

Panorama Enterprise Server

Panorama Enterprise Server Handbook (Version 5.5)
Copyright © 2008, ProVUE Development,
All Rights Reserved

ProVUE Development
18685-A Main Street PMB 356
Huntington Beach, CA 92648
USA

www.provue.com



Table of Contents



— Click on any entry to jump to the page —

| | |
|---|-----------|
| System Requirements (Server)..... | 13 |
| System Requirements (Client) | 13 |
| System Configuration (Local Database Sharing)..... | 14 |
| System Preparation (Database Web Publishing)..... | 14 |
| System Preparation (Internet Database Sharing) | 14 |
| Converting a Single User Database into a Shared Multi-User Database | 15 |
| Shared vs. Single User Database Operation | 15 |
| Converting a Shared Multi-User Database Back to Single User..... | 15 |
| Removing a Database from the Server..... | 15 |
| Introduction | 17 |
| Types of Networks | 17 |
| Client/Server Modes | 17 |
| Database Sharing Concepts and Operation | 18 |
| High Performance..... | 19 |
| Low Network/Server Load | 19 |
| Offline Operation..... | 19 |
| High Safety Margin | 19 |
| Zero-Loss Interruption Recovery | 19 |
| Automatic Live Backups | 19 |
| Detailed Logging and E-mail Notification | 19 |
| Rebuild from any client..... | 20 |
| No delicate index to get corrupted..... | 20 |
| How Distributed Data Sharing Works | 20 |
| Opening a Database | 21 |
| The Synchronization Process..... | 22 |
| A Note about Time Stamps | 22 |
| Editing and Record Locking | 22 |
| Modifying Data Offline | 27 |
| Database Web Publishing Concepts and Operation..... | 28 |
| Panorama Forms on the Web..... | 28 |
| Database Tables on the Web | 29 |
| Custom Web Programming | 29 |

| | |
|---|----|
| Installation, Configuration & Management | 31 |
| System Requirements..... | 31 |
| System Preparation | 32 |
| Setting up the Computer Name | 32 |
| Enable Remote Apple Events..... | 33 |
| Setting Up The User Account | 33 |
| A Brief Introduction to Static IP Addresses..... | 34 |
| Setting up a Server Directly Connected to the Internet | 34 |
| Setting up a Server Connected to the Internet using a Router..... | 37 |
| Enabling Internet Sharing | 38 |
| Ensuring Nonstop Server Operation..... | 39 |
| Enabling Remote Server Operation..... | 40 |
| Installing the Panorama Server Software | 43 |
| Launching the Server..... | 44 |
| Shutting Down the Server..... | 45 |
| Shutting Down the Server with Open Sharing Sessions | 46 |
| Setting up Panorama Server to Launch Automatically | 47 |
| Panorama Server Product Activation..... | 48 |
| Panorama Server Licensing Options..... | 48 |
| Panorama Client Licensing Options | 48 |
| Panorama Server Demo Mode..... | 48 |
| Basic Server Configuration | 49 |
| Unlocking the Server Configuration | 49 |
| Changing the Server Password..... | 51 |
| Remembering the Password | 51 |
| Changing the Auxiliary Passwords | 52 |
| Changing the Server Name | 53 |
| Local Network vs. Internet Settings | 54 |
| Enabling Database Sharing | 55 |
| Enabling Database Sharing over the Internet | 55 |
| Enabling Database Web Publishing | 56 |
| Setting the IP Address/Domain Name..... | 56 |
| Locking the Server Configuration | 57 |
| Client Configuration | 58 |
| Configuring Local Database Sharing | 58 |
| Configuring Local Database Sharing to use TCP/IP instead of Remote Apple Events | 59 |
| Configuring Bonjour..... | 60 |
| Bonjour Threshold | 60 |
| Locking the Bonjour Configuration | 60 |
| Debugging Local Database Sharing Connection Problems | 61 |
| Configuring Remote (Internet) Database Sharing | 62 |
| Further Testing the Sharing Connection..... | 63 |
| Testing Web Database Publishing (Server Status) | 64 |
| Restricting Access to the Server Status and Database Web Link Pages | 65 |
| Server Management (The Server Administration Wizard) | 68 |
| Choosing the Server to Manage..... | 69 |
| Updating the Snapshot..... | 69 |
| Server Snapshot Information..... | 69 |
| Server Database List | 70 |
| Listing Database Users | 71 |
| Forcing a Database to Close..... | 71 |
| Listing Locked Records | 72 |
| The Pop-up Database Context Menu..... | 72 |

| | |
|---|------------|
| Database Online/Database Offline..... | 73 |
| Display Users | 73 |
| Delete From Server..... | 73 |
| Browse Database Web Links | 73 |
| Close Database..... | 74 |
| Adjusting the Table Size..... | 75 |
| Disabled Sharing/Web Publishing Warning..... | 76 |
| Server Memory Usage..... | 77 |
| Adjusting Panorama's Memory Allocation | 77 |
| Active Session List..... | 78 |
| Listing Open Databases | 79 |
| Forcing a Session to Close | 79 |
| Adjusting the Table Size..... | 79 |
| Memorizing Server Passwords..... | 80 |
| Changing the Auxiliary Passwords | 80 |
| Monitoring Server Logs..... | 82 |
| Viewing a Log..... | 82 |
| Searching the Log | 83 |
| Viewing a Different Log | 84 |
| Refreshing the Log..... | 84 |
| Server Log Configuration | 85 |
| Configuring the Notification Wizard..... | 85 |
| When does the Notification Wizard appear? | 86 |
| Using Growl for Notifications | 87 |
| Using the Notification Wizard in your own database applications | 88 |
| Advanced Server Configuration | 89 |
| Notification Options..... | 90 |
| Configuring an Email Channel..... | 90 |
| Backup Options | 91 |
| Advanced Options | 92 |
| Automaticaly Unlock Records | 92 |
| Rebound After Server Crash | 92 |
| Ask to Confirm Before Quitting Server | 93 |
| Automatically Hide Server When Launched..... | 93 |
| Enable Server Activity Indicator | 94 |
| Internal/Debug Options..... | 95 |
| Editing the Server Configuration Text File (For Experts Only)..... | 96 |
| Editing the ServerConfig.dat File Directly on the Server | 97 |
| Server Configuration Tags | 98 |
| Online Database Sharing..... | 103 |
| Creating a Shared Database | 103 |
| Duplicate Database Conflicts on the Server | 108 |
| Transferring the Database to Other Client Computers | 110 |
| Using the Download Shared Databases Wizard | 110 |
| Downloading Offline Databases..... | 112 |
| Downloading with the Server Administration wizard. | 113 |
| Transferring the Database Manually | 114 |
| Opening a Shared Databases..... | 115 |
| Debugging Local Database Sharing Connection Problems | 115 |
| Opening a Shared Database on the Internet..... | 116 |
| Connecting an Already Open Database to the Server | 117 |
| Disconnecting from a Server | 118 |

| | |
|---|-----|
| Changing the Design of a Shared Database | 119 |
| Sharing “Generations” | 119 |
| Starting a New Sharing Generation | 119 |
| Adding and/or Removing Fields..... | 121 |
| Uploading the New Sharing Generation | 122 |
| Distributing the New Database Generation to All Clients | 122 |
| Design Changes to Forms (Graphics) and/or Procedures (Programming)..... | 123 |
| New Sharing Generation Options and Advanced Topics | 125 |
| Synchronization vs. Force Sync | 125 |
| Removing Sharing History..... | 125 |
| Making a New Sharing Generation Manually | 128 |
| Take the Database Offline | 128 |
| Synchronize/Force Sync | 128 |
| Download Server Variables..... | 128 |
| Convert Local Copy of Database to Single User..... | 128 |
| Make Design Changes..... | 128 |
| Remove Sharing History (Optional) | 129 |
| Re-Share & Re-Upload Database..... | 129 |
| Distributing the New Shared Generation of the Database | 129 |
| “Unsharing” a Shared Database | 130 |
| Forcing a Shared Database Back to Single User | 132 |
| Permanently Deleting a Database from the Server | 133 |
| Shared vs. Single User Database Operation | 135 |
| Editing Data and Record Locking | 136 |
| Record Lock Timeout (Client)..... | 139 |
| Changing the Record Lock Timeout..... | 140 |
| Record Lock Timeout (Server)..... | 142 |
| Locked Records Wizard..... | 142 |
| Finding a Locked Record | 142 |
| Manually Unlocking a Record..... | 143 |
| Manually Unlocking All Records in a Database..... | 143 |
| Synchronization | 144 |
| Synchronization and Record Order | 145 |
| Regular Synchronization vs. Force Synchronization | 147 |
| Adding New Records | 147 |
| Deleting Records | 147 |
| Working With Summary Records..... | 147 |
| Toggling Summary Records (not!)..... | 148 |
| Automatic Record Numbering..... | 148 |
| Manually Changing the Record Number Counter..... | 151 |
| Accessing the Next Record Number in a Procedure..... | 152 |
| Saving the Database..... | 152 |
| Revert to Saved..... | 152 |
| When does the Server Save?..... | 152 |
| Handling Interruptions in Server Operation (Crash Recovery) | 153 |
| Programming Shared Databases..... | 154 |
| Record Locking..... | 154 |
| Implicit Record Locking | 154 |
| Explicitly Locking/Unlocking Records in a Procedure | 154 |
| What Records are Locked? | 155 |
| Forcing the Server to Unlock All Records | 156 |
| Forcing the Server to Unlock a Specific Record..... | 156 |
| Forcing the Server to Lock a Specific Record | 156 |

| | |
|---|------------|
| Temporarily Disabling Record Locking (and Server Updates)..... | 157 |
| The info("serverupdate") Function..... | 157 |
| Synchronizing | 158 |
| Force Synchronization..... | 158 |
| Automatic Pre and Post Synchronization Procedures..... | 158 |
| Server Variables (Shared Variables) | 159 |
| Accessing Server Variables | 160 |
| Adjusting a Server Variable (Atomic Calculation)..... | 160 |
| Maintaining Server Variables when a database is Re-Shared | 161 |
| Data Transformations | 162 |
| Minimizing the amount of data changed by Fill commands..... | 162 |
| ServerFormulaFill — A Much Faster Option for Select/Formula Fill Operation..... | 163 |
| ServerFormulaFill Formula Restrictions | 164 |
| Record Locking and the ServerFormulaFill Statement..... | 165 |
| Minimizing the amount of data changed by ServerFormulaFill commands..... | 165 |
| Looking Up Data From Another Database..... | 166 |
| Temporarily Disabling Direct Lookups from the Server | 166 |
| Controlling and Monitoring the Server Connection | 167 |
| ConnectToServer Statement..... | 167 |
| DropServer Statement..... | 167 |
| The info("serverconnection") Function | 167 |
| The sharedusers(Function | 167 |
| The servername(Function | 168 |
| The serverdatabasename(Function | 168 |
| Shared Database Configuration | 168 |
| The EESetDBConfig Statement | 168 |
| The EEGetDBConfig Statement..... | 169 |
| The Configuration Dictionary..... | 169 |
| Deleting a Database from the Server | 171 |
| Offline Database Sharing | 173 |
| Offline Sharing Options..... | 173 |
| Is Offline Modification of Existing Records Appropriate for your Application? | 174 |
| Is Adding New Records While Offline Appropriate for your Application? | 175 |
| Deleting Records While Offline | 175 |
| Two Way Synchronization | 176 |
| Managing Synchronization Conflicts..... | 176 |
| Reviewing Conflicts in Large Fields | 183 |
| Overriding by Record (instead of Field)..... | 183 |
| Making and Dropping Server Connections | 184 |
| Designing a Database Primarily for Offline Operation | 189 |
| Configuring a Client Database for Primarily Offline Operation | 189 |
| Automatically Connect..... | 189 |
| Automatically Synchronize | 189 |
| Offline changes synchronized with server later | 190 |
| Offline changes are local only | 190 |
| Allow deleting records | 190 |
| Mixing Offline and Online Clients..... | 191 |
| Simulating Offline Operation (for testing) | 192 |
| Web Publishing | 193 |
| Web Publishing a Database..... | 193 |
| Preparing the Web Layout and Logic | 194 |

| | |
|---|------------|
| Uploading the Database to the Server..... | 194 |
| Database Sharing and Web Publishing..... | 195 |
| Web Publishing Options | 196 |
| Open automatically when server starts up | 196 |
| Automatically open as needed | 196 |
| Close database after each access | 196 |
| Automatically save database after each access | 196 |
| Use secret windows | 196 |
| Server Database Name | 197 |
| Uploading the Database to the Server | 197 |
| Testing a Web Database | 199 |
| The Server Status Page | 200 |
| The Database Web Links Page..... | 202 |
| Testing a Web Form..... | 204 |
| Testing a Web Procedure..... | 206 |
| Debugging Web Link Page Problems | 207 |
| Disabling the Web Link Page | 207 |
| Modifying a Web Published Database | 208 |
| Updating a Web Form/Adding a new Web Form | 208 |
| Updating All Web Forms..... | 209 |
| Updating a Table/Adding a new Web Table Template | 210 |
| Updating a Procedure/Adding a new Procedure | 211 |
| Uploading Multiple Procedures..... | 211 |
| Synchronizing data between the original copy and the server | 212 |
| Downloading Data from the Server | 212 |
| Uploading Data to the Server | 212 |
| Removing a Database from the Server | 212 |
| Associating a Database with Multiple Servers (Clones) | 214 |
| Creating a Clone | 214 |
| Updating a Clone Database's Procedures and Forms | 216 |
| Changing the Primary Server | 217 |
| Distributing Shared Clients for Clone Servers..... | 217 |
| Designing Your Web Database Application | 220 |
| Web Database URL Format | 220 |
| Standard Actions | 222 |
| Custom Actions..... | 222 |
| Action Sequences..... | 222 |
| Standard Data Entry Sequence..... | 222 |
| Search —> List —> Detail Sequence | 226 |
| Web Forms..... | 231 |
| Converting a Panorama Form into a Web Form | 231 |
| Converting a Panorama Form into a Web Page Form | 232 |
| The Web Form Converter Wizard | 235 |
| Preparing a Form..... | 236 |
| Conversion Limitations | 236 |
| Fields and Variables in Web Forms | 238 |
| Data Cells and Text Editor SuperObjects | 238 |
| Data Buttons (Checkboxes and Radio Buttons)..... | 238 |
| Pop-Up Menus | 239 |
| Lists..... | 239 |
| Push Buttons..... | 240 |
| Form Actions and Sequence | 241 |

| | |
|---|------------|
| Standard Form Action — FORMDUMP | 242 |
| Standard Form Action — NEWRECORD | 243 |
| Setting Up a Custom Response Page | 243 |
| Checking for Required Fields/Preventing Missing Fields | 245 |
| Standard Form Action — QUERY | 249 |
| Searching All Fields | 251 |
| Searching Multiple Fields instead of All Fields | 252 |
| Handling Failed Searches | 253 |
| Standard Form Action — UPDATERECORD | 253 |
| Custom Form Actions | 254 |
| Advanced Form Techniques | 255 |
| Font selection | 255 |
| Embedding HTML in a Text Display SuperObject | 256 |
| Linking to Other Web Pages | 259 |
| Linking to Blank Panorama Forms (from a Panorama Form) | 260 |
| Linking to Blank Panorama Forms (from a standard web page) | 261 |
| Linking to Panorama Procedures (from a Panorama Form) | 261 |
| Linking to Panorama Procedures (from a standard web page) | 262 |
| Linking to a JavaScript Script | 263 |
| Displaying Images in a Web Form | 264 |
| Displaying Images Based on a Field or Variable | 266 |
| Making an Image Link to Another Page | 267 |
| Making an Image a Submit Button | 268 |
| Making an Image a JavaScript Button | 269 |
| Triggering JavaScript with a Button | 270 |
| Hidden Data | 271 |
| Customizing the form HTML (Advanced) | 273 |
| Customize Page Dialog | 273 |
| Web Page Title | 273 |
| Form Tag Parameters | 273 |
| Form Prefix | 273 |
| Form Suffix | 273 |
| Style Tag Parameters | 274 |
| Style Prefix | 274 |
| Style Suffix | 274 |
| Page Template Dialog | 274 |
| Changing the Page Background Color | 275 |
| Adding JavaScript to a Page | 275 |
| Building a form in an external program | 276 |
| Web Tables | 277 |
| Web Tables | 277 |
| Text Export Wizard Refresher | 277 |
| Creating a Template | 278 |
| Using an Existing Template | 279 |
| Configuring the Table Columns (Title, Formula, Width and Alignment) | 280 |
| Customizing the Table Appearance | 283 |
| Web Page Title | 284 |
| Table Header Form | 285 |
| Table Margins | 286 |
| Table Border | 287 |
| Cell Spacing | 288 |
| Cell Padding | 288 |

| | |
|---|------------|
| Table Font | 289 |
| Text Size | 290 |
| Text and Background Colors | 291 |
| Color Selection Techniques | 291 |
| Main Text Color | 294 |
| Page Background Color | 295 |
| Title Color | 295 |
| Title Background Color | 296 |
| Table Data Color | 296 |
| Row Background Color | 297 |
| Multiple Background Colors | 297 |
| Linking a Table with a Query Form | 299 |
| Uploading a Table to the Server | 300 |
| Testing the Query and Table | 301 |
| Splitting a Long Table into Multiple Pages | 303 |
| The Multiple Page Table Dialog | 304 |
| Records per Page | 304 |
| Page Navigation Font | 305 |
| Text Size | 305 |
| Previous and Next Page Caption | 306 |
| Page Navigation Header and Footer | 306 |
| Linking Individual Table Rows to a Detail Form | 307 |
| The Individual Page Linking & Sorting dialog | 308 |
| Link Action | 308 |
| Link Table Column | 309 |
| Database Link Fields | 310 |
| Broken Links | 311 |
| Sort by | 312 |
| Editing/Updating a Record | 313 |
| Preparing a Database Update Form | 314 |
| Broken Record Identification | 314 |
| Customizing the table HTML (advanced) | 315 |
| Web Programming 101 | 317 |
| Creating a Simple Guestbook Web Database | 317 |
| Creating Web Procedures | 318 |
| Testing a Procedure in Advance | 318 |
| Create a Web Form | 319 |
| Assigning a Procedure to the Form | 320 |
| Testing the Form and Procedure using Simulation | 321 |
| Uploading the Guestbook Database | 323 |
| Testing Web Procedures on the Server | 324 |
| Modifying a Web Procedure | 325 |
| Testing a Procedure Assigned to a Web Form | 325 |
| Web Procedure Inputs and Outputs | 328 |
| Web Procedure URLs | 329 |
| Upper and Lower Case Characters in Procedure Names and Extra Parameters | 330 |
| Database and Procedure Names Containing Spaces | 330 |
| Generating a Web Procedure URL Without Typing | 330 |
| Run Web Procedure in the Debug menu | 330 |
| Server Status Page | 331 |
| Linking to a Web Procedure from Regular Web Pages | 331 |
| Linking to a Web Procedure from another Web Procedure | 331 |

| | |
|--|------------|
| Linking to a Web Procedure from a Form | 331 |
| URL Extra Parameters | 332 |
| Testing Procedures with Extra URL Parameters | 333 |
| HTTP Request Information | 336 |
| Form Input Data | 337 |
| Accessing Form Item Values in a Formula | 337 |
| Assigning a Form Item Value into a Field or Variable | 338 |
| Getting a List of Form Item Names | 338 |
| What Form Is This? | 338 |
| Hidden Form Items | 339 |
| Processing Hidden Form Items in a Procedure | 339 |
| Cookies | 339 |
| Generating a Web Page | 340 |
| The cgiHTML Global Variable | 340 |
| HTML (HyperText Markup Language) | 341 |
| JavaScript and CSS | 341 |
| Web Procedure Errors | 342 |
| Cookie Output | 344 |
| Getting Procedures onto the Server | 345 |
| Uploading All Procedures when the Database is Uploaded | 345 |
| Uploading All Procedures via a New Sharing Generation | 345 |
| Updating a Single Procedure/Adding a new Procedure | 345 |
| Uploading and Testing a Web Procedure | 346 |
| Uploading Multiple Procedures with the Database Sharing Options Wizard | 347 |
| Testing Web Procedures with the CGI Simulator Wizard | 348 |
| Query Mode (get vs. post) | 348 |
| Testing Get Queries | 350 |
| Repeating a Previous Query | 351 |
| Testing Get Queries from the Debug Menu | 352 |
| Testing Post Queries (Forms) | 354 |
| Testing Post Queries Directly from a Panorama Form | 356 |
| Testing Post Queries Directly from a Panorama Procedures | 356 |
| Testing Post Queries from a Subroutine | 357 |
| Testing Forms in Separate HTML Files | 358 |
| Simulating Multiple Request Sequences | 360 |
| Navigating Within a Session | 365 |
| Re-Simulating a Previous Query | 366 |
| Starting a New Session | 366 |
| Disabling Automatic Browser Preview | 366 |
| Simulating Cookies | 367 |
| Generating HTML | 369 |
| What is HTML? | 369 |
| Other Web Languages (JavaScript, CSS) | 370 |
| Directly Generating an HTML Page | 370 |
| Customizing the HTML Page Header | 373 |
| Placing Fields and Variables into the HTML Page | 374 |
| Fields or Variables with Special Characters | 375 |
| Links to Other Web Pages | 378 |
| Images | 379 |
| Generating a Page Using a Panorama Form Template | 379 |
| The RenderWebForm(Function | 379 |
| Using a Web Form to Display Data | 380 |

| | |
|--|------------|
| Specifying the Record to Display | 381 |
| Customizing the Web Form's HTML Header (Page Title, etc.) | 382 |
| Using Variables to Customize a Web Form on the Fly | 383 |
| Which Came First, the Chicken (Web Template) or the Egg (Variables)? | 385 |
| Using a Web Form to Submit Data | 386 |
| Designing Web Forms for Submitting Data | 386 |
| Displaying a Blank Web Form | 387 |
| Pre-Filling Database Fields | 388 |
| Pre-Filling Variables | 392 |
| Setting Hidden Form Values | 393 |
| What Record Are We Talking About? | 395 |
| "Roll Your Own" Web Navigation | 395 |
| Searching the Database in Reverse | 397 |
| Using WebURLFind For Navigation in Shared Databases | 398 |
| Using WebURLFind for Navigation in Non-Shared Databases | 401 |
| Generating an HTML Table from a Panorama Array | 403 |
| Array Rendering Options | 404 |
| Table Column Layout | 404 |
| Table Font, Font Size and Color | 408 |
| Table Borders and Spacing | 408 |
| Table Background Colors | 409 |
| Displaying an Array in a Web Form | 411 |
| Generating an HTML Table or List from Multiple Records | 413 |
| What Records are we Talking About? (The WebSelect Statement) | 413 |
| Using Variables in a WebSelect Statement | 413 |
| Other Web Selection Statements | 414 |
| Why Not Use the Select Statement? | 414 |
| Generating HTML Tables Using a Web Table Template | 414 |
| Generating HTML Tables Without a Template ("from scratch") | 414 |
| Table Field Layout | 415 |
| Table Font, Font Size and Color | 417 |
| Table Borders and Spacing | 417 |
| Table Background Colors | 418 |
| Table Sort Order | 420 |
| Linking Individual Table Rows to Detail Pages | 421 |
| Splitting a Long Table into Multiple Pages | 424 |
| Displaying an Empty Table | 426 |
| Table HTML Layout | 427 |
| Modifying a Web Table Template On the Fly | 427 |
| Rendering a List (and lists, Formatted Tables, JavaScript) | 428 |
| Linking a Table or List with a Detail Form | 432 |
| Processing Web Forms | 433 |
| Accessing Web Form Information | 434 |
| What Form was Submitted? (The WebFormName() Function) | 434 |
| What Items were Submitted? (The WebFormItems() Function) | 434 |
| Hidden Items | 434 |
| Accessing Form Item Values in a Formula | 435 |
| Assigning a Web Form Item Value into a Panorama Field or Variable | 435 |
| Assigning Multiple Items into Multiple Fields and/or Variables | 435 |
| Validating Data Entry (Error Checking) | 437 |
| Generating a List of Data Entry Errors | 437 |
| Buffered Data Entry | 438 |

| | |
|--|------------|
| Copying from the Variable Buffer into the Database | 440 |
| Data Pre-Processing in the Variable Buffer | 441 |
| Validating Data in the Variable Buffer | 441 |
| Identifying Items with Data Validation Problems | 442 |
| Displaying an Image Instead of an Exclamation Point | 444 |
| Displaying Error Explanations | 445 |
| Pre-Filling Database Fields when Displaying a Form that uses Buffered Data Entry | 447 |
| Non Data Entry Forms (Searching, Navigation, etc.) | 448 |
| Pre-Filling Non Data Entry Forms | 450 |
| Using Cookies to Remember Form Values | 451 |
| Advanced Topics | 453 |
| Working with Cookies | 453 |
| Cookie Name and Value | 453 |
| Creating a Cookie | 453 |
| Cookie Expiration Date | 453 |
| Cookie Options | 454 |
| Retrieving a Cookie Value | 454 |
| Listing Cookies | 454 |
| Building a Very Simple Shopping Cart | 454 |
| Displaying the Shopping Cart | 455 |
| Adding a New Item to the Cart | 457 |
| Clearing the Shopping Cart | 459 |
| Accessing Additional Web Server Information | 460 |
| Non HTML Content Types (text/plain) | 461 |
| Custom Handling of Programming Errors | 462 |
| Writing a Procedure to Handle Programming Errors | 463 |
| Opening and Closing Databases on the Server | 464 |
| Combining Web Publishing with Database Sharing | 465 |
| Web Publishing vs. Record Locking | 465 |
| Manually Locking and Unlocking Records | 465 |
| Checking Record Lock Status | 465 |
| Running a Program Every Minute on the Server | 466 |
| Web Based Data Entry with the WebFormToDatabase Statement | 467 |
| Web- | |
| FormToDatabase — Displaying Form Items | 471 |
| WebFormToDatabase — Adding a New Record | 472 |
| Setting up a Custom Response Page | 473 |
| Forcing Input to Upper or Lower Case | 473 |
| Checking for Missing Fields | 473 |
| Data Entry for Number and Dates | 476 |
| Saving the Modified Database | 476 |
| WebFormToDatabase — Modifying an Existing Record | 476 |
| Handling Missing or Ambiguous Records | 477 |
| WebFormToDatabase — Modifying an Existing Record with Embedded Record ID | 477 |
| Manually Creating Web Forms for use with the WebFormToDatabase Statement | 478 |
| Web Form Based Data Selection | 479 |
| Searching All Fields | 482 |
| Rendering Using an External Text File as a Template | 483 |
| Rendering Using an External Text File containing a Form as a Template | 484 |
| Enterprise Sharing vs. Butler | 489 |
| System Architecture | 489 |

| | |
|---|------------|
| Speed | 489 |
| Database Configuration | 490 |
| Data Storage Flexibility | 490 |
| Client Subsets | 490 |
| Converting from Butler to Enterprise Edition Server | 491 |
| Reprogramming your Application | 491 |
| Secrets of the Enterprise Folder | 493 |
| cgi-bin | 494 |
| Log Cache..... | 494 |
| Logs | 494 |
| PanoramaCGI | 494 |
| Public Databases | 494 |
| ServerConfig.dat | 494 |
| ServerList.dat..... | 494 |
| temp | 495 |

Panorama Enterprise Quick Reference



We highly recommend that you carefully read and study the following detailed chapters before setting up and using the Panorama Enterprise Edition Server. Don't try to learn how to use the system from the checklists below!

Once you are familiar with the material the following checklists will help keep you organized and help you follow all the steps for each task in the proper order.

System Requirements (Server)

To run the Panorama Enterprise Server you'll need a system that meets these requirements:

- Macintosh (PPC or Intel) Computer
- Memory at least 110% the size of all combined server databases (200% recommended)
- OS X 10.4 (Tiger) or later
- Panorama 5.5 Server License (database sharing, web, or both)

System Requirements (Client)

Each system that needs to access shared databases must meet these requirements:

- Macintosh (PPC or Intel) Computer
- OS X 10.3 (Panther) or later (OS X 10.4 Tiger recommended)
- Panorama 5.5 or Panorama Direct 5.5 License
Personal Use License may only log on from one computer at a time.

It's not required, but we highly recommend you install **Growl** for notifications. See "[Using Growl for Notifications](#)" on page 87 for more information on installing and using this open source package.

System Configuration (Local Database Sharing)

Before you can begin using the Panorama server to share databases over the local network you'll need to properly configure your server's operating system. Most of this configuration is done using the OS X **System Preferences** application, which you can launch from the dock.

- Set up the computer name (see "[Setting up the Computer Name](#)" on page 32)
- Enable remote Apple Events (see "[Enable Remote Apple Events](#)" on page 33)
- Turn off sleep (see "[Ensuring Nonstop Server Operation](#)" on page 39)
- Enable VNC for remote operation
(optional, see "[Enabling Remote Server Operation](#)" on page 40)
- Install Panorama 5.5 w/server (see "[Installing the Panorama Server Software](#)" on page 43)
- Enable automatic server restart when system reboots
(optional, see "[Setting up Panorama Server to Launch Automatically](#)" on page 47)
- Activate Panorama Server license (unless using demo version)
(see "[Panorama Server Product Activation](#)" on page 48)

Once the system is set up you'll need to configure the Panorama Server itself:

- Launch the Panorama Server (see "[Launching the Server](#)" on page 44)
- Unlock the server (see "[Unlocking the Server Configuration](#)" on page 49)
- Change the server password (see "[Changing the Server Password](#)" on page 51)
- Change the server name (see "[Changing the Server Name](#)" on page 53)
- Enable database sharing (see "[Enabling Database Sharing](#)" on page 55)
- Test the server (see "[Configuring Local Database Sharing](#)" on page 58)

System Preparation (Database Web Publishing)

Before you can begin using the Panorama server for publishing database information on the web you'll need to properly configure your server's operating system. Most of this configuration is done using the OS X **System Preferences** application, which you can launch from the dock.

- Make sure the computer has a static IP address
(see "[A Brief Introduction to Static IP Addresses](#)" on page 34)
- Turn on Personal Web Sharing (see "[Enabling Internet Sharing](#)" on page 38)
- Set up the computer name (see "[Setting up the Computer Name](#)" on page 32)
- Turn off sleep (see "[Ensuring Nonstop Server Operation](#)" on page 39)
- Enable VNC for remote operation
(optional, see "[Enabling Remote Server Operation](#)" on page 40)
- Install Panorama 5.5 w/server (see "[Installing the Panorama Server Software](#)" on page 43)
- Enable automatic server restart when system reboots
(optional, see "[Setting up Panorama Server to Launch Automatically](#)" on page 47)
- Activate Panorama Server license (unless using demo version)
(see "[Panorama Server Product Activation](#)" on page 48)

Once the system is set up you'll need to configure the Panorama Server itself:

- Launch the Panorama Server (see "[Launching the Server](#)" on page 44)
- Unlock the server (see "[Unlocking the Server Configuration](#)" on page 49)
- Change the server password (see "[Changing the Server Password](#)" on page 51)
- Change the server name (see "[Changing the Server Name](#)" on page 53)
- Enable web publishing (see "[Enabling Database Web Publishing](#)" on page 56)
- Set public IP address (see "[Setting the IP Address/Domain Name](#)" on page 56)
- Test the server (see "[Testing Web Database Publishing \(Server Status\)](#)" on page 64)

System Preparation (Internet Database Sharing)

If you want to use the Panorama server to share databases over the entire internet (instead of just the local network) you'll need to use both checklists above.

Converting a Single User Database into a Shared Multi-User Database

Follow this checklist to convert a single user database into a multi-user database. For more detailed instructions see [“Creating a Shared Database”](#) on page 103.

- Open the single user database
- Open the **Database Sharing Options** wizard (in the Sharing submenu of the Wizard menu)
- If not already selected, use the first pop-up menu to select the database.
- Use the second pop-up menu to select the server
- Check the **Local Database Sharing** checkbox
- Check the **Internet Database Sharing** checkbox (optional)
- Specify the auto-timeout value (optional, see [“Record Lock Timeout \(Client\)”](#) on page 139)
- If a database with this name already exists on the server, type in an alternate name
- Press the **Apply Changes** button or choose **Apply Option Changes** from the **Maintenance** menu
- Confirm that the settings are correct and press the **Apply Options** button.
- Transfer the client database to the other clients
(see [“Duplicate Database Conflicts on the Server”](#) on page 108)

Shared vs. Single User Database Operation

Operation of shared Panorama databases is very similar to single user use, but there are some differences. See [“Shared vs. Single User Database Operation”](#) on page 135 for a summary.

Converting a Shared Multi-User Database Back to Single User

Follow this checklist to convert a multi-user database into a single user database. For more detailed instructions see [“Saving the Database”](#) on page 152.

- Open the shared database
- Synchronize the database (optional, to make sure you have the most up-to-date data)
- Open the **Database Sharing Options** wizard (in the Sharing submenu of the Wizard menu)
- Uncheck the **Local Database Sharing** checkbox
- Press the **Apply Changes** button or choose **Apply Option Changes** from the **Maintenance** menu
- Confirm that the settings are correct and press the **Apply Options** button.

Note: The shared database remains on the server and can still be accessed by other users.

Adding or Removing Fields from a Shared Database

Follow this checklist to add or remove fields from a shared database. See [“Changing the Design of a Shared Database”](#) on page 119 for a more detailed explanation.

- Start a new shared generation of the database (see above)
- Using the **Setup** menu or the Design Sheet, add and/or remove fields
- Re-share the database

Removing a Database from the Server

Follow these steps to completely remove a database from the server.

- On any client computer, open the **Server Administration** wizard
(see [“Server Management \(The Server Administration Wizard\)”](#) on page 68)
- Locate the database in the wizard’s list of server databases
- Uncheck the **Online** option for this database
(see [“Database Online/Database Offline”](#) on page 73)
- Hold down the **Control** key and click on the database name
- Select **Delete From Server** from the pop-up menu (see [“Delete From Server”](#) on page 73)

This command is permanent and cannot be reversed or undone.

Chapter 1: Introduction



The Panorama Enterprise Edition Server extends the reach of Panorama beyond a single computer to a network of connected computers. Using this server you can share data on a local network or even across the entire Internet.

Types of Networks

From Panorama's point of view there are two types of networks: local networks and the internet. A local area network (LAN) is a computer network covering a small local area, like a home, office, or building. For Panorama to consider a network a local network, all of the computers on the network must be connected to a single router. Any other configuration is considered the internet.

When used on a local network, Panorama clients can use Apple's Bonjour (formerly Rendezvous) technology to automatically locate and connect to the Panorama Enterprise Edition Server (even if there are multiple servers on the network). Panorama normally uses Remote Apple Events to communicate on local networks, though direct TCP/IP connections are also allowed (but slower).

When used on the internet (any non-local network), the server address (domain name or IP address) must be manually set up on each client. The process only takes a few seconds per client (similar to typing in the url in a web browser), but is not automatic. Panorama uses TCP/IP to communicate over the internet.

Client/Server Modes

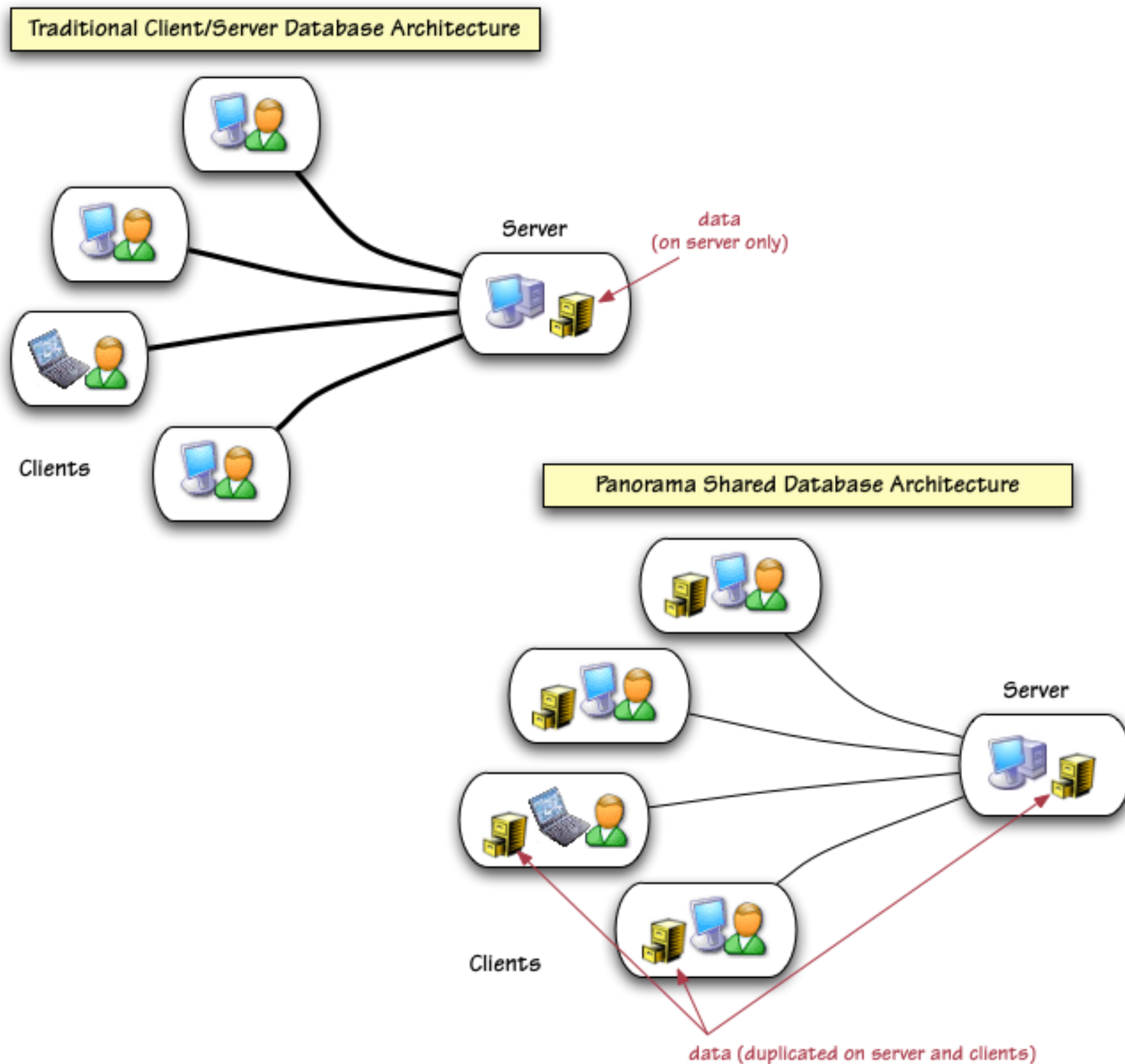
The Panorama Enterprise Edition Server can operate in two different client/server modes: database sharing and web publishing. The mode used depends on the client software. In **database sharing mode** the client is Panorama itself. Database sharing mode allows you to use Panorama on your local computer in almost the same way you would with a single user database. The server co-ordinates data flow between multiple clients so that data can be shared seamlessly. In most cases a single user Panorama database can be converted to a shared database with almost no preparation and will continue to operate almost identically to the single user version.

In **web publishing mode** the client is a web browser — Internet Explorer, Safari, Firefox, etc. This mode allows any authorized user on the internet to access and modify the database. Unlike database sharing mode, no copy of Panorama is required on the client computer, anyone with a web browser can join in. There are, however, some downsides to this mode. First, a database must be prepared to work in this mode. For simple applications this may take only an hour or two, while more complex applications may take hundreds of hours. Secondly, the user interface in web publishing mode is limited to the capabilities of a web browser. Advanced Panorama features like Clairvoyance[®], Smart Dates[™], Super Matrix, Elastic Forms, Word Processing, Custom Menus, Automatic Capitalization, Phone Dialing and more are not available when using a web based database. If you need universal global access to your database, however, web publishing mode is the way to go.

Database sharing mode and web publishing mode are not mutually exclusive — both modes can be in use on a single server simultaneously. In fact a single database may be used in both modes simultaneously, allowing different clients to take advantage of the appropriate mode for them. (For example you might create a product catalog database that is accessed using data sharing mode in house, while also making it publicly available in web publishing mode.)

Database Sharing Concepts and Operation

This section explains how Panorama's database sharing mode works, and the benefits of Panorama's unique distributed data sharing system. Traditional client/server database systems (FileMaker, SQL, etc.) store the data only on the central server computer. Panorama, in contrast, keeps copies of the data on the server but also keeps duplicate copies on every client.



By distributing multiple copies of the data across the network, Panorama uncorks the network/server bottleneck that hampers traditional client/server database systems. Distributing the data has several key benefits over these traditional systems: high performance, low network/server load, offline operation and high safety margin. Each of these benefits is explored further in the sections below.

High Performance

When sharing a Panorama database, each client keeps a full copy of the database in RAM on the client computer. Operations like searching, sorting and reports happen almost instantaneously in RAM with no network or server delay. The server only gets involved when data is modified (it handles record locking and updating other clients as necessary). (Traditional client/server systems, in contrast, channel every database action (query, sort, report, etc.) through the network and the server. The client is only used to display the results.)

Since Panorama is RAM based, it doesn't need or use indexes. (Indexes are special tables that disk based databases use to speed up searching and sorting.) Eliminating indexes simplifies database design, allows more flexible searching and also dramatically reduces RAM and disk requirements on each client (Panorama databases are typically much smaller than the traditional data + index tables combination — in some cases up to 90% smaller).

Low Network/Server Load

Since the network and server are not involved in a majority of database actions, the load on these components is much lower than when using a traditional client/server system. Bottom line — you'll be able to scale to more clients without upgrading your network or server hardware.

Offline Operation

Unlike a traditional client/server database, Panorama database clients don't "go dark" if a network connection is unavailable. Panorama database sharing allows off-line database browsing and even modification (configurable on a per database basis). If allowed, offline changes are automatically synchronized with the server when the client re-connects to the network. Because both client and server are RAM based, this synchronization is extremely fast. If there is a potential conflict between the modifications made offline and the modifications made by other users, you will be notified and given the option to resolve the conflict manually.

A Panorama database can even be configured to operate primarily offline. This is similar to the way e-mail works - users perform data entry offline, then press **Submit** or **Connect** to submit their data and receive updates. In the past applications like this had to be built from scratch, but Panorama database sharing allows this with little or no custom programming.

High Safety Margin

Engineers understand that critical structures like bridges and aircraft must be resiliently designed to eliminate the possibility of total failure. Your data is critical to you so the Panorama Enterprise Edition Server includes layered safeguards to protect every byte.

Zero-Loss Interruption Recovery

The Panorama server is RAM based for high performance, but it also keeps a disk based transaction journal for full data recovery after any kind of power failure or system interruption of any kind (the journal is a simple sequential file with minimal impact on performance). The server will automatically recover any unsaved data when the system reboots — no manual intervention is necessary.

Automatic Live Backups

The Panorama server includes a built in backup system that can perform automatic backup of "live" databases without shutting down the server. (Live backups are performed to another folder or hard drive, we highly recommend that you use a secondary backup system to regularly copy the backed up files to a remote location. The small file size of Panorama databases helps make this process smoother, especially if you are backing up over the Internet or other relatively slow network.)

Detailed Logging and E-mail Notification

The Panorama server keeps detailed, configurable logs of server activity and can also e-mail the administrator automatically if any kind of server error occurs (including reboots).

Rebuild from any client

In a worst case scenario Panorama's distributed architecture provides an automatic built-in backup system. If your regular backup is unavailable (earthquake? hurricane? tsunami?) you can always rebuild the server database from the data in any client.

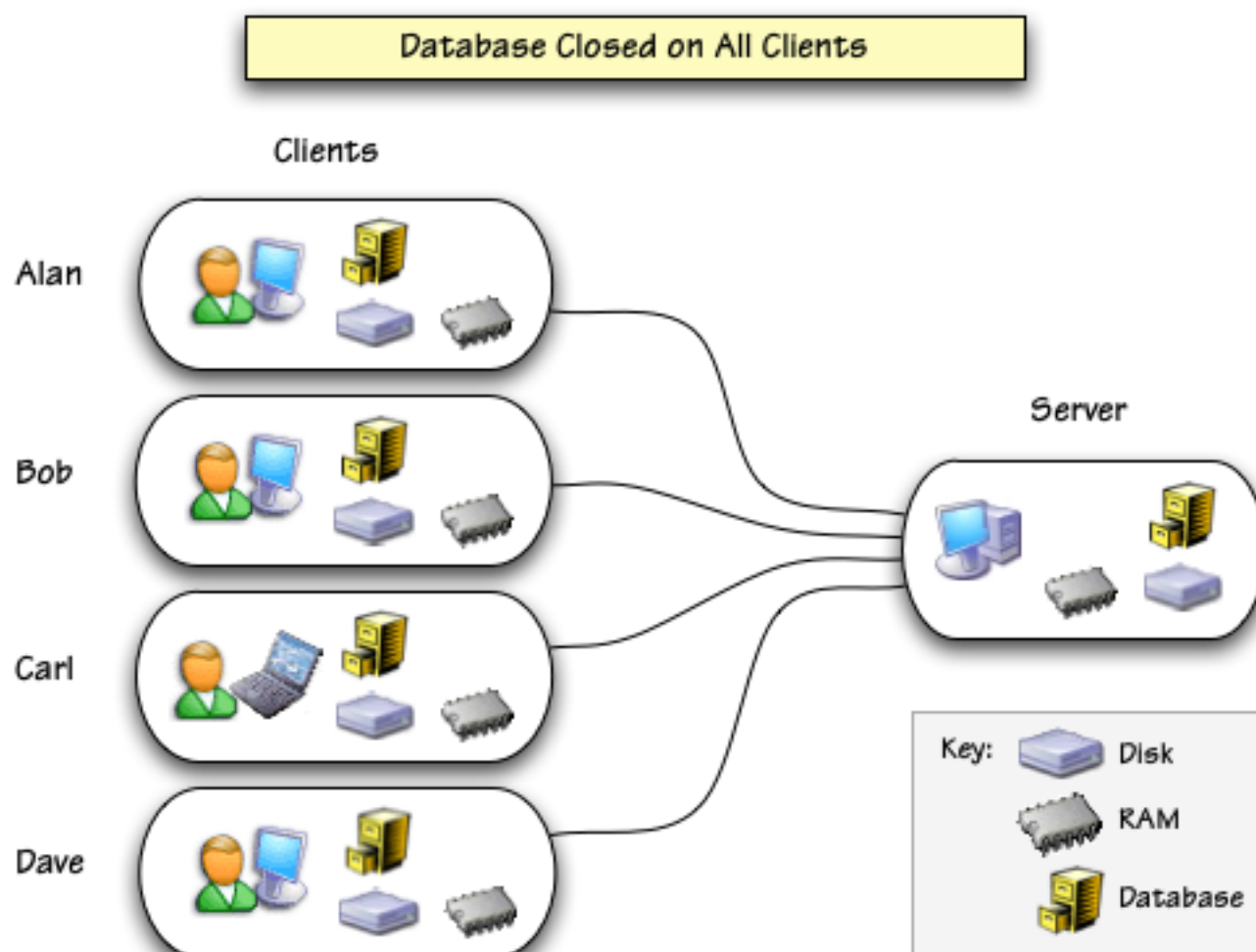
No delicate index to get corrupted

Since Panorama is RAM based, it doesn't need or use indexes (see "[High Performance](#)" on page 19). In addition to the excessive space they take up, indexes have another disadvantage — they are complex and fragile structures that are easily corrupted. Although a corrupted index usually doesn't mean the data itself is lost, a database may be offline for a considerable time while waiting for an index to be rebuilt. Since Panorama doesn't use indexes, the possibility of a corrupted index is eliminated.

How Distributed Data Sharing Works

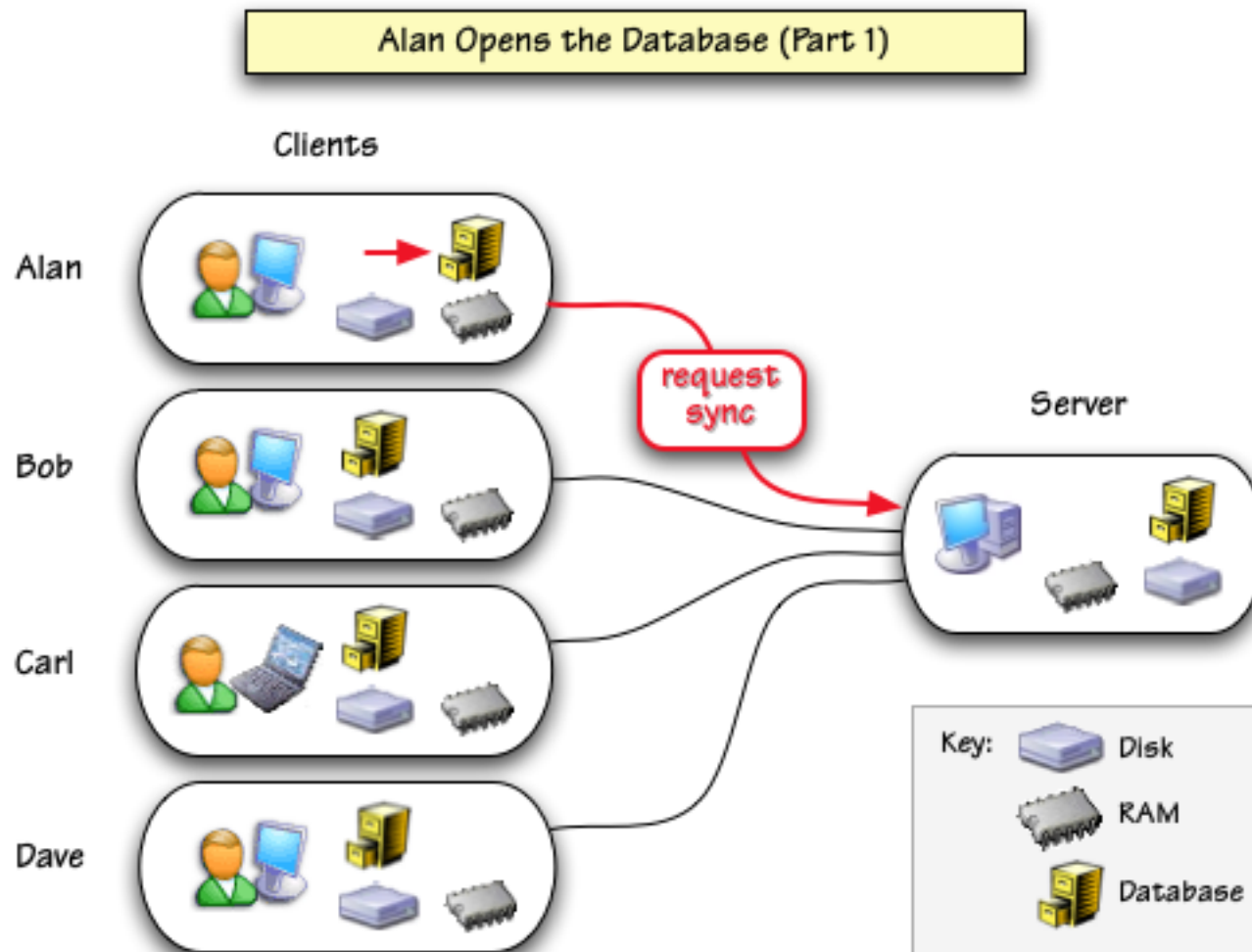
It's not necessary to understand how distributed data sharing works to use the system — you can simply open your databases and access them as you normally would. If you are not interested in the details you can simply skip this section. However if you've used other client/server systems you might be more comfortable knowing exactly how Panorama's distributed data sharing manages to juggle data across a network without spilling a drop. We think that the more you know about Panorama's unique system, the more you'll like it.

We'll use a series of diagrams to illustrate how distributed data sharing works and how data flows across the network. Our hypothetical network contains a server and four clients: Alan, Bob, Carl and Dave. For simplicity we'll assume that these users are only sharing a single database, which we'll simply call **the database**. We start with this database all set up and ready to use, with a copy of the database on the hard drive of each client and on the server as well. (To learn how these copies are originally set up see "[Creating a Shared Database](#)" on page 103.)

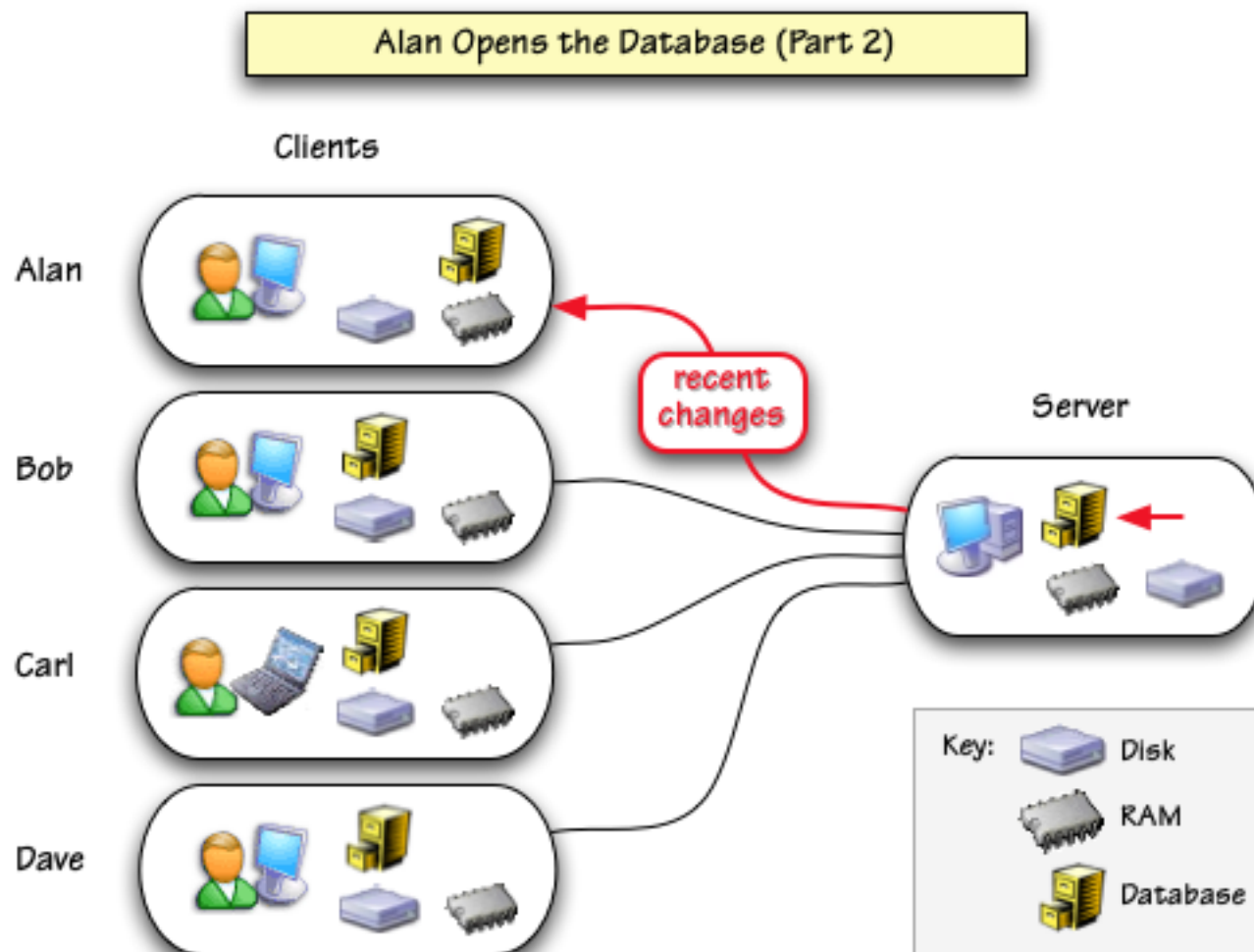


Opening a Database

The action begins as Alan double clicks on his copy of database to open it. Panorama loads the database into RAM, then contacts the server to request synchronization (more on synchronization in a moment).



In response, the server loads its copy of the database into RAM. It then gathers the most recently modified data and sends it back to Alan's computer.



Alan's computer takes these recent changes and updates its local copy of the database. Because Panorama is RAM based, this synchronization process is very rapid — often nearly instantaneous. Alan now has an up-to-date copy of the database available for searching, sorting, reports, etc. All of these operations are performed at blazing speed in RAM, just as it would be done if this was a single user database.

The Synchronization Process

How did the Panorama server know which changes were recent, and which changes Alan already had on his computer? And how did the Panorama client know how to merge this recently changed data with the data it already had?

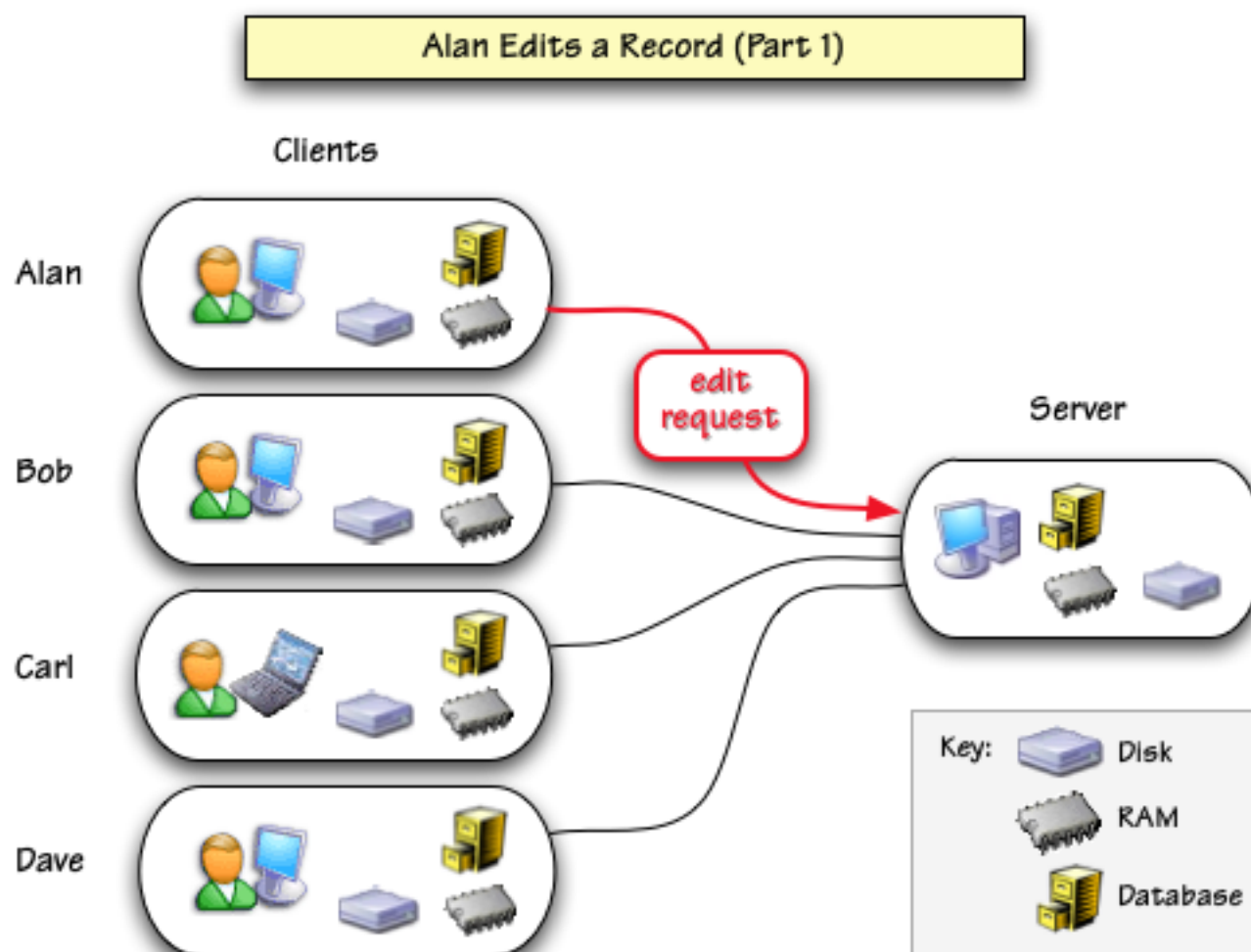
When a Panorama database is converted from single user to shared two invisible fields are added to each record in the database: **ID** and **Time Stamp**. The ID is a unique number that is assigned to the record when it is first created and never changes. The Time Stamp keeps track of when this record was last modified. (These extra fields are both invisible, and cannot be seen or modified by the user.)

Panorama also keeps an overall time stamp for each database. This overall time stamp keeps track of when the database was last synchronized with the server. When a client requests synchronization, it passes this overall time stamp as part of the request. The server gathers all records with later time stamps (including any brand new records) and sends them back to the client. The client matches up the ID numbers to update existing records, and simultaneously adds the new records to complete the synchronization process.

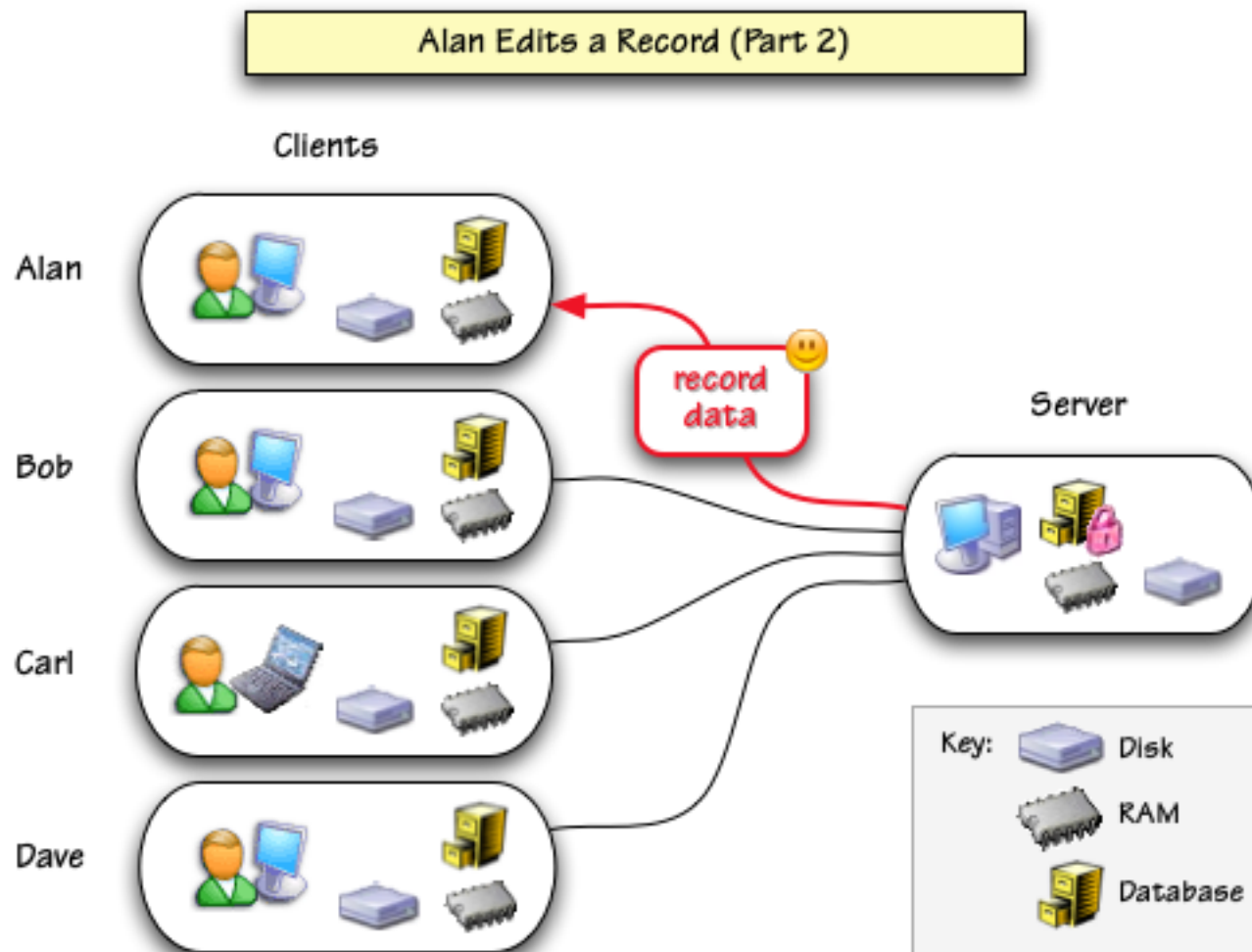
A Note about Time Stamps. Time stamps are not really “clock times” in the ordinary sense of days, hours, minutes and seconds. Instead, this “time” is really a count of the total number of changes made to this database since the database was first created. Like a clock, this number always increase as time goes on. But since we don't use a real clock, it doesn't matter if the time is incorrectly set on one or more of your computers, or even if some of the computers are in a different time zone.

Editing and Record Locking

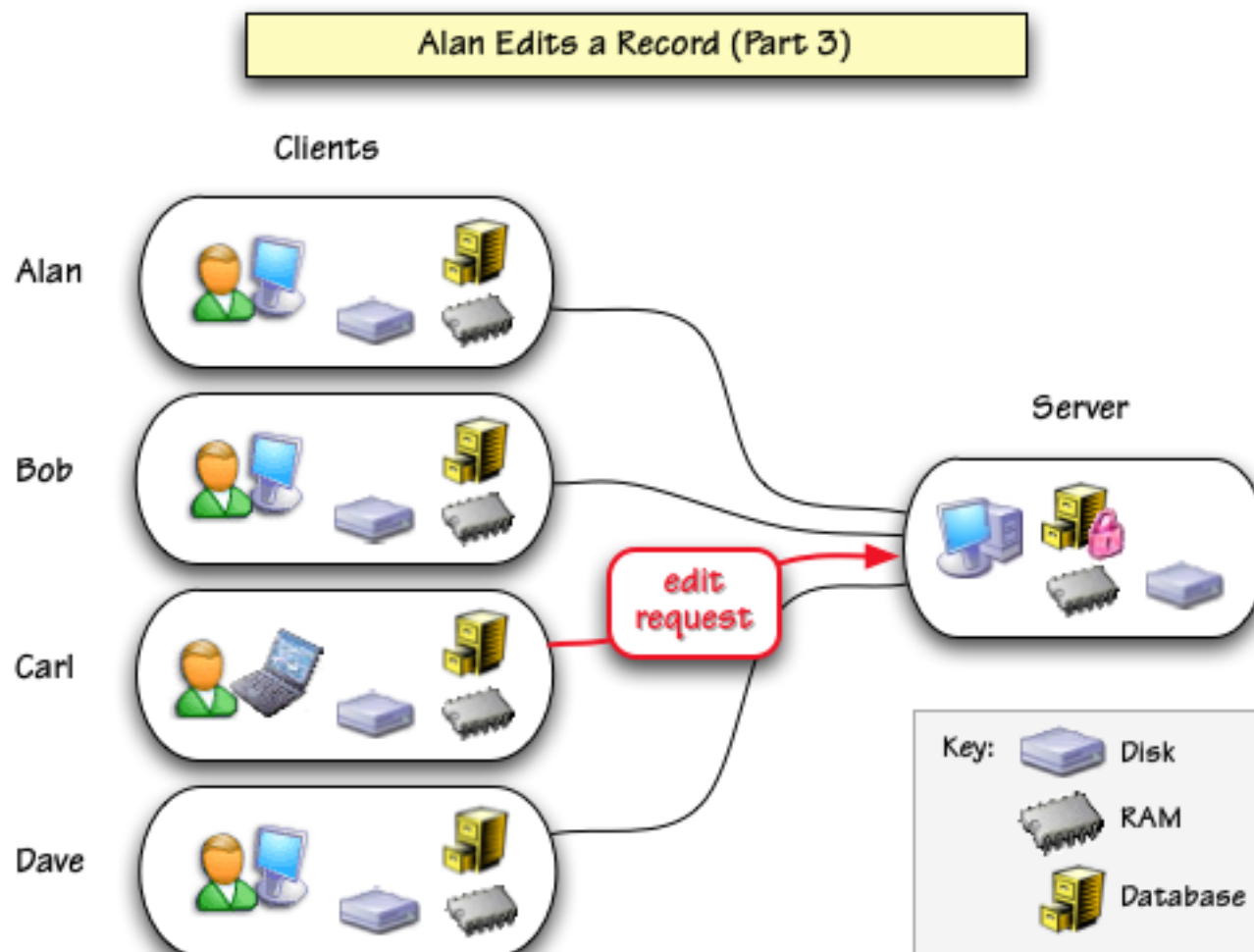
To edit a record, Alan double clicks a data cell in the record. Panorama sends a request to the server asking to edit the record (the request includes the records unique ID number). (Notice that we are assuming that Bob, Carl and Dave have now all opened this database, so the database is loaded into RAM on all four of these client machines.)



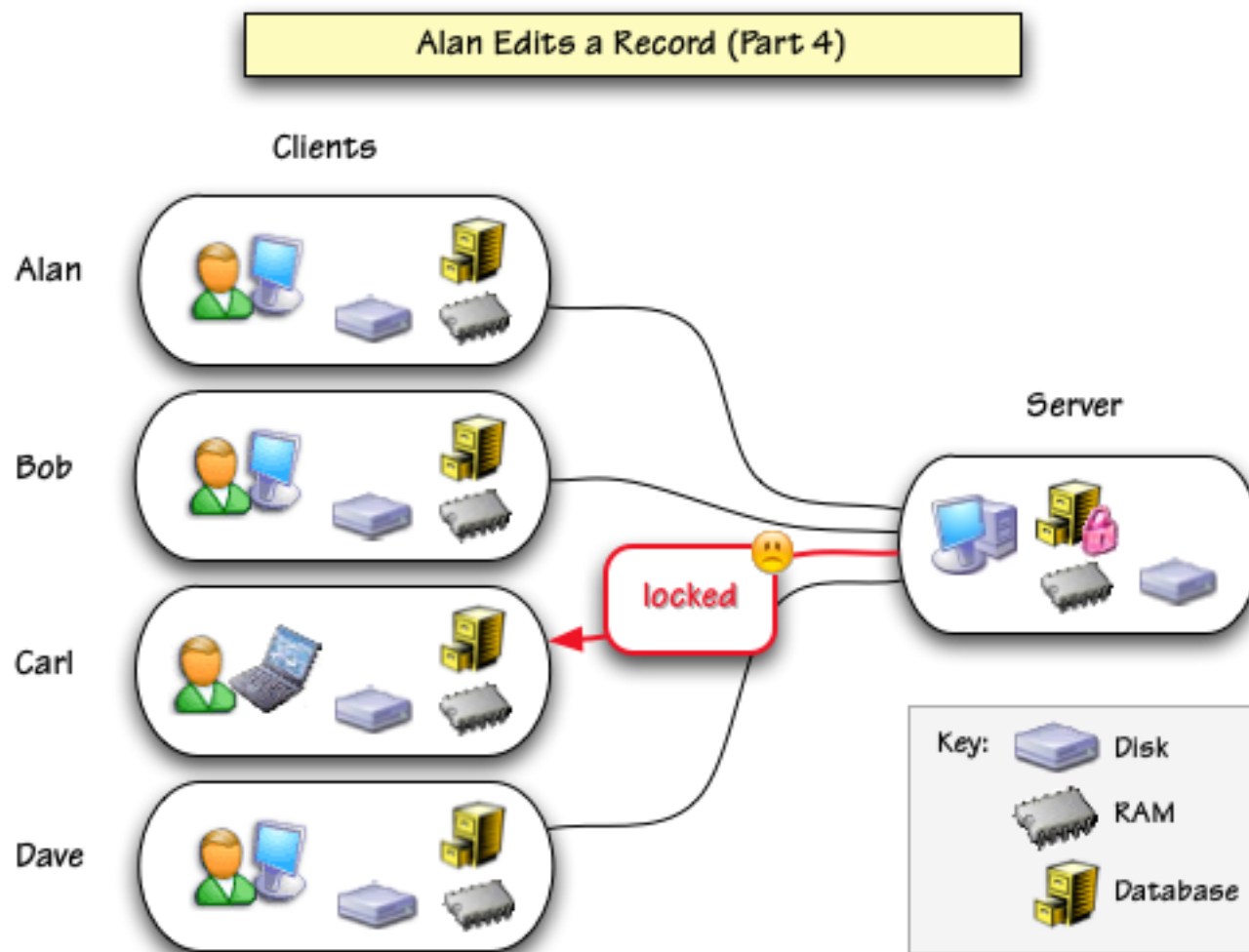
If no one else is currently editing that record, the server will respond that it is ok to edit the data. The server will also send a copy of the most up-to-date data for this record. Using this data, Panorama performs a “mini-synchronization” of just this record. This insures that the data in this record is completely up-to-date before Alan makes any changes.



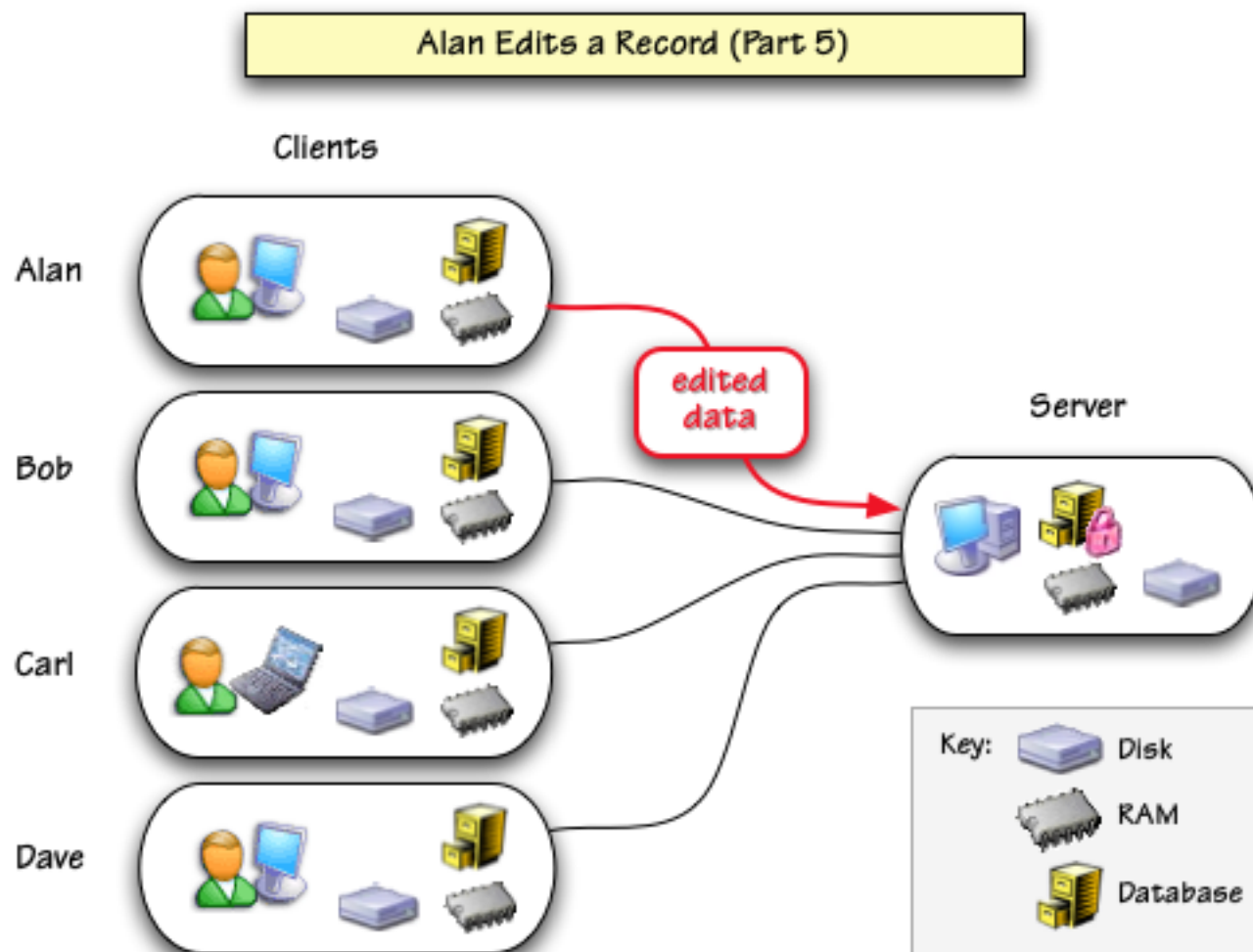
Suppose that Carl double clicks on a data cell in the same record before Alan finishes editing it? Just like before a request is sent to the server.



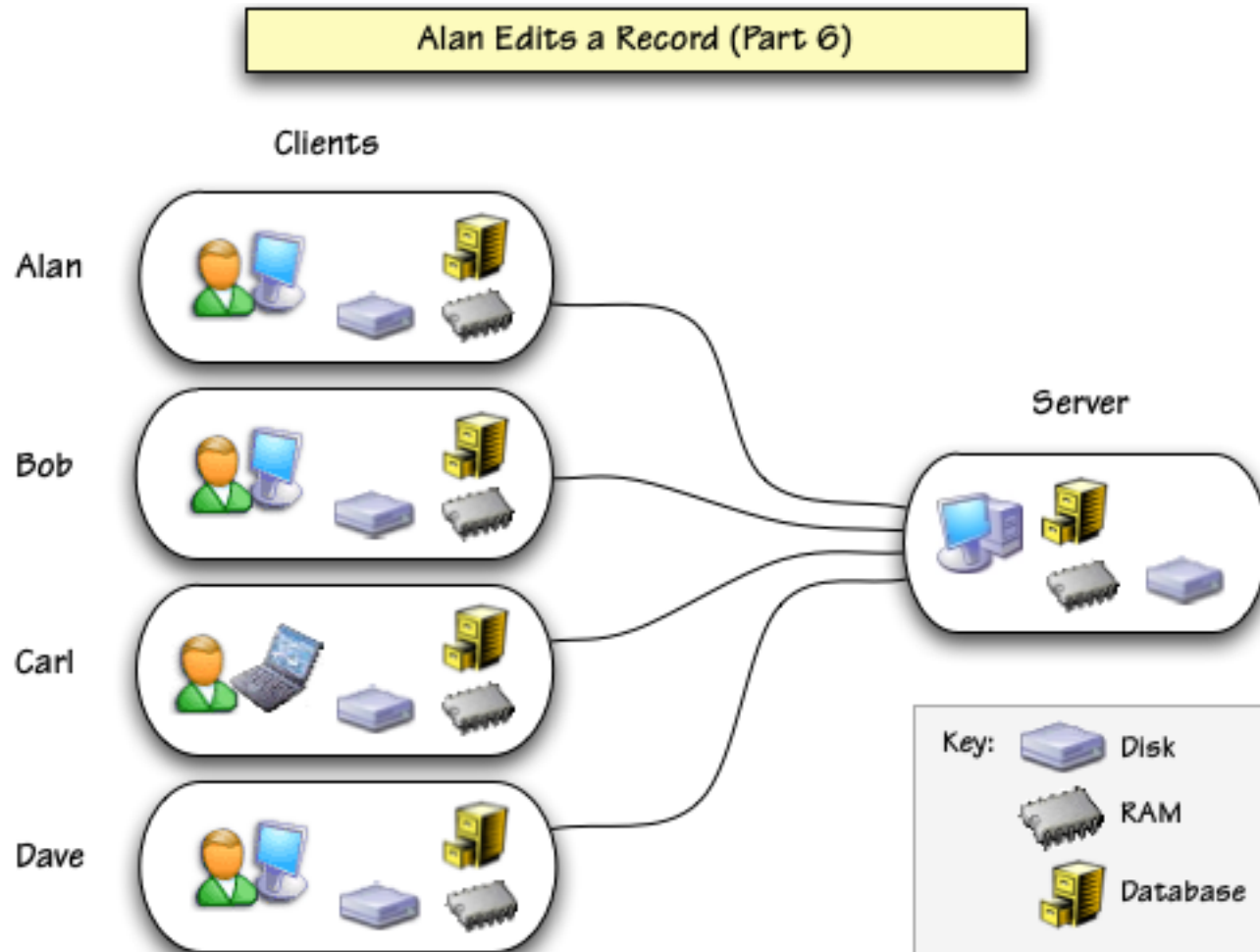
But Alan still has this record locked, so Carl's request is denied. Carl will see an alert suggesting that he try again later. Of course Carl can edit any other record in the database.



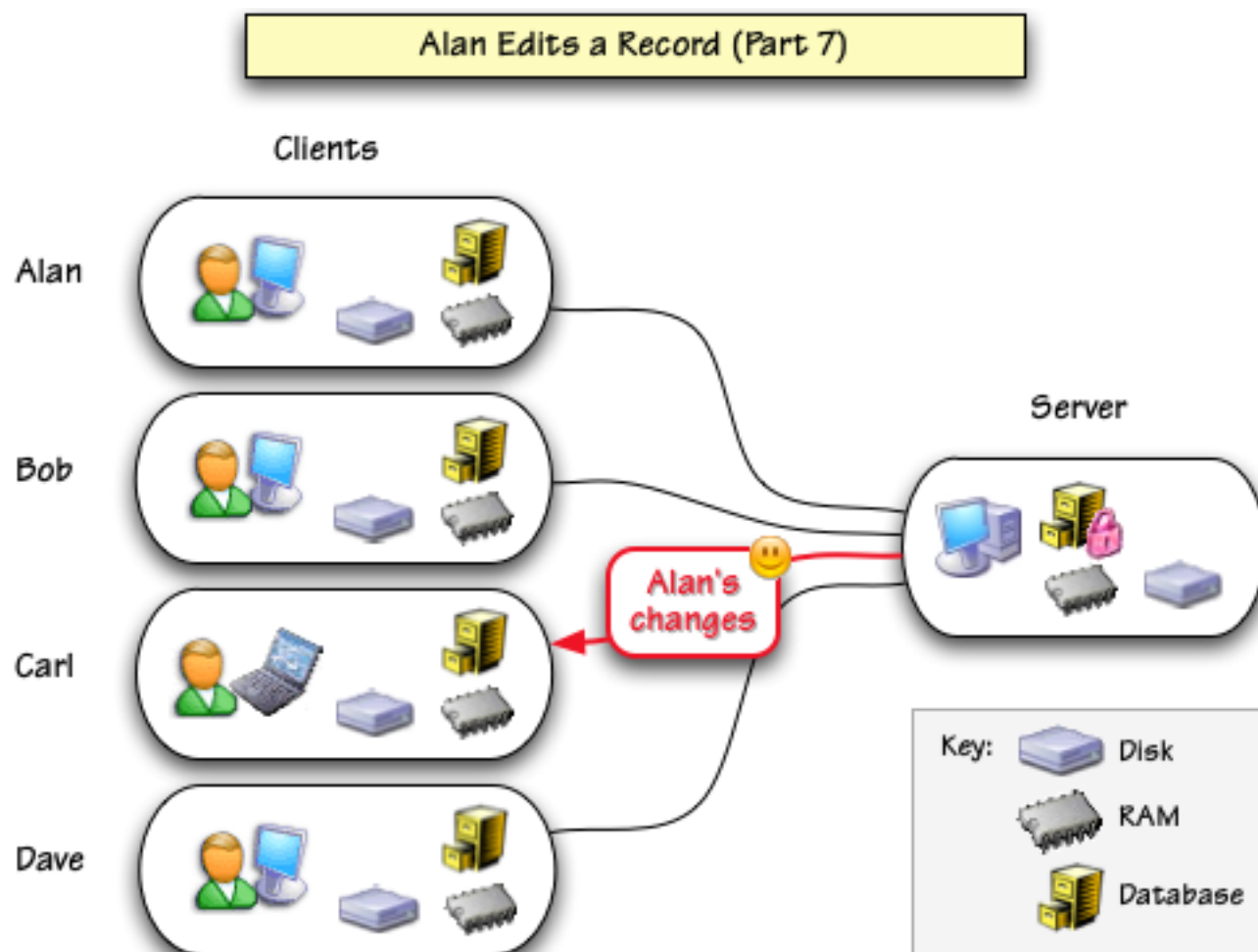
When Alan finishes editing the record (indicated by clicking on another record, clicking on another window, saving the database or simply being inactive too long) the freshly edited data is transmitted to the server.



The server updates its copy of the database and releases the lock.



Now suppose Carl tries to edit the same record again. This time his request will succeed.



As Carl double clicks on the data cell he'll see Alan's changes appear before he begins editing. In some cases this can be very important. For example, suppose this is an inventory database and Alan subtracted items from a particular inventory record. Now Carl wants to subtract additional items from the same record. It's vital that if Alan subtracted 12 and Carl now subtracts 9, the final amount will be the original amount minus

21. Because the server updates each client as the record is locked, this correct total is assured no matter how many calculations are chained together. In this example we've assumed that the data is being edited manually, but the same principle applies when modifications are made by a program. Whenever a record is modified it, is always locked and updated first, whether that modification is manual or done by a program.

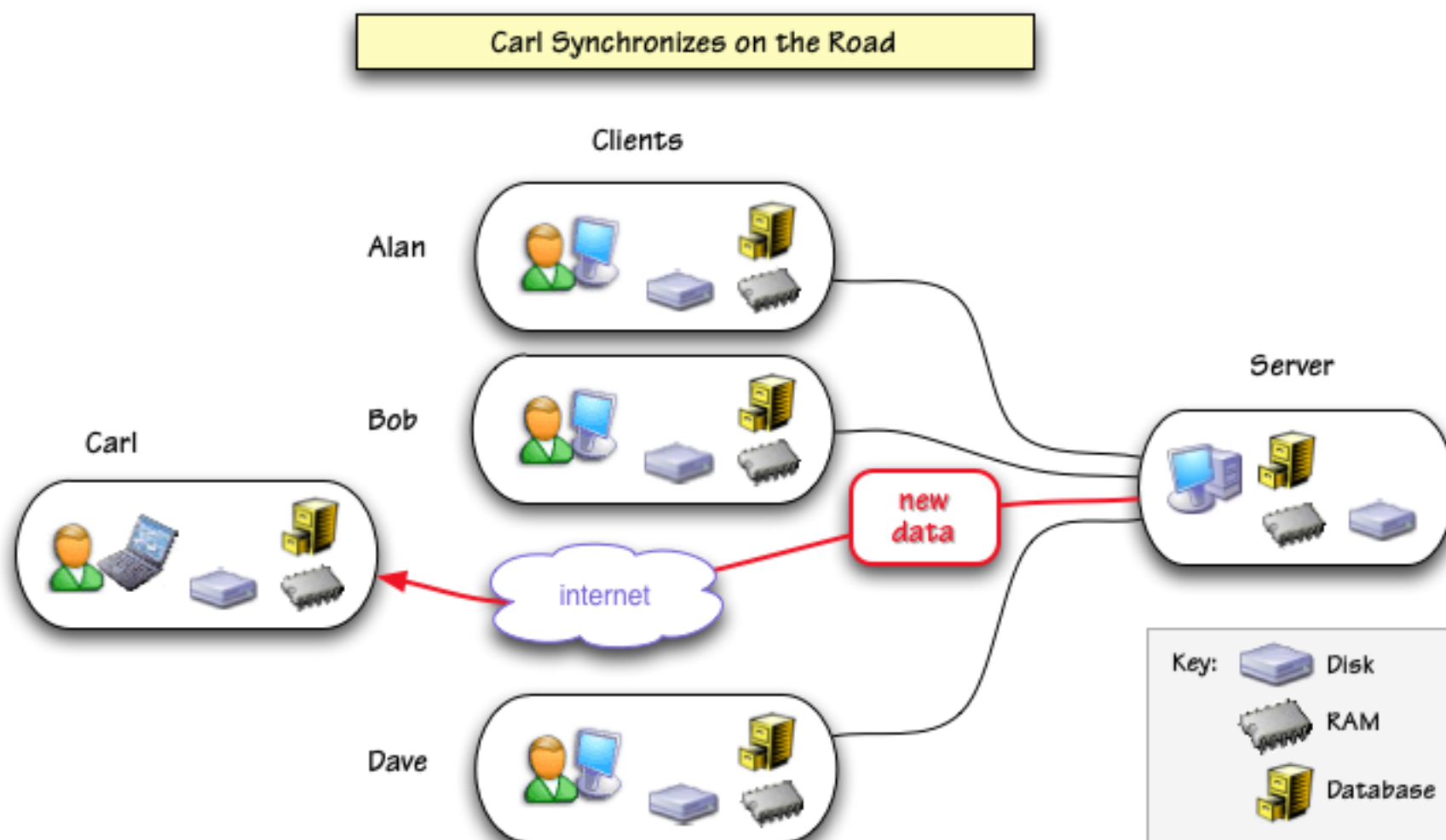
Taking Data On The Road (Offline Database Operation)

Mobile users can use their local copy of the database even when disconnected from the network. When mobile databases are reconnected to the network, the server and remote databases are automatically synchronized and reconciled with each other on a record-by-record basis.

When Carl takes his laptop on the road he may not always be able to connect to the server back at the office. Even when he's not connected, however, he can always open and view his local copy of the database. He can search, sort, even calculate summaries and print reports. Of course he won't have the most up-to-date data, but in many cases that is much better than nothing.



The next time Carl connects to the office, even over the Internet, he can synchronize to bring his local copy back up-to-date.



(Of course as long as Carl is connected to the server he can work normally, editing data and synchronizing just as if he were back at the office.)

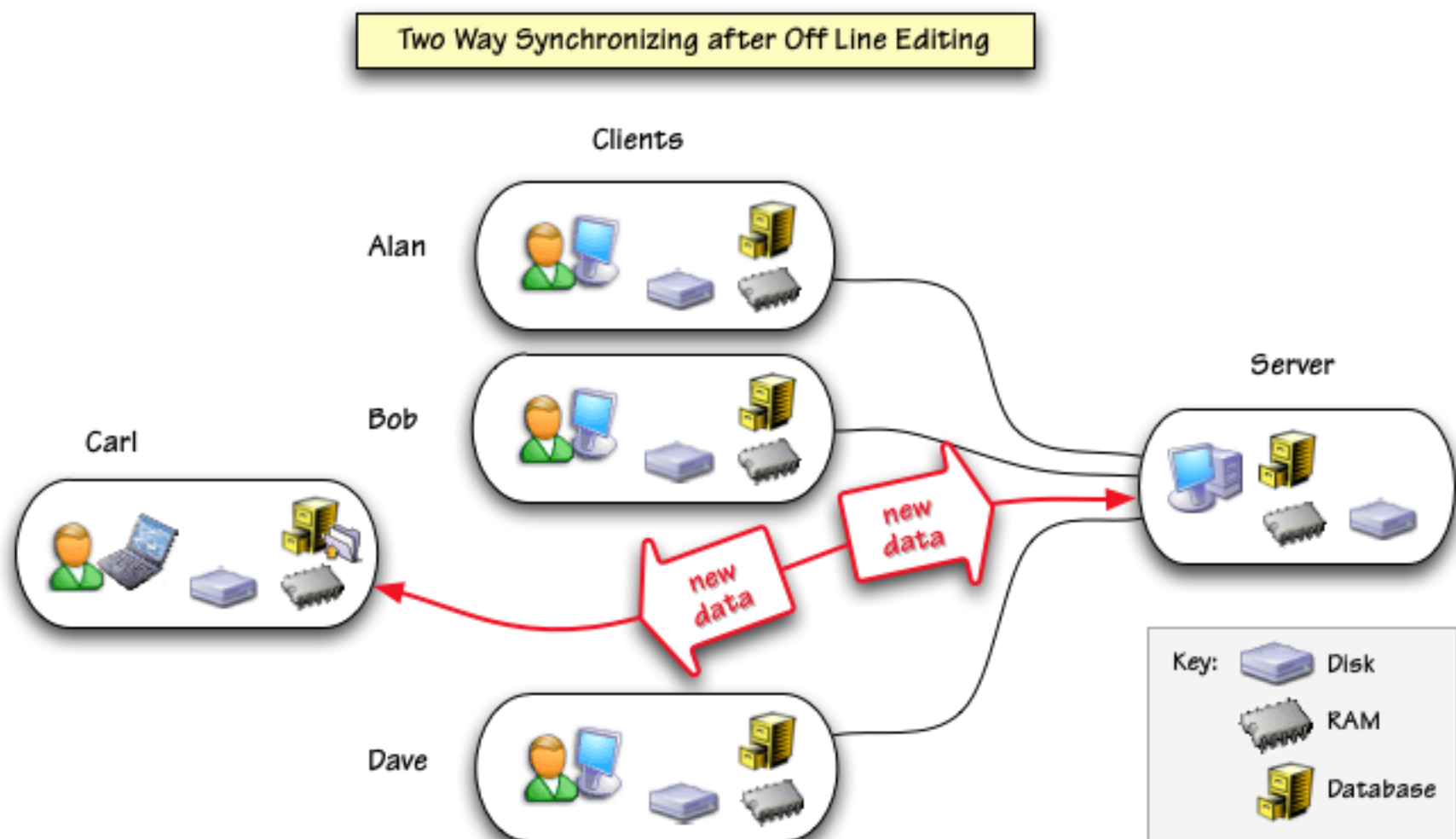
Modifying Data Offline

Can Carl edit his database when he is not connected to the server (off-line)? By default the answer is no. However, this ability can optionally be enabled when setting up the sharing options for any database. If it is enabled, Panorama will keep track of all of the changes Carl makes when he is off-line.



