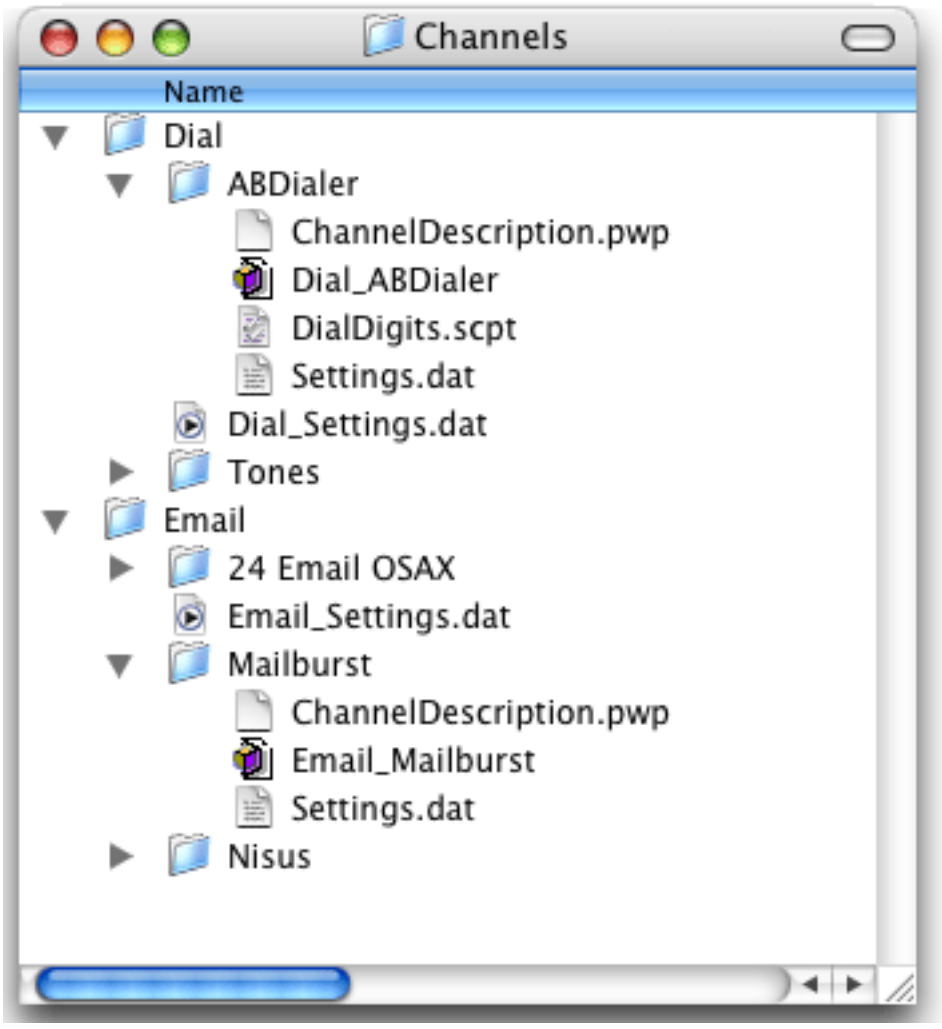
```
if formcomment(info("databasename"),info("formname")) contains "printable"
   print dialog
else
   message "Sorry, this form is not printable."
endif
```

A procedure cannot set the value of the form comments directly, but using the `formcomments` statement it can pause and allow the user to type in form comments (see “FORMCOMMENTS” on page 5264).

Writing Your Own Channel Modules

Panorama comes with a number of channel modules for sending e-mail, dialing the phone, and interfacing with other web sites and third party software. If you have programming experience you can write your own channel modules. To help make this easier we have created a **Channel Workshop** wizard that will create the core of your new module for you. The next few sections provide an overview of this process, but before you write your own new channel we recommend that you take some time to study the modules provided with Panorama.

Each module is contained in a folder that has the name of the module. This folder must contain a database that has the name of the channel and the name of the module, separated by an underscore (_). This is called the *module database*. The illustration below illustrates two of the modules that come with Panorama: ABDialer and Mailburst. (If you use the **Channel Workshop**, described below, it will automatically create the correct folders and database for you.)



As you can see, the module folder must be placed inside the folder for the corresponding channel type. In other words, dialing modules must be in the Dial folder, email modules, in the Email folder, etc.

In the illustration above, each folder contains a .pwp and .dat file. These files are created automatically by the **Channels** wizard. The ABDialer folder also contains an AppleScript file used by the module. You should store any additional files needed by the module within the same folder as the module itself.

The ModuleInformation Procedure

Every module database must contain a procedure named ModuleInformation. Panorama calls this procedure to find out about the settings and capabilities of this module. The procedure has two parameters. Parameter 1 is a text parameter that specifies what information Panorama is requesting. The procedure must return this information in the second parameter using the setparameter statement. Unless noted otherwise the information must be returned in data dictionary format (see “Data Dictionaries” on page 1728). To create a dictionary first create a variable and assign it a value of "". Then use the `setdictionaryvalue` name,value statement to add each name/value pair to the dictionary.

The types of information that Panorama will request are "settings", "settingnames", "settingdefaults", "setting-descriptions", and "url". The ModuleInformation procedure will usually use a case statement to process each of these options.

Type	Description
"settings"	The procedure must return a carriage return separated array listing the settings used by this module. The settings should not contain any spaces, punctuation or special characters.
"settingnames"	The procedure must return a dictionary that contains the settings and setting names. The setting name may contain spaces, punctuation, etc. For example, the AreaCode setting might have a name of Area Code.
"settingdefaults"	The procedure must return a dictionary that contains the settings and default values for each setting.
"settingdescriptions"	The procedure must return a dictionary that contains the settings and default values for each setting.
"url"	The procedure must return a dictionary that contains the URL's to be listed in the URL wizard. In each name / value pair the name is the URL itself and the value is a short description of the URL (perhaps the page title).

You don't have to write the ModuleInformation from scratch yourself. Instead, you can use the **Channel Workshop** wizard (described below) to write the procedure for you.

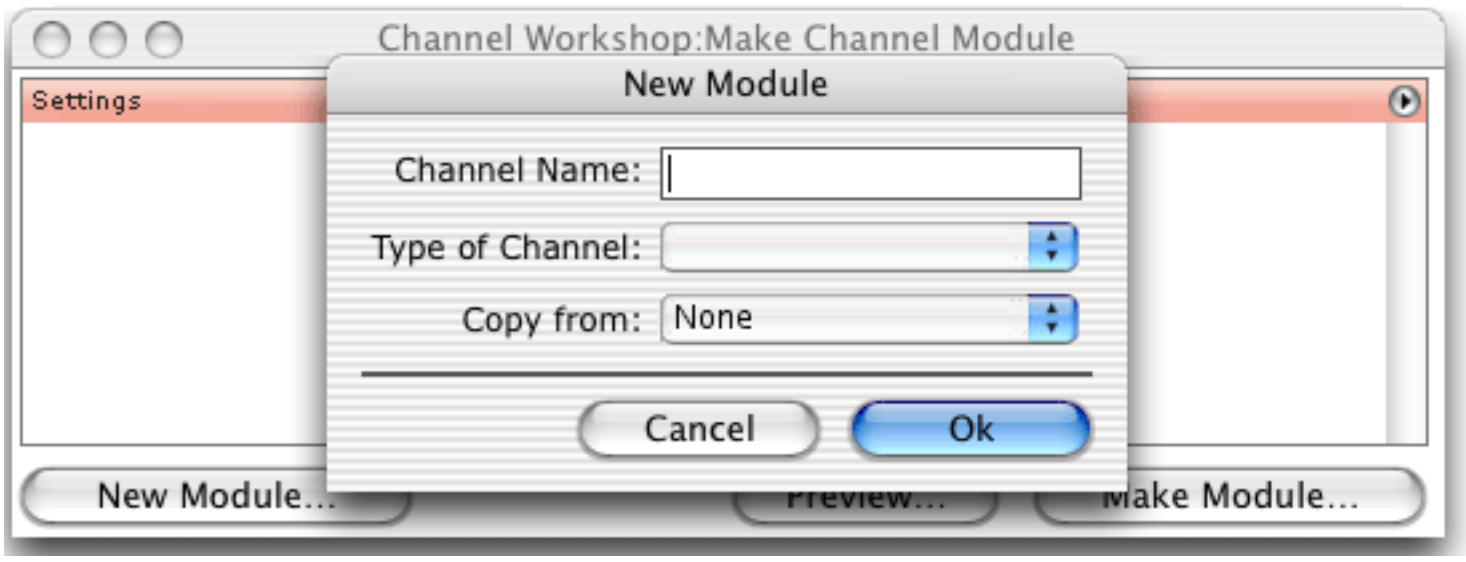
Channel Specific Procedures

Each type of channel will require one or more additional procedures to perform the work of the channel.

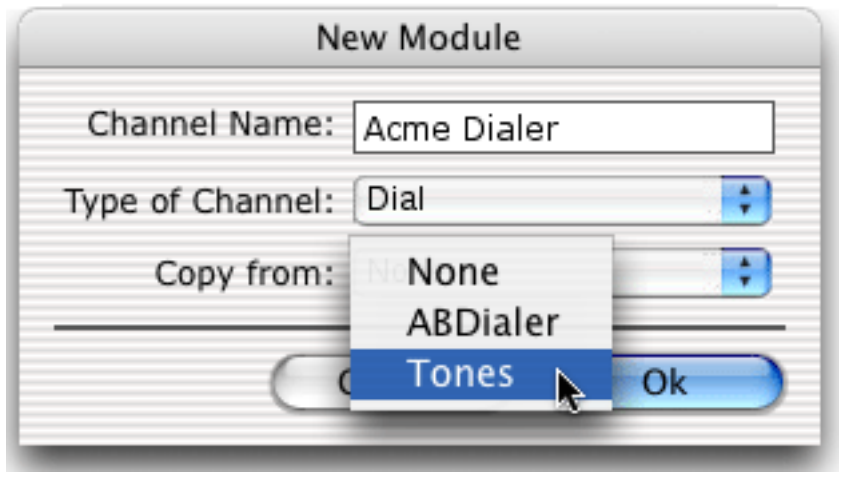
Type of Channel	Procedure Name	Description
Dial	DialDigits	This procedure has one parameter - the digits to dial. The procedure should dial these digits exactly, with no additions or subtractions.
Email	SendEmail	This procedure sends an e-mail. The procedure has two parameters: options and message. The options parameter must contain assignments for various e-mail options, including the recipient, sender, subject, server, etc. For more information see the documentation for the SendMail statement in the Programming Reference Wizard. The message parameter will contain the body of the e-mail message to be sent.

The Channel Workshop Wizard

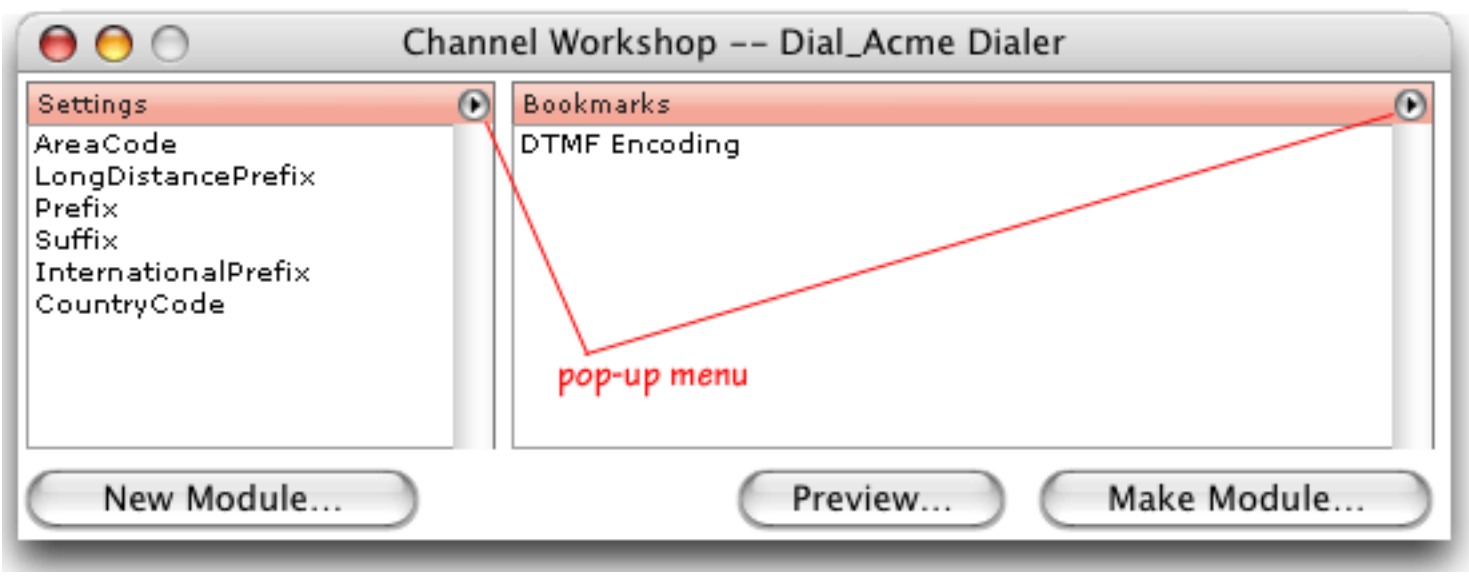
The **Channel Workshop** wizard automates much of the drudgery of creating a channel module. Of course the workshop can't write all of your code for you (it does write some of it) but it does automate the basic tasks required to create a channel, freeing you to focus on writing the actual code necessary to implement the channel. When you first open the **Channel Workshop** it automatically begins the process of creating a new channel.