Of course since Carl is not connected to the server there is no record locking. You should leave off-line editing disabled for any databases for which record locking is vital to database integrity.

The next time Carl connects to the server (whether locally or over the Internet), the system will perform a two way synchronization. All of Carl's new changes will be updated on the server, and all of the changes made by other users will be transferred to Carl.



Since there is no record locking when off-line, it's possible that Alan, Bob or Dave may have changed some of the same records that Carl has changed while off line. When you set up the database options you can configure either the server or the client to have priority. As a third options, Carl can review the conflicting changes manually and "cherry pick" the final data on a field-by-field basis. Keep in mind that off-line editing is entirely optional. If you are concerned about database integrity you should consider leaving this option disabled.

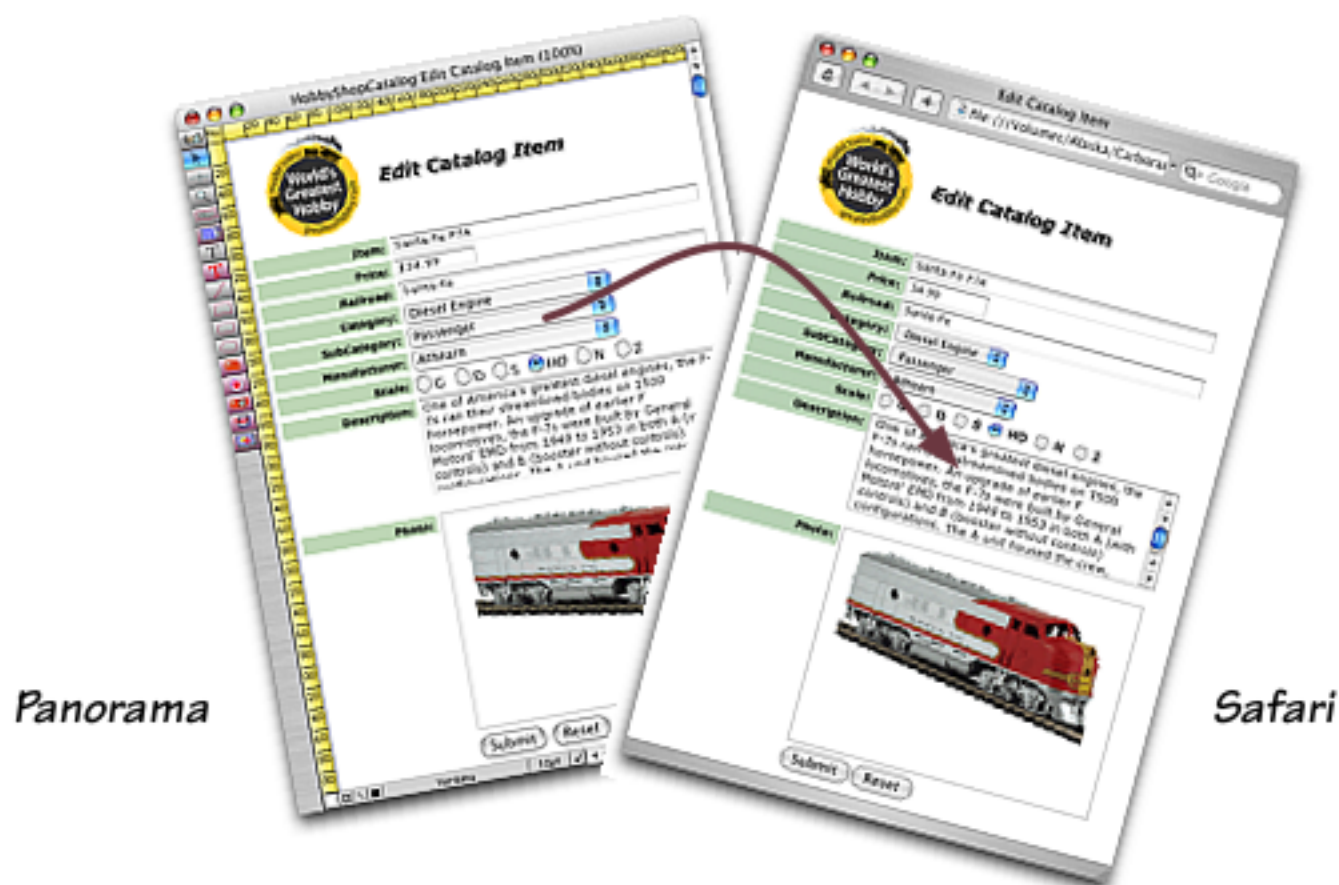
Database Web Publishing Concepts and Operation

Like database sharing, web publishing allows a database to be viewed and accessed remotely. Instead of using a custom Panorama client, web publishing allows the database to be used by anyone with a modern web browser. This approach makes possible for a truly global audience for your database. There are, however, some downsides to using web publishing instead of database sharing. First, a database must be prepared to work in this mode. For simple applications this may take only an hour or two, while more complex applications may take hundreds of hours. Secondly, the user interface in web publishing mode is limited to the capabilities of a web browser. Advanced Panorama features like Clairvoyance[®], Smart Dates[™], Super Matrix, Elastic Forms, Word Processing, Custom Menus, Automatic Capitalization, Phone Dialing and more are not available when using a web based database. If you need universal global access to your database, however, web publishing mode is the way to go.

Database sharing mode and web publishing mode are not mutually exclusive — both modes can be in use on a single server simultaneously. In fact a single database may be used in both modes simultaneously, allowing different clients to take advantage of the appropriate mode for them. (For example, you might create a product catalog database that is accessed using data sharing mode in house, while also making it publicly available in web publishing mode.)

Panorama Forms on the Web

Just as in Panorama itself, most database display and data entry on the web is accomplished through forms. You can create web forms with standard authoring tools like Dreamweaver, GoLive or (assuming knowledge of HTML) a basic text editor. Panorama also includes a wizard that converts your existing Panorama forms for use on the web. This wizard allows you to use Panorama's powerful graphic editing tools to create web based forms. The illustration below shows how this works. The screen shot on the left shows the original Panorama form. On the right is the nearly identical web based form.



Since web browsers don't have all the capabilities available in Panorama forms, some features cannot be translated. You may find that you want to create special forms specifically for use as web forms, just as you normally create special forms for printed reports. Nevertheless, you'll find that this capability allows you to leverage your existing skills (and much of your existing forms) to rapidly develop web based databases.

Database Tables on the Web

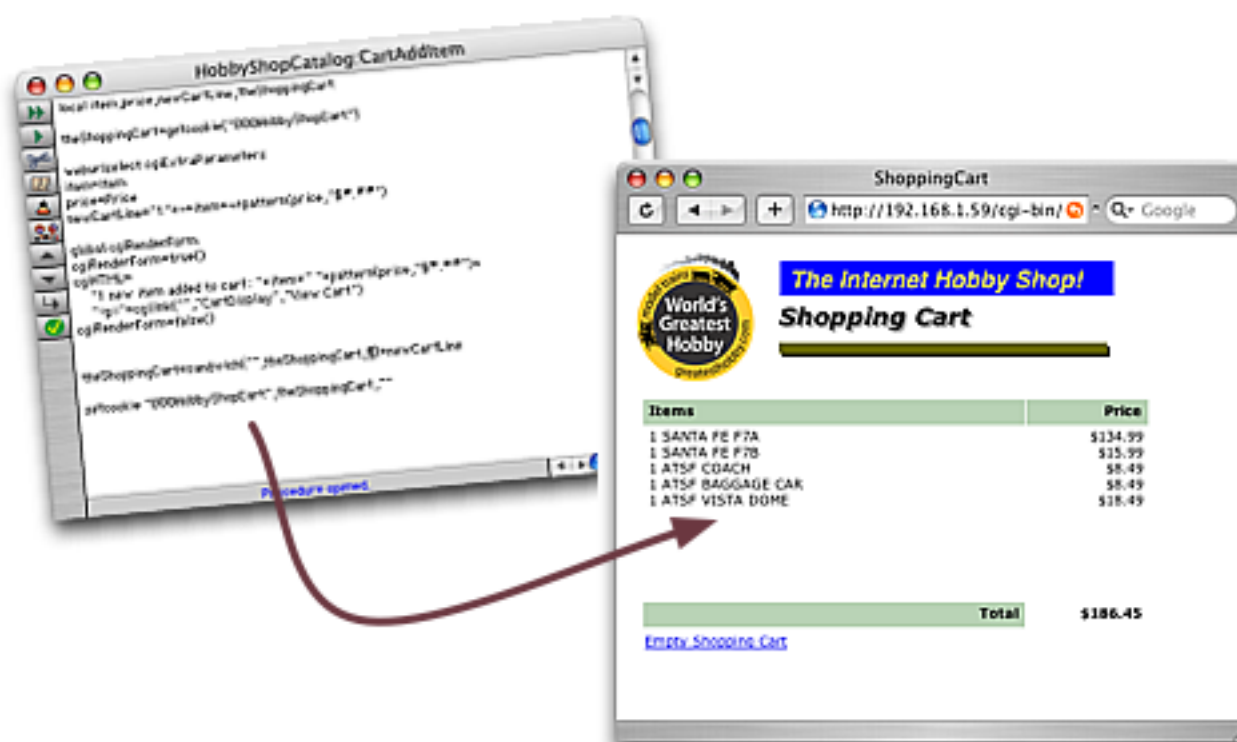
Panorama's form conversion wizard is great for displaying one record at a time. Another wizard makes it easy to display a web page with a table showing multiple records, like this price list.

| Scale | Manufacturer | Item | Price |
|-------|--------------|-----------------------------|-------|
| HO | Athearn | Santa Fe F7A | 34.99 |
| HO | Athearn | Santa Fe F7B | 15.99 |
| HO | Athearn | Santa Fe SD40-2 | 34.99 |
| HO | Athearn | Santa Fe F45 | 22.49 |
| HO | Athearn | Santa Fe 40' Reefer | 29.99 |
| HO | Athearn | Santa Fe 40' Pulp Flat | 14.99 |
| HO | Athearn | Santa Fe WideVision Caboose | 4.99 |
| HO | Athearn | Santa Fe GP60 | 5.49 |
| HO | Athearn | Santa Fe Caboose | 34.99 |
| HO | Athearn | Santa Fe GP50 | 5.49 |
| HO | Athearn | ATSF 50' Box (Grand Canyon) | 34.99 |
| HO | Athearn | ATSF 50' Box (Choir) | 4.49 |
| HO | Athearn | ATSF Tank Car | 4.49 |
| HO | Athearn | ATSF 4427 Covered Hopper | 4.49 |
| HO | Athearn | ATSF 50' Gondola | 20.79 |
| HO | Athearn | ATSF 40' Single Dome Tank | 5.49 |
| HO | Athearn | ATSF Vista Dome | 4.49 |
| HO | Athearn | ATSF Observation | 18.49 |
| HO | Athearn | ATSF Diner | 8.49 |
| HO | Athearn | ATSF Baggage Car | 8.49 |
| HO | Bachmann | Santa Fe 0-6-0 | 8.49 |
| HO | Walther's | Santa Fe SD70M | 39.95 |

The wizard makes it easy to set up this table in just a few minutes.

Custom Web Programming

The Panorama web publishing server can perform basic actions like searching the database, displaying tables, displaying forms, and data entry without any programming at all. By adding your own custom programming you can implement more advanced features. The illustration below shows a simple shopping cart based on a Panorama database. The short program on the left adds an item to the shopping cart.



With Panorama's powerful programming language the possibilities are limited only by your imagination!

Chapter 2: Installation, Configuration & Management



This chapter explains how to set up and manage your Panorama Enterprise Edition Server. Topics covered include system preparation, server installation, server configuration and administration.

System Requirements

Before you begin, make sure that your system meets the minimum requirements. The Panorama Server requires a Macintosh computer with **OS X 10.4 (Tiger)** or later. **OS X Server** is **not** required, but can also be used. (However, if you don't have any specific reason to use **OS X Server**, stick to regular **OS X**. Using regular OS X will save you money and is easier to configure than OS X Server. (If you do use OS X Server be sure to read "[Setting Up The User Account](#)" on page 33 which contains an important tip for OS X Server configuration.) For light to medium duty applications we've found that a Mac Mini makes an excellent and affordable server (either the PPC or Intel version).



For extremely light duty, it is possible to run the server on a machine that is also being used as a client. We generally don't recommend this configuration (especially if there are more than a couple of clients). Most of the discussion in this chapter will assume that the server is running on a separate, dedicated computer.

Of course the server computer must have a permanent connection to the network you are using. If you plan to do web publishing or share databases over the internet then you'll also need a static IP address for your server. Static IP addresses are discussed in more detail later in this chapter.

System Preparation

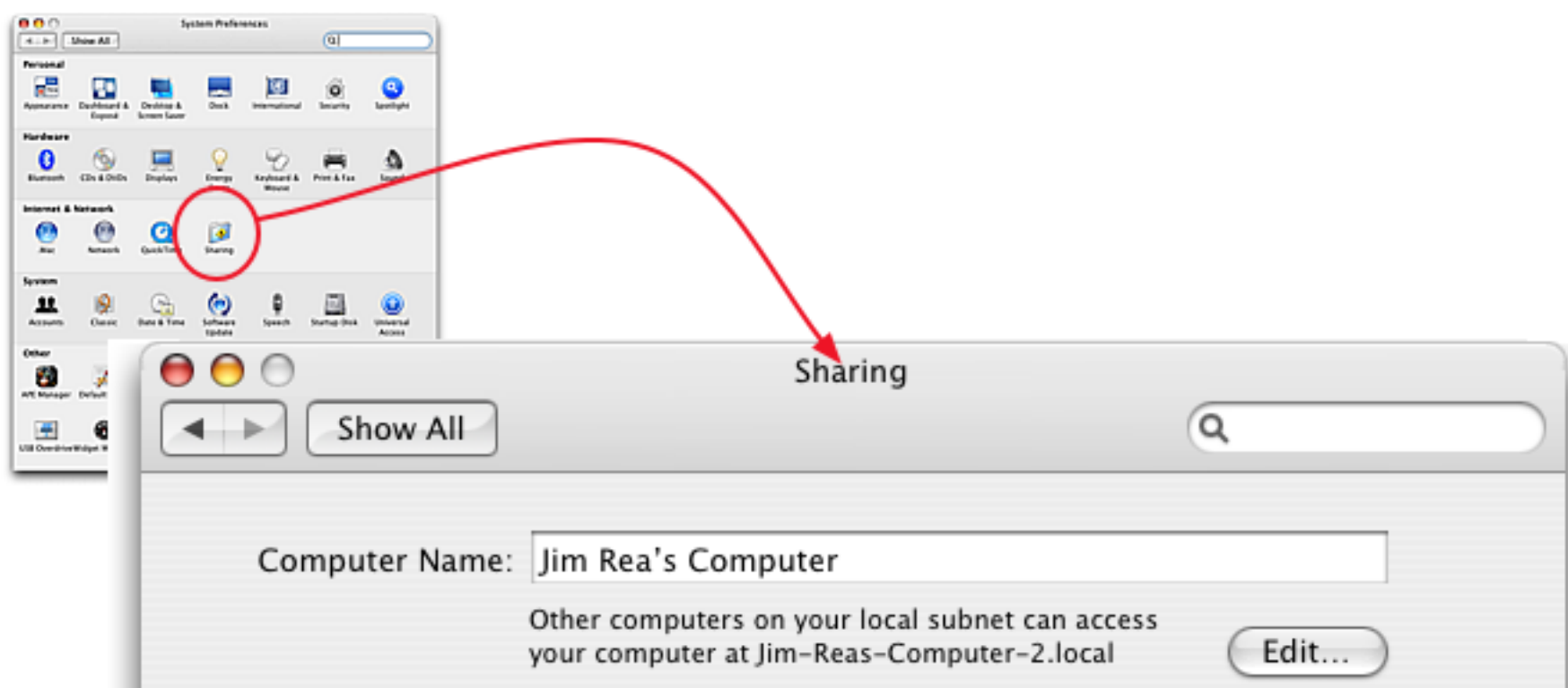
Before you can begin using the Panorama server, you'll need to properly configure your operating system. Most of this configuration is done using the OS X **System Preferences** application which you can launch from the dock.



Setting up the Computer Name

Every Macintosh computer has a name that identifies that computer on the local network. Mac OS normally sets this name up for you the first time you log onto the computer. You may, however, want to assign a more descriptive name. (If possible, it's better to stick with shorter computer names — some users have reported problems with names longer than 20 characters and/or names with unusual characters or punctuation.)

You must make absolutely sure that the server computer has a unique name that is not used by any other computer on the network. This can happen several ways: 1) If the computer is not connected to the network when it is first set up, and is later connected to the network. 2) If another computer added to the network and is already set up with a duplicate name. 3) If a computer is set up in a non-standard way, for example by using a disk cloning utility to clone the configuration of a machine already on the network. To check the name and assign a non-duplicate name use the **Sharing** panel in the System Preferences application.

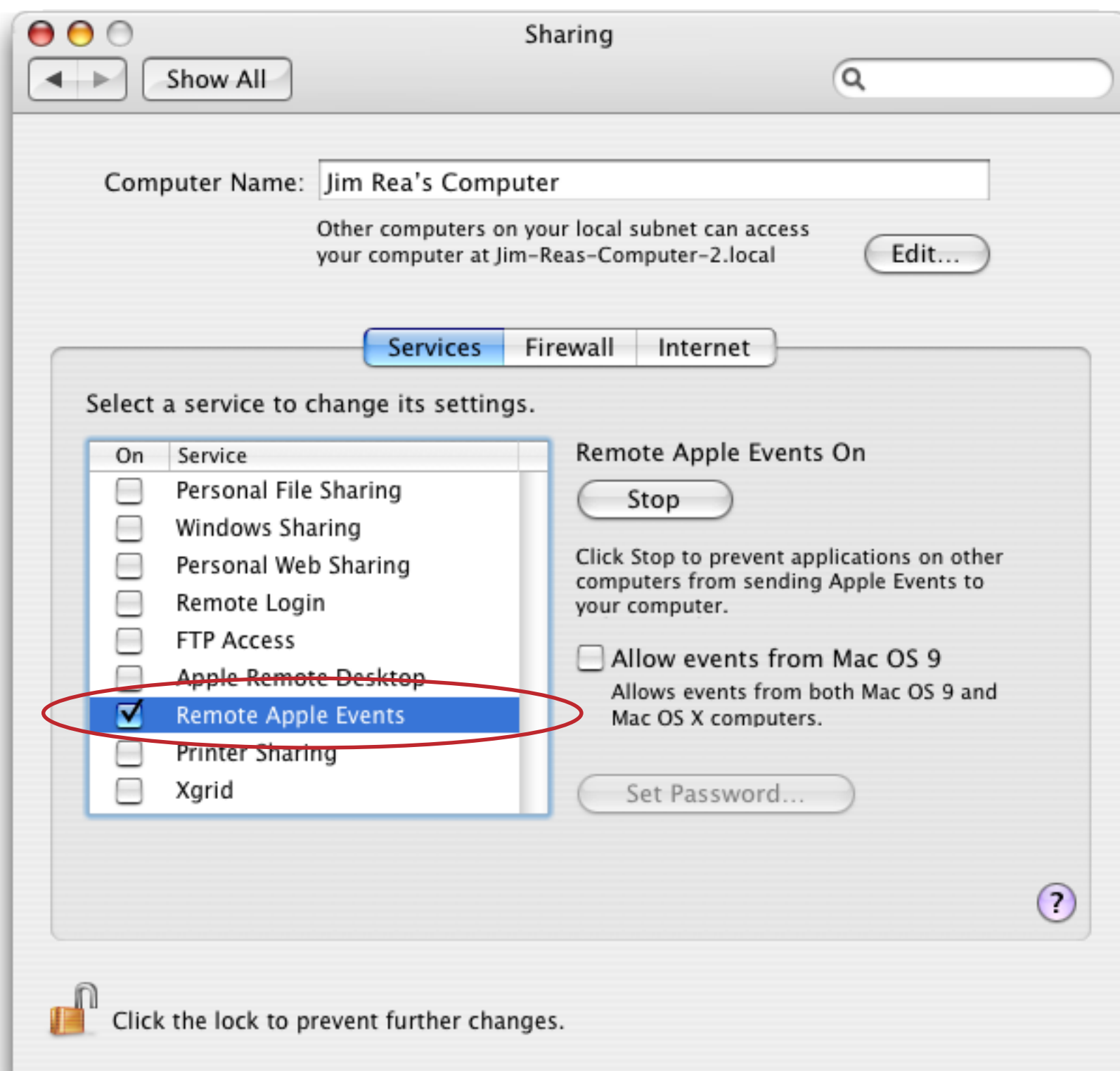


If you ever encounter problems communicating between computers on the local network and the server, one thing to check is that no computers have been added to the network with the same name as the server computer.

Note: If you ever need to change the name of the server computer, you should stop and then re-start the Panorama server's local sharing option (see "[Enabling Database Sharing](#)" on page 55).

Enable Remote Apple Events

If you are planning on accessing the server from a computer on the local network, you must enable the **Remote Apple Events** option in the sharing panel.



It is not necessary to enable the **Allow events from Mac OS 9** option.

Setting Up The User Account

The Panorama server must run on an account with administrator privileges. There's nothing special about this account, any account with administrator privileges may be used, with one exception: the account must not be named **root** or **admin**. A bug in OS X prevents Remote Apple Events from working with an account named either **root** or **admin**. This is especially an issue with OS X Server since the installer for OS X Server usually sets up a default account with one of these names. If you use OS X Server be sure to double check that you are not using **root** or **admin** as the name of the account used to run the server. (This bug has been confirmed with Apple for OS X 10.3 and 10.4, but may be fixed in future versions of OS X.)

A Brief Introduction to Static IP Addresses

If you are planning to access your server over the internet the server must have a static IP address. A static IP address never changes, so anyone wanting to contact your server will be able to do so at the same IP address every time — today, tomorrow and next year. (An IP address is four numbers separated by dots, for example [192.168.1.23](#)).

When the internet first started, every IP address was a static IP address, so this wasn't an issue. However, as the internet expanded there weren't enough numbers to go around. So now most IP addresses are dynamic — which means they are temporarily assigned as needed. If you have a dynamic IP address, your address today may be different from the one you have an hour from now, not to mention tomorrow or next week. This works fine if you are simply accessing other servers (for example web browsing) but not so well if you want to have your own server (because other users don't know where to find you at any given moment).

So, to run your own server you'll need your own static IP address. To get one you'll need to talk to your ISP. At a minimum they'll probably ask you to pay an extra fee. Worst case you may need to switch to a different ISP, since some don't allow static IP's at any price (especially cable companies, although some cable companies have begun to offer static IP address - usually you have to pay for "business" rather than "residential" service). So before you go any further, go knock on your ISP's door and arrange to get a static IP address. We'll wait for you to get back...



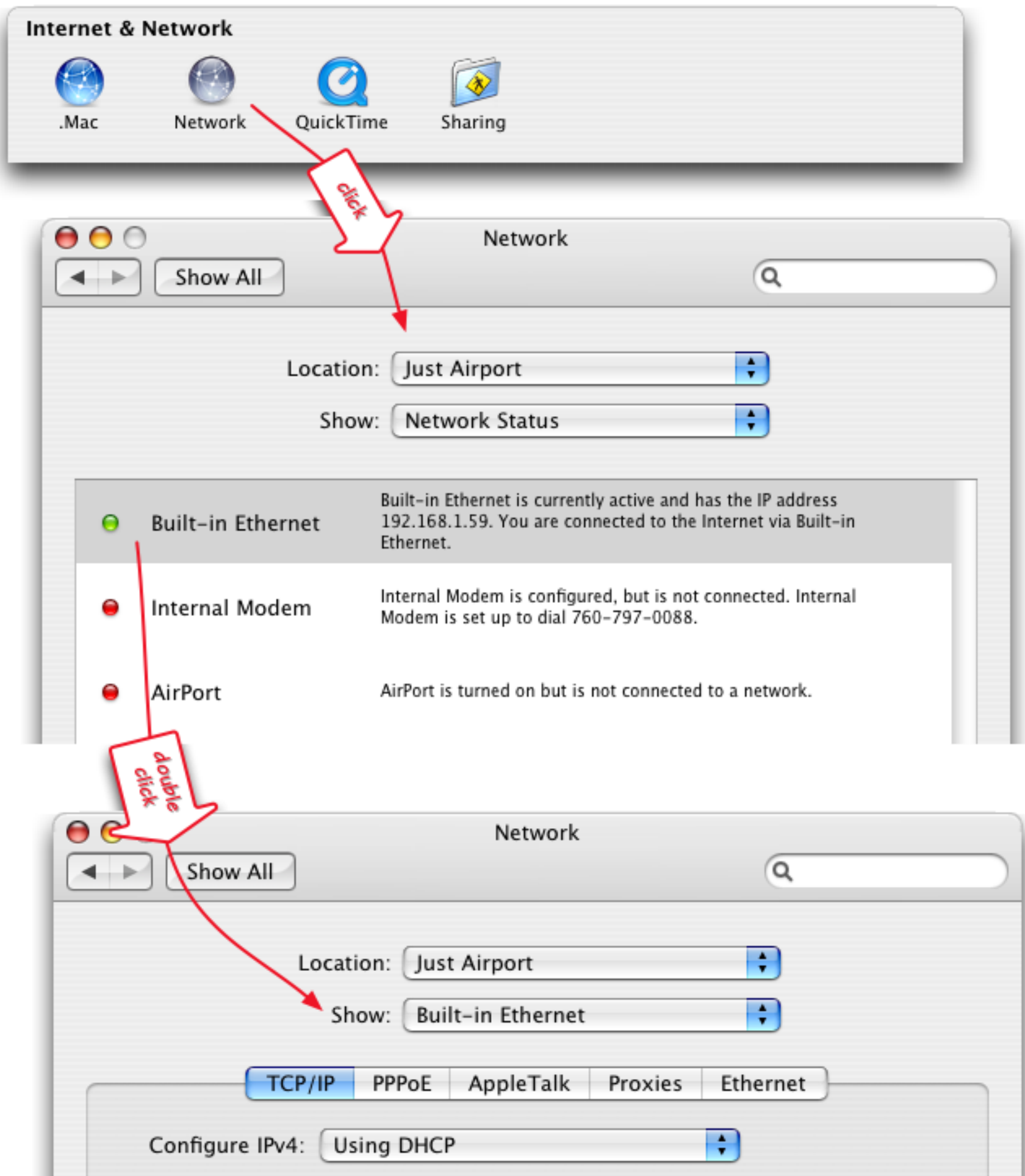
Ok, got that static IP address? Your ISP actually should have given you several numbers — an IP address, a subnet mask, a router (sometimes called gateway) and a couple of dns servers.

Setting up a Server Directly Connected to the Internet

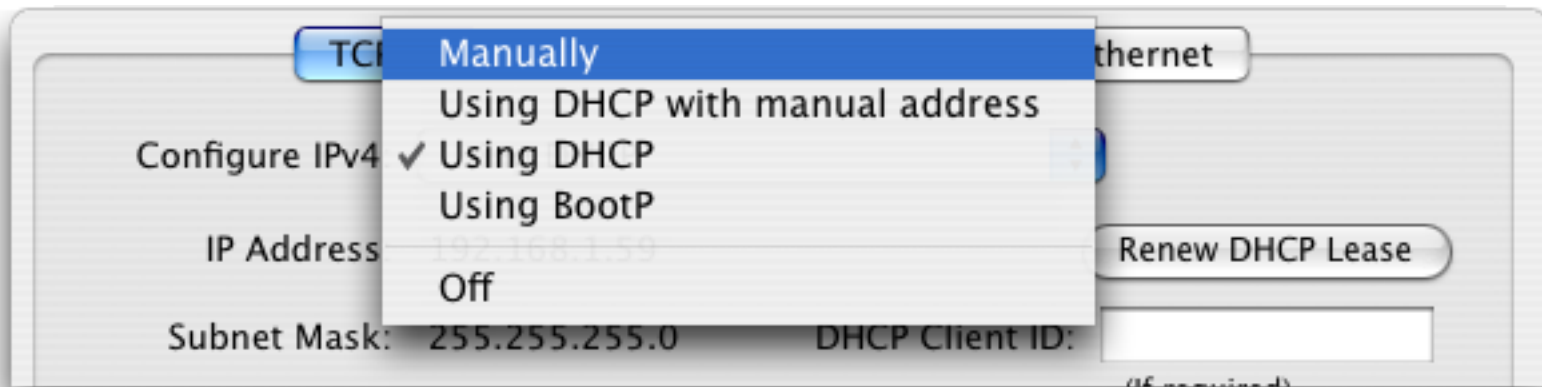
If your server computer is directly connected to the internet (without a router, as shown in the diagram below.) then IP configuration is relatively easy. (If you are using a router then skip to the next section.)



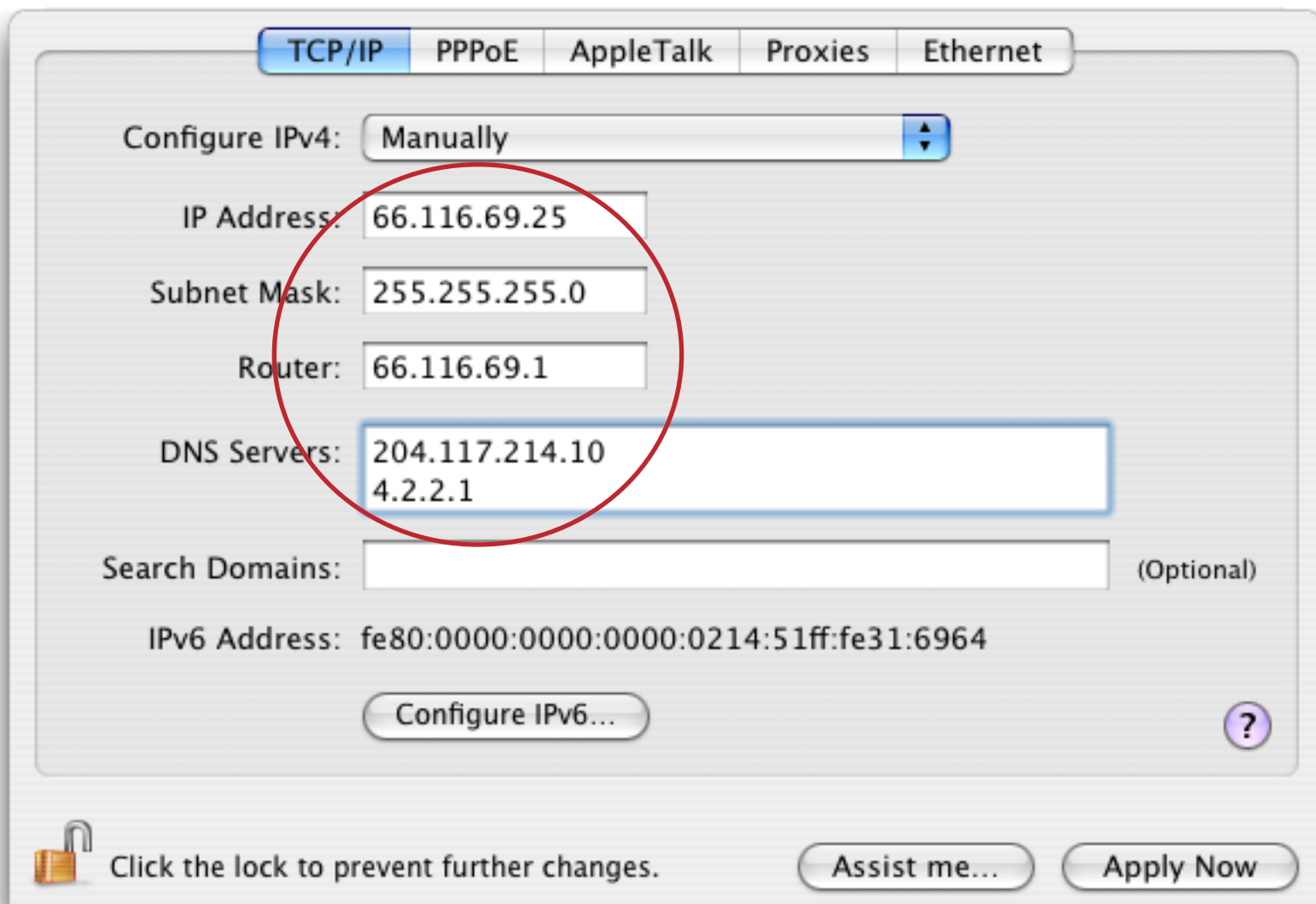
The first step is to open the **Built-in Ethernet** configuration panel. To do this first click on the **Network** icon, then double click on the **Built-in Ethernet** option.



Now use the pop-up menu to switch the **Configure IPv4** setting to **Manually**.



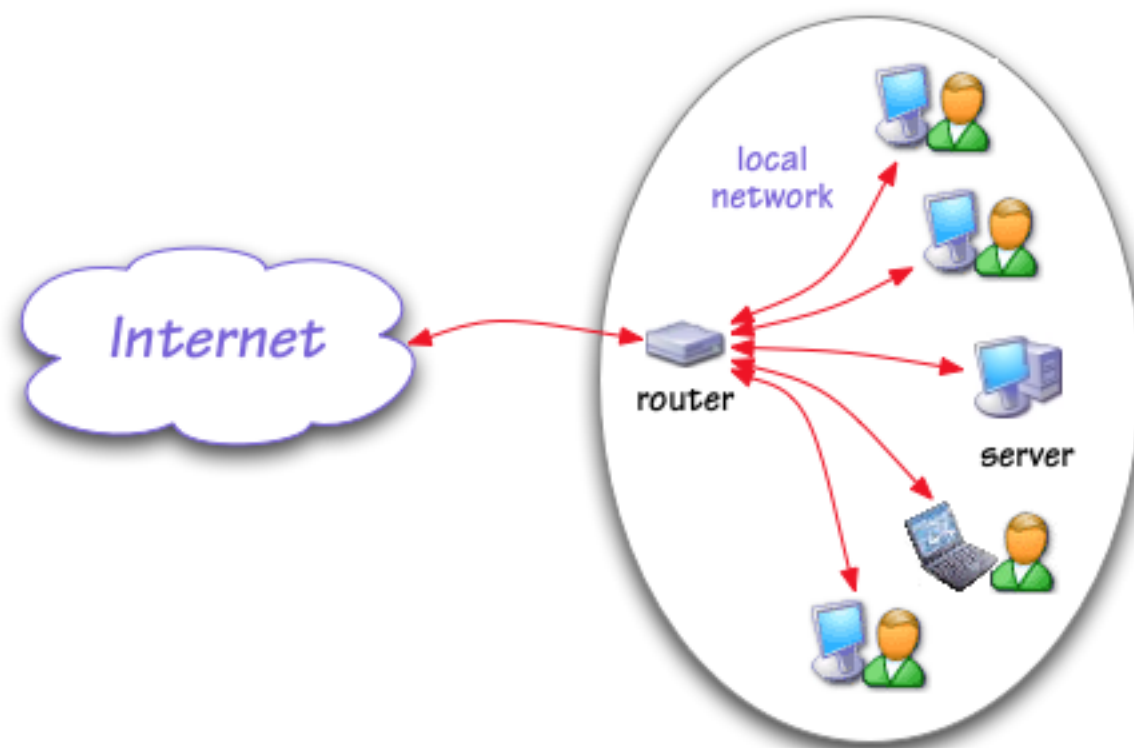
Now fill in the IP Address, Subnet Mask, Router and DNS Server information you got from your ISP.



When you're done, check everything twice and press the **Apply Now** button. Your static IP is now ready to use.

Setting up a Server Connected to the Internet using a Router

A more common configuration is for the server computer to be connected to the internet through a router, as shown in the diagram below.



In this configuration, typically all of the local computers will get their IP address automatically from the router (this is called DHCP, or dynamic host control protocol), while the router automatically gets a dynamic IP address from the ISP. Converting this to use a static IP is a three step process.

- 1) Set up the router to use the static IP address information provided by the ISP.
- 2) Set up server computer to use DHCP but with a manually assigned static IP address. This static IP address is not the one given to you by your ISP, but is determined by your router configuration.
- 3) Configure the router to pass incoming port 80 requests through to port 80 on your server computer (sometimes called “port-forwarding”).

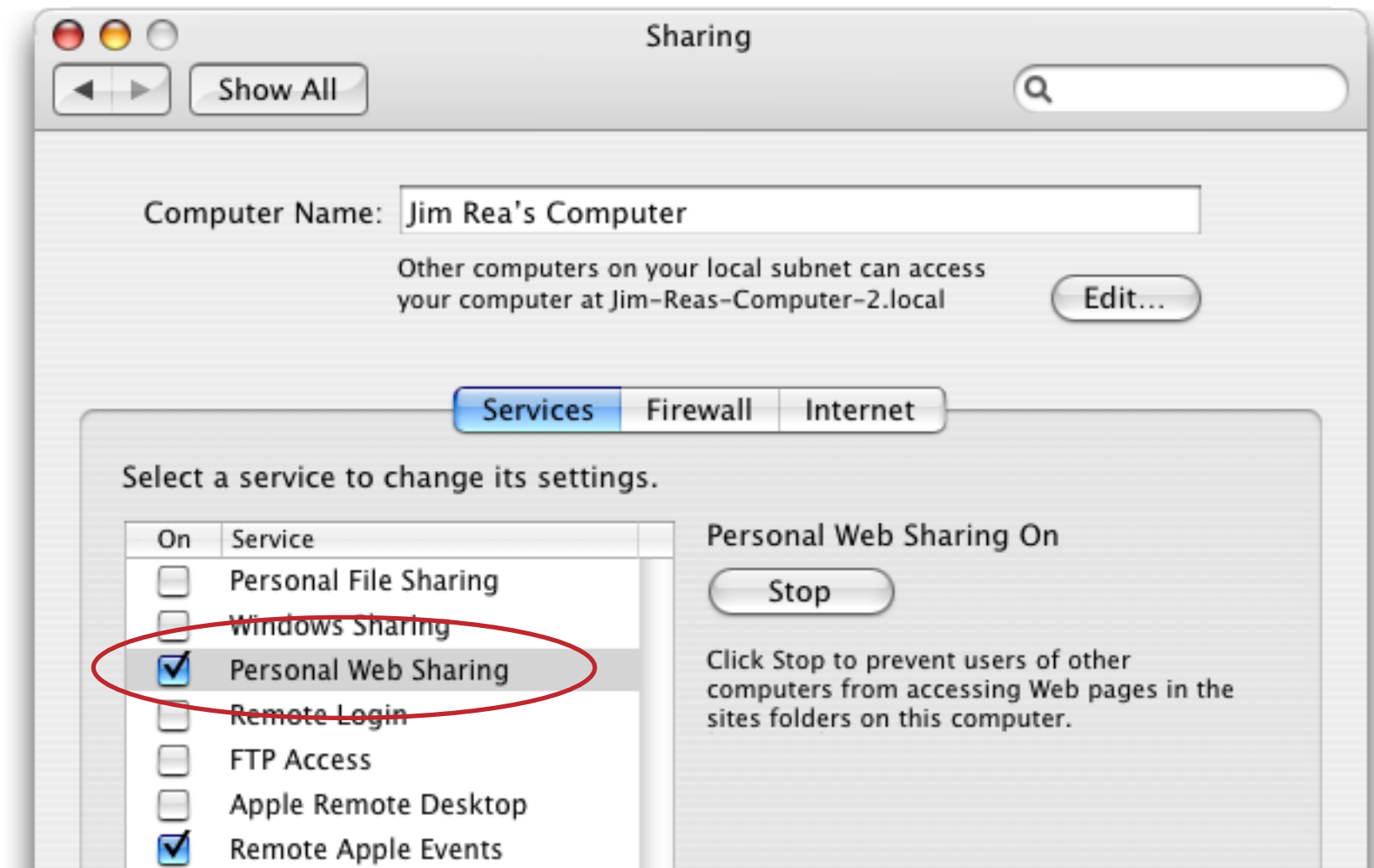
If your network has a System Administrator or Network IT staff, you’ll want to consult them. If you are your own System Administrator, you’ll need to consult the documentation that came with your router to learn how to set up your particular router (unfortunately, every one is different). You may also want to refer to the web site www.portforward.com, which has instructions and help for dozens of popular routers.



Note: ProVUE Development has no connection with the www.portforward.com web site.

Enabling Internet Sharing

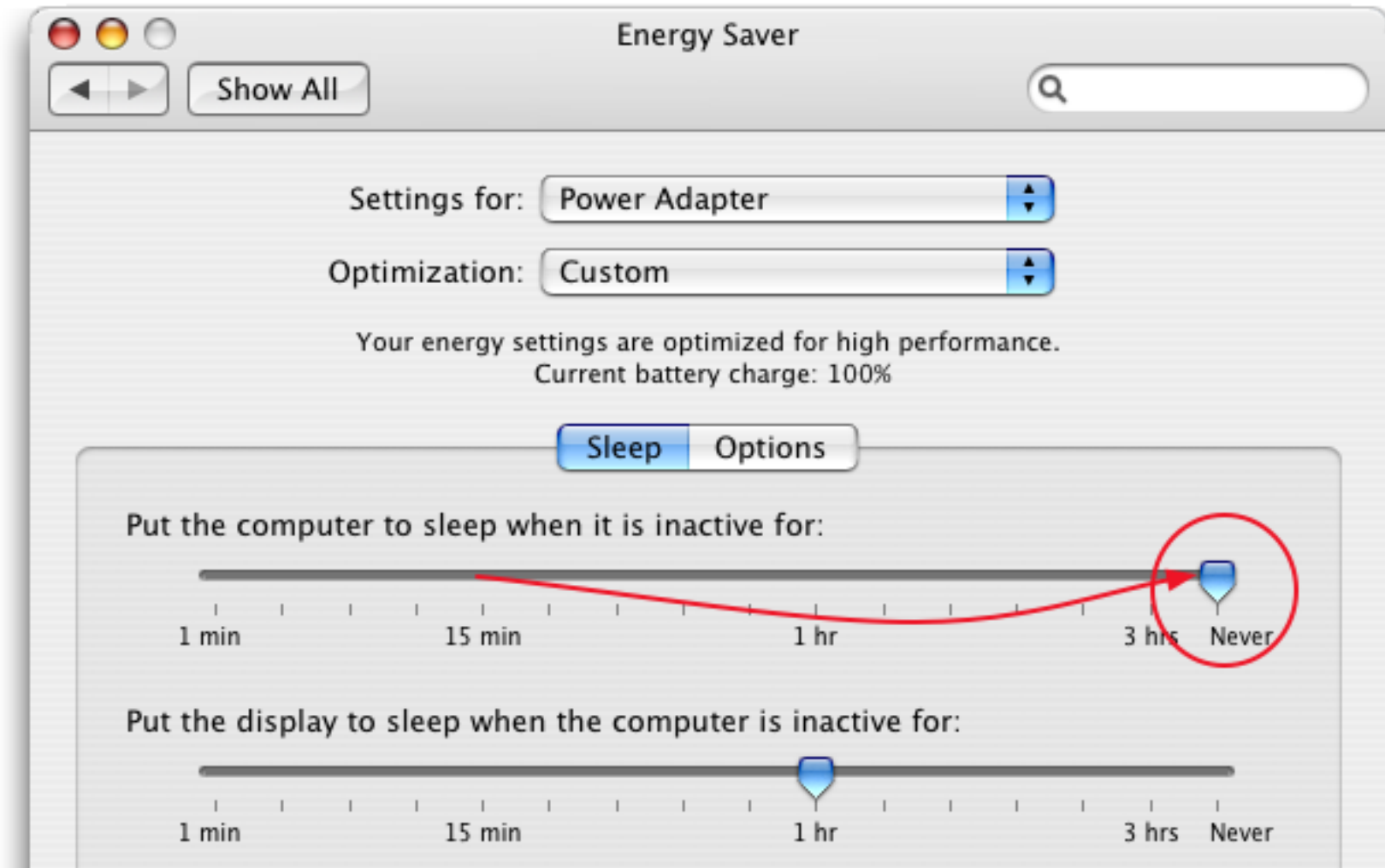
If you intend to share databases using the internet or to publish a database on the web, you must enable the **Personal Web Sharing** option in the **Sharing** pane of the **System Preferences** application.



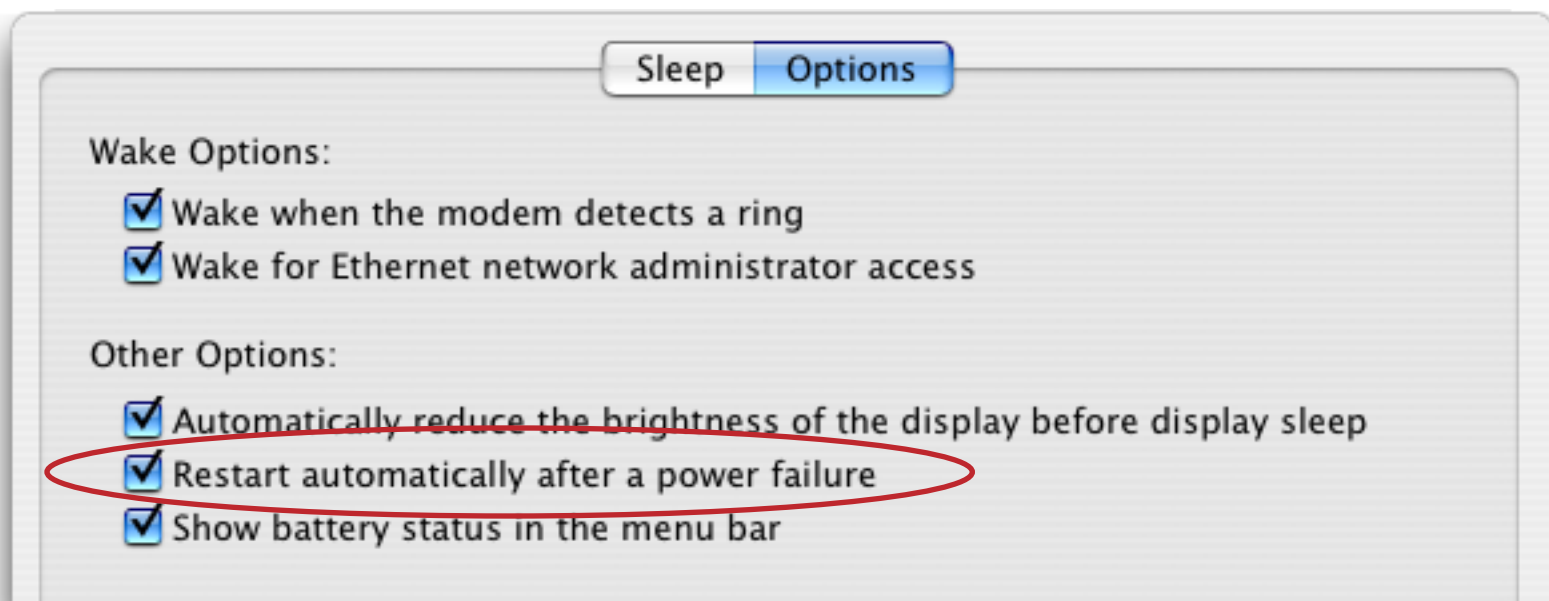
Note: Turning on this option actually starts the Apache web server that is built-in to every copy of OS X. The Panorama server uses Apache as a front end for all TCP/IP (internet) communications. Of course you can also use this copy of Apache to host your entire web site, not just the Panorama databases. The Apache server has earned such a reputation for rock-solid reliability that it currently hosts over half the websites on the Internet.

Ensuring Nonstop Server Operation

If you are setting up a production server (as opposed to in-house testing) you'll want to make sure that your server doesn't stop when it's been inactive for a while, and also make sure that it restarts after a power failure. These options are controlled in the **Energy Saver** pane of the **System Preferences** application. To make sure that the computer doesn't shut down after a period of activity, set the sleep option to **Never**. (It's generally ok to let the display go to sleep, however.)



Switch to the Options panel and check the **Restart automatically after a power failure** option. With this option enabled, the computer will turn itself back on automatically when power is restored. (Of course for 24/7 reliability you will probably want to use an uninterruptable power supply for your server and routers.)



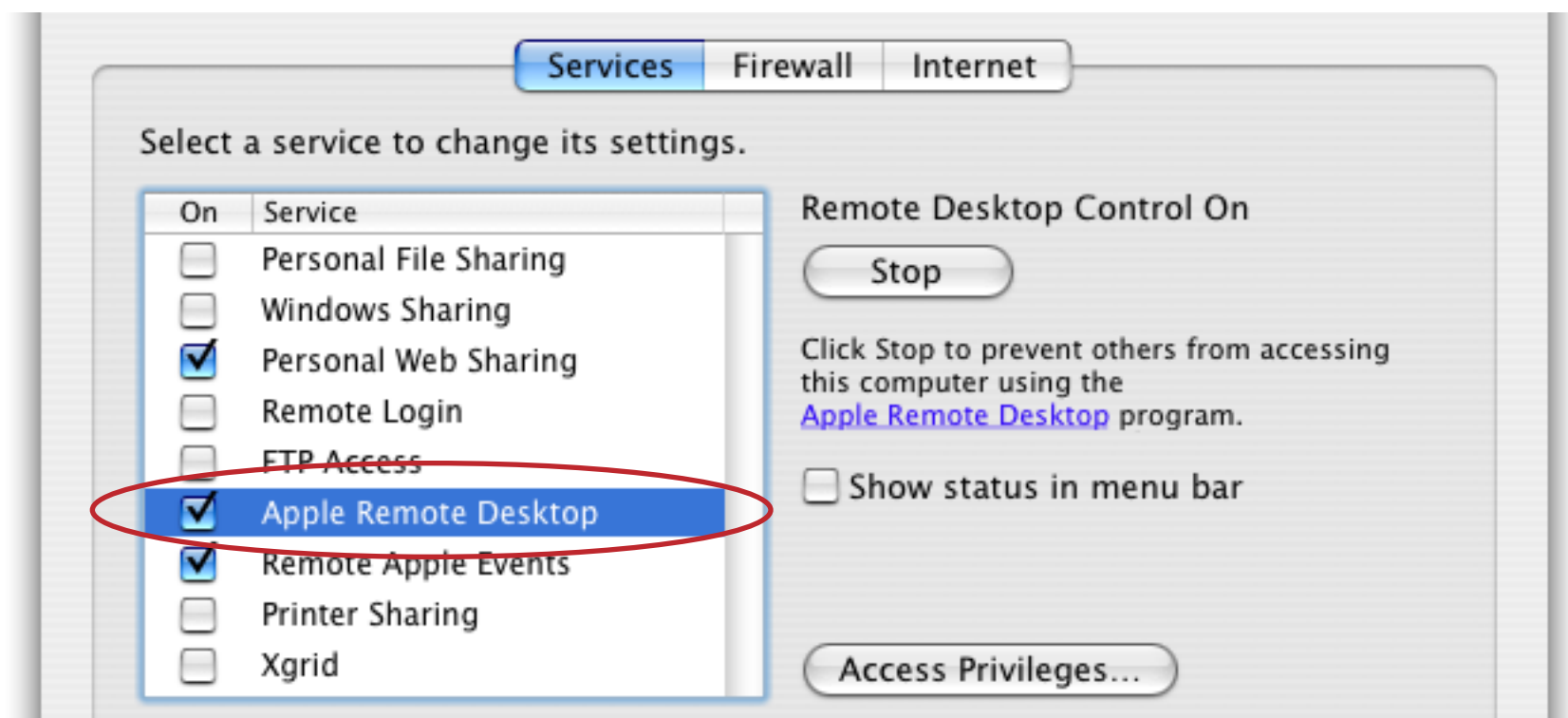
You'll also want to make sure that the Panorama Server launches automatically when the computer starts up. Of course you can't do that until the server is installed. See "[Setting up Panorama Server to Launch Automatically](#)" on page 47.

Enabling Remote Server Operation

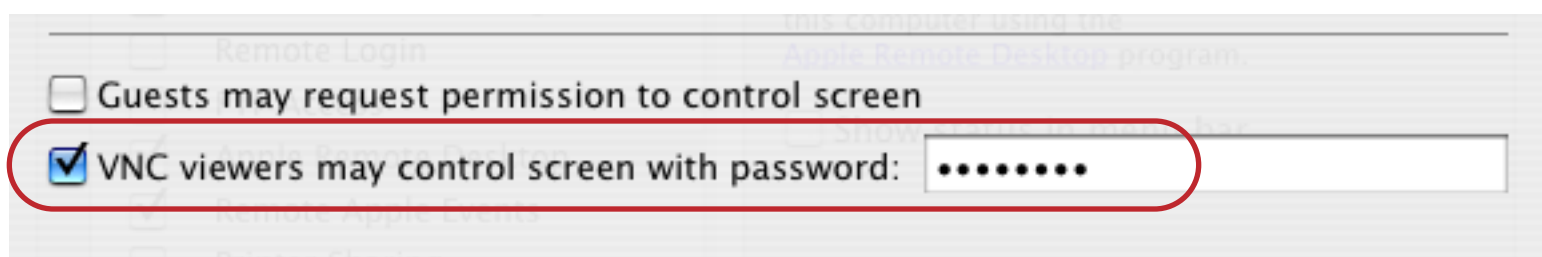
If your server is in a remote location (for example at your ISP) you may want to set up a method for operating the server by “remote control.” This will save you a trip in your car (or plane?) if you need to change something on the server. Although your Panorama server should rarely or never need to be operated remotely once it is set up, you may want to consider a backup remote control method just in case. There are three packages that allow a computer to be operated by remote control.

| Package | Vendor | URL |
|-----------------------------|-------------|---|
| Timbuktu® Pro | Netopia | http://www.netopia.com/software/products/tb2/ |
| Apple Remote Desktop | Apple | http://www.apple.com/remotedesktop/ |
| VNC | Open Source | http://sourceforge.net/projects/cotvnc/ |

Both of the commercial packages are highly rated and come with complete documentation. VNC is not quite as fancy as these two, but it is free and works well, especially on a local network. If you are using OS X 10.4 or later then a VNC server comes built-in — all you have to do is turn it on! You can turn this on from the **Sharing** panel of the **System Preferences** application, as shown below.

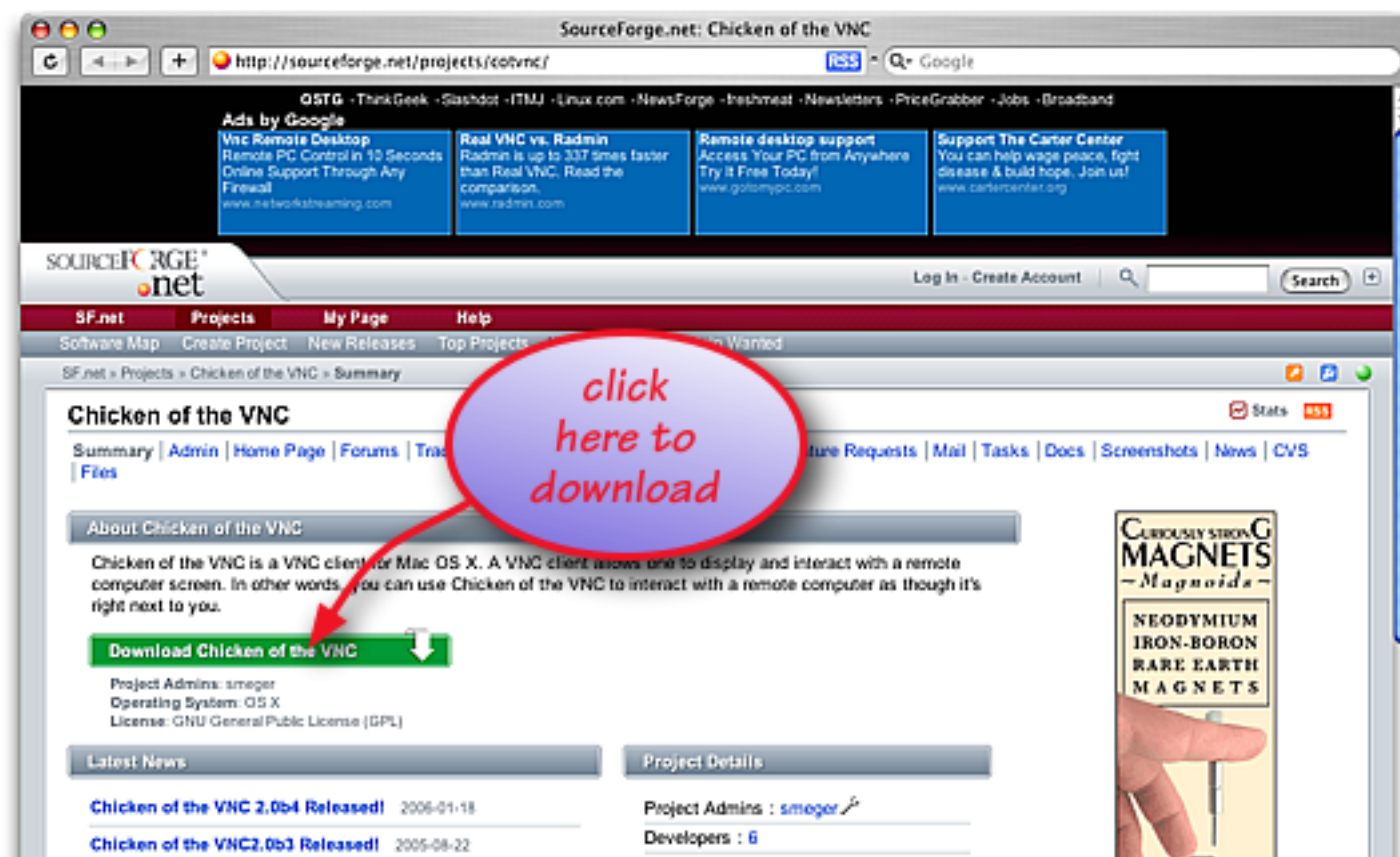


Once Remote Desktop Control is on, press the **Access Privileges** button, which opens a separate dialog sheet. This sheet contains a number of options that we’re not interested in. All you need to do is check the **VNC** box and enter a password, as shown here.



That’s all you need to do on the server side.

On the client computer (the one you are going to use to remotely control the server) you'll need to download and install the free open source "Chicken of the VNC" client. At the time this was written, the download page (<http://sourceforge.net/projects/cotvnc/>) looked like this.



After you download the application copy it into your Applications folder.

To remotely control your server, start by double clicking on the **Chicken of the VNC** application.



Chicken of the VNC.app

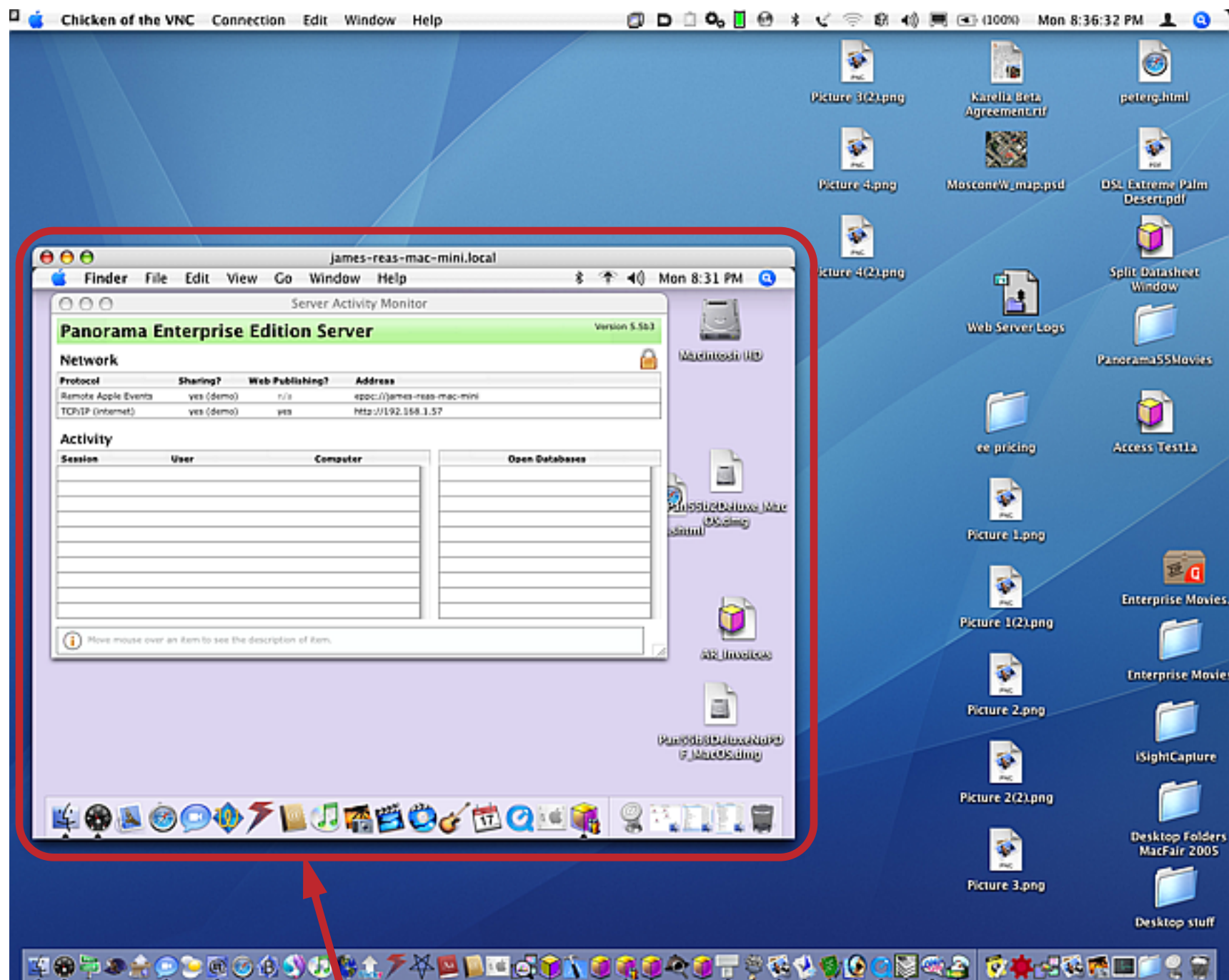
Select **New Connection** from the **Connection** menu. In the **Host** field enter either the IP address or the name of the server computer, then enter the password.



enter the IP address, domain name, or computer name of the server computer (the computer you want to control)

enter the VNC password you just set up in the Access Privileges sheet on the server (see illustration earlier in this section).

Now press the **Connect** button to remotely operate the server computer. You can operate the remote computer just as if you were sitting in front of it.

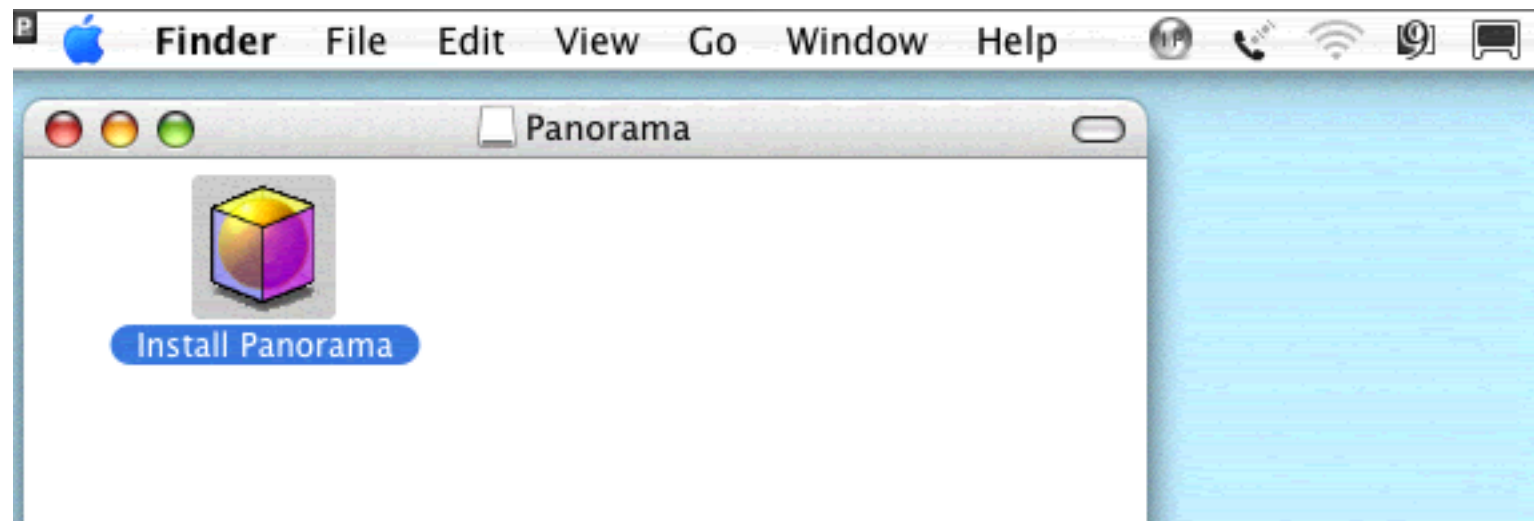


remote controlled computer appears in its own window.

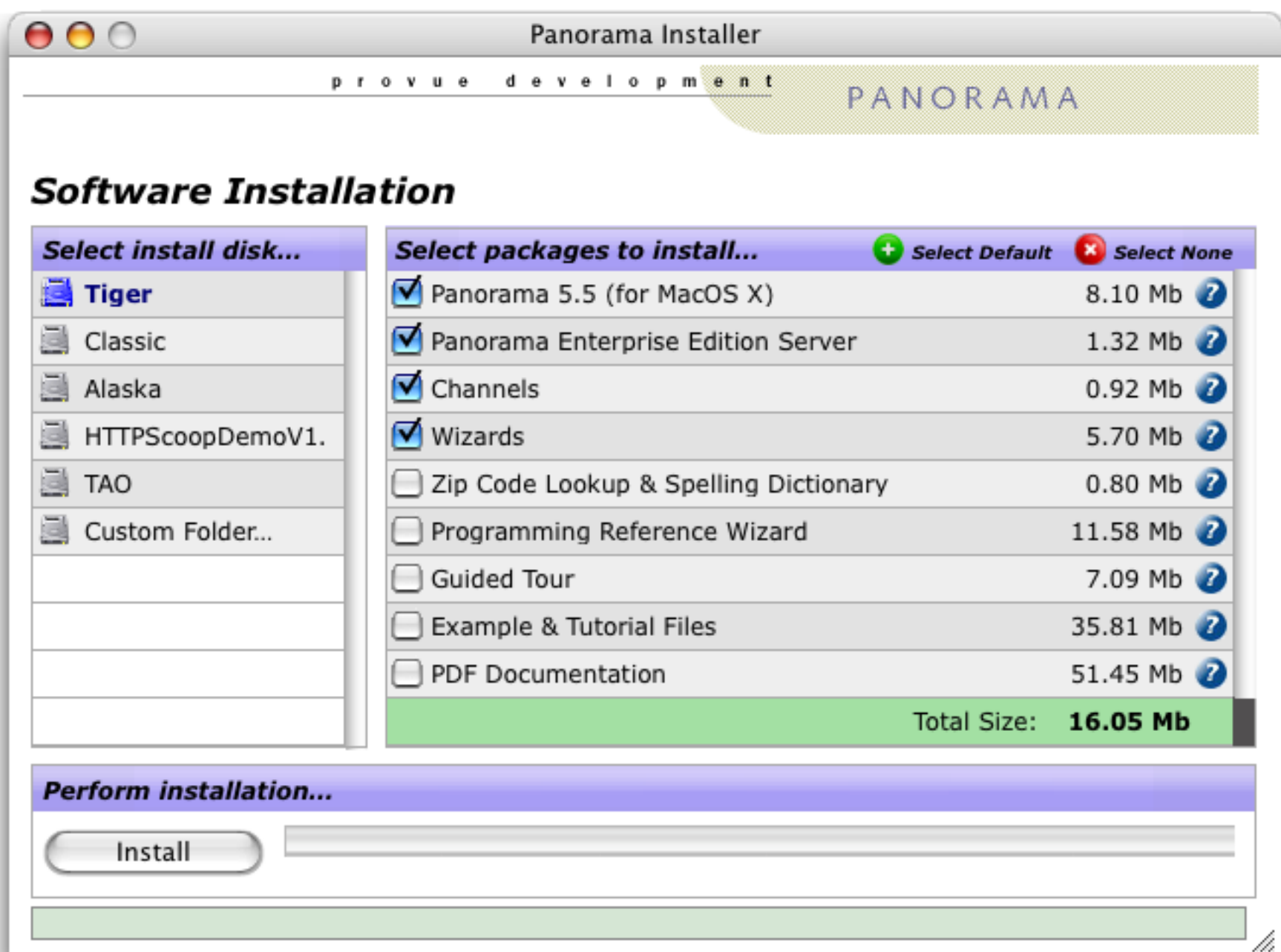
When you are done, simply close the remote window (or quit the **Chicken of the VNC** application).

Installing the Panorama Server Software

Now you're ready to actually install the Panorama server software. If you haven't already done so, download Panorama from the www.provue.com web site or insert the Panorama installation CD into the server computer.



Double click to launch the Panorama installer. Make sure you have the first four packages selected, as shown below. The remaining options aren't necessary for server operation, but won't hurt anything if the disk space is available.

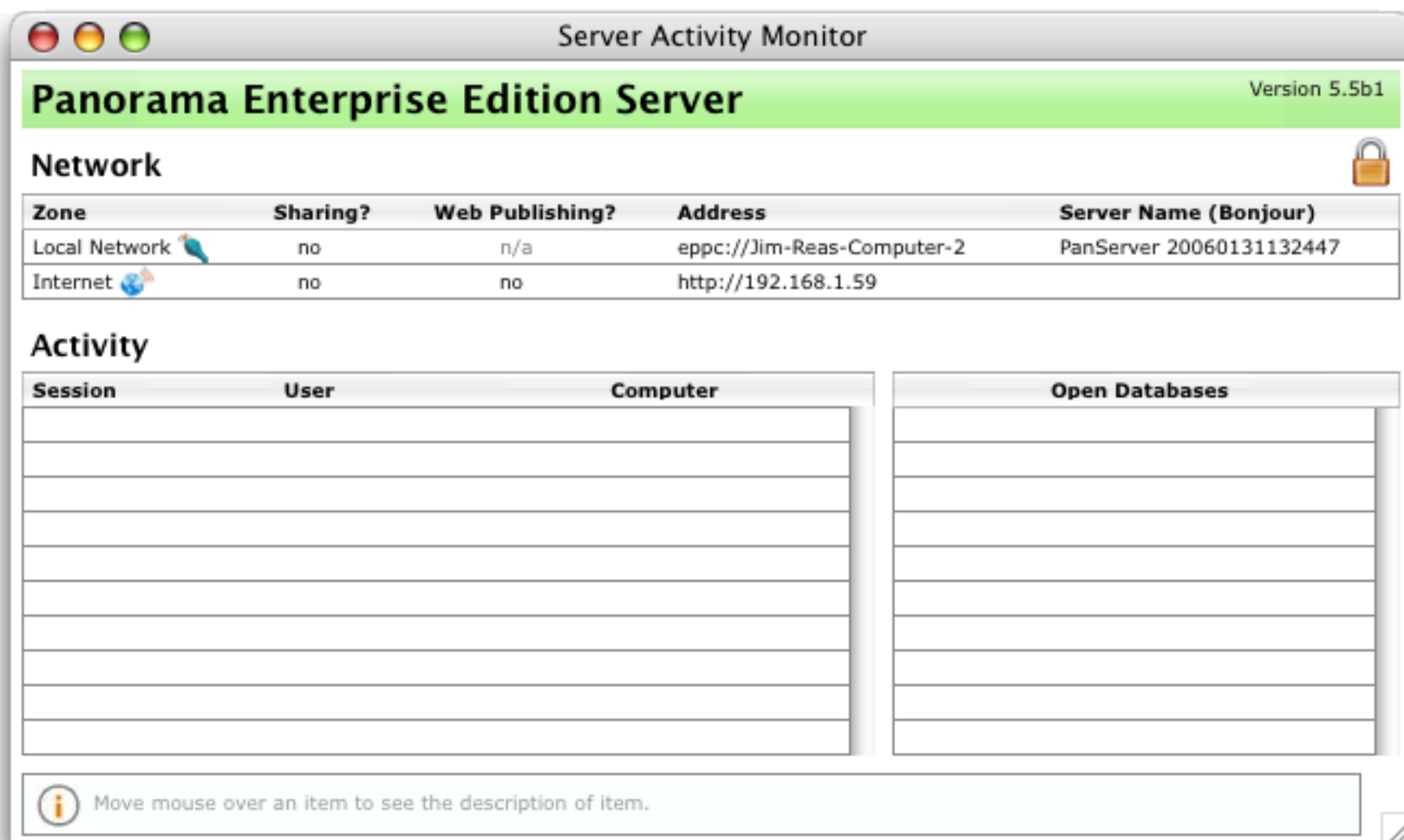


Press the **Install** button when the options you want are selected. If you've already purchased a serial number and product codes, you can enter the serial number at the end of the installation process to activate the software. Both a regular copy of Panorama and the Panorama Server will be installed. (The regular copy of Panorama is used for some configuration tasks, for example product activation or setting up an e-mail channel.)



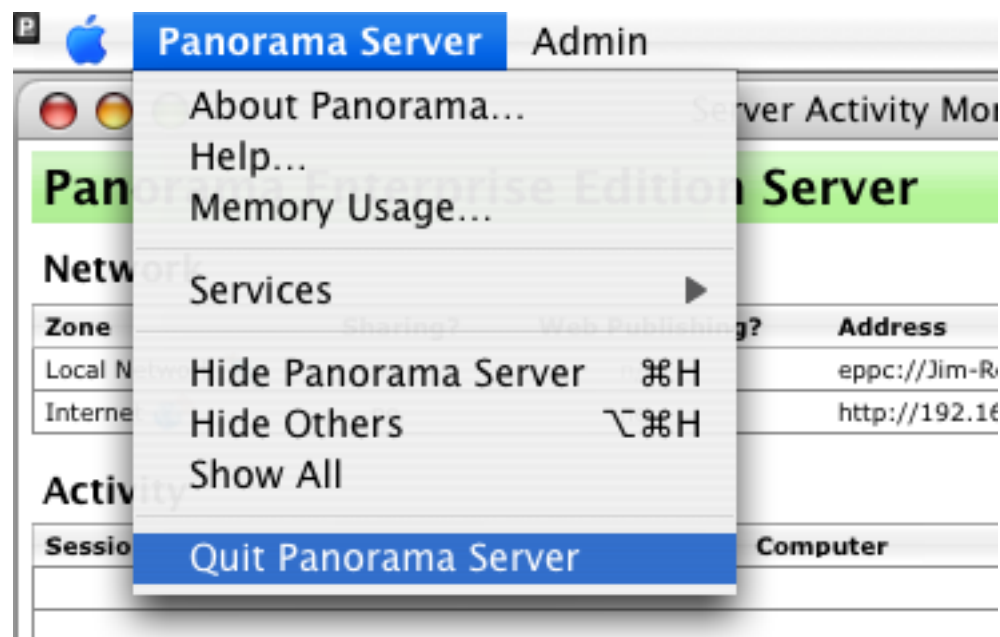
Launching the Server

To manually launch the server, simply double click on the Panorama Server icon. The server will open and the Server Activity Monitor window will appear.

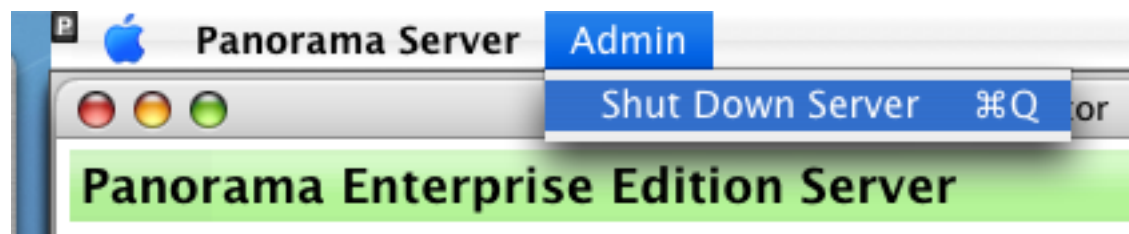


Shutting Down the Server

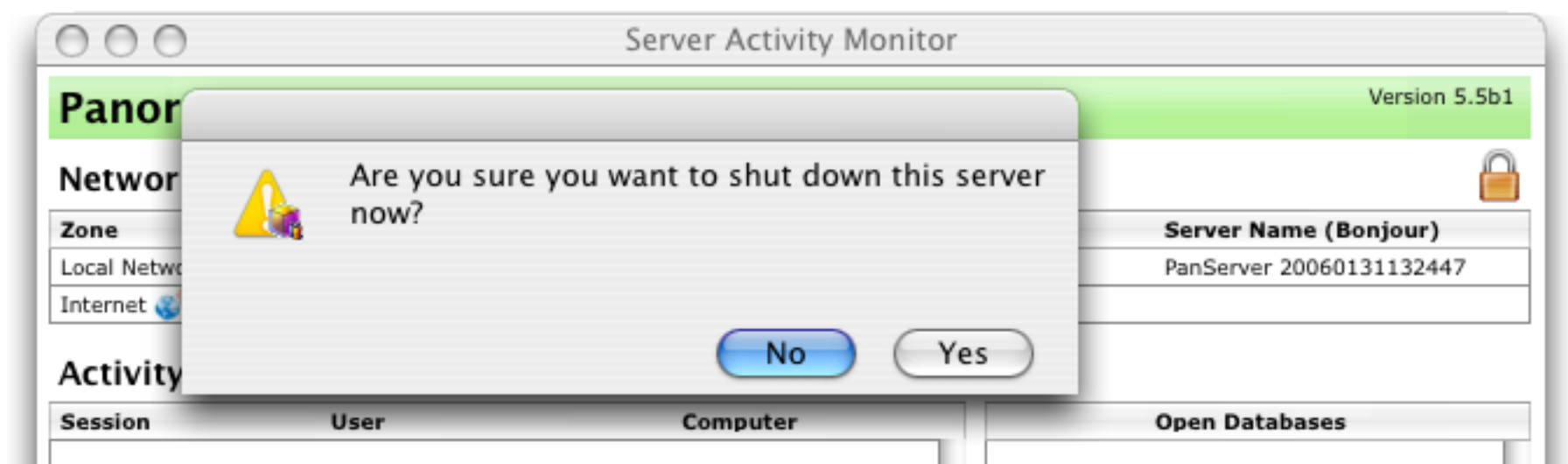
If you need to shut down the server, choose **Quit Panorama Server** from the **Panorama Server** menu.



Another option is to choose **Shut Down Server** from the **Admin** menu (these commands are identical).



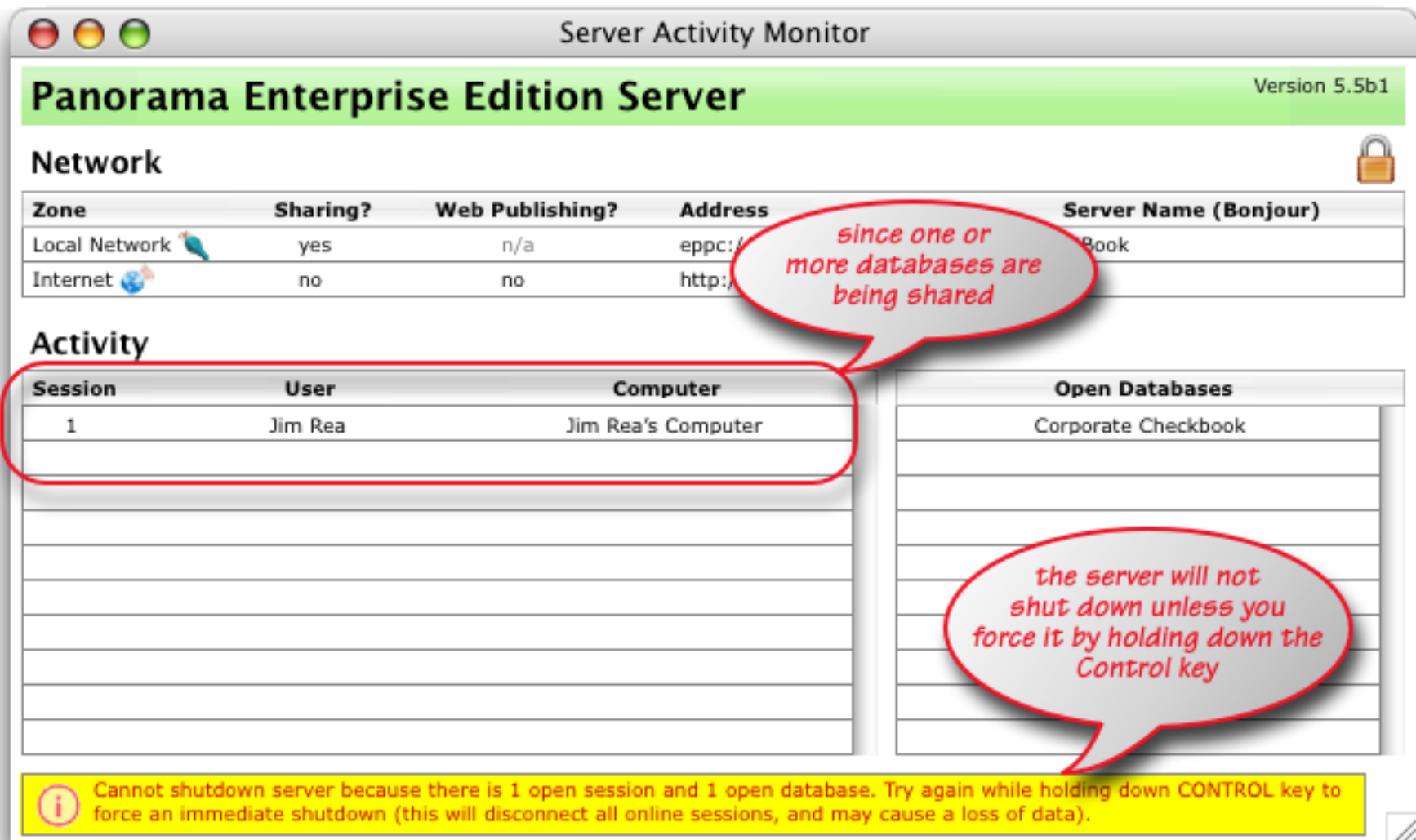
Whichever command you use, you will be asked to confirm that you really do want to shut down the server.



Press **Yes** if you really want to shut down the server. (You can configure the server to skip this confirmation and simply shut the server down, see “[Ask to Confirm Before Quitting Server](#)” on page 93).

Shutting Down the Server with Open Sharing Sessions

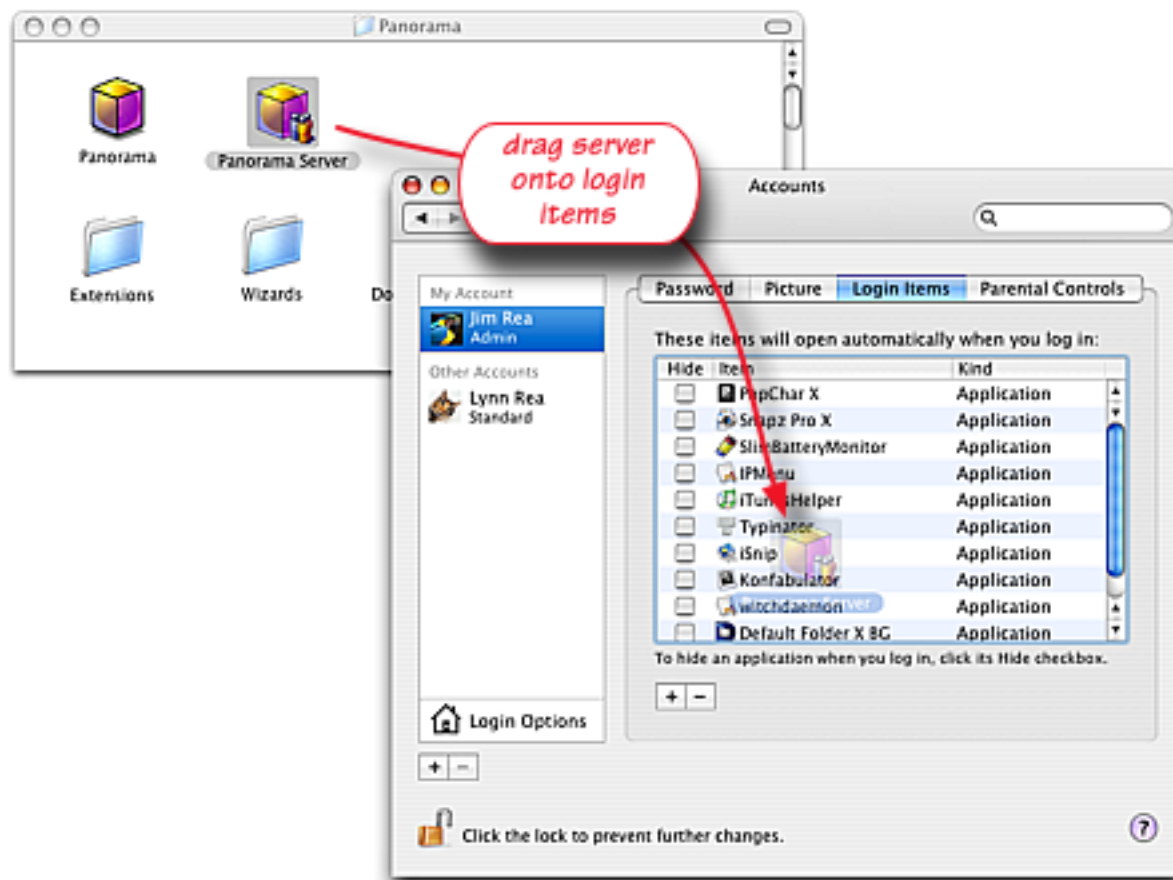
If the server is currently sharing one or more databases, it will not allow you to shut down the server. If you try to quit or shut down, it will beep and display an error message in red on a yellow background.



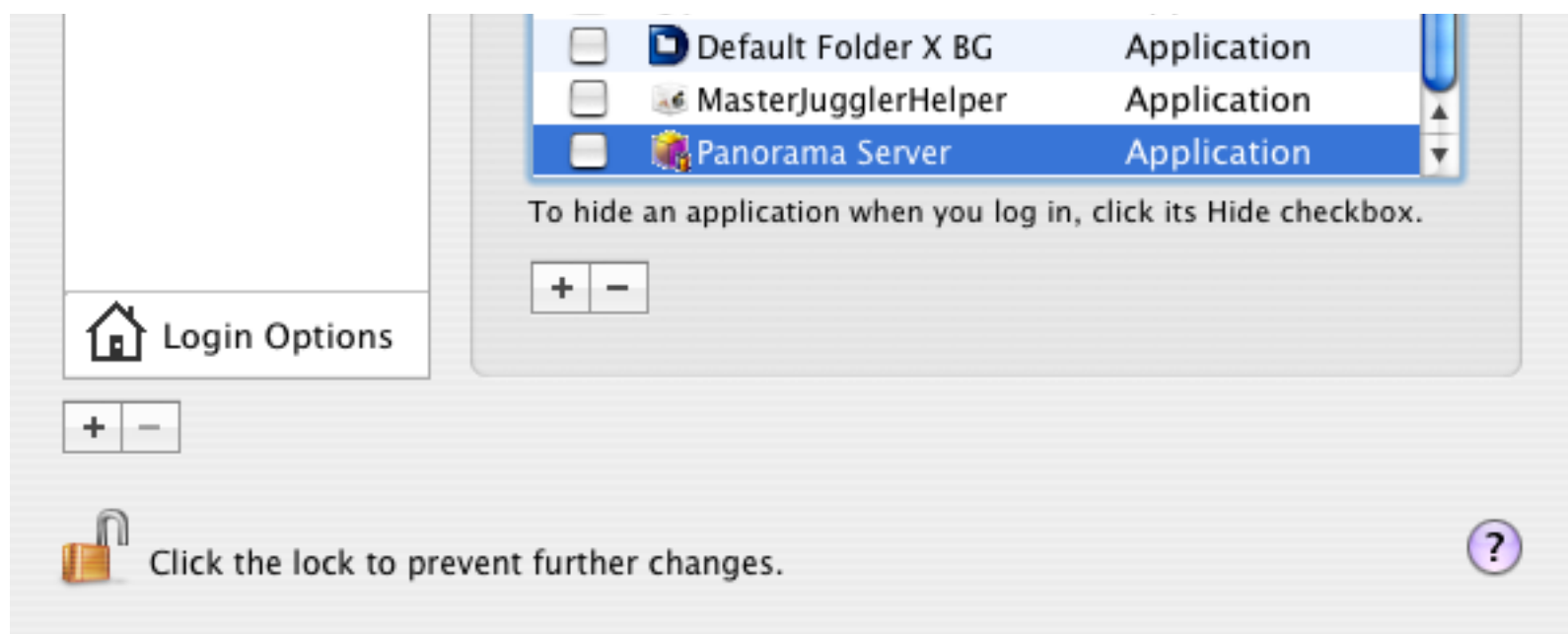
If you really want to shut down even though there are open databases being shared, hold down the **Control** key while you choose **Quit Panorama Server** or **Shut Down Server**. Keep in mind, however, that this will disconnect all users that are currently sharing databases on this server, and may cause a loss of data.

Setting up Panorama Server to Launch Automatically

If you are setting up a production server that needs to be on 24/7, you'll want to set up the system so that the Panorama Server will launch automatically whenever the system starts up (for example after a power failure). To do this, drag the **Panorama Server** icon from the Finder onto the **Login Items** pane of the **System Preferences** application (the Login Items pane is located in the **Accounts** section).



Panorama Server will now appear at the bottom of the list of items that will automatically open.



If you later decide that you no longer want Panorama Server to open automatically simply click on it to select it, then press the  button.

Panorama Server Product Activation

When you first install the Panorama Server on a computer, it initially acts as a demo version. If you haven't purchased Panorama Server yet, you can use this demo version to evaluate the software (see "[Panorama Server Demo Mode](#)" on page 48). If you have purchased the server, you'll want to activate your software as soon as possible. You can do this in a few minutes at any time of the day or night following the instructions in the Panorama Handbook (see the chapter "Installation and Activation"). When you activate the software, be sure to launch Panorama, not Panorama Server. Panorama Server does not have the **Registration** command needed to activate the software. After you activate the software, you'll need to shut down and relaunch the server (if it was already open).

Panorama Server Licensing Options

The Panorama Server database sharing and web publishing options are licensed separately. If you want the same server to be able to share databases and publish databases on the web, you'll need to purchase both licenses. (The server will have only a single Panorama serial number whether you have one or both of these licenses.)

Different Panorama Server database sharing licenses are available for different size organizations. At the time this was written, licenses are available for 6 users, 12 users, and unlimited users. See the [ProVUE.com](#) web site for the currently available configurations and pricing. It's also possible to upgrade a license at a later time (this is a special order item; contact the ProVUE sales staff if you need to do this.)

The Panorama Web Server license is only available in a single configuration, which allows unlimited database web publishing. See the ProVUE web site for pricing information.

If you want to run a Panorama client on the same computer as the server, you'll also need to purchase Panorama or Panorama Direct for the serial number in use on the server computer. Standard pricing applies, but this is a special order item. Contact the ProVUE sales staff if you need this option.

Because there are so many possible variations, you should carefully consider your needs before purchasing your Panorama Server license. Contact the ProVUE sales staff if you have any questions.

Panorama Client Licensing Options

Any licensed copy of Panorama or Panorama Direct (version 5.5 or later) can function as a client for database sharing with the Panorama Server. Demo copies, however, cannot be used to share databases.

The Panorama Personal Use License allows a single person to use Panorama on multiple computers. However, only one computer with this license can share databases at a time. In other words, since only the person who purchased the personal use license is allowed to use the Panorama with that license, that person cannot share databases on more than one computer at a time. (Shouldn't be a problem since you can't be in two places at one time!)

Panorama Server Demo Mode

If you haven't purchased a license for Panorama Server, you can still use it in a limited demo mode. This mode allows you a limited ability to work with and evaluate the software.

When in demo mode, database sharing can only be done by one user at a time. In other words, you can create shared databases and evaluate performance on your hardware and network but not actually "share" data between multiple users simultaneously. There is no time limit to demo mode or limit on the database size you can use.

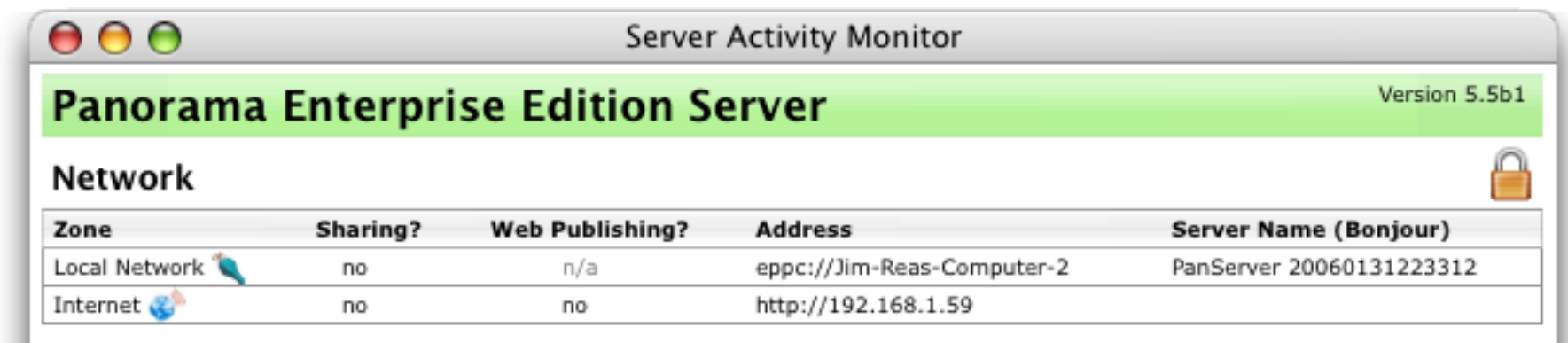
Web publishing demo mode is limited to publishing for 120 minutes. After two hours you'll need to refresh the server, which can be done with the **Refresh Demo** command in the **Admin** menu.

Basic Server Configuration

This section describes how to set up the fundamental server settings: password, server name, enabling and disabling database sharing and web publishing. Usually you'll set these options up once when you first set up the server and then never need to touch them again. A later section in this chapter describes more advanced server configuration (backup options, notification options, record locking options, etc.), see "[Advanced Server Configuration](#)" on page 89.

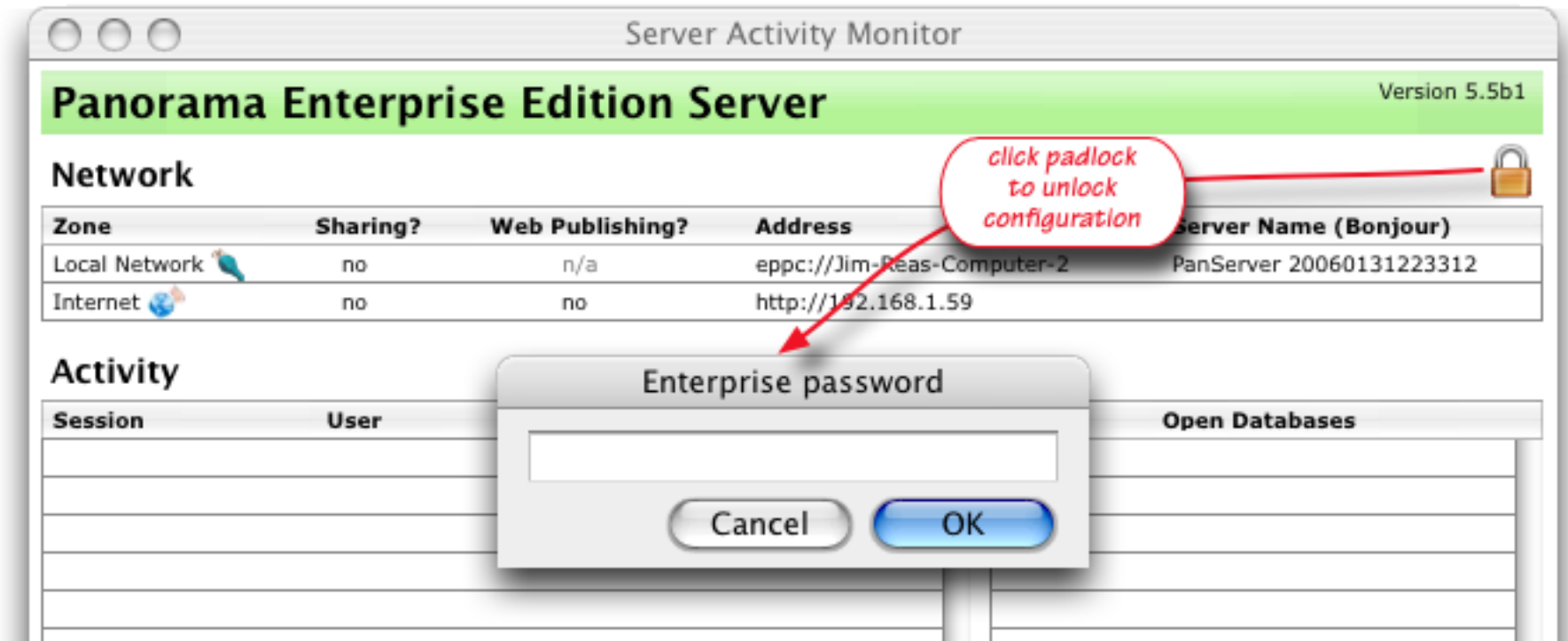
Unlocking the Server Configuration

The first time you start the Panorama Server, all services (sharing, web publishing) are turned off and the Server Activity Monitor will look something like this.



By default the configuration is locked, as indicated by the padlock in the upper right. This is to prevent accidental modification of the configuration. (This lock is not, however, designed to prevent malicious attacks by someone with physical or virtual access to the computer.)

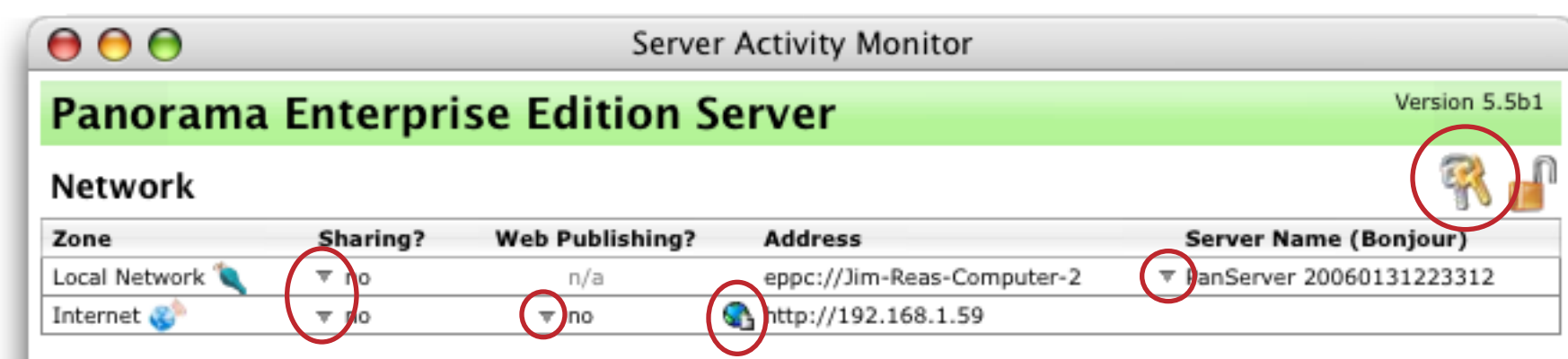
Click on the padlock to unlock the configuration.



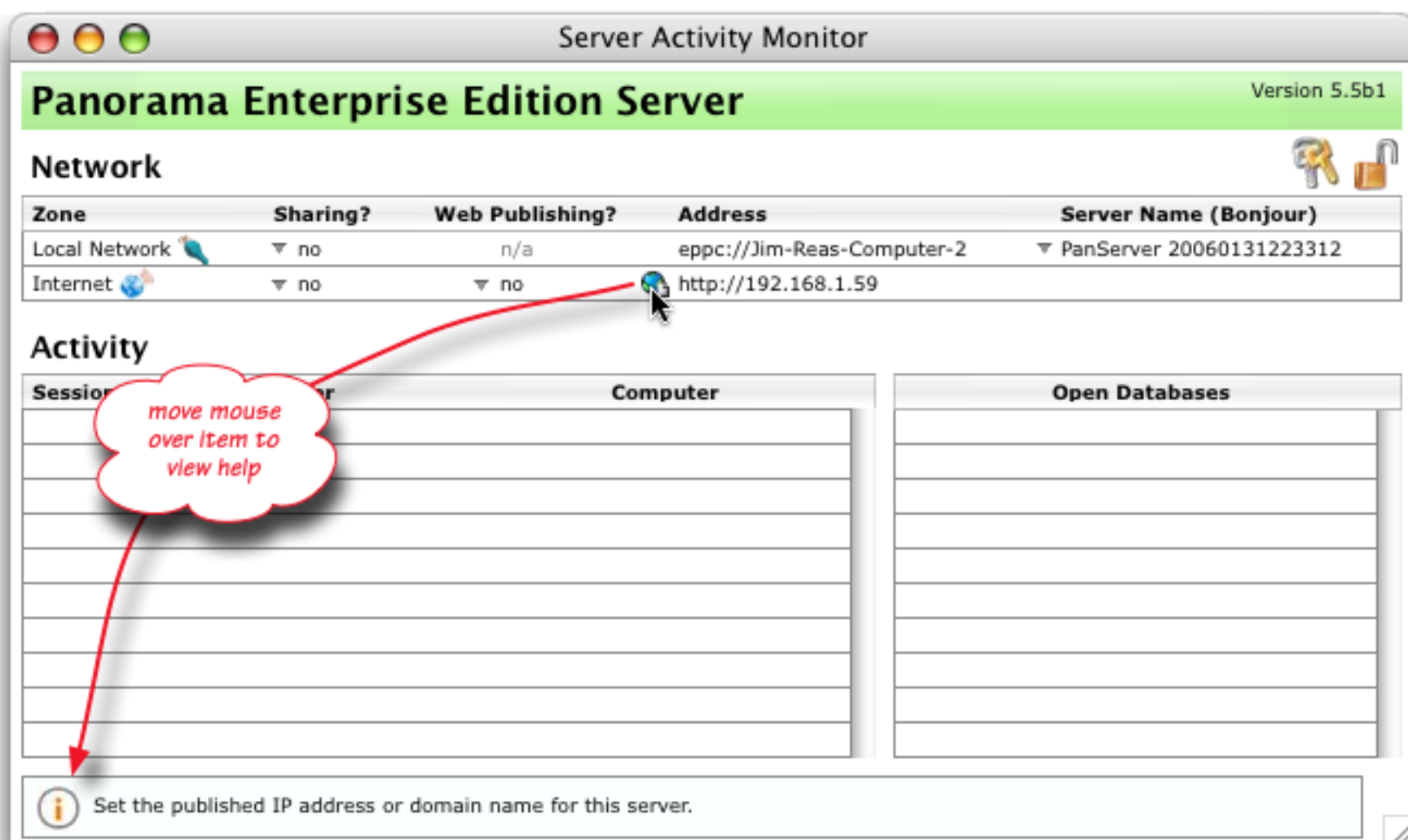
To unlock, you'll need to enter the password for this server. The default password is **enterprise** (we'll explain how to change the password in a moment.)



Press **Ok** to unlock the configuration. Now that the configuration is unlocked, you'll see several new buttons and pop-up menus.



The Server Activity Monitor has a built in help system. To see what any item means, simply move the mouse over that item and a short explanation will appear at the bottom of the window.

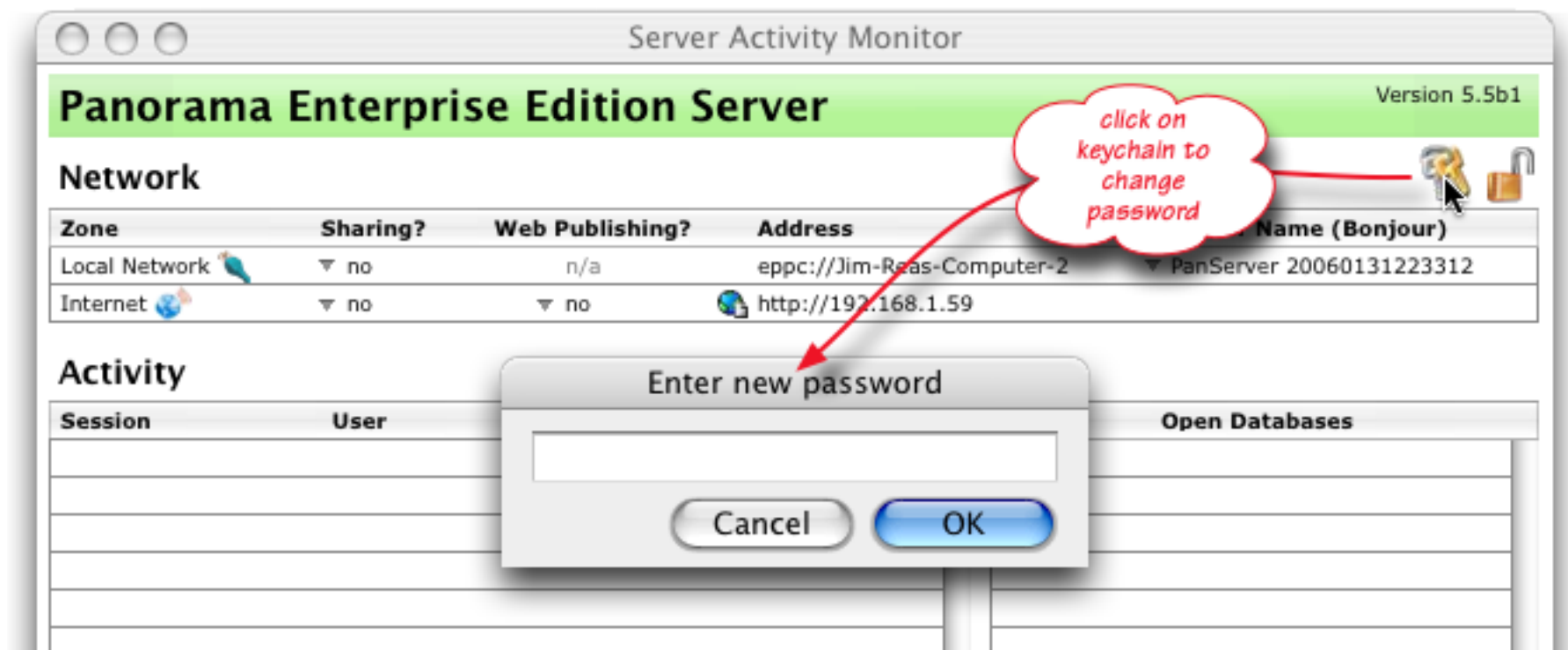


When you are done configuring the server you should click on the padlock again to re-lock the configuration.

Changing the Server Password

The default server password is **enterprise**. We *highly* recommend that you change this immediately. Anyone with the server password can act as the administrator of this server. Not only does the server password allow you to change the configuration on the server itself, it also allows you to manage the server remotely from any client on the network (or from anywhere on the Internet if your server has a static IP address). So choose your password carefully. (We recommend using a different password than your logon password.)

To actually change the password, click on the keys in the upper right hand corner (you must unlock the padlock to make the keys appear).



Enter your new password and click **Ok**. You'll be asked to confirm the new password before the change is made permanent.

Remembering the Password

The password is used both to unlock the server (clicking on the lock icon) and to remotely administer the server. However by choosing the **Remember Password** command in the **Admin** menu, you can eliminate the need to type in the password to unlock the server (it will still be needed for remote administration). This command will ask you to confirm that you really want to bypass password entry in the future. Doing so is convenient but of course less secure. For example you might want to bypass the password during configuration or testing, but then re-enable the password when actually deploying the server. To re-enable the password, simply choose the **Remember Password** command and press **No** when asked if you want to remember the password.



Note: We really don't recommend this option - you should be working on the server so infrequently that the password really shouldn't be inconvenient. However, here in our internal development work at ProVUE we of course have been doing a lot of work on the server and found this option very helpful. We left it in the final product just in case someone else also finds it useful. Caveat emptor!

Changing the Auxiliary Passwords

In addition to the main server password, there are also three optional auxiliary passwords: two for downloading and one for displaying server status from web browsers. You can edit these passwords using the **Auxiliary Passwords** command in the **Admin** menu on the server. (You can also open this dialog on a client computer using the **Server Administration** wizard, see “[Changing the Auxiliary Passwords](#)” on page 80.)

Auxiliary Passwords for TiBook

Download Databases to Local Clients:

Enter a password to restrict downloading databases to clients on the local network. This password is used by the Download Shared Databases wizard. If left blank anyone on the local network can download shared databases from this server using this wizard.

Download Databases to Remote Clients:

Enter a password to restrict downloading databases to clients on the internet. This password is used by the Download Shared Databases wizard. If left blank anyone on the internet can download shared databases from this server using this wizard.

Server & Database Web Status Pages:

Enter a password to restrict access to server and database status pages from web browsers. If left blank then anyone on the internet will be able to view a list of web published databases on this server, as well as lists of forms and procedures in each database.

Use these auxiliary passwords if you want to restrict access to downloading databases (see “[Using the Download Shared Databases Wizard](#)” on page 110) and/or displaying web status (see “[Testing Web Database Publishing \(Server Status\)](#)” on page 64). If used, each auxiliary password should be unique and should also be different from the main server password.

If an auxiliary password is left blank then that operation is not restricted. For example, if you want to allow anyone on your local network to download databases without a password, just leave the **Download Databases to Local Clients** password blank.

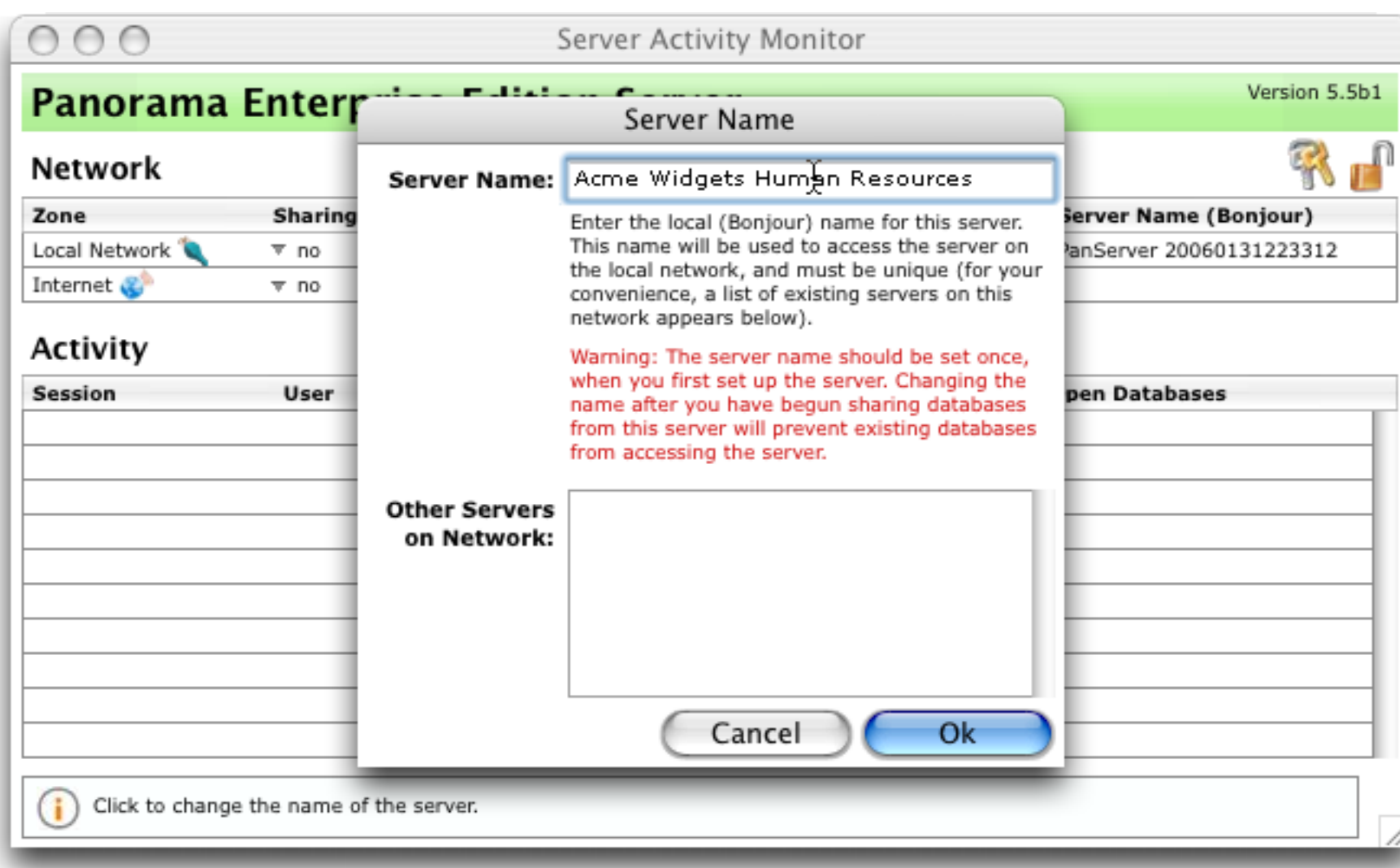
You can change these passwords at any time. Once the server is running we recommend changing the auxiliary passwords from the **Server Administration** wizard (see “[Changing the Auxiliary Passwords](#)” on page 80) rather than directly from the server.

Changing the Server Name

The server name is shown on the right hand side of the Server Activity Monitor. As you can see, the default name is PanServer followed by 14 or 15 numeric digits.



You'll probably want to change this to a unique name that describes this server, for example [Acme Widgets Payroll](#), [UW Forestry Lab](#) or [Mesa Verde Fellowship](#). To change the name, simply click on the name or the triangle to the left of the name.



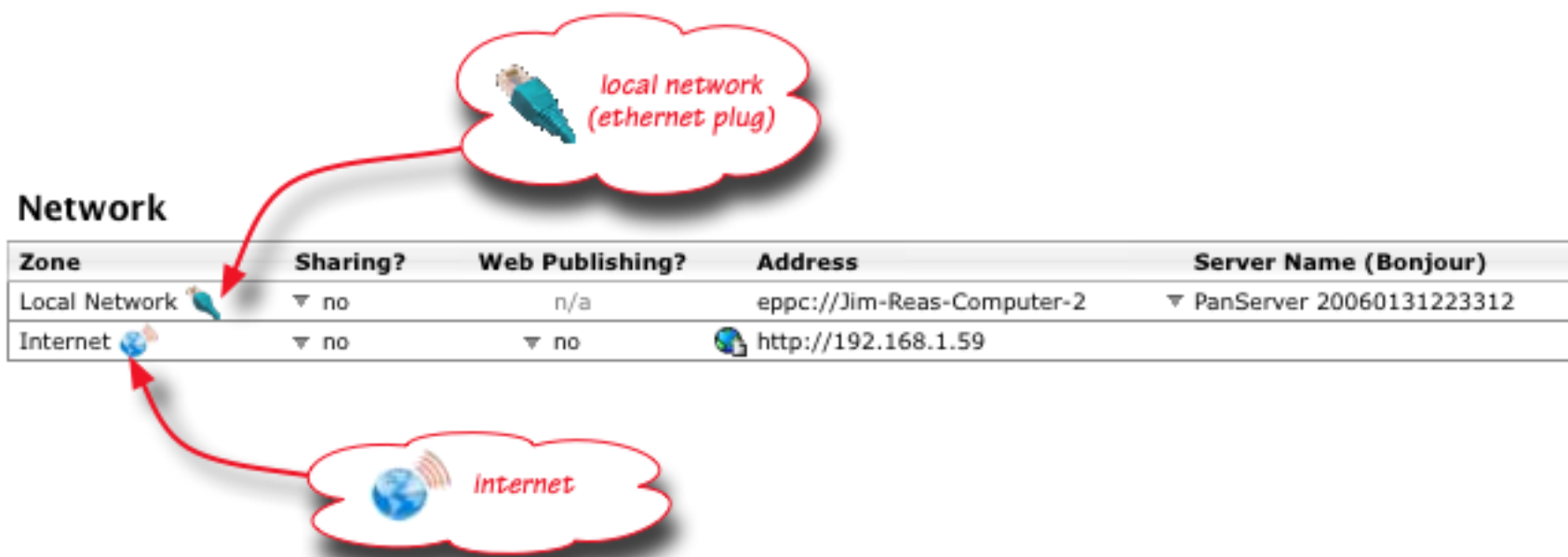
If you have more than one Panorama server on your local network each one must have a unique name. To help you pick a unique name, a list of current servers appears at the bottom of the dialog. The dialog will also object if you pick a name that is already in use. (However, the list of other servers only includes Panorama servers that are currently running. If you have any Panorama servers on your network that are not currently turned on, it is up to you to make sure that you don't duplicate the name.)

Warning: The server name will be embedded in every database that is shared on this server, so it should not be changed once you start creating shared databases.




If you are planning to share databases on the internet, you should avoid generic names like **Server** or **Payroll**. Any client computer can connect to only one Panorama server with a given name. For example suppose several departments in your organization have servers simply called Panorama Server. These servers can be accessed by anyone in the company over the internet but a given person will only be able to access one of them. For example the company president would have to choose whether he wanted to access the payroll, research, or human resources server, because he couldn't access all of them without reconfiguring his system. If the servers each have unique names then he can access all of them simultaneously.

Local Network vs. Internet Settings

The network section of the Server Activity Monitor is divided into two rows. The top row is for the local network, the bottom row for internet settings.



Network

| Zone | Sharing? | Web Publishing? | Address | Server Name (Bonjour) |
|---|----------|-----------------|---|----------------------------|
| Local Network  | ▼ no | n/a | eppc://Jim-Reas-Computer-2 | ▼ PanServer 20060131223312 |
| Internet  | ▼ no | ▼ no |  http://192.168.1.59 | |

The **Sharing?** column enables and disables database sharing. The **Web Publishing?** column enables and disables database web publishing. The **Address** column displays the local and internet address of the server computer.

Enabling Database Sharing

To enable database sharing on the local network, click on the ▼ triangle and select **yes** from the pop-up menu. (If you don't see the ▼ triangle you need to unlock the padlock.) If this is the first time you have turned on database sharing, you will be prompted to enter the System Administrator password. (**Important note:** This is your logon password for the computer, not the Panorama Server password.)

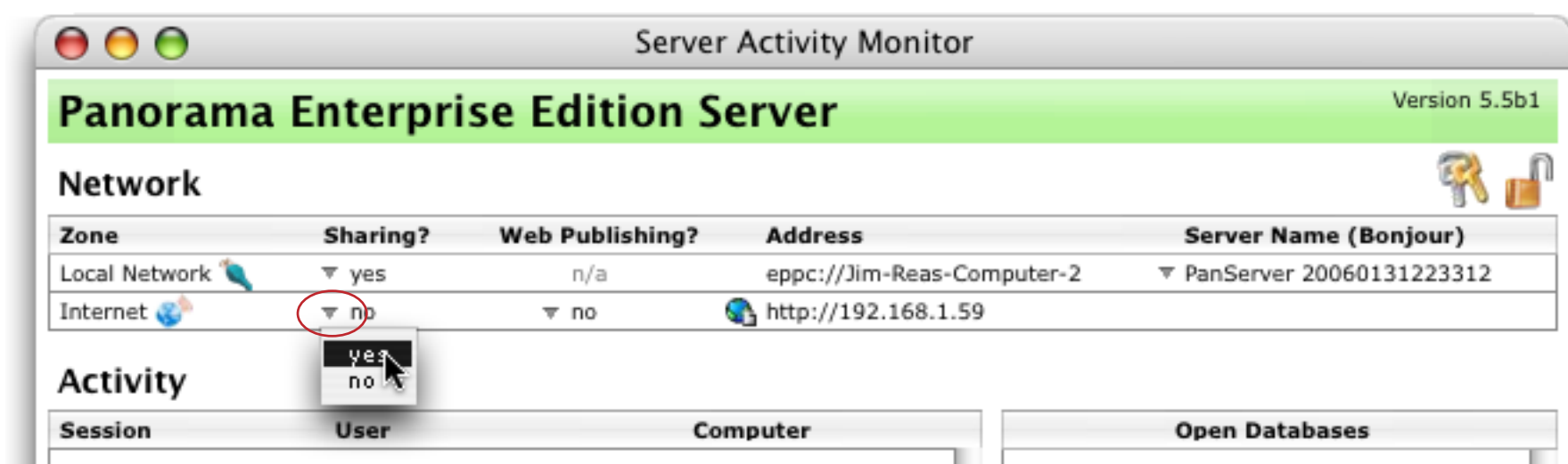


Assuming that you have entered the password correctly, the server is now ready to share databases. If the server is shut down and restarted, the local sharing will restart automatically. If you ever need to turn local sharing off again simply click on the ▼ triangle and select **no** from the pop-up menu.

Note: If the system administrator information (ID or password) ever changes, you must turn database sharing off and then back on again. You'll be prompted for the new password when sharing is turned back on.

Enabling Database Sharing over the Internet

If you want to allow databases from this server to be shared over the Internet then click on the bottom ▼ triangle and select **yes** from the pop-up menu. If web publishing is not already enabled, the server will prompt you to enter the System Administrator password. (This is your logon password for the computer, not the Panorama Server password.)

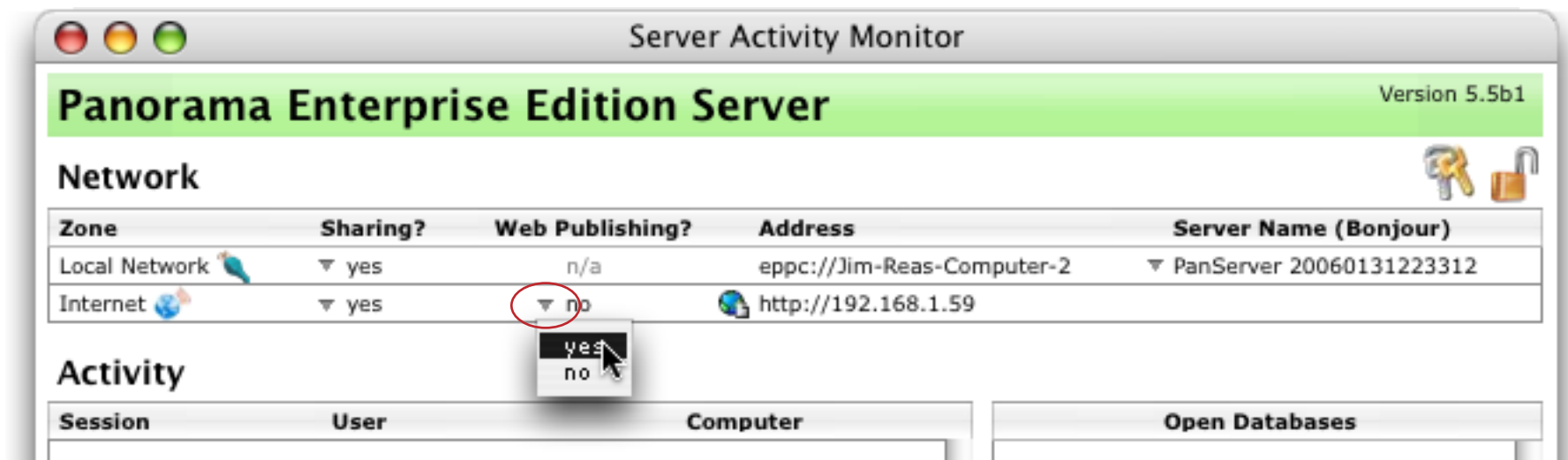


Although this option enables internet sharing, this option must also be enabled for each individual database you want to share. A database will not be sharable over the internet unless the internet sharing option is enabled *both* here and in the individual database (see "[Creating a Shared Database](#)" on page 103).

If you want all connections to the server to be made via tcp/ip (not remote apple events), you can enable internet sharing but leave local network sharing off. In our tests tcp/ip access is somewhat slower than remote apple events, but this does give you an option if your network does not support remote apple events (for example if the router is configured to block port 3031 on the local network).

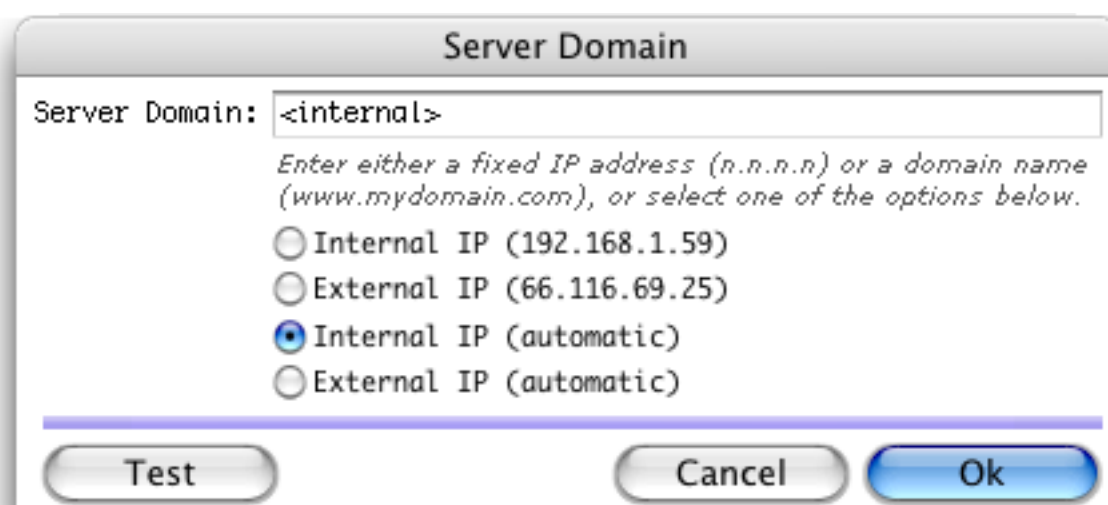
Enabling Database Web Publishing

To enable database web publishing, click on the ▼ triangle in the **Web Publishing?** column and select **yes** from the pop-up menu. The server may prompt you to enter the System Administrator password (if you haven't entered it already during this session). (This is your logon password for the computer, not the Panorama Server password.)



Setting the IP Address/Domain Name

When publishing web pages, the Panorama server sometimes needs to create links to other pages or images on the web server. To do that, it needs to know the URL of the web server. The server can't figure this out by itself, so you'll have to tell it what its IP address or domain name is. (If you are only using the server for database sharing, you can skip this section). To set the IP address or domain name click on the globe icon.



You can choose one of the four options or simply type in a domain name or IP address. For example if the domain name for your web server is www.acmewidgets.com you should type that into the **Server Domain** area. The four radio buttons set the IP address semi-automatically or fully automatically.

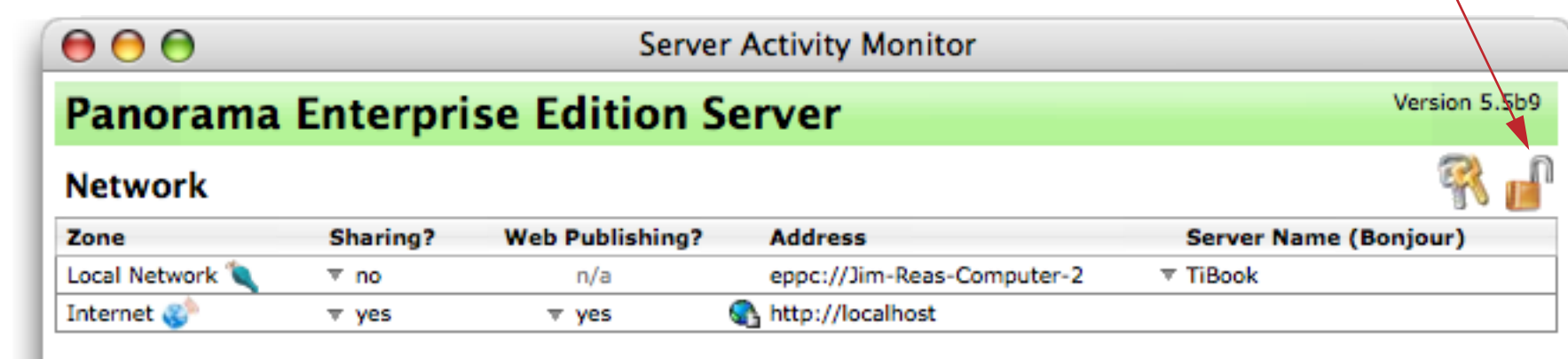
| Option | Description |
|---------------------------|---|
| Internal IP (nn.nn.nn.nn) | This sets the IP address to your computer's current IP address (which is shown in the parentheses). You can use this option if your computer has a static IP and is directly connected to the internet without a router. |
| External IP (nn.nn.nn.nn) | This sets the IP address to your router's current IP address (which is shown in parentheses). You can use this option if your router has a static IP and port 80 is forwarded to this computer. |
| Internal IP (automatic) | This option is similar to the first option except that it will check the computer's current IP address every time the server starts up. Every time the server starts it will adjust the IP accordingly. Of course for web serving you should usually have a static IP so the IP address should never change. This option is good for testing when you are accessing the server only from this computer. |
| External IP (automatic) | This option is similar to the first External IP option except that it will check the router's current IP address every time the server starts up. Of course for web serving you should usually have a static IP so the IP address should never change. |

The **Test** button allows you to verify that the IP address or domain name you have specified is correct. It will either display an error alert or a message indicating that the address verified as ok.

Locking the Server Configuration

When you are done configuring and testing the server, don't forget to lock the configuration by clicking on the padlock.

click here to lock server configuration




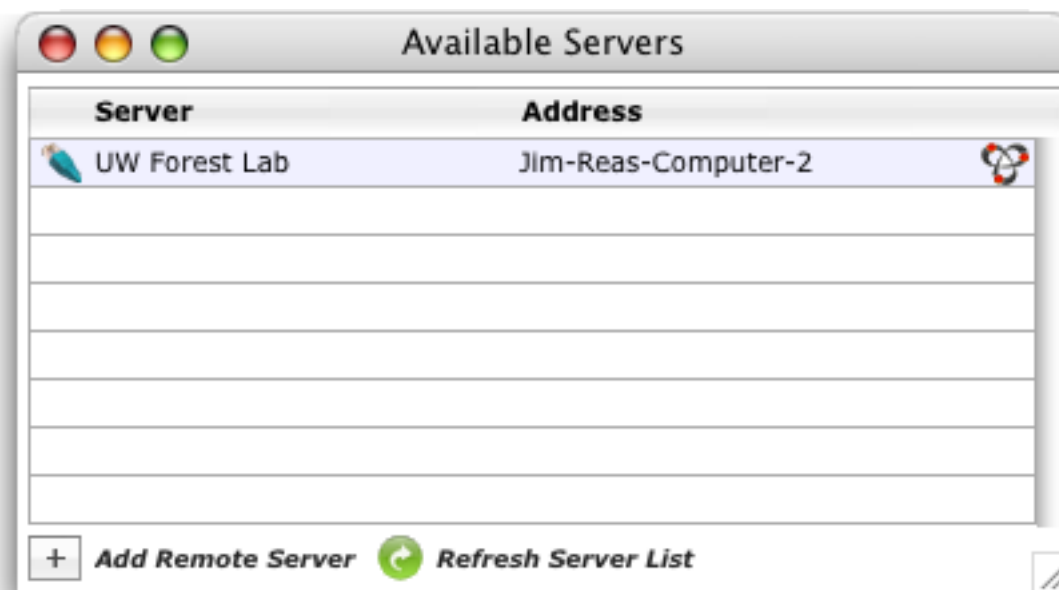
This helps insure against accidental modification of these important settings.

Client Configuration

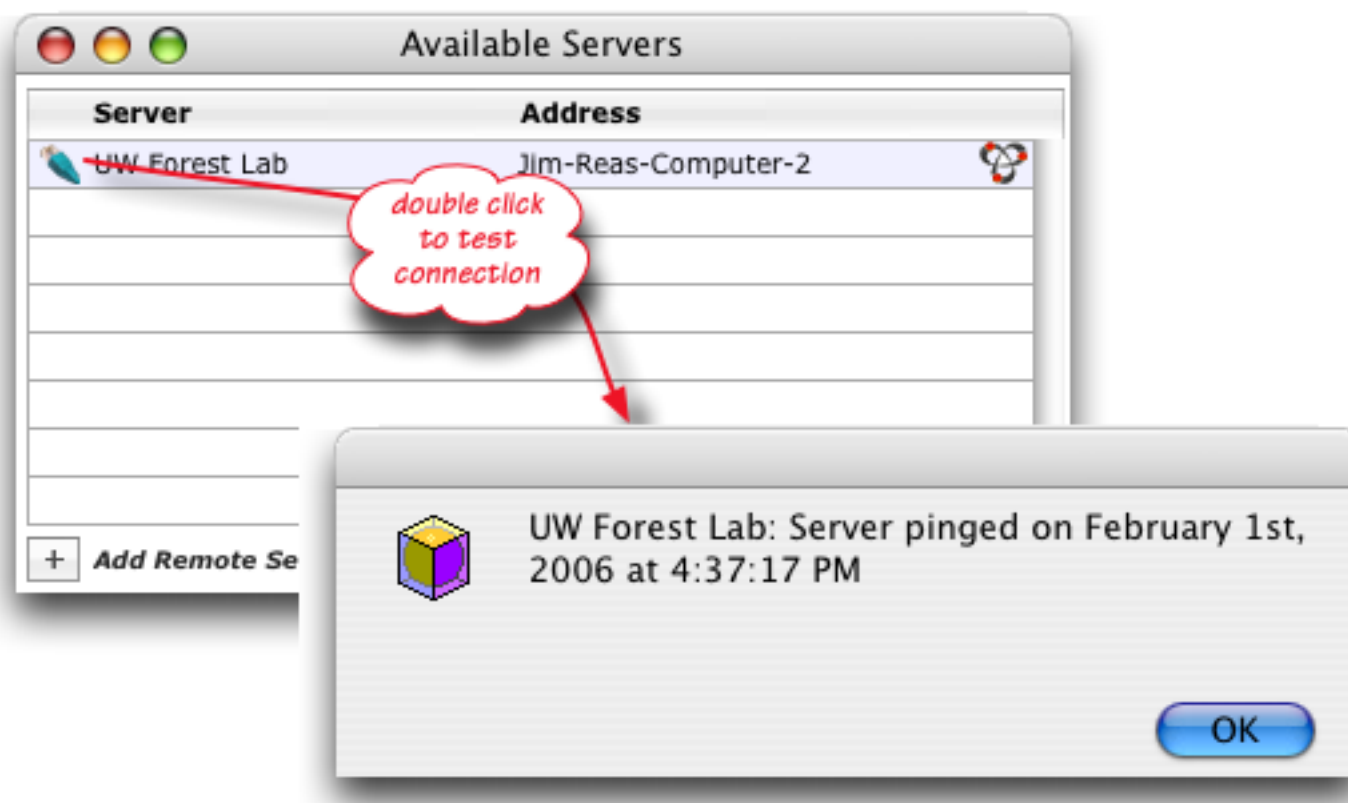
To actually start using the server, you'll need to configure at least one client.

Configuring Local Database Sharing

Any computer on the local network can be configured for database sharing, and in fact in most cases no actual configuration is required. However you can use the **Available Servers** wizard to check the configuration and to customize it if necessary. Start by opening the Panorama application (of course the server must already be running). Open **Available Servers** from the **Sharing** submenu of the **Wizard** menu. This window should show a list of available servers, including the server you have just set up. (The  symbol indicates that this is a server on the local network that was automatically discovered via Bonjour.)




The fact that a server shows up in the **Available Servers** window indicates that Bonjour is working, but does not prove that the client can actually communicate with the server. To test the actual connection double click on the ethernet plug icon next to the server name.

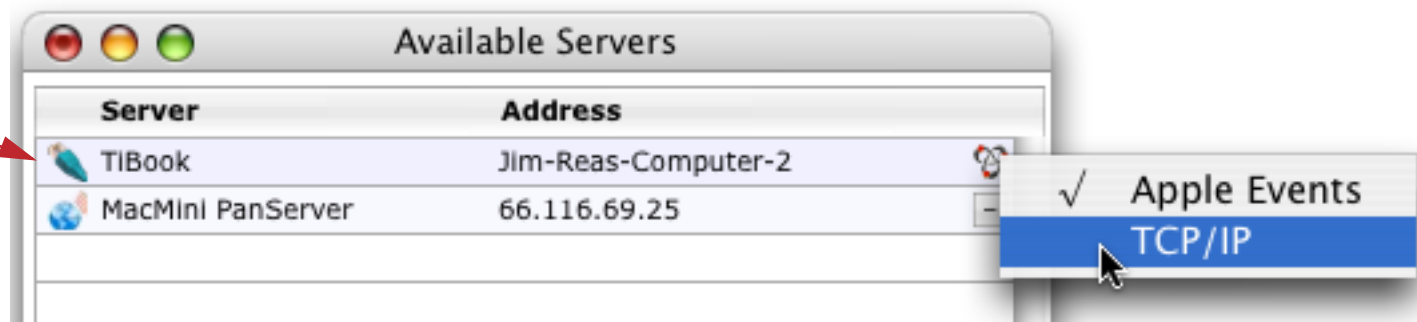


Notice that so far you really haven't done any configuration — you've simply tested Panorama's automatic default configuration. In fact, for local database sharing you usually don't have to do any configuration at all. You really only need to open the **Available Servers** wizard if there is a problem.

Configuring Local Database Sharing to use TCP/IP instead of Remote Apple Events

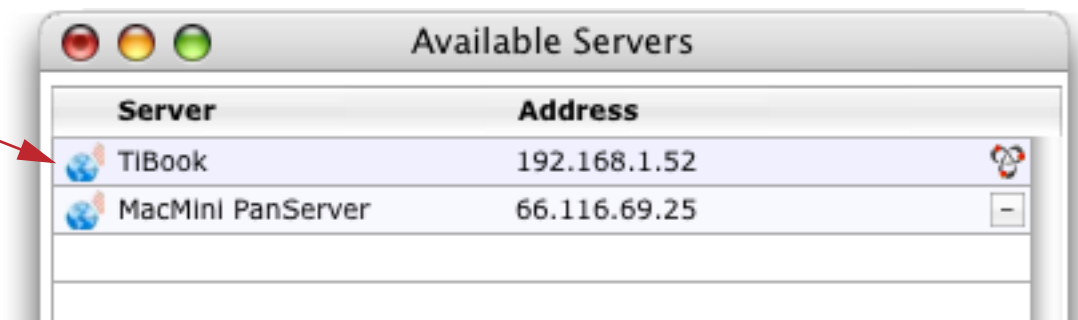
If you've enabled local database sharing on the server (see "[Enabling Database Sharing](#)" on page 55) Panorama will normally use Remote Apple Events to communicate between the client and the server. However, if internet database sharing is enabled on the server (see "[Enabling Database Sharing over the Internet](#)" on page 55) you can force a client to connect to the server using TCP/IP. To do this click on the Bonjour icon () and choose the connection method from the pop-up menu.

client will connect to this computer using remote Apple Events





The window will update to show the new status.

client will connect to this computer using TCP/IP



This selection will be permanent for this computer (unless you change it back) but will not affect how any other computers on the network connect with the server. If you want all computers on the network to connect using TCP/IP, you should go to the server and turn off the local database sharing option while leaving the Internet sharing option turned on.

Network

| Zone | Sharing? |
|---|----------|
| Local Network  | ▼ no |
| Internet  | ▼ yes |

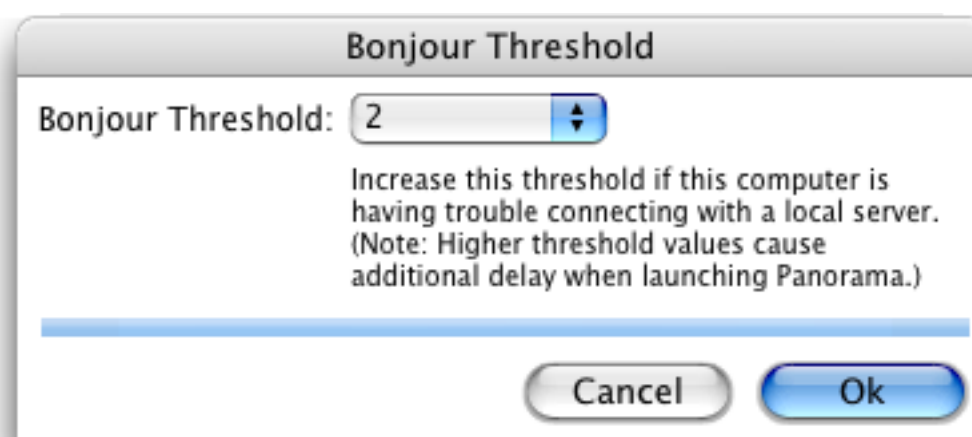
With this combination on the server, all local database clients will use TCP/IP instead of Remote Apple Events.

Configuring Bonjour

On the local network Panorama uses Apple's **Bonjour** technology to scan the network and locate any servers that are available. Normally this is completely automatic and something you don't have to worry about at all. However on some networks (especially networks with a lot of computers on them) you may need to tweak Panorama's Bonjour settings to get a reliable connection.

Bonjour Threshold

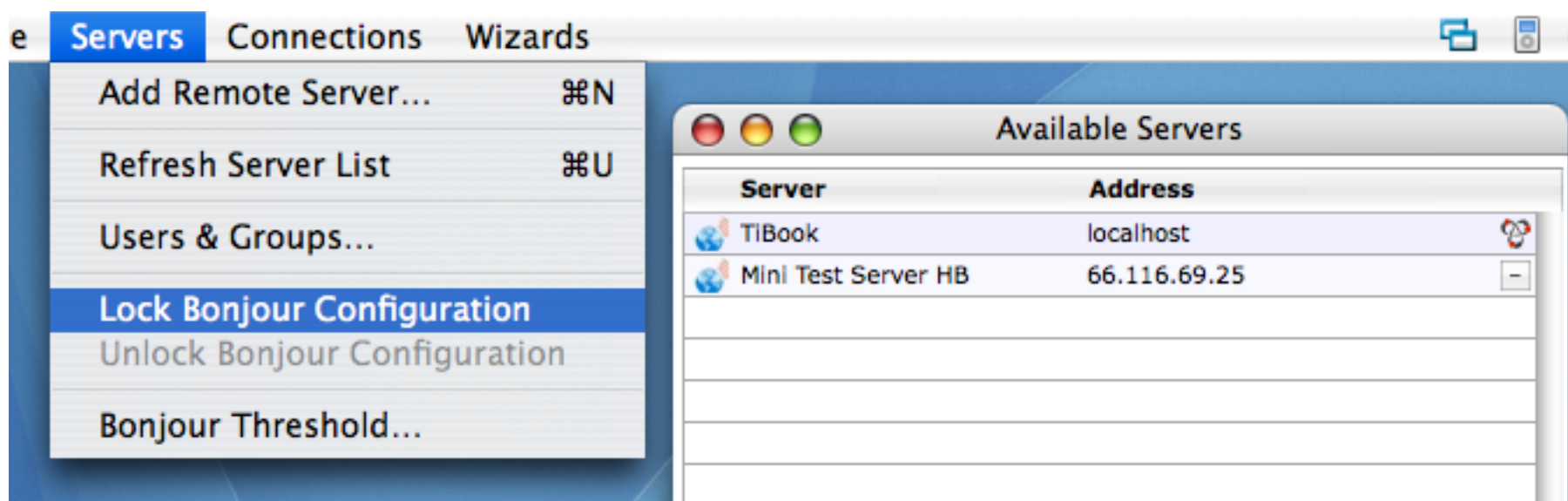
When it first tries to locate a server Panorama scans the network for a short interval to find out what servers are available. If a client does not reliably locate the server you can increase the amount of time Panorama will scan. This interval is called the **Bonjour threshold**. To increase this threshold use the **Bonjour Threshold** command in the **Servers** menu of the **Available Servers** wizard. Larger values may be needed for more complex networks (however, increasing the threshold causes an additional delay the first time the client connects to a server).



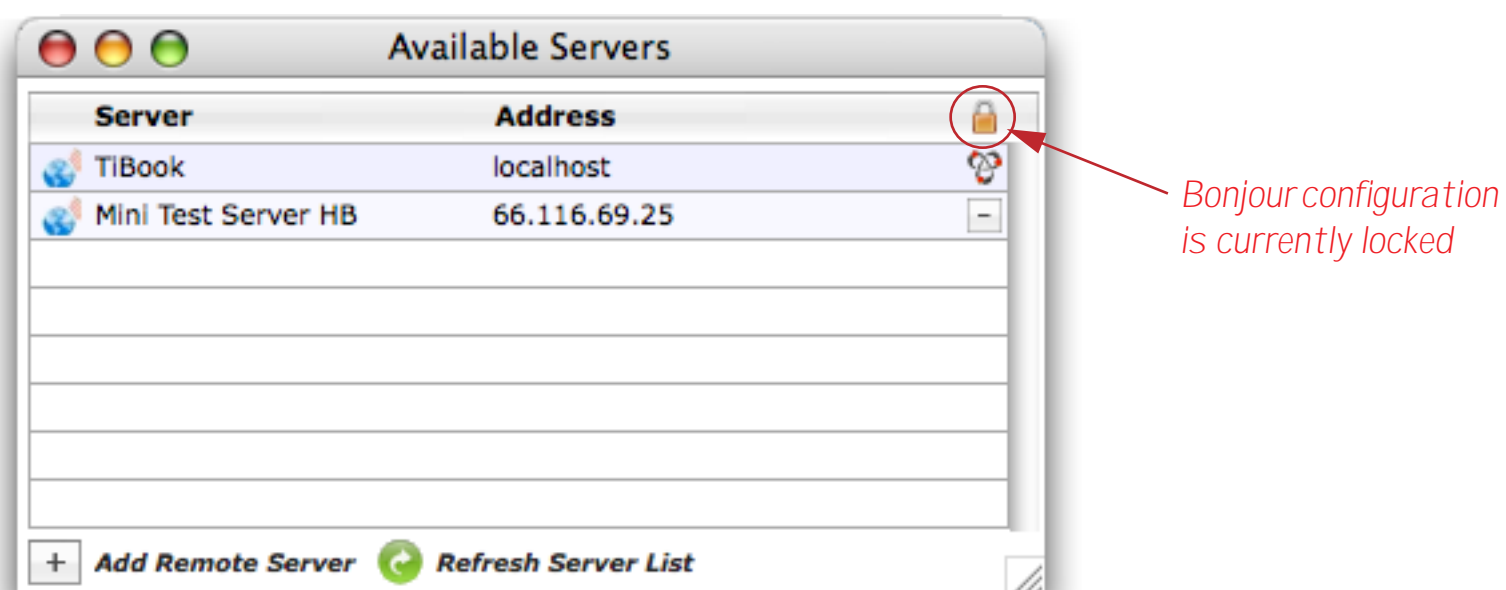
Locking the Bonjour Configuration

As described above, Panorama normally scans the network looking for servers the first time it needs to connect to a server (for example when you open a shared database). However you can also “lock” the Bonjour configuration, which essentially freezes Panorama's list of servers on the local network. You should only use this locking if you have a guaranteed stable server configuration on your local network. Locking the Bonjour configuration eliminates the need for Panorama to scan the network. This has two potential advantages: 1) The first shared database you open after launching Panorama will open faster, and 2) If Panorama is for some reason having problems scanning the network reliably locking Bonjour allows Panorama to find the server once and then “freeze” the configuration so that no further scanning is necessary, even when Panorama is closed and relaunched.

To lock the current Bonjour configuration open the **Available Servers** wizard and choose **Lock Bonjour Configuration** from the **Server** menu. Make sure that any servers you want to use are visible before you lock the configuration.



Once the configuration is locked a small lock icon will appear in the upper right hand corner of the wizard.



If the servers on your network ever change (for example you move the server software to a different computer or change the name of the server computer (or IP address if using TCP/IP) you will have to unlock the Bonjour configuration (and then lock it again, if desired.)

Debugging Local Database Sharing Connection Problems

If the server does not show up at all in the **Available Servers** wizard double check that both computers are connected to the same local network (the same router). If the client computer is not on the same local network as the server computer you'll have to use internet sharing to connect to the server (see "[Configuring Remote \(Internet\) Database Sharing](#)" on page 62). Also double check that Local Network sharing is enabled on the server (see "[Enabling Database Sharing](#)" on page 55). If these steps don't solve the problem then you may need to increase the Bonjour threshold value (see "[Configuring Local Database Sharing to use TCP/IP instead of Remote Apple Events](#)" on page 59).

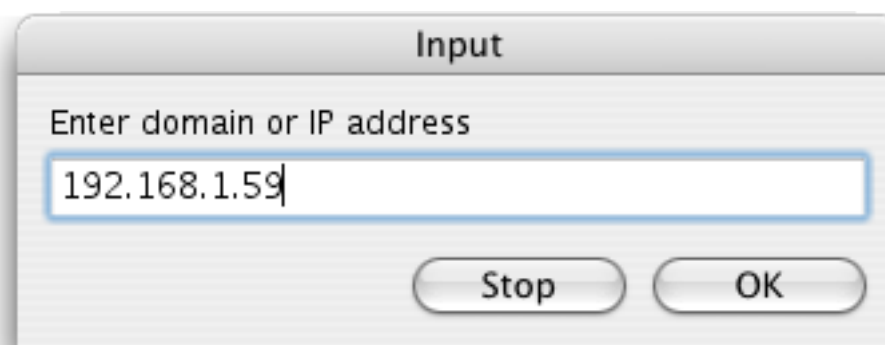
If the server does show up in the list but double clicking on the ethernet plug doesn't work then double check that the server computer has **Remote Apple Events** option enabled in the System Preferences application (see "[Enable Remote Apple Events](#)" on page 33). If that is ok then you can try turning sharing off and on again. When you turn sharing on again you may want to hold down the **control** or **option** key. This forces Panorama to ask you for the System Administrator password again (if this password is incorrect clients will not be able to connect to the server).

Configuring Remote (Internet) Database Sharing

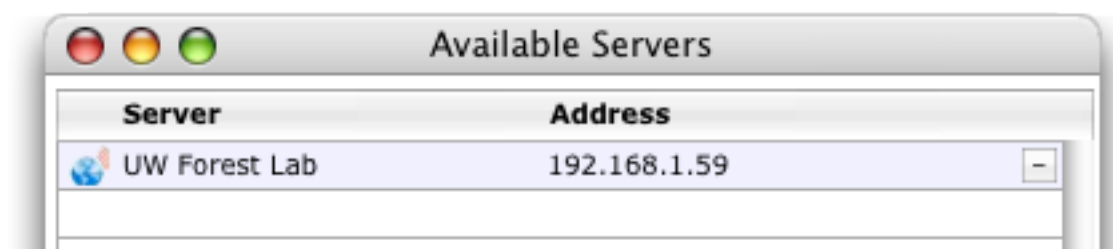
You can configure remote database sharing from any computer on the internet that has a copy of Panorama 5.5 or later. Start by opening the Panorama application. Open **Available Servers** from the **Sharing** submenu of the **Wizard** menu. Since the server is not on the local network it will not appear.



Press the **Add Remote Server** button and enter the domain name or IP address of the server.



If you've entered a correct domain name or IP address the server will appear in the list of available servers.

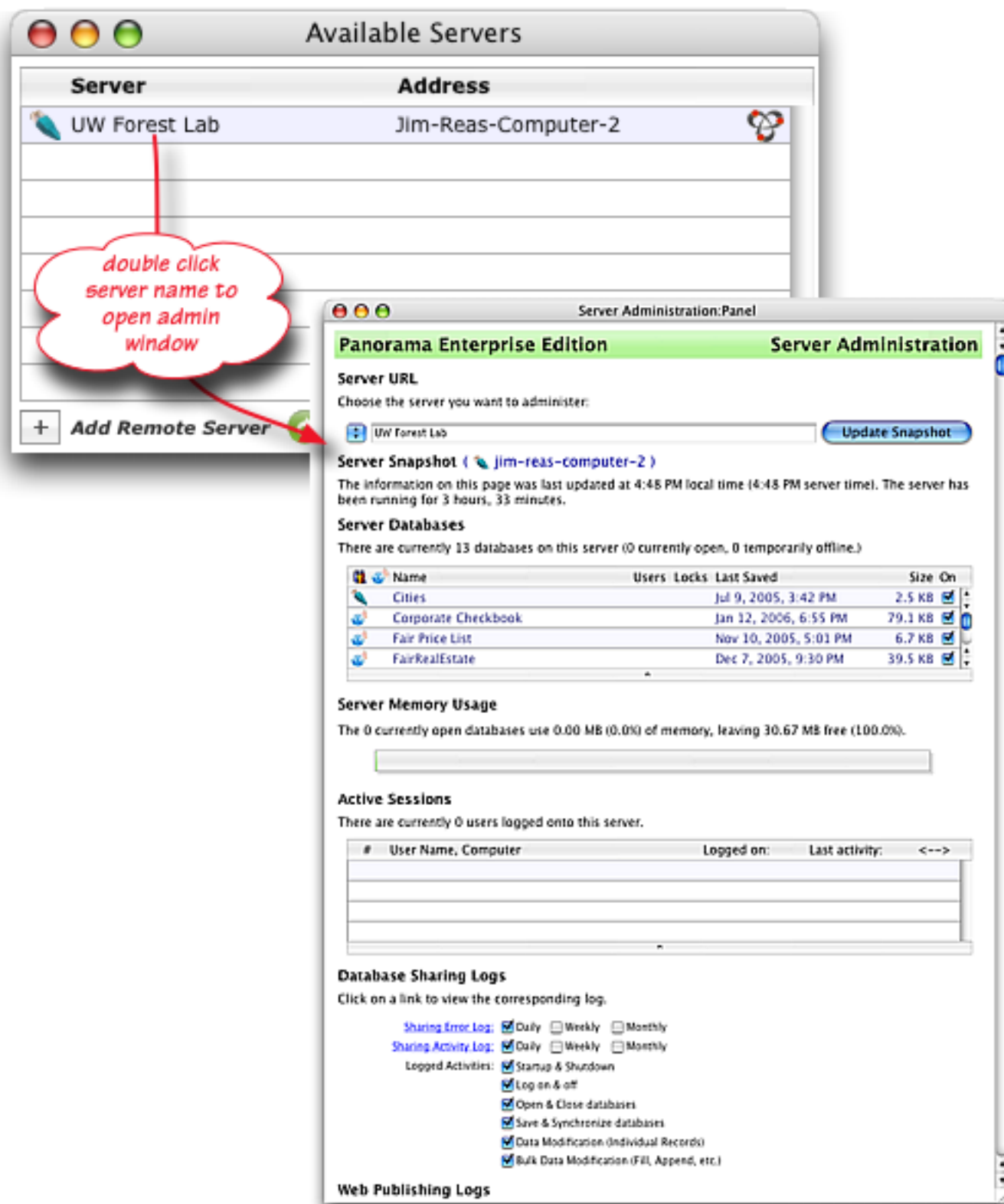


If the server name doesn't appear you'll of course need to track down why. Possible culprits include router misconfiguration (port 80 must be forwarded to the Panorama Server computer), incorrect (or no) static IP address, forgetting to turn on personal web sharing (see "[Enabling Internet Sharing](#)" on page 38) or general network problems. You may find that trying to connect from different computers may isolate the problem. For testing purposes it's possible to set up an internet connection to the server even from computers on the local network (or even from the server computer itself). You should remove this connection once it is working, however (see below).

If you are finished with a server and don't plan to ever connect to it again from this computer you can remove it from the list of available servers. Simply click the button to the right of the server name. After confirming that you really want to delete the server Panorama will remove the server from the list (this doesn't affect the ability of any other computer to connect to the server).

Further Testing the Sharing Connection

As a more comprehensive test of the client/server connection, you can double click on the server name to open the **Server Administration** wizard. After you enter the server password, this wizard displays detailed information about the server.



If this wizard is working you should have no trouble opening shared databases with this server. This wizard is discussed in more detail later in this chapter (see "[Server Management \(The Server Administration Wizard\)](#)" on page 68).

Testing Web Database Publishing (Server Status)

To test web database publishing launch Safari or the web browser of your choice, then enter the URL as shown below (substitute your actual server IP or domain name, or `localhost` if on the same computer as the server):

`http://www.yourserverdomain.com/cgi-bin/panorama.cgi?serverstatus`

The server should respond with a status page similar to the one shown below (of course if you haven't set up any databases yet, none will be listed).

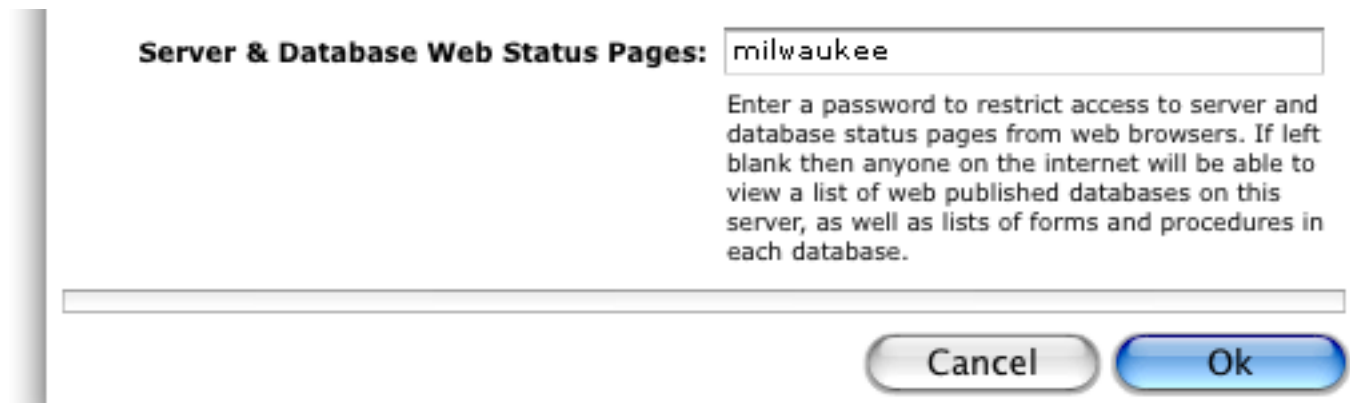
There are currently 13 web databases on this server.

| Database | Last Modified | Size |
|--|--------------------|---------|
| AAA Real Estate X Test | May 6 @ 4:56 PM | 50.5 kB |
| Checkbook MacFair 2006 | May 5 @ 1:14 PM | 75.2 kB |
| ColoradoHotels | Apr 29 @ 9:33 PM | 117 kB |
| Customers | Wed @ 4:50 PM | 2.55 MB |
| Customers2 | 07/19/06 @ 5:48 PM | 2.45 MB |
| HobbyShopCatalog | May 9 @ 11:22 AM | 238 kB |
| HobbyShopSales | Dec 6 @ 11:43 AM | 638 kB |
| Lookups | May 5 @ 10:05 AM | 15.2 kB |
| MailingList | 07/14/06 @ 5:41 PM | 2.04 MB |

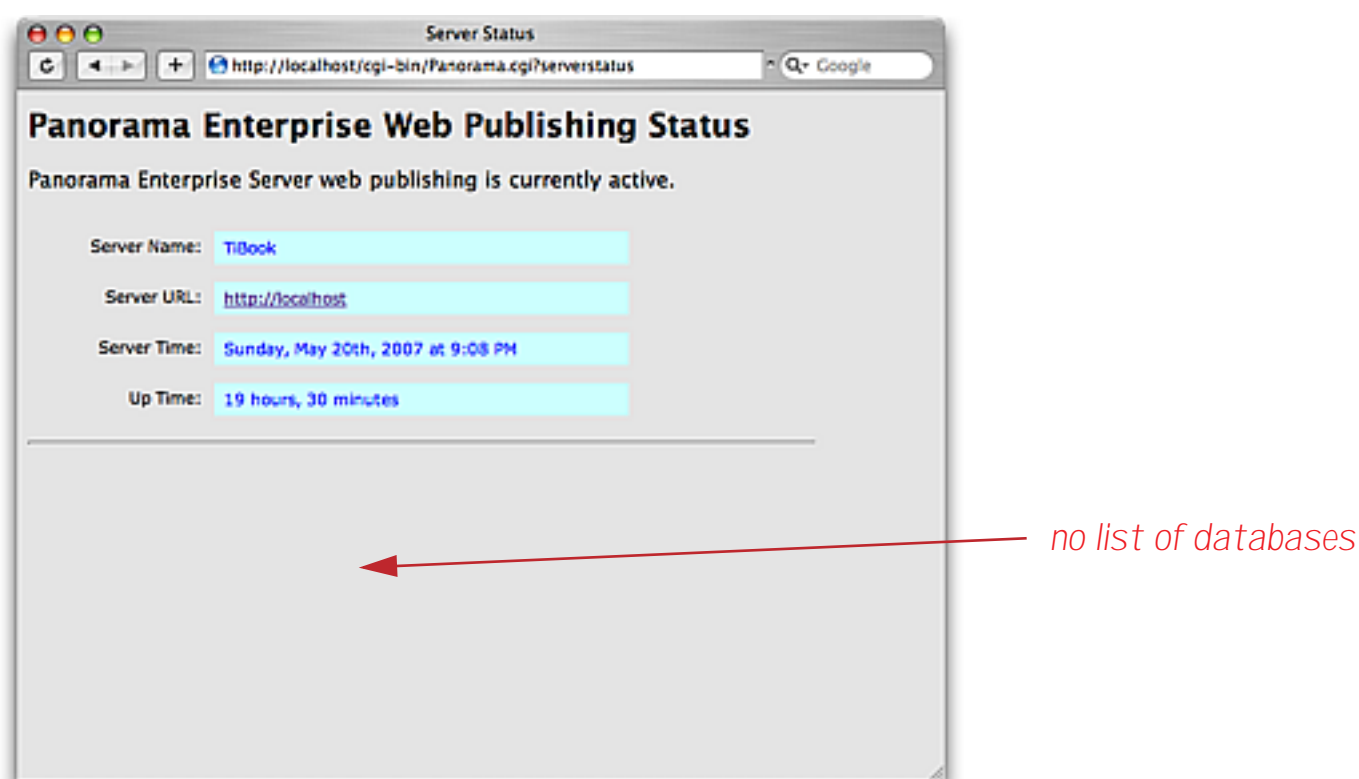
This page lists the server name (this is the “Bonjour” name you set up earlier, see “[Changing the Server Name](#)” on page 53), the server’s URL (see “[Setting the IP Address/Domain Name](#)” on page 56), the current date and time at the server’s location (which may be in a different time zone than you are) and the length of time the server has been running. If any databases have been uploaded to the server for web publishing these will be listed (you can make this list available only with a special password, see “[Changing the Auxiliary Passwords](#)” on page 80).

In addition to typing in the URL for the Server Status page manually, you can also let Panorama do the work for you. To do this, open the **Server Admin** wizard, choose the server, then choose **Server Web Status** from the **Admin** menu. The wizard will automatically open the browser and display the server status page. See “[Server Management \(The Server Administration Wizard\)](#)” on page 68 for more information on this wizard.

(see “[Changing the Auxiliary Passwords](#)” on page 80). The password should consist of letters and numbers, and we recommend that you make the Web Admin Password different from the main Enterprise Administrator password (see “[Changing the Server Password](#)” on page 51). The dialog does not disguise the password because it is going to be included in URL’s anyway.



Once this auxiliary password has been set up, the Server Status page will normally omit the detailed database information.

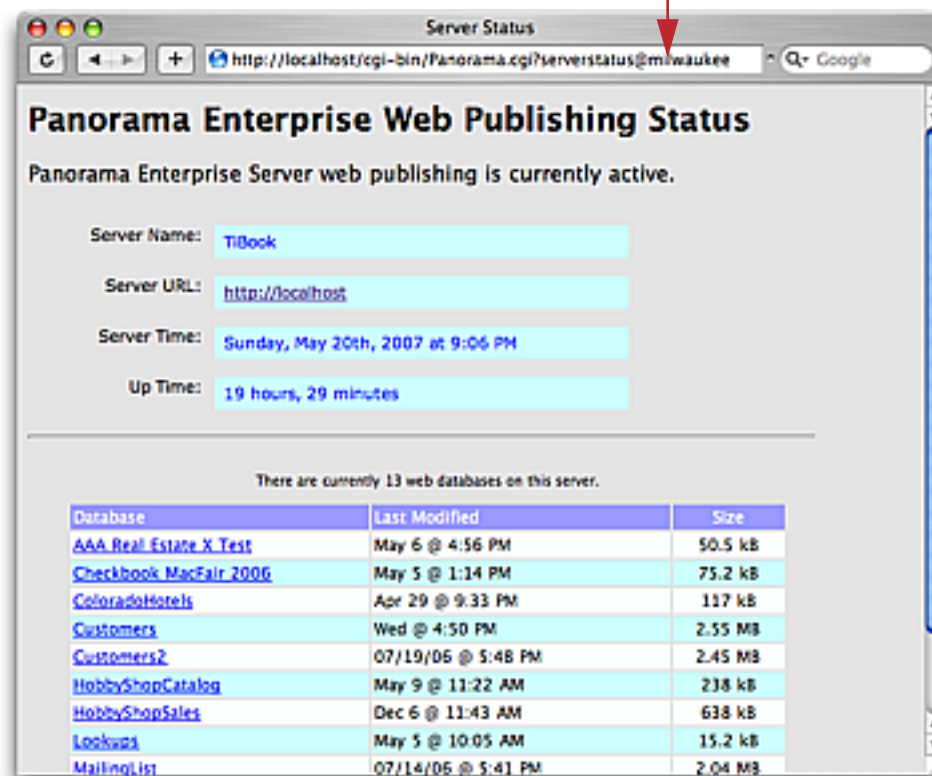


If you want to see the detailed information, you must append @ and the password to the end of the URL, like this:

`http://www.yourserverdomain.com/cgi-bin/panorama.cgi?serverstatus@password`

When the password is included in the url, the list of databases with links will be displayed.

auxiliary password = milwaukee



Note: If you accidentally submit an incorrect password, you must wait at least ten seconds before re-submitting the correct password. The server will ignore the correct password for ten seconds after an incorrect password is submitted. This delay is included to slow down brute-force attempts to crack the password by repeatedly querying the server.

If this auxiliary password has been set up, the Web Database Links pages also require an @ symbol and password appended to the end of the URL. If you let Panorama link to this page for you (either from the **Server Admin** or **Database Sharing Options** wizards or from the **Server Status** page) this is taken care of for you, but you'll have to do this yourself if you manually type in the URL like this:

<http://www.yourserverdomain.com/cgi-bin/panorama.cgi?databasename~admin@password>

See "[The Database Web Links Page](#)" on page 202 for more information on this web links page.

Server Management (The Server Administration Wizard)

Once you've configured your Panorama Server you'll probably never need to touch the actual server computer again. You can monitor and manage your server remotely from any computer that has Panorama 5.5 or later. The primary tool you'll use for this is the **Server Administration** wizard. This wizard allows you to monitor and manage shared databases on the server, memory usage, logged on users, activity logs and backup.

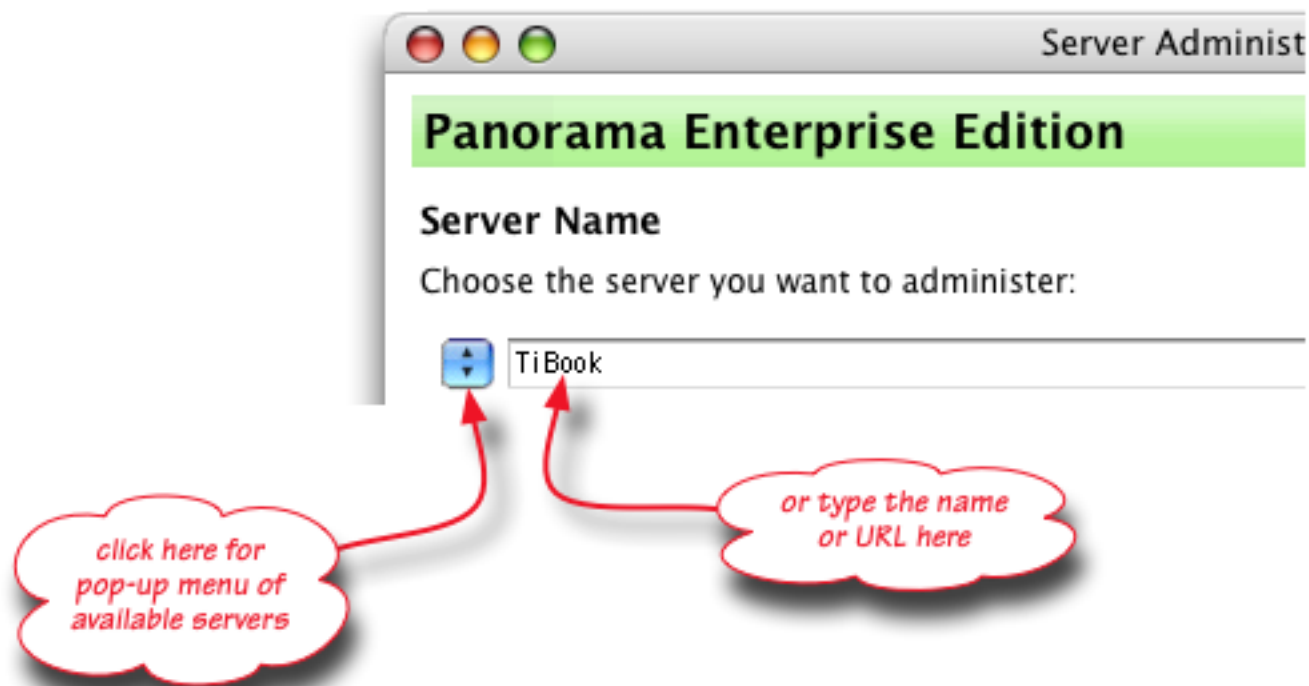
You can open the **Server Administration** wizard directly from the **Sharing** submenu of the **Wizard** menu, or by double clicking on a server name in the **Available Servers** wizard (see "[Further Testing the Sharing Connection](#)" on page 63). Here's a typical view of this wizard in action.



You'll need the Panorama Server password (see "[Changing the Server Password](#)" on page 51) to work with this wizard. It will ask you for the password when you first select the server you want to manage. It will ask for the password again if you perform an operation and haven't entered the password in the last five minutes.

Choosing the Server to Manage

There are three ways to select a server to manage: 1) Double clicking on the server name in the **Available Servers** wizard, or 2) Select the server name from the pop-up menu in the **Server Name** area, or 3) Type the server name or URL (internet sharing only) into the box in the **Server Name** area.



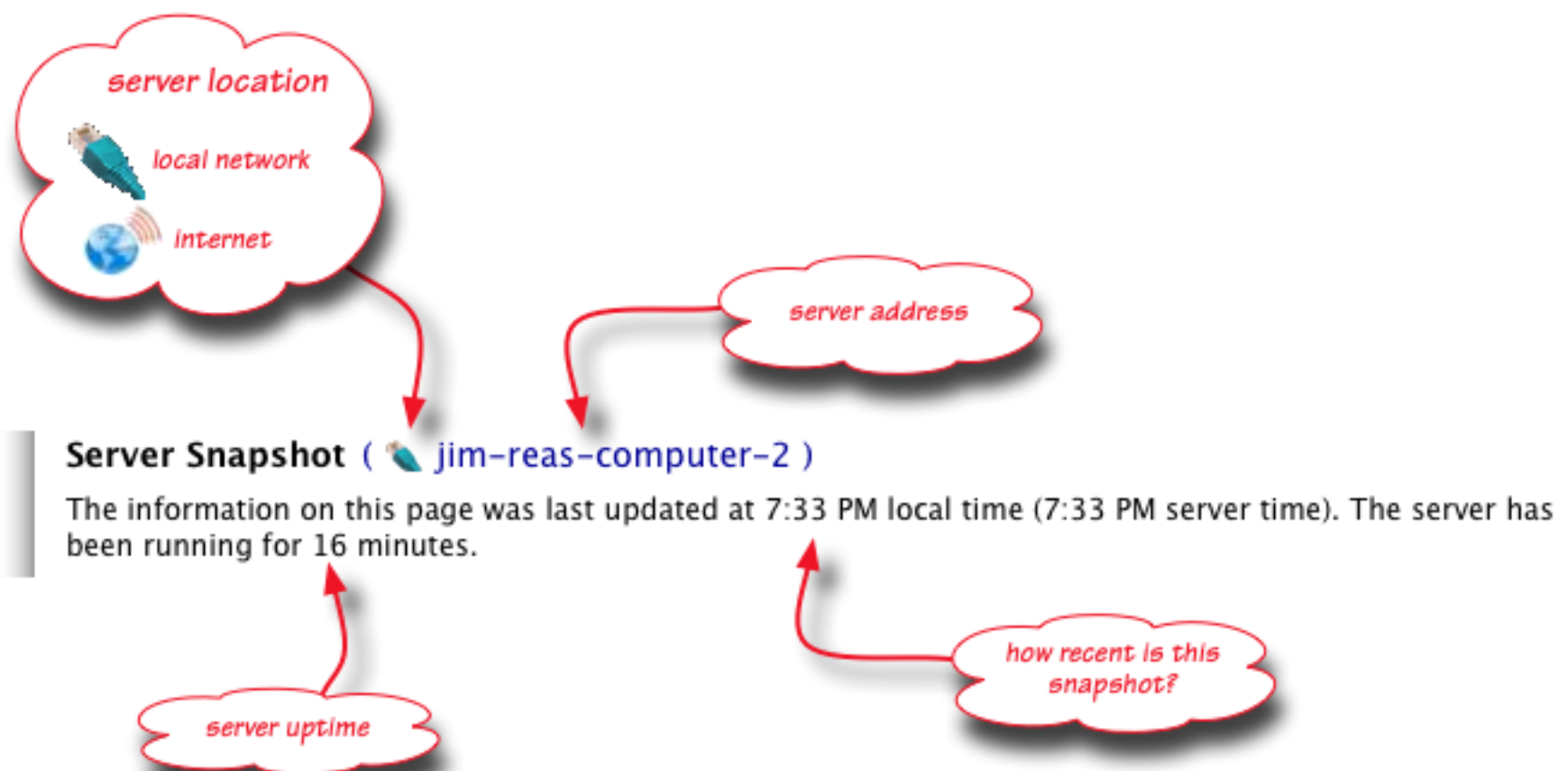
The wizard will ask you for the server password, then display a “snapshot” of the current status of the server.

Updating the Snapshot

The information displayed by the Server Administration wizard is a snapshot, a point in time. It is not updated automatically. If you want to update the snapshot display, press the **Update Snapshot** button. You can also choose **Update Snapshot** from the Administration menu.

Server Snapshot Information

This section of the wizard displays basic information about the server and this snapshot.



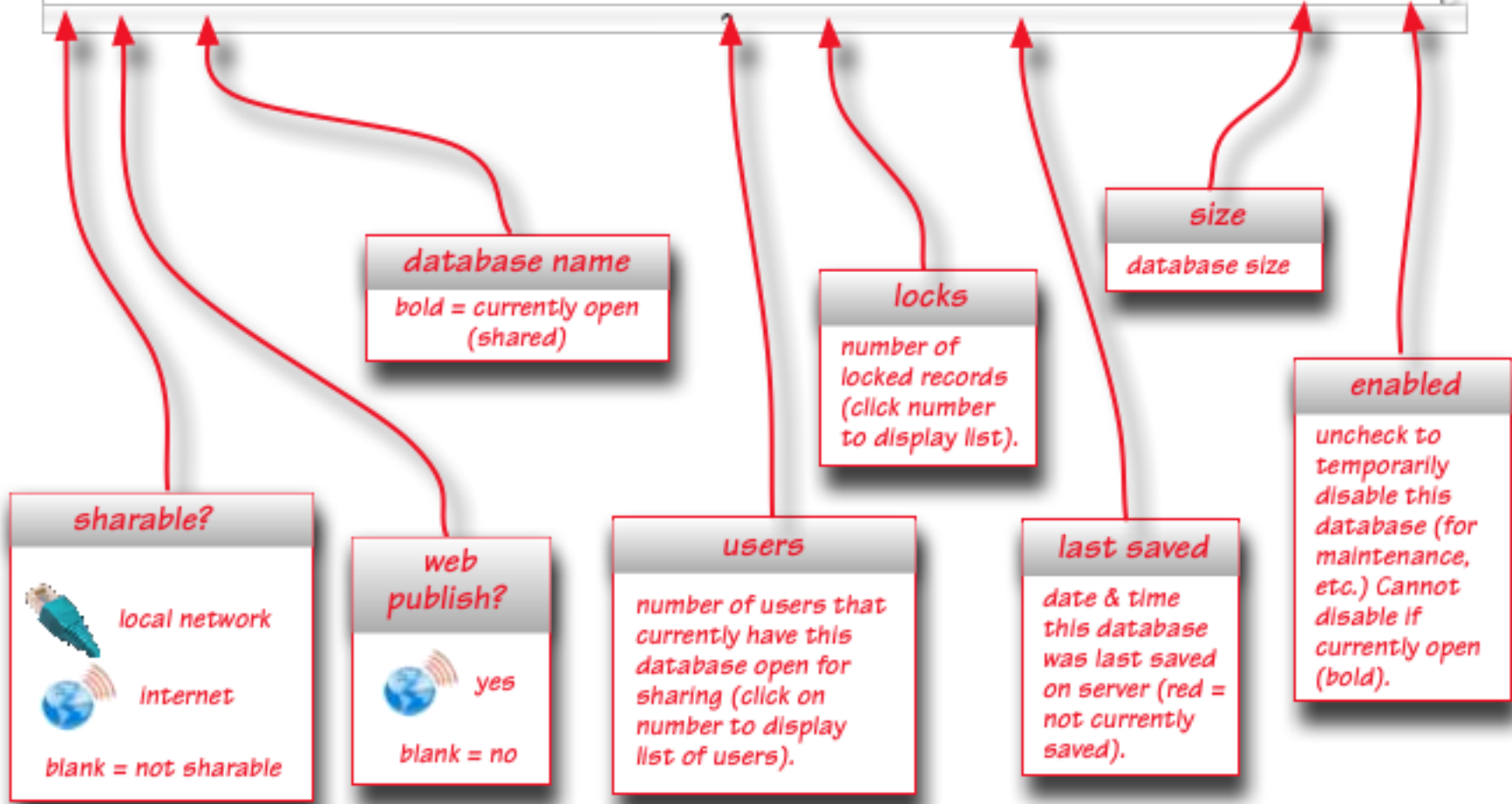
Server Database List

This section of the wizard displays all of the databases on the server and their status. Both open and closed databases are listed. (Installing databases on the server is explained in the next chapter, see “[Creating a Shared Database](#)” on page 103).

Server Databases

There are currently 13 databases on this server (1 currently open, 0 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|----------------------------|----------|-------|-----------------------|----------------|-------------------------------------|
| Cities | | | Jul 9, 2005, 3:42 PM | 2.5 KB | <input checked="" type="checkbox"/> |
| Corporate Checkbook | <u>1</u> | | Jan 12, 2006, 6:55 PM | 79.1 KB | <input checked="" type="checkbox"/> |
| Fair Price List | | | Nov 10, 2005, 5:01 PM | 6.7 KB | <input checked="" type="checkbox"/> |
| FairRealEstate | | | Dec 7, 2005, 9:30 PM | 39.5 KB | <input checked="" type="checkbox"/> |



Listing Locked Records

The database table also lists how many records in each database are locked. In the example below the [Corporate Checkbook](#) database currently has one locked record.

Server Databases

There are currently 58 databases on this server (1 currently open, 3 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|----------------------------|----------|----------|-----------------------|----------------|-------------------------------------|
| Contacts | | | Apr 14, 2006, 4:42 PM | 4.81 MB | <input checked="" type="checkbox"/> |
| Corporate Checkbook | <u>1</u> | <u>1</u> | Jan 12, 2006, 6:55 PM | 80.9 kB | <input checked="" type="checkbox"/> |
| Customers | | | May 16, 2007, 4:50 PM | 2.55 MB | <input checked="" type="checkbox"/> |

Clicking on this number opens a dialog with a list of locked records. Only the record ID's are listed, which usually isn't very helpful. Instead of using this dialog we recommend using the **Locked Record** wizard, see "[Locked Records Wizard](#)" on page 142.

The Pop-up Database Context Menu

If you hold down the **Control** key while you click on a row in the database table (or right-click if you have a two button mouse) a pop-up context menu will appear. The exact contents of this menu depend on the current status of the database, but it will look something like this:

Server Databases

There are currently 14 databases on this server (1 currently open, 0 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|-----------------------------|-------|-------|-----------------------|----------------|-------------------------------------|
| Offline Mailing List | | | Nov 4, 2005, 12:40 PM | 21.0 KB | <input checked="" type="checkbox"/> |
| Real Estate Listings | | | Feb 12, 2006, 2:35 PM | 48.9 KB | <input checked="" type="checkbox"/> |
| StateAbbreviationsTi | | | Sep 1, 2005, 4:26 PM | 2.2 KB | <input checked="" type="checkbox"/> |
| Test X Price List | | | Nov 11, 2005, 4:21 PM | 6.7 KB | <input checked="" type="checkbox"/> |

| |
|---|
| <input checked="" type="checkbox"/> Database Online |
| Database Offline |
| Display Users... |
| Delete from Server... |

You can use this pop-up menu to temporarily disable a database, to display the currently logged in users of a database, and to delete a database from the server.

Database Online/Database Offline. The first two commands in the pop-up menu work as a pair. They allow you to temporarily take a database offline (for example for maintenance) without actually deleting the database from the server. The checkmark in the menu indicates whether the database is currently online or offline (in the example above the [Real Estate Listings](#) database is currently online). Note: You can also see and modify the online/offline status of a database using the **On** column at the far right of the table.