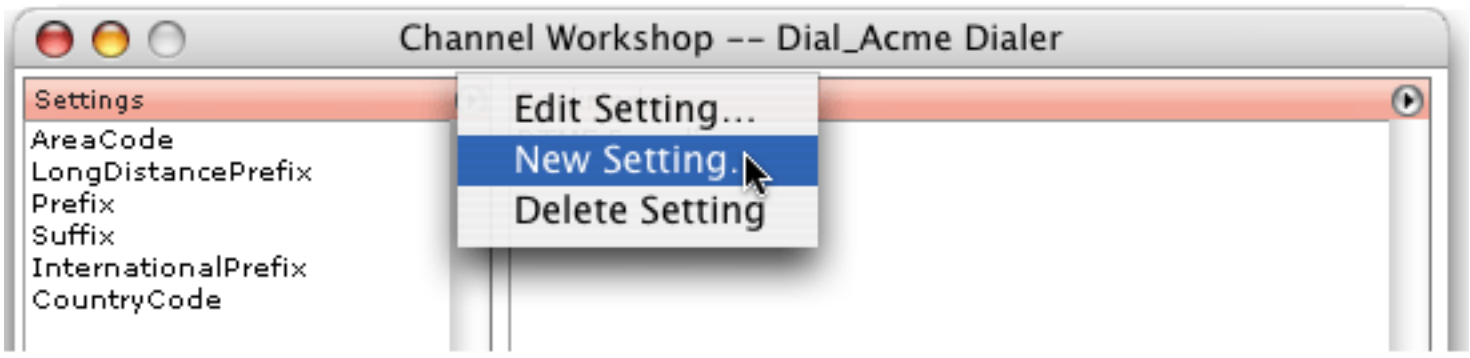
Your first step is to type in a name for the new channel, and to use the pop-up menu select the type of channel. Once you have selected the type of dialer you may want to use the bottom pop-up menu to initialize your new module with settings from an existing channel.



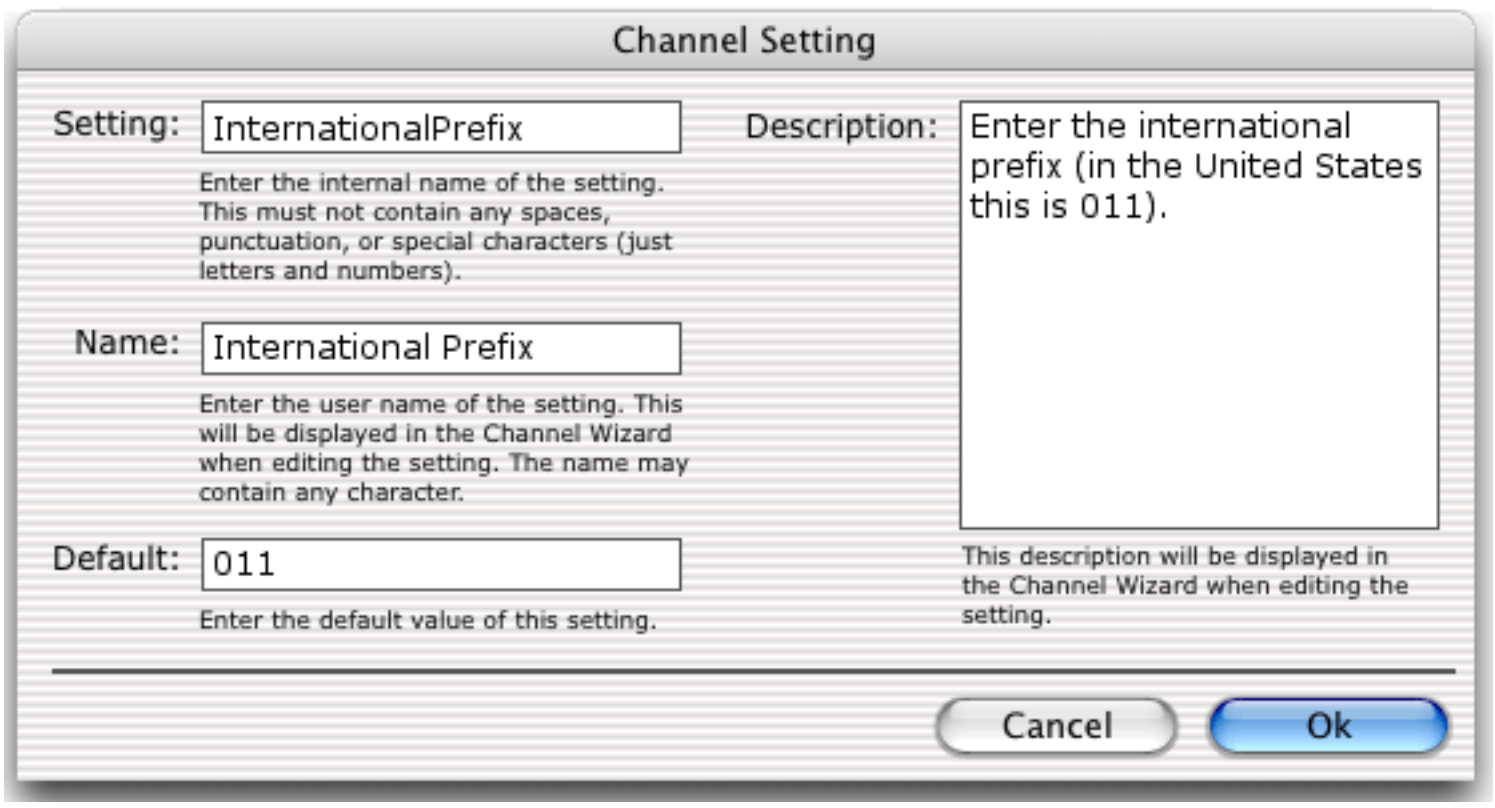
Press **Ok** to start configuring your new module. Since we selected the Copy from option we'll start with some settings already set up.



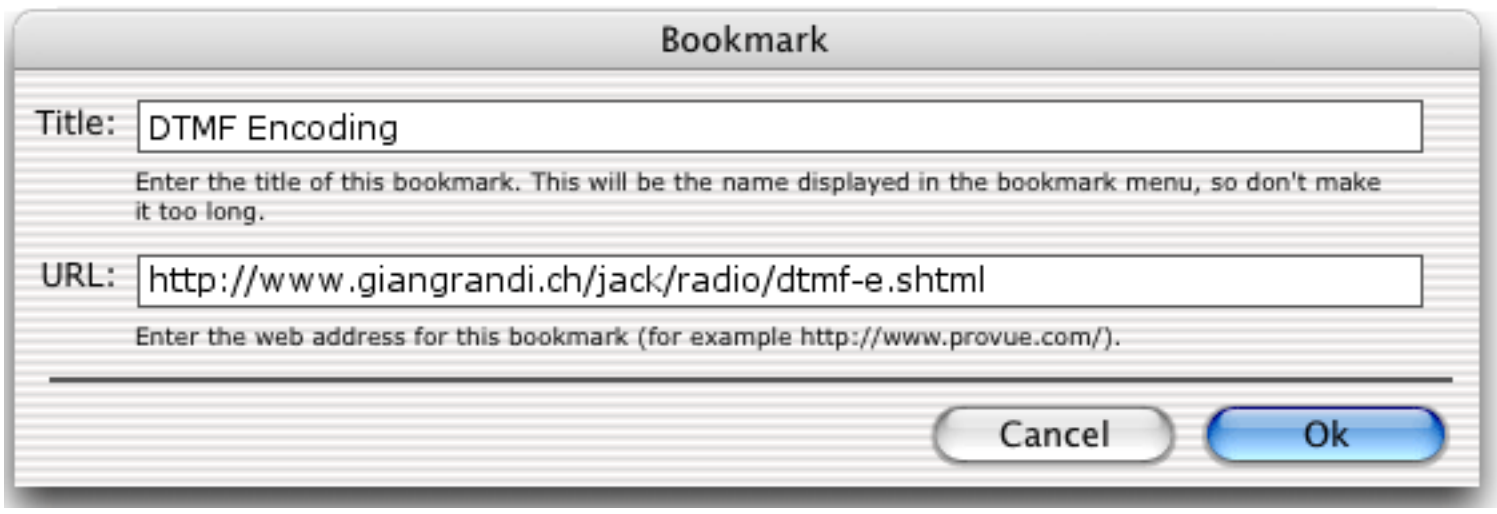
Use the pop-up menu in the upper right hand corner to add, delete, or edit settings.



You can also double click on a setting to edit it.



You can modify the bookmarks the same way.



You can also add bookmarks by dragging from your browser into the bookmark area.

Previewing the ModuleInfo Procedure

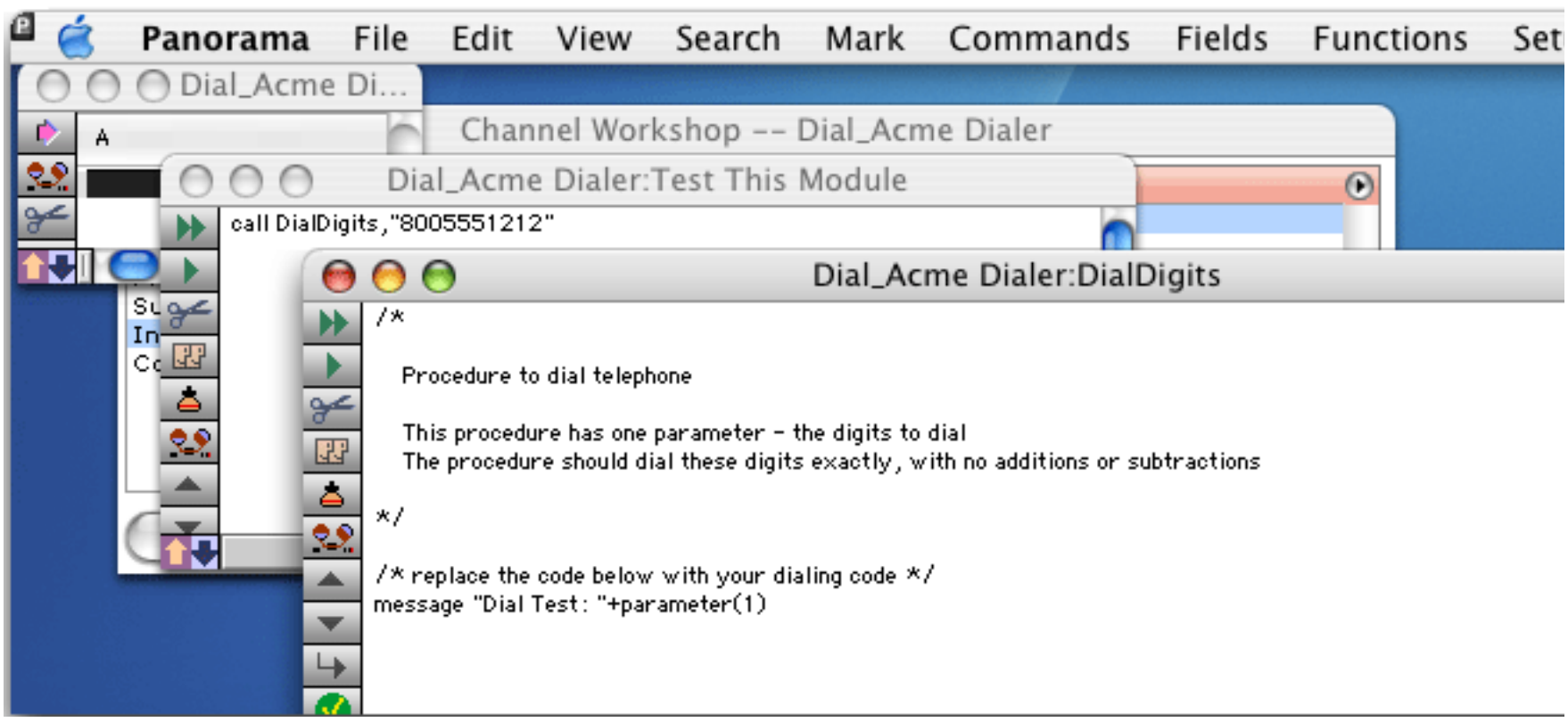
Press the **Preview** button if you'd like to see a preview of the code of the ModuleInfo procedure you are about to generate.



It’s not actually necessary to ever look at this code, but it does give you an overview of the configuration you are setting up.