Server Databases

There are currently 14 databases on this server (1 currently open, 1 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|----------------------|-------|-------|------------------------|----------|-------------------------------------|
| Forum | | | Nov 4, 2005, 1:04 PM | 16.6 KB | <input checked="" type="checkbox"/> |
| HobbyShopCatalog | | | 12, 2006, 7:08 PM | 150.9 KB | <input type="checkbox"/> |
| Invoices | | | Jun 27, 2005, 11:19 AM | 48.5 KB | <input checked="" type="checkbox"/> |
| Offline Mailing List | | | Nov 4, 2005, 12:40 PM | 21.0 KB | <input checked="" type="checkbox"/> |

database is offline

Notice that when a database is temporarily offline, it is dimmed (gray) in the database table.

Display Users. If a database is a shared database (not just web publishing) and is currently online you can use this command to display a list of the currently active users. This is the same as clicking on the number of users (see “[Listing Database Users](#)” on page 71).

Delete From Server. This command completely deletes this database from the server. This command is permanent and cannot be reversed or undone. You cannot delete an online database from the server. First you must take the database offline (see above), then you can delete it.

Browse Database Web Links. If the database is configured for web database publishing, you can use this option to display all of the web links (forms and procedures) for this database in your web browser.

Server Databases

There are currently 36 databases on this server (0 currently open, 0 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|------------------|-------|-------|------------------------|----------|-------------------------------------|
| Greek2 | | | Mar 19, 2006, 12:04 PM | 3.4 KB | <input checked="" type="checkbox"/> |
| HobbyShopCatalog | | | Aug 29, 2005, 9:12 PM | 150.7 KB | <input checked="" type="checkbox"/> |
| Invoices | | | Jun 27, 2005, 11:19 AM | 48.5 KB | <input checked="" type="checkbox"/> |
| Lookups | | | Jun 6, 2006, 8:26 AM | 14.9 KB | <input checked="" type="checkbox"/> |

Server Memory Usage

The 0 currently open databases use 0.0 MB of memory. 98.45 MB free (100.0%).

See “[Testing a Web Database](#)” on page 199 for more information on this command.

Close Database. If the database is configured for web database publishing you can use this option to close the database (for example for maintenance).

Server Databases

There are currently 36 databases on this server (0 currently open, 0 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|------------------|-------|-------|------------------------|----------|-------------------------------------|
| Greek2 | | | Mar 19, 2006, 12:04 PM | 3.4 KB | <input checked="" type="checkbox"/> |
| HobbyShopCatalog | | | Mar 19, 2006, 9:52 PM | 150.7 KB | <input checked="" type="checkbox"/> |
| Invoices | | | Mar 17, 2005, 11:19 AM | 48.5 KB | <input checked="" type="checkbox"/> |
| Lookups | | | Mar 19, 2006, 8:26 AM | 14.9 KB | <input checked="" type="checkbox"/> |

- Database Online
- Database Offline
- Display Users...
- Browse Database Web Links...
- Close Database**
- Delete from Server...

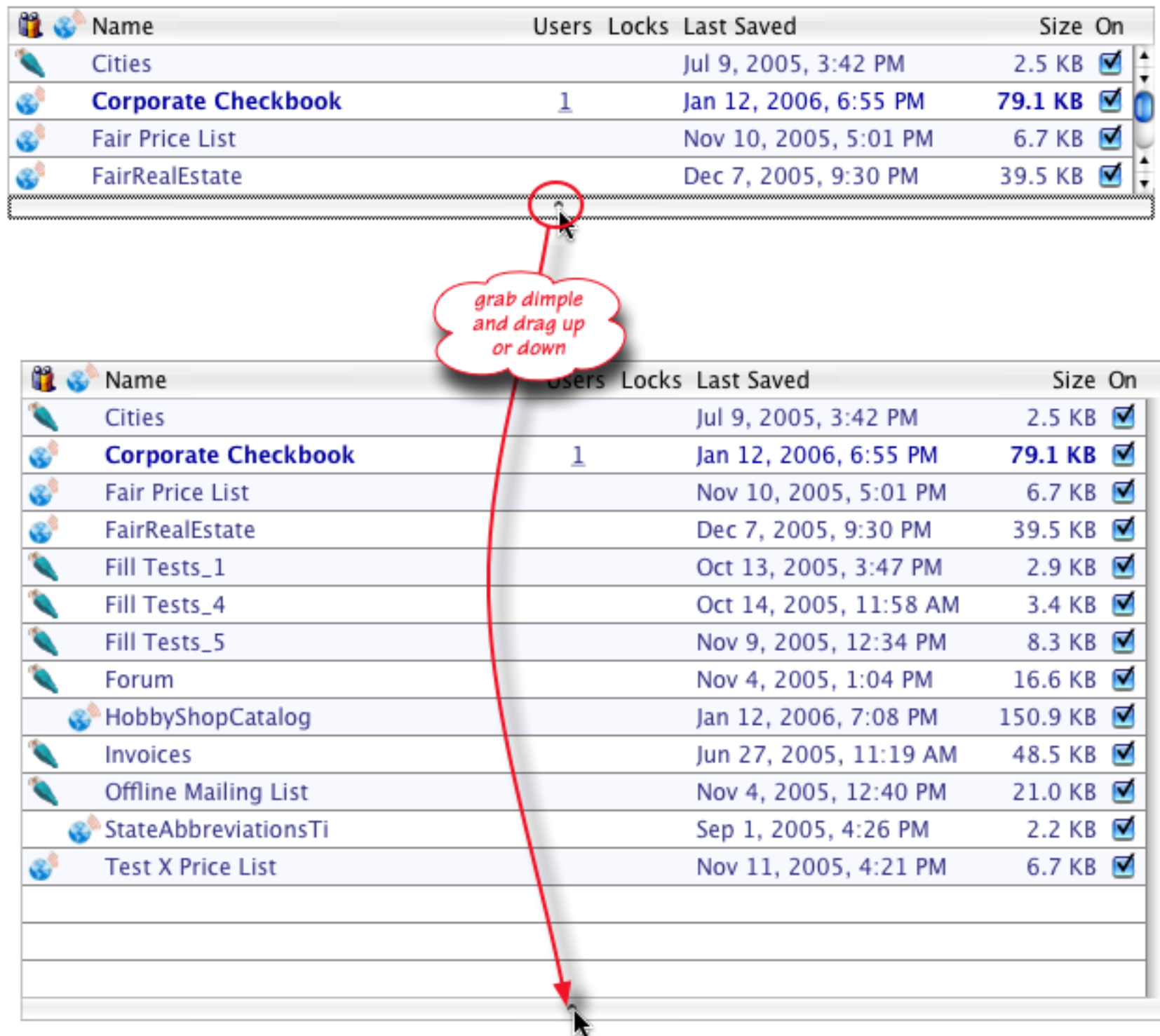
Server Memory Usage

The 0 currently open databases use 0.00 MB (0.0% of memory, leaving 98.45 MB free (100.0%).

This command cannot be used to close a database that has been opened for database sharing.

Adjusting the Table Size

If the server contains many databases you can scroll the table to see the additional databases. You can also expand the table by grabbing the “dimple” at the bottom of the table and dragging it down.



The image shows two screenshots of a database table. The top screenshot shows a table with four rows. The bottom screenshot shows the same table with many more rows, and a red callout bubble pointing to a small dimple at the bottom of the table. The callout bubble contains the text: "grab dimple and drag up or down".

| Name | Users | Locks | Last Saved | Size | On |
|----------------------------|----------|-------|------------------------|----------------|-------------------------------------|
| Cities | | | Jul 9, 2005, 3:42 PM | 2.5 KB | <input checked="" type="checkbox"/> |
| Corporate Checkbook | <u>1</u> | | Jan 12, 2006, 6:55 PM | 79.1 KB | <input checked="" type="checkbox"/> |
| Fair Price List | | | Nov 10, 2005, 5:01 PM | 6.7 KB | <input checked="" type="checkbox"/> |
| FairRealEstate | | | Dec 7, 2005, 9:30 PM | 39.5 KB | <input checked="" type="checkbox"/> |
| Fill Tests_1 | | | Oct 13, 2005, 3:47 PM | 2.9 KB | <input checked="" type="checkbox"/> |
| Fill Tests_4 | | | Oct 14, 2005, 11:58 AM | 3.4 KB | <input checked="" type="checkbox"/> |
| Fill Tests_5 | | | Nov 9, 2005, 12:34 PM | 8.3 KB | <input checked="" type="checkbox"/> |
| Forum | | | Nov 4, 2005, 1:04 PM | 16.6 KB | <input checked="" type="checkbox"/> |
| HobbyShopCatalog | | | Jan 12, 2006, 7:08 PM | 150.9 KB | <input checked="" type="checkbox"/> |
| Invoices | | | Jun 27, 2005, 11:19 AM | 48.5 KB | <input checked="" type="checkbox"/> |
| Offline Mailing List | | | Nov 4, 2005, 12:40 PM | 21.0 KB | <input checked="" type="checkbox"/> |
| StateAbbreviationsTi | | | Sep 1, 2005, 4:26 PM | 2.2 KB | <input checked="" type="checkbox"/> |
| Test X Price List | | | Nov 11, 2005, 4:21 PM | 6.7 KB | <input checked="" type="checkbox"/> |

To shrink the table, just grab the dimple and drag up.

Disabled Sharing/Web Publishing Warning

Usually you'll configure your server to that it automatically enables sharing and/or web publishing when it starts up. If one of these modes is disabled the Server Administration window will warn you with a message above the list of databases. The message shown below means that server is running, but local sharing is not enabled and clients on the local network will not be able to work with shared databases.





Server Snapshot (Jim-Reas-Computer-2)

The information on this page was last updated at 1:00 PM local time (1:00 PM server time). The server has been running for 1 day, 2 hours, 54 minutes.

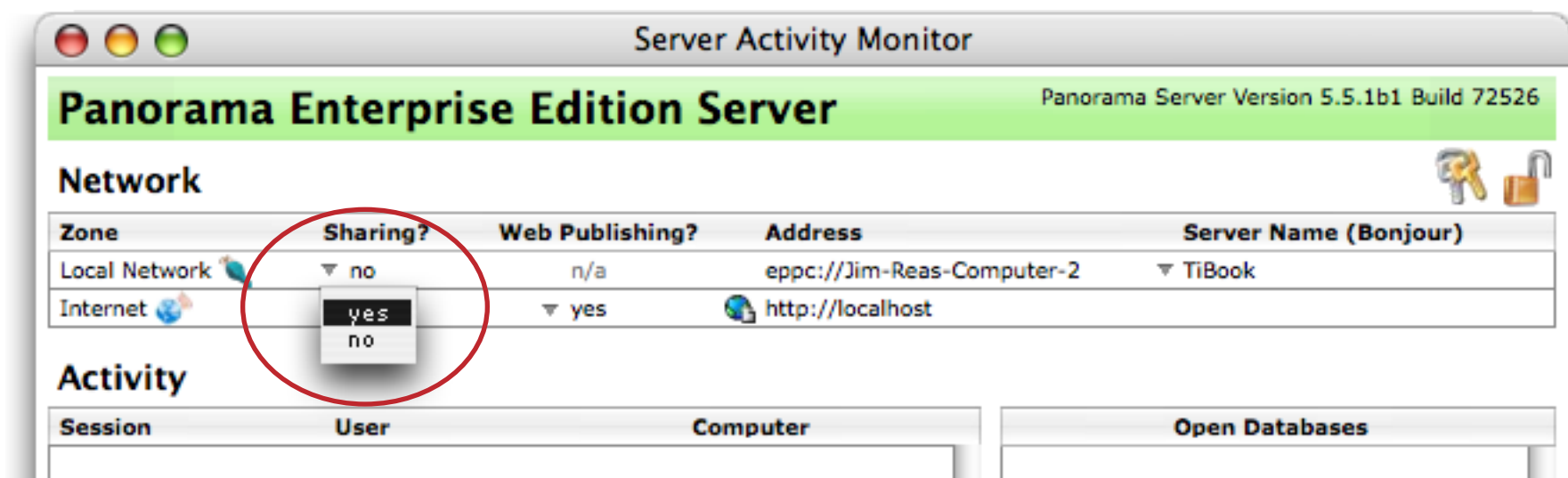
Server Databases

No Local Sharing

There are currently 58 databases on this server (0 currently open, 3 temporarily offline.)

|   | Name | Users | Locks | Last Saved | Size | On |
|---|--------------------|-------|-------|-----------------------|---------|-------------------------------------|
|  | #1 HHVC price list | | | May 31, 2006, 9:32 PM | 882 kB | <input checked="" type="checkbox"/> |
|  | #1 Patients | | | Mar 13, 2007, 5:48 PM | 11.0 MB | <input checked="" type="checkbox"/> |

If you see this message you'll probably want to go to the server and enable local sharing.



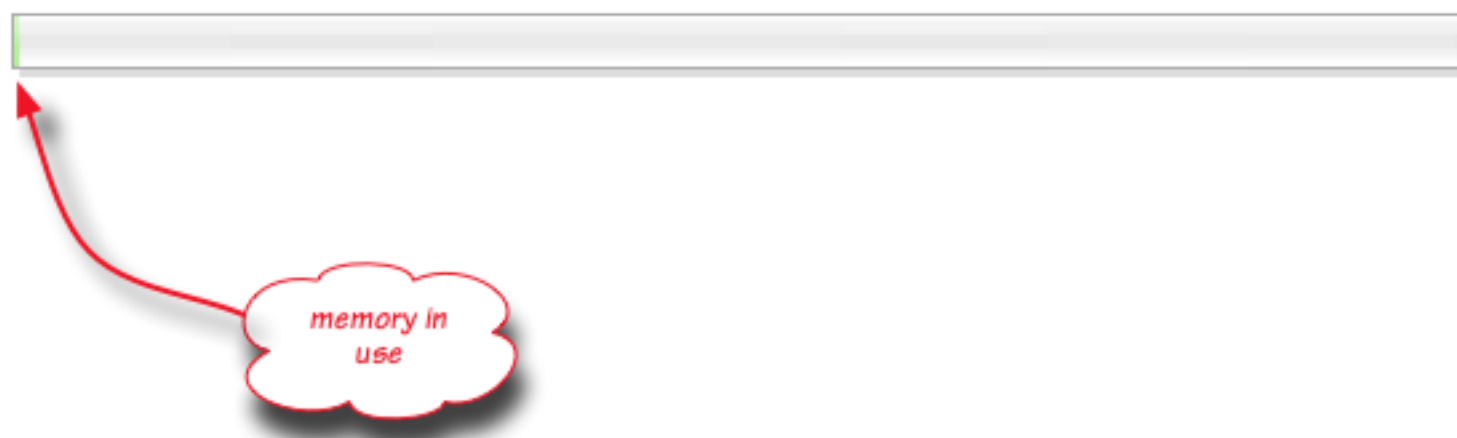
See “[Enabling Database Sharing](#)” on page 55 and “[Enabling Database Web Publishing](#)” on page 56 for more information about enabling these modes.

Server Memory Usage

This section of the wizard displays the amount of memory used and available on the server, in both numeric and graphical (bar graph) formats.

Server Memory Usage

The 1 currently open database uses 0.08 MB (0.2%) of memory, leaving 30.60 MB free (99.8%).



Adjusting Panorama's Memory Allocation

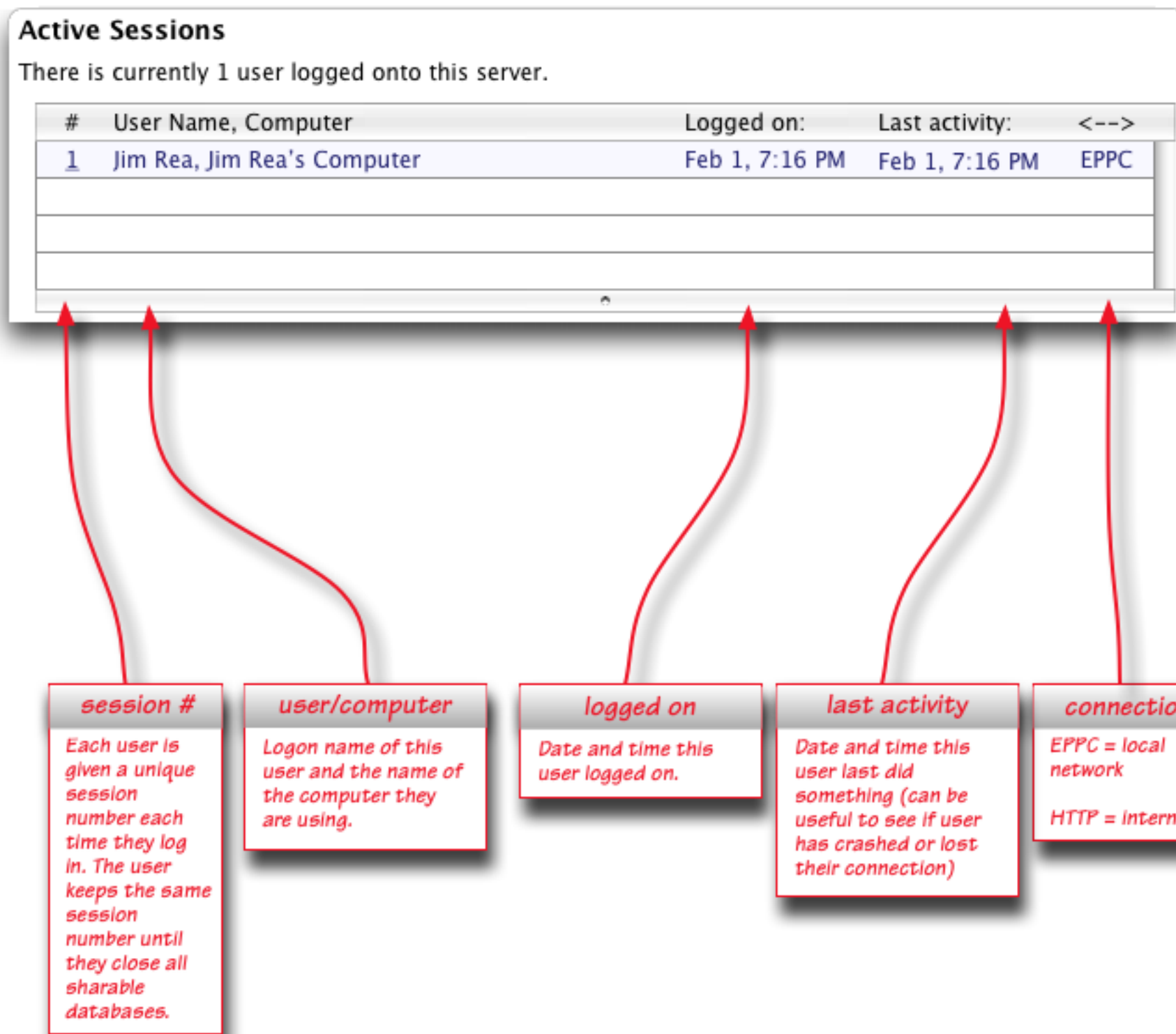
As shipped from the factory, Panorama normally allocates 96 megabytes of memory for databases. For most applications this is more than enough. However, if you wish to use extremely large databases you may need to increase this allocation. To do this, locate the [PanoramaPowerPC.ini](#) file. This file is located in the [Applications:Panorama](#) folder (the same folder as the Panorama application itself). Double click this file to open it in TextEdit. (Note: Be sure you open [PanoramaPowerPC.ini](#) 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:

```
PanoramaPowerPC.INI
databasememory 96M
target PanoramaNative
rez panorama.rsrc
rez panoramadiagnostics.rsrc
race
```

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 its performance may be severely degraded. Once you have set the new value, save and close the window, then relaunch Panorama Server if necessary.

Active Session List

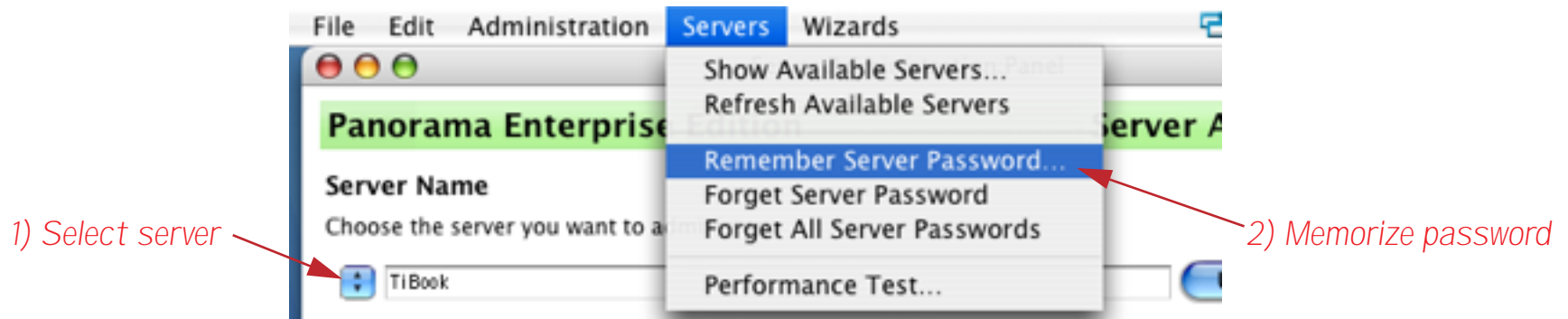
This section of the wizard lists all of the currently active sharing “sessions” taking place on the server. When a user first opens a shared database on a client computer, their copy of Panorama automatically connects to the server and establishes a “session.” Each session has a unique number, and all communications between the client and the server are tagged with the session number. If the user opens additional shared databases on the same server, the same session number is used. The session remains active until the user closes all shared databases on this server. (If the user later opens a shared database, again a new session number will be assigned.) The active session list displays a list of all of the users that currently have one or more databases open on this computer.



Memorizing Server Passwords

If you are doing a lot of work on a particular server, you may find it convenient to have Panorama remember the password for that server for you. Be careful, though, since doing this means that anyone with access to your computer will also have administrator level access to this server.

Before you ask Panorama to remember a password, you must already have selected the server in the **Server Administration** wizard and entered the password. Then choose **Remember Server Password** from the **Servers** menu.

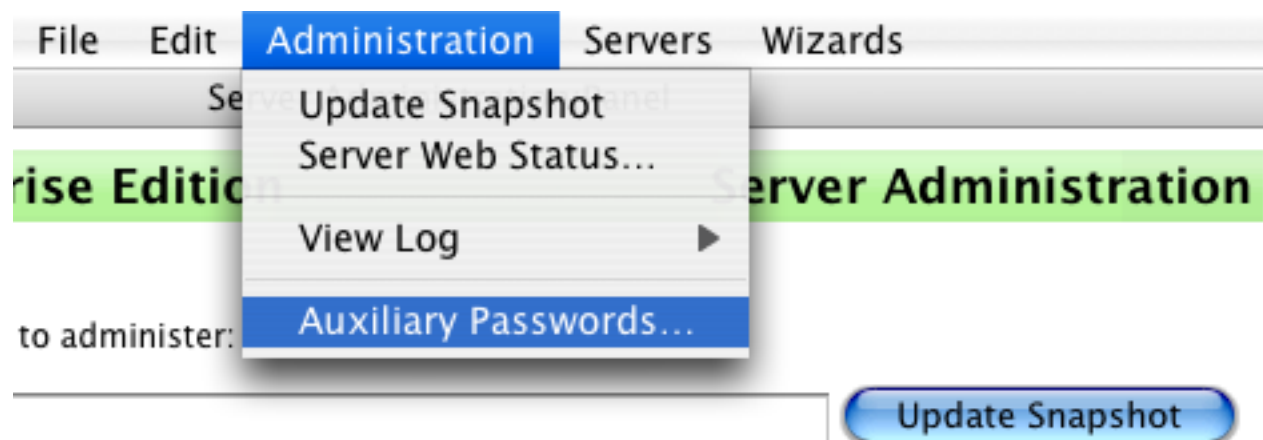


Once the wizard has memorized the password, you won't have to enter the password any more, even if you quit Panorama and then restart it later.

If you later decide that you no longer want Panorama to remember the password for the current server then choose the **Forget Server Password** command. The **Forget All Server Passwords** command causes Panorama to forget all server passwords, not just the password for the current server. For example you might want to use this command if you are taking your computer on the road or to a trade show, or any situation where other people might have access to your computer.

Changing the Auxiliary Passwords

In addition to the main server password, there are also three optional auxiliary passwords: two for downloading and one for displaying server status from web browsers. You can edit these passwords using the **Auxiliary Passwords** command in the **Administration** menu.



(You can also open this dialog on the server itself, see “[Changing the Auxiliary Passwords](#)” on page 52.)

Auxiliary Passwords

Download Databases from Local Clients:

Enter a password to restrict downloading databases from clients on the local network. This password is used by the Download Shared Databases wizard. If left blank anyone on the local network can download shared databases from this server using this wizard.

Download Databases from Remote Clients:

Enter a password to restrict downloading databases from clients on the internet. This password is used by the Download Shared Databases wizard. If left blank anyone on the internet can download shared databases from this server using this wizard.

Server & Database Web Status Pages:

Enter a password to restrict access to server and database status pages from web browsers. If left blank then anyone on the internet will be able to view a list of web published databases on this server, as well as lists of forms and procedures in each database.

Use these auxiliary passwords if you want to restrict access to downloading databases (see “[Using the Download Shared Databases Wizard](#)” on page 110) and/or displaying web status (see “[Testing Web Database Publishing \(Server Status\)](#)” on page 64). If used, each auxiliary password should be unique and should also be different from the main server password.

If an auxiliary password is left blank then that operation is not restricted. For example, if you want to allow anyone on your local network to download databases without a password, just leave the **Download Databases from Local Clients** password blank.

You can change these passwords at any time.

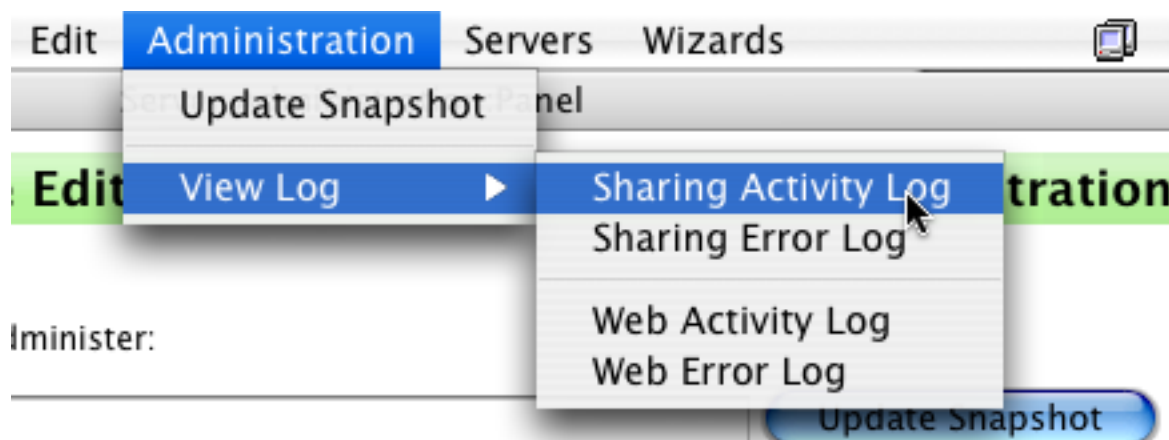
Monitoring Server Logs

The Panorama Server keeps logs of its activity and any errors that it encounters. You can use the **Server Administration** wizard to view these logs remotely. The server keeps four different logs:

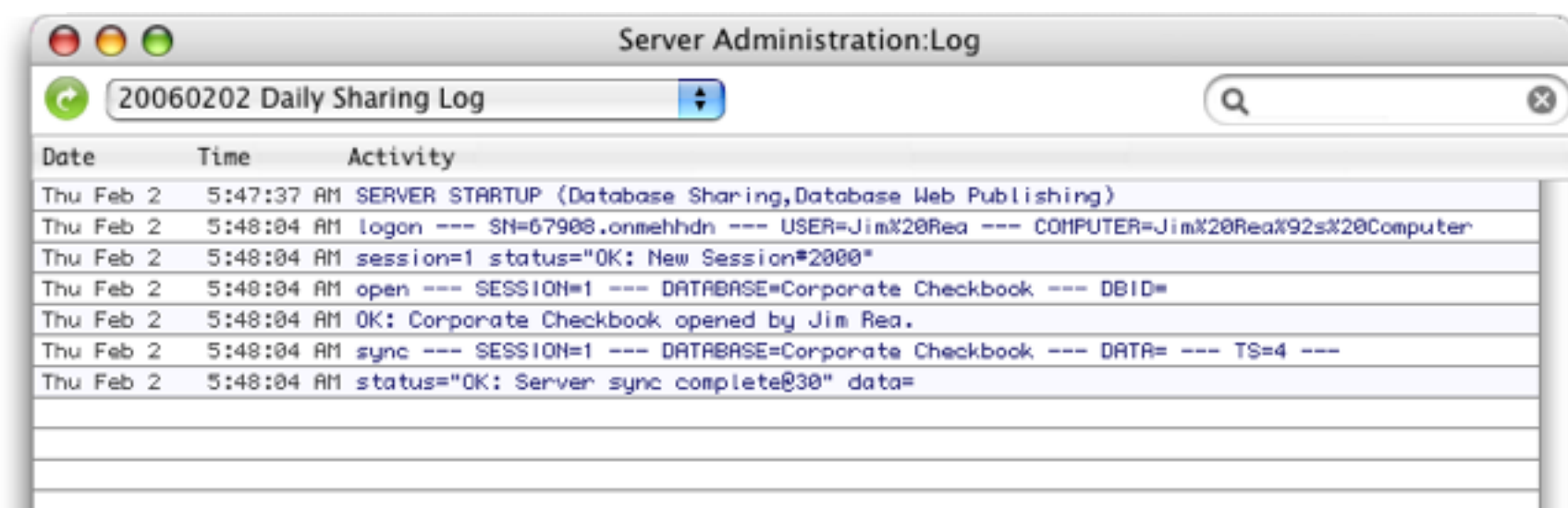
| Log Name | Description |
|----------------------|--|
| Sharing Error Log | This log records errors that occur when sharing databases. For example, this log will record crash recovery or unsuccessful attempts to log in to the server. |
| Sharing Activity Log | This log records normal database sharing activity. Activities that can be logged include server startup and shutdown, users logging on and off, opening and closing databases, saving and synchronizing databases, and data editing. The system administrator can decide which of these activities should be logged. |
| Web Error Log | This log records errors that occur when publishing databases as web pages. For example, this log will record missing databases or coding errors. |
| Web Activity Log | This log records the web queries (URL's and POST data) that come in to the server. The system administrator can decide whether GET or POST requests (or both) are logged. |

Viewing a Log

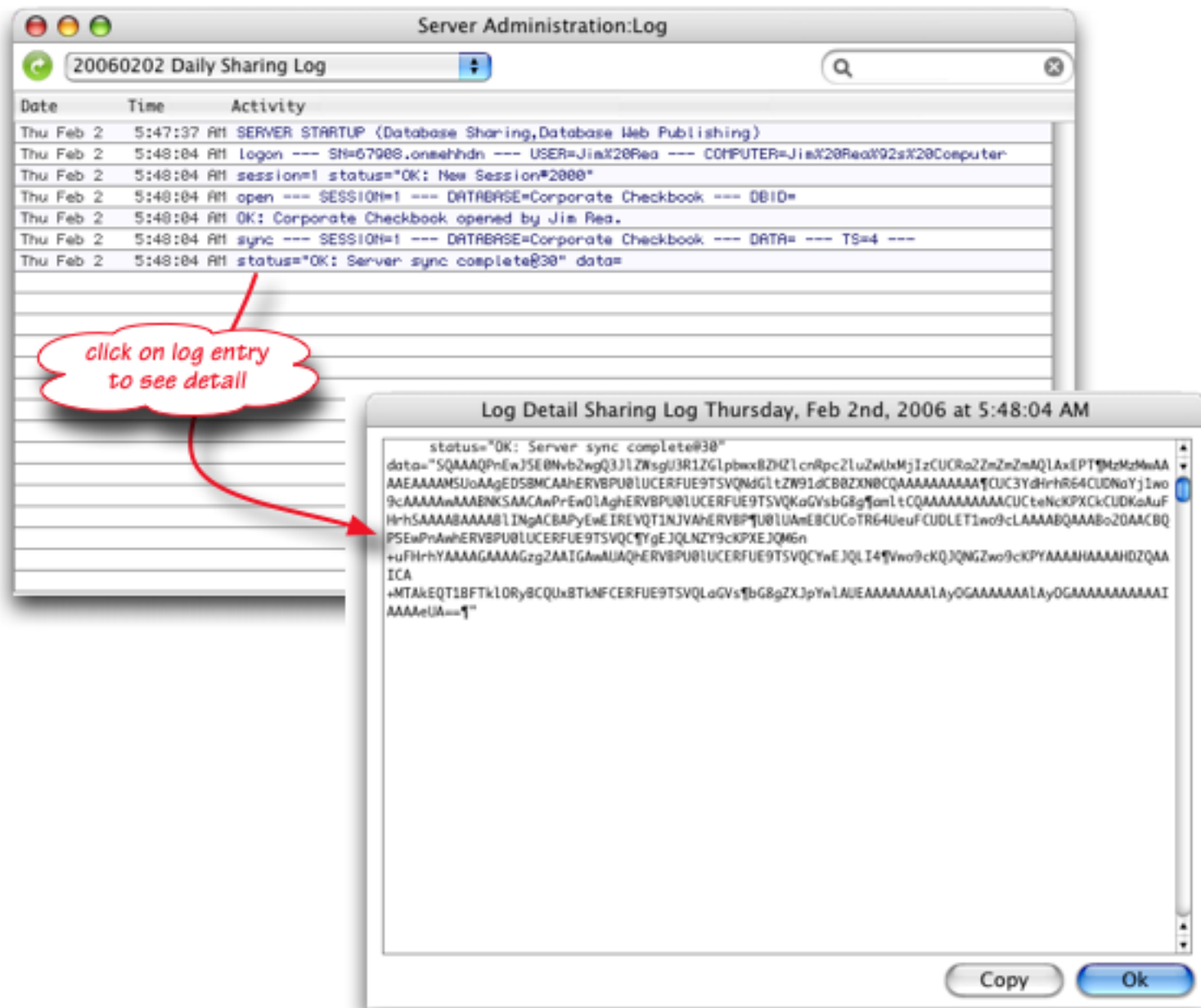
To view a log use the **View Log** submenu of the **Administration** menu.



The wizard will open the most recent log. Log entries are displayed with the most recent at the bottom.

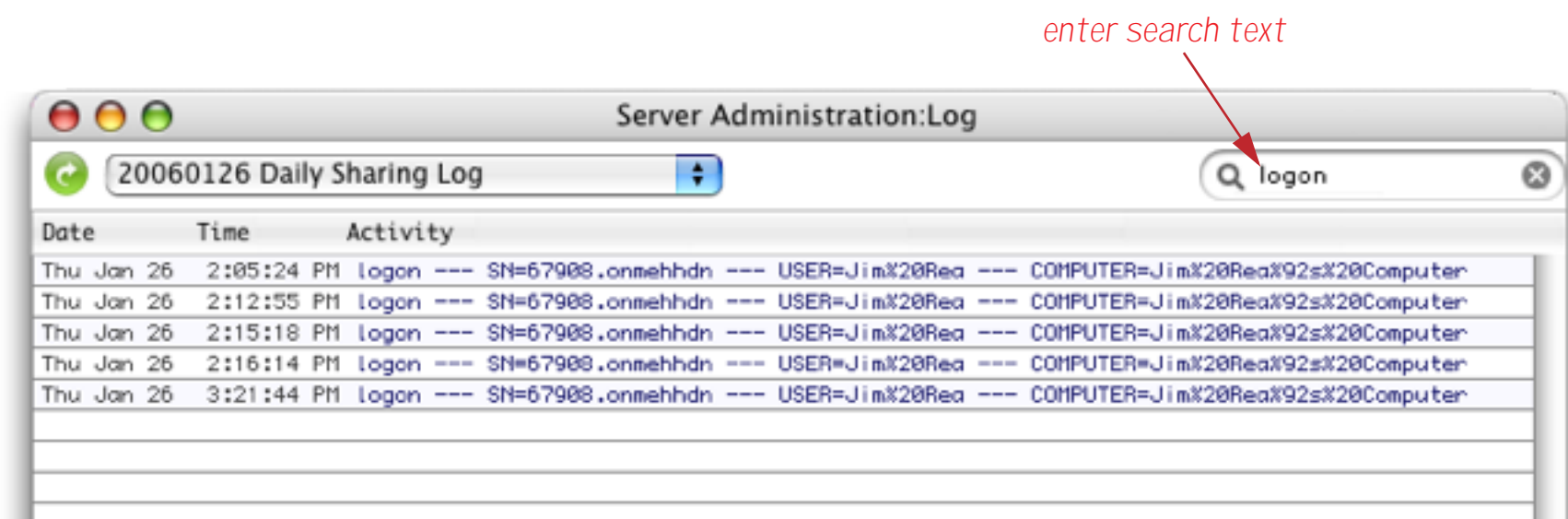


You can click on any line in the log to see additional detail.



Searching the Log

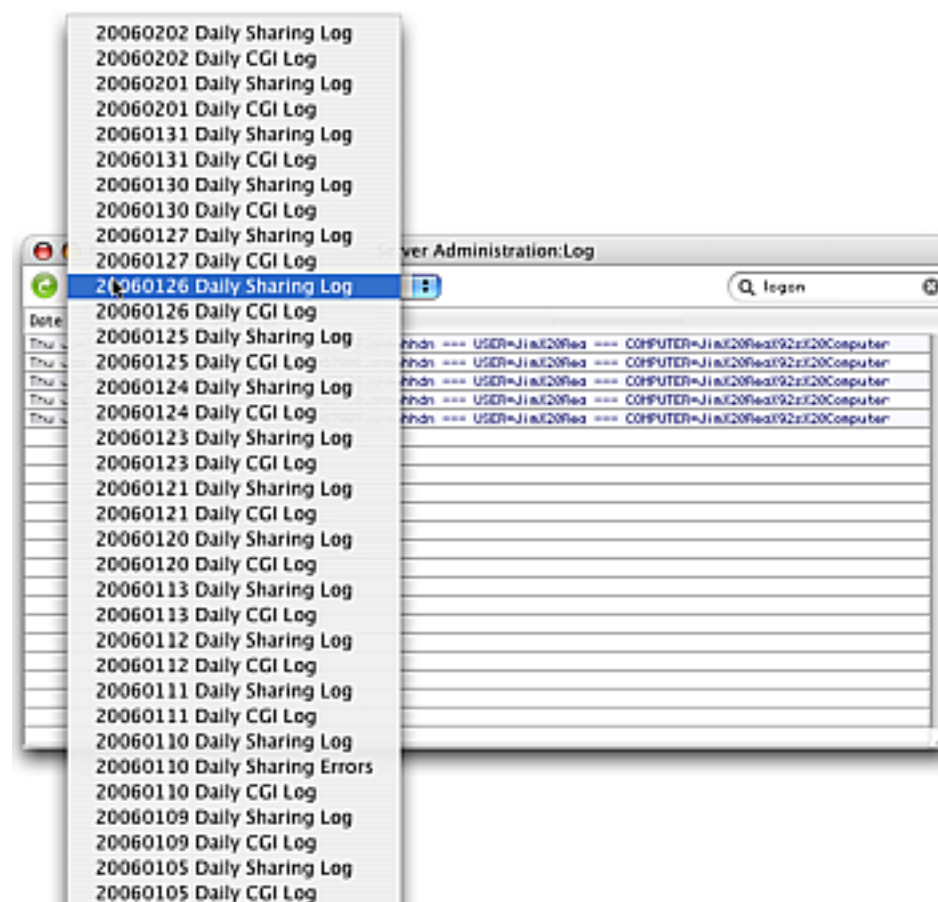
If a log contains hundreds or thousands of entries, it may be difficult to find the entry you want. To search for specific log entries, type a word or phrase into the search box in the upper right hand corner. For example, you can search for **logon** to quickly find out who logged on and when.



To go back to the full display, press the  cancel search button.


Viewing a Different Log

You can use the pop-up menu to switch to a different log.



The first eight digits of each line are the date of the log, in year month day format.

Refreshing the Log

The log display is a “snapshot” in time - it is not updated as new server activity occurs. To update the log display to show the most recent server activity, press the  button.

Server Log Configuration

The **Server Administration** wizard allows you to configure how logs are kept.

Database Sharing Logs

Click on a link to view the corresponding log.

- [Sharing Error Log](#): Daily Weekly Monthly
[Sharing Activity Log](#): Daily Weekly Monthly
- Logged Activities: Startup & Shutdown
 Log on & off
 Open & Close databases
 Save & Synchronize databases
 Data Modification (Individual Records)
 Bulk Data Modification (Fill, Append, etc.)

Web Publishing Logs

Click on a link to view the corresponding log.

- [Web Server Error Log](#): Daily Weekly Monthly
[Web Activity Log](#): Daily Weekly Monthly
- Logged Activities: Get requests (single line queries)
 Post requests (forms)

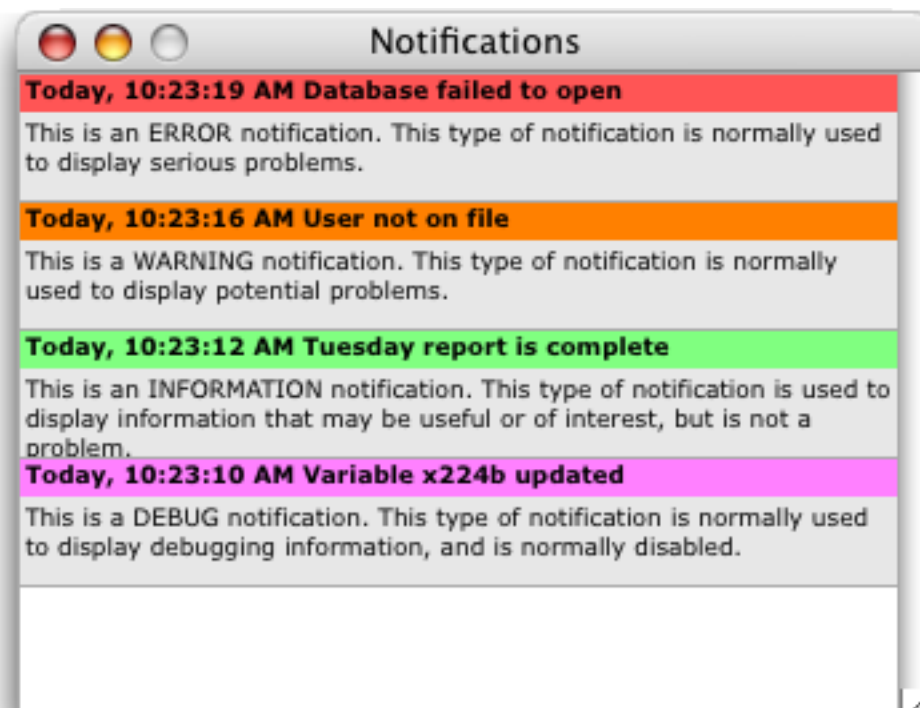
For each log you have a choice of Daily, Weekly or Monthly. This specifies how often the server will start a new log file. If you don't have much server activity you may want to keep a Monthly file, but if your server is busy you will probably want to use Daily or Weekly logs to keep the log file size more manageable.

The **Logged Activities** options control how much detail is recorded in the log. Recording more detail can adversely affect the performance of the server. For best performance, we recommend not logging Data Modification (the last two sharing options).

Configuring the Notification Wizard

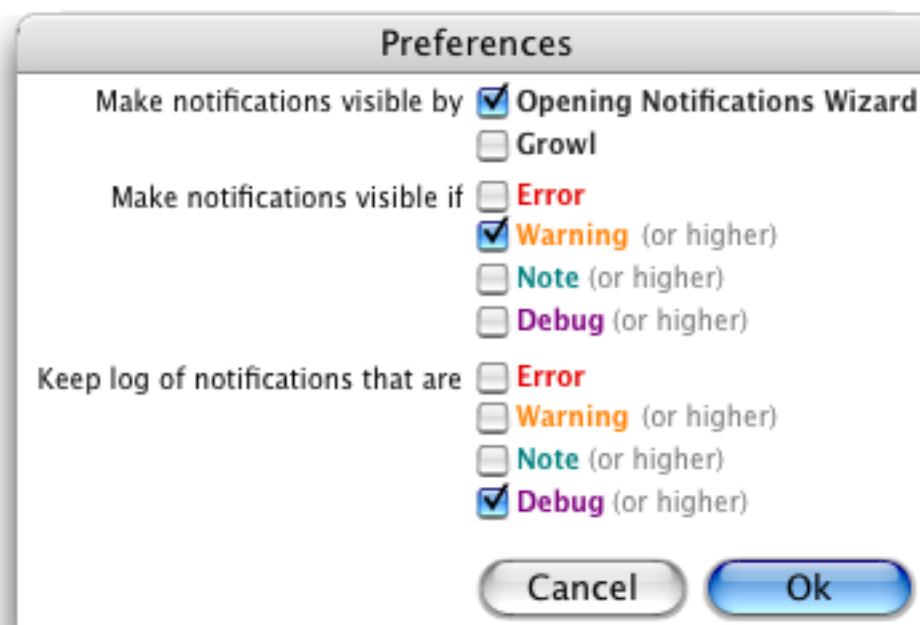
Panorama uses the **Notification Wizard** to inform you if problems occur during database sharing operation. For example, if there is a connection problem when opening a shared database, this wizard will appear automatically. Unlike an alert, the **Notification Wizard** doesn't interrupt your work by requiring you to dismiss the message before continuing (this is especially true if you use the optional **Growl** package, see "[Using Growl for Notifications](#)" on page 87). You can also review problems that occurred earlier in the session.

The **Notification** wizard displays a list of the notifications that have occurred since Panorama was launched. The most recent notification is at the top, earlier notifications are listed below. There are four types of notifications: Errors (red), Warnings (orange), Information (green), and Debug (purple). Each notice has a title and additional detail, as shown below (the notifications shown below are just examples, and do not reflect the actual notifications you may see).

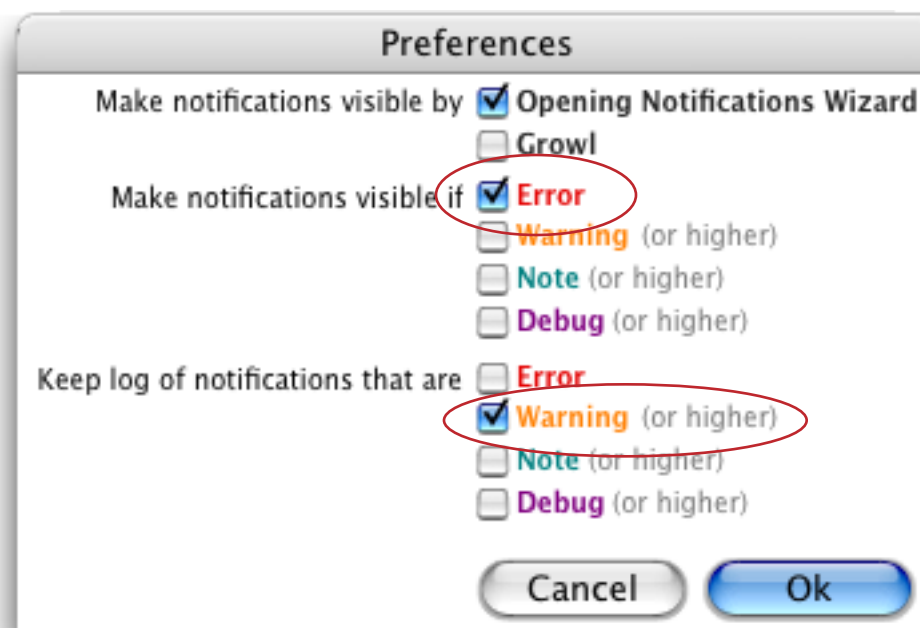


When does the Notification Wizard appear?

By default, the **Notification** wizard will appear automatically whenever an error or warning occurs, but not when an information or debug notice occurs. However, you can use the **Preferences** command (in the **Admin** menu) to change this behavior.



For example, you can modify the settings so that only **Errors** make the wizard appear automatically, and so that information (notes) and debug notices are simply discarded (they won't even appear in the wizard if you open it manually).



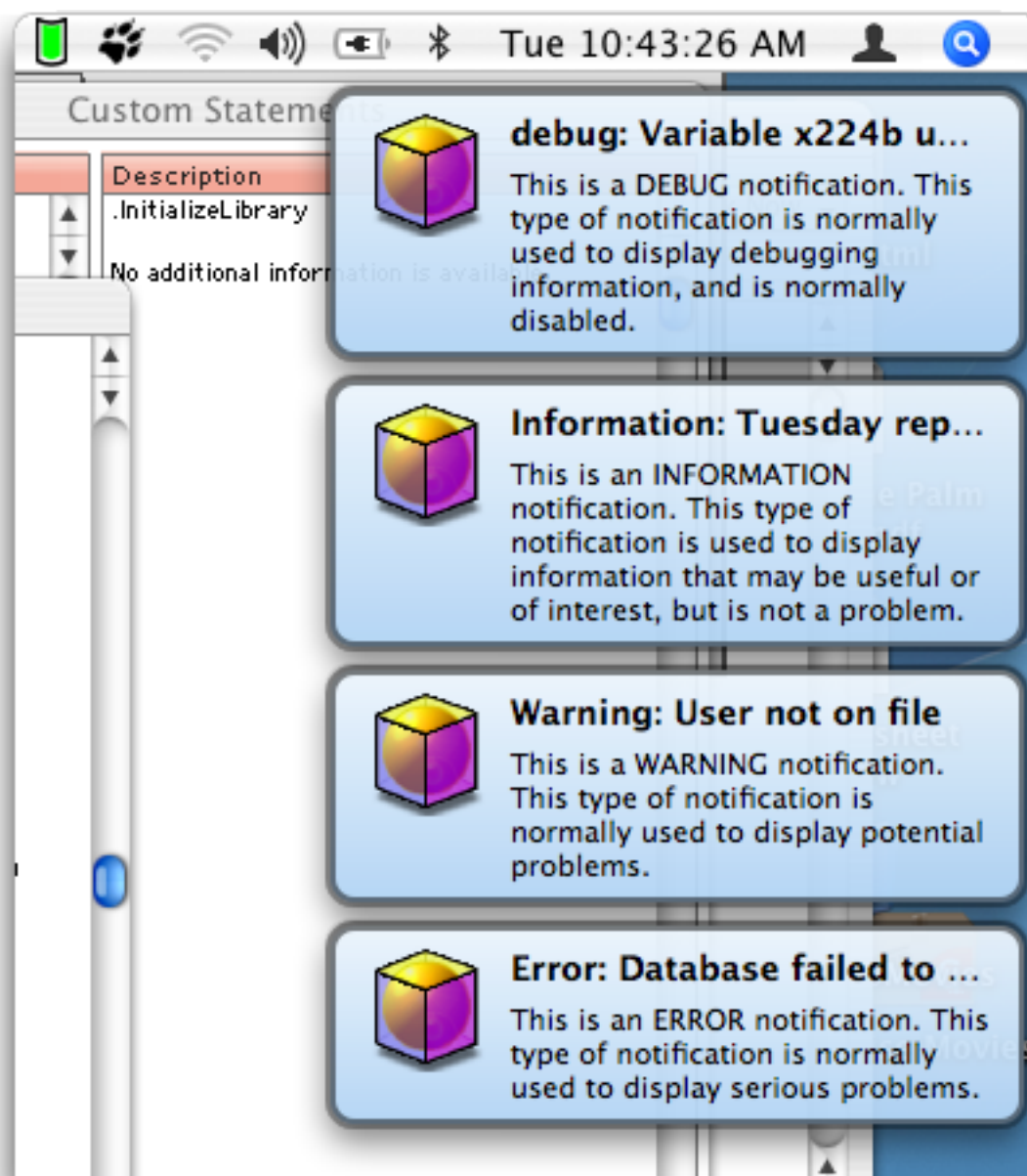
Note: The notification settings for each computer are separate. Setting the notification settings on the server has no effect on any client, and changing the notification options on any client doesn't affect the server or any other client. In other words, if you want to make a change across the network you'll need to go to each computer and make the change.

Using Growl for Notifications

Growl is a very cool free open source add-on for OS X that displays temporary messages that fade away automatically after a few seconds. In other words, a perfect way to display notifications! If Growl is installed on your computer then the **Notification** wizard Preferences dialog gives you the option of displaying notifications via Growl instead of by opening the **Notification** wizard (you can always open the **Notification** wizard manually from the **Wizard** menu). If you don't already have a copy of Growl, you can download it from this web site.

<http://growl.info/>

It's a small download (2 mb) and installs easily. Once it's installed, you can enable the Growl option and your notices will appear in floating "bubbles" while you can continue to work. (In the example below we also set the visible notification level to "Debug".)



After a short delay, the notices will fade away, or they will disappear immediately if you click on them.

Using the Notification Wizard in your own database applications

Using the `notify` statement your applications can create their own notifications for errors and significant events. See the [Programming Reference](#) wizard for details on how to use this statement.

Advanced Server Configuration

Earlier in this chapter you learned how to set up basic server settings (password, server name, enabling and disabling database sharing and web publishing), see “[Basic Server Configuration](#)” on page 49. The following sections explain how to set up more advanced server options, including backup options, advanced record locking options, notification options, and more. The server sets up the defaults for these advanced options to settings that will work for most typical applications, so if you are simply setting up a basic server on a local network you may simply want to skip this section and go directly to the next chapter (you can always come back later).

Unlike the basic server configuration which must be done on the server itself, the advanced server configuration is usually done remotely from a client with the **Server Administration** wizard (see “[Server Management \(The Server Administration Wizard\)](#)” on page 68). To review, this wizard is opened directly from the **Sharing** submenu of the **Wizard** menu, or by double clicking on a server name in the **Available Servers** wizard (see “[Further Testing the Sharing Connection](#)” on page 63). You can see the advanced options by scrolling to the bottom of this wizard:

Notification Options
E-mail notification configuration.

Administrator Email:

Send e-mail notification of server errors
 Allow users to send e-mail to administrator

Backup Options
Automatic daily backup configuration.

Daily Backup Time: (example: 4:30 AM)
Backup Path:

Last Server Backup: 2/24/07 3:31 PM

Advanced Options
Consult the Panorama Enterprise Handbook before setting up these options.

Automatically unlock locked records after minutes.
Rebound after server crash if restarted within minutes.

Ask to confirm before quitting server
 Automatically hide server when launched
 Enable Server Activity Indicator

Internal/Debug Options
These options should only be used under the direction of ProVUE technical support staff.

Keep detailed log of last activities performed by the server.
Don't relock when rebounding, allow minutes grace for committing records.

Disable all alerts on server (warning -- may interfere with some wizards)
 Full Integrity Checks (usually not necessary, server will run much slower)
 Create XLG file for message statements

Unless otherwise noted, any configuration changes you make with the **Server Administration** wizard will take place immediately — you don't need to restart the server.

Notification Options

If an e-mail channel has been set up on the server computer (see “[Configuring an Email Channel](#)” on page 90) the Panorama server can automatically send the administrator e-mail to notify him or her of situations that may need attention. Enter your e-mail address into the **Administrator Email** box. Press the **Send Test Email** button to test to make sure that the e-mail address is correct and that e-mail is correctly configured on the server.

Notification Options
E-mail notification configuration.

Administrator Email:

Send e-mail notification of server errors

Allow users to send e-mail to administrator

If you want to be notified whenever there is a problem with the server, check the **Send e-mail notification of server errors** box. The server will send you an e-mail whenever an error occurs. The e-mail will contain the error, and you can look in the server logs for additional details about the situation.

The **Allow users to send e-mail to administrator** option enables the `mailtoadministrator` statement. This statement can be used in shared Panorama databases to send e-mail to the administrator. The syntax of this statement is:

```
mailtoadministrator from,subject,body
```

The `from` parameter is the e-mail address of the user sending the e-mail (can be left blank in which case the server’s e-mail address will be used). The `subject` parameter is the subject of the e-mail, while `body` contains the main part of the e-mail message. When you design a shared database you might include a button or menu item that sends a message to the administrator using this statement.

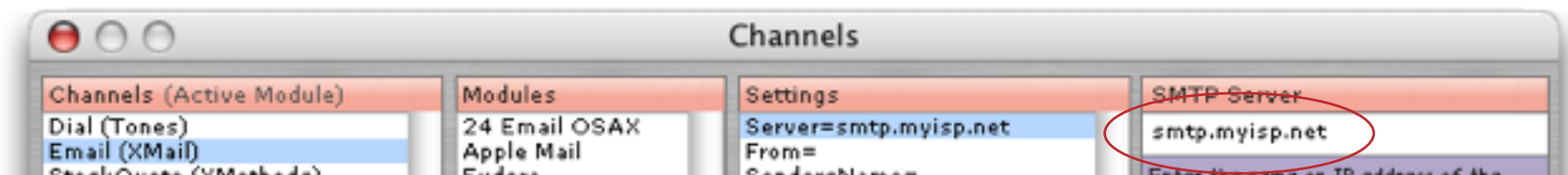
Configuring an Email Channel

If you want the server to send e-mails, you must set up an e-mail channel on the server computer. This must be done on the server computer itself, it cannot be done remotely. Go to the server computer and open the Panorama application (not Panorama Server). Then open the **Channels** wizard, which is in the **Preferences** submenu.

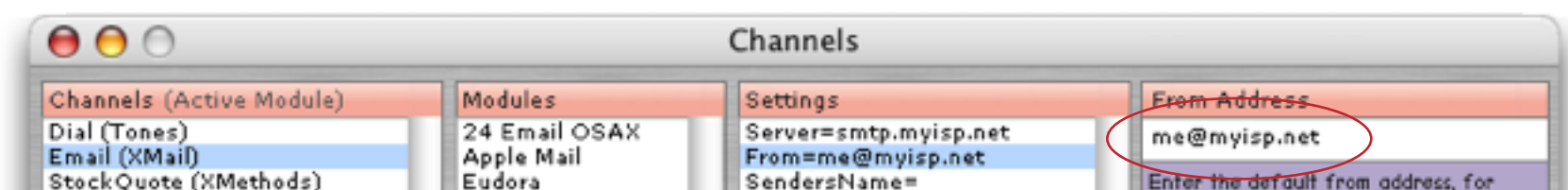


The first step is to click on **E-mail** in the left column, then select the module you want to use. In this case we've picked **XMail**, which uses a software package, called, surprisingly enough XMail (this software is separate and must be downloaded and installed before using it). You'll want to pick 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 XMail as an example. Since XMail 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.



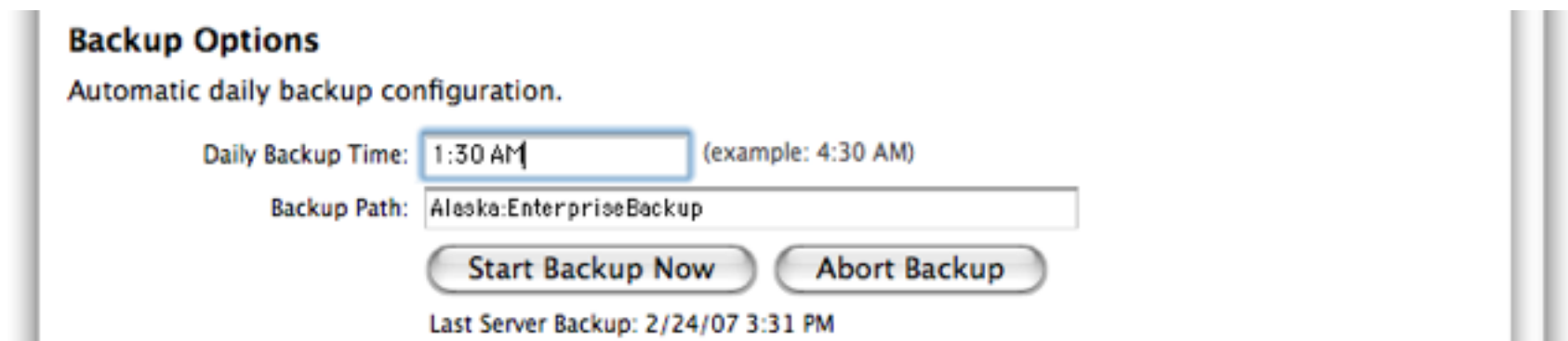
Now click on **From=** and type in your e-mail address.



Repeat to type in any additional information required by your ISP, then close the Channels wizard and quit Panorama when you are done.

Backup Options

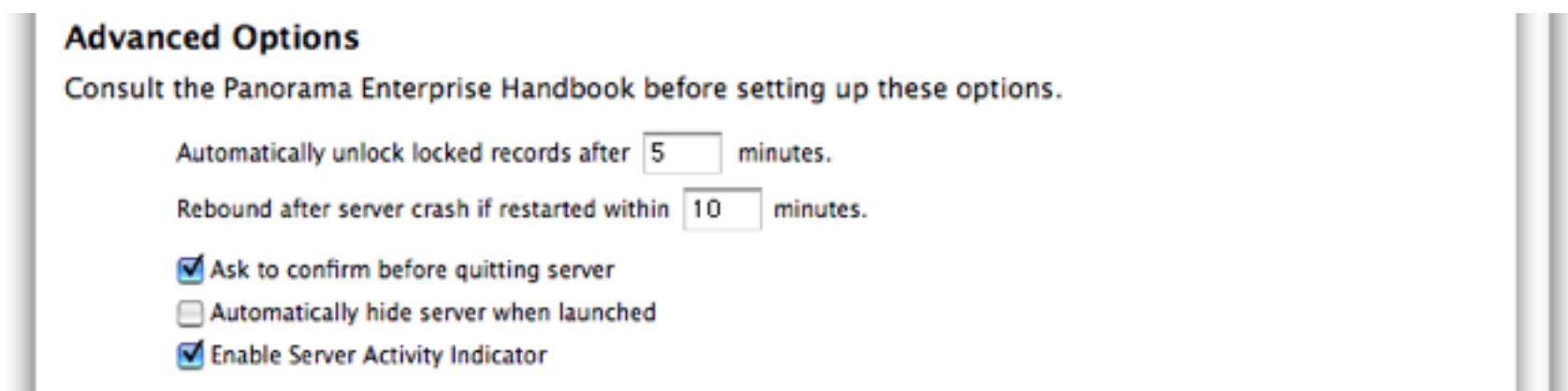
The Panorama Server can automatically do a daily first level backup of your database files to any folder or drive mounted to your computer. All databases are backed up, including open databases. The server in the illustration below has been configured to automatically backup at 1:30 am every day, to a folder named **EnterpriseBackup** on a hard drive named **Alaska**.



We highly recommend that you do a further backup of this data. Normally backup programs like Retrospect cannot backup files that are open, but once Panorama has backed up the files your regular backup program can back up the copies with no problem. This allows you to do a full backup without ever shutting down the server.

Advanced Options

The options in this section of the wizard control various advanced settings on the server.



Automatically Unlock Records

When a client locks a record in a shared database (see “[Editing Data and Record Locking](#)” on page 136) that record normally stays locked until the user is finished with it (or until the client auto-unlock time is exceeded, see “[Changing the Record Lock Timeout](#)” on page 140). Of course this is usually exactly what you want to happen. In certain situations, however, this can result in a record being “stuck” in the record lock mode. This can happen if the client disconnects from the server before unlocking the record (either due to a network connection failure, for example a laptop disconnecting from the network or an internet failure, or from the client actually crashing).

If records being “stuck” in locked mode is a problem on your server you can configure the server to automatically unlock records after a specified period of time. If another user on the network tries to edit the record after this period has expired the server will unlock the record from the user that originally locked it and give the lock to the new user. Don’t make the time period too short, because if you do then users that simply pause for a moment may find that the record they thought they had locked is now locked by someone else (in which case any changes they were trying to make to the record will be lost).

Note: The server will not automatically unlock a record unless someone else on the network asks for it. As long as no one else requests the record it will remain locked indefinitely. For example, suppose the automatic unlock timeout is set to 5 minutes. Now suppose that Bob locks record 1234 at 1:00 PM. At 1:03 PM Marla tries to edit 1234. The server will refuse because the record is locked by Bob. But if Marla tries again after 1:05 PM she’ll be able to edit the record, and Bob will be locked out.

Note: This automatic unlock server option is completely independent from the client auto-unlock feature (see “[Changing the Record Lock Timeout](#)” on page 140), and you can use both of these features simultaneously. If you do use both, you would usually want to set the server timeout longer than the client timeout.

Note: You can also manually unlock records at any time with the **Locked Records** wizard (see “[Locked Records Wizard](#)” on page 142).

Rebound After Server Crash

Hopefully your server will never suffer a power failure or crash. If it does, however, this option will allow the server to restart and get right back to work right where it left off. We call this “rebounding.” When the server rebounds it automatically resumes operations from where it left off before the crash — any client sharing sessions are restarted, databases that were open before are re-opened, and any locked records are re-locked (rebounding has no effect on web publishing, only database sharing). Once the server restarts any clients can continue to work with shared databases as if nothing has happened. (If the rebound option is not turned on then all clients must log off and then log on again after a server crash.)

Rebounding only occurs if the server restarts soon after the power failure or crash. You can specify the maximum interval allowed before the rebound feature is disabled. After this time the server will do a clean restart, without resuming sessions, open databases or record locks. (You can also force a clean restart by erasing the **Snapshot.dat** file in the **Enterprise** folder before re-starting the server.)

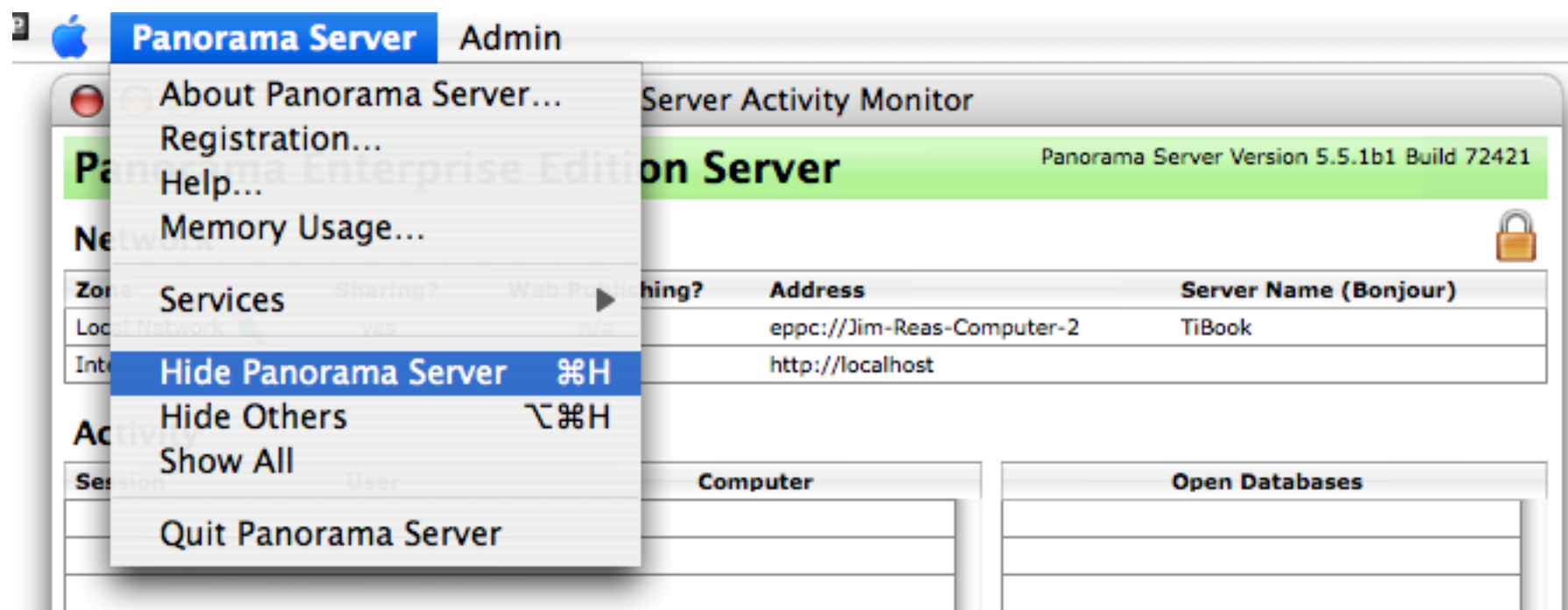
The rebound feature only comes into play if the server crashes. The server will always do a clean re-start after you intentionally **Quit** the server application.

Ask to Confirm Before Quitting Server

When you intentionally **Quit** the server application the server normally asks you to confirm before it actually shuts the server down (see “[Shutting Down the Server](#)” on page 45). Uncheck this option if you want the server to skip this confirmation and quit immediately when asked.

Automatically Hide Server When Launched

When this option is checked the server will automatically hide itself as it launches. This is the same as launching the application and then using the **Hide Panorama Server** command in the **Panorama Server** menu, except that it all happens automatically.

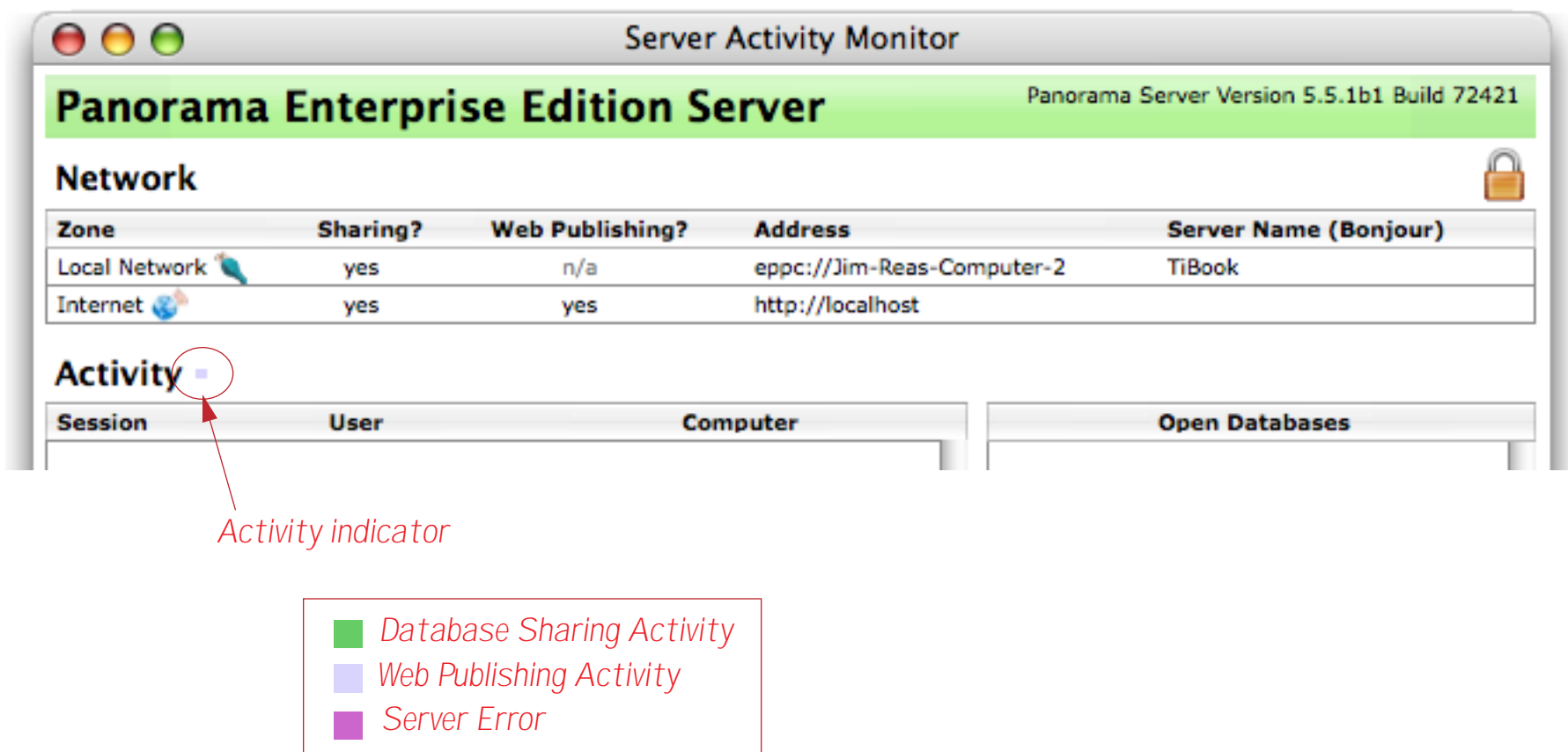


We highly recommend this option if anyone is actually using this computer for something other than serving. Hiding the server prevents someone sitting at the server from accidentally clicking on one of the server windows and possibly disrupting server operation.

If you need to un-hide the server for any reason you can either click on its dock icon or choose **Show All** from the Application menu of any open program (the Application menu is always the first menu after the Apple menu).

Enable Server Activity Indicator

The Panorama Server normally doesn't give any indication of activity while it is running. If you check the **Enable Server Activity Indicator** option the **Server Activity Monitor** window will display a blinking indicator as it is accessed by sharing and web clients. This indicator appears just to the right of the word **Activity** as shown in the illustration below.



As shown above, the blinking indicator is normally green-blue for sharing activity, light purple for web activity, and dark red/maroon for errors.

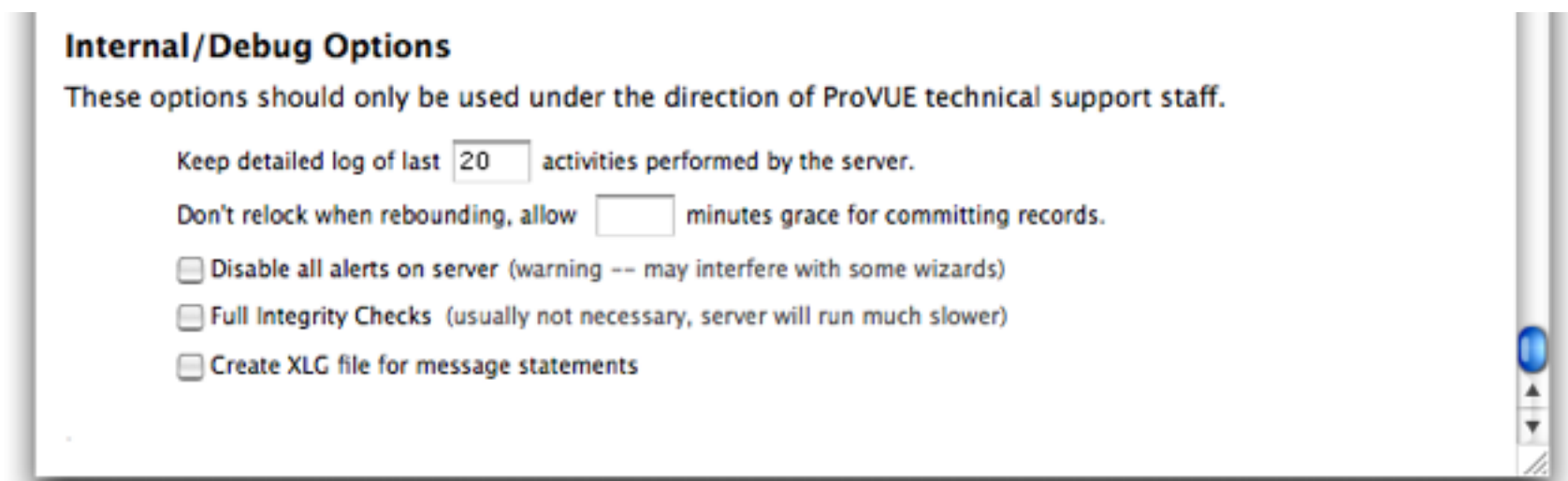
Advanced Note: If you don't like the standard indicator colors you can change them by editing the **Server-Config.dat** file (see "[Editing the Server Configuration Text File \(For Experts Only\)](#)" on page 96). To change the colors, add an **<ACTIVITYLIGHTCOLORS>** tag to this file, like this:

```
<HIDESERVER></HIDESERVER>
<ACTIVITYLIGHT>yes</ACTIVITYLIGHT>
<ACTIVITYLIGHTCOLORS>share="%%66CC66" web="%%D8D4FE" error="%%CC66CC"</ACTIVITYLIGHTCOLORS>
<NEVERNEVERSERVERALERTS></NEVERNEVERSERVERALERTS>
<SERVERLOCKTIMEOUT>5</SERVERLOCKTIMEOUT>
```

The tag may contain up to three parameters, **share=**, **web=** and **error=**. Each parameter must specify a color in HTML RGB format, and must be preceded with **%%**. For example **error="%%FF0000"** will cause errors to be displayed with a pure red indicator. If a parameter is missing the default color will be used.

Internal/Debug Options

The bottom portion of the Server Administration wizard contains internal and debug options.

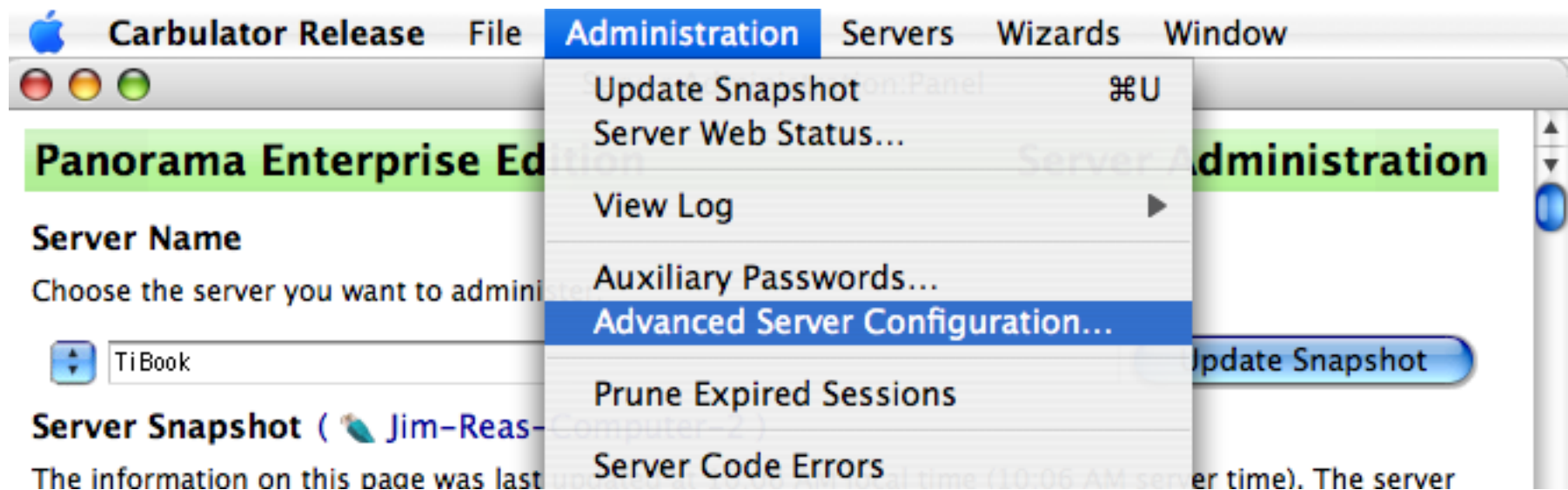


These options are intended for assisting ProVUE technical support staff in tracking down unusual problems, especially during the beta test phase of server development. You should not modify any of these options unless directed by a ProVUE staff member.

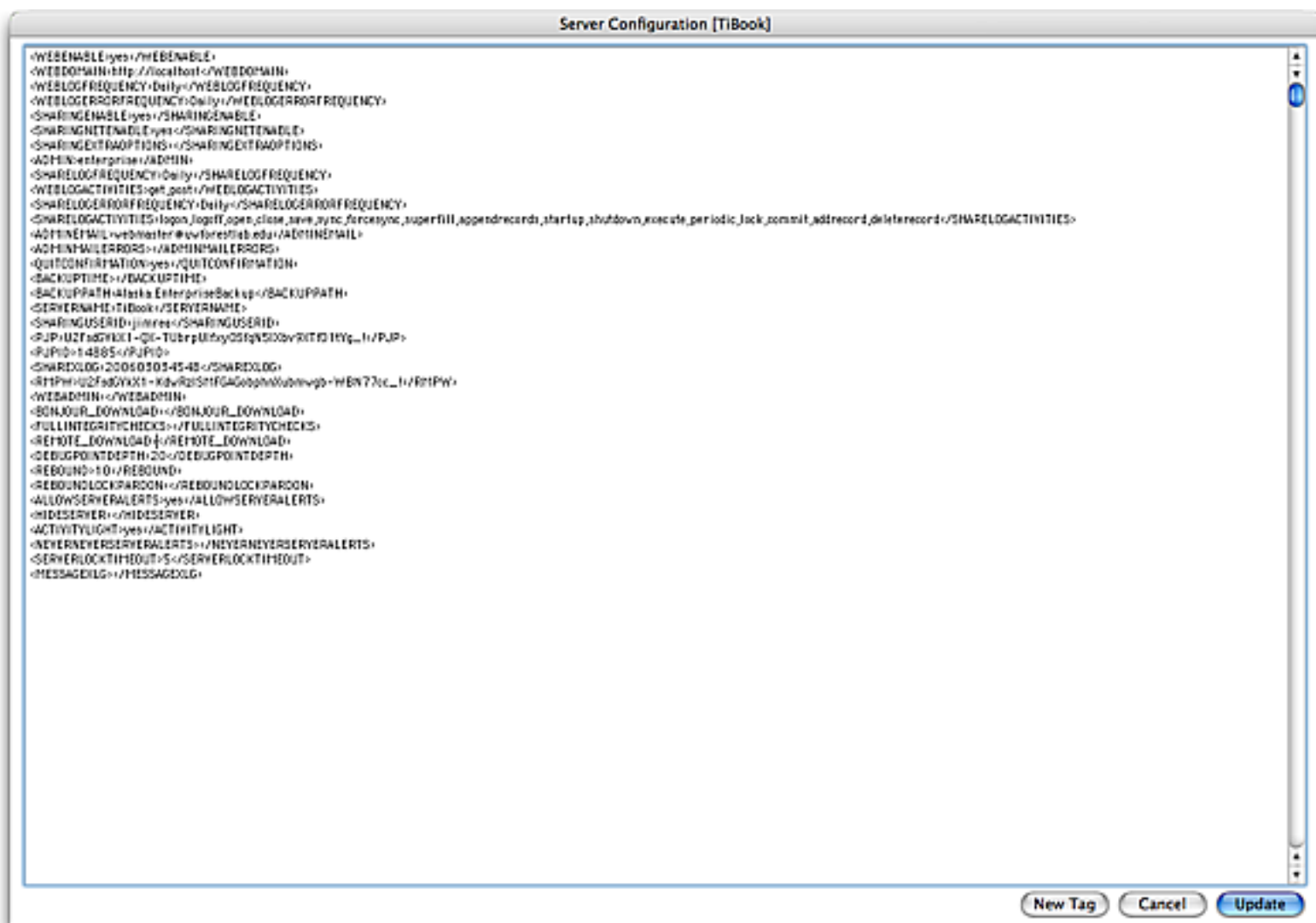
Editing the Server Configuration Text File (For Experts Only)

The server computer contains a special text file that controls the configuration of the server (called **Server-Config.dat**, see the next section). When you change options with the **Server Administration** wizard you are actually editing this file. The file can also be edited directly, either remotely from a client or directly on the server computer. Usually this is never necessary, as the normal Server Administration options should take care of every option you'll need to control. If you do decide to edit the configuration file directly, be very careful since a mistake may cause serious problems or even require that you re-install the server.

To edit the configuration file remotely from a client, open the **Server Administration** wizard, select the server, then choose **Advanced Server Configuration** from the **Administration** menu.



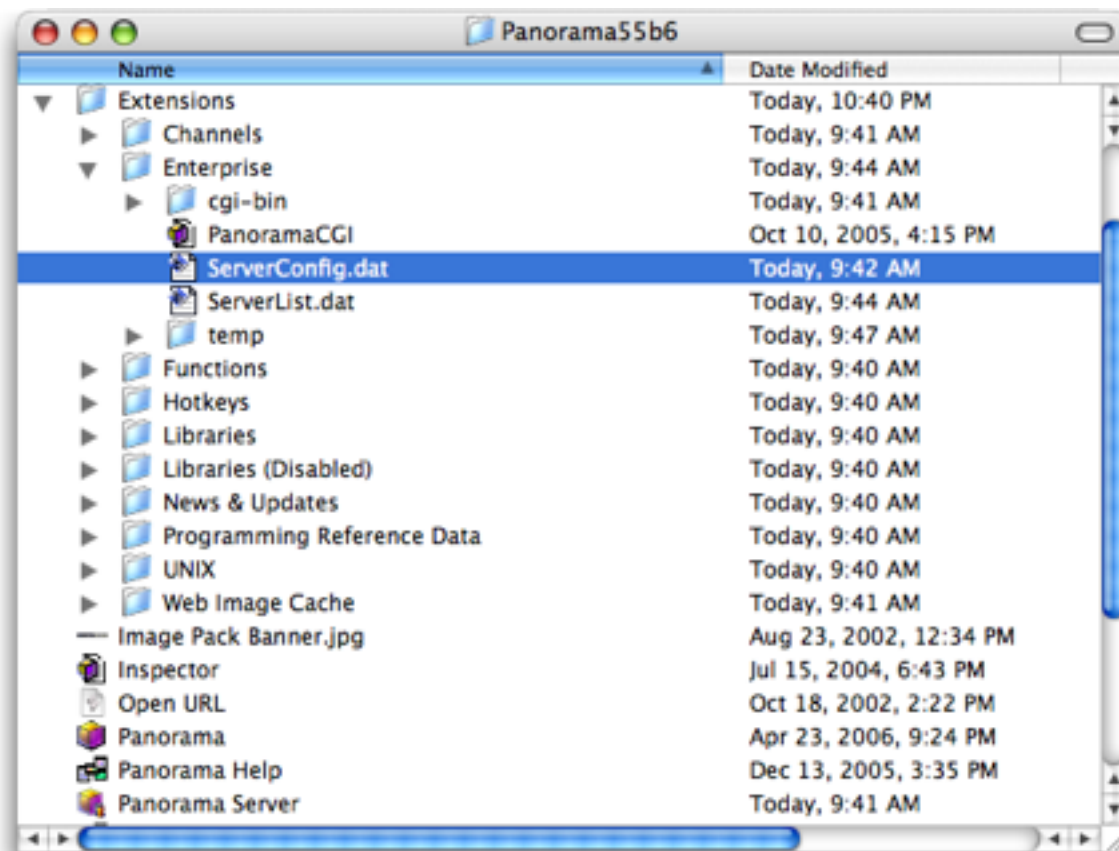
A dialog window appears listing the contents of the configuration file.



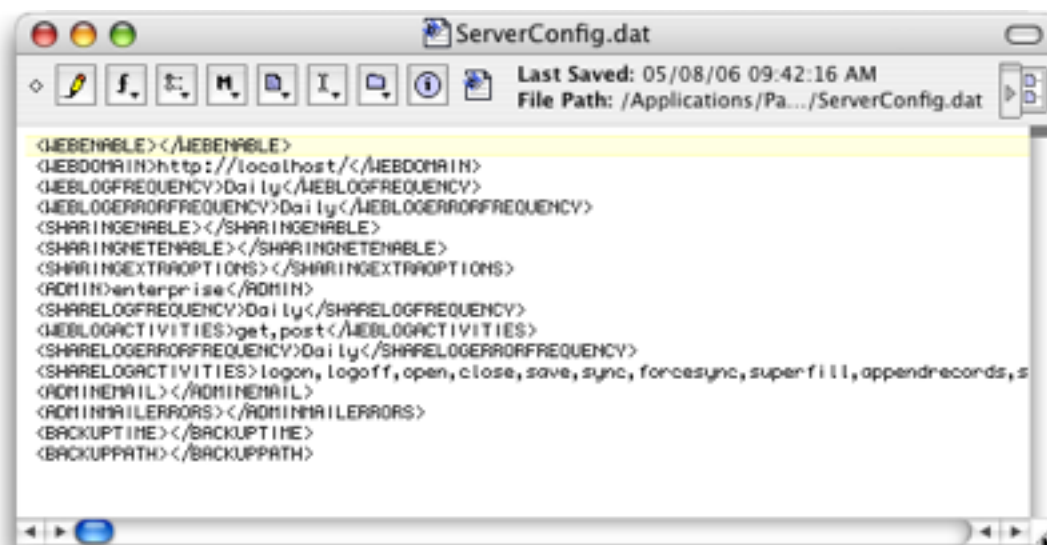
Carefully edit the configuration (see “[Server Configuration Tags](#)” on page 98), then press the **Update** button to update the file on the server.

Editing the ServerConfig.dat File Directly on the Server

The Panorama Server configuration file is called **ServerConfig.dat** file and is located in the **Enterprise** folder inside the **Extensions** folder inside the **Panorama** folder.



You can edit this file with BBEdit, TextWrangler, TextMate, Text Edit or your favorite text editor. When you open the file it should look something like this:



If the server is running when you edit this file, you'll need to shut down and re-launch the server software for any changes you make to take effect (note that this is not generally necessary when editing the file with the **Server Administration** wizard as described in the previous section).

Server Configuration Tags

The server configuration file contains a series of tags that control the configuration of the server. Most of these tags are described below (some tags are designed for internal server use, and should never be edited).

```
<ACTIVITYLIGHT>yes</ACTIVITYLIGHT>
```

This option enables a blinking "activity light" on the server. The blinking light appears just to the right of the word Activity above the list of users. Note that turning on this feature may slow down the server slightly, so the default is off if this option is missing. The blinking light is green-blue for sharing activity, light purple for web activity, and dark red/maroon for errors. You can also customize the colors with the ACTIVITYLIGHT-COLORS tag, see below.

```
<ACTIVITYLIGHTCOLORS>share="%%66CC66" web="%%D8D4FE" error="%%CC66CC"</ACTIVITYLIGHTCOLORS>
```

This option allows you to customize the activity light colors for different activities on the server. Each color is specified using HTML RGB format preceded by %, as shown above. If an activity is not included in the option the server will use the default color for that activity.

```
<ADMIN>enterprise</ADMIN>
```

This is the server administrator password. This option is normally changed using the Server Activity Monitor, you cannot edit it manually.

```
<ADMINEMAIL>someone@someisp</ADMINEMAIL>
```

This option specifies the e-mail address of the server administrator. Use this if you want the administrator to be able to receive e-mail messages from the server. This option can also be set with the Server Administration wizard.

```
<ADMINMAILERRORS>yes</ADMINMAILERRORS>
```

If this option is set to "yes" (or actually any non-blank value) and the ADMINEMAIL option (see above) contains a valid e-mail address (or at least an @ symbol) then any errors that occur on the server will be e-mailed to the system administrator. This option can also be set with the Server Administration wizard.

```
<ALLOWSERVERALERTS>yes</ALLOWSERVERALERTS>
```

This is a debugging feature. Panorama normally suppresses any alerts that may occur on the server. Instead of displaying the alerts, they are saved in the server log files. You can disable this feature by setting this option to "yes". Normally this option is only used by ProVUE programmers during software development. (Note: Alerts are not suppressed when you are using the CGI Simulator wizard.)

```
<BACKUPPATH>path</BACKUPPATH>
```

This option specifies the folder where backups (if any) should be made. This must be a full path beginning with the name of the disk. For example, if the backup should be made to the PanBack folder on the disk HD, the path should be HD:PanBack. This option can also be set with the Server Administration wizard.

```
<BACKUPTIME>time</BACKUPTIME>
```

This option specifies the time when backups (if any) should be made. The time may be specified in either am/pm or 24 hour format, for example 4am, 10:45 PM, or 22:15. This option can also be set with the Server Administration wizard.

```
<BONJOUR_DOWNLOAD>password</BONJOUR_DOWNLOAD>
```

This password restricts access to downloading databases from clients on the local network. This password is used by the Download Shared Databases wizard. If left blank anyone on the local network can download shared databases from this server using this wizard. This password can also be set from the "Auxiliary Passwords" dialog in the "Administration" menu of the Server Administration wizard, where it is called the "Download Databases from Local Clients" password.

```
<DEBUGPOINTDEPTH>nn</DEBUGPOINTDEPTH>
```

This is a debugging feature. When this feature is enabled the server will keep detailed log files of the last nn activities performed by the server before a crash. The files are kept in the Enterprise:Logs:dbg folder. The nn value is the max number of previous actions to keep in the log, for example the last 50 or last 100. Note that using this option will slow the server down somewhat, though probably not too much (the larger nn is, the more it will slow down). The contents of this log can be analyzed by ProVUE Development's technical staff, you would not usually enable this feature unless requested by ProVUE technical staff.

```
<FULLINTEGRITYCHECKS>yes</FULLINTEGRITYCHECKS>
```

This is a debugging feature. If this option is "yes" the server will check the integrity of the current database after every server communication. If there is a problem it will log this in a special Data Integrity Log in the Logs folder. Using this option may significantly slow down your server (or may not). The contents of this log can be analyzed by ProVUE Development's technical staff, you would not usually enable this feature unless requested by ProVUE technical staff.

```
<HIDESERVER>yes</HIDESERVER>
```

When this option is set to "yes" the server will automatically hide itself when it launches. This is the same as using the "Hide Panorama Server" command in the "Panorama Server" menu, except that it happens automatically. This is especially useful if you are running both client and server on the same machine, or running any other software on the same machine as the server. You can always make it visible again by clicking on the Panorama Server dock icon.

```
<MAXJOURNALSIZE>nnnn</MAXJOURNALSIZE>
```

The server normally limits the size of journal files to 32,000 bytes. If a journal file gets bigger than this the server saves the database and erases the journal file. The max size is actually adjustable using this option, where nnnn is a number specifying the maximum journal size in bytes. If nnnn is 0 then journaling is completely disabled and Panorama will save the entire database on every change. However if nnnn = "" (empty) then the default 32,000 threshold will be used.

```
<MESSAGEXLG>yes</MESSAGEXLG>
```

Normally when an alert occurs on the server the text of the alert is logged and the server creates a special .xlg log file with complete information about what the procedure was doing at the time of the alert (in addition to logging the alert, see <ALLOWSERVERALERTS> above.) Normally an .xlg file is NOT created for MESSAGE statements, but they will be if this option is set to "yes". The contents of .xlg log files can be analyzed by ProVUE Development's technical staff, they do not contain information that can be analyzed by users or system administrators.

```
<PJP>internal_data</PJP>
```

```
<PJPID>internal_data</PJPID>
```

These options contain internal data used by the server. You cannot (and must not) edit these options.

```
<QUITCONFIRMATION>no</QUITCONFIRMATION>
```

Use this option to disable quit confirmation for the server. If this option is "no" or "off" the server will not ask you to confirm when you Quit the server. Use this option if you want to use an automatic tool for remote reboots, but be careful not to quit the server accidentally!!

```
<RMPW>internal_data</RMPW>
```

This option contains internal data used by the server. You cannot (and must not) edit this option.

```
<REBOUND>nn</REBOUND>
```

This option enables the server's "rebound" option. When this option is enabled the server will "rebound" after a crash, i.e. re-open any sessions and databases that were open before the crash, allowing clients to continue working without closing and re-opening databases. (Of course they will still get errors if they try to use the server between the time of the crash and the time the server restarts.) NN is the number of minutes allowed between the last server activity and a restart. For example, suppose nn=20. If the server crashes and you reboot within 20 minutes then it will "rebound" and clients can continue to work. If you wait more than 20 minutes then the server will start up normally without rebounding. You can also start up without rebounding by erasing the Snapshot.dat file in the Enterprise folder before restarting the server.

```
<REBOUNDLOCKPARDON>nn</REBOUNDLOCKPARDON>
```

When rebounding (see above) the server normally restores all state, including locked records. If clients have shut down (or even clicked on other records or windows) in between crash and restart then you can be left with locked records that must be cleared either manually (with the serveradmin wizard or the forceunlock statement) or by shutting down the server. To avoid this you can use the <REBOUNDLOCKPARDON>nn</REBOUNDLOCKPARDON> option. If this option is used then rebound does not relock records when rebounding. However, clients may still think they have locked records and may try to update the server with revised records. If they do so in the first few minutes the server will allow this. The exact number of minutes is determined by nn. After the "lock pardon" period is over users will no longer be able to unlock records they locked before the crash (which means that any changes they have made will be lost). The advantage of this approach is that it will avoid extra locked records after a crash, but the disadvantage is that it breaks Panorama's rigid record locking rules and could potentially allow two users to edit the same record simultaneously. (IMPORTANT NOTE: This tag is now obsolete.)

```
<REMOTE_DOWNLOAD>password</REMOTE_DOWNLOAD>
```

This password restricts access to downloading databases from clients on the internet. This password is used by the Download Shared Databases wizard. If left blank anyone on the internet can download shared databases from this server using this wizard. This password can also be set from the "Auxiliary Passwords" dialog in the "Administration" menu of the Server Administration wizard, where it is called the "Download Databases from Remote Clients" password.

```
<SERVERNAME>name</SERVERNAME>
```

This is the bonjour name of this server. This option is normally changed using the Server Activity Monitor, you cannot edit it manually.

```
<SHARELOGACTIVITIES>actions</SHARELOGACTIVITIES>
```

This option controls what types of actions will be logged in the sharing log. The allowable actions are:

```
logon, logoff, open, close, save, sync, forcesync, superfill, appendrecords, startup, shutdown
execute, periodic
```

If multiple types should be logged they should be separated by a comma with no spaces, for example logon,logoff. You can also simply specify the action "all" to log all server activity, but this will make for gigantic logs and may impact server performance. This option can also be set with the Server Administration wizard, though some types of actions are not available thru that wizard.

```
<SHARELOGFREQUENCY>period</SHARELOGFREQUENCY>
```

This option controls how often the server starts a new Sharing Log (log of database sharing activity). The period may be Daily, Weekly, Monthly or "" if you don't want to keep any log at all. This option can also be set with the Server Administration wizard.

```
<SHARELOGERRORFREQUENCY>period</SHARELOGERRORFREQUENCY>
```

This option controls how often the server starts a new Sharing Error Log (log of database sharing errors). The period may be Daily, Weekly, Monthly or "" if you don't want to keep any log at all. This option can also be set with the Server Administration wizard.

```
<SHARINGENABLE>yes</SHARINGENABLE>
```

This option is normally changed using the Server Activity Monitor, you cannot edit it manually. If it is set to "yes" then database sharing using AppleEvents on the local network is enabled.

```
<SHARINGNETENABLE>yes</SHARINGNETENABLE>
```

This option is normally changed using the Server Activity Monitor, you cannot edit it manually. If it is set to "yes" then database sharing using TCP/IP is enabled.

```
<SHARINGEXTRAOPTIONS></SHARINGEXTRAOPTIONS>
```

This option is currently not used.

```
<SHARINGUSERID>id</SHARINGUSERID>
```

This is the unix name of the user the server is running under. You cannot (and must not) edit this option.

```
<WEBADMIN>password</WEBADMIN>
```

This password restricts access to server and database status pages from web browsers. If left blank then anyone on the internet will be able to view a list of web published databases on this server, as well as lists of forms and procedures in each database. This password can also be set from the "Auxiliary Passwords" dialog in the "Administration" menu of the Server Administration wizard, where it is called the "Server & Database Web Status Pages" password.

```
<WEBDOMAIN>http://yourdomain</WEBDOMAIN>
```

This option is normally changed using the Server Activity Monitor, but can be edited manually. It should be the ip address or domain name of this server. Note: This option is only necessary for web publishing, if you are only using database sharing you can leave this option blank.

```
<WEBENABLE>yes</WEBENABLE>
```

This option is normally changed using the Server Activity Monitor, you cannot edit it manually. If it is set to "yes" then web database publishing is enabled.

```
<WEBLOGACTIVITIES>actions</WEBLOGACTIVITIES>
```

This option controls what types of actions will be logged in the CGI log. The allowable actions are get and post. If multiple types should be logged they should be separated by a comma with no spaces, for example get,post. This option can also be set with the Server Administration wizard.

```
<WEBLOGFREQUENCY>period</WEBLOGFREQUENCY>
```

This option controls how often the server starts a new CGI Log (log of web publishing activity). The period may be Daily, Weekly, Monthly or "" if you don't want to keep any log at all. This option can also be set with the Server Administration wizard.

```
<WEBLOGERRORFREQUENCY>period</WEBLOGERRORFREQUENCY>
```

This option controls how often the server starts a new CGI ERROR Log (log of web publishing errors). The period may be Daily, Weekly, Monthly or "" if you don't want to keep any log at all. This option can also be set with the Server Administration wizard.

Chapter 3: Online Database Sharing



This chapter explains how to create and use shared Panorama databases. It assumes that you already have a working Panorama Server available (see Chapter 2).

Creating a Shared Database

The first step in creating a shared database is to create a single user database. This database can be old or new, empty or full. For this example, we'll use a database called [Real Estate Listings](#).



Real Estate Listings

If it's not already open, double click on the database to open it.

| Listed | Address | City | Asking | Agent | Offer | Closed |
|------------|--------------------------|------------------|-------------|--------------|-------------|------------|
| 02/12/2006 | 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | | |
| 03/26/2006 | 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 | 07/02/2006 |
| 03/15/2006 | 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 | 06/20/2006 |
| 01/08/2006 | 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | | |
| 02/14/2006 | 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | | |
| 03/25/2006 | 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | | |
| 01/23/2006 | 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 | 05/09/2006 |
| 01/30/2006 | 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | | |
| 02/26/2006 | 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | | |
| 01/27/2006 | 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 | 05/23/2006 |
| 02/19/2006 | 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | | |

So far this is a garden variety single user database. You can sort, select, add data, delete data — it all happens on your computer and has nothing to do with anyone else. In other words, it's not shared, but we're about to change that.

To start the process of changing this database into a shared database, open the **Database Sharing Options** wizard (in the **Sharing** submenu of the **Wizard** menu).

Database Sharing Options:Real Estate Listings


Panorama Enterprise Edition **Sharing Options**


Use this page to configure how a single user database will be shared, or to change the configuration of a database that is already being shared. When finished use 'Apply Changes' (in the Options menu or using the button at the bottom of the page) to actually change the configuration.


Database
Choose the database you want to configure: Real Estate Listings

Server
Choose the server on which this database will be hosted:

Sharing Mode
Choose how this database will be shared:

Local Database Sharing 
This option allows the database to be shared (multi-user) on the local network.

Internet Database Sharing 
This option allows the database to be shared (multi-user) on the internet.

Web Publishing 
This option allows the database content to be published on the web (for use by web browsers). The database must have one or more special procedures added to enable web publishing.

Basic Database Sharing Options
This option applies only if one of the database sharing options are checked above:


Start at the top of this form and work your way down. The first choice is the database. If it doesn't already list the database you want to convert then select it from the pop-up menu.

The next choice is the server. Use the pop-up menu to select the server that will host this shared database. (If no server appears in the pop-up menu then see "[Debugging Local Database Sharing Connection Problems](#)" on page 61.)

Database

Choose the database you want to configure: Real Estate Listings

Server

Choose the server on which this database will be hosted: TiBook 

The next choice is the sharing mode. Since for now we only want to share this database on the local server, we just check the **Local Database Sharing** option.

Sharing Mode

Choose how this database will be shared:

- Local Database Sharing 
This option allows the database to be shared (multi-user) on the local network.
- Internet Database Sharing 
This option allows the database to be shared (multi-user) on the internet.
- Web Publishing 
This option allows the database content to be published on the web (for use by web browsers). The database must have one or more special procedures added to enable web publishing.

The next choice is whether to enable an auto-timeout that will unlock records automatically if someone walks away from their computer in the middle of making a change. This is usually a good idea, so we'll set the timeout to 45 seconds.

Basic Database Sharing Options

This option applies only if one of the database sharing options are checked above:

- Auto unlock after seconds of inactivity
Panorama automatically locks the current record when you begin editing it, and unlocks when you move to another record. If this option is checked, Panorama will also unlock the current record after the specified amount of time has passed with no activity. In other words, if someone starts editing a record and then walks away, Panorama will automatically unlock the record after the specified time.

The next four sections of this form (*Offline Database Sharing Options*, *Connection/Synchronization Sharing Options*, *Database Web Publications* and *Server Database Name*) contain advanced options that can usually be left as is, so our database is ready to be shared! Just scroll to the bottom and press the **Apply Changes** button

Choose the name of this database on the server:

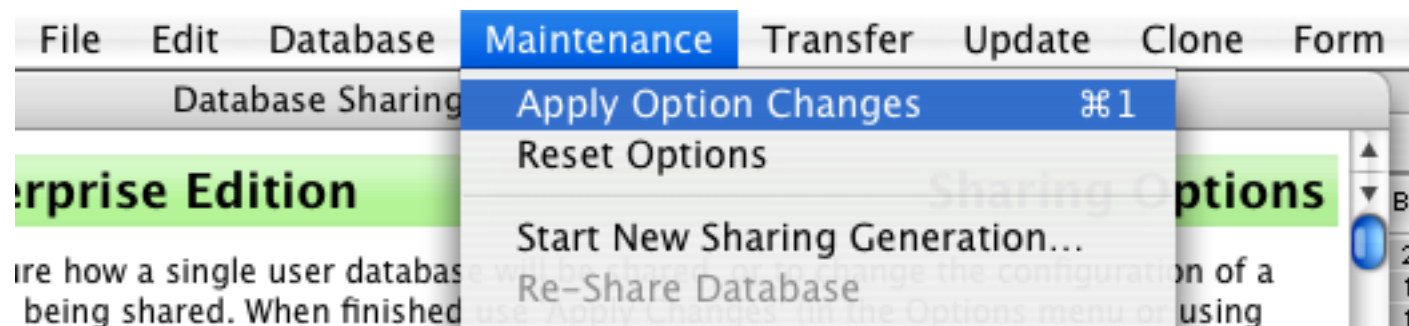
Real Estate Listings

The name of the database on the server is usually the same as the name on the client. However, you can use a different name if the client name is already in use on the server.

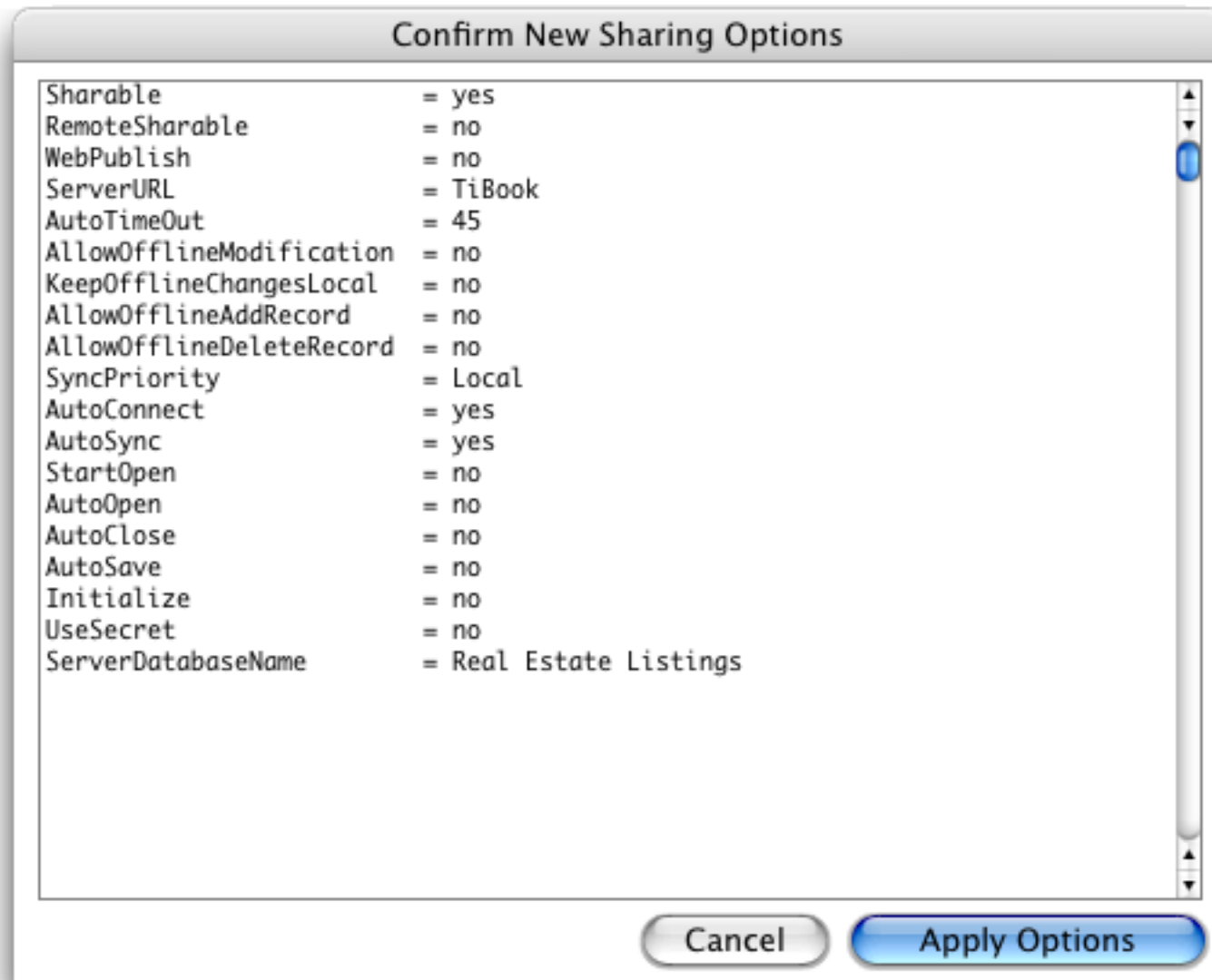
Reset

Apply Changes

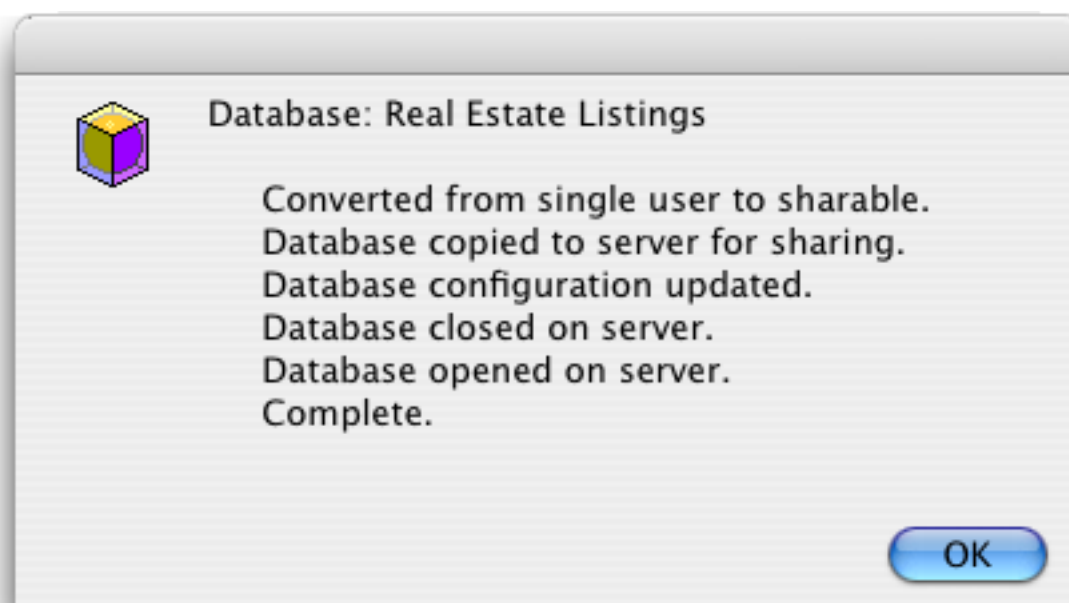
or choose **Apply Option Changes** from the **Maintenance** menu (or press **Command-1**).



The wizard will display a summary of the proposed new sharing options for this database.



Double check to make sure the options are correct, then press the **Apply Options** button. The database will be uploaded to the server as a shared database. You'll be notified when this process is complete.



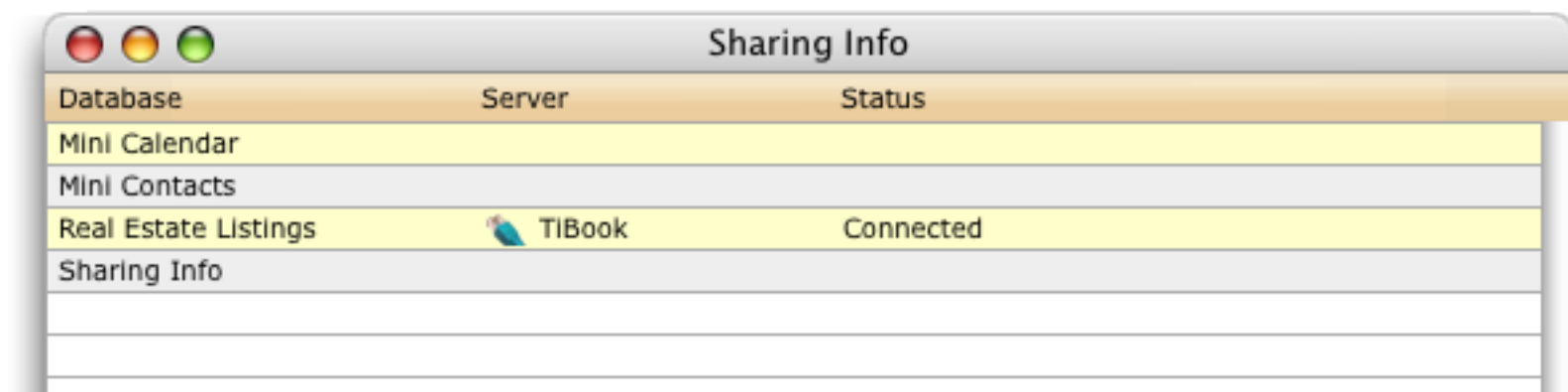
At this point you can close the Database Sharing Options wizard.

Our database is still open, and it looks the same as it did before we started. But it is now a fully shared multi-user database.



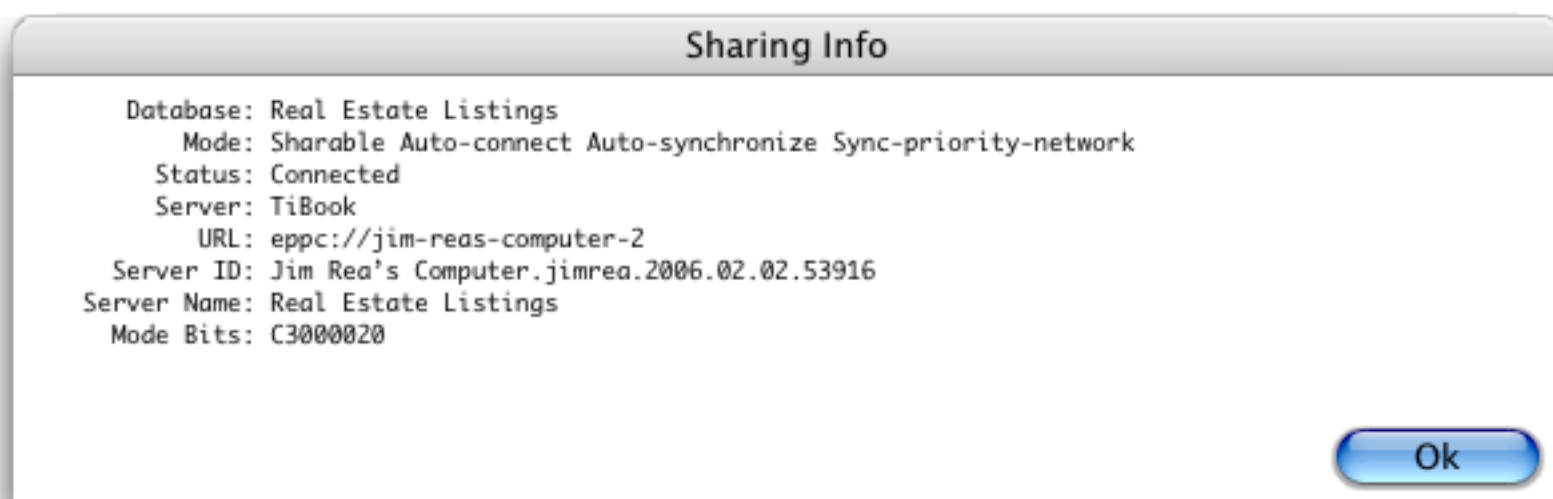
| Listed | Address | City | Asking | Agent | Offer | Closed |
|------------|---------------------|------------------|-------------|--------------|-------------|------------|
| 02/12/2006 | 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | | |
| 03/26/2006 | 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 | 07/02/2006 |
| 03/15/2006 | 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 | 06/20/2006 |
| 01/08/2006 | 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | | |
| 02/14/2006 | 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | | |
| 03/25/2006 | 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | | |
| 01/23/2006 | 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 | 05/09/2006 |
| 01/30/2006 | 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | | |
| 02/26/2006 | 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | | |
| 01/27/2006 | 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 | 05/23/2006 |

If you want to check to see if a database is sharable, you can use the **Sharing Info** wizard (in the **Sharing** sub-menu of the **Wizard** menu). This wizard shows that there are currently four databases open, three of which are single user databases. The **Real Estate Listings** database is shared on the **TiBook** server, and is currently open on that server.



| Database | Server | Status |
|----------------------|--------|-----------|
| Mini Calendar | | |
| Mini Contacts | | |
| Real Estate Listings | TiBook | Connected |
| Sharing Info | | |

You can click on the **Real Estate Listings** line to see more detail about the sharing status of this database. You shouldn't need this information in ordinary use, but if there is a problem, this additional information can sometimes help track it down.

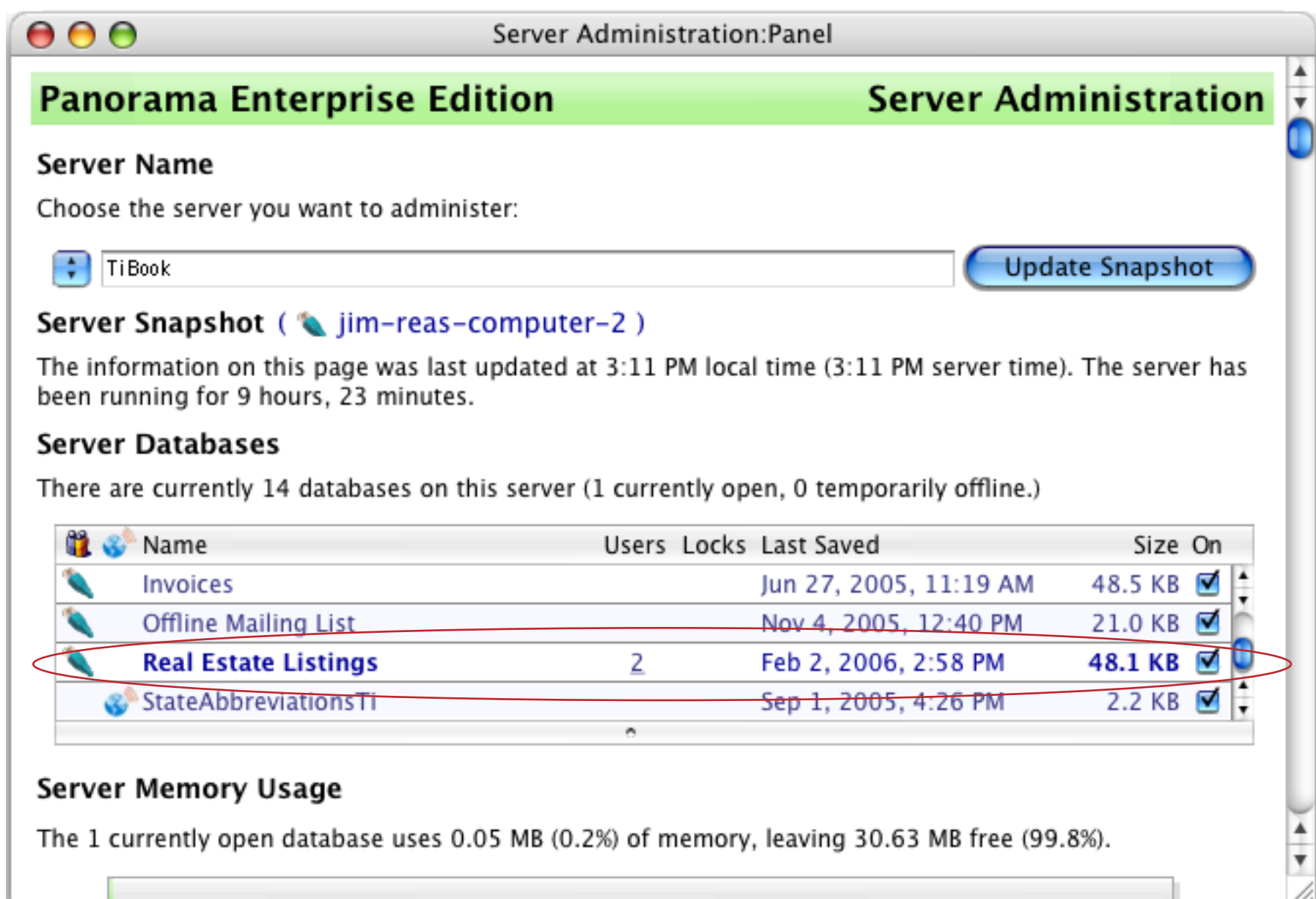


```

Database: Real Estate Listings
Mode: Sharable Auto-connect Auto-synchronize Sync-priority-network
Status: Connected
Server: TiBook
URL: eppc://jim-reas-computer-2
Server ID: Jim Rea's Computer.jimrea.2006.02.02.53916
Server Name: Real Estate Listings
Mode Bits: C3000020
  
```

Ok

If you check the list of server databases in the **Server Administration** wizard, you'll see the newly shared database there.



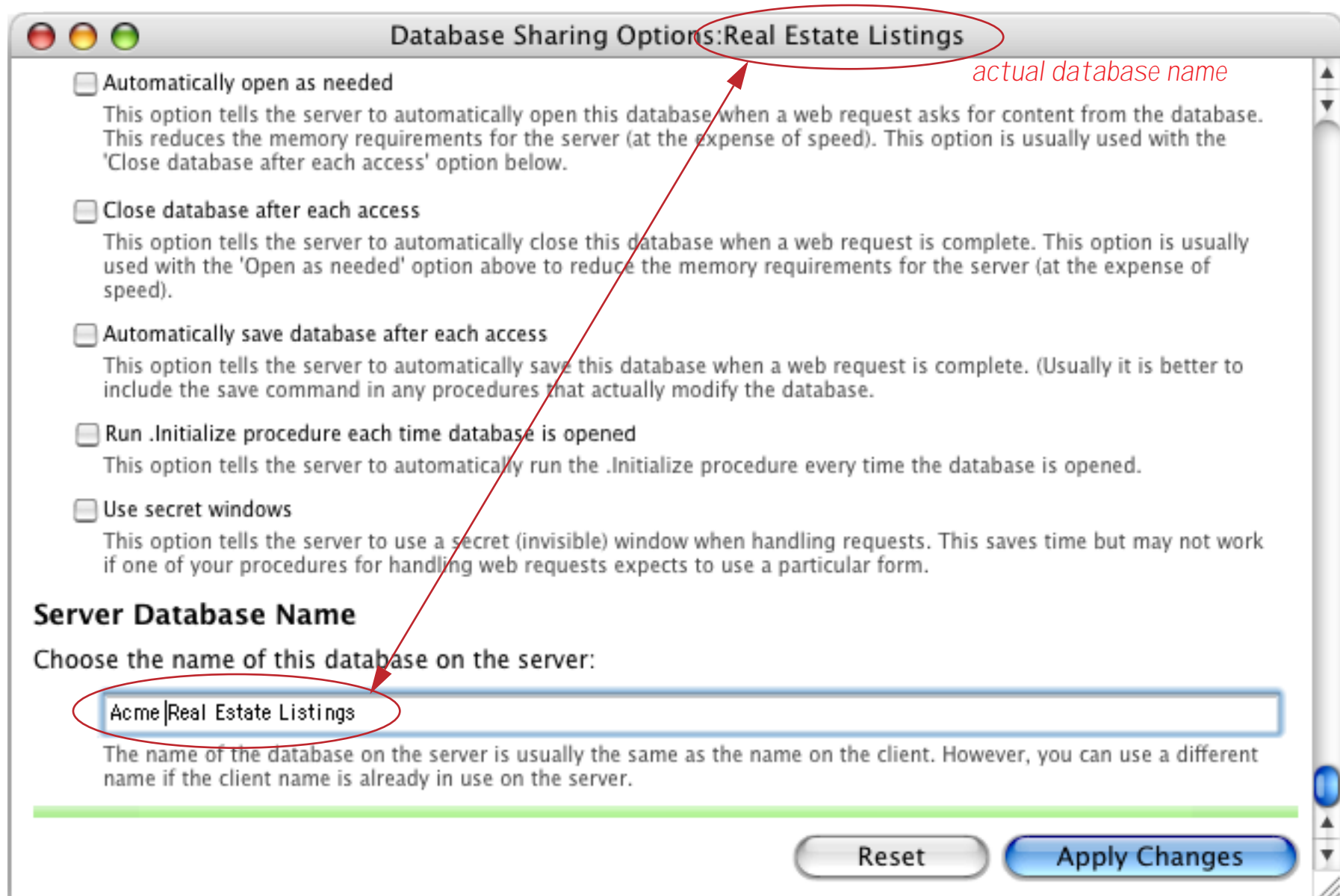
Duplicate Database Conflicts on the Server

Each database on the server must have its own unique name. In other words, you cannot have two databases on the server with the same name. For example, suppose that there already is a database named **Real Estate Listings** on the server and then you use the **Database Sharing Options** wizard to try to add another. The wizard will warn you that completing this operation will replace (erase) the copy of the database that is already on the server.



If this is a new version of the same database (see "**Sharing "Generations"**" on page 119) then you'll probably want to click **Yes**. Otherwise, you should click **No** (unless you are **absolutely, positively sure** that **no one** on the network will want to use the existing database ever again).

If you encounter this situation, one solution is to rename the original database. However, sometimes that is not possible. For example, the database may be part of a set with extensive lookups and programming between different databases in the set. In that case renaming a database could take hours or even days of work. To get around this problem, the Database Sharing Options wizard allows you to specify a **Server Database Name** that is separate from the actual database name. You'll find this at the very bottom of the wizard.



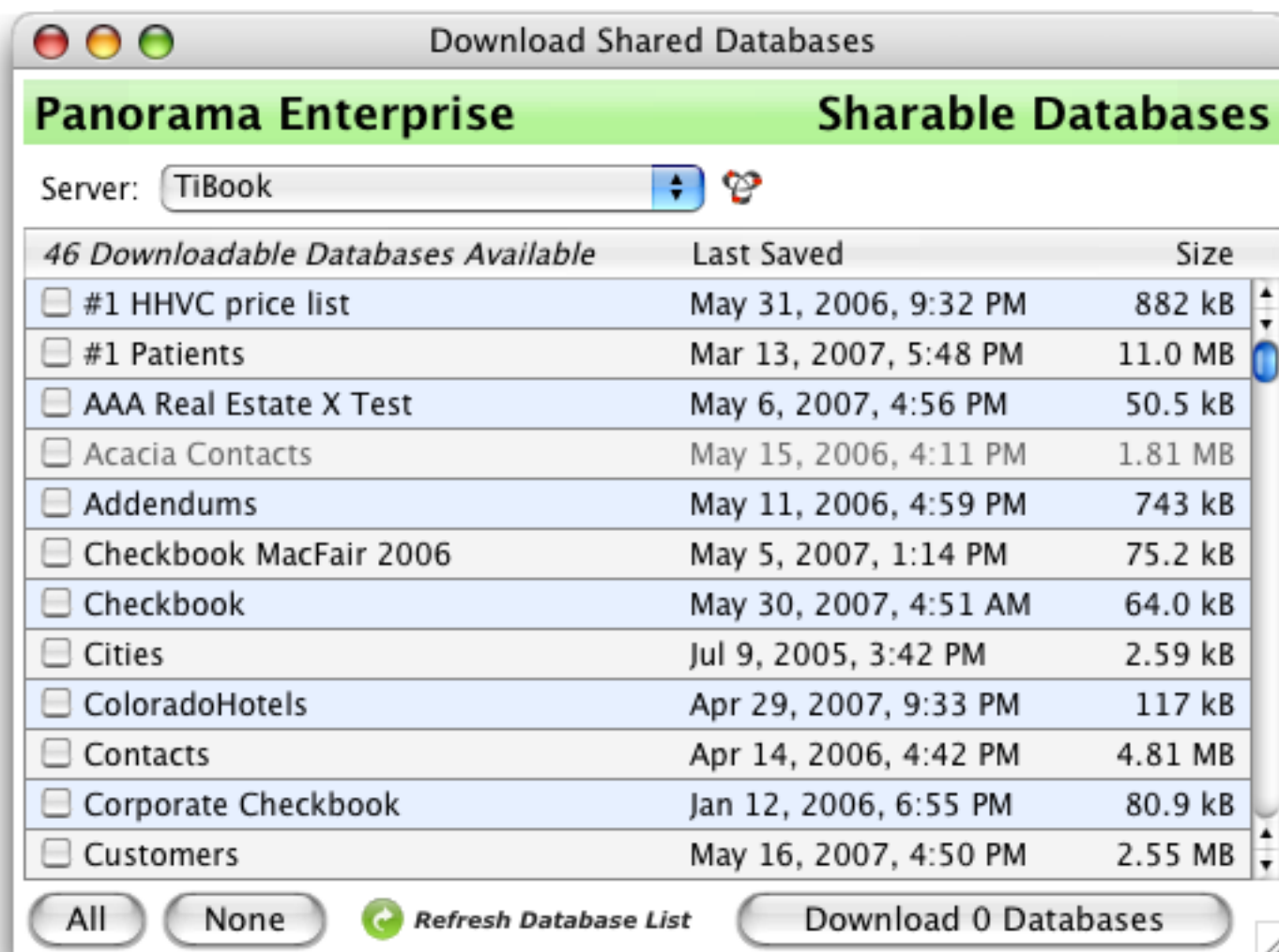
In this case we've changed the **Server Database Name** to **Acme Real Estate Listings**. This allows this database to co-exist on the server with the original **Real Estate Listings** database. The client database (on the local computer) is still called **Real Estate Listings**, so you won't need to reprogram any of the lookups or procedures.

Transferring the Database to Other Client Computers

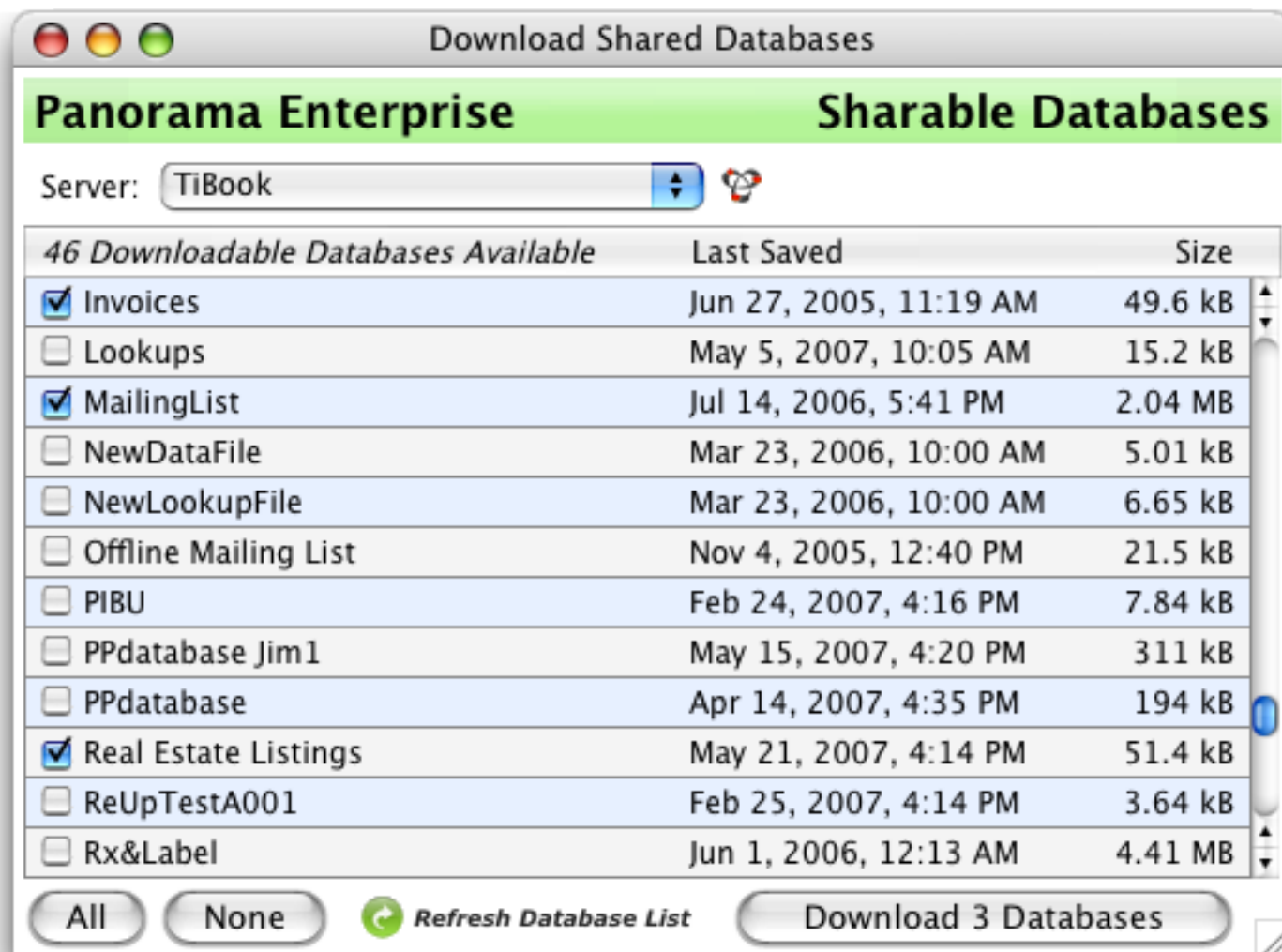
The original single user database has been transformed into a shared database. The final step is to transfer copies of this database to all of the other clients that need to access the data. There are several ways to transfer the database to the various clients: 1) using the **Download Shared Databases** wizard, 2) using the **Server Administration** wizard, or 3) manually (via file sharing, e-mail, USB drive, etc.).

Using the Download Shared Databases Wizard

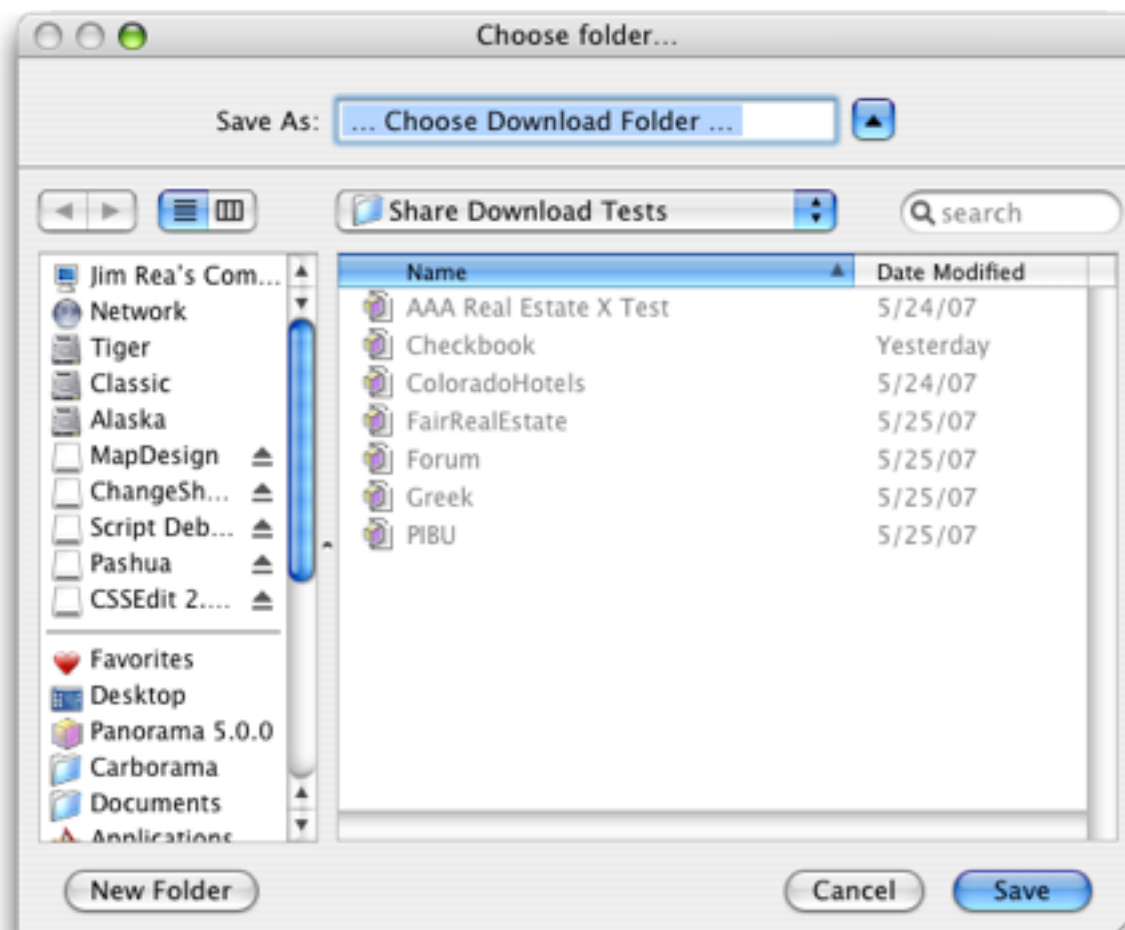
The most common way to download shared files from the server is to use the **Download Shared Databases** wizard. Start by opening this wizard on the client machine, then use the pop-up menu to choose the server that contains the shared database (if only one local server is available, it will be selected automatically when the wizard opens). You'll be asked for the appropriate download password, if any (see "[Changing the Auxiliary Passwords](#)" on page 80). After the password is entered, a list of shared databases on the server is displayed (databases that are web published but not sharable are not listed).



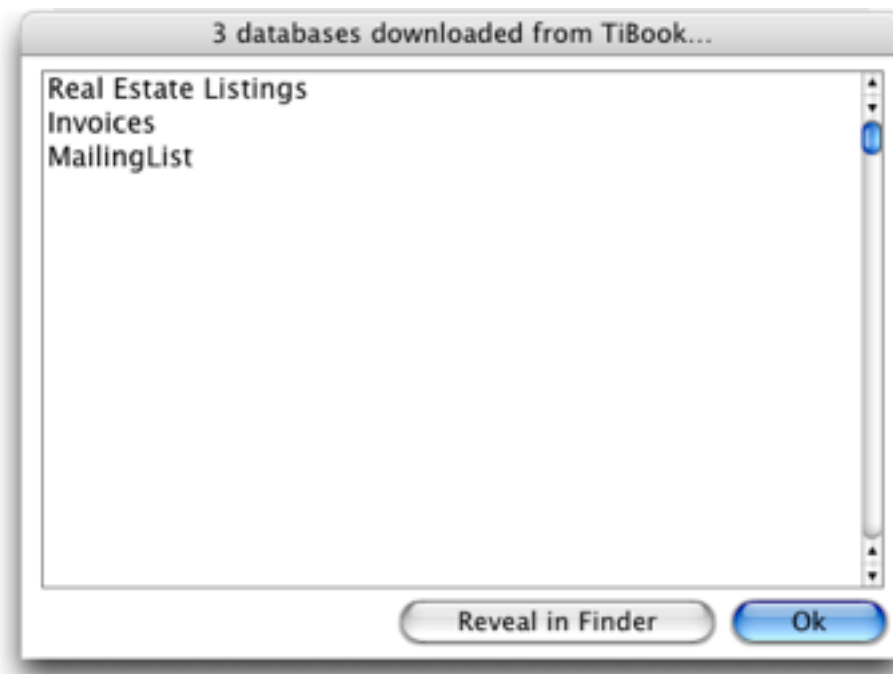
To download one or more databases, simply click on their check boxes and press the **Download Databases** button.



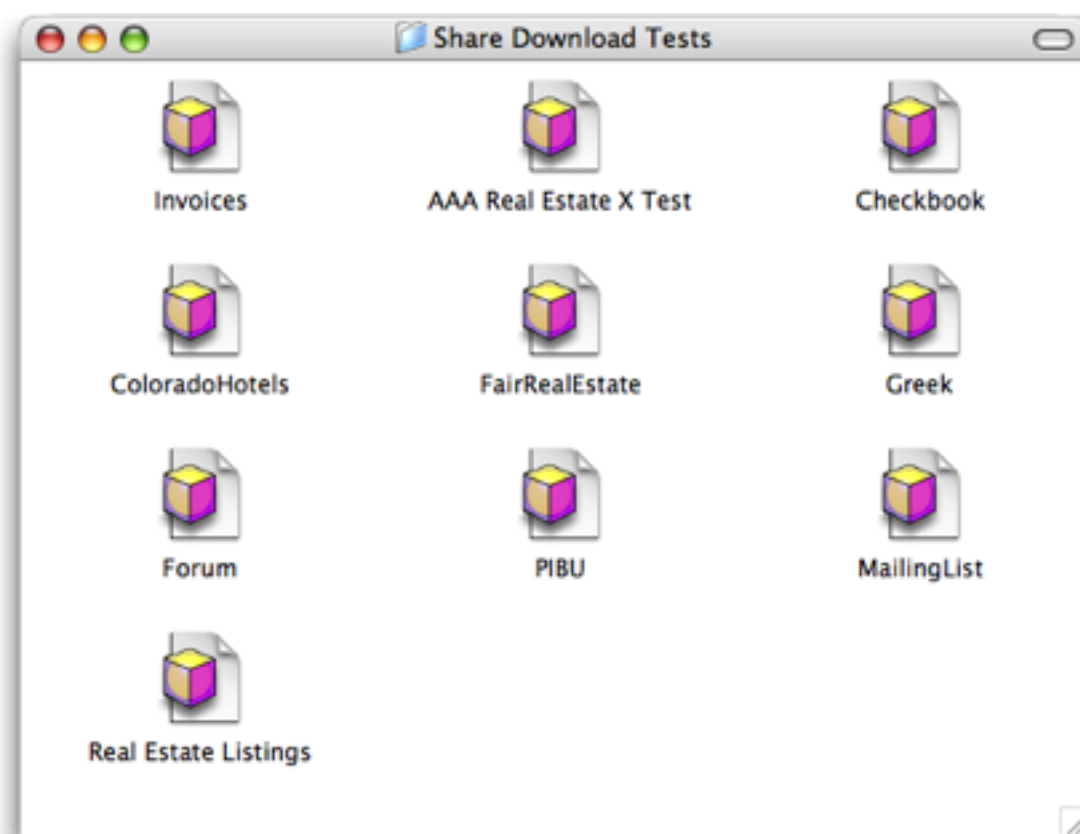
When you press the **Download Databases** button, Panorama will ask you where you would like to put the databases on the local computer. Navigate to the folder you want to use and press the **Save** button.



When the download is complete, the wizard will display a list of the databases that have been downloaded.



You can simply press **Ok**, or press **Reveal in Finder** to display the folder containing the newly downloaded files.



Once a shared database has been downloaded you can open it simply by double clicking (see “[Opening a Shared Databases](#)” on page 115).

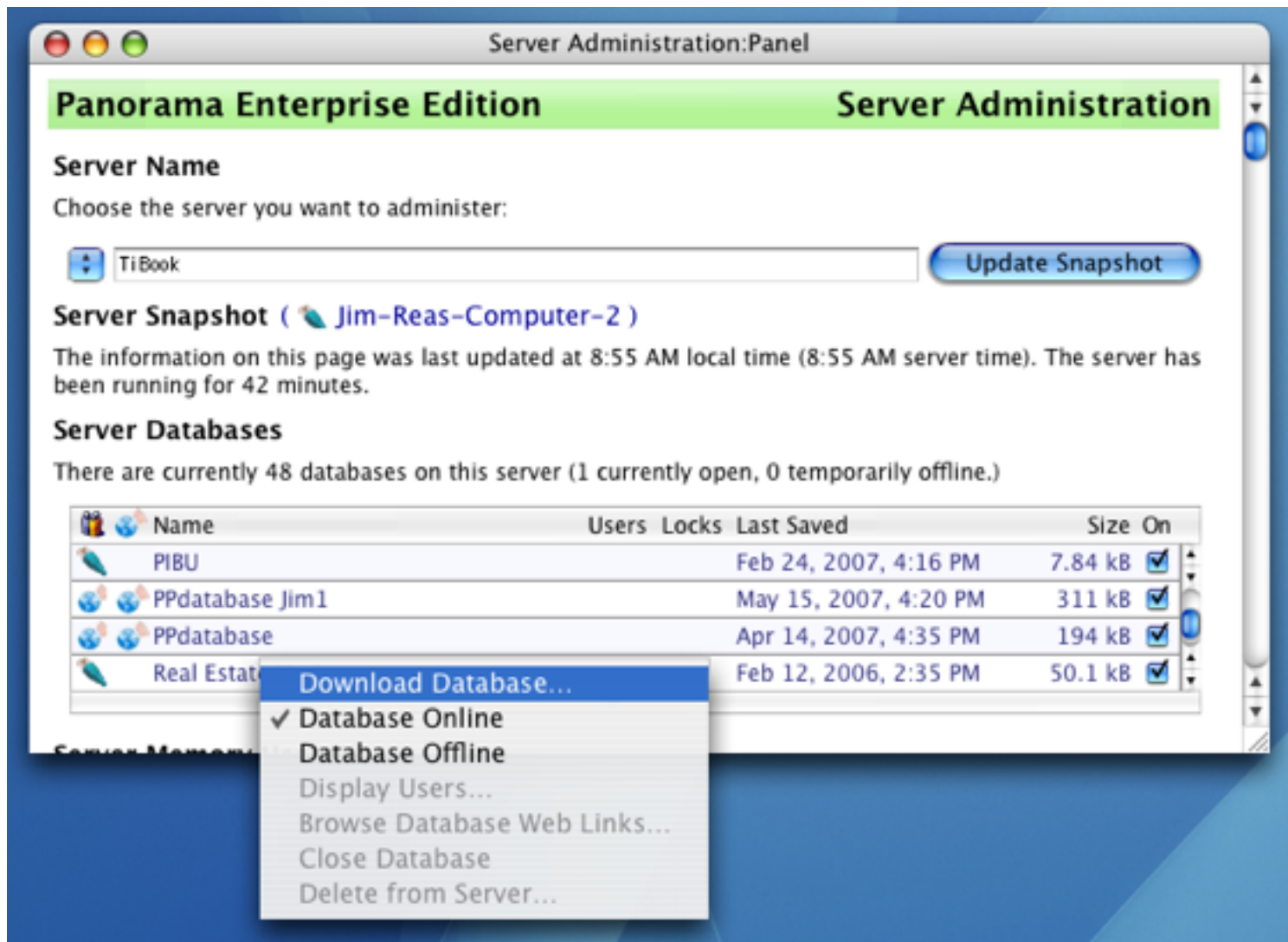
Downloading Offline Databases. If a database is currently off-line, it will be dimmed in the listing (like [Acacia Contacts](#) in the list below).

| | | |
|---|-----------------------|---------|
| <input type="checkbox"/> AAA Real Estate X Test | May 6, 2007, 4:50 PM | 50.5 KB |
| <input type="checkbox"/> Acacia Contacts | May 15, 2006, 4:11 PM | 1.81 MB |
| <input type="checkbox"/> Addendums | May 11, 2006, 4:59 PM | 743 kB |

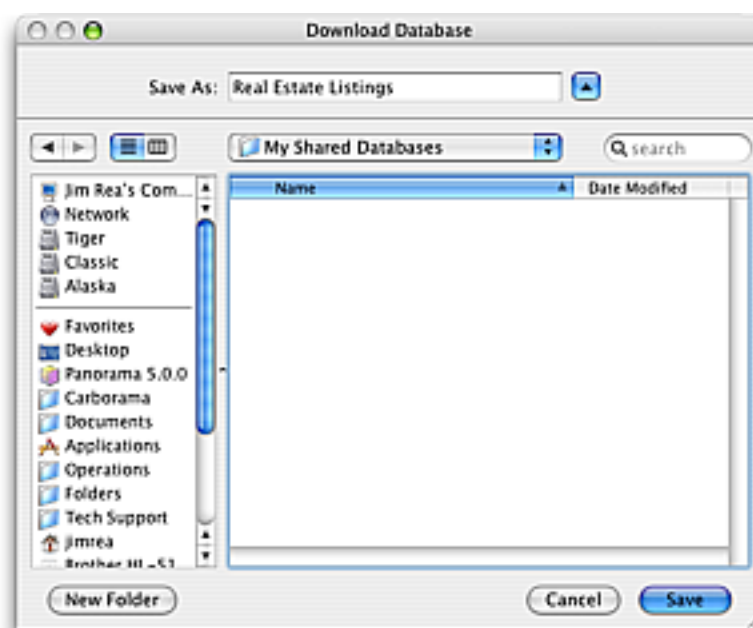
You can download an offline database, but it cannot be opened until the administrator brings it on-line again (see “[Database Online/Database Offline](#)” on page 73). The most common reason for taking a database offline is to update its fields, forms or procedures, so you may want to wait to download this database until the updated version is available.

Downloading with the Server Administration wizard.

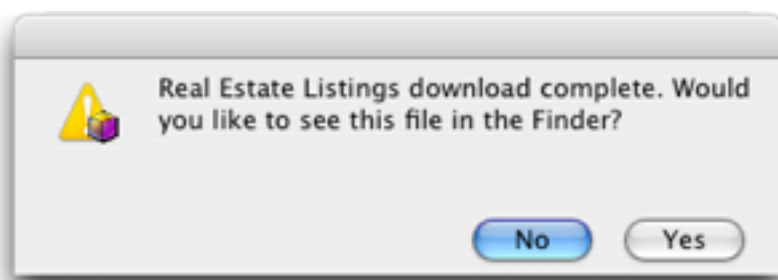
An alternate way to download shared files from the server is to use the **Server Administration** wizard. Start by opening this wizard on the client machine, then choose the server that contains the shared database (you'll be asked for the administrator password). Locate the database you want to download and hold down the **Control** key while you click on the database name. A pop-up menu will appear.



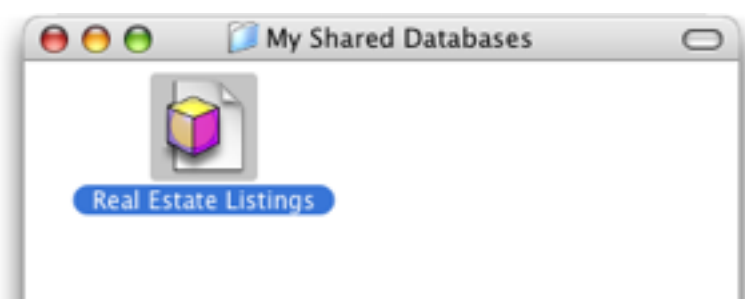
The first choice in this pop-up menu is **Download Database**. When you choose this option, the wizard will ask you where you want to place the downloaded file.



Select a folder, press the **Save** button and the wizard will download the database to the client machine. When the download is complete, the wizard will ask you if you want to see this file in the Finder.



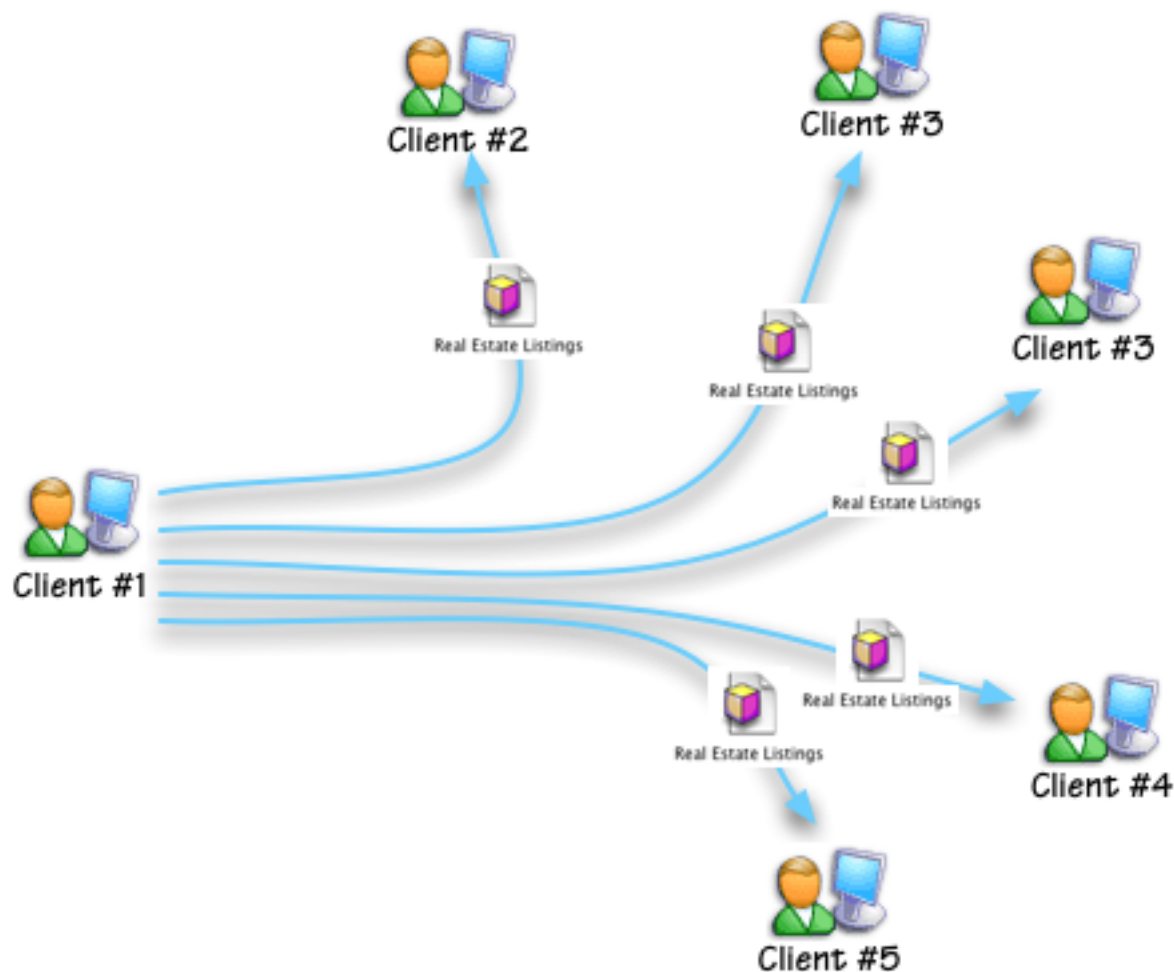
Press **Yes** to switch to the Finder and see the folder that contains the newly downloaded file.



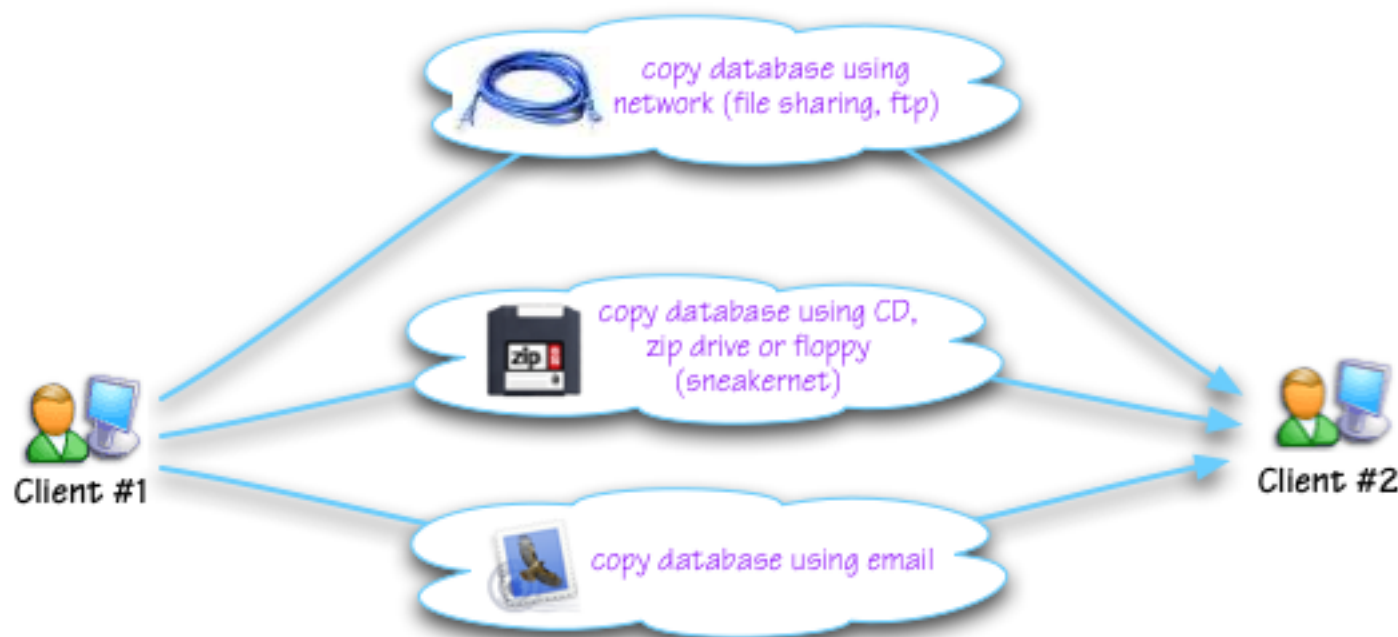
Now you can open this file and begin sharing.

Transferring the Database Manually

As an alternative to downloading the database with a wizard, you can also manually copy the database to each client computer.



An advantage of this approach is that it doesn't require knowledge of the download or server administrator password. Simply use the normal methods you would use for copying any other file. You can transfer the file over a network, using file sharing, FTP, or .Mac. You can transfer the file via sneakernet, using a CD, USB thumb drive, zip drive or (gulp) floppy. You can even e-mail the file as an attachment. Or you can use a combination of these techniques.



If you transfer the file as an e-mail attachment we recommend that you compress the file before sending it (Zip or Stuffit). We've found that some e-mail clients tend to slightly modify "naked" attachment files, which Panorama doesn't like at all. Compressing solves the problem.

If you transfer files via sneakernet, make sure that the media you are using is formatted for Macintosh (unless you are transferring to a Windows computer). If that is not possible then compressing the file usually allows it to be transferred cleanly between systems.

Opening a Shared Databases

Once a database file has been copied to another computer on the local network, it can be opened simply by double clicking on it (assuming, of course, that the computer has a licensed copy of Panorama). Panorama will open the database (load the data into RAM), then connect to the server and synchronize with the server to get the latest modifications (see "[Synchronization](#)" on page 144). Basically it's the same as single user operation — just double click on the database and get to work!

Debugging Local Database Sharing Connection Problems

If the server does not show up at all in the **Available Servers** wizard, double check that both computers are connected to the same local network (the same router). If the client computer is not on the same local network as the server computer you'll have to use internet sharing to connect to the server (see "[Configuring Remote \(Internet\) Database Sharing](#)" on page 62). Also double check that Local Network sharing is enabled on the server (see "[Enabling Database Sharing](#)" on page 55). If these steps don't solve the problem then you may need to increase the Bonjour threshold value (see "[Configuring Local Database Sharing to use TCP/IP instead of Remote Apple Events](#)" on page 59).

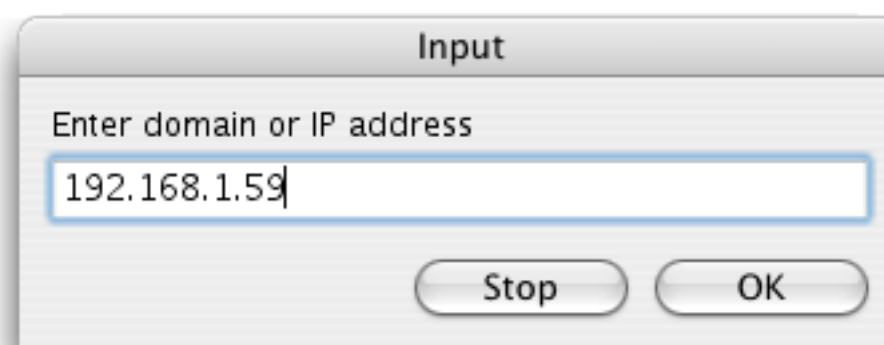
If the server does show up in the list but double clicking on the ethernet plug doesn't work then double check that the server computer has **Remote Apple Events** option enabled in the System Preferences application (see "[Enable Remote Apple Events](#)" on page 33). If that is ok then you can try turning sharing off and on again. When you turn sharing on again, you may want to hold down the **control** or **option** key. This forces Panorama to ask you for the System Administrator password again (if this password is incorrect clients will not be able to connect to the server).

Opening a Shared Database on the Internet

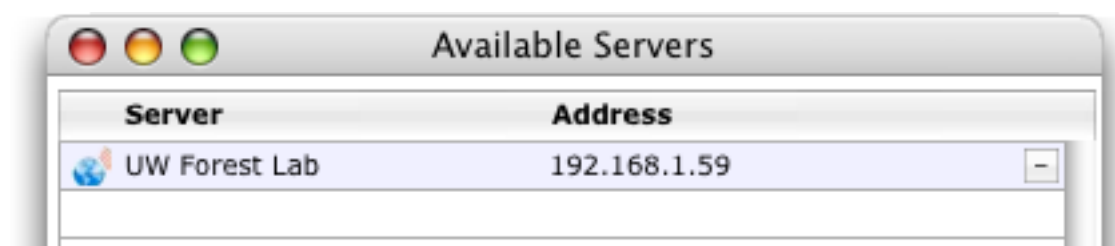
On the local network, Panorama can locate and contact the server automatically. If the server is not on the local network, you must add the server to the list of available servers before the database can be used. This is done with the **Available Servers** wizard (in the **Sharing** submenu of the **Wizard** menu).



Press the **Add Remote Server** button and enter the domain name or IP address of the server.



If you've entered a correct domain name or IP address the server will appear in the list of available servers (see "[Configuring Remote \(Internet\) Database Sharing](#)" on page 62).



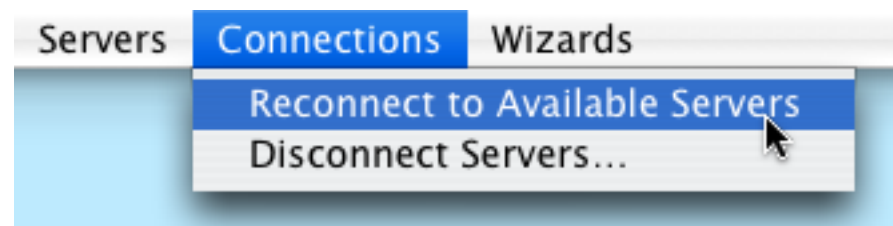
Once the server has been added to the list, you can open the shared database simply by double clicking on it. The database will open, connect to the server and synchronize. You're ready to get to work!

Connecting an Already Open Database to the Server

When you open a shared database, Panorama will normally connect and synchronize the database to the server automatically. However, if there is a connection problem (client not connected to the network, network down, server down) Panorama will still open the client database, but won't connect and synchronize without additional steps.

The first step in making the connection is to resolve whatever problem is preventing the connection. If the client is unplugged from the network, plug it in! If the server is turned off, turn it on, etc.

Simply fixing the network or server isn't enough to get databases that are already open to connect and sync. To force any disconnected open databases to connect and sync, first open the **Available Servers** wizard and choose the **Reconnect to Available Servers** command from the **Connections** menu.



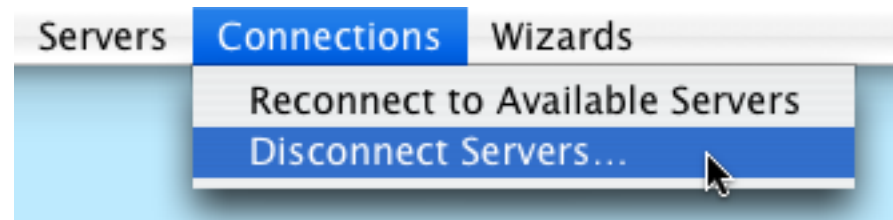
This command will scan all open databases. When it finds a shared database that is not currently connected to the server it checks to see if the server is available, and if so, reconnects and synchronizes the database. This is the fastest and easiest way to connect all databases. When all databases have been scanned a dialog lists the databases that were connected.

(Note: Another method is to close the client database and then re-open it, but this is usually more work than using the **Reconnect to Available Servers** command.)

Disconnecting from a Server

Once a shared database is open and connected to the server, it normally remains connected until you close the database. However, sometimes you may want to disconnect from the network without closing your shared databases (for example to take a laptop on the road). You can simply unplug from the network, but if you do the server will continue to think you are connected (see “[Forcing a Session to Close](#)” on page 79).

If you know you are going to be disconnecting from the network, first open the **Available Servers** wizard and choose the **Disconnect Servers** command from the **Connections** menu.



A dialog will appear listing all of the currently connected servers.



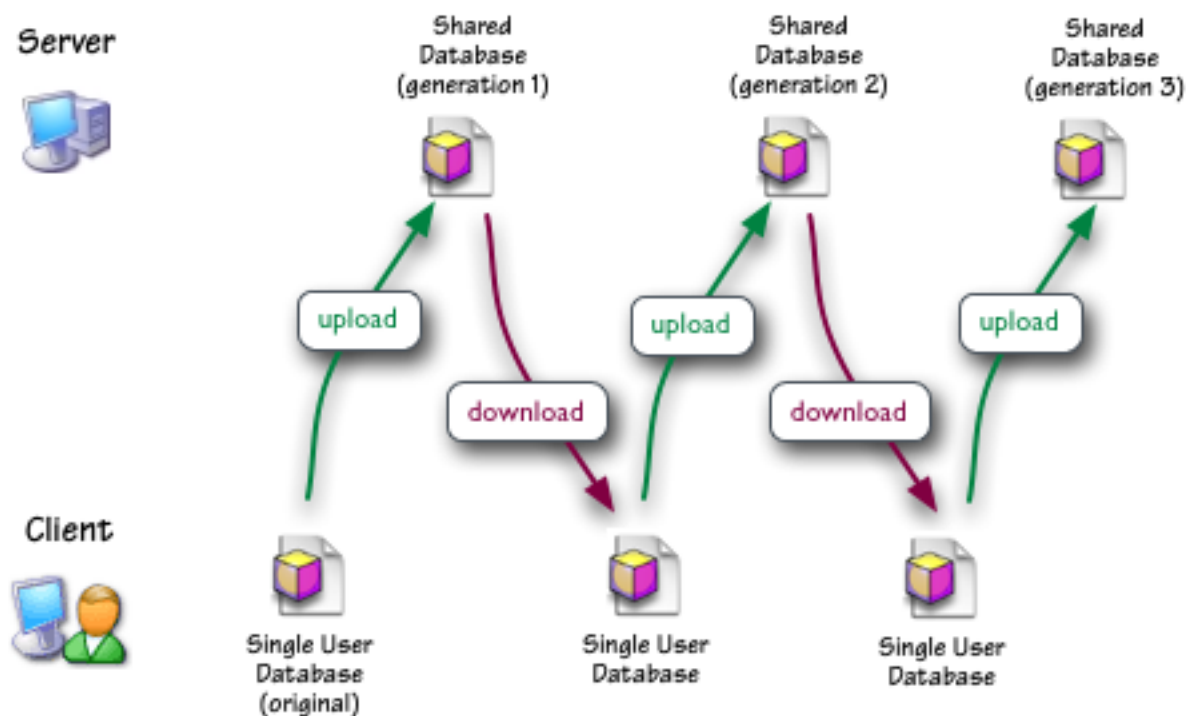
Select the servers you want to disconnect from and press the **Disconnect** button. Any open databases connected to these servers will be disconnected, but will remain open in offline mode (see “[Offline Database Sharing](#)” on page 173). If you later reconnect to the server, you can use the **Reconnect to Available Servers** command to re-establish the connection with the server (see previous section).

Changing the Design of a Shared Database

In a perfect world you might be able to design a perfect database the first time, never having to add, remove or change fields, forms or procedures. In the real world you'll often have to make these sort of changes. The following sections describe how to do this.

Sharing "Generations"

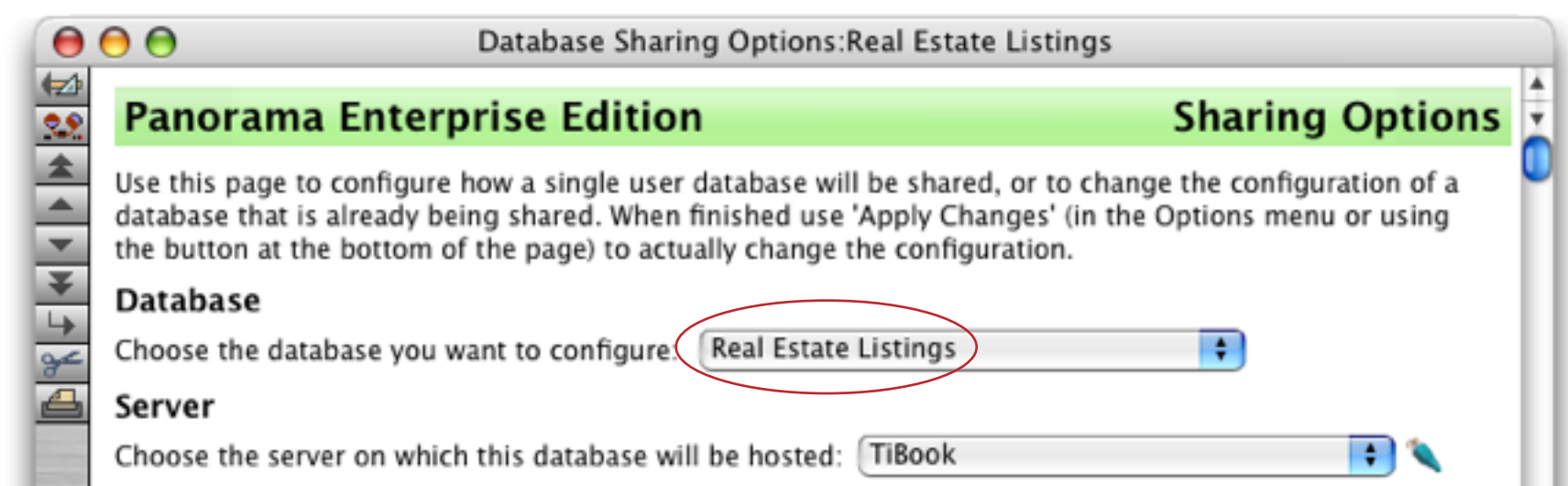
The Panorama Server does not allow you to make design changes directly on the server. To make a design change (adding fields, etc.), you have to download the data to a local computer, make the design changes, then upload the database to the server again. Each time you do this is called a sharing generation.



The shared database will be offline during the process of creating a new sharing generation. This means that other users cannot use the database until the new generation is complete (Panorama will not let you start the new sharing generation process if anyone else is currently using the database.) If possible, you'll usually want to perform this operation during "off hours," for example in the evening or on a weekend.

Starting a New Sharing Generation

The first step in the process of creating a new sharing generation is to open the **Database Sharing Options** wizard (in the **Sharing** submenu of the **Wizard** menu). Make sure that the database to be changed is selected in the pop-up menu (if the database is not already open, open it now).



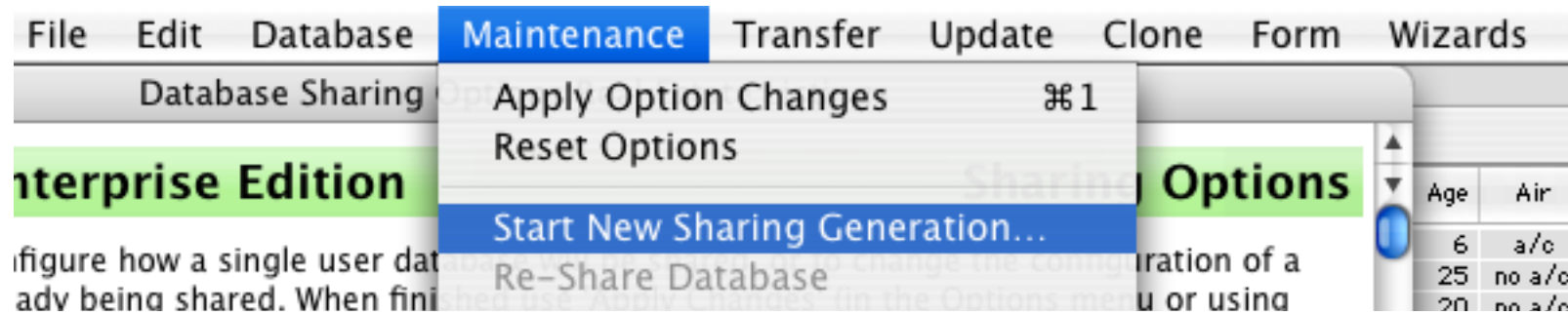
Now press the **New Sharing Generation** button

Server
Choose the server on which this database will be hosted: TiBook

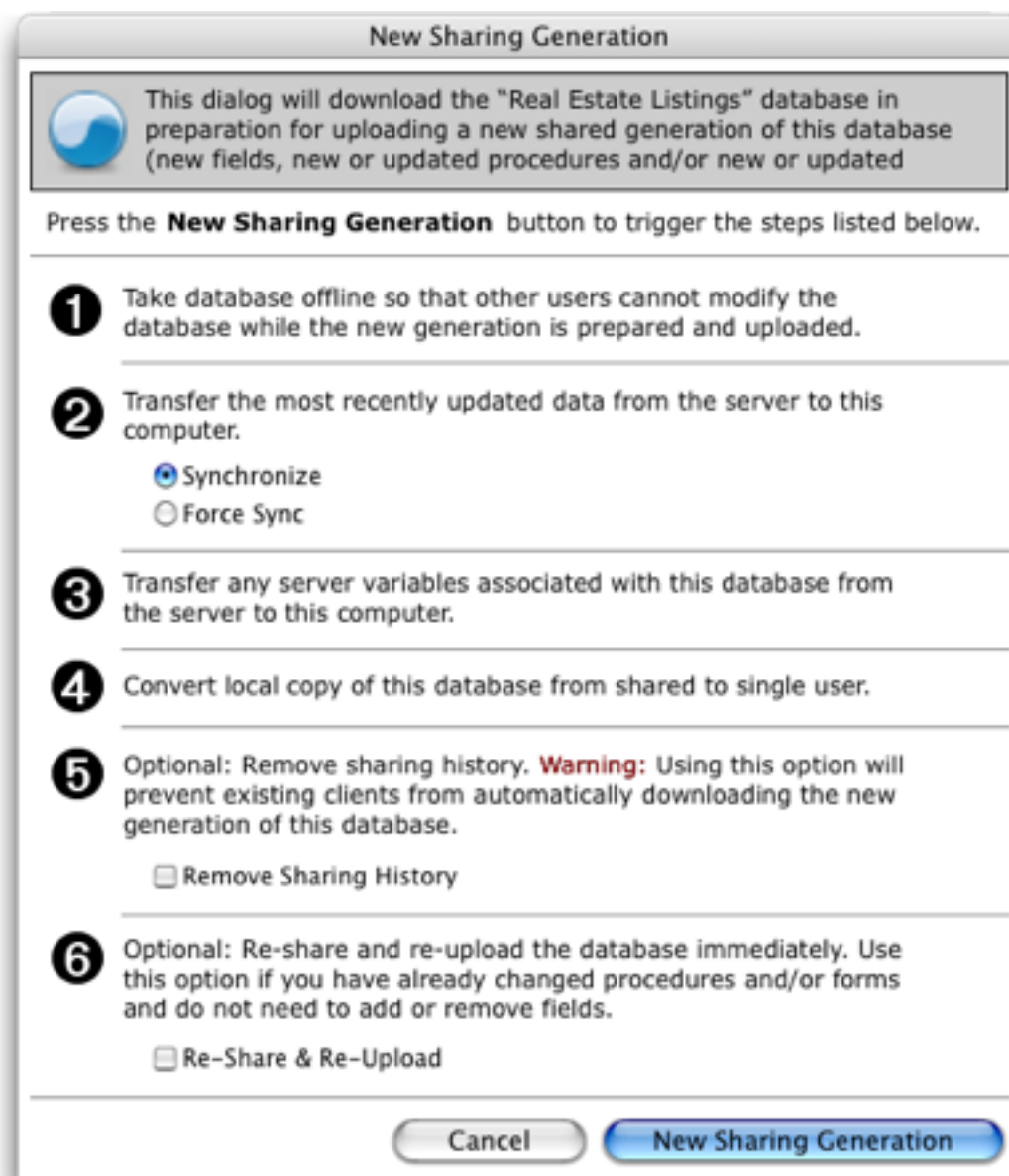
Sharing Mode
Choose how this database will be shared: **New Sharing Generation**

Local Database Sharing Internet Database Sharing Web Publishing

or choose the Start New Sharing Generation command from the Maintenance menu.

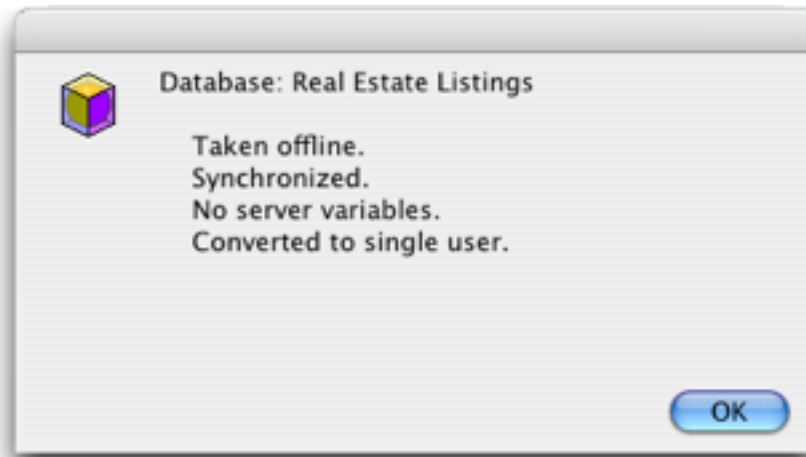


This command displays a dialog that explains what this command is about to do:



This dialog has several options that will be explained later. In most cases, however, you can simply leave these options alone and press the **New Sharing Generation** button.

When the process is complete the wizard will display a dialog like this.



Note: If anyone else is currently using the database the wizard will refuse to perform any of these steps, and will display an error message indicating that other users are currently using the database. You'll have to wait until the other users are finished before starting the new sharing generation process. (Or, if you want to be really rude, you can use the Server Administration wizard to terminate the other users sessions, see "[Forcing a Session to Close](#)" on page 79. Of course this is pretty much guaranteed to make you very unpopular, so this should be avoided if at all possible.)

Adding and/or Removing Fields

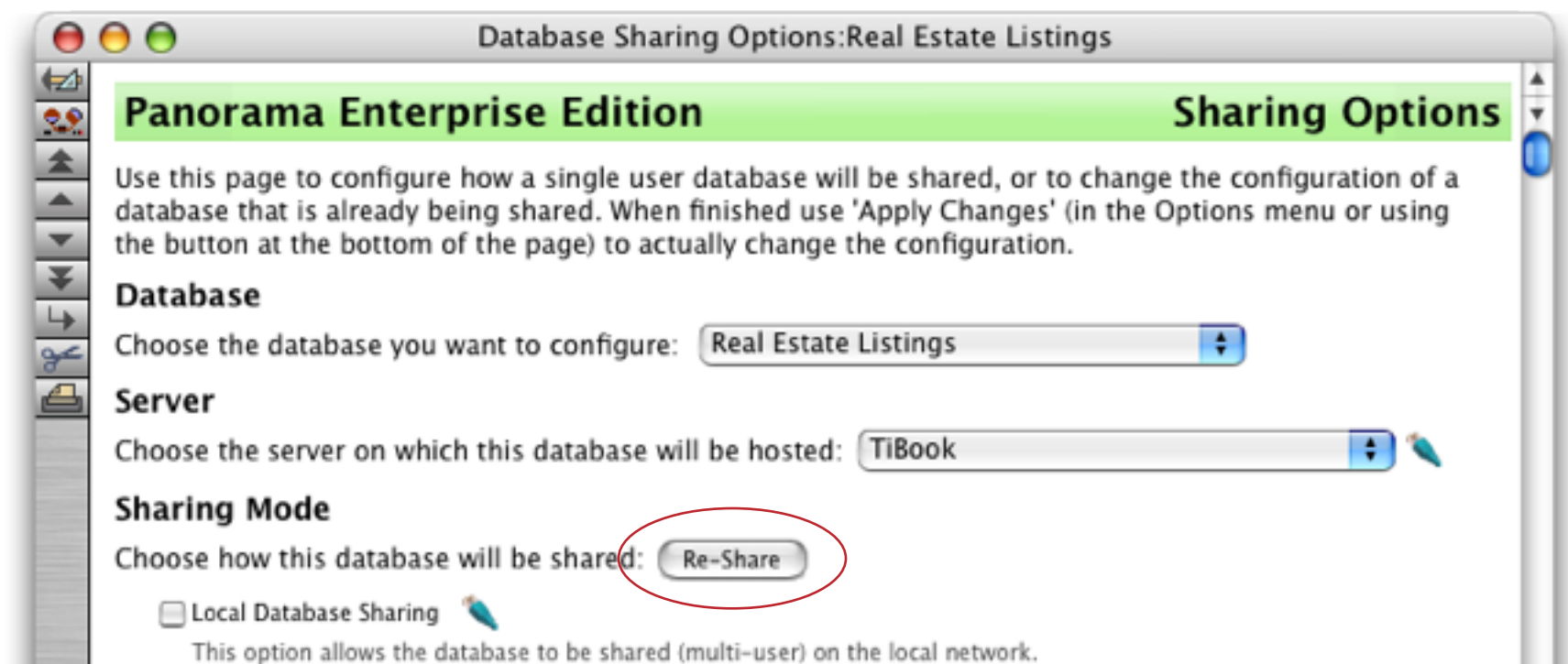
At this point you are ready to make whatever design changes you want to the database. We don't have any special tips for adding or removing fields — simply use either the Design Sheet or the **Setup** menu to add or remove fields as needed. In this case we have added a new **Zip** field for zip codes.

| Address | City | Zip | Asking | Agent |
|--------------------------|------------------|-----|-------------|--------------|
| 33 Elm Street | Huntington Beach | | \$1,053,000 | Stan Bryant |
| 525 Sunset Lane | Irvine | | \$1,260,000 | Ann Rachels |
| 1101 S. Meeker | Garden Grove | | \$513,000 | Bob Weather |
| 9631 Sailfish Drive | Huntington Beach | | \$597,000 | Mary Jacks |
| 8912 Shore Circle | Huntington Beach | | \$660,000 | Bob Weather |
| 735 Elliott Place | Santa Ana | | \$989,400 | Ann Rachels |
| 12241 Fallingleaf | Westminister | | \$552,000 | Mary Jacks |
| 203 N. Walnuthaven Drive | Costa Mesa | | \$540,000 | Stan Bryant |
| 623 Geneva Avenue | Huntington Beach | | \$690,000 | Bob Weather |
| 7836 Connie Lane | Huntington Beach | | \$912,000 | Bob Weather |
| 14802 Adams Avenue | Midway City | | \$714,000 | Tim Tobin |
| 1222 Wildwood Drive | Huntington Beach | | \$990,000 | Tim Tobin |
| 2205 Pacific Avenue | Costa Mesa | | \$779,400 | Rose Richard |

See Chapter 5 of the Panorama Handbook for more information on adding and removing fields.