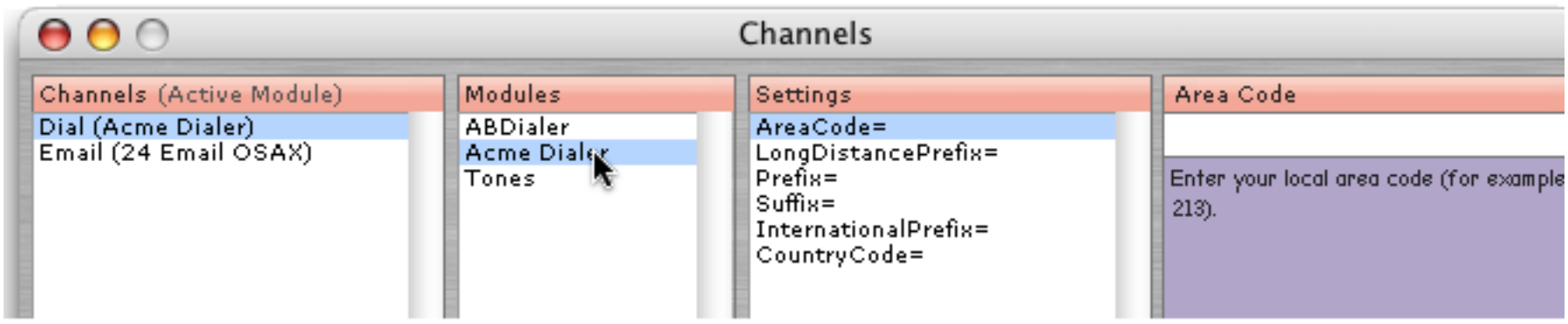
Creating the Module

When you are ready to create the module press the **Make Module** button. As shown below, Panorama will create the new module in the correct location (making any new folders that are necessary). It will then open the new module for you so you can immediately begin working on the code for your new module. To test your new module, bring the Test This Module window to the front and press **Command-R**.



Tip: If your procedure needs to use an AppleScript to control another application don't forget about the new `ExecuteAppleScript` statement, which allows you to build and execute an AppleScript on the fly.

Once your new module is finished you can select and configure it with the **Channels** wizard, just like any other channel module.



Chapter 26: Cross Platform Databases



Most Panorama databases can be prepared for cross platform operation in a few seconds. In fact, for most files the process of transferring a file from the Macintosh to the PC is as simple as adding **.pan** to the file name and transferring the file to the PC.

File Type/Creator vs. Extensions

The Mac OS uses an invisible 8 character designator to identify the type of each file. The designator is divided into a 4 character **type code** and a 4 character **creator code**. If you are not a programmer you may not have ever realized these codes were there, because they are completely hidden.

The Windows operating system does not have an invisible designator to keep track of file types. Instead, Windows uses a visible designator appended to the end of the file name. This designator, called an **extension**, is a period followed by 3 or 4 characters. For example, .txt is a text file, .exe is a program file (application), and .pan is a Panorama database. If a filename doesn't have an appropriate extension, Windows can't tell what kind of file it is.

Although there are hundreds of different extensions, there are only a handful that apply to Panorama databases and their associated files.

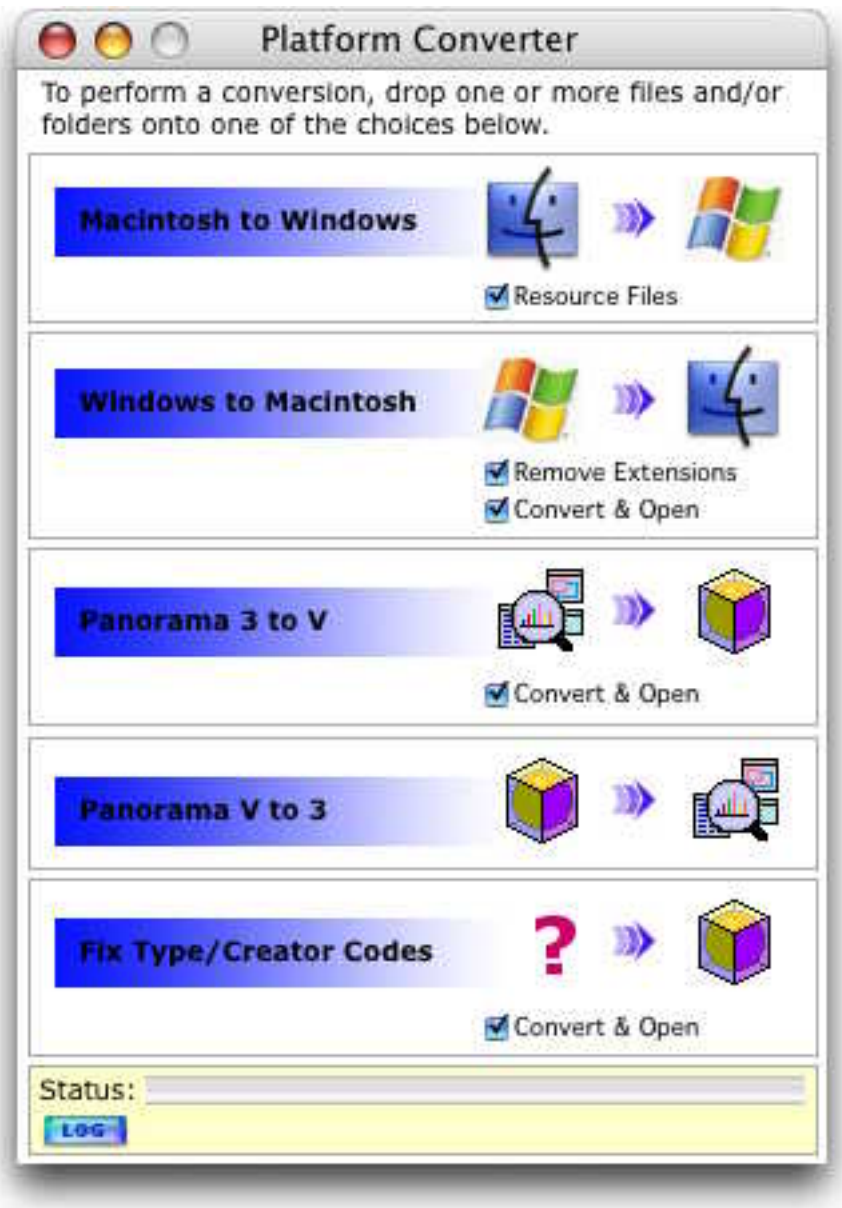
Extension	Type of File
.pan	Panorama Database
.pnz	Panorama File Set
.pwp	Panorama Word Processor File
.pct	Macintosh Picture (PICT) File
.rsr	Macintosh style Resource File
.txt	Text File

Before a file may be used on the PC, it must have the correct extension added. If you only have a few files, you may wish to simply type in the extensions yourself.

Since the Macintosh version of Panorama does not normally use extensions, we've mostly tried to hide them within Panorama itself. For example, if you open a database named Checkbook.pan, Panorama will display it simply as Checkbook in the window title, without the .pan extension. When you save a file with the **Save As** dialog, it is not necessary to type .pan if the file original had a .pan extension -- Panorama will add the extension for you.

Platform Converter Wizard

If you have more than a few files to convert you can use the **Platform Converter** wizard to help. The converter can automatically convert an entire folder of files (including subfolders, if any). The converter examines the hidden type and creator codes for each file, and automatically adds the appropriate extension. The Platform Converter can also convert Macintosh resource files for you (more on that later). If you are using MacOS you'll find the **Platform Converter** wizard in the **Utilities** submenu of the **Wizard** menu. (The **Platform Converter** wizard is not available on Windows PC systems because by the time a file has been copied to the PC, the hidden codes needed by the converter are gone. Platform conversion must be performed on a Macintosh system, since that's where the hidden codes are accessible.)



Converting from Macintosh to PC

To convert a file, group of files, or folder full of files from Mac to PC simply select the files in the Finder, then drag them and drop them on the Macintosh to Windows section of the wizard.



The converter will scan each file and folder that was dropped (and subfolders, if any). Based on the hidden file type and translation, the converter will add the appropriate extension to each file. As it performs the conversion, the program keeps a log of everything it does. You can see the log by pressing the **Log** button. The log also shows any errors encountered by the converter. Typical errors include a file name that would be more than 31 characters long with the extension added (the PC allows 255 but the Mac only allows 31), a file name that contains characters not allowed by the PC (for example slash or backslash), or a file type that cannot be converted to the PC (an application, for instance).

Converting Resources

A Macintosh file may actually contain two separate components, called the data fork and the resource fork. The resource fork can be used to hold custom menus, icons, text, pictures, and other items. Windows files, however, only contain one "fork," which corresponds to the data fork.

The Panorama Platform Converter, however, can convert any file that contains a resource fork so that it may be used on the PC. If the Resource Files option is checked, the converter will check each file to see if it contains a resource fork. If it does, the converter will copy the resource information into the data fork of a new file, with an extension of .rsr. The PC version of Panorama knows how to access the items (custom menus, etc.) that are stored inside the .rsr file.

Converting from PC to Macintosh

If you create a Panorama database on the PC and then copy it back to the Mac, the new file will appear as a generic icon on the desktop. To convert a file, group of files, or folder full of files from PC to Macintosh simply select the files in the Finder, then drag them and drop them on the Windows to Macintosh section of the wizard.



The converter will scan each file and folder that was dropped (and subfolders, if any). Based on the extension (.pan, .pnz, etc.) it sets the proper hidden type and creator code for each file, allowing the file to be accessed properly on the Macintosh. If the Remove Extensions option is checked the converter also removes the extension from the file name. It does not, however, convert .rsr files back into Macintosh resource files. If the Convert & Open option is checked then Panorama will open any databases after they are converted. (Note: Databases converted from PC to Mac can only be opened with Panorama 4.0 or later. Older versions of Panorama will not open these files!)

Converting from Panorama 3.x to V (Macintosh)

Panorama 3.1 and Panorama V can both co-exist on the same Power Macintosh computer, and in fact both can be running at the same time! Databases initially created with Panorama 3.1 will automatically open Panorama 3.1 when double clicked; databases created with Panorama V will automatically open Panorama V when double clicked. Panorama V can open databases created with Panorama 3.1 or earlier simply by dragging these files onto Panorama V or by using the **Open File** dialog. If you want Panorama V to launch automatically when a Panorama 3.1 database is double clicked, you must convert it using the **Platform Converter** wizard. After the database is converted it's icon will change from the old Panorama icon to a new icon.

To convert databases files so that they will automatically launch Panorama V instead of 3.1 select the files in the Finder, then drag them and drop them on the Panorama 3 to V section of the wizard.



The wizard will scan the files and change each one to launch Panorama V instead of Panorama 3. If the Convert & Open option is checked then Panorama will immediately open the databases after they are converted.

Technical Note: The Panorama 3 to V conversion does not actually change the internal structure of the file. Panorama 3, 4 and V share the exact same internal file structure. The only modification this conversion makes is to change the hidden Type and Creator codes that the Macintosh OS uses to identify the application to be launched when a file is double clicked.

Converting from Panorama V to 3.x (Macintosh)

To convert databases files so that they will automatically launch Panorama 3 instead of V select the files in the Finder, then drag them and drop them on the Panorama V to 3 section of the wizard.



The wizard will scan the files and change each one to launch Panorama 3 instead of Panorama V. Note: If a database was lasted opened on a Windows computer Panorama 3.1 will not be able to open the file, and it will not be converted.

Fixing a Database With Missing or Incorrect Type/Creator Information

The Macintosh operating system normally maintains special invisible codes that identify the type of each file, and the application that created that file. These are called type and creator codes. Sometimes, however, these invisible codes can get stripped from a file. When this happens the file will have a generic or incorrect icon. One of the most common ways for this to happen is when sending a file via e-mail — some e-mail programs do not correctly transmit the type and creator codes. It can also happen if the file is copied onto a PC formatted disk, then back to the Macintosh.

If a Panorama database loses it's type and creator code, you can restore these codes by dragging the file (or files) onto the Fix Type/Creator Codes section of the **Platform Converter** wizard.



Any file that is dropped on this section will have it's type and creator codes modified to the correct codes for a Panorama database. **Be sure that you only drop files that you are sure are Panorama databases on this section!** If you drop a file that is not a Panorama database on this section, it will be modified so that its type and creator code will fool the system into thinking that it is a Panorama database. Double clicking such a file will cause Panorama to try to open the file thinking that it is a database, and may cause Panorama to crash.

Sharing Databases Across a Cross Platform Network

In addition to transferring files back and forth between a Mac and a PC, you can actually share a database (or collection of databases) across a cross platform network. It's annoying on the Macintosh, but if you want to use a database on both the Mac and PC, you must include the extension as part of the filename (.pan, etc.) even when you are using the file on the Macintosh. To help keep the transition smooth, the Macintosh version of Panorama slightly modifies it's behavior when it detects a database with the .pan extension. Just as when using the PC version of Panorama, the extension is removed for internal use. So if you open a file named Checkbook.pan, the window title will simply be Checkbook, without the .pan extension. When a file that was originally opened with the .pan extension is saved, the .pan extension is automatically added to the filename, whether you use **Save**, **Save As**, or **Save A Copy As**. The main goal is to keep any existing procedures that reference file names working without changes whether there is an extension or not.

Cross Platform vs. Older Versions of Panorama

The processor used in Windows computers (x86/Pentium) stores numbers in a different format from the processors used in Macintosh computers (PowerPC/68K). Since Panorama files contain many numbers, a conversion must be performed when a database is moved to a different platform. Let's suppose a database is created on a Macintosh. The first time the database is opened on any Windows machine, the numbers inside the database are automatically converted to PC format in memory. The conversion only takes a split second, so Panorama doesn't even notify you that the conversion is happening. When the file is saved, the file with the converted numbers is written to disk. When you re-open the file on the PC, no further conversion is necessary. However, if you transfer the file back to a Macintosh computer and open it, Panorama must re-convert the numbers in the file. Again, this happens automatically, and in a split second. In fact, the whole process is so transparent, you'll never notice it with one exception. The exception is if you attempt to open a file that has been saved on the PC on an older (Panorama 3) version of Panorama. Since older versions of Panorama do not have the conversion code, they will be unable to open the file. For now, the only solution is to make sure you open and save the file on a Macintosh computer before attempting to use the file with an older version of Panorama.