Uploading the New Sharing Generation

When your design changes are complete, you can use the **Database Sharing Wizard** to upload the new generation to the server. To do so, simply press the **Re-Share** button that magically appeared when the **Start New Sharing Generation** command was complete. You can also use the **Re-Share Database** command in the **Maintenance** menu.



Pressing this button will upload the new generation of this database to the server, completing the new sharing generation process.

Note: If for any reason the **Re-Share** button does not appear (for example if you have quit and relaunched Panorama) you can still re-share the database. Simply check the desired sharing options (**Local Database Sharing**, **Internet Database Sharing**, etc.) and press the **Apply Changes** button, just like you did when you first shared the database. When using this technique the wizard will ask you to confirm that you want to replace the database that was previously uploaded to the server.

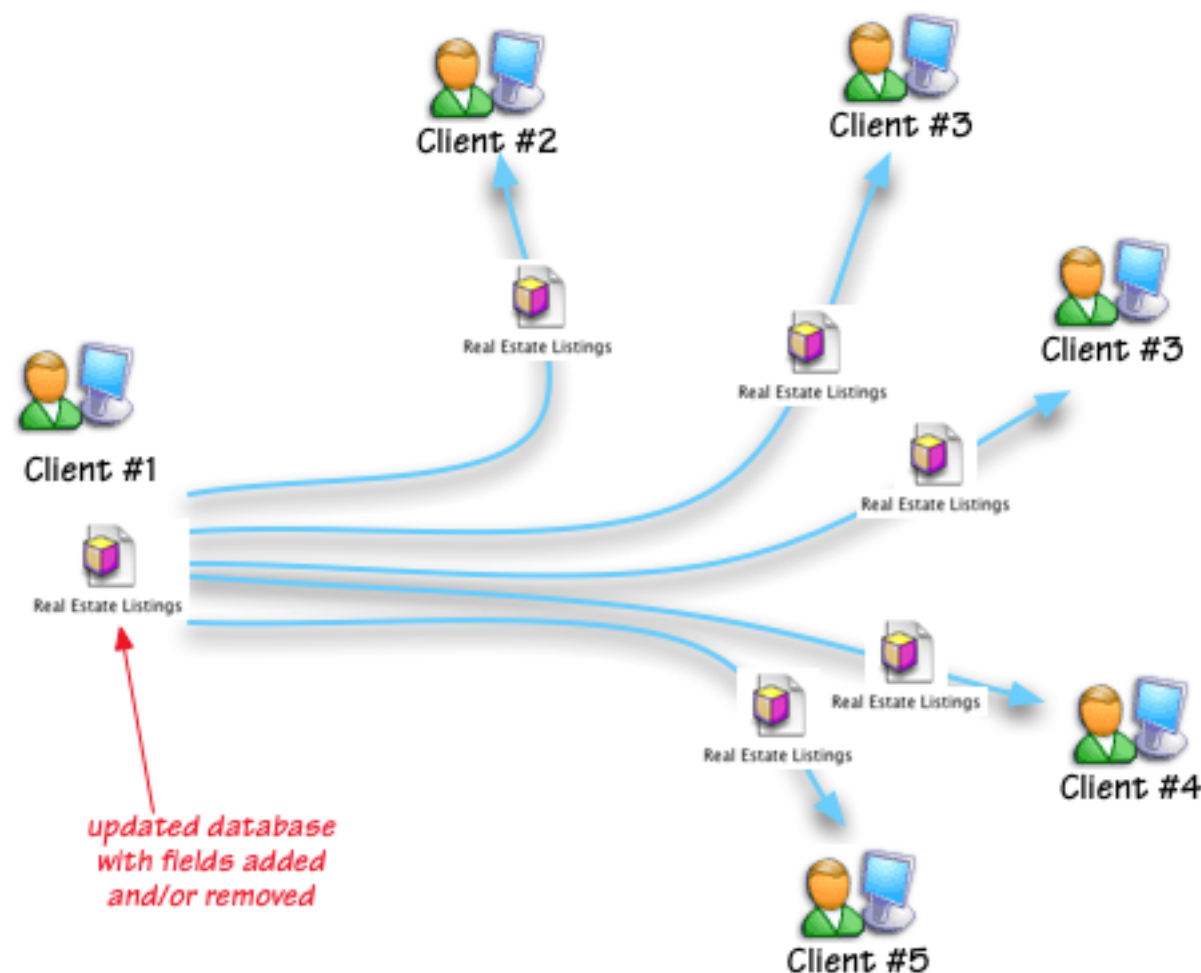
Distributing the New Database Generation to All Clients

The final step is to distribute the updated database generation to all of the other clients on the network (replacing the local copy of the database they already have). The easiest way to do that is to let Panorama do it for you. If a user tries to open an older generation of the database they'll see a message like this:



Click the **Yes** button and Panorama will automatically download the revised database from the server and open it. That's all there is to it!

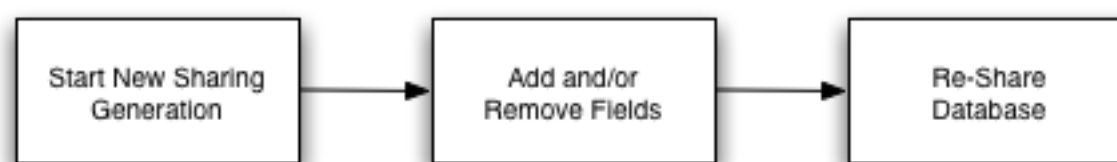
Of course you can also transfer the file directly to the client computers. You can transfer the file any way you like — using the **Download Shared Database** wizard (see “[Using the Download Shared Databases Wizard](#)” on page 110), the **Server Administration Wizard** (see “[Downloading with the Server Administration wizard.](#)” on page 113), or manually using file sharing, FTP, .Mac., via sneakernet using a CD, USB thumb drive, zip drive or (gulp) floppy. You can even e-mail the file as an attachment. Or you can use a combination of these techniques (see “[Duplicate Database Conflicts on the Server](#)” on page 108 for additional tips).



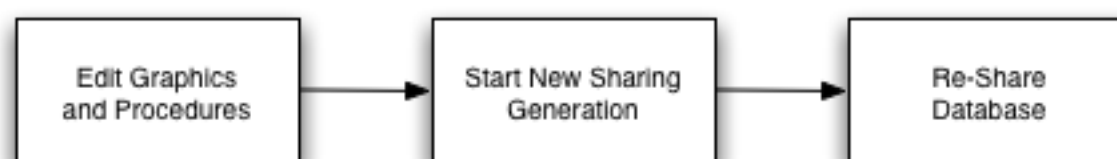
Design Changes to Forms (Graphics) and/or Procedures (Programming)

The previous discussion explained how to add and remove fields. You can use the same technique to change forms and programming. However if you are only making changes to forms and/or programming and are not adding or removing fields there is a shortcut you can take that streamlines the new sharing generation process.

When adding or removing fields you *must* make the actual design changes in the middle of the new sharing generation process, like this:

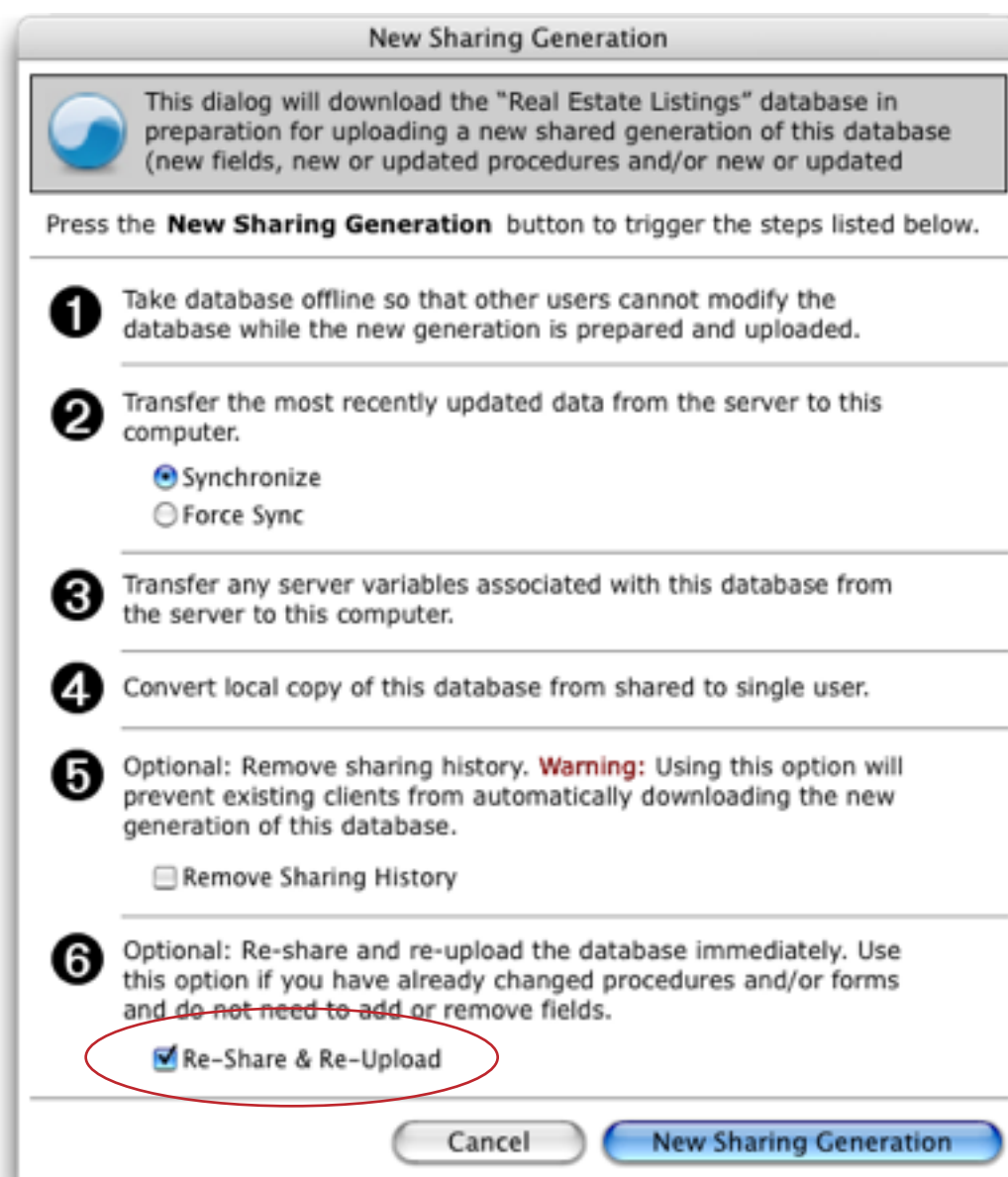


You can make form or procedure changes in the middle of the new sharing generation process as well. But you can also make these changes *before* you start the new sharing generation, like this:

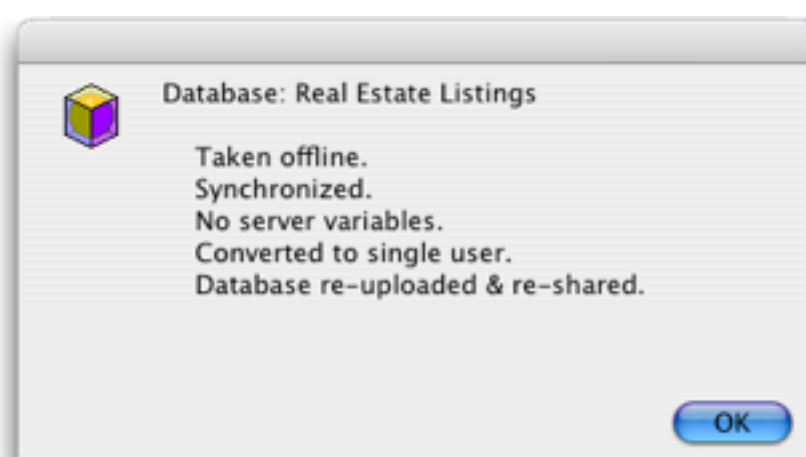


Here are the step-by-step instructions for doing this. Start by opening the database and editing the graphics and/or procedures as needed. (You can actually do this over an extended period of time — it doesn't have to be done immediately before the new sharing generation process is started.)

When the graphics and programming changes are complete, open the **Database Sharing Options** wizard. Then choose the **Start New Sharing Generation** command.



Since you've already made the design changes you need, check the **Re-Share & Re-Upload** option, then press the **New Sharing Generation** button. The wizard will download the latest data from the server and then immediately re-upload the new database generation with the graphic and programming changes you have made.



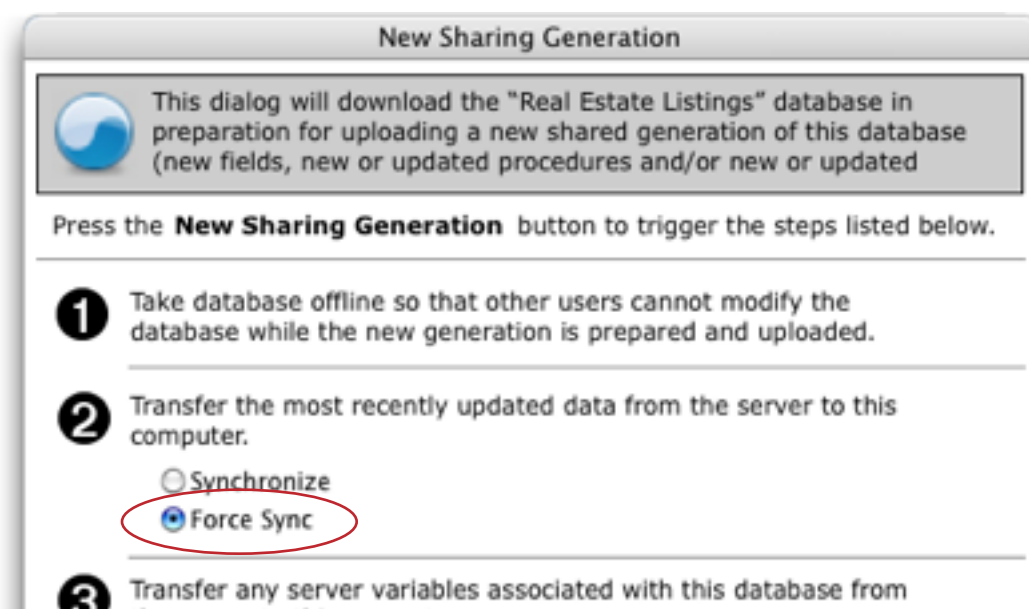
That's it! The only remaining step is to transfer the new generation to the other clients on the network (see ["Transferring the Database to Other Client Computers"](#) on page 110).

New Sharing Generation Options and Advanced Topics

The following sections describe the options available in the **Start New Sharing Generation** dialog, and also explain how to perform the New Sharing Generation process manually (without using the **Start New Sharing Generation** dialog).

Synchronization vs. Force Sync

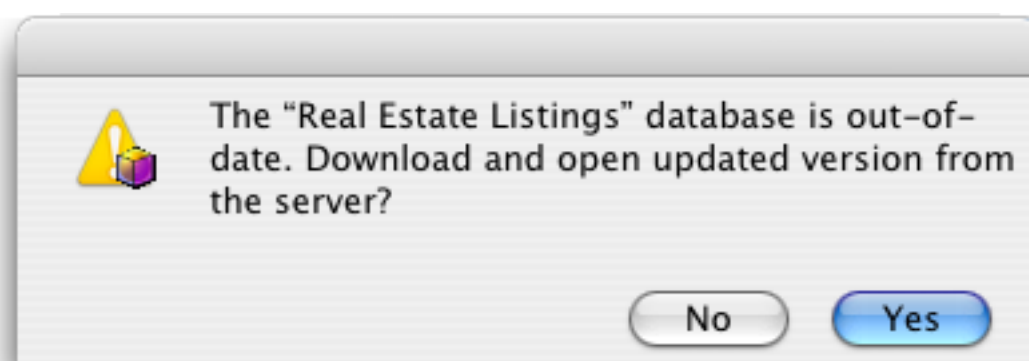
After taking the database offline, the first step in the new sharing generation process is downloading the most recent data from the server to the local computer. This is normally done by synchronizing, which transfers only the records that have actually changed. These updated records are merged with the existing unchanged records in the local database. However you also have the choice of using the **Force Sync** option.



When this option is used Panorama will transfer the entire database from the server to the local database, including both changed and unchanged records (so it will be slower than the normal synchronization process). Theoretically you should never need to use the **Force Sync** option, but the option is there if for any reason you think a database might not synchronize properly (see “[Synchronization](#)” on page 144 for more information on this topic).

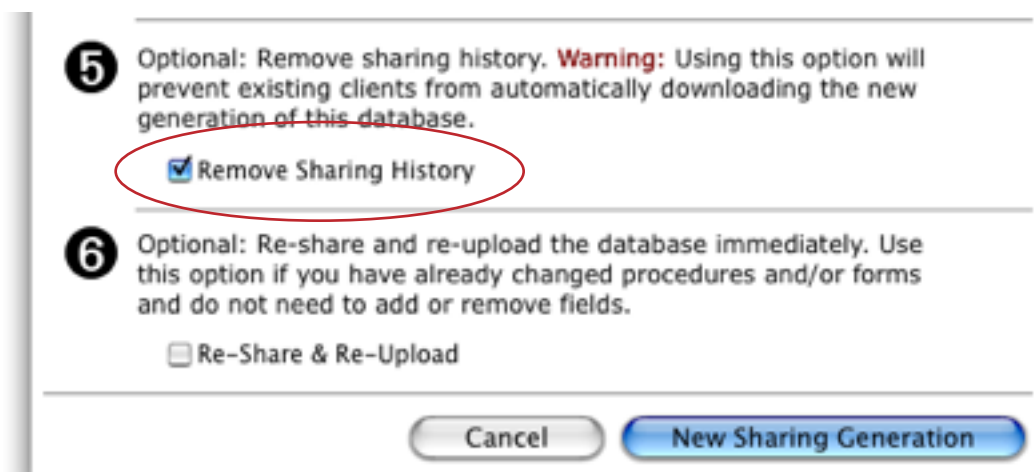
Removing Sharing History

Previously you learned how Panorama can automatically download updates to clients across the network (see “[Distributing the New Database Generation to All Clients](#)” on page 122). To do this Panorama keeps a sharing history for each shared database. If it detects that you are trying to open an older generation of a shared database, it will offer to download the latest version for you.

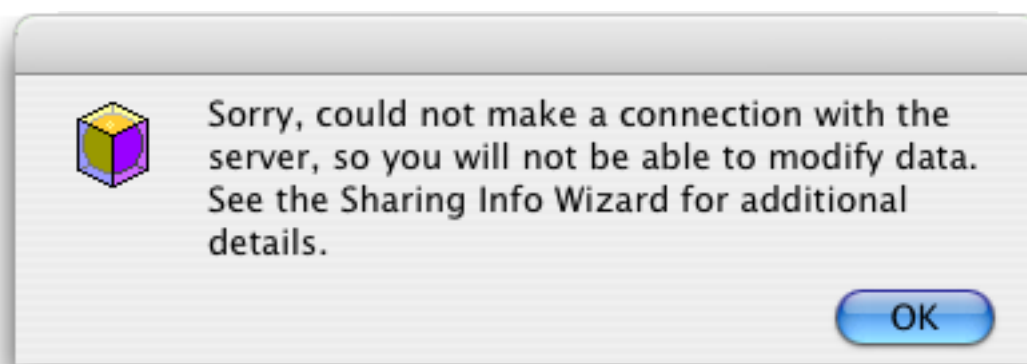


However in some situations you may want to distribute the new client databases manually, rather than allowing automatic downloads. For example, you may decide that some users that had access to this database in the past should not have access in the future. You can do this by removing the sharing history from the database (if these users also had access to the **Download Shared Databases** wizard you will also need to change the appropriate auxiliary passwords, see “[Changing the Auxiliary Passwords](#)” on page 80).

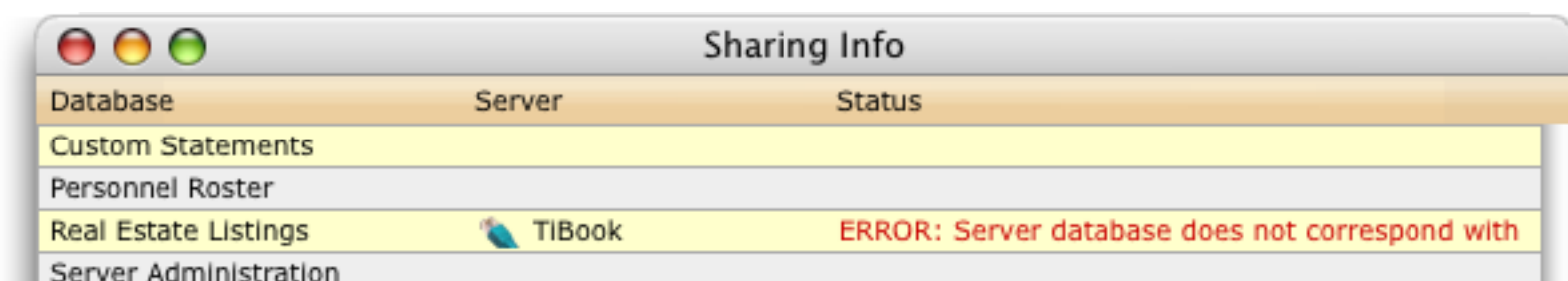
To remove the sharing history simply check the appropriate checkbox on the **Start New Sharing Generation** dialog box.



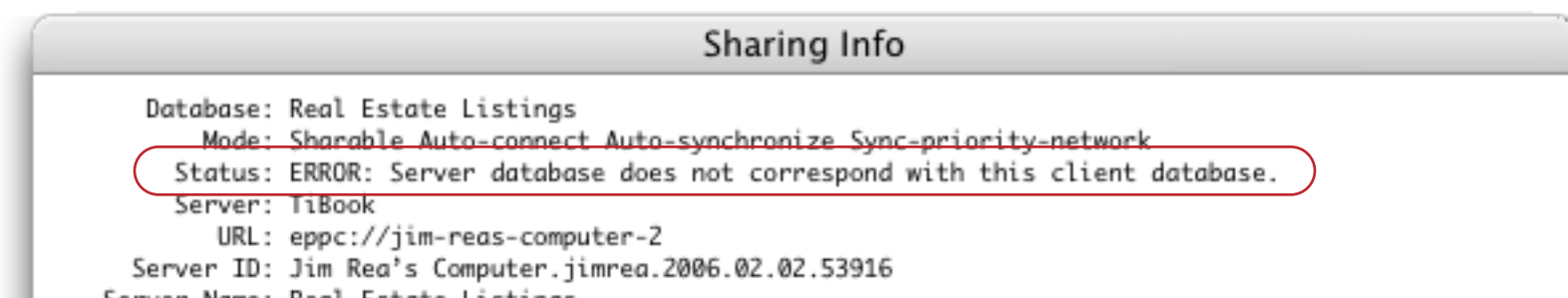
When the database is re-shared it will no longer contain the history of any previous sharing generations. If a user tries to access the server with an older generation of the database it will open, but the database will not connect to the server.



Open the **Sharing Info** wizard to see exactly what the problem is.



To see the full error message either make the **Sharing Info** window wider or double click on this line.



Since the fields in this copy of the database do not match the fields in the server copy of the database Panorama cannot connect them.



| Address | City | Asking | Agent | |
|--------------------------|------------------|-------------|---------------|-------------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,000 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,000 |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,000 |
| 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | |
| 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | |
| 623 Geneva Avenue | Huntington Beach | \$690,000 | Bob Weathers | |
| 7836 Connie Lane | Huntington Beach | \$912,000 | Bob Weathers | |
| 14802 Adams Avenue | Midway City | \$714,000 | Tim Tobin | |
| 1222 Wildwood Drive | Huntington Beach | \$990,000 | Tim Tobin | \$990,000 |
| 2205 Pacific Avenue | Costa Mesa | \$779,400 | Rose Richards | |



Fields do not match!



| Address | City | Zip | Asking | Agent |
|--------------------------|------------------|-----|-------------|-------------|
| 33 Elm Street | Huntington Beach | | \$1,053,000 | Stan Bryant |
| 525 Sunset Lane | Irvine | | \$1,260,000 | Ann Rachels |
| 1101 S. Meeker | Garden Grove | | \$513,000 | Bob Weather |
| 9631 Sailfish Drive | Huntington Beach | | \$597,000 | Mary Jacks |
| 8912 Shore Circle | Huntington Beach | | \$660,000 | Bob Weather |
| 735 Elliott Place | Santa Ana | | \$989,400 | Ann Rachels |
| 12241 Fallingleaf | Westminister | | \$552,000 | Mary Jacks |
| 203 N. Walnuthaven Drive | Costa Mesa | | \$540,000 | Stan Bryant |
| 623 Geneva Avenue | Huntington Beach | | \$690,000 | Bob Weather |
| 7836 Connie Lane | Huntington Beach | | \$912,000 | Bob Weather |
| 14802 Adams Avenue | Midway City | | \$714,000 | Tim Tobin |
| 1222 Wildwood Drive | Huntington Beach | | \$990,000 | Tim Tobin |
| 2205 Pacific Avenue | Costa Mesa | | \$779,400 | Rose Richar |

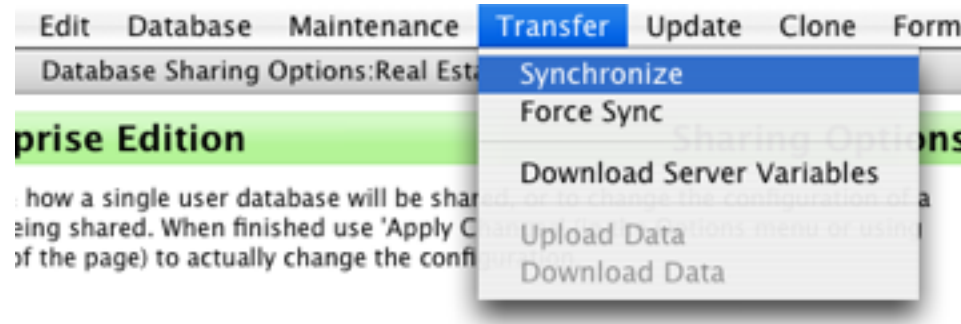
If you later decide that you want this user to be able to access the database you'll need to make arrangements to manually copy the updated generation of the database onto their computer, either with the **Download Shared Databases** wizard (see "[Using the Download Shared Databases Wizard](#)" on page 110), the **Server Administration** wizard, (see "[Downloading with the Server Administration wizard.](#)" on page 113) or using some other method (see "[Transferring the Database Manually](#)" on page 114).

Making a New Sharing Generation Manually

You'll normally want to use the **Start New Sharing Generation** and **Re-Share Database** commands to create new generations of a database. However, it is also possible to do this manually, using separate commands.

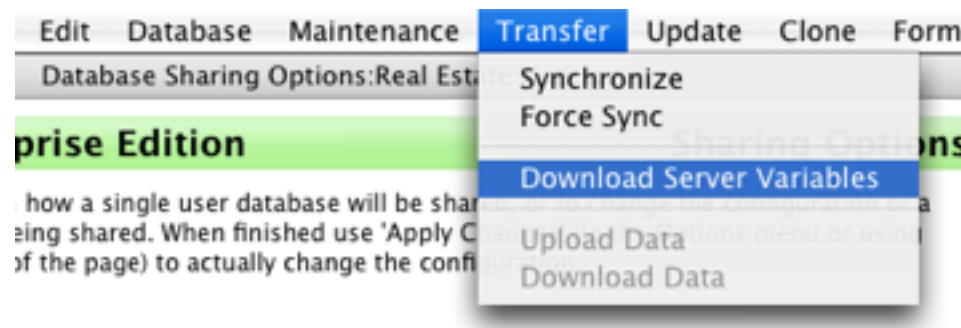
Take the Database Offline. This can be done with the **Database Offline** command in the **Maintenance** menu of the **Database Sharing Option** wizard, or with the pop-up menu in the **Server Administration** wizard (see "[The Pop-up Database Context Menu](#)" on page 72).

Synchronize/Force Sync. This can be done from the within **Database Sharing Options** wizard using commands in the **Transfer** menu.



It can also be done by clicking on the database itself and using the **Synchronize** command in the **File** menu (see "[Synchronization](#)" on page 144).

Download Server Variables. Use the **Transfer** menu in the **Database Sharing Options** wizard for this step.



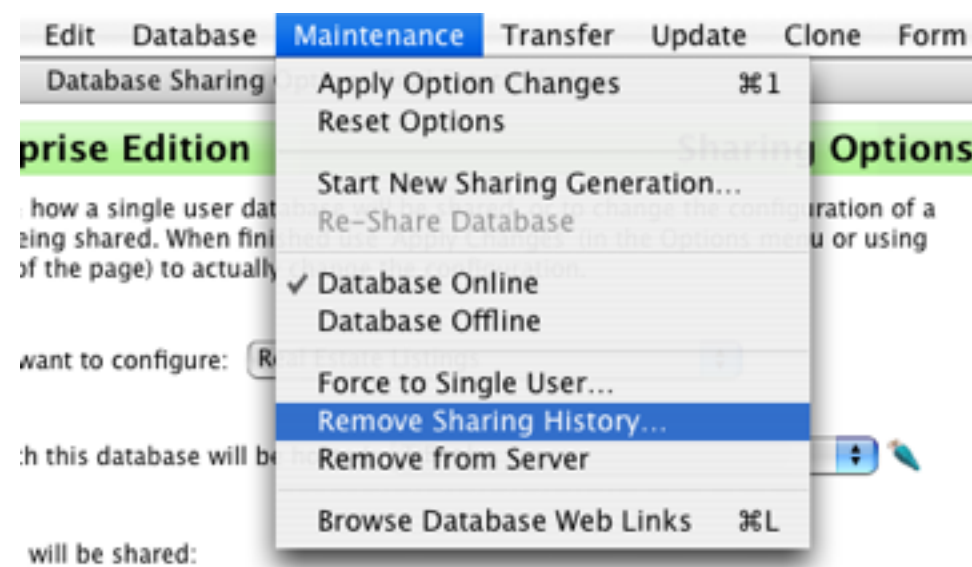
Convert Local Copy of Database to Single User. Using the **Database Sharing Options** wizard, uncheck all of the sharing options for this database. (Note: If you are using "Expert" mode (in the **Form** menu) this window will look slightly different.)



Then press the **Apply Changes** button (at the bottom of the form) or choose **Apply Option Changes** in the **Maintenance** menu. The wizard will ask you to confirm, then will convert the local copy of the database to single user.

Make Design Changes. If you need to add or remove fields, now is the time to do that (see "[Adding and/or Removing Fields](#)" on page 121).

Remove Sharing History (Optional). If you want to remove the history of previous generations you can do so now using the command in the Maintenance menu.



See “[Removing Sharing History](#)” on page 125 for more information on this topic.

Re-Share & Re-Upload Database. Simply check the desired sharing mode options ([Local Database Sharing](#), [Internet Database Sharing](#), etc.). (Note: If you are using "Expert" mode (in the Form menu) this window will look slightly different.)



Then press the **Apply Changes** button (at the bottom of the form) or choose **Apply Option Changes** in the Maintenance menu. The wizard will ask you to confirm the options, then it will ask you to confirm that you want to replace the database that was previously uploaded to the server. Click **Yes** to complete the process.

Distributing the New Shared Generation of the Database. Distribution of the new generation of this database is handled the same way as it was after using the **Start New Shared Generation** command, see “[Distributing the New Database Generation to All Clients](#)” on page 122.

“Unsharing” a Shared Database

Occasionally you may want to permanently convert a shared database back into a single user database. To do this, first open the database, then open the **Database Sharing Options** wizard. (You may want to **Synchronize** first, to make sure you have the most recent data.) This wizard will show that the database is currently being shared. (Note: If you are using "Expert" mode (in the **Form** menu) this window will look slightly different.)

Database Sharing Options:Real Estate Listings

Panorama Enterprise Edition **Sharing Options**

Use this page to configure how a single user database will be shared, or to change the configuration of a database that is already being shared. When finished use 'Apply Changes' (in the Options menu or using the button at the bottom of the page) to actually change the configuration.

Database
Choose the database you want to configure: Real Estate Listings

Server
Choose the server on which this database will be hosted: TiBook

Sharing Mode
Choose how this database will be shared:

- Local Database Sharing This option allows the database to be shared (multi-user) on the local network.
- Internet Database Sharing This option allows the database to be shared (multi-user) on the internet.
- Web Publishing This option allows the database content to be published on the web (for use by web browsers). The database must have

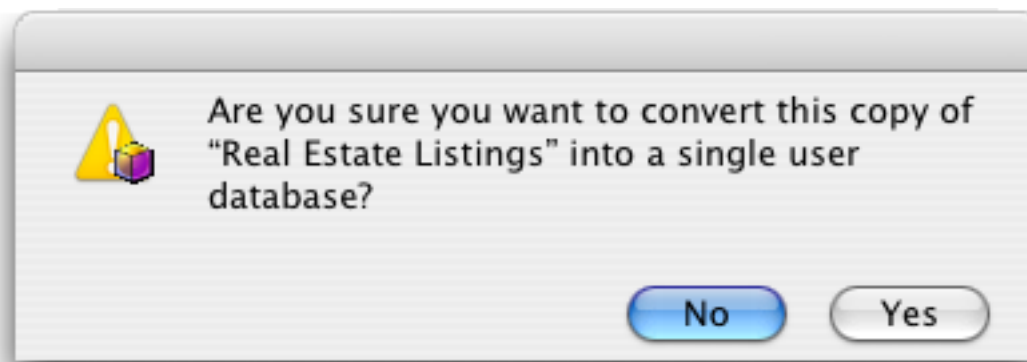
To convert back to single user operation, uncheck the **Local Database Sharing** and **Internet Database Sharing** options, then choose **Apply Option Changes** from the **Maintenance** menu (or press the **Apply Changes** button at the bottom of the form). The wizard will ask you to confirm the new settings.

Confirm New Sharing Options

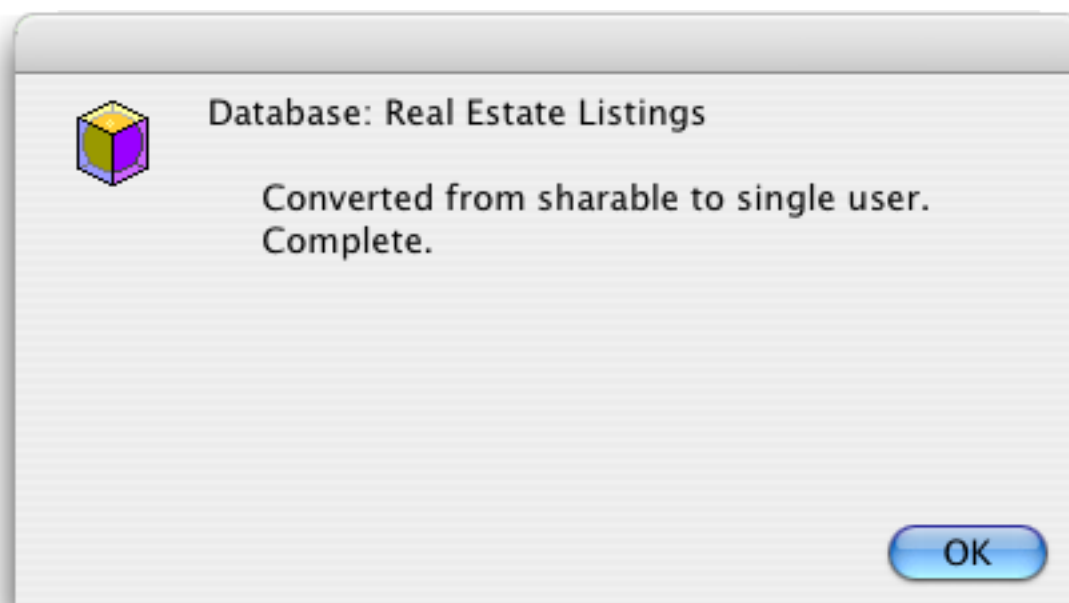
| | |
|--------------------------|------------------------|
| Sharable | = no |
| RemoteSharable | = no |
| WebPublish | = no |
| ServerURL | = TiBook |
| AutoTimeOut | = 20 |
| AllowOfflineModification | = no |
| KeepOfflineChangesLocal | = no |
| AllowOfflineAddRecord | = no |
| AllowOfflineDeleteRecord | = no |
| SyncPriority | = Local |
| AutoConnect | = yes |
| AutoSync | = yes |
| StartOpen | = no |
| AutoOpen | = no |
| AutoClose | = no |
| AutoSave | = no |
| Initialize | = no |
| UseSecret | = no |
| ServerDatabaseName | = Real Estate Listings |

Cancel Apply Options

The wizard will give you one last chance to back out.



If you press **Yes**, the database will be “un-shared.”

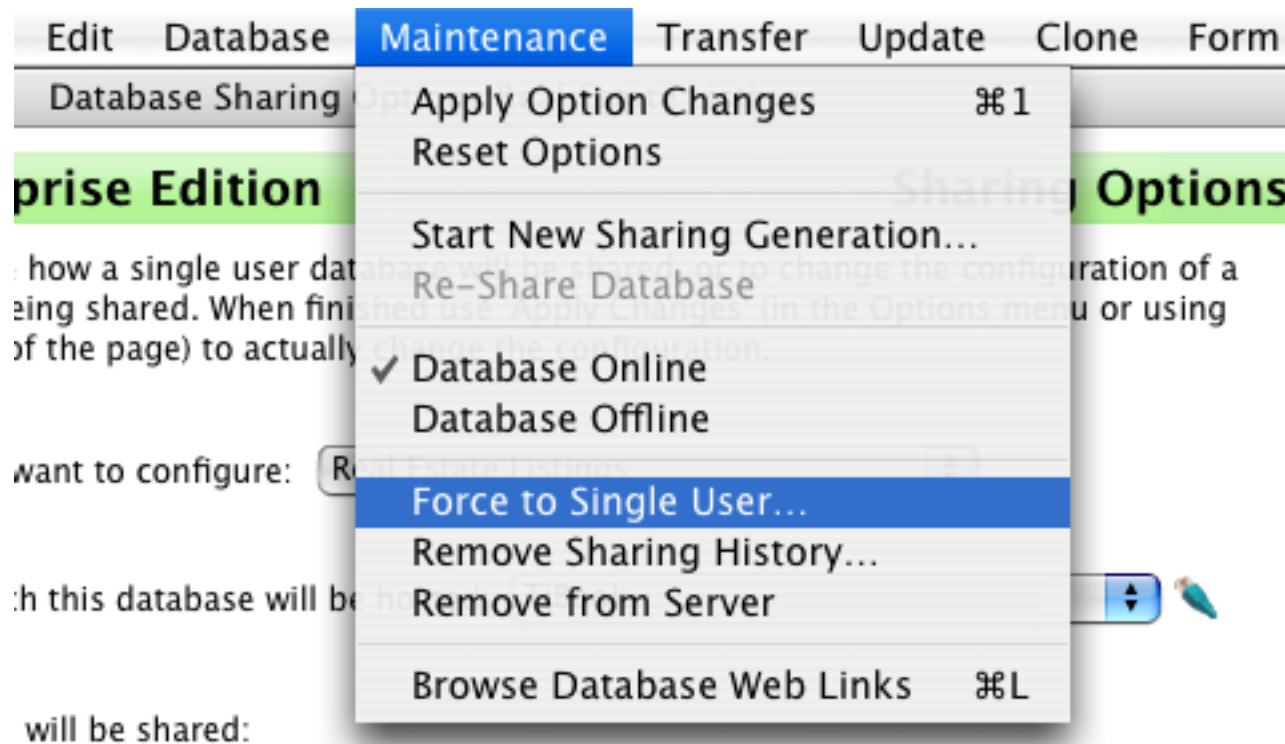


This copy of the database is now a single user database again, with no connection to the server.

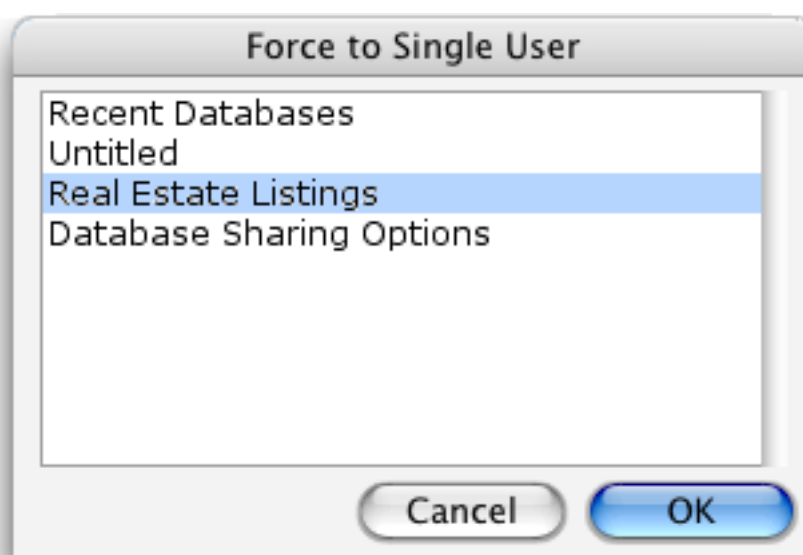
IMPORTANT NOTE: Converting a copy of the database on a client computer only affects that copy of the database. It does not affect the server or other users of the database, which can continue to share the database even though you’ve converted this one copy of the database back to single user. If you want to actually remove the database from the server see [“Permanently Deleting a Database from the Server”](#) on page 133.

Forcing a Shared Database Back to Single User

Sometimes the unsharing process described in the previous section may not work for one reason or another. For example the original server may no longer be available, or the database sharing information inside the database may be garbled. If the normal conversion to single user described above does not work, you can use the **Force to Single User** command in the **Maintenance** menu of the **Database Sharing Options** wizard.



The command displays a dialog listing the currently open databases.



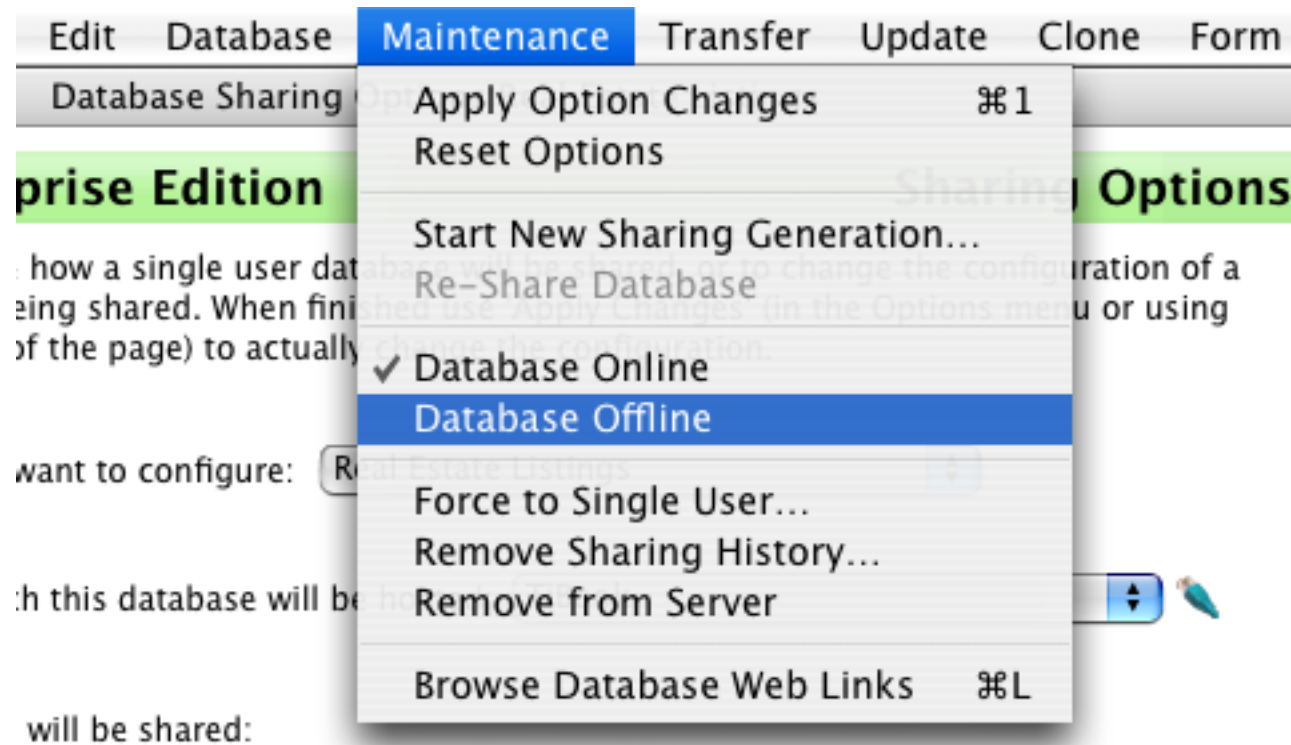
Choose the database that you want to force to single user and press the **Ok** button. This will force the database to single user mode and reset all server related options back to their default values, removing any trace of the fact that this database was once shared (including the sharing history, see “[Removing Sharing History](#)” on page 125). For example you might want to use this command if you wanted to send a single user copy of the database to another person. If you plan to convert the database back to shared again on the original server you should probably avoid this command.

Note: Forcing a database to single user on a client computer only affects that copy of the database. It does not affect the server or other users of the database, which can continue to share the database even though you’ve forced this one copy of the database back to single user. If you want to actually remove the database from the server see “[Permanently Deleting a Database from the Server](#)” on page 133.

Permanently Deleting a Database from the Server

There are two ways that you can permanently delete a database from the server — using the **Database Sharing Options** wizard or using the **Server Administration** wizard. Either method is permanent and cannot be reversed or undone. (The **Server Administration** wizard method has the advantage that you can delete a database without having a copy of the database on your local computer.)

To delete a database using the **Database Sharing Options** wizard, first open the database on the local computer and then open the wizard. Before you can actually delete the database you must first take it offline. This can be done with the **Maintenance** menu.



If any other users are currently using the database, Panorama will refuse to take the database offline. You'll have to wait until they finish to delete the database. (On the other hand, if other users are using this database perhaps you should reconsider whether or not it should be permanently deleted!)

Once the database is offline you can delete it from the server using the **Remove from Server** command.

Deleting a database with the **Server Administration** wizard is similar. Once the wizard is open, hold down the **Control** key while you click on a row in the database table (or right-click if you have a two button mouse). This will cause a pop-up context menu to appear. The exact contents of this menu depend on the current status of the database, but it will look something like this:

Server Databases

There are currently 14 databases on this server (1 currently open, 0 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|-----------------------------|-------|-------|-----------------------|----------------|-------------------------------------|
| Offline Mailing List | | | Nov 4, 2005, 12:40 PM | 21.0 KB | <input checked="" type="checkbox"/> |
| Real Estate Listings | | | Feb 12, 2006, 2:35 PM | 48.9 KB | <input checked="" type="checkbox"/> |
| StateAbbreviationsTi | | | Sep 1, 2005, 4:26 PM | 2.2 KB | <input checked="" type="checkbox"/> |
| Test X Price List | | | Nov 11, 2005, 4:21 PM | 6.7 KB | <input checked="" type="checkbox"/> |

Use this pop-up menu to take the database offline.

Server Databases

There are currently 14 databases on this server (1 currently open, 1 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|----------------------|-------|-------|------------------------|----------|-------------------------------------|
| Forum | | | Nov 4, 2005, 1:04 PM | 16.6 KB | <input checked="" type="checkbox"/> |
| HobbyShopCatalog | | | 12, 2006, 7:08 PM | 150.9 KB | <input type="checkbox"/> |
| Invoices | | | Jun 27, 2005, 11:19 AM | 48.5 KB | <input checked="" type="checkbox"/> |
| Offline Mailing List | | | Nov 4, 2005, 12:40 PM | 21.0 KB | <input checked="" type="checkbox"/> |

database is offline

Notice that when a database is temporarily offline, it is dimmed (gray) in the database table.

Once the database is offline, use the pop-up menu again to actually delete the database from the server.

Shared vs. Single User Database Operation

So far we've discussed how to create, open and change the design of shared databases. The following sections describe how shared databases work once they are opened. Operation of shared Panorama databases is very similar to single user use, but there are some differences.

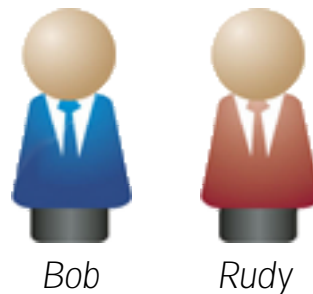
| Operation | Notes |
|------------------------------------|---|
| Open Database | Opens database from local hard disk, then connects to and synchronizes with server (see " Synchronization " on page 144). |
| Close Database | Closes database on local computer (may remain open on other computers and on server) |
| Save | Saves database on local hard disk, and also saves on server as well. (Note: Server will maintain changes even if server operation is interrupted without saving, see " Handling Interruptions in Server Operation (Crash Recovery) " on page 153.) |
| Revert to Saved | Cannot roll back data with this command (master copy of data on server cannot be reverted), however you can still use this command to revert work done in graphics mode or procedures. |
| Searching, Sorting, Printing, etc. | Exactly the same as single user. Operations which don't modify the database take place entirely in the local computer's RAM and are just as fast as when using a single user Panorama database. |
| Data Editing | Works the same as single user unless someone is already editing the same record. The Panorama Server has full record locking to prevent conflicts. |
| Summary Records | Grouping and summary operations (Total, Average, etc.) work exactly the same as single user. Summary records are not shared and are not record locked, so different users can group and summarize the database different ways at the same time. You cannot manually create or destroy individual summary records. |
| Fills | To ensure record locking Panorama must contact the server for every cell that is filled, making these commands (Fill, Formula Fill, etc.) significantly slower than when used in single user mode. As much as possible you should minimize the number of cells that are filled. There is also a new programming statement that performs the fill on the server instead of the client, this is much faster but requires modification to your procedures. See " ServerFormulaFill — A Much Faster Option for Select/Formula Fill Operation " on page 163. |
| Procedures | Procedures operated exactly the same as single user. Additional statements allow the programmer to take explicit control over record locking and to create and access server variables. |
| Flash Art™ | Since Flash Art is normally used to display images from the local hard drive it usually needs to be redesigned to work in a shared environment. Flash Art can now display images directly from the web, so one option is to put all of the database images on a web server and display them from there. |
| Import | Appending imported text is not quite as fast as single user, but pretty close. |
| Export | Exactly the same as single user. |

The majority of Panorama databases can be easily converted to shared databases with little or no modifications to data, graphics or procedures.

If you have previously used the Panorama 3-4/Butler database sharing system the Panorama Enterprise Edition Server is similar, but with some big improvements. See "[Enterprise Sharing vs. Butler](#)" on page 489 for information on these differences and for instructions on converting a database shared with this previous system to the new Enterprise server.

Editing Data and Record Locking

Panorama only allows one user to edit each record at a time. Once you begin editing the record remains locked until you move to another record, another window, or save the database. The best way to understand record locking is to see it in action. We'll follow along as "Bob Blue" and "Rudy Red" both edit the Real Estate Listings database.



Bob

Rudy

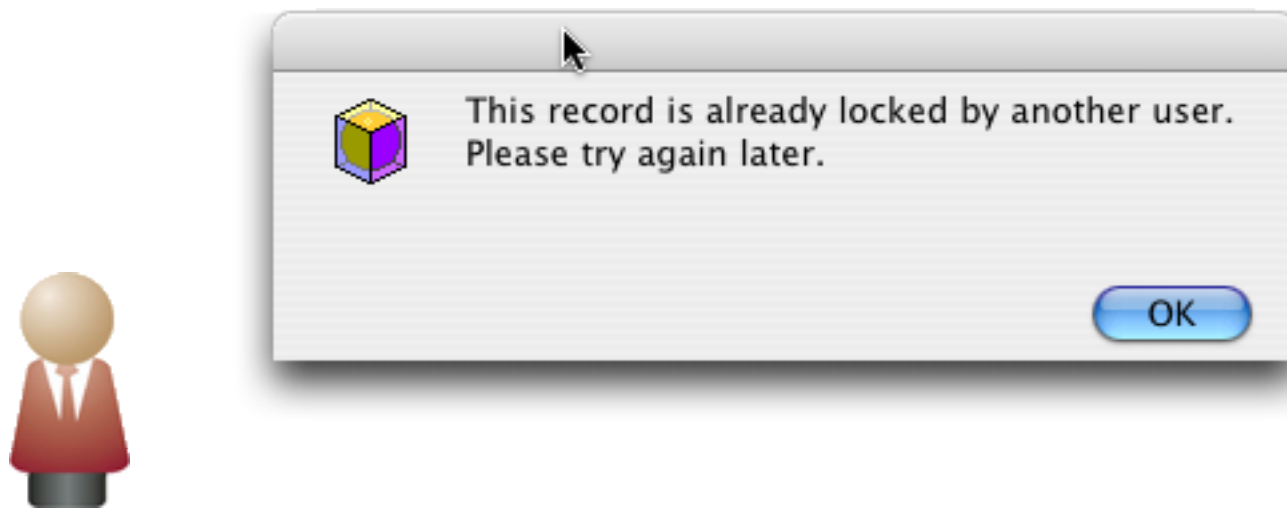
Bob gets a phone call that there is an offer of \$675,000 on the Geneva Street house! So he opens the listings database and double clicks to enter the new offer.

| Address | City | Asking | Agent | Offer |
|--------------------------|------------------|-------------|--------------|-------------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | 675000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 |
| 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | |
| 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | |
| 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 |
| 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | |
| 623 Geneva Avenue | Huntington Beach | \$690,000 | Bob Weathers | |
| 7836 Connie Lane | Huntington Beach | \$912,000 | Bob Weathers | |

At the same time, Rudy is checking his e-mail and receives a message that **Tim Tobin** is not the Agent on the 632 Geneva Street house, it should be **Mary Jackson**. So he opens the listing database and double clicks on the agent name.

| Address | City | Asking | Agent | Offer |
|--------------------------|------------------|-------------|--------------|-------------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 |
| 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | |
| 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | |
| 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 |
| 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | |
| 623 Geneva Avenue | Huntington Beach | \$690,000 | Bob Weathers | |
| 7836 Connie Lane | Huntington Beach | \$912,000 | Bob Weathers | |

But when Rudy double clicks, Panorama doesn't let him edit the agent name. Instead, it tells him that this record is already being edited by someone else.



Bob has finished entering the new value, but he still hasn't moved to another record.



When Rudy tries again to edit the Agent name he finds that it is still locked.



Now Bob moves on to a different record.



| Address | City | Asking | Agent | Offer |
|--------------------------|------------------|-------------|--------------|-------------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | \$675,000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 |
| 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | |
| 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | |
| 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 |
| 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | |
| 623 Geneva Avenue | Huntington Beach | \$690,000 | Bob Weathers | |
| 7836 Connie Lane | Huntington Beach | \$912,000 | Bob Weathers | |

This unlocks the **623 Geneva Street** record so that Rudy can now edit it.



| Address | City | Asking | Agent | Offer |
|--------------------------|------------------|-------------|--------------|-------------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | | \$675,000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 |
| 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | |
| 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | |
| 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 |
| 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | |
| 623 Geneva Avenue | Huntington Beach | \$690,000 | Bob Weathers | |
| 7836 Connie Lane | Huntington Beach | \$912,000 | Bob Weathers | |

Notice that the new \$675,000 offer that Bob entered now appears in Rudy's copy of the database. Whenever you begin editing a record Panorama not only locks that record, it also updates your copy of the record with the latest data from the server.

Rudy completes his modification by typing in the new agent name and then moving to another record to unlock the record he just modified. (A record can also unlock automatically after a specified period of inactivity, see "[Record Lock Timeout \(Client\)](#)" on page 139.)



| Address | City | Asking | Agent | Offer |
|--------------------------|------------------|-------------|--------------|-------------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Mary Jackson | \$675,000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 |
| 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | |
| 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | |
| 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 |
| 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | |
| 623 Geneva Avenue | Huntington Beach | \$690,000 | Bob Weathers | |
| 7836 Connie Lane | Huntington Beach | \$912,000 | Bob Weathers | |

Bob can't see Rudy's change right away, but it will appear if he edits this record again or chooses **Synchronize** from the **File** menu.



| Address | City | Asking | Agent | Offer |
|---------------------|------------------|-------------|--------------|-------------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Mary Jackson | \$675,000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 |
| 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | |
| 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | |
| 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 |

Synchronization will be discussed in detail later in this chapter (see “[Synchronization](#)” on page 144).

Record Lock Timeout (Client)

When you edit a record, Panorama normally keeps it locked until you do one of three things—move to another record in the same database, bring a different database to the front, or save the database. But what if someone starts editing a record and then gets distracted? Perhaps they get a phone call, or leave for lunch. Meanwhile no one else can edit that record.

Fortunately this can be resolved by enabling Panorama's record lock timeout. You may recall that when we made [Real Estate Listing](#) sharable we set this timeout to 45 seconds.

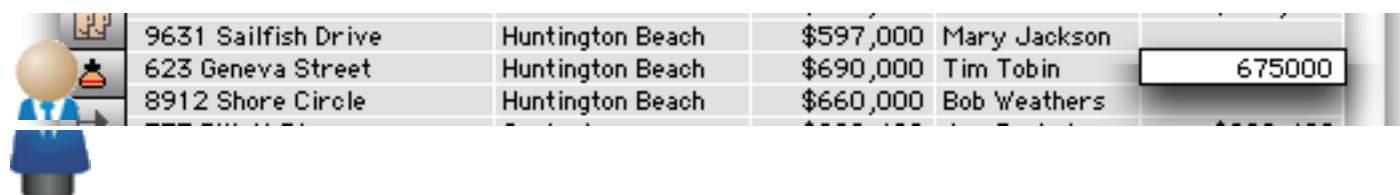
Basic Database Sharing Options

This option applies only if one of the database sharing options are checked above:

Auto unlock after seconds of inactivity

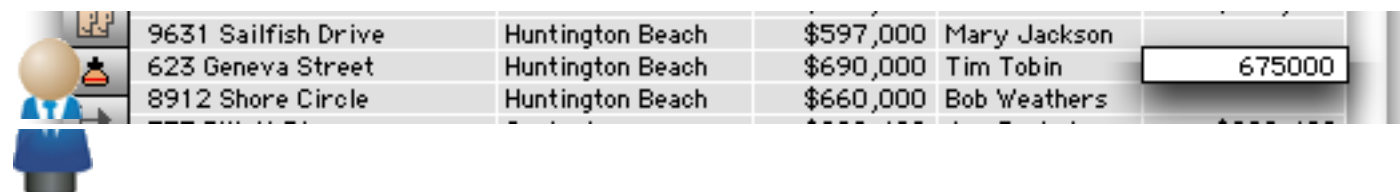
Panorama automatically locks the current record when you begin editing it, and unlocks when you move to another record. If this option is checked, Panorama will also unlock the current record after the specified amount of time has passed with no activity. In other words, if someone starts editing a record and then walks away, Panorama will automatically unlock the record after the specified time.

Let's revisit the record locking example from the previous section. Suppose Bob starts editing the record and then walks away.



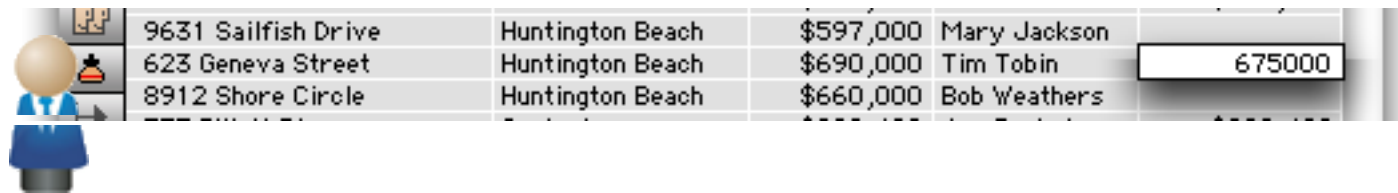
| | | | | |
|---------------------|------------------|-----------|--------------|--------|
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | 675000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |

After 15 seconds, nothing has changed.



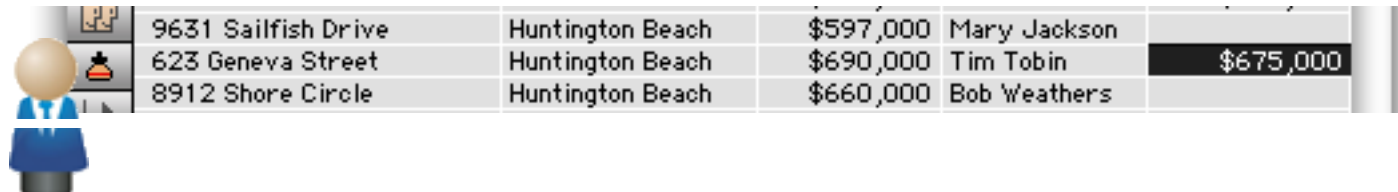
| | | | | |
|---------------------|------------------|-----------|--------------|--------|
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | 675000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |

Still no change after 30 seconds.



| | | | | |
|---------------------|------------------|-----------|--------------|--------|
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | 675000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |

After 45 seconds, however, Panorama automatically completes Bob's editing and unlocks the record.

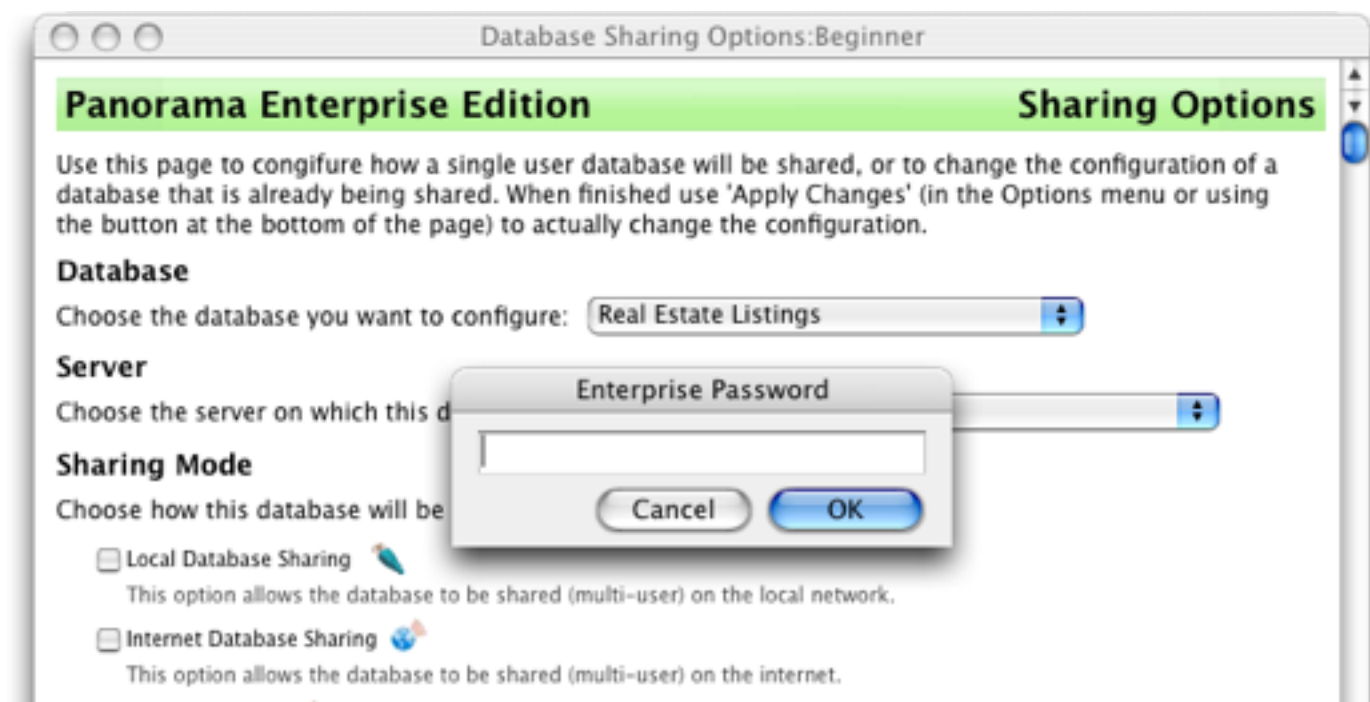


| | | | | |
|---------------------|------------------|-----------|--------------|-----------|
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Tim Tobin | \$675,000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |

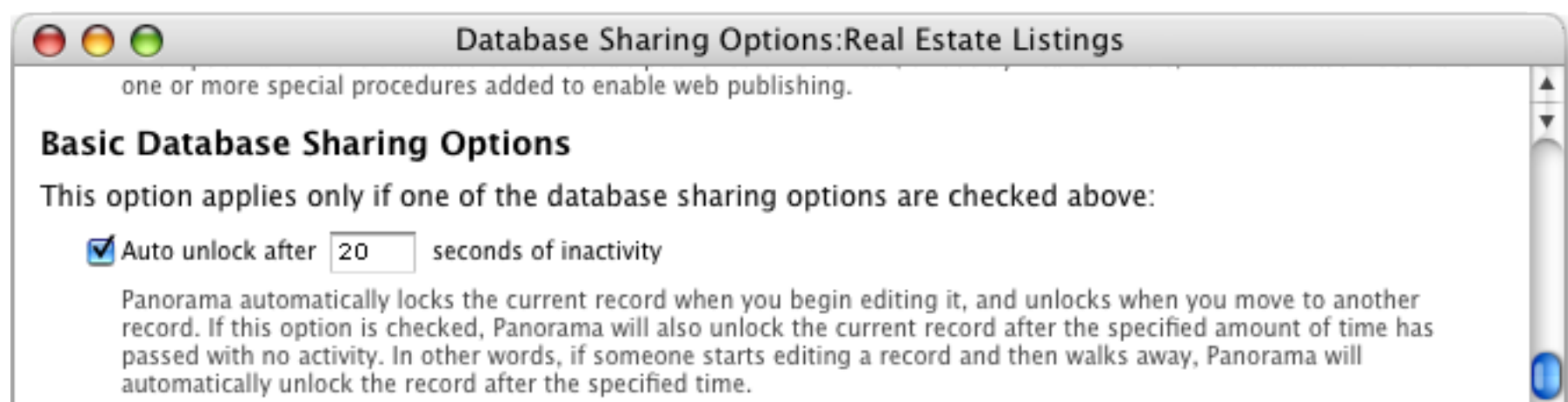
Now Rudy (or anyone else) can edit the record.

Changing the Record Lock Timeout

To change the timeout value, start with the database you want to change on top. Then open the **Database Sharing Options** wizard. The wizard will ask you to enter the server password.

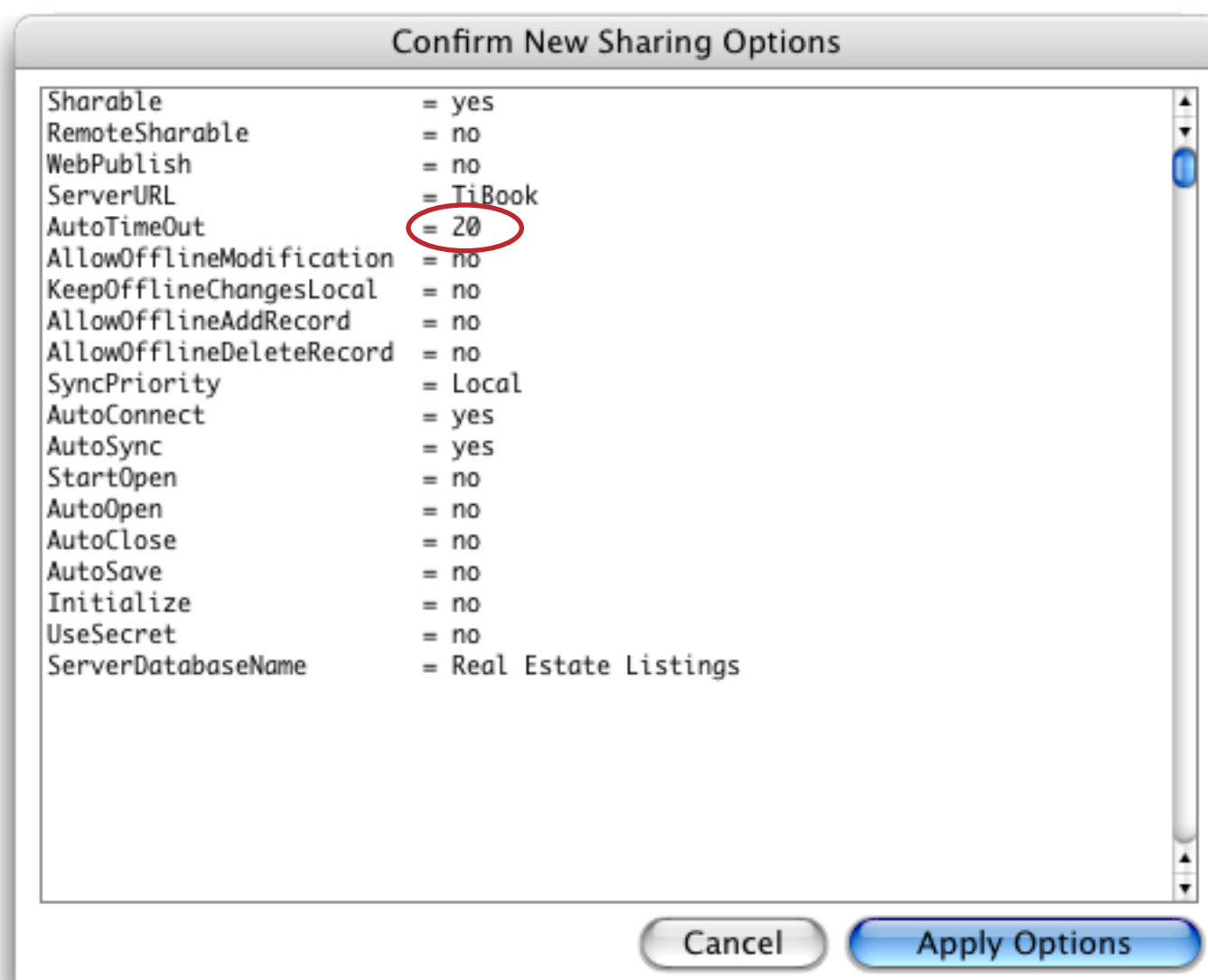


After you enter the password, it will display the current settings. Scroll down to the **Basic Database Sharing Options** and change the auto unlock timeout to the desired setting.

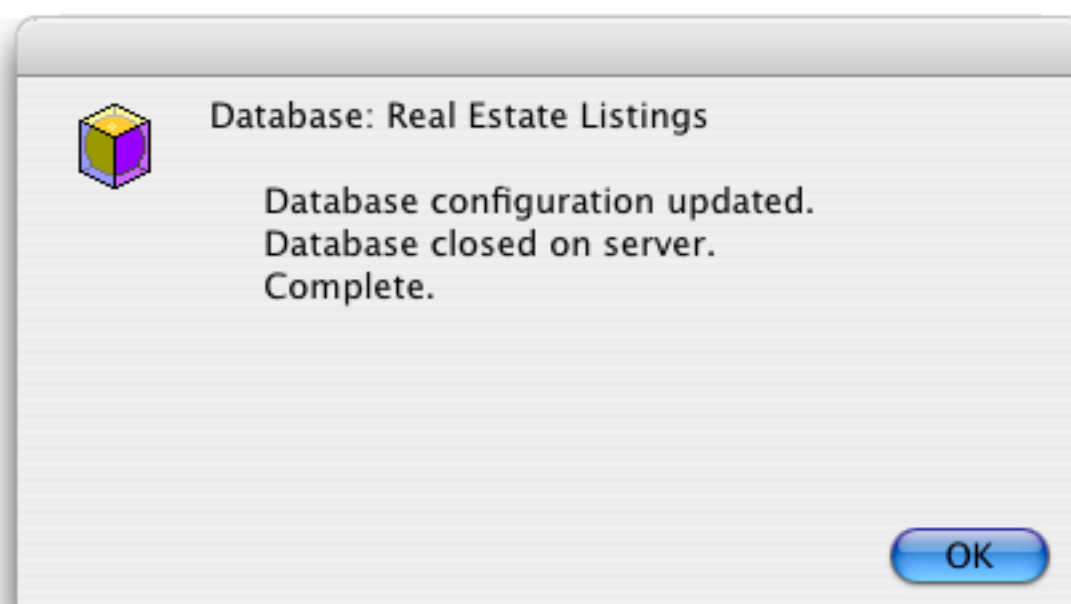


When editing, the timer is reset every time you hit a key or click the mouse. So setting this value to 20 means that the record will automatically unlock if you don't hit a key or click the mouse for 20 seconds.

Choose **Apply Changes** from the **Form** menu or press the **Apply Changes** button at the bottom of the form to submit the change. The wizard will ask you to confirm the new value.



After you press **Apply Options** the wizard will ask you for the server password again and then confirm that the change has been made.



Note: When the timeout is modified *after* the database has already been changed to a sharable database the change applies only to the current computer. You must repeat it on the other computers on the network if you want them to use the same settings. (Conversely, this allows you to use different timeouts on different computers.)

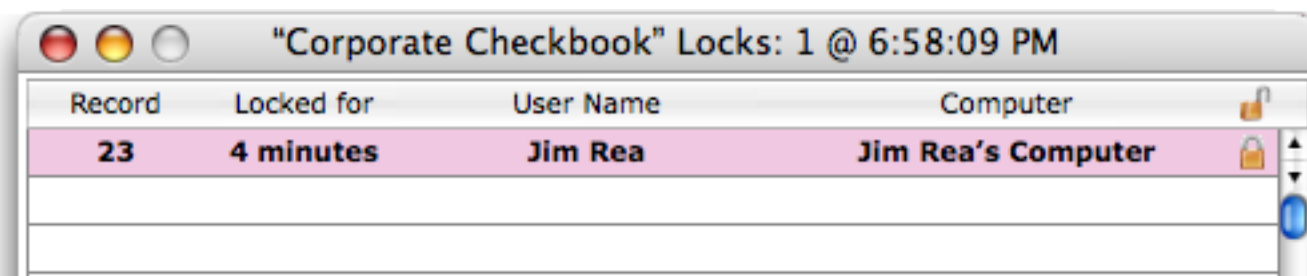
Record Lock Timeout (Server)

In addition to the record lock timeout described above, which happens on the client computer, the server administrator can also set up a timeout on the server itself (see “[Automatically Unlock Records](#)” on page 92). In normal operation the server timeout is unnecessary if you use the client timeout feature. However, if a client becomes disconnected from the network (for example if a laptop is unplugged from the network or loses its WiFi signal, or if there is a power outage that affects the client but not the server) the server timeout will ensure that any records locked by that client will not remain stuck indefinitely, inaccessible to other users. (Note: If a client locks a record, then is disconnected or crashes, then logs back on, any record they had locked will automatically become unlocked when they log back on, which also helps to prevent “orphan” locked records.)

Locked Records Wizard

Usually the server handles record locking and unlocking automatically, with no intervention from the users or the server administrator. In certain situations, however, you may want to manually monitor record locking operation, or even manually intervene to unlock a record that appears to be orphaned (you can almost completely eliminate orphan record locks with judicious use of client and server record lock timeouts).

The Locked Records wizard can be used to examine the locked records for any open database. With a shared database open, simply choose Locked Records from the Sharing submenu of the Wizard menu.

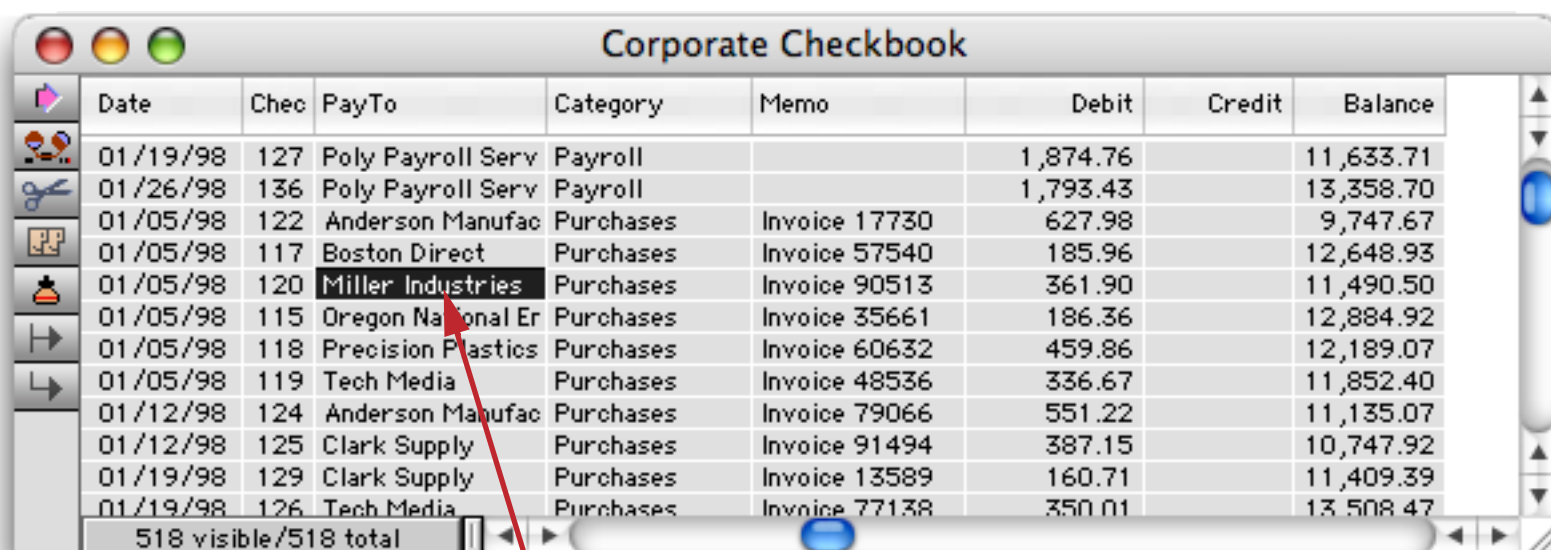


| Record | Locked for | User Name | Computer |
|--------|------------|-----------|--------------------|
| 23 | 4 minutes | Jim Rea | Jim Rea's Computer |

As you can see, the wizard lists each of the locked records in this database. For each record, the table lists the record ID, the amount of time the record has been locked, the user name, and the name of the computer that user is using.

Finding a Locked Record

The window above indicates that record 23 is locked. But which record is 23? To find out, simply double click anywhere on the line (except for the lock icon). Panorama will bring up the original database and find the locked record.

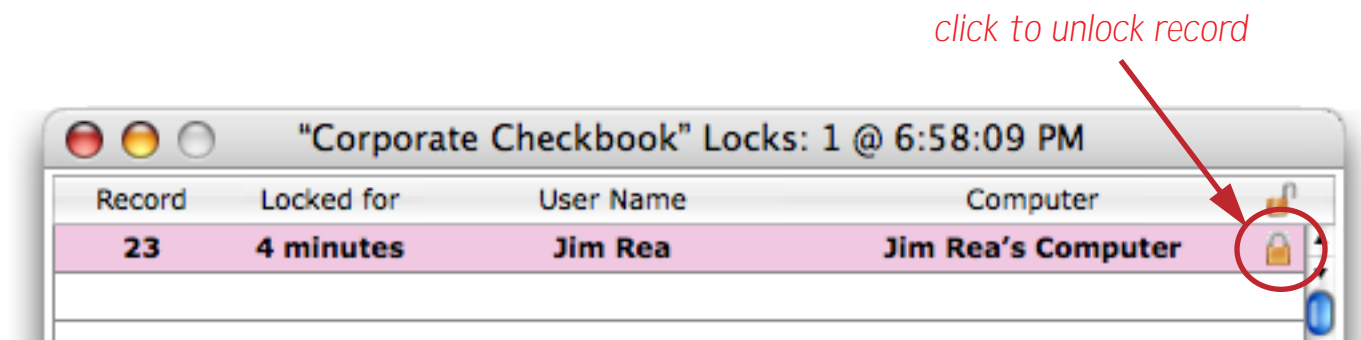


| Date | Chec | PayTo | Category | Memo | Debit | Credit | Balance |
|----------|------|--------------------|-----------|---------------|----------|--------|-----------|
| 01/19/98 | 127 | Poly Payroll Serv | Payroll | | 1,874.76 | | 11,633.71 |
| 01/26/98 | 136 | Poly Payroll Serv | Payroll | | 1,793.43 | | 13,358.70 |
| 01/05/98 | 122 | Anderson Manufac | Purchases | Invoice 17730 | 627.98 | | 9,747.67 |
| 01/05/98 | 117 | Boston Direct | Purchases | Invoice 57540 | 185.96 | | 12,648.93 |
| 01/05/98 | 120 | Miller Industries | Purchases | Invoice 90513 | 361.90 | | 11,490.50 |
| 01/05/98 | 115 | Oregon National Er | Purchases | Invoice 35661 | 186.36 | | 12,884.92 |
| 01/05/98 | 118 | Precision Plastics | Purchases | Invoice 60632 | 459.86 | | 12,189.07 |
| 01/05/98 | 119 | Tech Media | Purchases | Invoice 48536 | 336.67 | | 11,852.40 |
| 01/12/98 | 124 | Anderson Manufac | Purchases | Invoice 79066 | 551.22 | | 11,135.07 |
| 01/12/98 | 125 | Clark Supply | Purchases | Invoice 91494 | 387.15 | | 10,747.92 |
| 01/19/98 | 129 | Clark Supply | Purchases | Invoice 13589 | 160.71 | | 11,409.39 |
| 01/19/98 | 126 | Tech Media | Purchases | Invoice 77138 | 350.01 | | 13,508.47 |

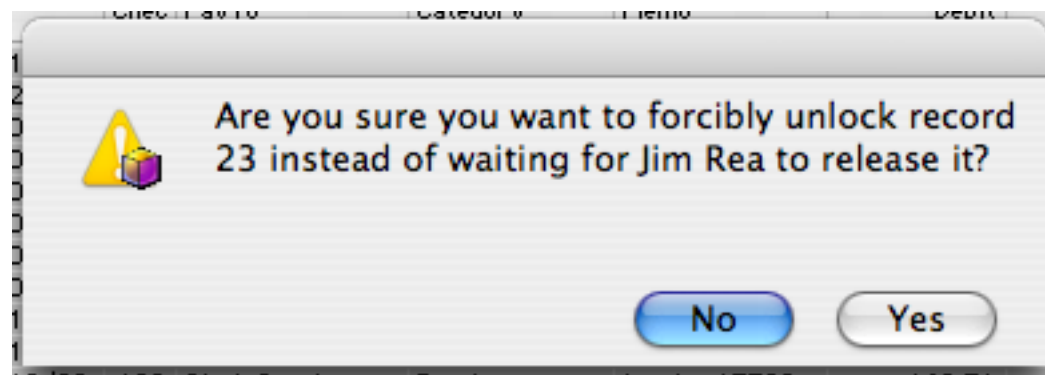
record 23

Manually Unlocking a Record

To manually unlock a record, click on the lock icon for that record.



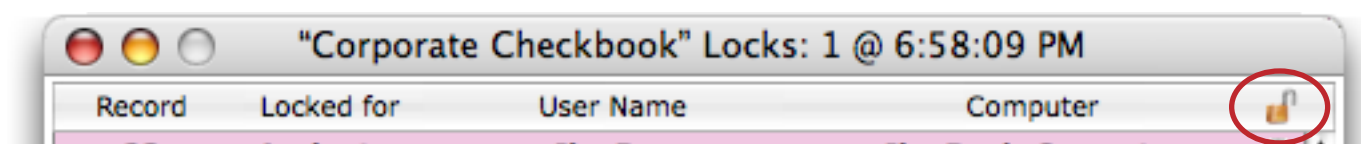
The wizard will ask you to confirm that you really want to unlock the record.



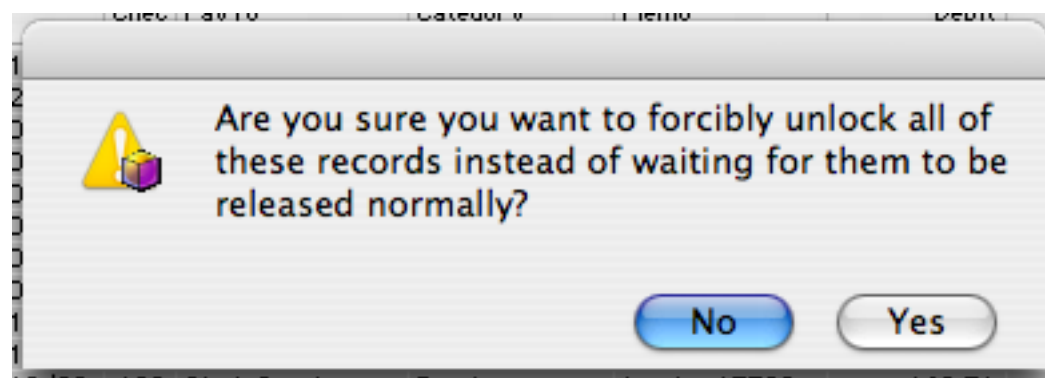
Think carefully before pressing the **Yes** button. Once you do so, the person who originally locked the record will no longer be able to complete the editing they were working on. If they are still connected to the network they probably won't be very pleased with you!

Manually Unlocking All Records in a Database

To manually unlock all of the records in a database, click on the open lock in the top right corner of the wizard.



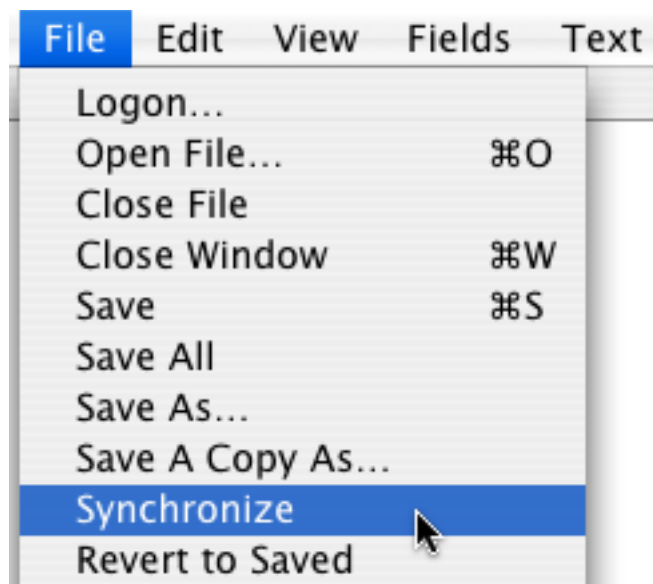
You'll be asked to confirm that you really want to do this.



Make sure you really know what you are doing, or you could create quite a few unhappy co-workers by pressing the **Yes** button.

Synchronization

Since Panorama database sharing uses duplicate databases spread out all over the network, synchronization is the key to keeping all the data in order. Synchronization happens automatically whenever a shared database is opened, and can also be done “on command” at any time from the **File** menu.



Synchronizing updates the local copy of the database with the server. When you synchronize a database, you may see any of these changes happen on the local database:

- 1) Existing records may change.
- 2) New records may be added.
- 3) Existing records may be deleted.
- 4) The order (top to bottom) of records may change, and summary records will be removed.
- 5) If a subset of records is visible, all records will now be selected (as if you had performed a **Select All** command).

These changes all happen simultaneously so you won't see individual records appearing, changing, and deleting.

Synchronization and Record Order

The last point in the list above merits more attention. If you have been using the single user version of Panorama, you are accustomed to the record order staying the same each time you open the file. When working with multi-user databases, that isn't the case. Each time the database is synchronized, the order of the data may (and probably will) change.

For example, suppose you have used the **Sort Down** command to rank the listings database by asking price, so that the highest price properties appear at the top.

| Address | City | Asking | Agent | Offer |
|---------------------|------------------|-------------|---------------|-----------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 6541 Starshine Lane | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 9102 Pioneer Way | Huntington Beach | \$1,050,000 | Stan Bryant | |
| 18540 Santa Andrea | Fountain Valley | \$1,050,000 | Ann Rachels | |
| 3121 Linda Way | Santa Ana | \$1,050,000 | Stan Bryant | |
| 9203 Columbine | Fountain Valley | \$1,049,400 | Rose Richards | |
| 17122 Fraser Lane | Huntington Beach | \$1,019,400 | Stan Bryant | |
| 9571 Dumbreck Ave. | Huntington Beach | \$1,019,400 | Mary Jackson | |
| 2231 Almond Way | Orange | \$1,019,400 | Stan Bryant | \$999,012 |
| 15691 Foxhills St. | Westminister | \$1,019,400 | Bob Weathers | \$917,460 |
| 3309 Sisson Way | Santa Ana | \$1,019,400 | Stan Bryant | |
| 6 Robin Street | Irvine | \$1,017,000 | Mary Jackson | |
| 9168 El Azul Avenue | Fountain Valley | \$1,014,000 | Tim Tobin | |

When you synchronize your local copy of the database, the data is no longer sorted by asking price. If you need to see the updated data in sorted order, simply use the Sort command again (fortunately, Panorama's RAM based sorting is very fast).

| Address | City | Asking | Agent | Offer |
|--------------------------|------------------|-------------|--------------|-------------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,200 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502,740 |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 623 Geneva Street | Huntington Beach | \$690,000 | Mary Jackson | \$675,000 |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,400 |
| 12241 Fallingleaf | Westminister | \$552,000 | Mary Jackson | |
| 11112 Jerry Lane | Garden Grove | \$564,000 | Bob Weathers | |
| 7612 California | Westminister | \$570,000 | Ann Rachels | \$570,000 |
| 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | |
| 623 Geneva Avenue | Huntington Beach | \$690,000 | Bob Weathers | |
| 7836 Connie Lane | Huntington Beach | \$912,000 | Bob Weathers | |

The same principle applies to grouping and summary records. The following four steps have been used to rank the cities in this database by average home prices.

- 1) Click on City field and **Group Up**.
- 2) Click on Asking field and **Average**.
- 3) Using the **Outline Level** command, collapse to see only cities.
- 4) Rank cities from most to least expensive using **Sort Down**.

(See Chapter 10 of the Panorama Handbook for more detail on these steps.)

| Address | City | Asking | Agent | 01 |
|---------|--------------------------|--------------------|-------|----|
| | Newport Beach | \$5,551,850 | | |
| | Corona Del Mar | \$4,770,000 | | |
| | Tustin | \$2,754,000 | | |
| | Surfside | \$2,418,000 | | |
| | Laguna Hills | \$2,010,000 | | |
| | Sunset Beach | \$1,850,000 | | |
| | Huntington Beach | \$1,697,390 | | |
| | Laguna Beach | \$1,616,450 | | |
| | Seal Beach | \$1,159,360 | | |
| | Orange | \$1,140,650 | | |
| | Irvine | \$1,070,057 | | |
| | Pacific Palisades | \$1,068,000 | | |
| | Fountain Valley | \$1,051,536 | | |

26 visible/350 total

Choosing **Synchronize** from the **File** menu updates the local database with all of the changes made by other users. It also removes all of the summary records and goes back to the original data record display.

| Address | City | Asking | Agent | 01 |
|--------------------------|-------------------------|-------------|--------------|-----------|
| 33 Elm Street | Huntington Beach | \$1,053,000 | Stan Bryant | |
| 525 Sunset Lane | Irvine | \$1,260,000 | Ann Rachels | \$1,159,2 |
| 1101 S. Meeker | Garden Grove | \$513,000 | Bob Weathers | \$502, |
| 9631 Sailfish Drive | Huntington Beach | \$597,000 | Mary Jackson | |
| 8912 Shore Circle | Huntington Beach | \$660,000 | Bob Weathers | |
| 735 Elliott Place | Santa Ana | \$989,400 | Ann Rachels | \$989,4 |
| 12241 Fallingleaf | Westminster | \$552,000 | Mary Jackson | |
| 203 N. Walnuthaven Drive | Costa Mesa | \$540,000 | Stan Bryant | |
| 623 Geneva Avenue | Huntington Beach | \$690,000 | Bob Weathers | |
| 7836 Connie Lane | Huntington Beach | \$912,000 | Bob Weathers | |

If you are going to frequently produce a summary or report you can automate the process with a procedure that starts by synchronizing before it does any data processing. This ensures that your summary or report will contain the latest, most up-to-date information.

| Icon | Command |
|------|----------------|
| ▶▶ | synchronize |
| ▶ | field City |
| ▶ | groupup |
| ✂ | field Asking |
| ▶ | average |
| ☰ | outlinelevel 2 |
| ⬇ | sortdown |
| 🔍 | firstrecord |

Regular Synchronization vs. Force Synchronization

When Panorama synchronizes with the server, it transfers only the records that have actually changed. These updated records are merged with the existing unchanged records in the local database. However if you hold down the **Option** key when you choose the **Synchronization** command then Panorama will transfer the entire database from the server to the local database, including both changed and unchanged records. This is called *Force Synchronization*. Theoretically you should never need to do a Force Synchronize, but the option is there if for any reason you think a database is not synchronizing properly. You can also perform this operation in a procedure with this statement.

```
forcesynchronize
```

Adding New Records

You can add new records to a shared database just as you would in single user mode. Just add the record and start entering data into it. The new record will appear in other user's copies of the database the next time they synchronize.

Note: When working with a shared database, you can use the **Insert Record** tool (or the **Return** key in a data sheet) to insert a record in the middle of a database. However, if you do this, keep in mind that the order of the records will change every time you synchronize the database. A record inserted in the middle will not stay in the same position for long in a shared database. If you need a record to appear in a specific spot you should include a field with a value that will position the record in the proper spot when you sort the record.

Deleting Records

You can delete records from a shared database just as you would in a single user database. The deleted record will disappear from other user's copies of the database the next time they synchronize. If another user attempts to edit a record that has already been deleted by another user, Panorama will not let them edit the record. Instead, an alert will appear warning them that this record has been deleted.

Panorama has two commands that can delete large numbers of records at a time — the **Delete All** command (in the **Edit** menu) and the **Remove Unselected** command (in the **Search** menu). Both of these commands are disabled when using a shared database. The only way to delete records from a shared database is one record at a time.

Working With Summary Records

Unlike ordinary data records, summary records are not shared with other users or stored in the server copy of the database. Instead, summary records are always kept individually on each local database. This allows each individual user to group his or her local database in his or her own way. Because summary records are not shared, they are also not locked. Double clicking on a summary record does not lock the summary record (because no one else could possibly modify your summary record). Likewise, the **Group**, **Total**, **Average**, and other Math menu commands are not affected by record locking.

In general, summary records work pretty much the way they do in single user databases (described in Chapter 10 of the Panorama Handbook). You start by using the **Group** command to organize the database into sections (you may want to synchronize the database first to make sure you are analyzing the latest data, see "[Synchronization](#)" on page 144). Then you can use the commands in the top section of the Math menu (**Total**, **Count**, **Average**, etc.) to calculate subtotals, averages, etc. If desired, you can use the **Outline Level** command to collapse the summaries and hide the original detail. When you're done with the summary records you can use the **Remove Summaries** command to remove them (synchronizing will also remove the summary records). All of this happens on your local computer — no one else on the network can see your summaries, and different users on the network can do the same or different summaries simultaneously.

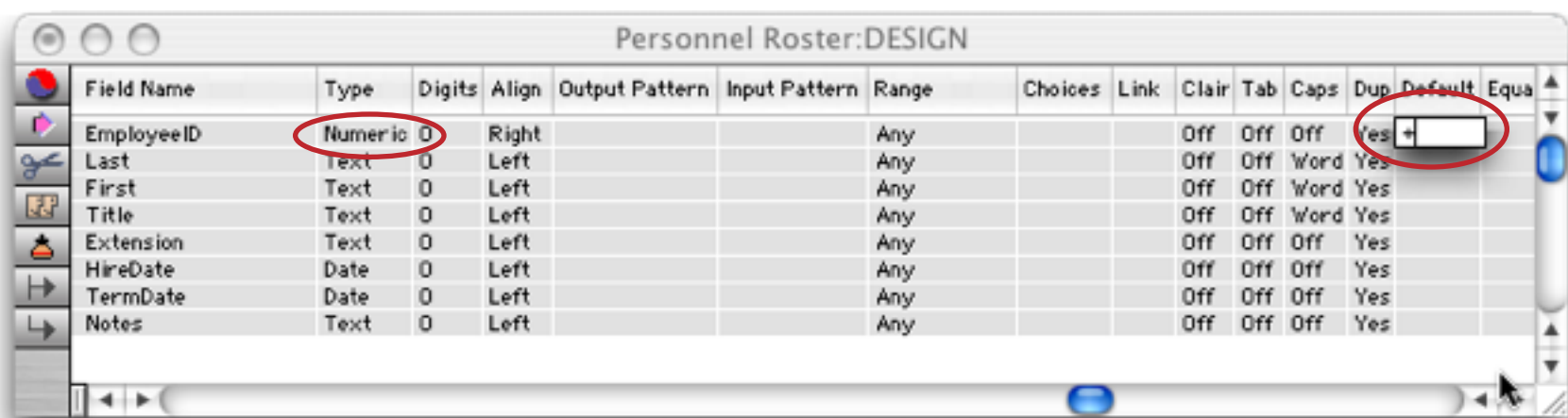
Toggle Summary Records (not!)

The single user version of Panorama allows you to convert an ordinary data record into a summary record or a summary record into a data record. This is done using the **Toggle Summary Level** command in the Sort menu. This command does not work with shared databases, and an error alert will appear if you try to use it. (Why doesn't this command work? Since summary records are not shared, converting a data record into a summary record would require deleting it from the server and all other users. We hope you agree that it wouldn't make any sense to do this.)

Automatic Record Numbering

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.

Setting up automatic record numbering must be done before you convert the database from single user to sharable. First set up the field that will contain the automatic record number. This field 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.



For each database, the server has a counter that keeps track of the next record number. Every time a new record is created, the local copy of Panorama will ask the server what the next record number should be (the server then automatically increments the counter for next time). Even if the record is later deleted, the number will never be re-used (unless you reset the counter manually as described below).

To illustrate automatic numbering works let's assume that Bob Blue and Rudy Red have been hired to do some data entry on this personnel roster. The database starts looking like this:

| EmployeeID | Last | First | Title | Extension | HireDate | TermDate |
|------------|---------|--------|------------|-----------|----------|----------|
| 1 | Booth | Sara | Owner | 11 | 08/01/02 | |
| 2 | Parker | Mark | Agent | | 10/15/02 | 04/15/04 |
| 3 | Diaz | Rachel | Secretary | 10 | 11/01/02 | |
| 4 | Moody | Dianne | Agent | 17 | 02/15/03 | |
| 5 | Hope | Alan | Agent | 15 | 07/25/03 | |
| 6 | Chase | Steven | Agent | | 09/01/03 | 07/08/05 |
| 7 | Metzger | Alan | Bookkeeper | 12 | 02/10/05 | |

Meanwhile Bob has finished entering the first record and adds another. This is employee number 10.



The screenshot shows a window titled "Personnel Roster" with a table of employee data. The table has columns for EmployeeID, Last, First, Title, Extension, HireDate, and TermDate. The data is as follows:

| EmployeeID | Last | First | Title | Extension | HireDate | TermDate |
|------------|---------|--------|------------|-----------|----------|----------|
| 1 | Booth | Sara | Owner | 11 | 08/01/02 | |
| 2 | Parker | Mark | Agent | | 10/15/02 | 04/15/04 |
| 3 | Diaz | Rachel | Secretary | 10 | 11/01/02 | |
| 4 | Moody | Dianne | Agent | 17 | 02/15/03 | |
| 5 | Hope | Alan | Agent | 15 | 07/25/03 | |
| 6 | Chase | Steven | Agent | | 09/01/03 | 07/08/05 |
| 7 | Metzger | Alan | Bookkeeper | 12 | 02/10/05 | |
| 8 | Knox | Gary | Agent | 24 | 06/12/05 | |
| 10 | | | | | | |

The status bar at the bottom indicates "9 visible/9 total". A small icon of a person in a blue suit is visible in the bottom left corner of the window.

As they are adding new records, Bob and Rudy cannot see the records added by each other (or anyone else). If they want to see all of the new records, they can simply choose **Synchronize** from the File menu, and the records entered by other users will appear.



The screenshot shows the same "Personnel Roster" window after synchronization. The table now includes 10 records:

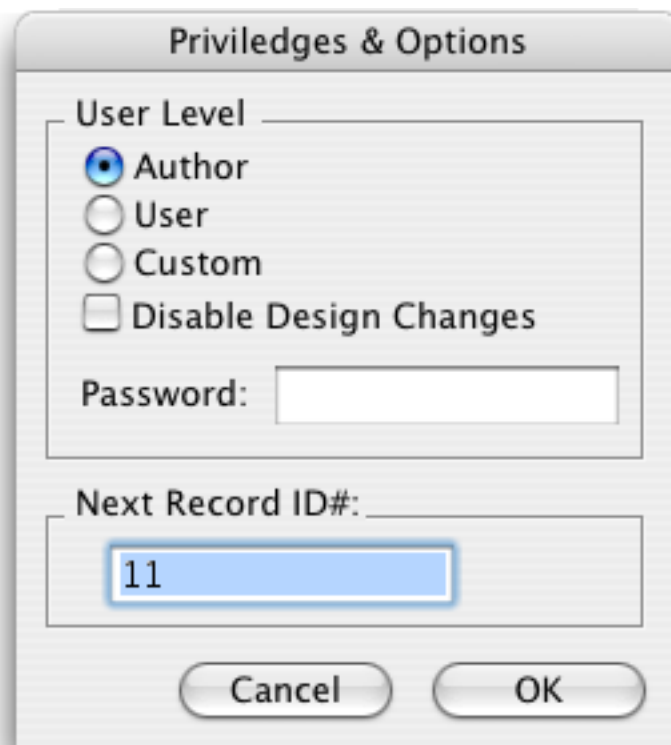
| EmployeeID | Last | First | Title | Extension | HireDate | TermDate |
|------------|---------|--------|------------|-----------|----------|----------|
| 1 | Booth | Sara | Owner | 11 | 08/01/02 | |
| 2 | Parker | Mark | Agent | | 10/15/02 | 04/15/04 |
| 3 | Diaz | Rachel | Secretary | 10 | 11/01/02 | |
| 4 | Moody | Dianne | Agent | 17 | 02/15/03 | |
| 5 | Hope | Alan | Agent | 15 | 07/25/03 | |
| 6 | Chase | Steven | Agent | | 09/01/03 | 07/08/05 |
| 7 | Metzger | Alan | Bookkeeper | 12 | 02/10/05 | |
| 8 | Knox | Gary | Agent | 24 | 06/12/05 | |
| 9 | Wu | Lynn | Agent | 25 | 06/14/05 | |
| 10 | Ramond | Patti | Agent | 22 | 08/27/05 | |

The status bar at the bottom now indicates "10 visible/10 total".

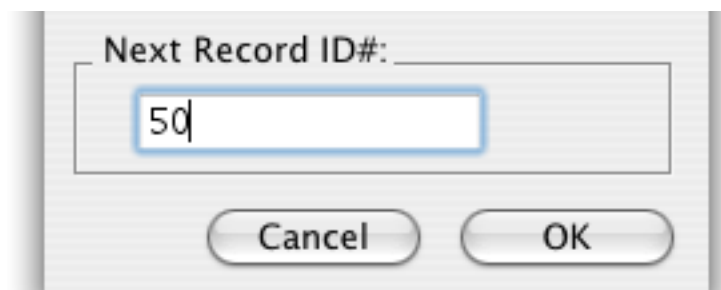
If you want Panorama to automatically synchronize each time a new record is added, simply set up a **.NewRecord** procedure containing a **synchronize** statement. This procedure will automatically be triggered whenever a new record is added. See Chapter 24 of the Panorama Handbook for more information about **.NewRecord** procedures.

Manually Changing the Record Number Counter

The automatic record number normally increments by one each time a new record is added, but you can manually change the record number counter at any time using the **Privileges** dialog. 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. (The User Level portion of this dialog is discussed in Chapter 3 of the Panorama Handbook.)



To change the next record ID # simply type in a new number and press Ok. For example, suppose you want the next employee number to be 50. Simply type in 50 and press Ok.



The next record added will be 50.

| | | | | | | |
|----|---------|-------|------------|----|----------|--|
| 7 | Metzger | Alan | Bookkeeper | 12 | 02/10/05 | |
| 8 | Knox | Gary | Agent | 24 | 06/12/05 | |
| 9 | Wu | Lynn | Agent | 25 | 06/14/05 | |
| 10 | Ramond | Patti | Agent | 22 | 08/27/05 | |
| 50 | | | | | | |

Keep in mind that this is the next record added anywhere on the network. If someone else adds a record before you do you'll see 51, 52, or a higher number on your computer.

Accessing the Next Record Number in a Procedure

It is also possible to access and modify this ID number in a procedure with the `GetAutoNumber` and `SetAutoNumber` statements. 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)+"."
```

Keep in mind that since this is a multi-user system someone else may use that number before you get a chance to, even if the `addrrecord` statement is the next statement after the `GetAutoNumber` statement.

To change the next automatic record number use the `SetAutoNumber` statement. This one line procedure uses the `SetAutoNumber` statement to reset the record ID number to 1000.

```
SetAutoNumber 1000
```

This is exactly the same as setting the number with the **Privileges** dialog.

Saving the Database

Like most programs that work with files (but unlike most databases) Panorama has a **Save** command in the **File** menu. However, when working with a shared database this command works a bit differently than it does with most programs.

The **Save** command saves a local copy of the database on the local hard disk. However, Panorama also keeps a copy of the data on the server. Even if you never, ever use the **Save** command, Panorama will still keep a permanent copy of the database on the server. Theoretically you could never use the **Save** command at all and all of your data would still be safe. However, you should still use the **Save** command periodically because doing so will make synchronization faster the next time you open the database. If you've saved recent changes on the local hard disk Panorama won't have to update those records the next time it synchronizes the database, so synchronization will happen faster. Over time this performance increase could become substantial.

There is another reason to use the **Save** command — this is the only way to save changes you've made to graphics (forms) and procedures. Only data is saved on the server. So if you've made changes to graphics or to procedures be sure to use the **Save** command (just as you normally would with any single user Panorama database).

Revert to Saved

The **Revert to Saved** command reloads the local copy of the database from the local hard drive. With a shared database this doesn't revert the data, because it does an immediate synchronization which brings the latest data back from the server. However, it can still be useful if you decide you want to discard unsaved changes you've made to graphics (forms) or procedures.

When does the Server Save?

When working in single user mode Panorama will save the database from RAM to disk on your command. The server will save it's RAM copy of a database when any one of these three actions occurs.

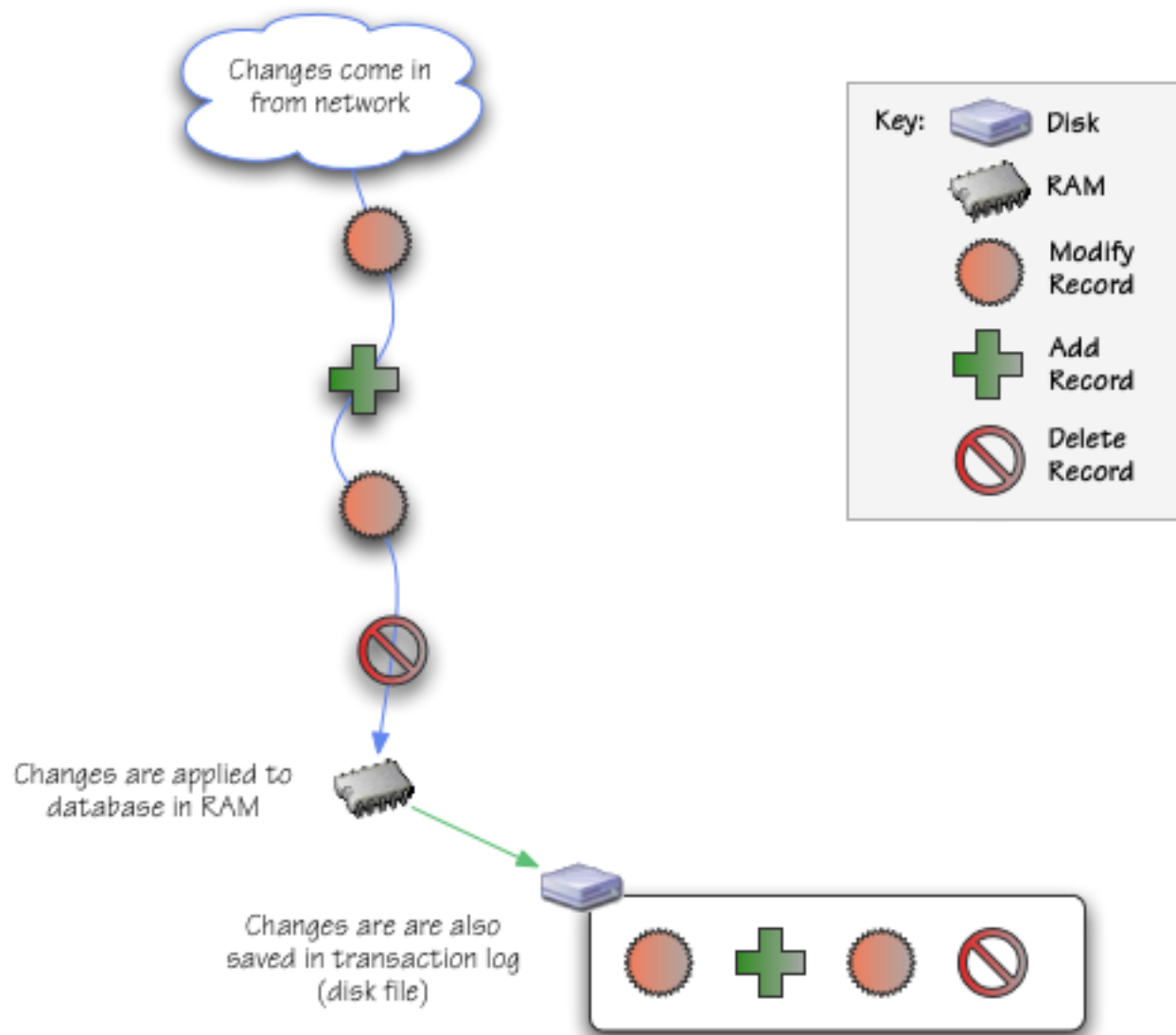
- 1) Any user on the network saves their local copy of the database.
- 2) Any user on the network closes their local copy of the database.
- 3) Any multi-record operation is performed (fill, append, etc.).

It's possible under these rules that the server copy of a database may not get saved for a long period of time, even hours at a time. However, as you'll see in the next section, your changes can never be lost even if server operation is interrupted for any reason (power failure, system crash, etc.)

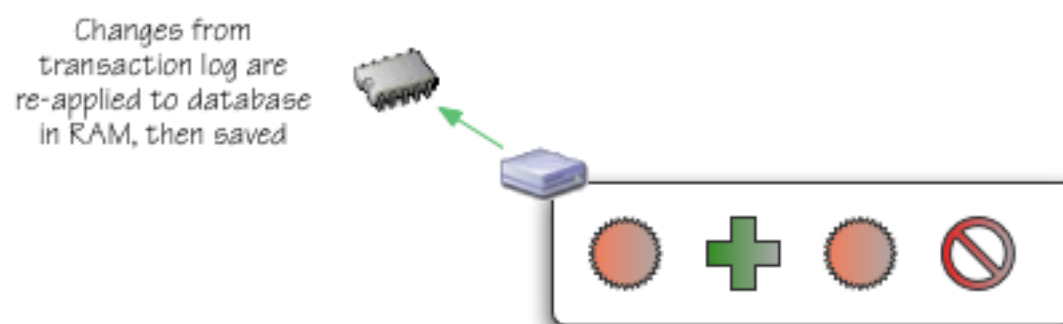
Handling Interruptions in Server Operation (Crash Recovery)

This section describes how the server saves data, and how it recovers after any kind of interruption in server operation (power failure, system crash, etc.) If you're not interested in the details, the short version is that your data is safe no matter what (the only exception is data you've just modified in the current record, before the record is unlocked).

Whenever you add, delete, or modify a record, Panorama updates the database on the server as well as the local database. But the server database is RAM based, so the data could be lost if there is any interruption in server operation. Therefore, Panorama also saves changes to a special transaction log file. Each entry in this log file contains the operation being performed (add/modify/delete), the record id number, and for modifications, the new data. Because this is a simple sequential file writing to it is very fast (it has none of the overhead associated with the type of indexed file typically used by disk based database programs).



When the server copy of the database is saved to disk the transaction log file is deleted, since it is no longer needed. But if the server is interrupted for any reason before getting a chance to save, the transaction log remains on the disk. The next time Panorama Server starts up again it checks to see if there are any transaction log files. If there are, it immediately re-applies the changes saved in the transaction log and then saves the database. Server operation can then continue normally with no data loss, as if nothing had happened. (When the server needs to use the transaction log to recover changes it will record this in both the sharing and sharing error logs (see "[Monitoring Server Logs](#)" on page 82), and it will also send you an e-mail if you have configured it to do so (see "[Configuring the Notification Wizard](#)" on page 85)).



Programming Shared Databases

So far in this chapter, we've concentrated primarily on manual operation of shared databases. However, shared databases fully support Panorama's programming language. A Panorama procedure can lock and unlock records, synchronize, create and modify server variables, connect to and disconnect from the server, rapidly modify selected records, and more. (Note: Unless otherwise specified, the statements described in the following sections are completely ignored if the current database is not a shared database.)

Record Locking

Procedures often modify the data in a database. When a procedure modifies data in a shared database record locking is in effect just as it is when data is modified manually.

Implicit Record Locking

As a procedure runs, Panorama will automatically lock and unlock records as necessary. This is called **implicit record locking**. Implicit record locking follows two simple rules — 1) The record is locked whenever any field in the record is about to be modified, and 2) The record remains locked until the procedure moves to another record, moves to another database, saves, or explicitly unlocks the record (see next section).

The most common method for a procedure to modify a record is with an equation. For example, consider the equation below which adjusts the quantity level in an inventory database. This example assumes that **Qty** is a numeric field in the current sharable database, and **SaleQty** is a field or variable that contains a numeric value.

```
Qty=Qty-SaleQty
```

When Panorama is running a procedure and encounters an equation that modifies a field in a sharable database it first attempts to lock the current record (if it is not already locked). To do this Panorama contacts the server to see if anyone else has locked this record. If not, the server responds that everything is ok and updates the record that is about to be locked (see "[Editing Data and Record Locking](#)" on page 136). It then calculates the formula and modifies the **Qty** field. Since the data in the local copy of the record is updated as part of the record locking mechanism, the formula result is guaranteed to be based on the most recent data. (For example suppose the quantity starts at 100. Then Bob subtracts 11 and Kelly subtracts 14. The final quantity will always be 75, not 89 or 86.)

But what if the record is already locked by another user on the network? In that case the procedure will simply wait for the other user to finish and unlock the record. The computer running the procedure will be frozen until the other user finishes (actually only Panorama is frozen, other applications work fine). When the record is available, Panorama will lock the record on this machine and continue as described in the previous paragraph.

Before this equation begins running, Panorama automatically locks the current record. That way another user cannot change the Qty value while the equation is being calculated, possibly causing a calculation error. (If another user (or another procedure equation) has already locked this record, the procedure will stop and wait for this user to finish and unlock the record. The user running the procedure will be frozen until the other user finishes.) When the calculation is complete, the procedure stores the new Qty value and unlocks the record. (If you don't like the part about Panorama becoming temporarily "frozen", continue on to the next section for a solution.)

Explicitly Locking/Unlocking Records in a Procedure

In the last section, you saw how a Panorama will implicitly lock and unlock a record during a procedure. However, sometimes the programmer may want to control exactly when and how records are locked and unlocked. The programmer may also want to control what happens when a record is already locked by another user. To accomplish these goals, the programmer can use the **LockRecord**, **LockOrStop** and **UnlockRecord** statements.

The **LockRecord** statement attempts to lock the current record. If the current record is not already locked by someone else on the network, the record is locked and the procedure continues. If the current record is already locked, the procedure will wait until the record is available. While it is waiting the statement displays a dialog telling the person running the procedure to wait. If the person doesn't want to wait, he or she can press **Command-Period**. If the user presses **Command-Period** the procedure stops and an error dialog appears. However, the programmer can intercept this error by putting an **if error** statement after the **LockRecord** statement. In that case when the user presses **Command-Period**, the procedure will continue running with the first statement after the if error statement.

```
local ReceivedQuantity
ReceivedQuantity=""
gettext "How many "+Item+" were received?",ReceivedQuantity
ReceivedQuantity=val(ReceivedQuantity)
lockrecord
if error
    message "These items could not be added to the database quantity. "+
        "Be sure to process them again later."
    rtn
endif
Qty=Qty+ReceivedQuantity
```

The **LockOrStop** statement attempts to lock the current record. If the current record is not being edited by someone else, the record is locked and the procedure continues. If the current record is already locked, the procedure stops immediately and an error dialog appears. However, the programmer can intercept this error by putting an **if error** statement after the **LockOrStop** statement.

The **UnlockRecord** statement unlocks the current record. If the record wasn't locked, the **UnlockRecord** statement does nothing. Remember, a locked record is also unlocked automatically whenever Panorama moves to another record in the current database or any other database.

Here is an example of a procedure that explicitly locks and unlocks the current record to get around the frozen user problem mentioned in the last section. In this example, the user is not frozen if another user has locked the record, they simply see an error message.

```
local ReceivedQuantity
ReceivedQuantity=""
gettext "How many "+Item+" were received?",ReceivedQuantity
ReceivedQuantity=val(ReceivedQuantity)
LockOrStop
if error
    message "Could not update inventory, try again later."
    stop
endif
Qty=Qty+ReceivedQuantity
UnlockRecord
```

What Records are Locked?

The **lockedrecordlist** statement returns the number of records locked in the current database, and also returns a list of those records (listed by record ID number). This statement has one parameter, the name of a field or variable to receive the list. The result will be formatted like this:

```
Ok: 2 records locked
378
864
```

This procedure displays the number of records locked in the current database.

```
local locks
lockedrecordlist locks
message firstline(locks)[4,-1]
```

Forcing the Server to Unlock All Records

The `forceunlockallrecords` statement will tell the server to unlock ALL of the records in the current database. You should only use this statement if a client has disconnected from the network or crashed, and no other clients have any locked records at the moment. If another client had locked any of these records, they will be unlocked behind their back. They will get an error message when they try to unlock the record, and their changes will not be saved. This makes most users very unhappy, so be very careful before using this command. Note: You can also manually unlock records with the **Locked Records** wizard, see also.

Forcing the Server to Unlock a Specific Record

The `forceunlockrecord` statement will tell the server to unlock the current record in the current database. You should only use this statement if a client has locked this record and disconnected from the network or crashed. Make sure that you know that it is the disconnected client that locked this record. If another client that is still running is the one who locked this record, they will be unlocked behind their back. They will get an error message when they try to unlock the record, and their changes will not be saved. This makes most users very unhappy, so be very careful before using this command. Note: You can also manually unlock records with the **Locked Records** wizard, see also.

Forcing the Server to Lock a Specific Record

The `forcelockrecord` statement tells the server to lock this record (the current record on the client) on the server, as if another client had locked the record. Once this is done, the only way to unlock the record is with the `forceunlockallrecords` or `forceunlockrecord` statement (or with the **Locked Records** wizard, see also). This statement can be used for debugging if you don't have two client computers available for testing.

Temporarily Disabling Record Locking (and Server Updates)

When using a shared database, Panorama normally updates the server with full record locking every time any change is made to the database. Sometimes, however, you may want to make temporary changes to a database simply for the purposes of data analysis. For example, perhaps you need to use the `formulafill` statement to prepare data for printing, but will no longer need the calculated data after the report is printed. If you don't want to keep these changes you can temporarily turn off record locking, essentially turning a shared database back into single user mode for a short while.

To temporarily disable record locking use the `serverupdate` statement.

```
serverupdate truefalse
```

This statement has one parameter: `truefalse`. This parameter indicates whether you want to disable or enable record locking and server updates. To disable updates and record locking, use "off", "no", or "false". To enable updates and record locking, use "on", "yes", or "true".

The example below turns off server updates, then uses the `formulafill` statement to calculate P/E ratios. Since server updates (and record locking) are turned off, the `formulafill` will be very fast. In this case we don't need to keep the calculated P/E ratios after we are finished printing, so it is acceptable to turn off the server update.

```
serverupdate "off"
field Ratio
formulafill Price/Earnings
print dialog
serverupdate "on"
forcesynchronize /* restore original data from server */
```

The last line in this example may not be necessary. If you don't ever use this procedure for permanent changes to the database for permanent data, you can leave this line out.

You can only turn server updates off for a short time. We recommend that you explicitly turn them back on as soon as possible, but if you don't, they will automatically turn back on when the procedure finishes, or if the procedure switches to another database (via the `window` or `openfile` statements).

The `info("serverupdate")` Function

The `info("serverupdate")` can be used to check if server updates are currently enabled. The function returns the true if the server update option is currently turned on (this is the default). The example below shows how this function can be used. The procedure turns off server updates and performs a formula fill. It then turns server updates back on, but only if they were already on before this procedure began.

```
local sup
sup=info("serverupdate")
serverupdate "off"
field Z
formulafill A+B
if sup
  serverupdate "on"
endif
```

If `serverupdate` was off before this procedure was called, it will remain off.

Synchronizing

To synchronize the current database with the server use the `synchronize` statement.

```
synchronize
```

This is exactly the same as choosing **Synchronize** from the **File** menu (see “[Synchronization](#)” on page 144).

Force Synchronization

When Panorama synchronizes with the server, it transfers only the records that have actually changed. These updated records are merged with the existing unchanged records in the local database. A more drastic way to synchronize is called *Force Synchronization* which transfers the entire database from the server to the local database, including both changed and unchanged records. Here’s how to do this in a procedure:

```
forcesynchronize
```

Theoretically you should never need to do a Force Synchronize, but the option is there if for any reason you think a database is not synchronizing properly. This statement is the same as holding down the **Option** key when you choose the **Synchronization** command from the File menu (see “[Regular Synchronization vs. Force Synchronization](#)” on page 147).

Automatic Pre and Post Synchronization Procedures

You may want to run a procedure automatically before and/or after each synchronization is performed. For example, you may want to make sure that the database is sorted a certain way after each synchronization, or you may want the current record to remain the same after a synchronization. To make this possible, Panorama checks for two special procedures in your database — **..PreSynchronize** and **..PostSynchronize**. If it finds a **..PreSynchronize** procedure it will run it before the synchronization is performed. If it finds a **..PostSynchronize** procedure it will run after the synchronization is complete. The **..PostSynchronize** procedure has four parameters that tell you about what happened during the synchronization process:

| Parameter | Description |
|-----------|---|
| 1 | True if one or more records downloaded from the server, false if no records downloaded. |
| 2 | True if one or more records deleted from the local copy of the database (because they were deleted by another user), false if no records deleted. |
| 3 | True if one or more records uploaded to the server (can only happen if off-line sharing is allowed, see Chapter 4). |
| 4 | Number of conflicts with other users, if any. |

The **..PreSynchronize** procedure can be used to save status about the database before the synchronization. For example, here is a typical **..PreSynchronize** procedure that saves the ID of the current record.

```
fileglobal BeforeSyncRecordID
BeforeSyncRecordID=info("serverrecordid")
```

If you also add this **..PreSynchronize** procedure to your database Panorama will make sure that the current record is the same before and after the synchronization process (unless the current record has been deleted).

```
fileglobal BeforeSyncRecordID
findbackwards BeforeSyncRecordID=info("serverrecordid")
```

Note: We used the `findbackwards` statement (instead of `find`) because in most cases on average it will be slightly faster (it makes no provision for undo). However, `find` could also be used.

Note: The pre and post synchronization procedures are both Handler procedures, which means that they should not switch database windows or open or close any databases.

Server Variables (Shared Variables)

When programming a single user database, you can store values (text or numbers) that you want to keep in one or more permanent variables. These values will be saved when you save the database. Permanent values can be used with shared databases, but they aren't shared. Each local copy of the database has its own private copy of the permanent variables. There is no way that one user can find out what another user's permanent variables contain, and no way to change what values are stored in another user's variables.

If you want to share a variable across the network, you have to use a new kind of variable, a server variable. As you might guess, server variables are kept on the server. Panorama has several statements and functions that allow a procedure to create, modify and access server variables.

To create or modify a server variable, use the `setservervariable` statement.

```
setservervariable database,variable,formula,initialvalue,response
```

This statement has five parameters:

| Parameter | Description |
|--------------|---|
| database | This is the name of the database that contains the variable (like permanent variables, server variables are always associated with a specific database). The specified database must be an open shared database that is currently connected to the server. If the database name is specified as "" then the current database will be used. |
| variable | This is the name of the variable. (You should NOT put quotes around the variable name.) The variable name must not contain spaces or other punctuation. |
| formula | This formula specifies the value to be stored in the variable. This can be a constant (37, "Arizona") or a more complex formula. If you want to include a field or variable you must enclose the field or variable with « and », even if that would not normally be required (for example «Name» or «factor»). The formula can include server variables. These are typed in directly, not using the <code>servervariable()</code> function, and must not have « and » around the name. For example if there is a server variable named <code>ReportCount</code> the formula should simply say <code>ReportCount</code> to reference this variable. (Note: these special rules apply only to this formula in the <code>setservervariable</code> statement, not to any other type of formula.) |
| initialvalue | This optional parameter specifies the initial value of the permanent variable if it hasn't been defined yet. This could be a numeric or text value. If no initial value is specified, the default value is "". |
| response | This is an optional parameter that you should usually leave off. It is used by the <code>adjustservervariable</code> statement. If this parameter is supplied, it should be true or false. If it is true, then when done, the global variable <code>_ServerVariable</code> is set to the new value of the variable. This allows you to not only set the value of the server variable, but also to find out what the new value is (if the formula contains a server variable this is the only way to find out the new value). |

As you can see using the `setservervariable` statement can be quite involved, but in most real applications it is much simpler. This example creates a shared variable named `PrintCount` for the current database, and assigns it the value 1.

```
setservervariable "",PrintCount,1
```

Here's a slightly more complex example. This procedure prompts the user to enter a tax rate, then stores the value in a server variable named `TaxRate`. Notice that in the `setservervariable` statement the local variable `newTaxRate` must have `«` and `»` around it, i.e. `«newTaxRate»`. If you don't include the `«` and `»` the statement will assume that you mean a server variable named `newTaxRate` and the procedure will stop and display an error message (unless there is such a server variable).

```
local newTaxRate
gettext "Tax Rate:",newTaxRate
setservervariable "",TaxRate,val(«newTaxRate»)
```

Variable names without `«` and `»` are assumed to be server variables, all other fields or variables must have `«` and `»` around them. This example increments (adds one) the `PrintCount` server variable. Notice that the initial value is set to zero, so if the `PrintCount` server variable has never existed before, it will be set to 1.

```
SetServerVariable "",PrintCount,PrintCount+1,0
```

In a real world application, you'll often want to use the `adjustservervariable()` function to increment a server variable (see "[Adjusting a Server Variable \(Atomic Calculation\)](#)" on page 160, just below).

Accessing Server Variables

To find out the current value of a server variable, use the `servervariable()` function. This function has two parameters — the name of the database (which may be `""` if the variable is in the current database) and the name of the variable (which must be in quotes). This procedure calculates the sales tax based on the shared `TaxRate` server variable.

```
SalesTax=SubTotal*(servervariable("", "TaxRate")/100)
```

Adjusting a Server Variable (Atomic Calculation)

Suppose you want the footer of a document to include something like this:

```
This form has been printed 327 times.
```

You could do this by using a procedure to print the form, like this:

```
SetServerVariable "",PrintCount,PrintCount+1,0
print dialog
```

and then including a formula like this in a Text Display SuperObject near the bottom of the form.

```
"This form has been printed "+str(servervariable("", "PrintCount"))+" times."
```

But wait! What if someone else prints this form 1/10th of a second after you do? They will update the print count and your print count will be wrong (because it will include both increments). It's not very likely to happen, but in a shared system the goal is to design your application so that problems like this can *never* happen. Happily this is easy to do with the `adjustservervariable()` function. Here is a revised version of the procedure.

```
fileglobal myPrintCount
myPrintCount=adjustservervariable("", "PrintCount",1)
print dialog
```

and the revised formula for the Text Display SuperObject.

```
"This form has been printed "+str(myPrintCount)+" times."
```

Now the print count is always correct, even if two users print at exactly the same time? How can that be? If the server gets two requests simultaneously, it randomly picks one and saves the other for later. The `adjustservervariable()` function increments the variable and returns its value all in one request. So there's no way for two or more users to get mixed up.

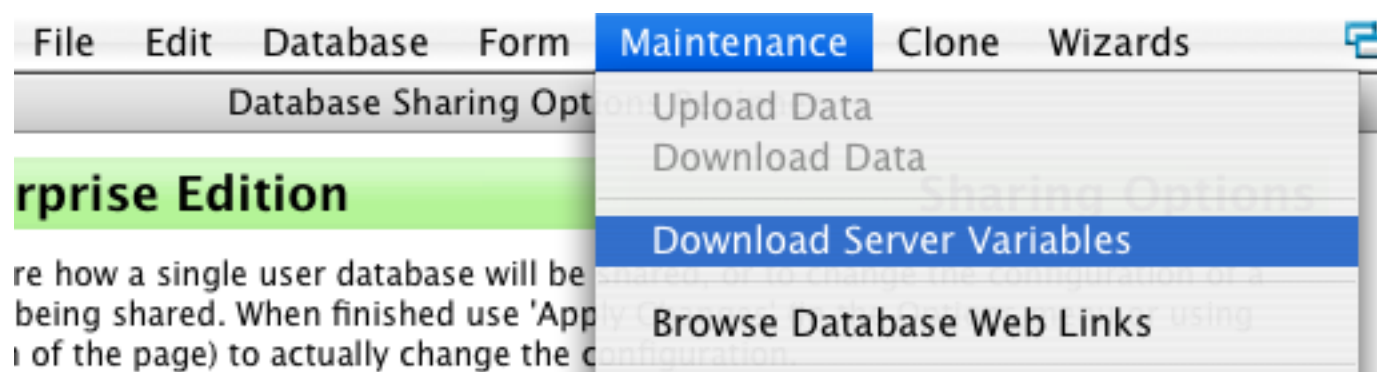
The `adjustservervariable()` function has three parameters:

```
adjustservervariable(database,variable,adjustmentvalue)
```

The first parameter is the name of the database (which may be "" if the variable is in the current database). The second parameter is the name of the variable (which must be in quotes). The final value is the amount that the variable will be adjusted by. This must be a numeric value. If the adjustment is positive, the value will be increased. If it is negative, it will be decreased.

Maintaining Server Variables when a database is Re-Shared

If a database contains server variables, you need to perform an extra step if you manually unshare and then share the database (see “[Making a New Sharing Generation Manually](#)” on page 128). In addition to synchronizing the database to get the latest data, you should also download the server variables (this step is taken care of for you automatically if you use the **Start New Sharing Generation** command, see “[Starting a New Sharing Generation](#)” on page 119). This is done with the **Download Server Variables** command in the **Maintenance** menu of the **Database Sharing Options** wizard.



Once the server variables are downloaded, you can continue with the process of converting the database to single user, modifying the database (adding or removing fields, etc.) and then re-sharing the database. The server variables will be re-uploaded to the server when you re-share the database.

Data Transformations

The rest of this chapter deals with operations on individual data cells and records. Panorama, however, also has a number of powerful commands that modify the current field in every selected record (**Fill**, **Empty Fill**, **Formula Fill**, **Propagate**, **UnPropagate**, **Running Total**, **Running Difference** and **Change**). In a shared database, these commands operate just as if you edited each cell by hand, with full record locking. In other words, these commands start with the first record, locks and synchronizes it, modifies it, unlocks the record, then goes on and repeats for each selected record in the databases. Since the server must be contacted twice for each record (once to lock, then again to unlock) this process is much, *much* slower than you are accustomed to when working with single-user databases. You'll want to minimize the use of these commands when working with shared databases, and be sure to select the minimum number of records necessary for the operation. When working with a single user database, it's no big deal to use **Formula Fill** on the entire database instead of just the few records that are actually going to change, but when working with a shared database, you want to make sure that, if possible, you select only the records that are actually going to be affected.

What if one of these commands (fill, etc.) encounters a locked record? In that case the locked record will be skipped and the command will continue with the next record. When the command is complete it will display an error message indicating that one or more records is locked. Depending on the situation, you may wish to repeat the operation later (when the record [or records] is no longer locked) or you may wish to **Undo** the data transformation operation. (Note: The **Undo** command may itself run into locked records that could cause errors!)

When you are using a procedure to perform a data transformation, the programmer can use the **if error** statement to trap and automatically process the record lock problem, as in the example below. In this example, the procedure will restore the fields to it's original state if it is unable to complete the propagate process.

```
propagate
if error
  undo
  message "Could not complete because someone else is editing one or more cells!"
  stop
endif
```

Here is another procedure that tries up to 5 times to fill a field, then gives up.

```
local try,maxtries
try=1
maxtries=5
field Total
loop
  formulafill Qty*Price
  if error
    try=try+1
  else
    rtn
  endif
while try<=maxtries
  rtnerror "Cannot complete operation."
```

Minimizing the amount of data changed by Fill commands

In the single user version of Panorama the various fill commands are lightning fast, and it doesn't really matter if you fill more data than you need to. In a shared environment, however, you need to be more careful and as much as possible only fill cells that actually need to change. Not only will this make the fill operation faster, but filling cells unnecessarily can interfere with and slow down Panorama's synchronization logic.

As an example, consider this procedure from an invoice database. The procedure re-calculates how much is owed on each invoice, then selects the unpaid invoices that are more than 30 days old.

```
field Balance
selectall
formulafill Total-Payments
select Balance>0 and InvoiceDate<today()-30
```

The problem with the procedure above is that it will fill a cell in every record in the database. If your database contains 20,000 invoices then all of them will be modified, even though most probably haven't really changed. The `formulafill` operation will take a long time, and subsequent synchronizations will take a long time also since they will have to synchronize all 20,000 records.

A better approach is to select only the records that actually are going to change and fill only those. Here is a rewritten version of this procedure:

```
field Balance
select Balance <> Total-Payments
formulafill Total-Payments
select Balance>0 and InvoiceDate<today()-30
```

Suppose you run this procedure every week — probably only few dozen records will have their balance change from week to week. So instead of filling 20,000 records only a few dozen will be filled. The fill will be much faster than before, possibly even hundreds of times faster, and synchronization will be smoother also. When converting a database from single user to shared operation, you should review any `fill` or `formulafill` statements to see if this technique could be used to improve multi-user operation.

You can improve this procedure even further by using the `serverformulafill` statement — see the next section.

ServerFormulaFill — A Much Faster Option for Select/Formula Fill Operation

It's very common to write a procedure that uses the `select` and `formulafill` statements consecutively. For example the procedure snippet below selects invoices that are ready to print, fills in the `PrintDate` field with today's date, then prints the invoices.

```
field PrintDate
select Ready match "yes"
formulafill today()
print dialog
```

If there are a lot of records ready to print the `formulafill` statement can be quite slow, as described in the previous section. This procedure can be made much faster by using the `serverformulafill` statement, which combines the selection and fill into a single statement, as shown here.

```
field PrintDate
serverformulafill «Ready» match "yes",today()
print dialog
```

This procedure is much faster than the first. Instead of individually locking and unlocking each record from the client, the `serverformulafill` statement performs the bulk of the work on the server. This allows the statement to modify 1000 records in a couple of seconds, rather than the several minutes it would take using the regular `formulafill` statement. Panorama still checks each record to see if it is record locked (more on that in a minute), but this checking is done completely on the server with no network overhead at all.

The `serverformulafill` statement has two parameters:

```
serverformulafill selectionFormula,fillFormula
```

The first parameter is a formula that specifies what records to select. After the statement is complete these records will be selected, just as if you had used the regular `select` statement. *Important Note:* You cannot select the records to fill in advance (for example with a series of `select`, `selectadditional` and/or `selectwith` statements, or simply an ad-hoc selection of records). You must specify the records with the first parameter of this statement. If you cannot come up with a single formula that will select the records you want to modify you cannot use the `serverformulafill` statement.

The second parameter is a formula that specifies the new values for the data, just like the formula parameter to the `formulafill` statement (subject to the restrictions in the next section).

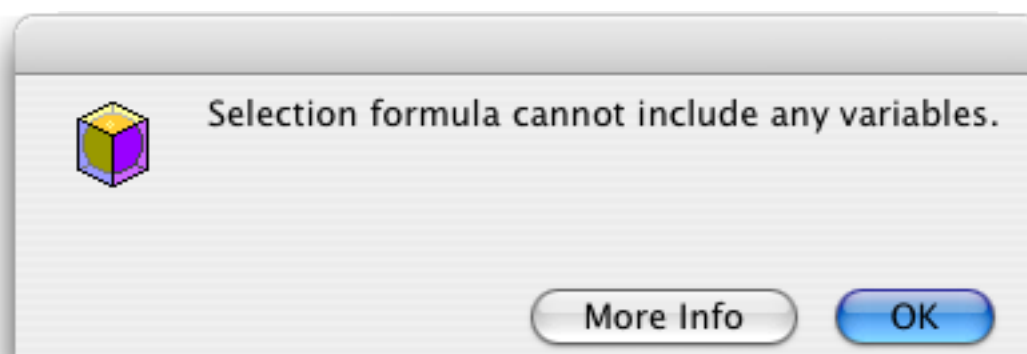
Here's how this statement works. First, it checks the two formulas and then passes them to the server. Only the formulas are passed to the server — this statement does not move any data in either direction. The server then selects the records specified by the selection formula and fills the cells in the current field with the new data calculated by the fill formula. This all happens very quickly because it is all happening in RAM on the server computer. Once the process is complete Panorama performs the same steps on the client computer — select and fill. In other words it performs the same select and fill operations twice — once on the server and once on the client. Since the calculations are the same the results are the same on both computers. (The only possible fly in the ointment is if one or more records are locked. This will be discussed in a moment.)

ServerFormulaFill Formula Restrictions

Because these formulas will be processed both on the server and the client they cannot use any variables (including server variables), only fields and constants. This means that the procedure below will not work.

```
local TaxRate,OurState
TaxRate=7.75
OurState="CA"
field Tax
serverformulafill State=OurState,Subtotal*(TaxRate/100)
```

If you try to run this procedure an error message like this will appear:



One way to fix this is to simply embed the actual values into the `serverformulafill` statement:

```
field Tax
serverformulafill State="CA",Subtotal*(7.75/100)
```

Another option is to use the `execute` statement and the `constantvalue()` function, as shown in this example:

```
local TaxRate,OurState
TaxRate=7.75
OurState="CA"
field Tax
execute {serverformulafill State=}  
constantvalue("OurState")+  
{,Subtotal*({}  
constantvalue("TaxRate")+{/100)}
```

This example works by building a short procedure with the variable values embedded into the formulas, then executing that.

Record Locking and the ServerFormulaFill Statement

When the `ServerFormulaFill` statement fills data on the server it will skip any records that are locked by another user on the network. These records will be skipped both on the server and on the client. If the statement was unable to fill all requested records because some were locked the procedure will stop and display an error message. If necessary, the procedure can trap this error with the `if error` statement. The error message, which can be retrieved with the `info("error")` function, will indicate the number of locked records and also contain a list of the record ID numbers for the locked record. The procedure shown below will try up to 5 times to fill the selected records, then it will give up.

```
local try,maxtries
try=1
maxtries=5
field Tax
loop
  serverformulafill State="CA",Subtotal*0.0775
  if error
    try=try+1
  else
    rtn
  endif
while try<=maxtries
rtnerror "Cannot complete operation."
```

Minimizing the amount of data changed by ServerFormulaFill commands

Although the `ServerFormulaFill` statement is much faster than the `formulafill` statement it is still important for smooth synchronization to keep the number of records modified to a minimum. Here is an example of how to do this, using the same Invoice database to calculate Balances as in the example on the previous page (see "[Minimizing the amount of data changed by Fill commands](#)" on page 162).

```
field Balance
serverformulafill Balance <> Total-Payments,Total-Payments
select Balance>0 and InvoiceDate<today()-30
```

When this procedure runs only records where the `Balance` has actually changed will be modified.

Looking Up Data From Another Database

Panorama has several functions that look up data from other open databases. Most of these work just as they do in single user mode, looking up data from the local RAM copy of the requested database. The `lookupselected()`, `lookuplast()`, `table()` and `lookupall()` functions always lookup from the local copy of the database.

The `lookup()` function, however, goes straight to the server copy of the database in some situations. This ensures that the `lookup()` function always retrieves the most up-to-date information. The `lookup()` function will get data straight from the server except in the following situations:

- 1) If the `serverlookup` statement has been used to temporarily disable looking up data directly from the server (see below).
- 2) If the `lookup()` function is in an object on a form and the **Use Server for Lookup** option is turned off in the **Form Preferences** dialog.
- 3) If the local computer is not connected to the server.
- 4) If the `lookup()` function specifies a summary record (because summaries are not stored on the server)
- 5) If the formula containing the `lookup()` function is being calculated repeatedly, for example if the function is being used in a `formulafill`, `select`, `arraybuild`, `arrayfilter`, etc.

In any of these five situations the `lookup()` function will retrieve from the local copy of the database. In the last situation you should synchronize the target database if you want to ensure that the looked up values are completely up-to-date.

Temporarily Disabling Direct Lookups from the Server

The `serverlookup` statement controls whether lookups are made from the local copy of the target database or are made directly to the server. This statement allows the database designer to trade off speed vs. up-to-the-minute accuracy in lookups made in a procedure. This option only affects shared databases. For up-to-the-minute accuracy lookups should be made directly from the server. However lookups from the server are substantially slower than lookups from the local database. The example below forces the status to be looked up from the local database instead of from the server.

```
serverlookup "off"
Status = lookup( "Patients", "SSN", "SSN", "Status", "", 0)
serverlookup "on"
```

The `info("serverlookup")` function allows you to determine if server lookups are enabled. If they are, it returns true (this is the default). This example turns off server lookups and looks up a value. It then turns the server lookup option back on, but only if it was already on before this procedure began.

```
local slook
slook=info("serverlookup")
serverlookup "off"
Z=lookup("Prices", "Item", i, "Price", "", 0)
if slook
  serverlookup "on"
endif
```

The `formserverlookup` statement allows a procedure to turn the **Use Server for Lookup** option on or off. (This option can also be turned on or off in the **Form Preferences** Dialog.) This statement allows the database designer to trade off speed vs. up-to-the-minute accuracy in lookups on a form. This option does not affect lookups made in procedures, only lookups in auto-wrap text objects and Text Display SuperObjects. The example forces lookups in the form `Patient Status` to be made from the local database instead of directly from the server.

```
openform "Patient Status"
formserverlookup "off"
```


Controlling and Monitoring the Server Connection

Panorama normally connects to the server automatically when a database is opened, and disconnects when it is closed. However it is possible to override this default behavior and connect and disconnect on command.

ConnectToServer Statement

The `connecttoserver` statement tells Panorama to connect the current database with the server (of course this only works for shared databases, and only if it is not currently connected). The statement has no parameters. For example suppose the network was not working when a shared database was opened — perhaps the Ethernet cable was not plugged in. After plugging in the cable you can use a procedure with the `connecttoserver` statement to establish the connection (otherwise you would have to close and re-open the database). The statement can also be used with databases that do not automatically connect to the server when they are opened (see “[Automatically Connect](#)” on page 189).

DropServer Statement

The `dropserver` statement tells Panorama to disconnect the current shared database from the server (of course this only works for shared databases, and only if the database is currently connected). For example you might use this statement if you are planning on unplugging your network connection but want to continue working with the database offline (see “[Making and Dropping Server Connections](#)” on page 184). This statement has no parameters.

The info(“serverconnection”) Function

The `info(“serverconnection”)` function returns `true` if the current database is currently connected to the server, `false` if it is not connected. The example below displays an error message if the database is not currently connected.

```
if (not info("serverconnection"))
  message "Database is not currently connected."
  rtn
endif
```

In a real procedure the code would probably perform some function after the `endif`.

The sharedusers(Function

The `sharedusers(` function returns a list of users that are currently sharing a database. The function has one parameter, the name of the database. The specified database must currently be connected to the server on this computer. If the database is not connected, or is not a sharable database, the result will be `""`. If the database name is `""` then the current database will be used.

If this is a connected database the result will be a carriage return separated array, with each line containing the session id, user name, and user's computer's name separated by tabs. Here's a typical example of the output from this function.

```
12   Rudy Red      Conference Room Computer
15   Marie Ellis   Marie's Computer
29   Mark Watson   Mark's Computer
```

This example procedure checks to see if Rudy Red is using the same shared database we are right now.

```
if sharedusers("") contains ^+"Rudy Red"+^
  message "Rudy is online right now!"
endif
```

The `sharedusers(` function can also be used to list all users on the server, not just the ones using a specified database. To do this set the database name to `"*"` (the current database must be a shared database for this to work).

The `servername()` Function

This function returns the name of the server associated with a shared database. The function has one parameter, the name of the database. If the database name is "" the server name for the current database will be returned.

The `serverdatabasename()` Function

The name of a database on the server is not necessarily the same as the name of the same database on the client computer. (In fact, you can actually have more than one copy of a single database on a client computer with different names even though each connects to the same database on the server, though we can't think of any reason why you'd want to do that.) Procedures usually don't need to know the name of the database on the server, but if they do they can use the `serverdatabasename()` function. This function has one parameter, the name of the database. If the database name is "" the server name for the current database will be returned. (One case where you do need this name is if you want to delete the server database, see "[Deleting a Database from the Server](#)" on page 171).

Shared Database Configuration

Configuration of shared databases is usually done through the **Database Sharing Options** wizard. This wizard, however, is actually a graphical front end for the `eesetdbconfig` and `eegetdbconfig` statements, which do the actual work.

The `EESetDBConfig` Statement

This statement is used to configure a shared or web publishing database. If the database is currently single user, this statement can set it up on server. If the database is already on the server, this statement can change sharing and web publishing options as specified (or even convert the database back to single user mode). This statement is used by the **Database Sharing Options** wizard, but may also be used separately by applications that need to automatically set up shared or web published databases. (Note: You will be prompted for the server password when this statement is used.) The statement has three parameters

```
eesetdbconfig database,options,performed
```

The first parameter, `database`, is the name of database on this computer. The database must be already open.

The second parameter, `options`, is a dictionary of sharing and web publishing options (see "[The Configuration Dictionary](#)" on page 169).

The third parameter, `performed`, is the name of a field or variable. When the `eesetdbconfig` is complete this field or variable will be set to a list of the changes actually made. The **Database Sharing Options** wizard displays this result to let the user see what has happened, you may want to do the same.

If the `eesetdbconfig` statement can't complete the requested changes (for example if the server is not currently available on the network) it will return an error. The procedure can trap this with the `if error` statement, and can find out what the error was with the `info("error")` function.

Here is a basic procedure which takes the current database (a single user database) and converts it into a shared database on the server [Acme Widgets Server](#).

```
local cfoptions,cfresult
initializedictionary cfoptions,
    "ServerURL","Acme Widgets Server",
    "ReplaceExisting","no",
    "ZapSummaries","yes",
    "Sharable","yes",
    "AutoTimeOut","20"
eesetdbconfig "",cfoptions,cfresult
if error
    message info("error")
    rtn
else
    giantmessage cfresult
endif
```

If the database is already on the server then you should first use the `eegetdbconfig` statement (see below) to get the current configuration, then change the settings you want to change, then use the `eesetdbconfig` statement. This example changes the record lock auto timeout value to 40 seconds.

```
local cfoptions,cfresult
eegetdbconfig "",cfoptions
setdictionaryvalue cfoptions,"AutoTimeOut","40"
eesetdbconfig "",cfoptions,cfresult
```

The EEGetDBConfig Statement

This statement is used to get the current sharing configuration of any database (including a single user database). This statement is used by the **Database Sharing Options** wizard, but may also be used separately by applications that need to find out how a database is configured. (Note: If the database is set up on the server, you will be prompted for the server password when this statement is used.)

The statement has two parameters

```
eegetdbconfig database,options
```

The first parameter, `database`, is the name of database on this computer. The database must be already open.

The second parameter, `options`, is a field or variable. The statement will set this field or variable to a dictionary of sharing and web publishing options (see "[The Configuration Dictionary](#)" on page 169). You can then use the `getdictionaryvalue()` function to check how individual settings are configured.

See the previous section for an example of this statement in use.

The Configuration Dictionary

Both the `eesetdbconfig` and `eegetdbconfig` statements use a "data dictionary" structure to work with the sharing configuration. This is not an English dictionary, but a set of key/value pairs. If you are not familiar with data dictionaries you should review Data Dictionaries in Chapter 25 of the Panorama Handbook. You can also look in the online **Programming Reference** wizard for `SetDictionaryValue`, `GetDictionaryValue`(and `InitializeDictionaryValues`.

The table below lists each of the dictionary keys used by the `eesetdbconfig` and `eegetdbconfig` statements.

| Key | Description |
|---|---|
| <code>"ServerURL"</code> | Server on which this database is to be hosted (or already hosted). |
| <code>"ServerDatabaseName"</code> | Name to be used on for this database on the server. The name does not have to be the same as the original database name, but it there must not be a database with this name already on the server. If this key is omitted, the original database name will be used. |
| <code>"ReplaceExisting"</code> | If the <code>ServerDatabaseName</code> is already in use, this option specifies what to do. The options are: Yes (delete the previous database without asking), No (do not delete previous database, this is the default if not specified), YesNo (ask user if they want to delete, defaulting to yes), NoYes (ask user if they want to delete, defaulting to no). |
| <code>"ZapSummaries"</code> | If a database contains summary records they must be removed before the database can be made sharable. This option determines what should be done if the database contains summary records. The options are: Yes (automatically delete summary records without asking), No (do not delete summary records, do not convert database to shared, and statement returns an error), YesNo (ask user what to do, defaulting to yes), NoYes (ask user what to do, defaulting to no). If the user chooses not to remove summary the database is not converted which can be checked by examining the results (the <code>performed</code> parameter). |
| <code>"Sharable"</code> | This option must be set to "yes" if the database is to be made (or continue to be) sharable remotely over the internet. Otherwise this option should be " " (the default). |
| <code>"RemoteSharable"</code> | This option must be set to "yes" if the database is to be made (or continue to be) sharable. Otherwise this option should be " " (the default). |
| <code>"AutoTimeOut"</code> | Number of seconds of inactivity before client unlocks record (0 = no timeout, which is the default value if not specified). |
| <code>"AllowOfflineModification"</code> | This option must be set to "yes" if the modifications are allowed when the database is disconnected from the server (see Chapter 4). Otherwise this option should be " " (the default). |
| <code>"SyncPriority"</code> | This option controls how the database is synchronized after being modified offline (see Chapter 4). It should be set to "local" or "server" . |
| <code>"AutoConnect"</code> | This option must be set to "yes" if the database should automatically attempt to connect to the server when opened. Otherwise this option should be " " (the default). |
| <code>"AutoSync"</code> | This option must be set to "yes" if the database should automatically attempt to synchronize each time it is connected to the server. Otherwise this option should be " " (the default). |
| <code>"KeepOfflineChangesLocal"</code> | This option must be set to "yes" if offline changes should not be updated to the server. Otherwise this option should be " " (the default). |
| <code>"AllowOfflineAddRecord"</code> | This option must be set to "yes" if the database allows adding records when disconnected from the server. Otherwise this option should be " " (the default). |
| <code>"AllowOfflineDeleteRecord"</code> | This option must be set to "yes" if the database allows deleting records when disconnected from the server. Otherwise this option should be " " (the default). Note: The records are deleted from the local copy only, there is no way to delete records on the server when off-line. |

| Key | Description |
|--------------|--|
| "WebPublish" | This option must be set to "yes" if the database is to be made (or continue to be) web publishable. Otherwise this option should be "" (the default). |
| "StartOpen" | This option must be set to "yes" if this a web publishing database and should be loaded when the server launches. Otherwise this option should be "" (the default). |
| "AutoOpen" | This option must be set to "yes" if this a web publishing database and should be loaded automatically when needed. Otherwise this option should be "" (the default). |
| "AutoClose" | This option must be set to "yes" if this a web publishing database and should be closed automatically after each access. Otherwise this option should be "" (the default). |
| "UseSecret" | This option must be set to "yes" if this a web publishing database and should use "secret" windows instead of a real window. Otherwise this option should be "" (the default). |
| "Initialize" | This option must be set to "yes" if this a web publishing database and the server should call its .Initialize procedure when opening the file. Generally this should be avoided if possible, especially if the StartOpen option is not used. Otherwise this option should be "" (the default). |

Deleting a Database from the Server

Deleting a database from the server is usually done using the **Server Administration** wizard (see “[Server Management \(The Server Administration Wizard\)](#)” on page 68). However, this can be done in a procedure, though we normally don't recommend doing it that way. To actually delete a database from the server in a procedure use the `deleteserverdatabase` statement.

```
deleteserverdatabase server,database
```

The first parameter, *server*, is the name of the server that contains this database.

The second parameter, *database*, is server name of the database to be deleted. Keep in mind that this may not be the same as the name of the database on the client. If the database is open you can find out the server name using the `serverdatabasename(database)` function. However, you must close the database before you delete it. The Panorama Server will not delete any database that is currently open.

Here is a procedure that will convert the current database to single user, then delete the current database from the server.

```
local cfoptions,cfresult,svname,dbname,svdbname
eegetdbconfig "",cfoptions
if getdictionaryvalue(cfoptions,"Sharable")<>"yes"
  message "This database is already single user (not sharable)."
  rtn
endif
setdictionaryvalue cfoptions,"Sharable",""
eesetdbconfig "",cfoptions,cfresult      /* convert to single user */
save
dbname=info("database")
svname=servername("")                    /* what is name of this server? */
svdbname=serverdatabasename("")         /* what is name of this database on server? */
closefile
loop
  deleteserverdatabase svname,svdbname
  if error
    if info("error") notcontains "currently open"
      message info("error")
```