Cross Platform Font Usage

If a font has the same name on the Macintosh and the PC then it can be used in a database on either type of computer. If the database is transferred from a Macintosh to the PC or PC to Macintosh the font will continue to work properly.

Panorama has special handling for four special fonts.

Macintosh	Windows
Geneva	Alpine
New York	Yankee
Chicago	City
Monaco	Block

On the Macintosh these four fonts are always present as universal fonts, so you can rely on them always being available. We have created the four equivalent fonts for Windows computers to guarantee that these fonts are always available on any computer. For example, if you create an object using the Geneva font on a Macintosh computer it will automatically be translated to the Alpine font when displayed on a Windows PC computer. If you want to make sure that your database will display properly on any computer you should restrict yourself to using only these four fonts.

Cross Platform Programming

If you've been doing Panorama programming with a previous version of Panorama, you're probably wondering what it will take to get your procedures and formulas to work cross platform. The good news is, probably nothing! We've gone to great lengths to make Panorama for the PC completely compatible with previous versions, as you'll see below. The payoff is that we have successfully ported several large Panorama applications to the PC without making a single change to the applications. No procedures were changed, no forms, fields, nothing. These applications include the Panorama 3 MegaDemo, Power Team, and several complex third party applications, including one with dozens of files and hundreds of forms and procedures.

So far we have encountered only one database that required changes to work on the PC - the Panorama On-Line Reference. The changes required about 10 minutes to complete, and were needed because several procedures referred to subfolders named •Statements, •Functions(, etc. Unfortunately, the • character is not allowed in a Windows file or folder name, so it had to be changed. Basically, unless your database uses special Macintosh only features (the System folder, AppleScripts, special Apple only characters) you shouldn't have to touch your databases at all, just add extensions and go!

File Name Extensions and the OpenFile Statement

Windows files have extensions (.txt, .pan, etc.) and Macintosh files do not. We've programmed Panorama for the PC so that in almost all cases, your existing procedures will work just fine even if they open other databases. For example, suppose you have a database named Checkbook, and you want to open it inside a procedure. It's simple, right? Just use the `openfile` statement:

```
openfile "Checkbook"
```

However, on the PC, the file that is opened is actually named Checkbook.pan. Don't worry, however -- Panorama will automatically add the extension for you. You don't have to change your procedure at all, and it will automatically work on either the Macintosh or the PC. By the way, it's ok to include the extension if you wish:

```
openfile "Checkbook.pan"
```

However, this code is not portable. It will not work on the Macintosh unless the file is actually named Checkbook.pan.

By the way, there is one case where the .pan extension will be automatically added even if you are on the Macintosh. If the currently open file has a .pan extension, Panorama will assume that the file you want to open has a .pan extension. This allows you to build a set of files that can be shared cross platform on a server (which must all have a .pan extension, even when used on the Mac).

Confused? Don't be. The bottom line is you should pretty much always be able to leave off the extension when using the `openfile` statement.

When programming on the Macintosh you can use the `nodefaultextension` statement to open a database that doesn't have a .pan extension even if the current database does have a .pan extension. For example, suppose that you are working with a database named Contacts.pan. The procedure below will open the database named Schedule.pan.

```
openfile "Schedule"
```

However, what if the database you want to open is actually called Schedule, not Schedule.pan. In that case you must add the `nodefaultextension` statement immediately before the `openfile` statement, like this.

```
nodefaultextension
openfile "Schedule"
```

This revised procedure will open the Schedule database.

Name Extensions and Window Names

Panorama removes the .pan extension from the in-memory copy of the database. This means that you won't see the extension in the window title, and should not include the extension when using the `window` statement. Panorama also will not include the extension as part of the file name returned the `info("databasename")`, `info("windowname")`, or `info("files")` functions. You also should not include the extension in any statements or functions that require you to specify the name of an open database (for example `lookup(`, `grabdata(`, `arraybuild`, etc.) Bottom line -- just keep programming the same way you always have.

Flash Art Formulas

When the current database has a .pan extension and the Flash Art formula refers to a disk file (as opposed to a picture in the Flash Art Gallery or a resource) Panorama will automatically add the extension .pct to the final result (unless the formula generates an extension itself). You should make sure that any picture files used in a Flash Art or Super Flash Art formula end with the .pct extension (the Panorama Platform Converter will take care of this for you).

Using Partial Paths to Reference SubFolders

On the Macintosh you can use a "partial path" to reference a sub-folder of the folder that contains the database. Partial paths always begin with a colon. For example, ":Photos:Grand Canyon" refers to a file named Grand Canyon in the Photos folder (the Photos folder must be in the same folder as the database). When this partial path is used on the PC, Panorama automatically converts the colons into backslashes for you. For example, you might use a partial path like this in a Flash Art SuperObject or in the `openfile` statement.

Hard Coded Folder Locations

If your program contains hard coded folder locations (for example "My Disk:Samples:Contacts") these will have to be changed. Of course, you probably don't have any, since these would not work on different Macintosh systems either.

If you build a folder location with the `folderpath(` and `dbinfo(` functions, you'll still be alright. On the PC, this will result in a path that looks something like C:\Samples\1999\, which can be fed into the `folder(` function or used anywhere a path name may be used (for example Flash Art or the `openfile` statement).

The `info("panoramafolder")` function also works on both the Macintosh and the PC.

Is It a Mac or a PC?

Panorama V includes true-false functions for determining whether the current computer is running Windows, OS 9 or OS X.

Function	Description
windows()	True if running on a Windows computer
osx()	True if running on a Macintosh OS X computer
os9()	True if running on a Macintosh OS 9 computer

These functions are not available in earlier versions of Panorama. If you want your database to be compatible with Panorama 4 you can test for Macintosh vs. Windows this way:

```
if folderpath(info("panoramafolder"))[2,3]=":\"
   /* PC */
else
   /* Macintosh */
endif
```

This works because all PC pathnames begin with a letter followed by :\, for example C:\ (main hard disk) or D:\ (cd-rom).

Chapter 27: AppleScript



AppleScript is a programming language included with the Macintosh operating system. Using AppleScript a program can be written that works across multiple applications. For Panorama users the primary advantage of AppleScript is that it allows you to program Panorama to work with other applications.

Learning Basic AppleScript

This chapter assumes that you are already familiar with the basics of AppleScript programming (often called "scripting"). If you haven't done AppleScript programming before, here are some of the books that we have found useful:



AppleScript : The Definitive Guide

- Author: Matt Neuburg
- Paperback: 476 pages
- Publisher: O'Reilly; 1 edition (November 20, 2003)
- ISBN: 0596005571

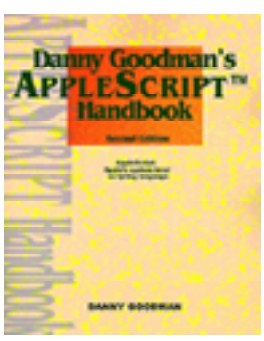
If you've already got some programming experience, this book is the ultimate reference for AppleScript. Rather than rely on published documentation, the author did his own testing to uncover all the nooks and corners of this complex language, and he doesn't hesitate to point out the shortcomings as well as the strengths of the AppleScript system. Be sure to pick up at least one other AppleScript book, because this one covers only the language itself, and does not include any specifics of how to use AppleScript with other applications.



AppleScript For Applications

- Author: Ethan Wilde
- Paperback: 480 pages
- Publisher: Pearson Education; 1st edition (November 15, 2001)
- ISBN: 0201716135

This book takes a cookbook approach to AppleScript. The book includes complete ready-to-use scripts for popular Mac apps such as Word, Photoshop, and QuarkXPress plus a companion Web site for downloads and helpful links. These scripts can be used as is or as a starting point for your own custom scripts. It also includes a step-by-step beginner's tutorial to getting started with AppleScript.



Danny Goodman's AppleScript Handbook

- Author: Danny Goodman
- Paperback: 576 pages
- Publisher: iUniverse; 2nd edition (April 1, 2000)
- ISBN: 0966551419

This is the classic AppleScript book, originally published many years ago when AppleScript was first released. It's a bit dated, but still a good resource.

Script Debugger

AppleScripts can be edited with Apple's free Script Editor and also within Panorama's procedure editor. However, if you are planning on doing any serious work with AppleScript we highly recommend that you investigate a product called **Script Debugger**, by *Late Night Software*.

```
http://www.latenightsw.com/index.html
```

This product isn't cheap, but the investment will pay for itself quickly if you are writing scripts of any complexity at all. It's ability to debug, examine variables, and explore the object model have turned AppleScript projects we work on here at ProVUE Development from a frustrating hit or miss experience into a smooth productive workflow. Working with **Script Debugger** is like turning on the light in a dark room. We don't have any connection with *Late Night Software*, but we do recognize great tools when we use them, so we wanted to pass along the secret to you.

AppleScript and Panorama

AppleScript is an unusual language in that it is not constant. Instead of creating a complete language, Apple developed only a very skeletal framework. The rest is filled in by the actual application being programmed. The result is that each application is programmed somewhat differently.

If you are an experienced AppleScript programmer, this list describes how Panorama fits into the AppleScript scheme of things. (If this list doesn't mean anything to you, go back and review one of the AppleScript books listed above.)

• Panorama is not recordable.
• Panorama supports the object model for transferring numbers and strings between AppleScript and Panorama.
• Panorama does not support the **whose** clause.
• AppleScripts can launch Panorama procedures.
• Panorama procedures can be included within a script.
• Panorama procedures can launch AppleScripts.

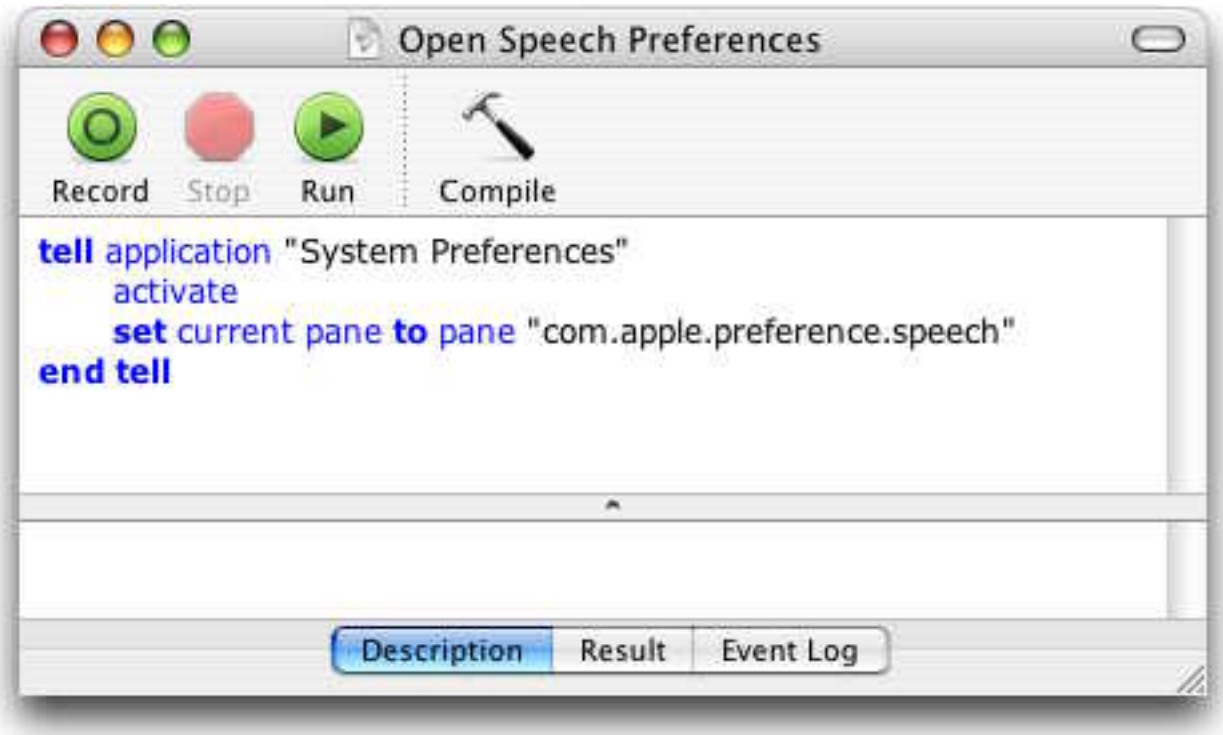
If you examine this list carefully, you can see that AppleScript really gives you the same control over Panorama that Panorama's built in programming language gives you. Anything that can be done in a Panorama procedure can also be done within an AppleScript program, and other applications can be programmed as well.

Using AppleScript to Control Other Applications from Panorama

AppleScript extends the power of Panorama's programming language to other programs on your computer. Using AppleScript you can control the Finder, web browsers, e-mail clients, FTP software, or any other application that supports AppleScript. You can also use AppleScript to access the UNIX command line. There are two ways to use AppleScript from within Panorama — you can launch a script that you have pre-defined using Apple's Script Editor, or you can include AppleScript's directly within your Panorama procedures. You can even build and execute an AppleScript on-the-fly.

Launching an External Script from Panorama

Using the `startscript` statement a procedure can launch an AppleScript that you have previously created with Apple's **Script Editor** (or any other script editing software, like Late Night Software's **Script Debugger**).



Within a Panorama procedure you can use the script to open the control panels folder by using the `startscript` statement.

```
startscript "Open Speech Preferences"
```

When this procedure is execute the Speech Preferences control panel will open.



This example assumes that the Open Speech Preferences script has been saved in the same folder as the database. If it is not in the same folder you can specify the complete path name for the script, for example Hard Disk:My Scripts:Open Speech Preferences.

Building a Script Within a Panorama Procedure

The executeapplescript statement allows you to build and execute an AppleScript on-the-fly within a Panorama procedure. This means that you don't have to keep track of a bunch of separate script files and you can easily change the script to match changing conditions. (As you'll see, it also makes it possible for Panorama to easily access the result of a script.) The executeapplescript statement has one parameter:

```
executeapplescript scripttext
```

The scripttext parameter is the text of the script that you want to launch. This text should be typed in exactly as you would enter it into the *Script Editor* program. For example, here is an all-in-one procedure that opens the *Speech Preferences* control panel.

```
executeapplescript “tell application "System Preferences"
   activate
   set current pane to pane "com.apple.preference.speech"
end tell”
```

Unlike the example in the previous section, this procedure does not require a separate script file — the procedure is completely self contained.

The example below also opens the *System Preferences* panel, but it allows the users to choose which panel they want to open. It then builds the AppleScript on-the-fly depending on which panel was selected.

```
local sysPrefPanels,sysPrefPanel
sysPrefPanels="com.apple.preference.speech"+¶+
   "com.apple.preference.expose"+¶+
   "com.apple.preference.internet"+¶+
   "com.apple.preference.classic"+¶+
   "com.apple.preference.datetime"
superchoicedialog sysPrefPanels,sysPrefPanel,“title="Select Panel" height=400 width=300”
executeapplescript “tell application "System Preferences"
   activate
   set current pane to pane ”+applescriptstring(sysPrefPanel)+“
end tell”
```

This example uses the applescriptstring(function to place the selected panel into the AppleScript. This function takes the text parameter and adds quotes around it so that it can be used as a literal value within the script. It also escapes any quotes within the text so that AppleScript will understand the resulting value.

Getting the Result from an AppleScript

If there are no compilation errors, the executeapplescript statement runs the script you give it. Panorama waits while the script is running, and when it is finished the script result is placed in the global variable AppleScriptResult. (Note: The script should not have any tell blocks that reference Panorama itself. Since Panorama is waiting for the script to finish, this will cause the script to hang. If you need to get a value from Panorama into the script, simply include it in the source code as a constant, as described in the previous section.) The example below uses AppleScript to find out the names of all currently mounted disks.

```
executeapplescript “tell application "Finder"
   return name of every disk
   end tell”
message  AppleScriptResult
```

The result will be something like this:

```
{"Panther", "Classic", "Video"}
```

Of course this example is rather contrived since you could easily get this information with Panorama's built in info("volumes") function, but it does server to illustrate the power of this feature for controlling and communicating with other software on the computer.

Using Panorama Field and Variable Values within an AppleScript

You've already learned how to use the `applescriptstring(` function to include field and variable values within a script. A simpler method is to use the `executeapplescriptformula` statement. This statement is the same as the `executeapplescript` statement except that it does some special pre-processing before creating the AppleScript. This pre-processing checks for special tags that you have inserted in the script text. The

```
"«formula»"
```

tag may be used to insert the result of any Panorama formula into the script. The formula must produce a text value, and will be inserted into the script as an AppleScript constant. The

```
#«formula»#
```

tag may be used to insert the result of any numeric Panorama formula into the script. The formula must produce a numeric value, and will be inserted into the script as an AppleScript numeric constant.

Here is a revised version of the procedure to select and open a *System Preference* panel.

```
local sysPrefPanels,sysPrefPanel
sysPrefPanels="com.apple.preference.speech"+¶+
   "com.apple.preference.expose"+¶+
   "com.apple.preference.internet"+¶+
   "com.apple.preference.classic"+¶+
   "com.apple.preference.datetime"
superchoicedialog sysPrefPanels,sysPrefPanel,“title="Select Panel" height=400 width=300”
executeapplescriptformula
   “tell application "System Preferences"
      activate
      set current pane to pane "«sysPrefPanel»"
   end tell”
```

Before the `executeapplescriptformula` statement compiles the AppleScript it replaces the `"«sysPrefPanel»"` tag with a quoted form of whatever panel was selected, for example `"com.apple.preference.internet"` or `"com.apple.preference.expose".`

Using the UNIX Command Line

The `executeunix` statement executes a UNIX shell command.

```
executeunix shellcommand,result
```

The shellcommand parameter is simply the command you want to execute, for example ls, curl, ps, etc.

The result parameter must be the name of a field or variable. When the shell command is complete, this field or variable will contain whatever output was produced from this command (standard output).

This example use the ps command (process status) command to check whether or not an application called *Path Finder* is currently running. This procedure uses the `info("unixusername")` to find out the name of the currently logged in user.

```
local shellResult,runningPathFinder
runningPathFinder=false()
executeunix “ps -w -w -U ”+info("unixusername") shellResult
if AppleScriptResult contains "Path Finder.app"
   runningPathFinder=true()
endif
```

The `executeunix` statement opens the entire universe of the power of UNIX to Panorama programmers. Entire books could be written about the UNIX command line, and in fact they have! We'll refer you to Amazon, Barnes & Noble or Borders for more information on the UNIX command line.

Using AppleScript to Control Panorama from Other Applications

So far we've covered controlling other applications from Panorama, which is the most common need of most Panorama users. However, it is also possible to control Panorama itself from an AppleScript.

Everything You Really Need to Know…

Although Panorama's AppleScript dictionary includes over a dozen commands that can be used in a multitude of combinations, most scripts involving Panorama boil down to two basic operations: 1) transferring data between AppleScript variables and Panorama fields and variables, and 2) running programs written in Panorama's built–in programming language. The next two pages will show you the easiest methods to accomplish these two operations, and should meet 99.95% of your Panorama AppleScripting needs without even having to read the rest of the chapter.

Value of Cell

Within scripts, you'll use the phrase `value of cell` to access and modify Panorama fields and variables. This phrase must be followed by the name of the field or variable (in quotes). For example, this script checks to see if the PaymentMethod field in the current record of the current database is MasterCard.

```
tell application "Panorama"
   if Value of Cell "PaymentMethod" = "MasterCard" then
      -- process master card
   end if
end tell
```

The `value of cell` phrase may be used to access either fields or global variables. When it is used to access a field, that field must be in the currently active database, i.e. the database with the frontmost window within Panorama. (However, Panorama itself does not have to be the active application.)

Using the AppleScript `set` statement, a script can copy Panorama database fields (or variables) into AppleScript variables. This example pulls the name and address out of the current database into the AppleScript variable LabelText, then makes a label in WordPerfect.

```
tell application "Panorama"
   set LabelText to ¬
      Value of Cell "Name" & return & ¬
      Value of Cell "Address" & return & ¬
      Value of Cell "City" & ", " & ¬
      Value of Cell "State" & space & ¬
      Value of Cell "Zip"
end tell
tell application "WordPerfect"
   copy LabelText
      to beginning of paragraph 1
end tell
```

By reversing the order of the parameters, the `set` statement can be used to copy data into Panorama fields or variables. This example gets the name of the topmost window in the Finder, then puts that name into a field named Folder in the current database. (If there is a global variable named Folder, the name will go into the variable instead of into a field.)

```
tell application "Finder"
   set ActiveFolder to name of window 1
end tell
tell application "Panorama"
   set Value of Cell "Folder" to ActiveFolder
end tell
```

It is also possible to access database cells by number instead of by name, although this is rarely of any use. For example, to get the value of the first field in the database use value of cell 1, for the second field value of cell 2, etc.

Executing Panorama Procedures

Using the **execute** statement, you can put a Panorama procedure right in the middle of any AppleScript. Simply type the procedure in quotes after the word `execute`. This example tells Panorama to open the form `Shipping`.

```
tell application "Panorama"
   execute "openform “Shipping”"
end tell
```

This example includes a single Panorama statement, but your procedure may be as complex as you wish. Notice, however, that since the entire procedure must be surrounded by double quotes, you cannot use double quotes within your procedure. There are several solutions to this: 1) you can use smart quotes, as shown above, 2) you can use curly braces {} instead of quotes, 3) you can use single quotes instead of double quotes, or 4) you can use \" for each double quote. All four of these methods are shown in this example:

```
tell application "Panorama"
   execute "openform “Shipping”"
   execute "openform {Shipping}"
   execute "openform 'Shipping'"
   execute "openform \"Shipping\""
end tell
```

The `execute` statement is not limited to a single statement or a single line. It can include complex procedures like this:

```
tell application "Panorama"
   Execute "field {Machine Type}
   formulafill array(SystemInfo,1,{-})
   field {System Version}
   formulafill array(SystemInfo,2,{-})
   groupup
   field {Machine Type}
   count
   outlinelevel 1"
end tell
```

The `execute` statement does not make Panorama the frontmost application. If the procedure is going to display a dialog or allow the user to interact with a window, the script should activate Panorama (bring it to the front) before using the execute statement. To bring Panorama to the front, use the AppleScript **activate** statement.

Transferring Data Between AppleScript and a Panorama Program

Using the `value of cell` phrase, it’s easy to transfer data between AppleScript and the procedure in the `execute` statement. This example uses Panorama’s `dbinfo(` function to get a list of the fields in the currently active database.

```
tell application "Panorama"
   execute "global zFieldList
   zFieldList=dbinfo(“fields”,“”)"
end tell
set dataFields to value of cell "zFieldList"
if dataFields contains "Address"
   (* process address ... *)
end if
```

Here is a script that passes data to a Panorama procedure. This script is designed to be saved as an application. When you drag and drop a text file (or files) on this application it will automatically import and append the text files into the current database.

```
on open fileList
   tell application "Finder"
      repeat with oneFile in fileList
         set filePath to oneFile as string
         tell application "Panorama"
            Execute "global importFile"
            set Value of Cell "importFile" to filePath
            Execute "openfile {+}+importFile"
         end tell
      end repeat
   end tell
end open
```

Note: This example requires the Scriptable Finder, which is included with System 7.5 or later.

Working with Lists

One of AppleScripts powerful features is the **List** data type. Panorama does not directly support the list data type, but you can easily convert between Panorama text arrays and lists, and back again.

Here is an example that transfers a text array to an AppleScript variable and then converts that variable into a list.

```
tell application "Panorama"
   Execute "global aString
   aString=dbinfo({fields},{})"
   set databaseFields to Value of Cell "aString"
end tell
set AppleScript's text item delimiters to return
set databaseFields to¬
every text item of databaseFields
```

This example gets a list of all the currently running programs (called processes) and then converts that list to a comma separated text array.

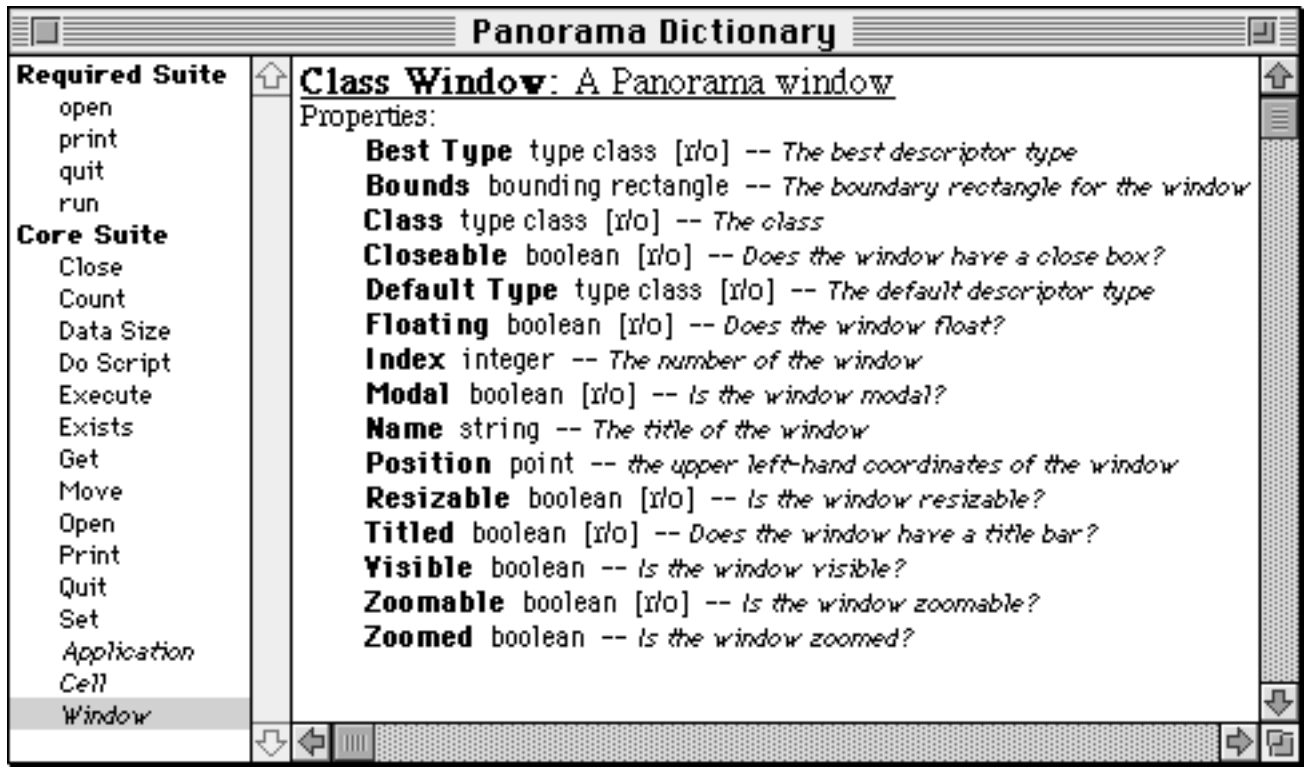
```
tell application "Finder"
   set ProcessList to name of every process
end tell
set AppleScript's text item delimiters to ","
set ProcessList to ProcessList as text
```

The script could continue by passing the text array to Panorama for further processing. (Note: This example requires the Scriptable Finder, which is included with System 7.5 or later.)

AppleScript & Panorama… The Rest of the Story

The previous pages cover everything you really need to know to do work with AppleScript and Panorama. However, there is more to the story. Most of the material that follows really falls into the category of “more ways to do the same things” and is not really vital.

Our guide throughout the rest of this appendix is the Panorama AppleScript dictionary. You can open and view this dictionary from within the AppleScript **Script Editor** application.



The Required Suite

Like all other AppleScript savvy applications, Panorama supports the four required statements: `open`, `close`, `quit`, and `run`.

The `open` statement opens one or more database files. This statement requires one parameter, a reference to one or more files. If you want to reference a single file, simply include the word file followed by the name of the file in quotes.

```
tell application "Panorama"
   Open file "Address List"
end tell
```

Notice that in this example, the word file is part of the parameter, not part of the command. This is different from Panorama's `openfile` statement, which is all one word. (Of course you could also use the `openfile` statement to open files, as shown here:)

```
tell application "Panorama"
   execute "Openfile {Address List}"
end tell
```

The `print` statement prints one or more database files. This statement is identical to selecting the files in the Finder and choosing **Print** from the File menu. However, this method of printing give you no control over what form is used for printing, or what records are selected for printing. To get control over these parameters, use the `execute` statement with Panorama's `print` statement.

```
tell application "Panorama"
   activate
   execute "Openfile {Address List}
   openform {My Report}
   select Date>date({1/1/96})
   print dialog"
end tell
```

The `quit` statement shuts down Panorama. The `run` statement starts Panorama up if it is not running. However, this statement is not really very useful because Panorama will start up automatically any time a script asks it to do something by including the phrase tell application "Panorama" in the script.

The Core Suite

The Core Suite includes 9 additional statements you can use in your scripts.

The `close` statement closes a window. Unlike Panorama's `close` statement, the AppleScript `close` statement can close any window, not just the top window. For example, to close the 3rd window from the top, use this script:

```
tell application "Panorama"
   close window 3
end tell
```

The `count` statement counts windows or fields. Here is a simple script that counts the current number of open windows in Panorama.

```
tell application "Panorama"
   get Count windows
end tell
```

A slightly different format is required to count fields. This script counts the number of fields in the currently active database.

```
tell application "Panorama"
   get Count of every Cell
end tell
```

(Note: This `of every` format works with windows also, one of the many examples of redundancy in the AppleScript language. This redundancy is not consistent, however, and you cannot say get `Count cells` to find the number of fields in a database. Of course you could also create these examples with the `execute` statement, using Panorama itself to count the windows or fields.)

The data size statement can be used to get the size of the contents of a field or variable. For example, suppose the field Name in the current record contains John. In that case, the result of this script will be the number 4 (the number of characters in the name John).

```
tell application "Panorama"
   get Data Size of Value of Cell "Name"
end tell
```

The `do script` statement launches a Panorama procedure. The procedure must be pre-defined in the current database.

```
tell application "Panorama"
   do script "Year End Totals"
end tell
```

The `execute` statement lets you put a Panorama procedure inside an AppleScript. Unlike the `do script` statement, the `execute` statement requires no advance preparation in the database itself.

The `exists` statement can be used to determine if a field, variable, or window exists or not. This statement returns a true-false result, and is usually used with the `if` statement. This script checks to see if the current database has a field called Name. If it does contain such a field, the script converts the field to all upper case.

```
tell application "Panorama"
   if Exists of Cell "Name" then
      execute "field Name
      formulafill upper(Name)"
   end if
end tell
```

The `get` statement is the standard AppleScript get statement—it simply evaluates a value and puts it in the **Result** variable. In the script editor you can open the Result window to see the contents of this variable, which can be useful for debugging.

The `move` statement works with windows. Using this statement, you can change the order of the windows within Panorama. Here are some examples of how the move statement can be used.

```
Move Window 2 to beginning

Move Window 1 to end

Move Window 1 to after Window 2

Move Window 1 to before Window 5

Move Window 1 to back of Window 2
```

The `set` statement is the AppleScript equivalent of the assignment statement in most programming languages. For example, in most programming languages you would add two numbers like this:

```
Sum=3+4
```

But in AppleScript this assignment is written like this:

```
set Sum to 3+4
```

The `set` statement is one of the most frequently used statements in the AppleScript language.

The Objects

Panorama has three types of objects that you can use in your scripts: **application**, **window**, and **cell**.

The **application** object refers to Panorama itself. You cannot modify this object, but you can get useful information about it-the version, name, etc. This script prints the current database, but only if Panorama is the topmost application.

```
tell application "Panorama"
   if frontmost
      execute "print dialog"
   end if
end tell
```

(Note: One of the properties of an application is its version number. The dictionary says that this is a string. However it is not a string or a number, but a special class. There is not much you can do with this special version class. Unfortunately, this is consistent with other applications, including the Scriptable Finder.)

The window object refers to Panorama windows. A window may be identified by its name or by a number (with 1 being the topmost window). Here is a script that gets the name of the topmost window:

```
tell application "Panorama"
   get Name of Window 1
end tell
```

In addition to the name, there are many other properties of a window that you can access: the window location, its size, whether or not it has a close box or zoom box, and many more. See the Panorama AppleScript dictionary for a complete listing of window properties.

Some window properties can be changed from AppleScript with the `set` statement. For example, you can move a window to a new position using the **bounds** property.

```
tell application "Panorama"
   set Bounds of Window 1 to {50, 100, 400, 250}
end tell
```

You can change the order of windows with the `move` statement. This script moves the third window to the front.

```
tell application "Panorama"
   activate
   Move Window 3 to beginning
end tell
```

The `activate` statement is not actually necessary. It brings Panorama to the front, which makes it easier to see the windows change order.

The `move` statement is very flexible for changing the order of windows. Here are some more examples of possible options.

```
Move Window 2 to beginning

Move Window 1 to end

Move Window 1 to after Window 2

Move Window 1 to before Window 2

Move Window 1 to back of Window 2
```

The **cell** object type is used for working with Panorama fields and variables. The most commonly used property of cell objects is their value, as seen throughout this appendix (Value of cell "Name", etc.)

Another cell property is the index, or field number. (This property only applies to fields, not variables.) Here is an example that gets the field number (1, 2, 3, etc.) of the field City.

```
tell application "Panorama"
   get Index of Cell "City"
end tell
```

If the database contains fields called Name, Address, City, State and Zip then this script will return the value 3.

Another cell property is the cell Name. Again, this really only applies to fields, not variables. This script uses the Name property to build a list of all the fields in a database.

```
tell application "Panorama"
   set CellNames to {}
   repeat with cellnumber from 1 to Count of every Cell
      set CellNames to (CellNames & Name of Cell cellnumber)
   end repeat
   get CellNames
end tell
```

The script above will work fine and illustrates the Name property well, but by letting Panorama itself do some of the work we can create a script that runs much faster.

```
tell application "Panorama"
   Execute "global aString aString=dbinfo(“fields”,“”)"
   set databaseFields to Value of Cell "aString"
end tell
set AppleScript's text item delimiters to return
set databaseFields to every text item of databaseFields
get databaseFields
```

The dictionary lists several other properties of cell objects (Best Type, Class, Default Type) but these really aren’t of any use to AppleScript programmers (although they are used internally by AppleScript itself).

Additional Resources



Whew! This manual contains a wealth of material for learning and using Panorama. However, if you've gotten through all this material and are looking for more, good for you!

On-Line Resources

For the latest news about new versions of Panorama, accessory products, and announcements of upcoming events be sure to visit our web site frequently — `www.provue.com`.



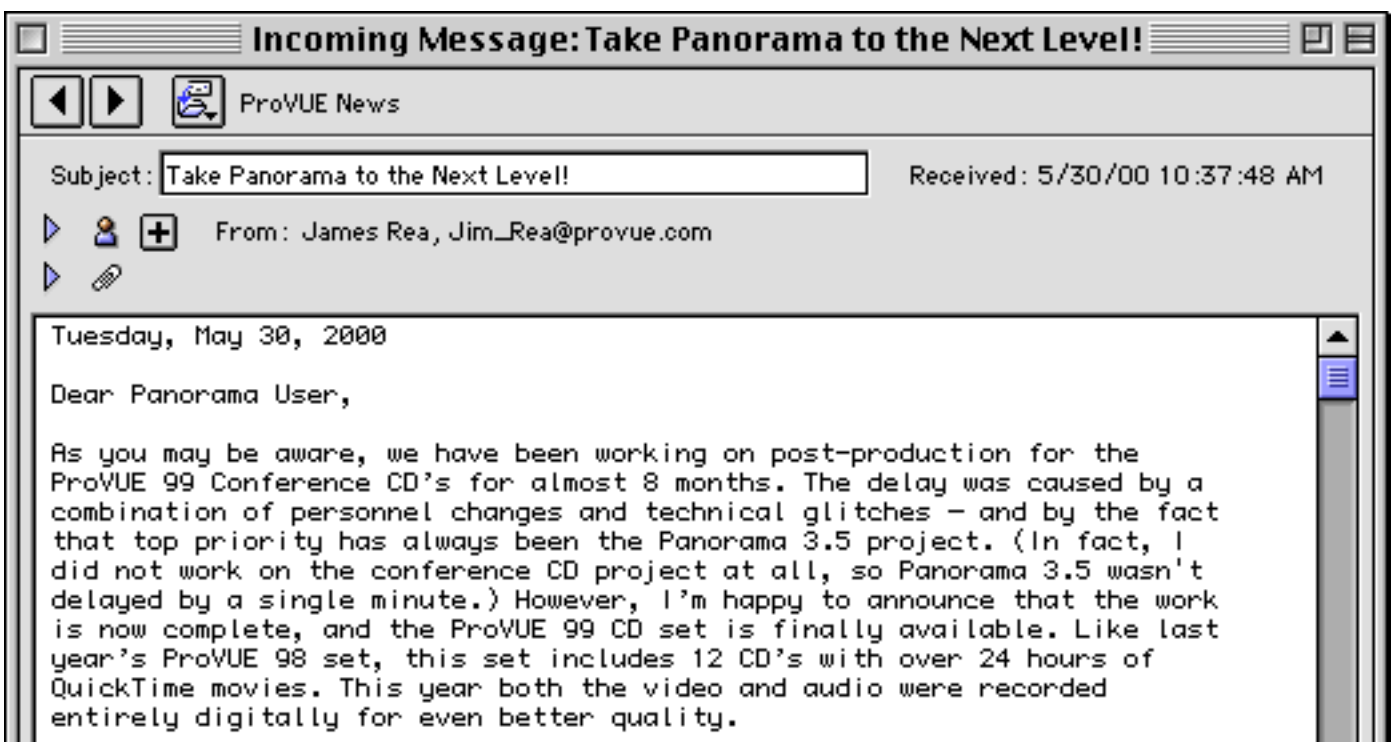
Signing Up For Panorama News Via E-Mail

Sign up to the ProVUE News e-mail list to receive the latest news and announcements from ProVUE, including new products, updates, special offers and upcoming events. (We promise not to send too many announcements, so don't worry about being flooded with email. Typically the average is between five and twenty messages per year.)

The easiest way to subscribe (or unsubscribe) to this list is to use the ProVUE web site. Simply type in your e-mail address, select Subscribe or Unsubscribe and then press the **Submit** button.

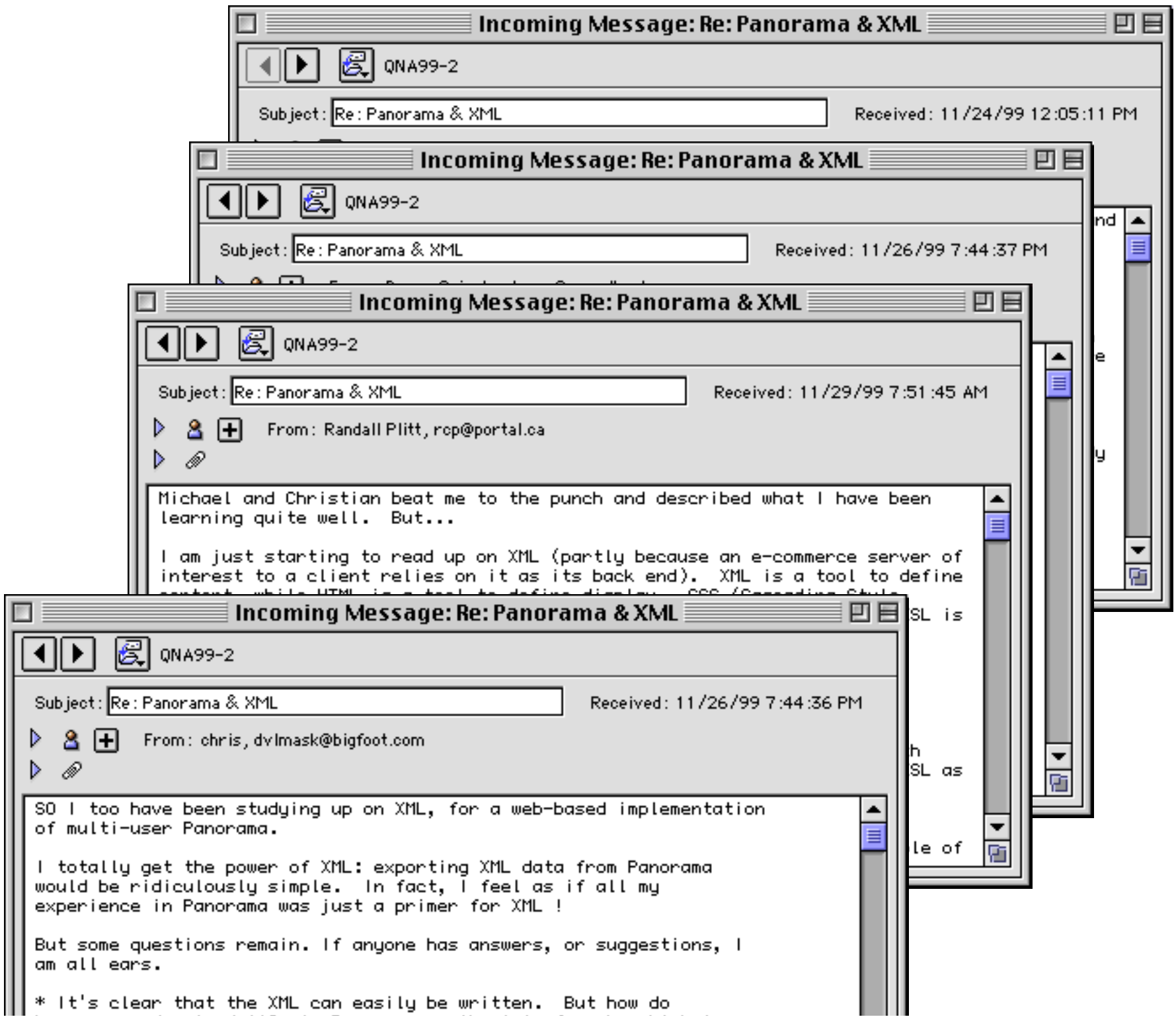


If you wish you can also subscribe by sending an e-mail message to `requests@lists.provue.com`. The body of the message should be either `subscribe news` or `unsubscribe news`.



Signing Up to Join Other Panorama Users On-Line (QNA List)

As a Panorama user you can become part of a vibrant on-line community of Panorama aficionados all around the world. The Panorama **QNA list** (Question and Answer) is an e-mail based community of Panorama users from beginners to experts. You can post questions, answer other users questions, or simply lurk and watch what's going on.



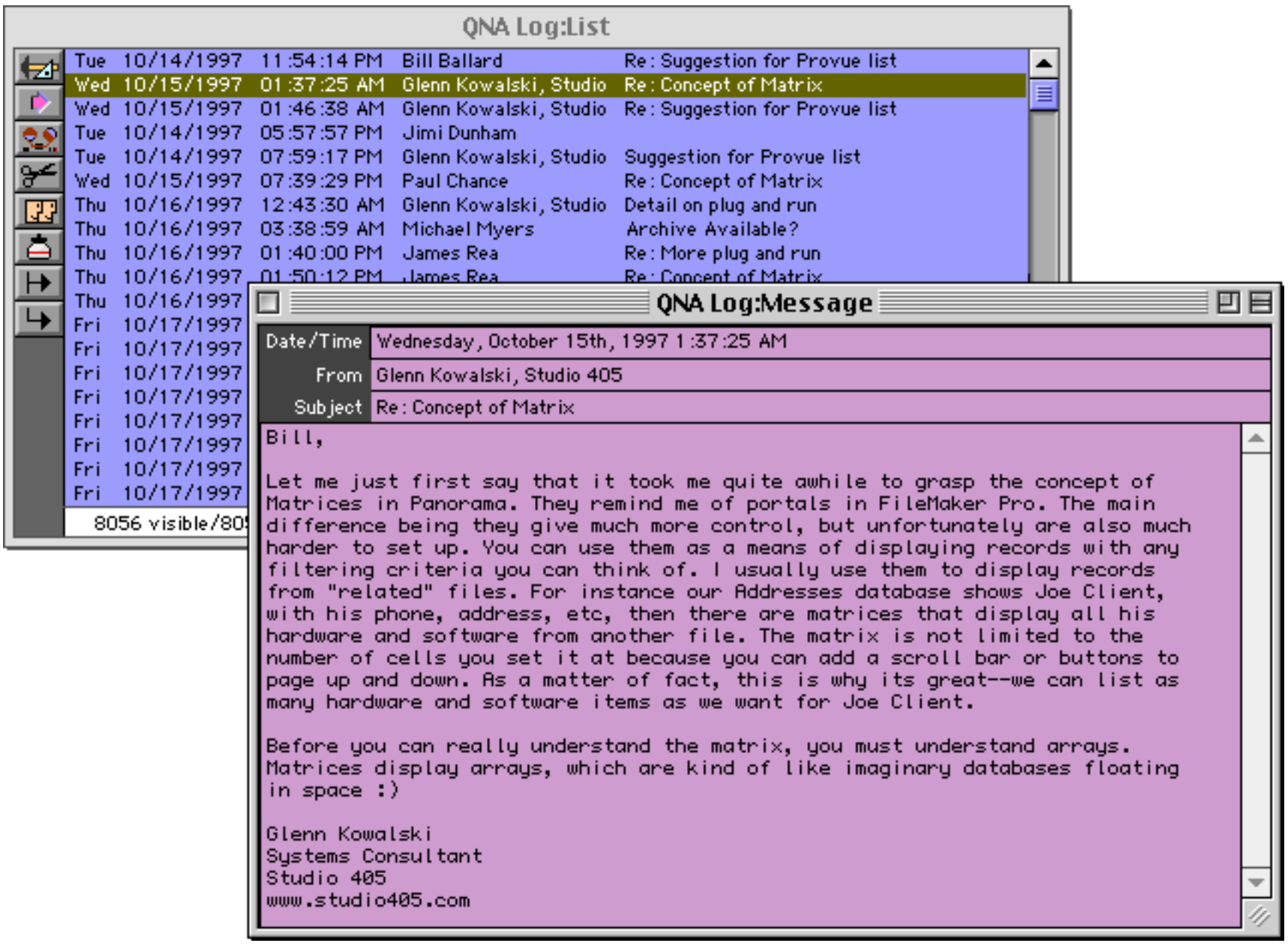
The easiest way to subscribe (or unsubscribe) to this list is to use the ProVUE web site. Simply type in your e-mail address, select Subscribe or Unsubscribe and then press the **Submit** button.You can easily unsubscribe at any time, and your address will not be transferred to any third party. If you wish you can also subscribe by sending an e-mail message to `requests@lists.provue.com`. The body of the message should be either `subscribe qna` or `unsubscribe qna`.

QNA Digest Mode

Traffic on the QNA list varies from a few messages a week to a few messages a day. When you subscribe in digest mode all of the day's messages will be combined and sent to you as a single e-mail. This limits the number of e-mails you receive to one per day, but can also make it a little more difficult to follow the threads in the list. You can subscribe to the digest using the web site or by sending an e-mail message with a body of either `subscribe qna digest` or `unsubscribe qna digest`.

QNA Log Database

The QNA Log Database archives several years worth of QNA messages from past years. You can use Panorama's searching tools to look for the information you need. A copy of this database is supplied on the Panorama CD and can be found using the Favorite Databases wizard.



From time to time we will release updated versions of this database on our web site.

Several Panorama users have created a searchable on-line archive of the QNA list, which can be found at:

`http://www.exegesis.com/panorama/index.html`

Many thanks to Alan Weiss, Chris Watts and Mick Matousek for setting up and maintaining this archive.

Technical Support

Technical support for ProVUE products is available online. Go to www.provue.com and click on the Support button. You'll find a link for submitting a technical support incident on that page.

Getting the Most from Technical Support

A little bit of preparation can make your technical support experience more productive. Before you submit an incident, try to make sure that you are clear exactly what your question is. Don't overload with unnecessary details, but be sure to supply enough information to describe the problem. Remember, our technical support staff cannot see your computer and has no idea of the context in which your problem is occuring. Try to narrow down the problem to the essential facts. For example, suppose you have a problem printing. Is it always a problem, or just with certain forms, or certain printers If only certain forms, is there anything that you can see in common between those forms? What if you preview the form, does it work then? The more you can narrow down the problem the faster we'll be able to resolve your problem.

If you are encountering an intermittent problem, try to isolate exactly what is causing the problem before contacting technical support. If you are using a Macintosh system it's always a good idea to try running your system with the minimum number of extensions. If possible, try rebooting your system with the **Shift** key held down to disable all extensions. This will often eliminate intermittent problems. Also, check that you have the latest version of Panorama and any other software (for example printer drivers) that you may be using.

Panorama Conferences

What do you dream you can do with Panorama? One way to help make those dreams come true is to come to Los Angeles for one of our ProVUE Conferences. Whether you're a novice or a ten year Panorama veteran, attending one of these conferences will help you get the most from your Panorama investment. You won't want to miss this golden opportunity to meet with ProVUE's expert staff, as well as network with other Panorama users and developers like yourself. Visit our web site for details on upcoming conference dates.



Can't make it to LA? No problem! We digitally record each conference and make them available as a set of twelve compact discs. That's right -- you can actually attend every conference session yourself right in the comfort of your own home or office!

Each CD-ROM contains two or three sessions in QuickTime movie format. It's just like having a front row seat to every session. Perhaps even better...since you can pause, back up, or even skip forward at any time. We've included all of the sample files that were used in each session, so you can even follow along with the instructor right on your own computer!



Whether you're just starting out or having been using ProVUE products since OverVUE days, these CD's will help you get the most from your Panorama investment. If you're the kind of person that has trouble learning from a manual, now you can actually see powerful Panorama techniques demonstrated right on your screen.

Don't confuse our digitally recorded QuickTime movies with fuzzy video taped presentations you may have seen in the past. We used the latest software technology to record directly from the screen to the hard disk. Each recording is pixel perfect -- you see the screen exactly as it appeared with no loss or distortion. The audio was recorded separately and added to the movie after the conference. It's never been so easy or fun to learn how to use Panorama more effectively!

ProVUE 98 Conference

This conference was held on August 31 thru September 1, 1998. The 27 sessions were divided up into five different tracks. (Each session is approximately one hour in length). See our web site or contact ProVUE for information on purchasing this 12 CD set.



Panorama Skills Track

These sessions will help you get the most from your Panorama investment by building basic and intermediate Panorama skills, including building reports, import/export, lookups, data posting, using the word processor, and getting started with Butler/SQL.

Session	Description
Introduction to Panorama 3.0	If you are one of the many Panorama users who are not taking advantage of some of the terrific new Panorama 3.0 and 3.1 features, this session is for you! We'll concentrate on the easy-to-use new form features as well as covering changes in the way formulas work.
Installing and Using Butler SQL	If you are considering moving up to Panorama's multi-user Partner/Server system, this session will show you the way. You'll see how to install the Butler server and client software, and then go through the steps of converting a single user Panorama database to use the SQL Partner/Server system.
Getting the Most from Panorama's Word Processor	Panorama 3.0 includes a full-featured word processor that can be included in any database. In this session you'll learn how easy it is to use this word processor for common applications like organizing correspondence and creating customized form letters.
Introduction to Procedures	Are you taking advantage of Panorama procedures to automate your work? It's easier than you may think. In this session you'll learn how to create a procedure, how to make decisions in a procedure, basic subroutines, data types, and more.
Data Lookup and Posting	Most significant database applications require lookup up and posting data from one database to another. In this session you'll learn how to lookup one or many records from another database (or even the same database!) using the lookup and lookupall statements. You'll also learn how to post data from one database to another using a simple procedure.
Report Tiles	Panorama's report tile system is extremely flexible for creating a wide variety of reports, labels and correspondence. This session will help you unleash the power of report tiles for your databases.
Advanced Printing	This session covers advanced printing techniques, including overflow tiles, multiple body tiles (including selectively printing multiple pages per record), groups and sidebars, double sided printing, printing for organizer notebooks, and using the printusing statement.
Panorama Import/Export and File I/O	Panorama excels at importing and exporting data. This session starts with basic import/export techniques, then moves up into more advanced techniques that allow you to re-arrange data as it is imported/exported. You'll learn how to import/export space delimited data.

Advanced Track

This track covers advanced topics, including credit card processing, the future of Panorama/SQL, and behind the scenes looks at several ProVUE programming projects.

Session	Description
ProVUE's In House Order Entry System	Of course ProVUE uses Panorama to do its own in-house order entry system. In this session we'll show you how we did it. The system includes unlimited line items (can you say SuperMatrix?), configurable product panels, and multiple price levels per product (for resellers, distributors, etc.).
Credit Card Processing with Panorama	Using Tellan's "MacAuthorize" credit card software you can automatically process credit card transactions from a Panorama database. In this session we'll show how to set up and program MacAuthorize from Panorama.
SQL Partner/Server Architecture and Future Directions	This session will describe the current internal operation of the Partner/Server system, and cover our future plans for SQL on both the Mac and PC platforms.
Customizing Power Team and SurfScout	This session will show you how to get at the insides of Power Team and SurfScout and make changes without breaking everything! If time allows, we'll also cover plans for future enhancements to Power Team.
Advanced SuperObjects	In this session we'll look at two of the most advanced SuperObjects - Lists, and Matrixes. You'll learn how to set up these SuperObjects for various applications, and how to modify them on-the-fly via procedures.
Future of Panorama	In this session we'll lay out some of our future plans and ideas for Panorama as it moves into its second decade, and we'll also give you the chance to submit your own ideas and suggestions.

Programming Track

This track covers intermediate and advanced programming topics. If you've taught yourself Panorama programming the hard way these sessions will help you step up to the next level.

Session	Description
Custom Menus, Dialogs, and Alerts	Learn how to create professional menus and dialogs in your Panorama applications. You'll learn how to use ResEdit and Resourcer to create custom menus, and how to assign custom menus to your windows. You'll also learn how to change custom menus on the fly. Panorama 3.1 allows you to use forms to create custom dialogs that look and behave just like regular dialogs. You'll learn how to create a default OK button, how to center a dialog, and how to create and use common dialog elements like text editing boxes, checkboxes, radio buttons, and scrolling lists.
Using AppleScript with Panorama	Using AppleScript a Panorama programmer can control many other software programs, including the Finder, web browsers (both Netscape Navigator and Microsoft Internet Explorer), email software and more. Other programs can also use AppleScript to control Panorama. In this session you'll learn how to create AppleScripts that can be used by Panorama to control other programs, including passing data back and forth between Panorama and other software. We'll also dissect some of the mysteries of using the AppleScript dictionaries for learning how to script other software packages.
Cool Panorama Programming Tricks	In this session we'll cover some of the cool programming tricks that can save you time, and in some cases make what seem to be impossible programming problems easily solvable. We'll cover neat things you can do with arrays, some really cool things you can do with the execute statement, show you how to use secret windows and more!
Writing your own XCALLS using Metrowerks C	Did you know that Panorama can be extended with your own custom routines written in C or Pascal? In this session we will walk you through the creation of a simple C module that can be accessed from any Panorama procedure.

Internet Track

This track mostly focuses on efficient techniques for creating both static and dynamic web content thru Panorama, SiteWarrior, and JavaScript. The internet is more than the web though, so there's also a very cool e-mail session.

Session	Description
On Line Database Publishing with Panorama	Using the Panorama CGI you can publish Panorama databases on the web. Users can add and update data over the web, as well as search databases on line. In this session we'll actually set up a web server and show how to program your databases for the web.
Using Panorama to Process and Generate Email	Panorama can be an excellent program for working with email. In this session you'll learn how to reliably transfer data with email, how to generate custom mass email automatically, and how to automatically read email from your mail server. You can even use Panorama to create your own customizable e-mail client!
Database Publishing with SiteWarrior	One of SiteWarrior's primary strengths is the ability to take any database and convert it into finished HTML pages that are ready to upload to your web server. In this session you'll learn how to design automatically generated HTML database reports as well as automatically generated sets of pages (for example product catalogs, real estate listings or classified ads).
Using SiteWarrior to Build Large Web Sites	SiteWarrior's "HTML Factories" make creating and maintaining large web sites much easier and faster. In this session you'll learn how to automatically create a table of contents both within a page and across a site. The session will also show how to automatically link a page with its neighbors (previous page and next page), how to create entire pages (or subsections) from a template, and how to create easily modifiable "navigation banners" across the top or down the side of a page.
Beginning HTML (with SiteWarrior)	HTML (Hyper Text Markup Language) is the language of the web. At first glance it looks complicated, but there are really only about a dozen simple tags you need to learn. At the end of this session you'll be ready to start creating your own web pages directly in HTML. (Knowledge of HTML is also a must if you want to get into publishing your own databases on the web.)
Introduction to JavaScript	JavaScript is to web browsers what AppleScript is to the Mac. Using JavaScript you can do client-side processing like verifying data entry, looking up prices, and performing calculations-all operations that are especially useful when publishing databases on the web. This session will introduce the basics of JavaScript for use with web-based forms.

Cross Platform Track

The big news for 1998 was that Panorama was going cross-platform and native.

Session	Description
Taking Panorama Cross Platform	The Windows version of Panorama is coming soon! In this session we'll cover the issues we've discovered in making the switch, including techniques for copying files between platforms, preparing resource files for use with Windows, font and character set issues, memory allocation and more.
Introduction to Windows	If you're a long time Mac user, you may be dreading the switch to Windows. This session will take away some of the sting for first-time Windows users. The session will cover use of the two-button mouse (context menus), the Start menu, disk organization, and using the network . We've also collected a set of handy shareware and freeware utilities that will help you keep your Windows system organized, and we'll show how these utilities are installed and used.

ProVUE 99 Conference

This conference was held on August 30-31 1999. The sessions were divided up into four different tracks. (Each session is approximately one hour in length). See our web site or contact ProVUE for information on purchasing this 12 CD set.



Web Track

ProVUE 99 included eight full hours of instruction on connecting Panorama to the Web. If you're already a webmaster you'll be putting your databases on-line in no time. If your a web newbie don't worry, we'll start you with the basics of HTML and web graphics so that you can work your way up.

Session	Description
Panorama CGI Introduction	This is the first of four sessions devoted to connecting Panorama to the web. This session covers the basics, starting with setting up and configuring your server and the Panorama CGI. Next, we'll cover using the Panorama CGI Test Jig to build and test your CGI procedures "off-line" so that they'll work the first time when you install them live on your server. Finally, we'll create a simple status CGI that displays a list of the Panorama files currently open on the server.
Web Forms	Part two of our web server curriculum covers creating and processing forms. You'll learn how to create forms in HTML, and how to connect those forms to your Panorama databases for data entry or queries.
JavaScript	Did you know that your web browser has a programming language built into it? Both Navigator and Explorer can be programmed with JavaScript, a "C like" language. JavaScript can work together with Panorama on the server to make a faster, easier to use web site. In this session we'll introduce the basic's of programming with JavaScript.
Building the ProVUE Shopping Cart	This session puts it all together. We'll show you how we used Panorama on the server and JavaScript on the client to build what we think is a very cool shopping cart. The conference CD-ROM includes complete source code for this shopping cart system.
Basic HTML	HTML is the language of the web -- do you speak it? At first glance it looks complicated, but there are really only about a dozen simple tags you need to learn. At the end of this session you'll be ready to start creating your own web pages directly in HTML. (Knowledge of HTML is also a must if you want to get into publishing your own databases on the web.)
HTML Tables	The table tag is HTML's most powerful and least understood formatting tool, and not just for creating tables. In this session you'll learn how to use the table tag both to build tables and for general page formatting.
Preparing Images for the Web	The web isn't just about text -- it's also about images. This session covers tools for creating GIF, JPEG and animated GIF images, and shows how to use these images in your HTML pages.
Generating HTML with the Word Processor	In this session we'll build a procedure that automatically converts the text in a word processor into HTML, including converting attributes like bold, italic, font and color into the appropriate HTML tags.

Basic Skills Track

If you are just getting started with Panorama these sessions will help you get productive in a hurry. Even if you have been using Panorama for a while you may find that some of this material is a useful refresher.

Session	Description
Basic Panorama Refresher	If you are new to Panorama, or just need a refresher in the basics, come on down! Even old Panorama hands may discover a trick or two they didn't know. We'll cover the view menu, data sheet, design sheet, data entry options, sorting, selecting, totals, propagating, and other basics.
Forms & Graphic Tools	If you are only using the data sheet, you're not tapping the full power of Panorama. We'll show you how easy it is to create forms to display, edit and print data. Topics covered will include creating forms, creating shapes, using the graphic control strip, grouping objects, cluster resize, alignment, spacing, and auto-cell layout.
Working with Text and Images	In this session you'll learn how to use forms to display and edit text, and to display images. For handling text we'll cover the text editor, text display, and word processor SuperObjects™. We'll also demystify the secrets of displaying images with Flash Art™.
Basic Programming Part 1	If you think programming is too advanced for you, check out these two sessions! Even if you've never done it before, you can learn how to program in Panorama. Even simple programming skills can often save you hours of repetitive hand work. We'll start with the basics - creating a program, making decisions, repeating a task (loops).
Basic Programming Part 2	Our introduction to programming continues with an introduction to variables, and a review of programming techniques for controlling Panorama itself (sorting, selecting, etc.) We'll also cover techniques for finding and eliminating the dreaded "type mismatch" error.

Intermediate Skills Track

If you have been using Panorama for a while and are ready to move on to the next level then these sessions are for you.

Session	Description
Building Cross Platform Databases	In this session you'll learn how to build databases that work well on both Macintosh and Windows platforms. Topics covered include basic cross platform issues, images, file handling, resources, and fonts.
Calendars and Date / Time Processing	Panorama has powerful features for building calendars and working with date and time information. In this session we'll build a simple monthly calendar and a time card application.
Partner / Server Programming	The Partner / Server system allows a Panorama database to be used in a multi-user environment. When you're sharing your database with others, you have to learn a few new rules. We'll show you how to get the job done, including a discussion of record locking and techniques for improving the performance of Partner / Server databases.
Cataloging Images	Panorama makes an excellent tool for cataloging and organizing images, and the new ability to display TIFF and JPEG images completes this capability. In this session you'll learn strategies for displaying images (lists, matrix, etc.,), techniques for determining image size, finding image files on the disk, and more.
Making Money with Panorama	Do you want to use your Panorama skills to make money? Learn how other Panorama developers have done it in this roundtable discussion.
Automatic Procedures	Did you know that Panorama can automatically perform a procedure whenever you open a file, add or delete a record, close a window, or even resize a form? Learn how to set up and use these "automatic procedures" in this session.

Session	Description
Formatted Export with Panorama	Need to convert a database into a Quark publication? An HTML document? Some other special format? The entire process can be automated by using Panorama functions to export text with embedded tags, often saving hours or even days of work.
Info Functions	Panorama has literally dozens of "info" functions that allow you to find out information about your database, your computer, and Panorama itself. Learn more about these functions and how they can be used.

Advanced Skills Track

Are you a Panorama thunder lizard? These sessions will be especially helpful if you are planning on creating Panorama databases for distribution and sale.

Session	Description
Windows and Secret Windows	This session is really two parts. In part one, you'll learn how to control window appearance and location to get just the "look and feel" you want for your database. In part two, you'll learn how to eliminate windows altogether - using "secret windows" to eliminate unnecessary "flashing" as Panorama switches between databases, and to allow access to hidden databases (rate tables, etc.).
Building Installers with Panorama (also Panorama Engine)	Are you distributing applications written in Panorama? You can use Panorama itself to install your software, including placing files in the correct folders, setting up resources, and modifying the registry (Windows). You get total control over the installation process, and don't need to pay for a 3rd party installation program!
Windows Registry	The registry is the secret control panel of Window 95 and NT, and is used by both Windows itself and virtually all applications. You'll learn about how the registry is structured and used, and how to access and modify the registry from within Panorama.
Advanced Programming Techniques	The Execute statement is one of the most powerful tools in a Panorama programmer's arsenal. It allows you to build program segments "on-the-fly" to rapidly complete tasks that otherwise would be impossible or require slow loops. This session will also cover other advanced programming topics, including arrays and techniques for parsing structured data (for example HTML).

Publications

In addition to this manual there are several other ProVUE publications that contain information helpful to Panorama users and developers. See our web site or contact ProVUE for information on purchasing these publications.

Panorama Real World Programming Guide

Panorama has over 350 different statements, over 200 functions, and almost two dozen types of graphical objects to choose from when building an application. The **Panorama 3 Real World Programming Guide** picks up where the Panorama Handbook leaves off and shows how these hundreds of different elements work together to create a professional quality database. You'll get a solid foundation in Panorama programming fundamentals and clear-cut directions on how to handle dozens of specific real-world situations like: lookup up and posting data to multiple databases, trapping illegal data entry, creating and using menus, displaying negative numbers in red, extracting and converting data, calculating date and / or time intervals, synchronizing multiple databases, accessing data created in other programs, centering a window on the screen, line item import / export / analysis, combining numbers with text, building and displaying a list of items in a field, drag and drop and much more. The Guide is filled with literally hundreds of program examples. Whether you are a Panorama beginner or a seasoned veteran, this 200 page book will give you a solid foundation in Panorama programming fundamentals and clear-cut directions on how to handle dozens of specific real-world situations.



Panorama Security Handbook

The **Panorama 3 Database Security Supplement** describes how to safeguard your Panorama databases from unauthorized access and modification. The 40 page supplement describes how security can be built in for a single computer or across an entire network.

Panorama Partner/Server Handbook

The **Panorama 3 Partner/Server Handbook** describes how to create and use multi-user databases. This handbook is not available separately but is provided when you purchase the Butler server.

Consulting Services

Do you need someone to design or support a Panorama database application for you? Although ProVUE itself does not provide consulting services at this time, we can provide you with referrals to consultants in your area. Contact the ProVUE technical support staff for further information.