```
    stoploopif l=1
else
    superalert "This database (" + dbname + ") cannot be deleted "+
        "because other users currently have it open. "+
        "When everyone has closed the database press OK to delete this database, "+
        "or press Cancel to give up.", {}
    stoploopif info("dialogtrigger") contains "cancel"
endif
else
    stoploopif l=1 /* always stop loop if no error */
endif
while forever
```

Of course once you've deleted the database from the server no one can open it again anywhere on the network.

Chapter 4: Offline Database Sharing

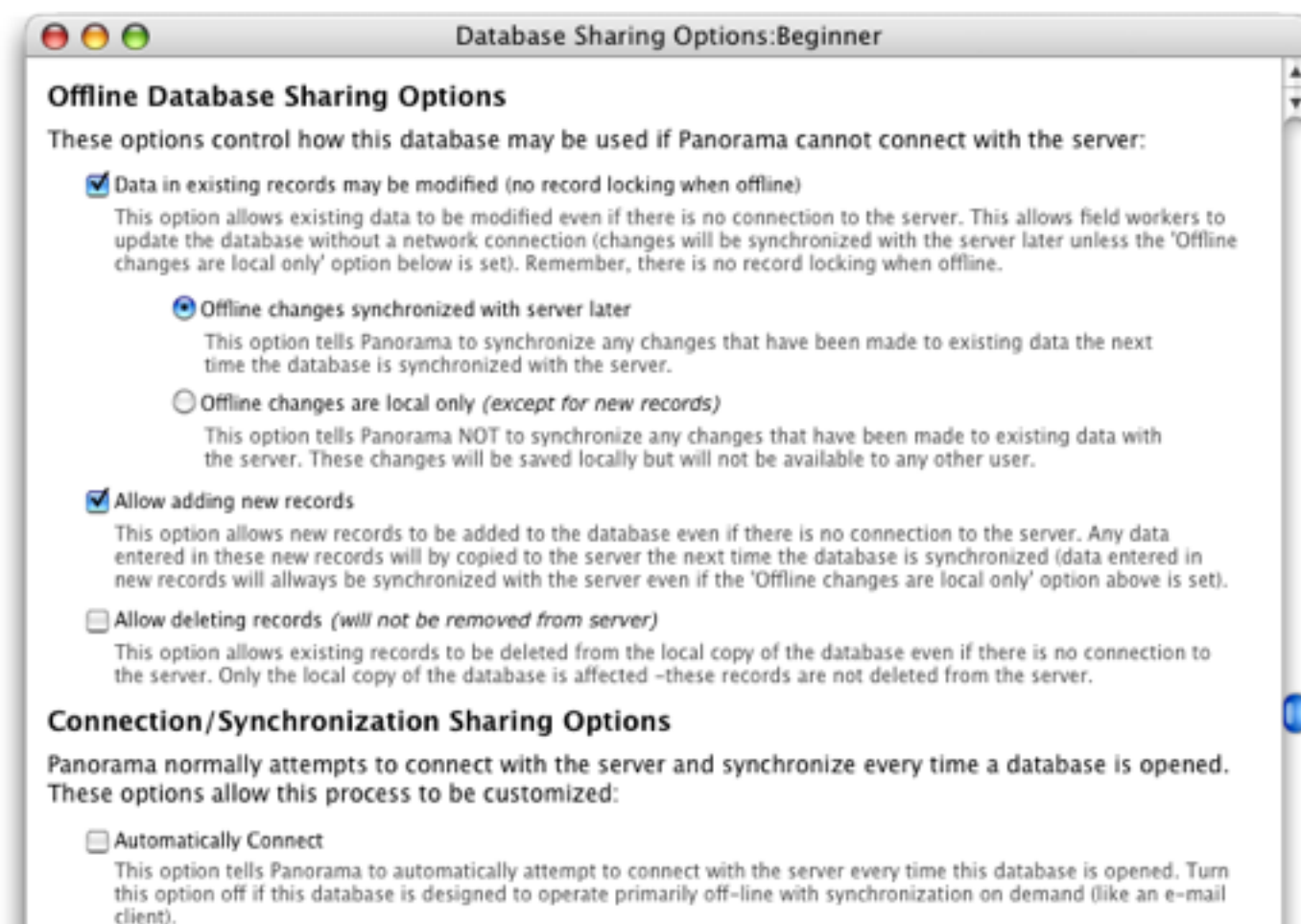


Unlike a traditional client/server database, Panorama database clients don't "go dark" if a network connection is unavailable. Panorama database sharing allows off-line database browsing and even modification (configurable on a per database basis). If allowed, offline changes are automatically synchronized with the server when the client re-connects to the network. Because both client and server are RAM based, this synchronization is extremely fast. If there is a potential conflict between the modifications made offline and the modifications made by other users you will be notified and given the option to resolve the conflict manually.

A Panorama database can even be configured to operate primarily offline. This is similar to the way e-mail works - users perform data entry offline, then press **Submit** or **Connect** to submit their data and receive updates. In the past applications like this had to be built from scratch, but Panorama database sharing allows this with little or no custom programming.

Offline Sharing Options

Each shared database has its own separate settings that specify how the database will behave when it is offline and what will happen when the database is connected to the server after being modified offline. These settings can be set up and modified with the **Database Sharing Options** wizard.



Important Note: The **Database Sharing Options** wizard affects only the current copy of the database on the current computer. If you've already set up a database on several client machines (see "[Duplicate Database Conflicts on the Server](#)" on page 108) then changing the options, one client won't affect any of the other clients (each client will retain its own previous settings). You'll either need to use the **Database Sharing Options** wizard on each client or you'll need to transfer a copy of the modified database to each of the other clients. (In some cases you can use this feature to advantage — for example you might want the offline settings for a supervisor to be different from the settings for other employees.)

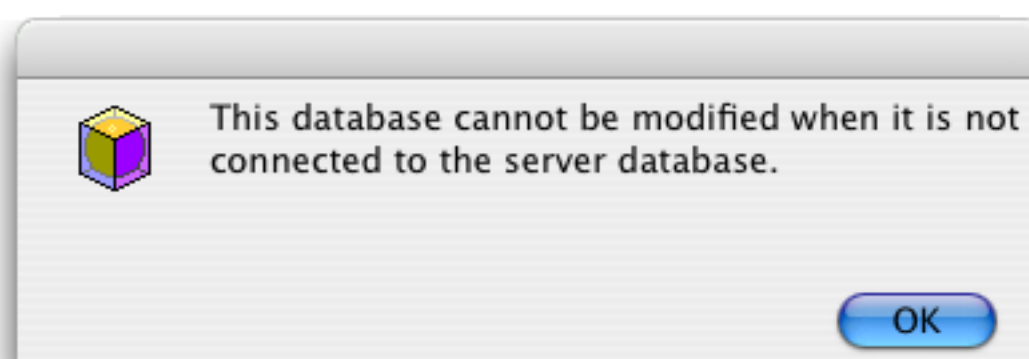
Is Offline Modification of Existing Records Appropriate for your Application?

The first decision you'll need to make is whether offline modification is appropriate for a particular database. When a modifications are made offline, there is no record locking (see "[Editing Data and Record Locking](#)" on page 136), no network wide automatic record numbering (see "[Automatic Record Numbering](#)" on page 148) and no access to shared server variables (see "[Server Variables \(Shared Variables\)](#)" on page 159). For some applications these limitations won't matter, but for others, one or more of these could be deal killers. For example, consider an application for recording results at a track meet. In this application you could have many disconnected laptops each being used to enter results which are then synchronized to the main server later. Since each laptop is being used at a different event, there won't be any conflicts and off-line data entry will work well. But an application like hotel reservations won't work so well. If the machines aren't connected then two different computers might rent out the same room at the same time. In this application off-line modification isn't really an option. You'll need to carefully consider how each database will be used before you enable the off-line modification option.

Data in existing records may be modified (no record locking when offline)

This option allows existing data to be modified even if there is no connection to the server. This allows field workers to update the database without a network connection (changes will be synchronized with the server later unless the 'Offline changes are local only' option below is set). Remember, there is no record locking when offline.

If this option is turned off (the default) you can still open the database when the client is off-line. However, Panorama will not allow you to modify the database. If you try to type into or click on a field to edit it, Panorama will display an alert explaining that the database cannot be modified.



If the option is turned on you'll be able to modify the database even when off-line. The changes made while off-line will be synchronized with the server the next time the database is connected to the server (subject to some of the other options described below).

Is Adding New Records While Offline Appropriate for your Application?

Even if modifying existing records offline is not appropriate for a particular application, it may be acceptable to add new records. For example this might be appropriate for the track meet application described in the previous section. Suppose that the track meet database has the **Data in existing records may be modified** option turned off, but the **Allow adding new records** option turned on. In this case, each laptop could enter new results and sync these new results to the server later, but they could not modify any data that had been entered by anyone else unless they were connected to the server. (In fact, once synchronized they could not even modify the data they had entered themselves.)

Allow adding new records

This option allows new records to be added to the database even if there is no connection to the server. Any data entered in these new records will be copied to the server the next time the database is synchronized (data entered in new records will always be synchronized with the server even if the 'Offline changes are local only' option above is set).

If you want to allow both modifying existing data and adding new data you'll need to enable both of these options. This allows the offline experience to be as close as possible to normal on-line database access.

Deleting Records While Offline

Bottom line — deleting shared records while offline isn't allowed. While technically it would be possible to delete records offline and then delete them from the server the next time the database was synchronized, we felt that this was simply way too dangerous. So you must actually be connected to the server to delete one or more records.

It is possible, however, to delete records from the *local* copy of the database while offline.

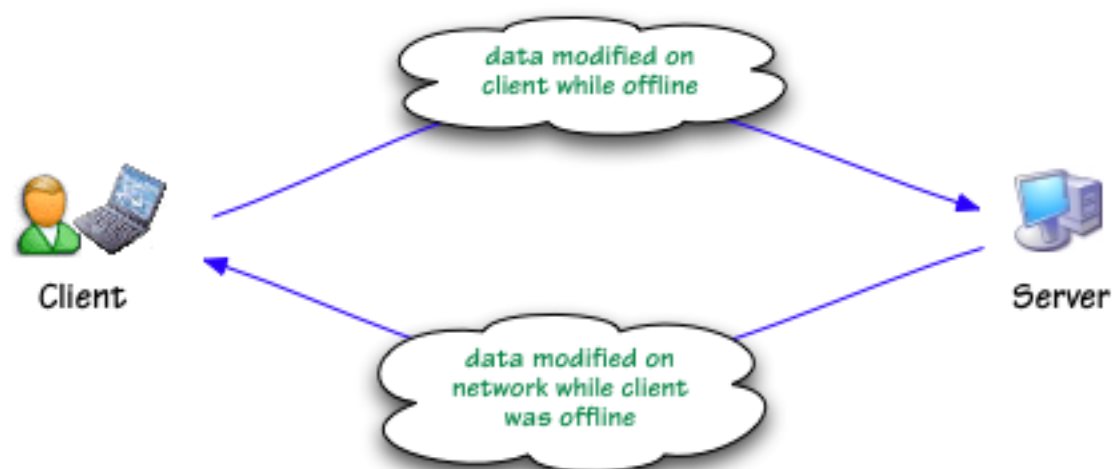
Allow deleting records (will not be removed from server)

This option allows existing records to be deleted from the local copy of the database even if there is no connection to the server. Only the local copy of the database is affected —these records are not deleted from the server.

Why would you do this? Consider the track meet example with the **Data in existing records may be modified** option turned off and the **Allow adding new records** option turned on. In this case, you can't modify the records you've already entered after synchronizing, so perhaps you'd like to delete them and start with a blank slate. This option allows you to do that on the local client computer without actually deleting the shared data already on the server. If you use this option, however, be very careful. Don't delete records when online unless you really want them gone. If you use this option you should probably also consider turning off the **Automatically Connect** option (see "[Configuring a Client Database for Primarily Offline Operation](#)" on page 189) so that you don't accidentally delete shared records unintentionally.

Two Way Synchronization

The first time a database connects to the server after being modified offline Panorama does a two way synchronization, instead of the normal one way (server to client) synchronization. This is necessary because records may have been modified at both ends.



This two way synchronization works without conflict as long as none of the records modified offline were also modified by other clients that were connected to the network.

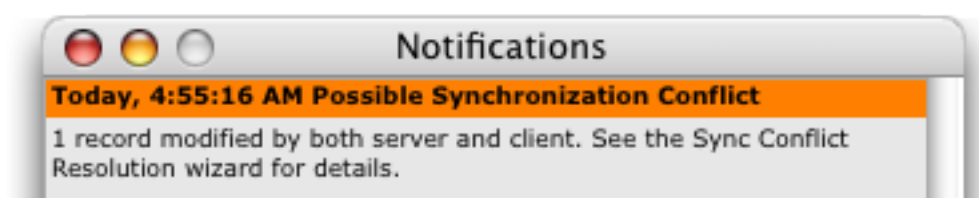
Managing Synchronization Conflicts

In the real world synchronization conflicts will sometimes occur. Panorama uses a two level system for resolving such conflicts when they occur. The first level is to simply give priority to either the offline client or the rest of the network. When a record is modified at both ends, the version on the end with priority is retained, while the other version is discarded by the synchronization process. Use the **Database Sharing Options** wizard to specify which end has priority.

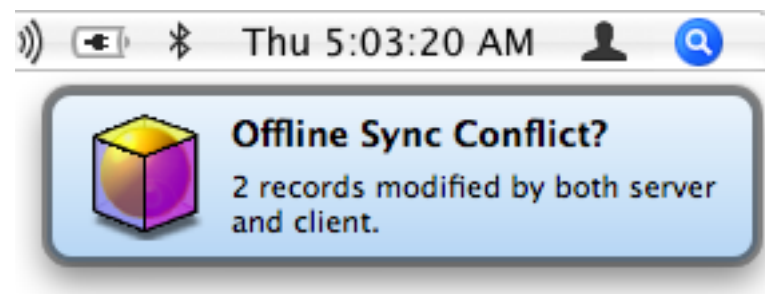
When synchronizing with the server after offline modifications:

- Give priority to changes made on the client computer
If a record has been modified offline and also modified by another user on the network, this option gives priority to the copy of the database that was modified offline.
- Give priority to changes made on the rest of the network
If a record has been modified offline and also modified by another user on the network, this option gives priority to the copy of the database on the server (the offline changes in that record will be discarded).

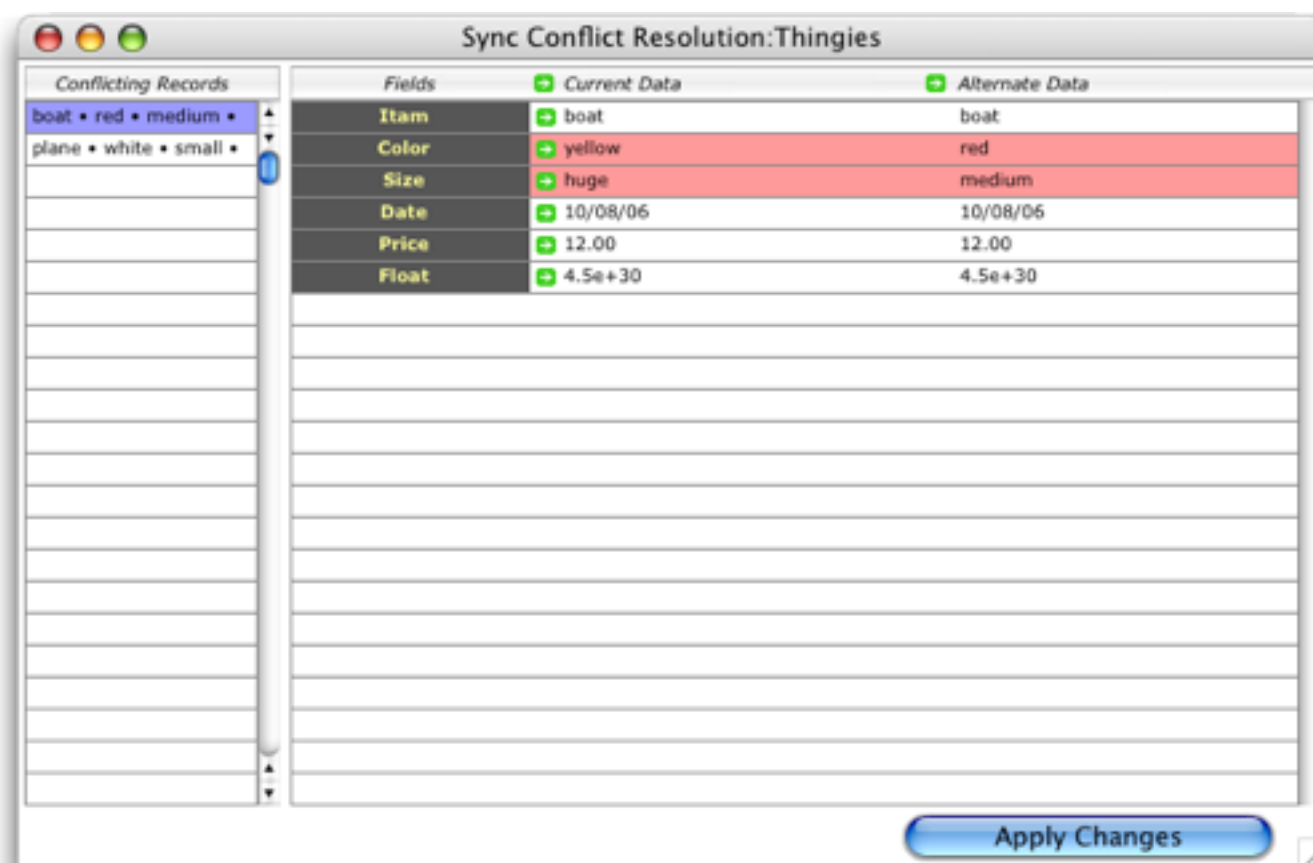
When a synchronization conflict does occur, Panorama will warn you.



Panorama normally uses the **Notification** wizard to warn you about conflicts, but if you have installed and enabled **Growl** (see “[Using Growl for Notifications](#)” on page 87) then the warning message will appear temporarily and then fade away.

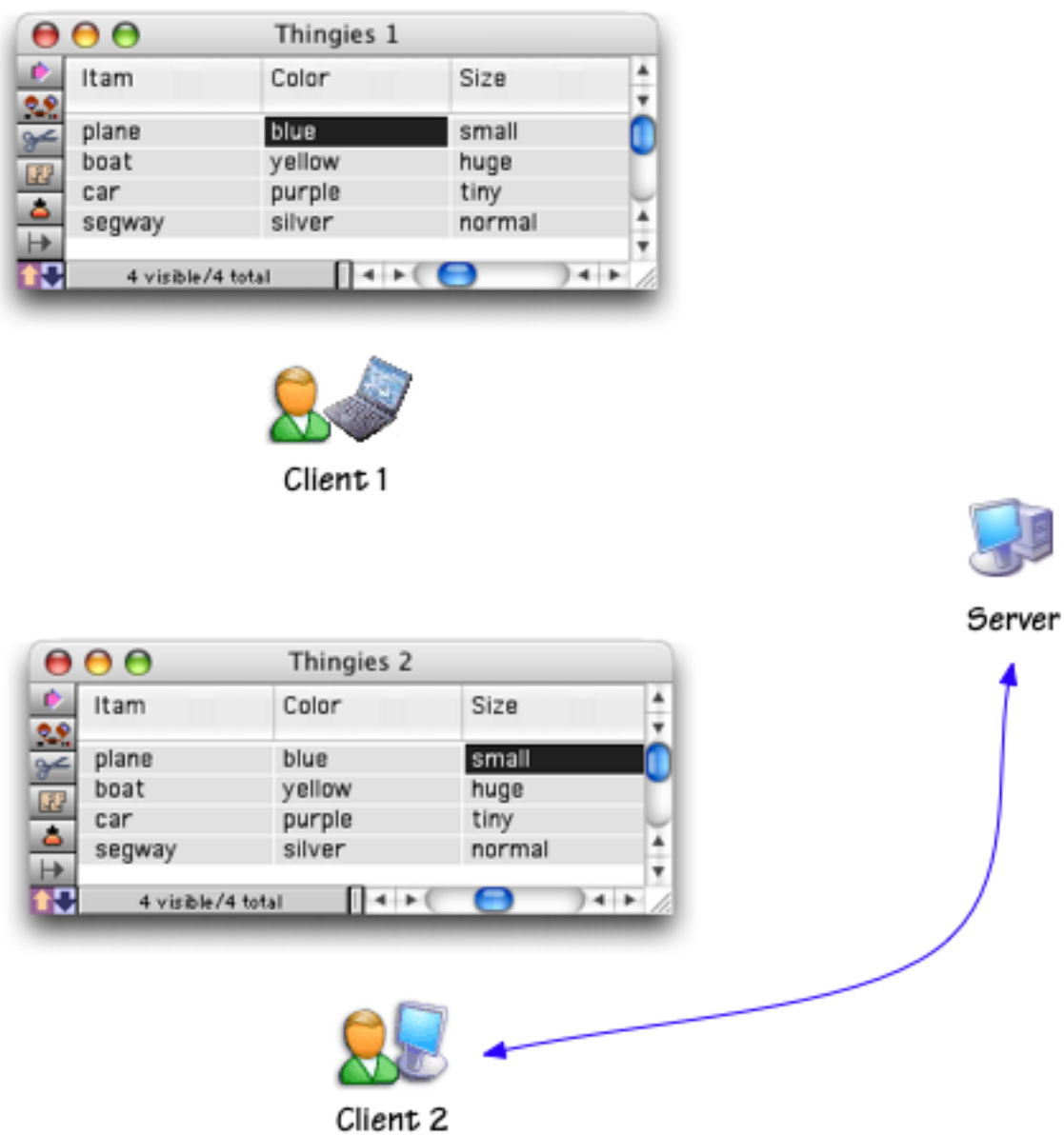


To see exactly what the conflicts are (and possibly change the resolution) open the **Sync Conflict Resolution** wizard (in the **Sharing** submenu of the **Wizard** menu).



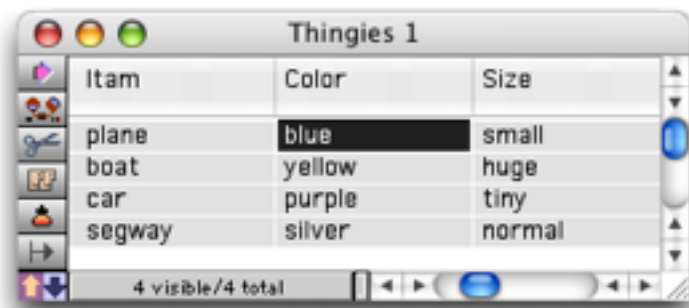
The left hand side of this wizard lists all of the records that contain conflicts (data modified on both ends). In this example there are two conflicting records. Click on a record to see the exact fields that have been changed on the right. The right hand side of the wizard contains three columns — the names of the fields in the database, the current data (after synchronization) and the alternate data that was automatically discarded by the synchronization process. Fields containing conflicts are highlighted in red.

The **Sync Conflict Resolution** wizard is best explained by example. We'll use a very simple database called **Thingies** which has six fields and four records.



Thingies is on two clients, Client 1 and Client 2 (normally the database name would be the same on both clients, but we've renamed them to **Thingies 1** and **Thingies 2** for clarity). The database starts out exactly the same on both client machines.

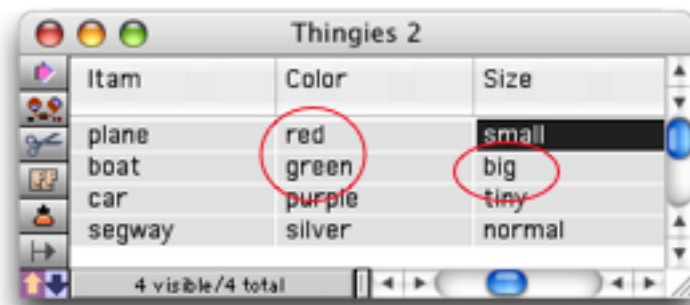
The action starts with Client 2 making some changes. The plane is changed from **blue** to **red**, the boat from **yellow** to **green** and from **huge** to **big**.



| Item | Color | Size |
|--------|--------|--------|
| plane | blue | small |
| boat | yellow | huge |
| car | purple | tiny |
| segway | silver | normal |



Client 1



| Item | Color | Size |
|--------|--------|--------|
| plane | red | small |
| boat | green | big |
| car | purple | tiny |
| segway | silver | normal |



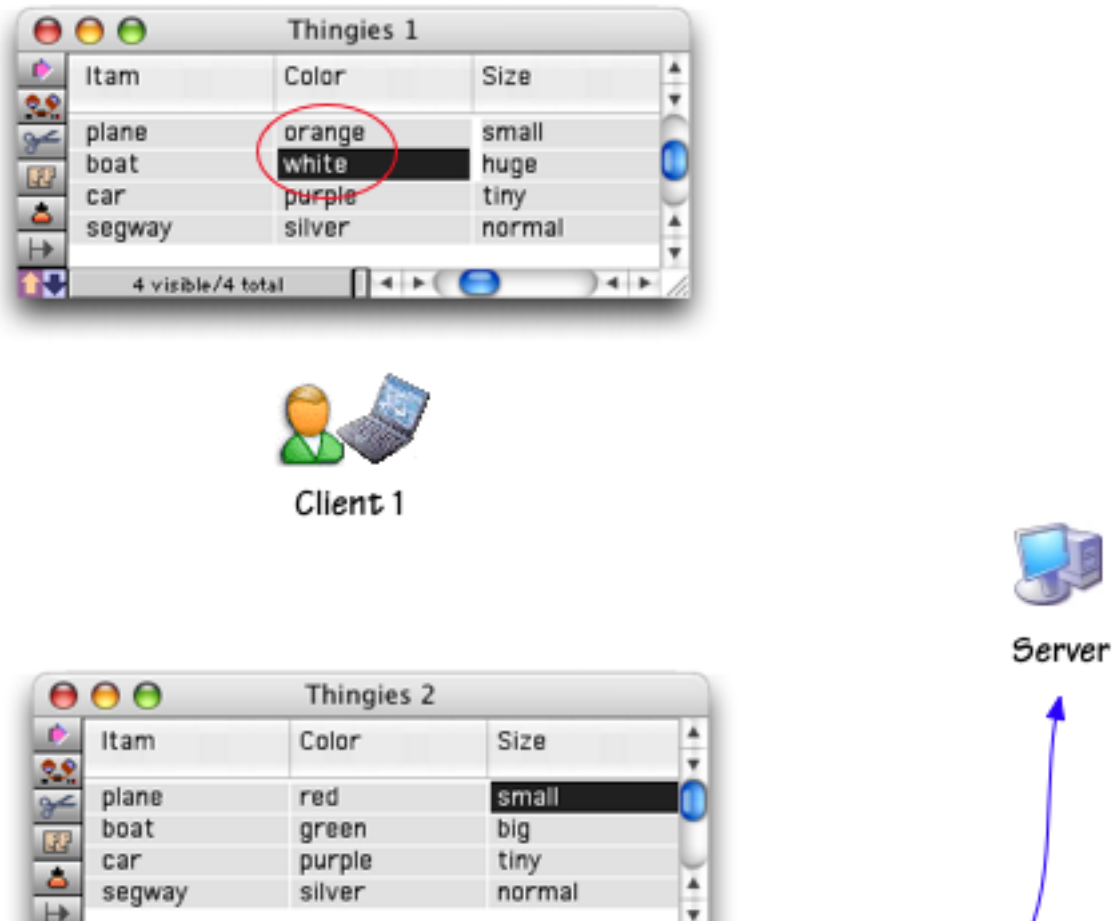
Client 2



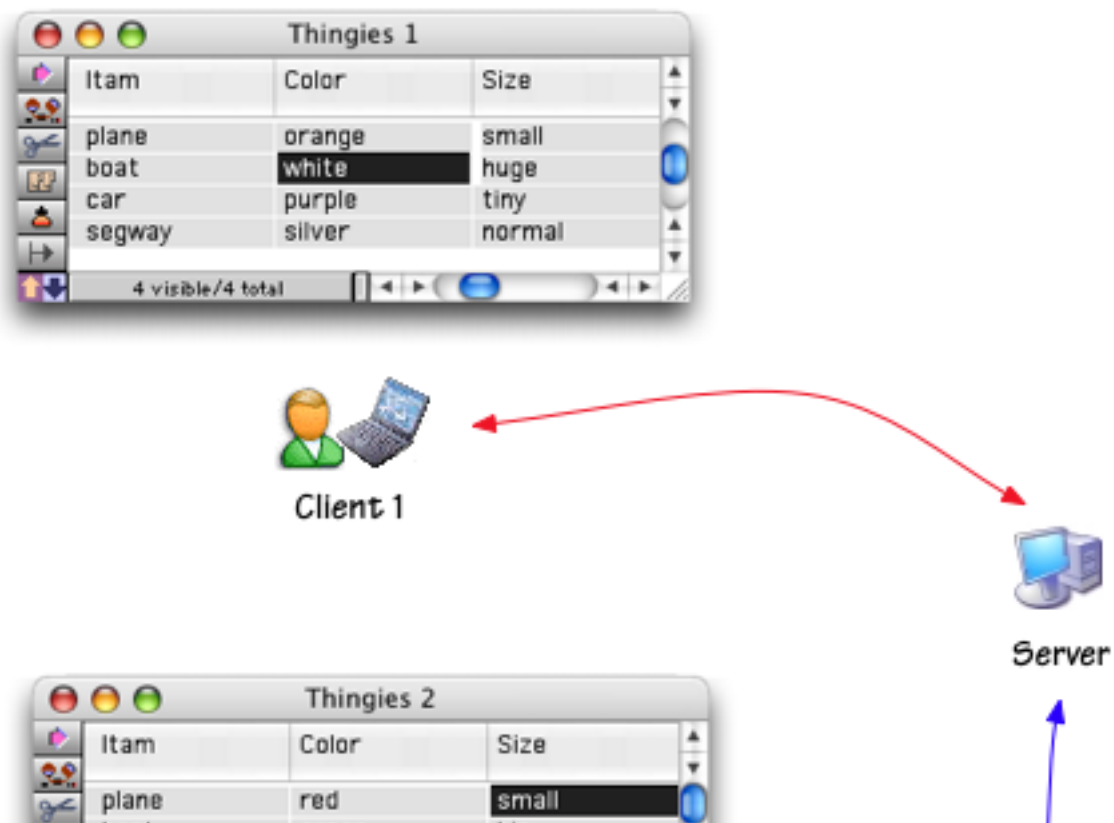
Server



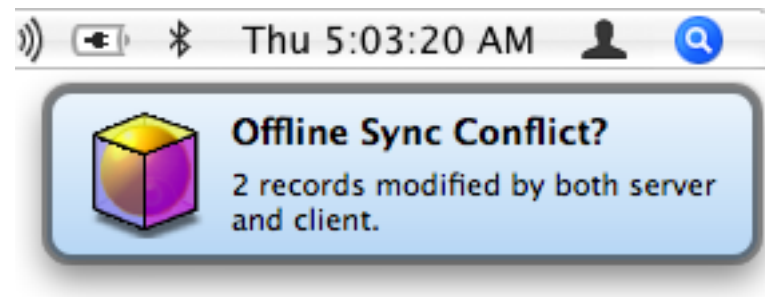
Since Client 2 is connected to the server these changes are immediately transferred to the server copy of the database. Client 1, however, still has the original data, and now he starts making changes. If Client 1 was connected to the server then the updated data would be transferred to his computer as soon as he started to change a record (see “[Editing Data and Record Locking](#)” on page 136), but since he isn’t connected that doesn’t happen. He changes the plane from blue to orange and the boat from yellow to white.



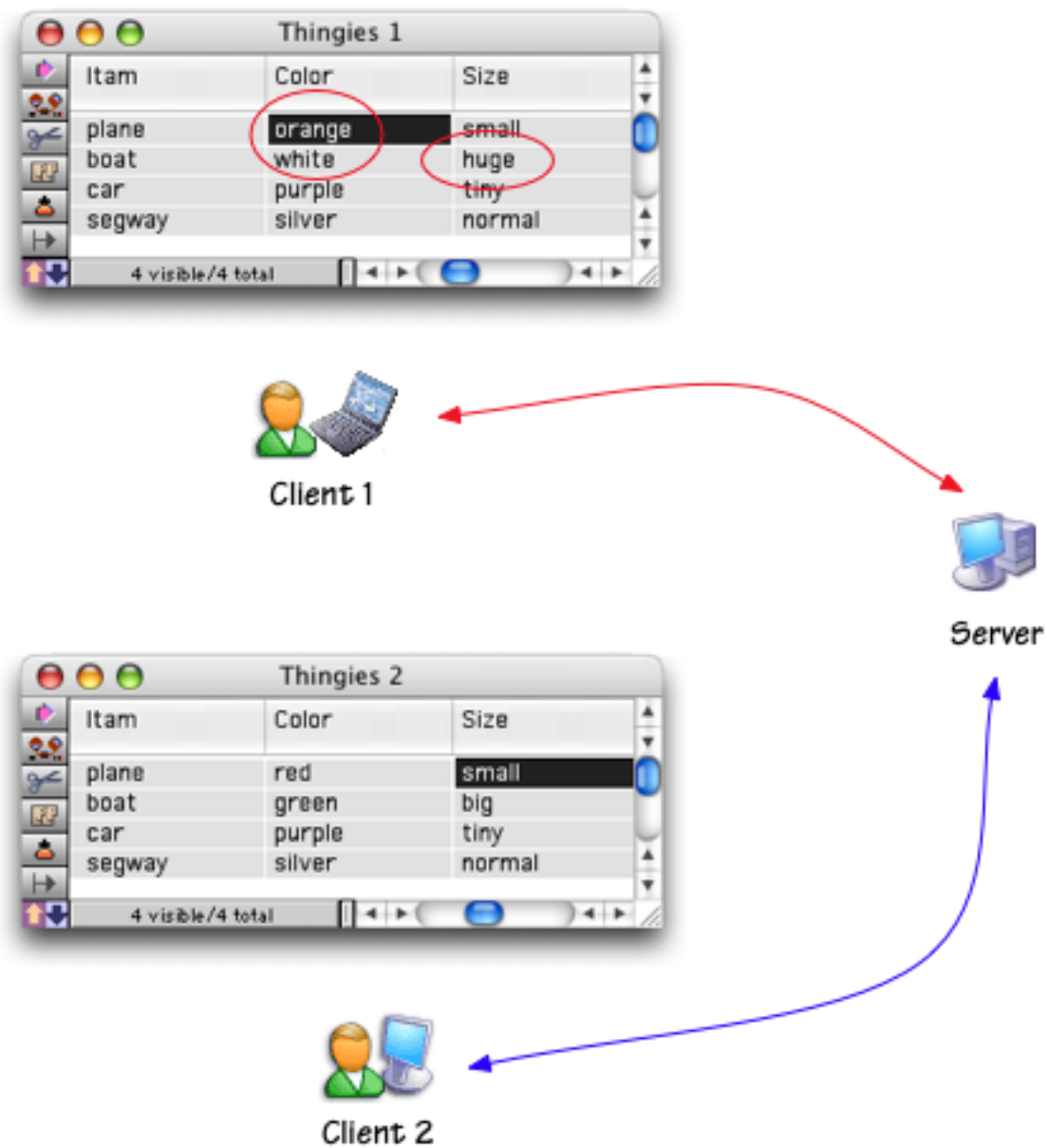
Now Client 1 returns to the office (or perhaps simply connects to the Internet) and synchronizes with the server.



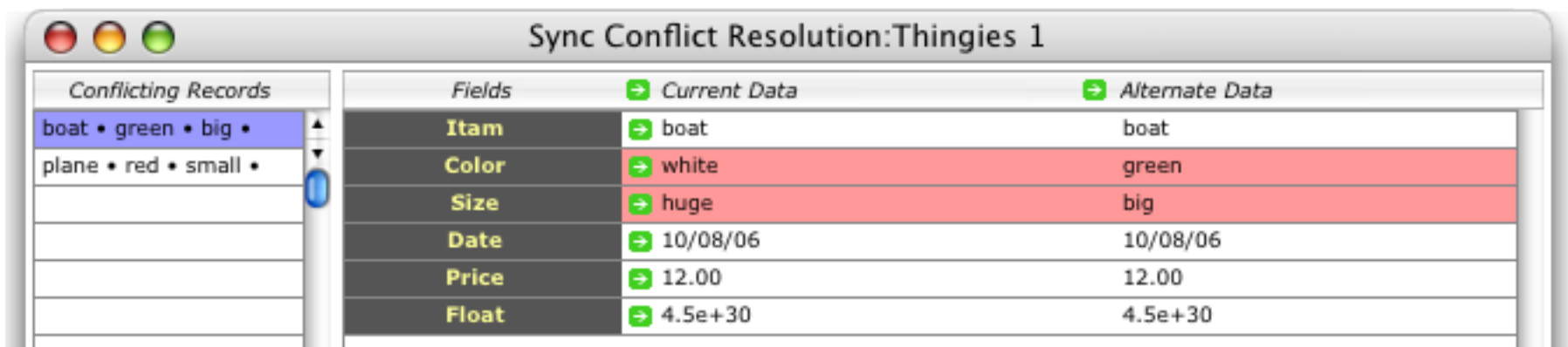
During the synchronization process Panorama notices that two records have been modified both online (by Client 2) and offline (by Client 1). Client 1 will see a notification of this conflict.



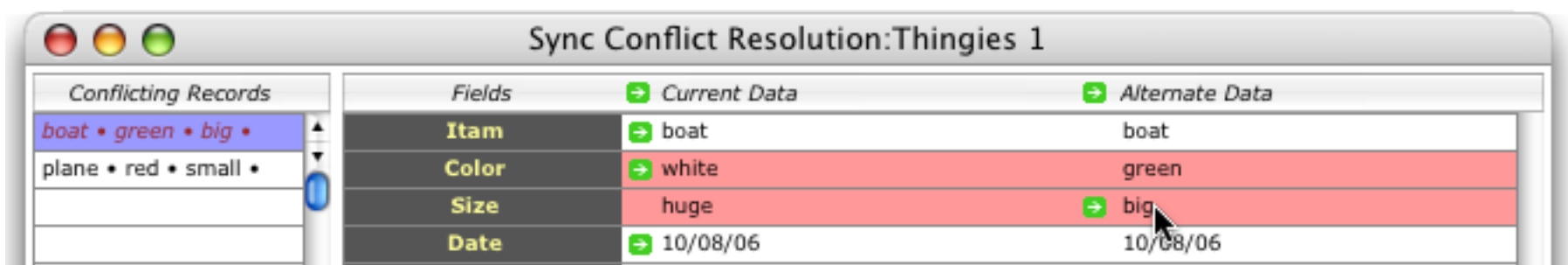
In this case the Database Sharing Options wizard has been used to give the offline client priority, so the synchronization process has automatically kept the changes made by Client 1 and discarded the changes made by Client 2.



Perhaps, however, Client 1 isn't sure that his changes should be given priority. He now has the opportunity to open the **Sync Conflict Resolution** wizard which he can use to manually override the automatic synchronization priority on a field by field basis. Here's what he'll see if he opens this wizard.

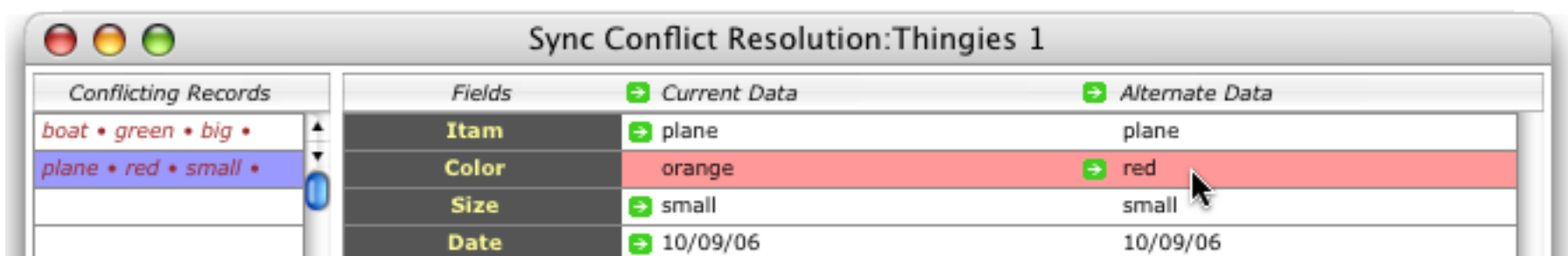


The left side of the wizard displays a list of records with conflicts, the right hand side the fields within an individual record. Client 1 can immediately see that there is some disagreement about the color and size of the boat. After further research he determines that the boat is actually just **big**, not **huge**, so he clicks on the word **big** in the Alternate Data column.

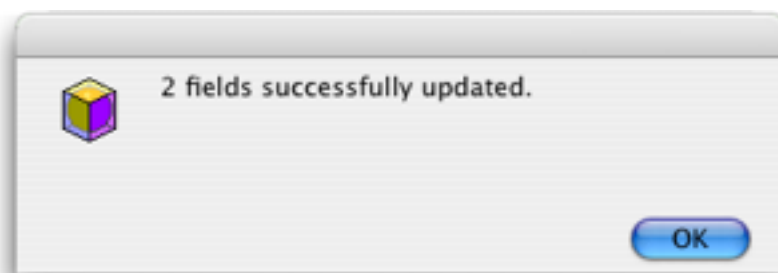


The green arrow shifts to the Alternate Data column, indicating that the normal synchronization priority should be overridden for this field.

Now Client 1 clicks on the second conflicting record, and further research indicates that the color of the plane is actually red.



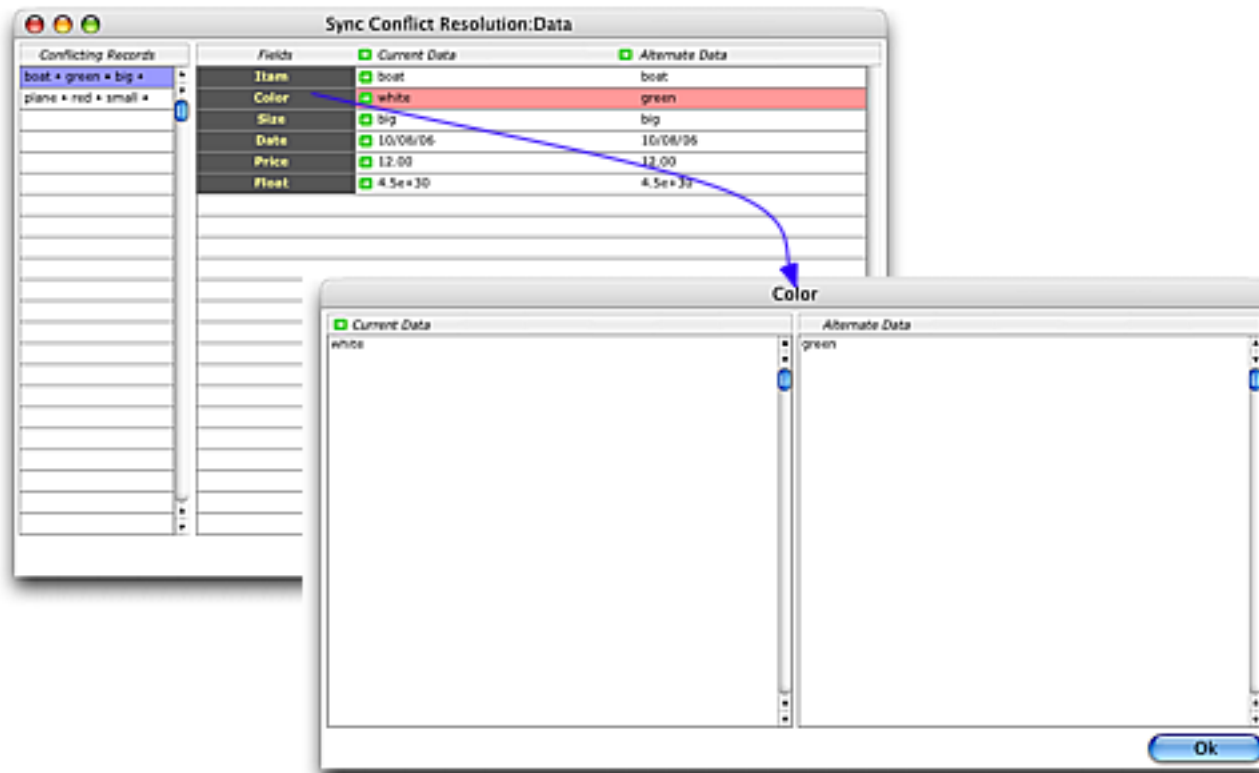
Now that Client 1 has researched and resolved all of the conflicts he can press the **Apply Changes** button to actually update the database with the resolved changes. Panorama will confirm that the changes have been made.



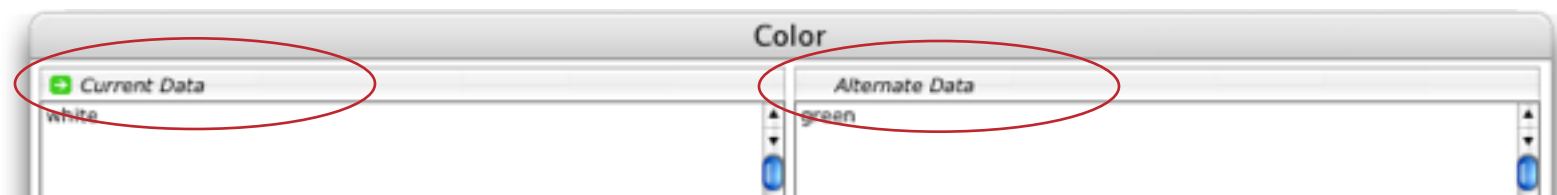
The process is complete and the **Sync Conflict Resolution** wizard can now be closed.

Reviewing Conflicts in Large Fields

The **Sync Conflict Resolution** wizard normally shows only the first line of a data cell. If a data cell contains a lot of data you can click on the field name to see the entire cell.

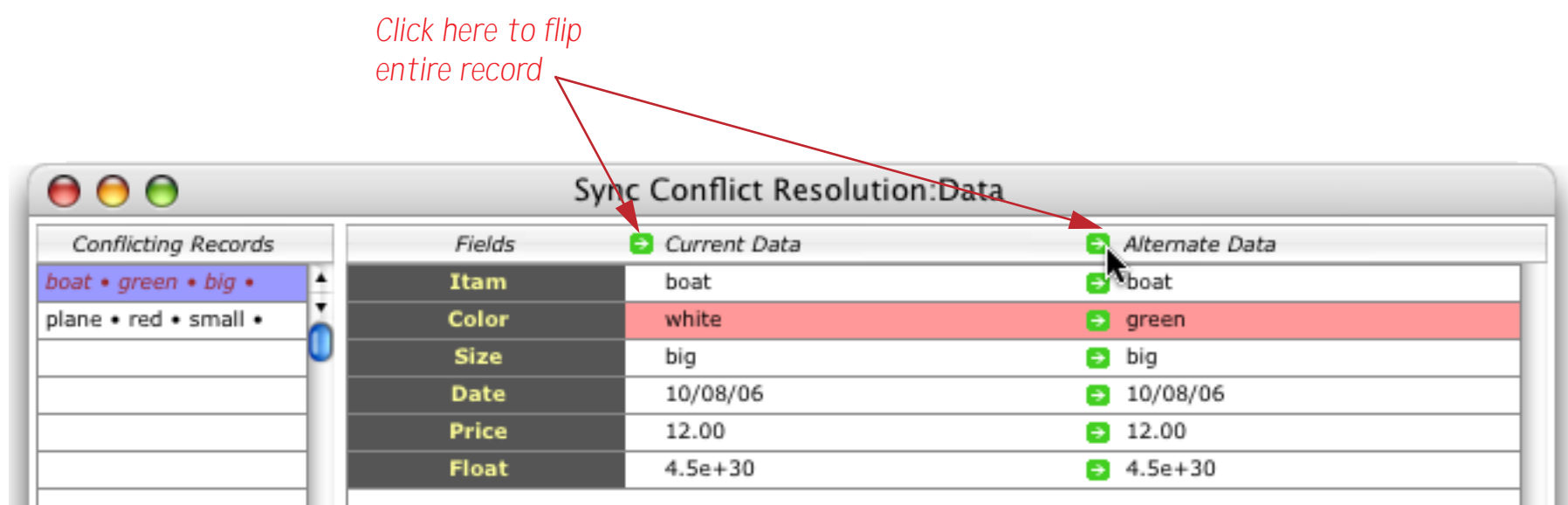


You can click in the header area to choose which version of the data you want to keep.



Overriding by Record (instead of Field)

Normally you'll click on individual fields to decide which version to keep and which to discard. However, if you click on the column header you can select all of the fields in the record at once.

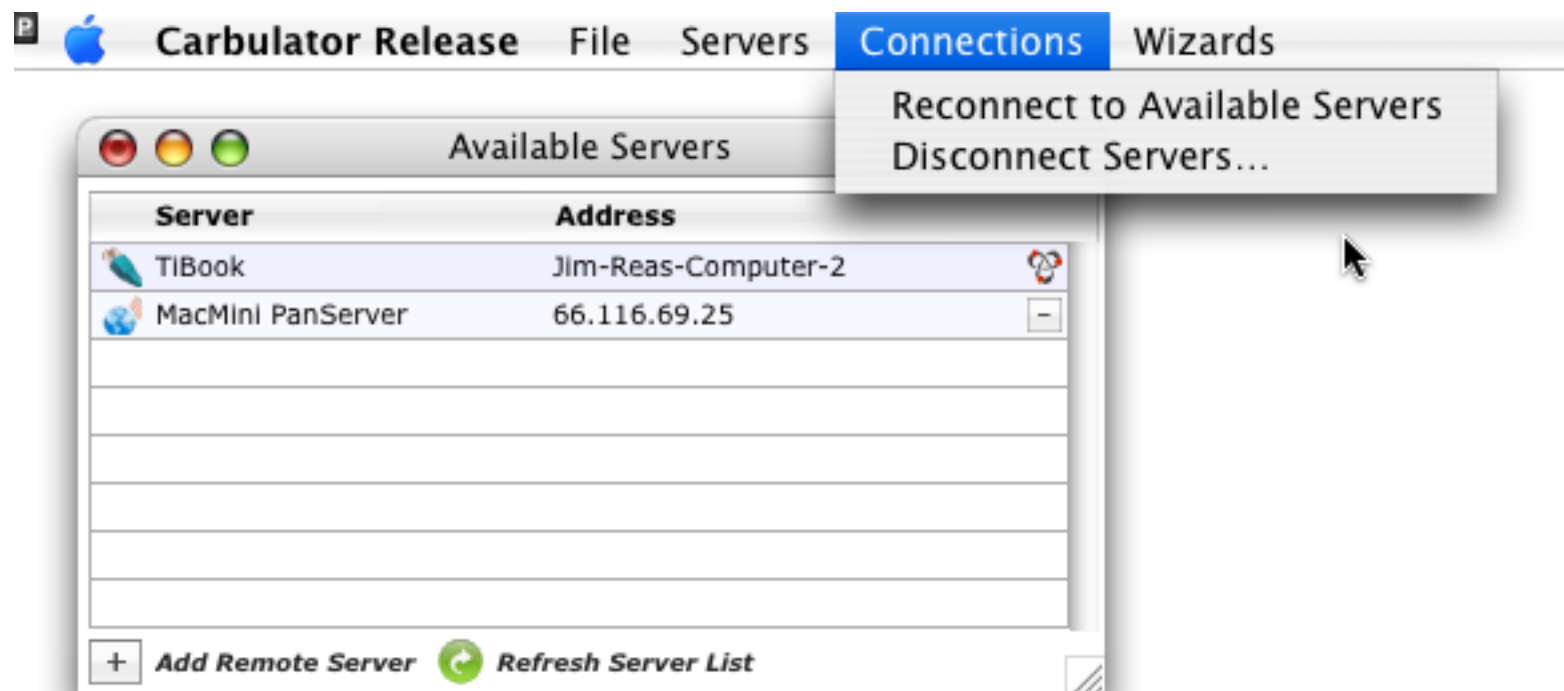


Of course you can go back and flip individual fields after you've flipped the entire record.

Making and Dropping Server Connections

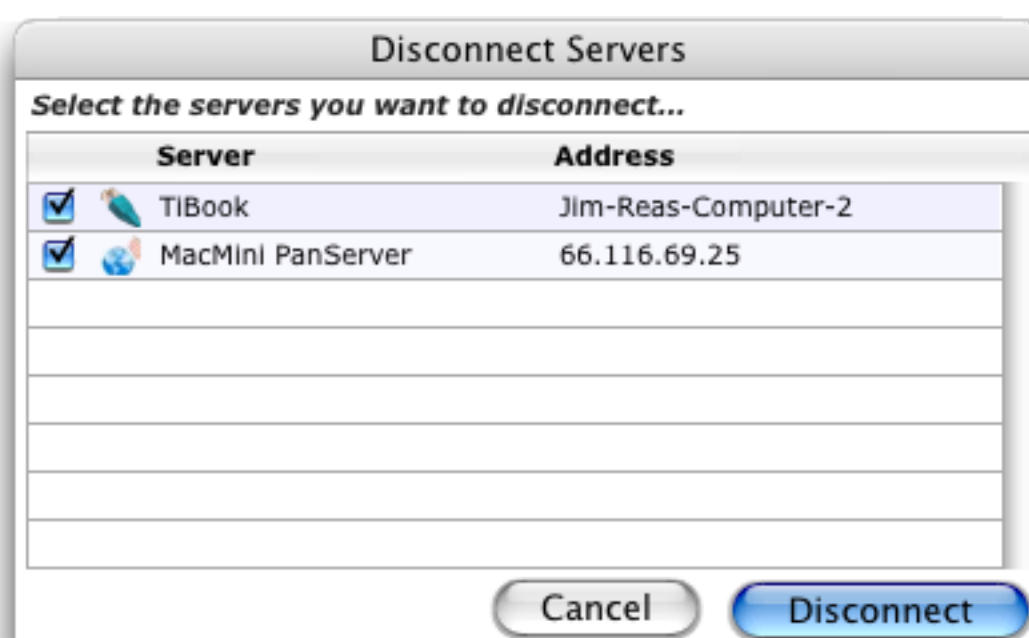
Panorama normally doesn't expect your computer to add or drop server connections while it is running. If Panorama can connect to a server when it starts up, it expects to be able to continue to connect to that server. If a server is not available when Panorama starts it won't automatically check to see if the server is available later. However you can manually tell Panorama that connections have been made or dropped with the **Available Servers** wizard (in the **Sharing** submenu of the **Wizard** menu).

To tell Panorama that a server is now online or offline you use the **Connections** menu.



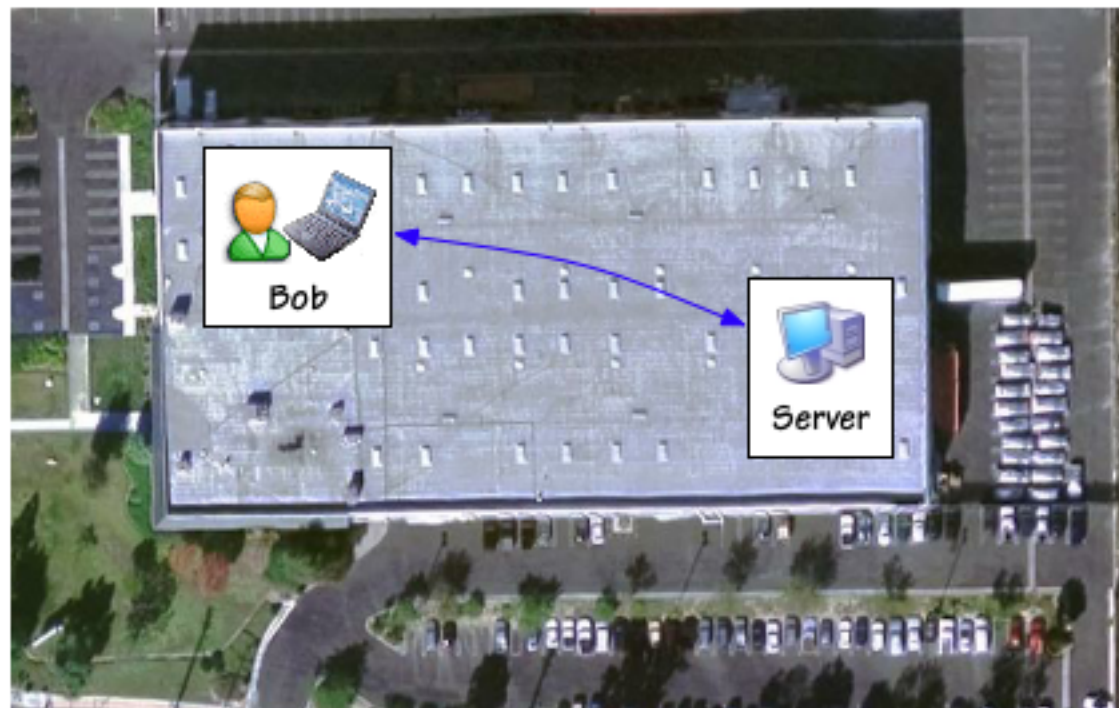
If you've been offline and are now connected to a server, use the **Reconnect to Available Servers** command. This command checks all open databases to see if they can now be connected to a server. If the server associated with a database is now available, Panorama will automatically make the connection and synchronize the database.

If you've been online and are about to go offline use the **Disconnect Servers** command. This will display a list of the available servers.

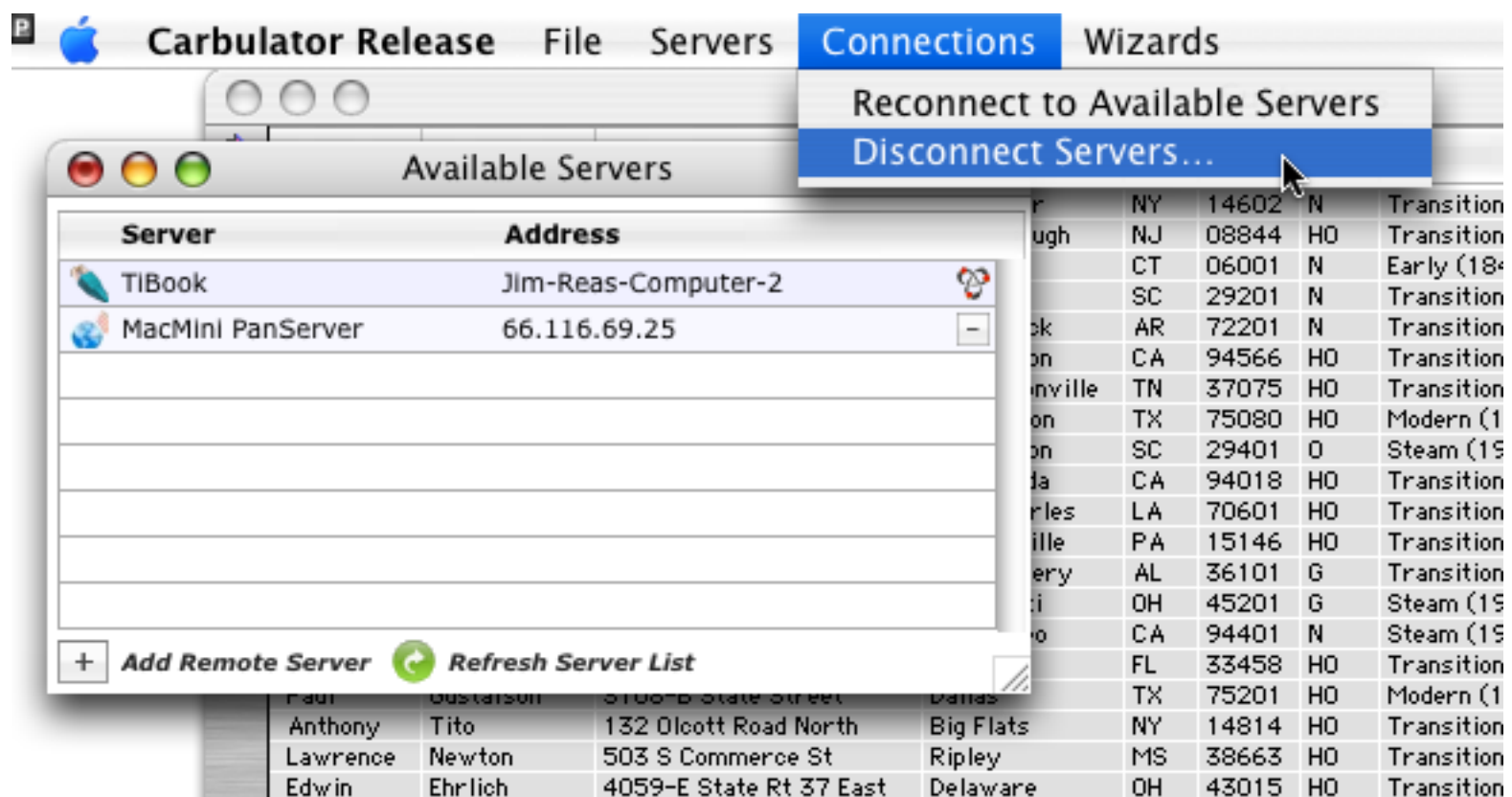


Pressing the **Disconnect** button will detach any open databases that are connected to the specified servers. You can then unplug from the network and continue using these databases in off-line mode.

An example will better illustrate the use of these commands. Suppose Bob is working on a shared Panorama database in his office



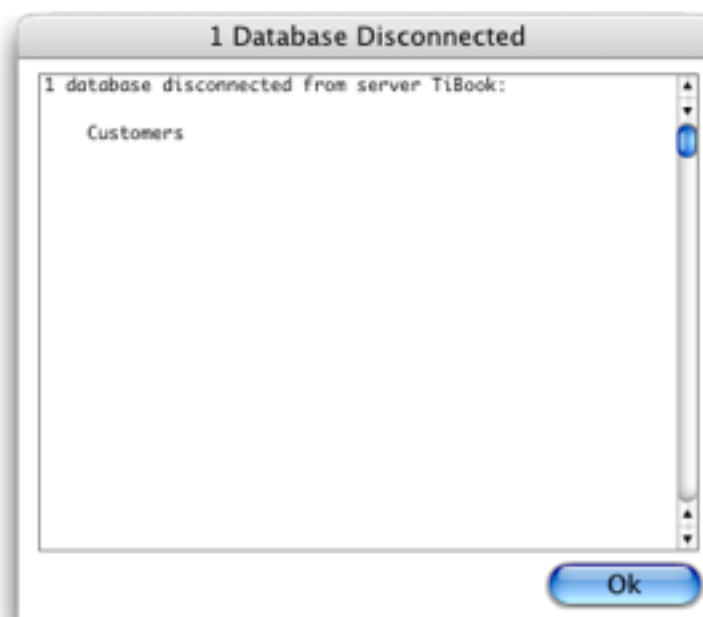
Now he has to go to an off-site meeting, but he wants to continue to be able to access his database even though the company server is not available on the internet. So Bob opens the **Available Servers** wizard and chooses the **Disconnect Servers...** command.



The wizard asks him which servers he wants to disconnect. Since he is going off-site, he simply presses the **Disconnect** button to disconnect all of them.



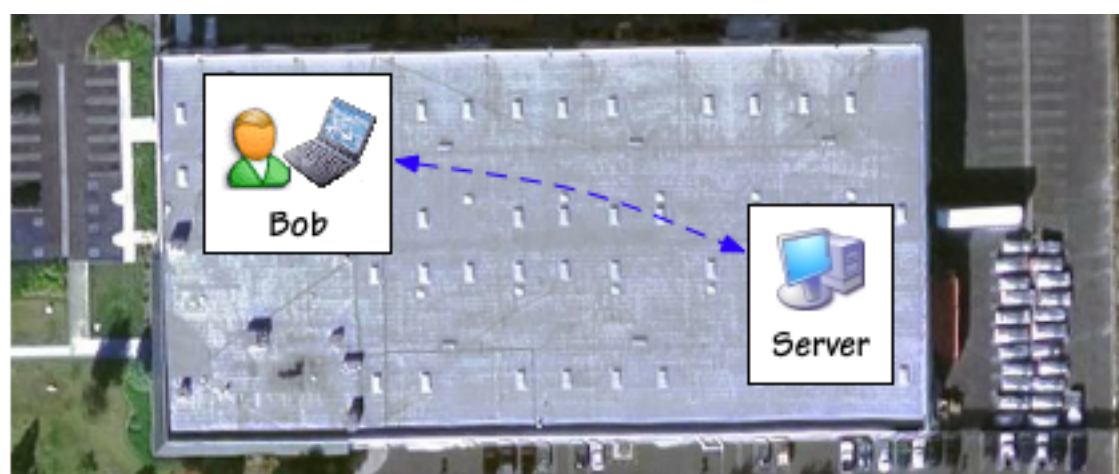
Panorama responds with a list of open databases that are now disconnected from the server.



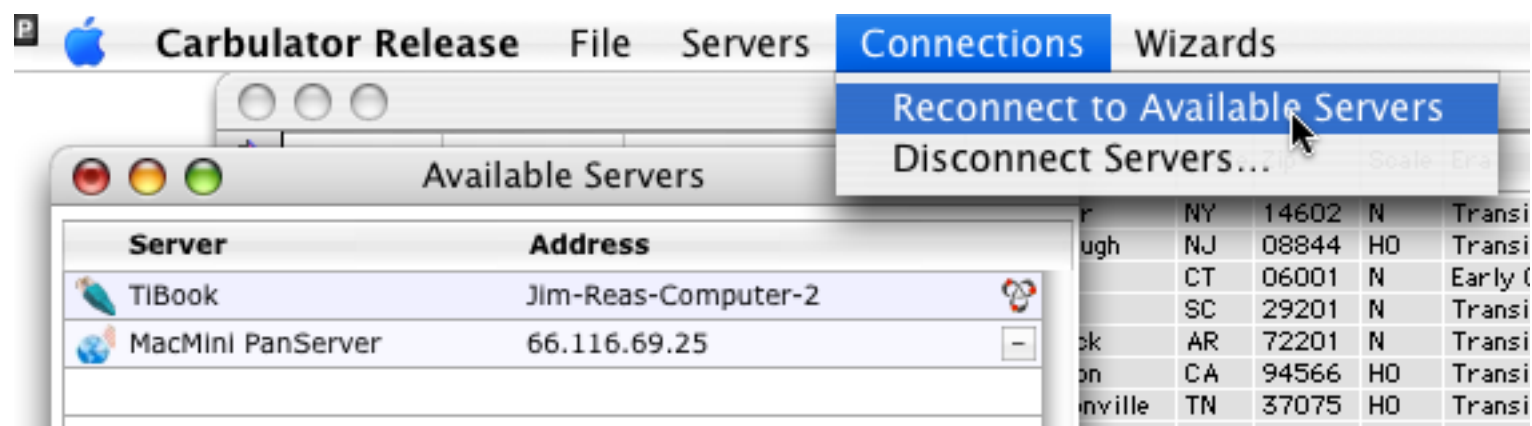
Now Bob can unplug from the network (or simply walk away if using a wireless network) and go to his off-site meeting.



While offsite Bob can still access his database, and even modify the data if that is allowed (see “[Offline Sharing Options](#)” on page 173). When he’s done with the meeting Bob comes back to the office and hooks up with the network again. However, even though the computer is physically hooked up, Panorama isn’t yet aware of the hookup, as indicated by the dotted line.



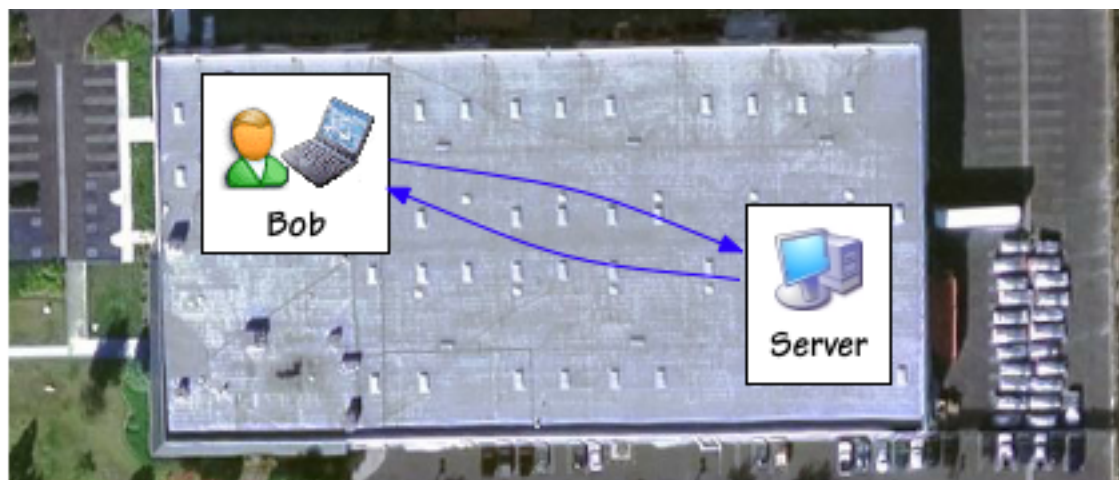
To tell Panorama about the connection, Bob opens the Available Servers wizard and chooses the Reconnect to Available Servers command.



Panorama will confirm that your databases are now connected.



It will also synchronize each of the databases as it connects them.



Bob is now ready to continue to use his databases in full on-line sharing mode.

Designing a Database Primarily for Offline Operation

Normally if there is a connection to the server, Panorama will automatically open a connection to that server and operate the database in full shared mode, with record locking. It will only drop into offline operation if there is no connection to the server. It is possible, however, to configure a database so that it operates in offline mode all the time unless specifically programmed to connect or synchronize. If a database is set up this way it will always operate offline unless specifically commanded to connect (usually by a custom procedure).

When used this way, Panorama operates similarly to an e-mail client like Apple's Mail.app or Microsoft Outlook. When using these e-mail programs you normally compose and read e-mail offline, then connect briefly to send and receive mail. In primarily offline mode Panorama works the same way — mostly working offline but occasionally connecting briefly to synchronize. For example, a database for submitting orders could be set up this way. Actual order entry would be done offline. When an order (or orders) is complete the user presses a button to submit the order to the server. Behind the scenes this button triggers a procedure which briefly connects to the server, synchronizes, then disconnects.

Configuring a Client Database for Primarily Offline Operation

To configure a database for primarily offline operation use the **Database Sharing Options** wizard. The options for configuring primarily offline operation are described below.

Automatically Connect

If this option is turned off Panorama will not automatically connect to the server when the database is opened.

Automatically Connect

This option tells Panorama to automatically attempt to connect with the server every time this database is opened. Turn this option off if this database is designed to operate primarily off-line with synchronization on demand (like an e-mail client).

When this option is turned off, a procedure can still connect and disconnect when you want it to using the **ConnectToServer** and **DropServer** statements. The **info("serverconnection")** function returns the current status. See "[Controlling and Monitoring the Server Connection](#)" on page 167 for more information about these statements.

You can also temporarily connect using the **Synchronize** command. If this command is used when the **Automatically Connect** option is turned off, Panorama will temporarily connect, synchronize, and then disconnect from the server.

Automatically Synchronize

If this is turned off Panorama will not automatically synchronize when the database is opened (including with the **ConnectToServer** statement).

Automatically Synchronize

This option tells Panorama to automatically synchronize with the server every time this database is opened. Turn this option off if this database is designed to operate primarily off-line with synchronization on demand (like an e-mail client).

If the **Automatically Synchronize** option is turned off then synchronization must be done manually. A procedure can do this with the **Synchronize** or **ForceSynchronize** statements (see "[Synchronizing](#)" on page 158).

Offline changes synchronized with server later

If this option is on then changes will be two-way synchronized with the server the next time this client synchronizes.

- Offline changes synchronized with server later
This option tells Panorama to synchronize any changes that have been made to existing data the next time the database is synchronized with the server.
- Offline changes are local only (*except for new records*)
This option tells Panorama NOT to synchronize any changes that have been made to existing data with the server. These changes will be saved locally but will not be available to any other user.

This is the normal setting.

Offline changes are local only

In this mode a user can change data, but changes are local only and will be erased if this record is changed by another user and then synchronized.

- Offline changes synchronized with server later
This option tells Panorama to synchronize any changes that have been made to existing data the next time the database is synchronized with the server.
- Offline changes are local only (*except for new records*)
This option tells Panorama NOT to synchronize any changes that have been made to existing data with the server. These changes will be saved locally but will not be available to any other user.

Usually this option would be used only if the automatically connect/synchronize options are turned off. In that case it could be used to create a database in which new records can be added but old records can never be changed on the server (or only changed under program control). However, users could add their own local changes (for example adding notes).

For example, consider the order submission database discussed earlier. Once an order is submitted to the server it should not be changed. Turning on this option allows new orders to be created and submitted, but orders that have been previously submitted can no longer be modified (or at least these modifications cannot be sent to the server, just as you can't make changes to an outgoing e-mail once you have submitted it).

Allow deleting records

Panorama normally doesn't allow existing records to be deleted when the database is offline — deleting records is only allowed when the database is connected to the server. (You can, however, delete new records that haven't been submitted to the server yet.)

- Allow deleting records (*will not be removed from server*)
This option allows existing records to be deleted from the local copy of the database even if there is no connection to the server. Only the local copy of the database is affected — these records are not deleted from the server.

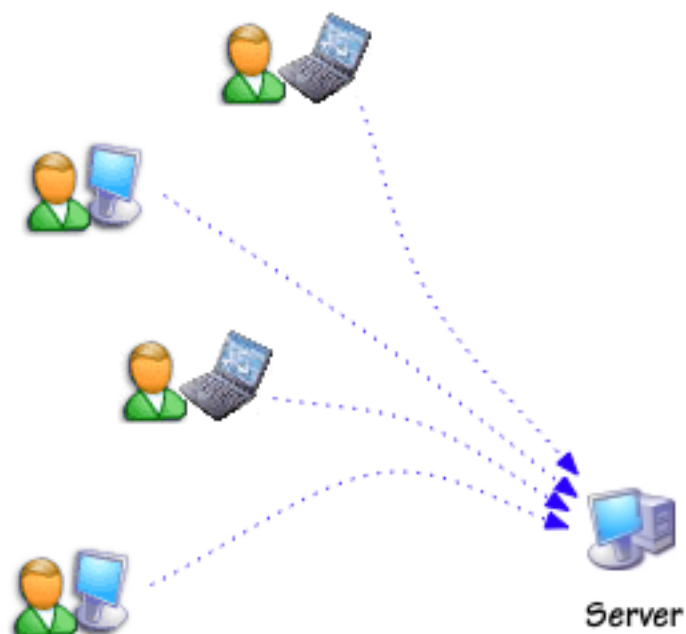
However, if the **Allow deleting records** option is turned on then you *can* delete records. However, these records will only be deleted from the local copy of the database. They will not be deleted from the server, either immediately or later when the database is synchronized.

For example, consider the order submission database discussed earlier. Once an order is submitted to the server it should not be changed, but it still hangs around in the client database. Turning on the **Allow deleting records** option allows an order to be deleted from the local client copy of the database after it is submitted. Out of sight, out of mind. You might even want to automate this with a procedure, so that orders are automatically deleted from the client database as soon as they are submitted. Though the orders are deleted from the client that submitted them, they are still available on the server where they can be accessed by other clients (see next section).

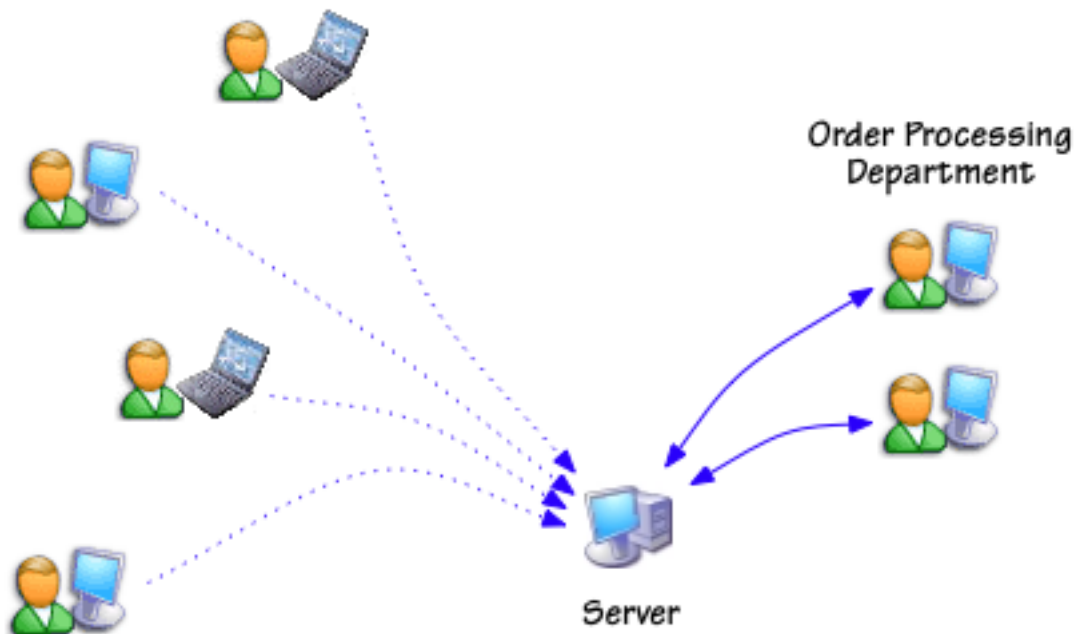
Mixing Offline and Online Clients

The **Database Sharing Wizard** configures only the local copy of the database, on that machine. This means that it is possible to have clients on different machines with different options. For example, some computers might have databases that operate in primarily offline mode, while others are allowed full online access to the database.

For example, consider the hypothetical order submission database discussed in several locations earlier in this chapter. In this application a number of clients create orders offline and submit them to the server.



But how are these orders processed? The solution is to set up the clients in the order processing department so that they have full online access to the database.

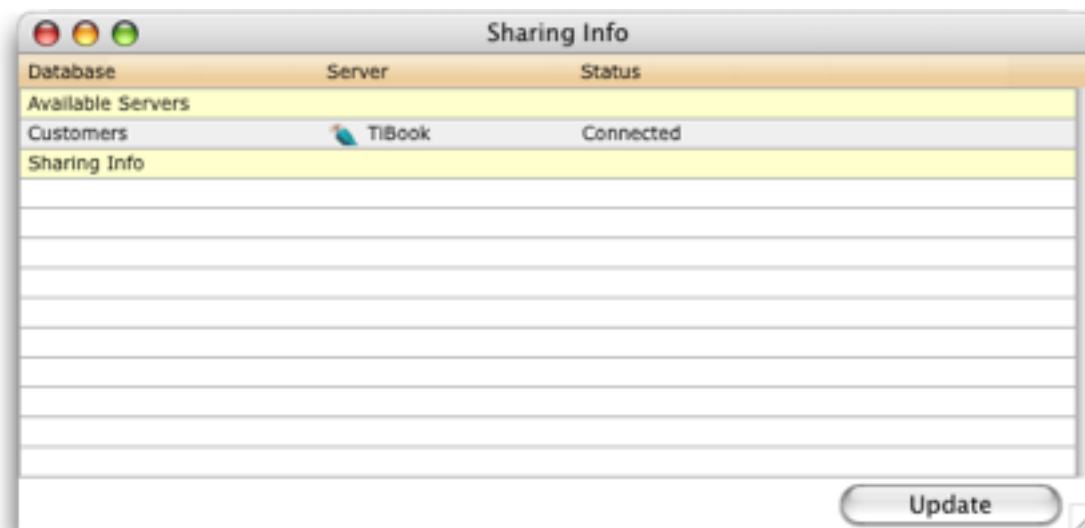


As orders come in from the off-line clients they are processed and fulfilled by operators using the on-line clients.

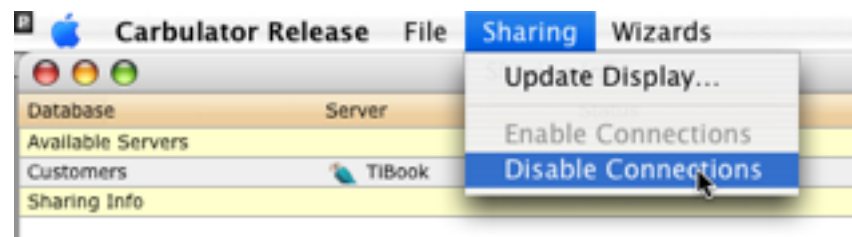
Simulating Offline Operation (for testing)

In the process of creating the Panorama Server's offline capabilities we needed to connect and disconnect Panorama from the server dozens and even hundreds of times. Since plugging and unplugging the ethernet cable that many times would be very tedious (and also knock out all of our other network connections — web, e-mail, etc.) we designed a method for simulating a disconnected network connection within Panorama. This method is available for testing your own offline database applications.

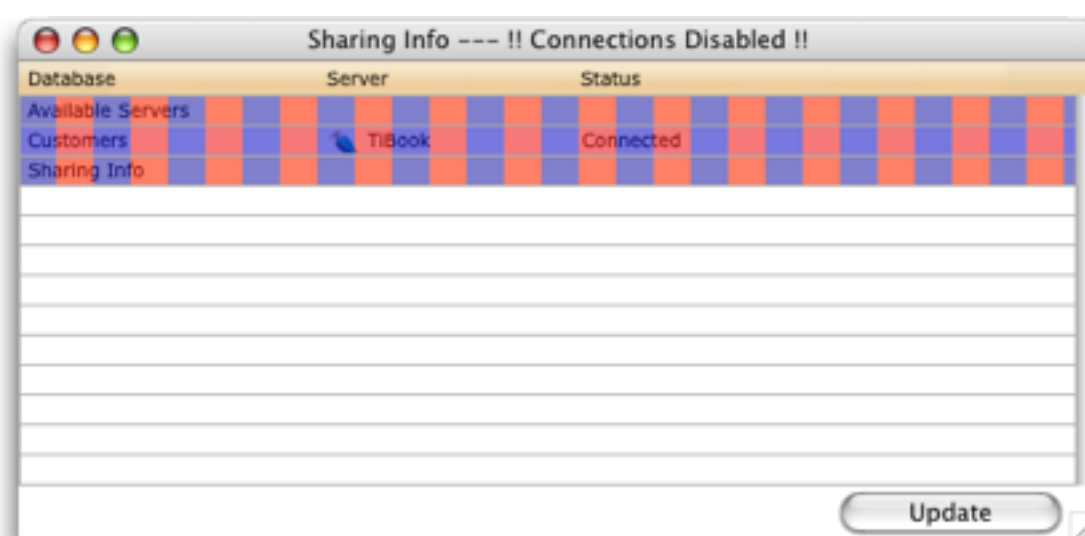
Step 1 — Open the **Sharing Info** wizard.



Step 2 — Select the **Disable Connections** command in the **Sharing** menu. (Alternate shortcut: Hold down the **Control** key and click anywhere in the body of the wizard.)



The **Sharing Info** wizard display will change to indicate that connections are currently disabled.



To turn the connections back on again either use the **Enable Connection** command in the **Sharing** menu, hold down the **Control** key and click anywhere in the body of the wizard, or simply close the **Sharing Info** wizard.

Chapter 5: Web Publishing



This chapter explains how to create and use Panorama databases for web publishing, and introduces the fundamentals of designing a web application. It assumes that you already have a working Panorama Server available (see Chapter 2).

Web Publishing a Database

The first step in creating a shared database is to create a single user database. This database can be old or new, empty or full. For this example we'll use a database called [Customers](#).



Customers

If it's not already open, double click on the database to open it.

| First | Last | Address | City | State | Zip | Scale | Era | Railroad | Email |
|----------|-----------|-------------------------|----------------|-------|-------|-------|------------------------------|---------------------------|--------------------------|
| William | Reed | 28 Huntington Hills | Rochester | NY | 14602 | N | Transition (1940's - 1950's) | Burlington Northern (BN) | william265@pimed.com |
| Norman | Crosby | 29 Independence Drive | Hillsborough | NJ | 08844 | HO | Transition (1940's - 1950's) | Erie | norman_crosby@amug.o |
| Frank | Atkinson | 7 Chelsea Place | Avon | CT | 06001 | N | Early (1840's - 1910's) | New York Central (NYC) | fatkinson41@jnlcom.com |
| Warren | Ware | 6315 Winyah Drive | Columbia | SC | 29201 | N | Transition (1940's - 1950's) | Pennsylvania | warren901@bmg.org |
| Jonathan | Snyder | 108 Dickson Dr | Little Rock | AR | 72201 | N | Transition (1940's - 1950's) | Western Pacific (WP) | jonathan_snyder@telusp |
| Donald | Davies | 2350 Santa Rita Road | Pleasanton | CA | 94566 | HO | Transition (1940's - 1950's) | New York Central (NYC) | donald265@sprntmail.c |
| Herbert | Bothman | 101 Oakvale Ct | Hendersonville | TN | 37075 | HO | Transition (1940's - 1950's) | Santa Fe (ATSF) | herbert860@best.com |
| Edwin | Pollock | 2401 Buttercup Drive | Richardson | TX | 75080 | HO | Modern (1960's - Present) | Montana Rail Link | edwin_pollock@synergo |
| Kenneth | Peterson | 910 Melrose Dr | Charleston | SC | 29401 | O | Steam (1920's - 1930's) | Chicago & North Western (| kenneth.peterson@telep |
| Aaron | Camp | 155 Dolphine Ave. | El Granada | CA | 94018 | HO | Transition (1940's - 1950's) | New York Central (NYC) | acamp60@bigfoot.com |
| Carl | Wolfson | 1675 Campfire Rd. | Lake Charles | LA | 70601 | HO | Transition (1940's - 1950's) | Pennsylvania | carl_wolfson@yahoo.co |
| Robert | Anderson | 109 Pagoda Drive | Monroeville | PA | 15146 | HO | Transition (1940's - 1950's) | Union Pacific (UP) | randeron69@amgen.co |
| Arthur | Clark | 3308 Montezuma Road | Montgomery | AL | 36101 | G | Transition (1940's - 1950's) | Boston & Maine | arthur_clark@catskill.n |
| Steven | Phelps | 11970 Brookway Dr. | Cincinnati | OH | 45201 | O | Steam (1920's - 1930's) | Santa Fe (ATSF) | steven927@southcoast.r |
| Thomas | Jensen | 457 Virginia Ave. | San Mateo | CA | 94401 | N | Steam (1920's - 1930's) | Seaboard Coast Line | thomas_jensen@aol.com |
| Gary | Christian | 109 Palomino Drive | Jupiter | FL | 33458 | HO | Transition (1940's - 1950's) | Norfolk & Western | gary_christian@callaha |
| Paul | Gustafson | 3108-B State Street | Dallas | TX | 75201 | HO | Modern (1960's - Present) | Milwaukee (MILW) | paul_gustafson@avana.n |
| Anthony | Tite | 132 Olcott Road North | Big Flats | NY | 14814 | HO | Transition (1940's - 1950's) | New York Central (NYC) | anthony218@concentric |
| Lawrence | Newton | 503 S Commerce St | Ripley | MS | 38663 | HO | Transition (1940's - 1950's) | Northern Pacific (NP) | lawrence303@pobox.co |
| Edwin | Ehrlich | 4059-E State Rt 37 East | Delaware | OH | 43015 | HO | Transition (1940's - 1950's) | New Haven (NH) | edwin197@ix.netcom.c |
| Daniel | Oswald | 201 Pinebrook Boulevard | New Rochelle | NY | 10801 | HO | Steam (1920's - 1930's) | Western Pacific (WP) | doswald40@mindspring. |
| George | Mueller | 181 Baboosic Lake Road | Merrimack | NH | 03054 | HO | Steam (1920's - 1930's) | Southern | george538@iguest.net |
| Robert | Bishop | 3223 Floyd Ave | Richmond | VA | 23218 | O | Transition (1940's - 1950's) | CSX | robert_bishop@sprynet. |
| Jerome | Baker | 1518 Longview St. | Madison | WI | 53701 | S | Modern (1960's - Present) | Western Pacific (WP) | jerome_baker@gate.net |
| John | Kroll | 400 Carroll St | Brooklyn | NY | 11201 | HO | Transition (1940's - 1950's) | Pennsylvania | john_kroll@forinternet.r |
| Frank | Block | 142 Via Lantana | Aptos | CA | 95001 | HO | Transition (1940's - 1950's) | Conrail | frank_block@asemp.org |

So far this is a garden variety single user database. You can sort, select, add data, delete data — it all happens privately on your computer and has nothing to do with anyone else on another computer or the web. In other words, it's not shared or web published, but we're about to change that.

Preparing the Web Layout and Logic

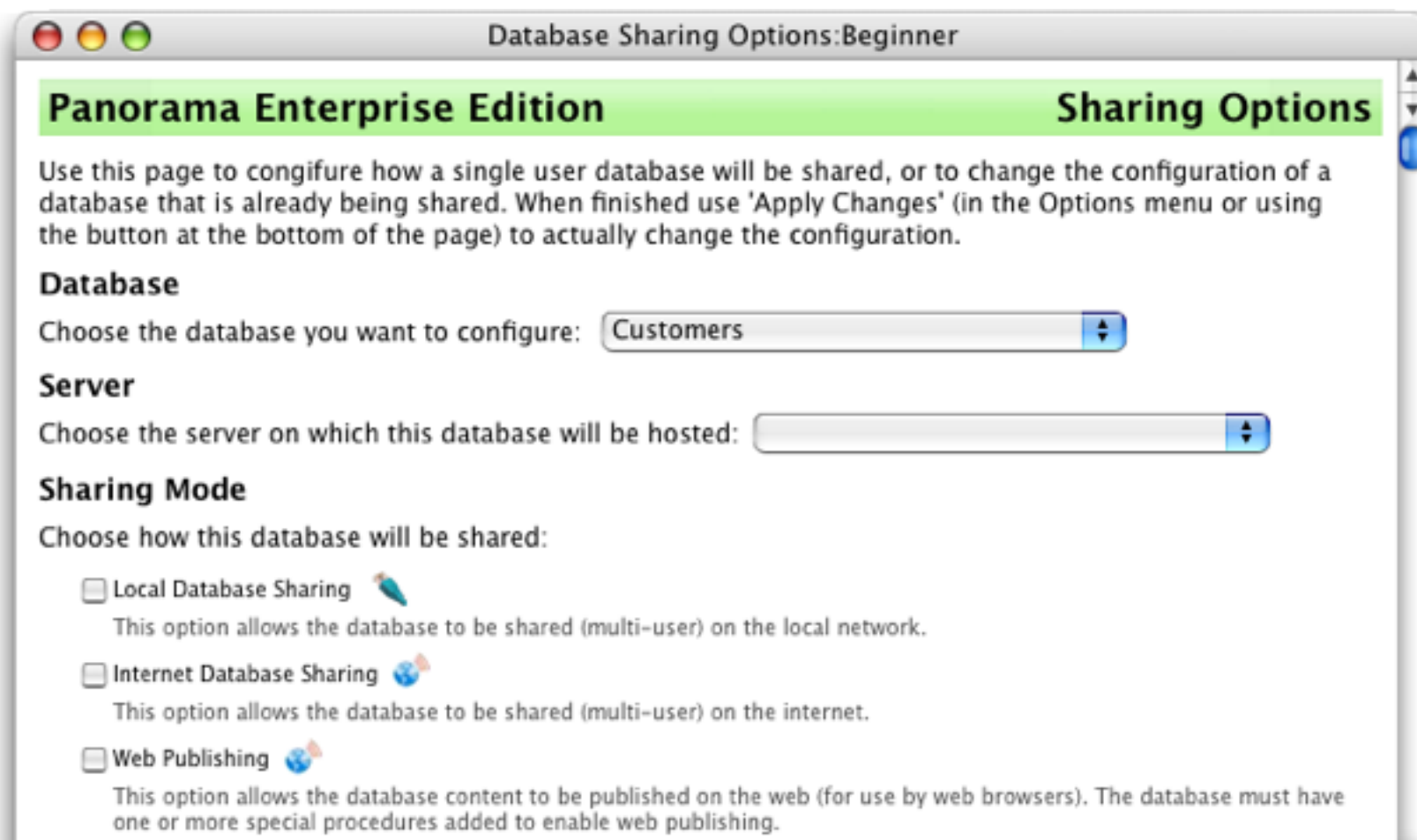
Before a database can be used for web publishing the web layout (forms & tables) and logic (how database activity flows from web page to web page) must be designed and implemented. There are three components to this preparation:

| Forms (see “ Web Forms ” on page 231) | Tables (see “ Web Tables ” on page 277) | Logic & Programming (see “ Web Programming 101 ” on page 317) |
|--|--|---|
|  |  |  |

You can do this preparation either before or after the database has been uploaded to the server (or both). In the following sections we’ll assume that the **Customers** database already has all of the forms, tables, and web logic set up. This prep process can be as simple as setting up a few forms and filling in a few dialogs for a simple web database, or it can take months of programming for a complex web application.

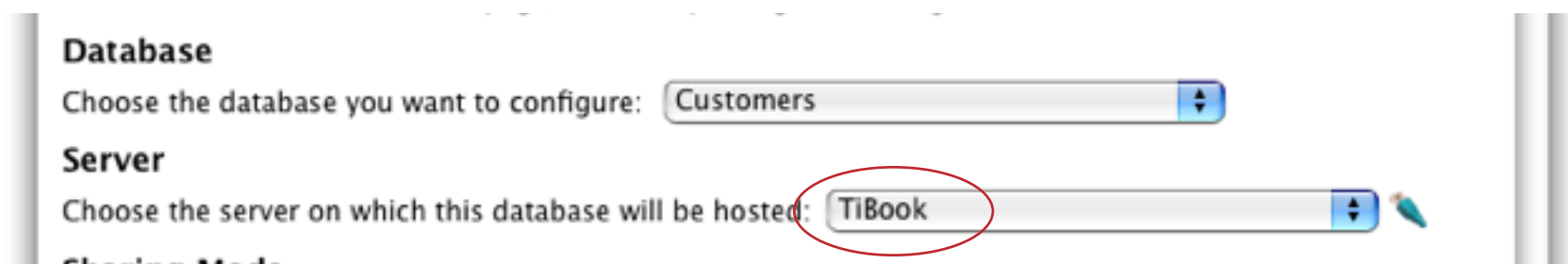
Uploading the Database to the Server

To start the process of changing this database into a web published database open the **Database Sharing Options** wizard (in the **Sharing** submenu of the **Wizard** menu).



Start at the top of this form and work your way down. The first choice is the database. If it doesn’t already list the database you want to convert then select it from the pop-up menu.

The next choice is the server. Use the pop-up menu to select the server that will host this shared database. (If no server appears in the pop-up menu then see “[Client Configuration](#)” on page 58.)

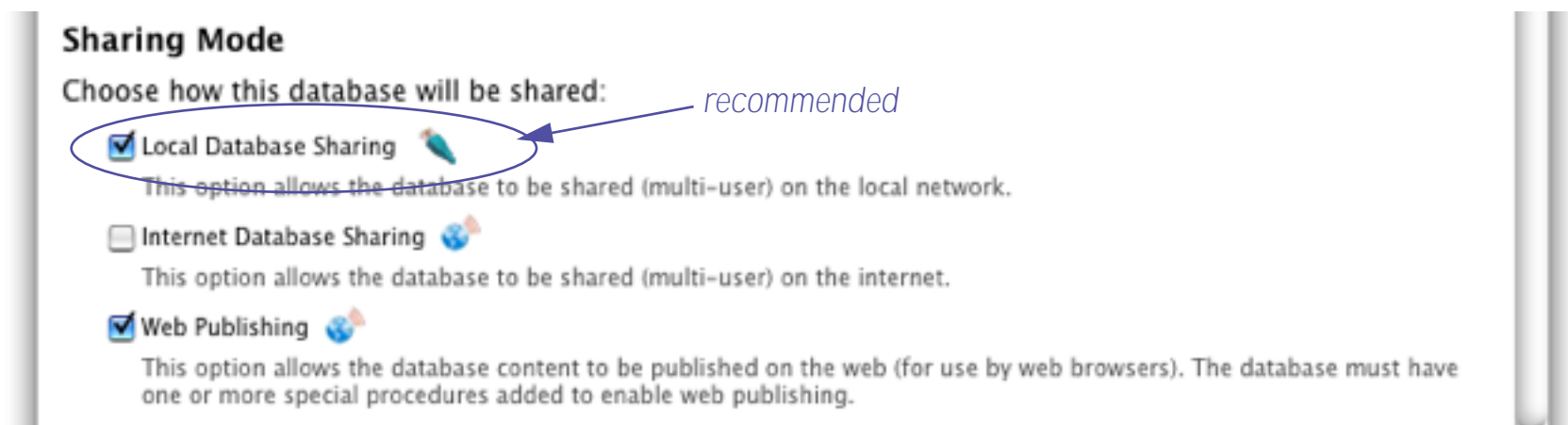


The next choice is the sharing mode. At a minimum you must check the **Web Publishing** option.



Database Sharing and Web Publishing

ProVUE highly recommends that whenever you web publish a database you also make it sharable (at least locally).



There are three reasons for this recommendation. First, when Panorama makes a database sharable it adds a unique ID number to every record (and any new records that get added later). This makes it much simpler to generate a URL that uniquely links to a particular record in the database. If your database isn't shared you'll have to rely on one or more fields for this — hopefully there is a combination of fields that can be guaranteed to uniquely identify a record. Using a shared database eliminates the possibility of any confusion or duplication.

The second advantage of using a shared database is that it makes it much easier to manage the data in the database. When you open the database on the client it will automatically synchronize with the server, and then allow you to make changes on the live database. You don't have to worry about uploading or downloading data or the possibility of the local database being out of sync with the server — Panorama manages that for you.

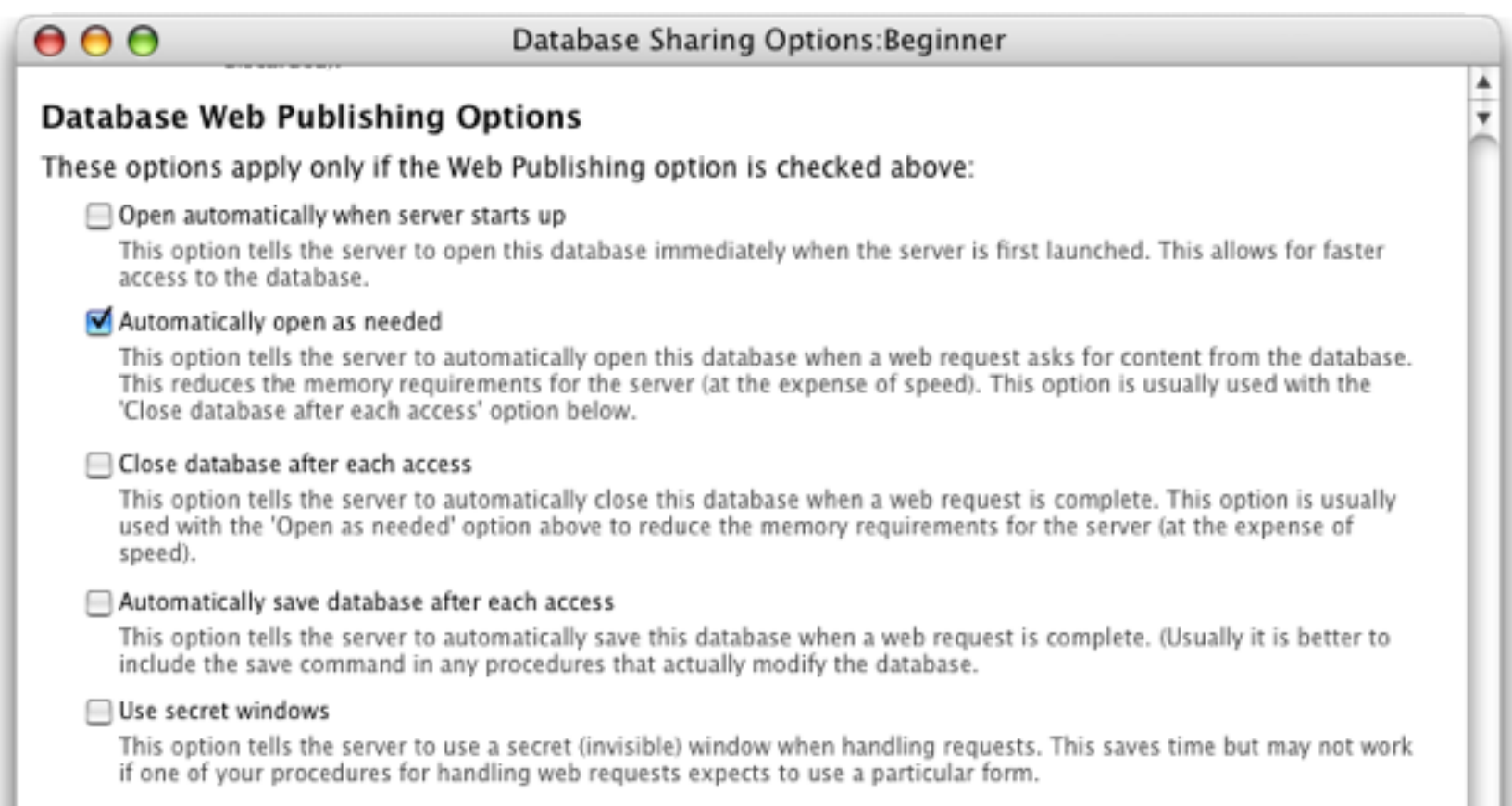
The third advantage of using a shared database is that only shared databases can take advantage of the server's journal capability for recovering the database after a power failure or crash (see “[Handling Interruptions in Server Operation \(Crash Recovery\)](#)” on page 153). This allows your web databases to operate as fast as possible without any worry about losing data. Journalling is automatically enabled when you turn on the sharing option.

If you have a specific reason not to share the database, that's fine, web publishing will still work though without the advantages listed above. (Of course making the database sharable doesn't mean that you actually have to share the database with anyone else — only users that you have given copies of the client database to will be able to access the shared database.) Even if you have purchased only an Enterprise Server web sharing license you can still make the database sharable — in that situation the server will only allow one user to share the database at a time (probably you as administrator).

For the most part the design and operation of web published databases is the same whether database sharing is turned on and off. We will point out any situations where this is not the case as they occur through the next few chapters.

Web Publishing Options

Towards the bottom of the **Database Sharing Options** wizard are a handful of options that allow you to customize how the server works with your web shared databases.



You can use the default settings or modify them as needed.

Open automatically when server starts up. When this option is checked, the database will be opened automatically when the server starts up, before any requests come in. The database will remain open permanently (as long as the server is running). This allows for faster access to the database, but may increase the amount of memory required on the server computer.

Automatically open as needed. When this option is checked, the database will not be opened until it is accessed, either by a web request or by a Panorama client for database sharing.

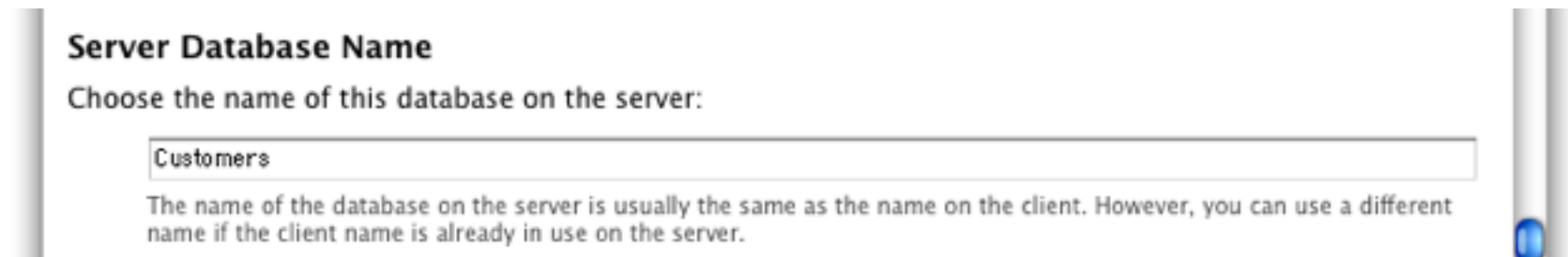
Close database after each access. When this option is checked, the database will be closed after each web request (unless it is currently being shared by one or more Panorama users).

Automatically save database after each access. When this option is checked, the server will automatically save the entire database after each web request. This option will save the database whether or not the database has actually been modified. If the database is shared you should leave this option off — saving is handled automatically in that situation.

Use secret windows. When this option is checked, the server will open the database without any windows. If the database is not shared, this makes access a bit faster. If the database is shared, it's better to leave this option off.

Server Database Name

The name of the database doesn't have to be the same as the original database name. You can type in a different name here.



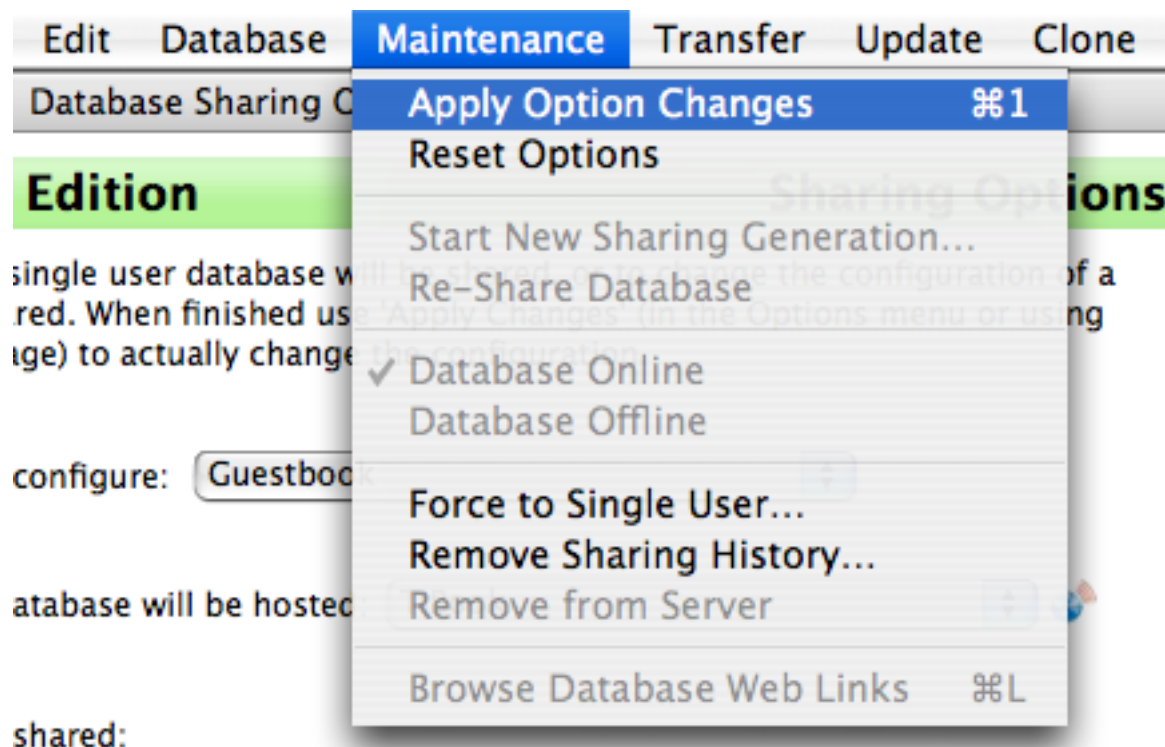
If the database name contains non “url-friendly” spaces or punctuation, it's a good idea to pick a server name that eliminates these characters. (For example [Real Estate Listings](#) might become [RealEstateListings](#), while [Products & Services](#) might become [ProductsServices](#).)

Uploading the Database to the Server

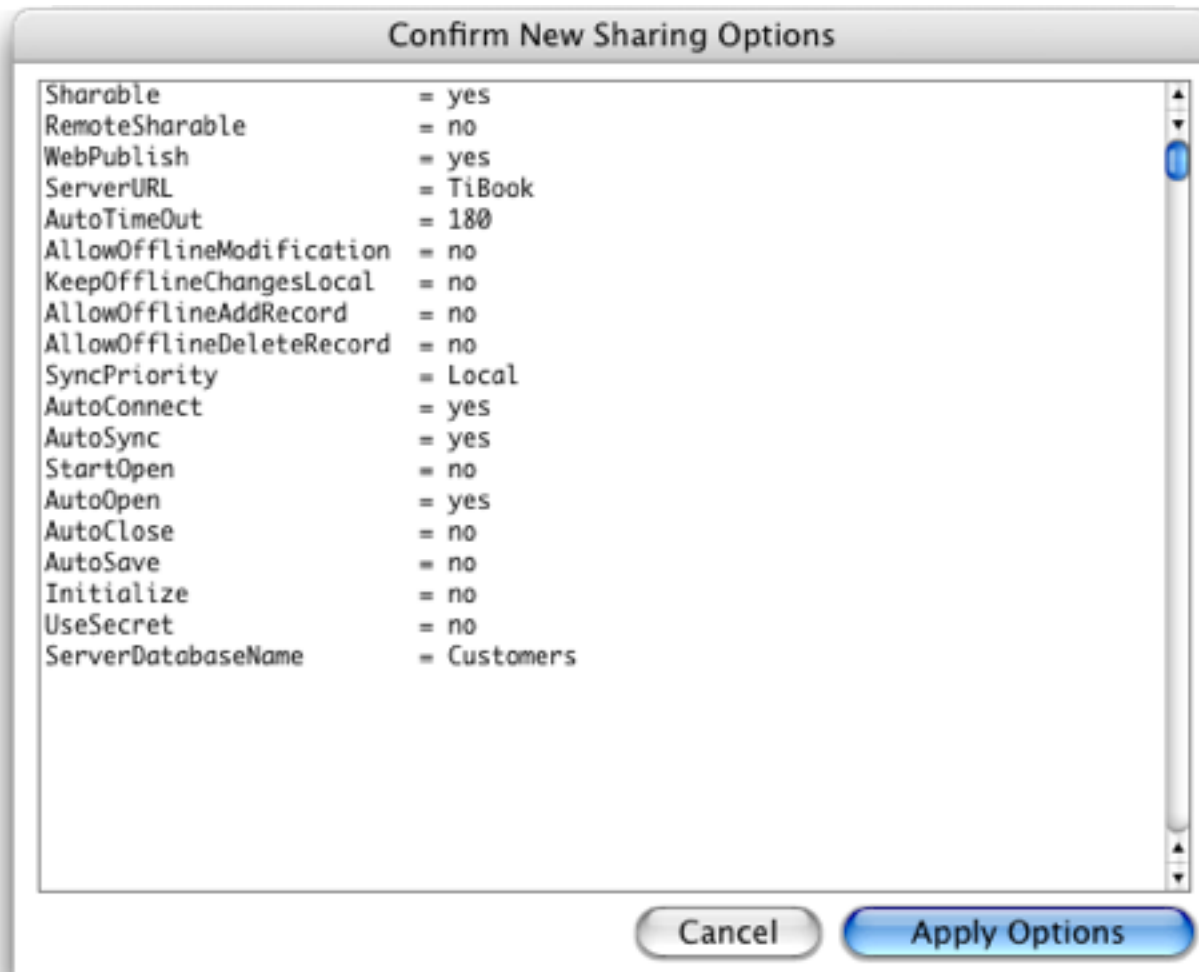
Once all of the options are set, it's time to actually upload the database to the server. Just scroll to the bottom and press the **Apply Changes** button



or choose **Apply Changes** from the Maintenance menu (or press **Command-1**).



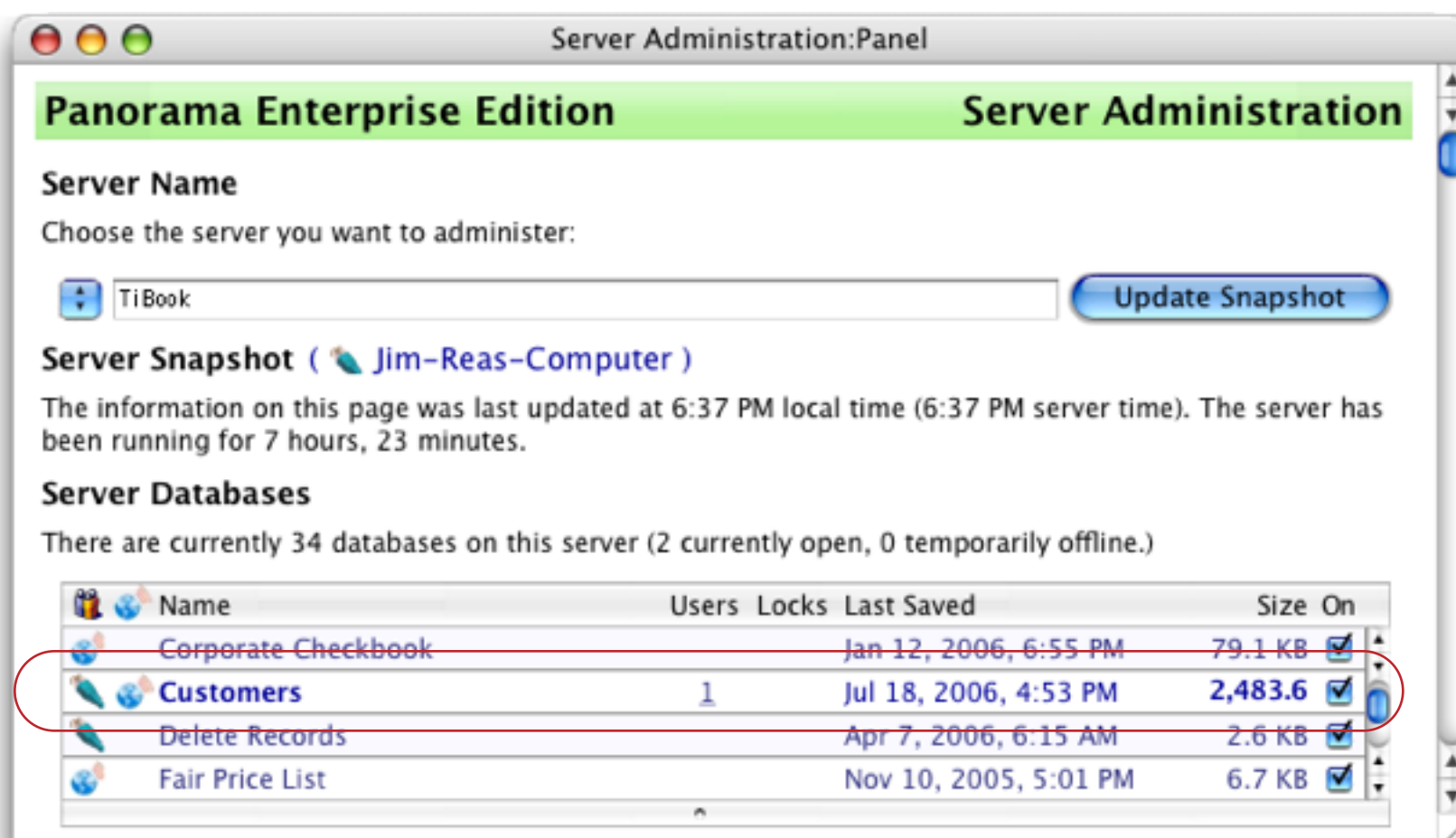
The wizard will display a summary of the proposed new sharing options for this database.



Double check to make sure the options are correct, then press the **Apply Options** button. The database will be uploaded to the server. You'll be notified when this process is complete.



If you want to verify that the database has been uploaded, check the list of server databases in the Server Administration wizard.



Be sure to save your original copy of the database. You'll need it if you need to make changes to the web database, including changing forms, tables, or procedures.

Testing a Web Database

To test your new web database you could simply open your web browser and type in the url for a form or procedure. But who wants to do all that typing? If you know the password, Panorama allows you to access any web form or procedure with a few mouse clicks.

The Server Status Page

The **Server Status Page** displays the current status of the web server on any web browser, including links for each database on the server.

Server Status

http://localhost/cgi-bin/Panorama.cgi?serverstatus@milwaukee

Panorama Enterprise Web Publishing Status

Panorama Enterprise Server web publishing is currently active.

Server Name: [TiBook](#)

Server URL: <http://localhost>

Server Time: [Friday, June 1st, 2007 at 4:42 PM](#)

Up Time: [20 hours, 2 minutes](#)

There are currently 14 web databases on this server.

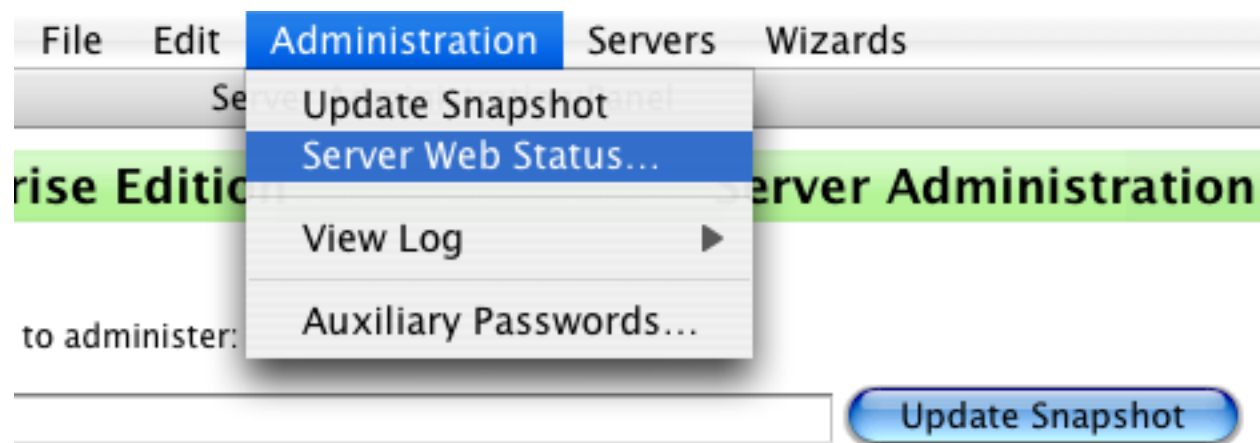
| Database | Last Modified | Size |
|--|--------------------|---------|
| AAA Real Estate X Test | May 6 @ 4:56 PM | 50.5 kB |
| Checkbook MacFair 2006 | May 5 @ 1:14 PM | 75.2 kB |
| Checkbook | Wed @ 4:51 AM | 64.0 kB |
| ColoradoHotels | Apr 29 @ 9:33 PM | 117 kB |
| Customers | May 16 @ 4:50 PM | 2.55 MB |
| Customers2 | 07/19/06 @ 5:48 PM | 2.45 MB |
| HobbyShopCatalog | May 9 @ 11:22 AM | 238 kB |
| HobbyShopSales | Dec 6 @ 11:43 AM | 638 kB |
| Lookups | May 5 @ 10:05 AM | 15.2 kB |
| MailingList | 07/14/06 @ 5:41 PM | 2.04 MB |
| PPdatabase Jim1 | May 15 @ 4:20 PM | 311 kB |
| PPdatabase | Apr 14 @ 4:35 PM | 194 kB |

To access this page enter the URL as shown below. Substitute your actual server IP or domain name (or [localhost](#) if on the same computer as the server) for [www.yourserverdomain.com](#). Substitute your Web Status Auxiliary password for [password](#) (see “[Changing the Auxiliary Passwords](#)” on page 80).

<http://www.yourserverdomain.com/cgi-bin/panorama.cgi?serverstatus@password>

From this page you can access any database form or procedure on the server within two clicks. We recommend that you bookmark this page for easy access as you are developing your web database applications. For more information see “[Testing Web Database Publishing \(Server Status\)](#)” on page 64.

As an alternative to typing in the URL for the server status page you can open the **Server Administration** wizard and choose the **Server Web Status** command.



This command will open your default web browser and open the server status page in a new browser window. The command automatically types in the correct domain name and password for you. See “[Server Management \(The Server Administration Wizard\)](#)” on page 68 for more information on this wizard.

The Database Web Links Page

The database names in the Server Status page are actually links. Clicking on the link displays the web links page for that database.

Server Status Page

Server Status

http://localhost/cgi-bin/Panorama.cgi?serverstatus

Panorama Enterprise Web Publishing Status

Panorama Enterprise Server web publishing is currently active.

Server Name: **TiBook**

Server URL: **http://localhost**

Server Time: **Sunday, May 20th, 2007 at 8:34 PM**

Up Time: **18 hours, 56 minutes**

There are currently 13 web databases on this server.

| Database | Last Modified | Size |
|--|--------------------|---------|
| AAA Real Estate X Test | May 6 @ 4:56 PM | 50.5 kB |
| Checkbook MacFair 2006 | May 5 @ 1:14 PM | 75.2 kB |
| ColoradoHotels | Apr 29 @ 9:33 PM | 117 kB |
| Customers | Wed @ 4:50 PM | 2.55 MB |
| Customers2 | 07/19/06 @ 5:48 PM | 2.45 MB |
| HobbyShopCatalog | May 9 @ 11:22 AM | 238 kB |
| HobbyShopSales | Dec 6 @ 11:43 AM | 638 kB |
| Lookups | May 5 @ 10:05 AM | 15.2 kB |
| MailingList | 07/14/06 @ 5:41 PM | 2.04 MB |

Database Web Links Page

Web Links for HobbyShopCatalog database

http://localhost/cgi-bin/Panorama.cgi?HobbyShopCatalog-admin

Panorama Enterprise Edition Server TiBook @ http://localhost

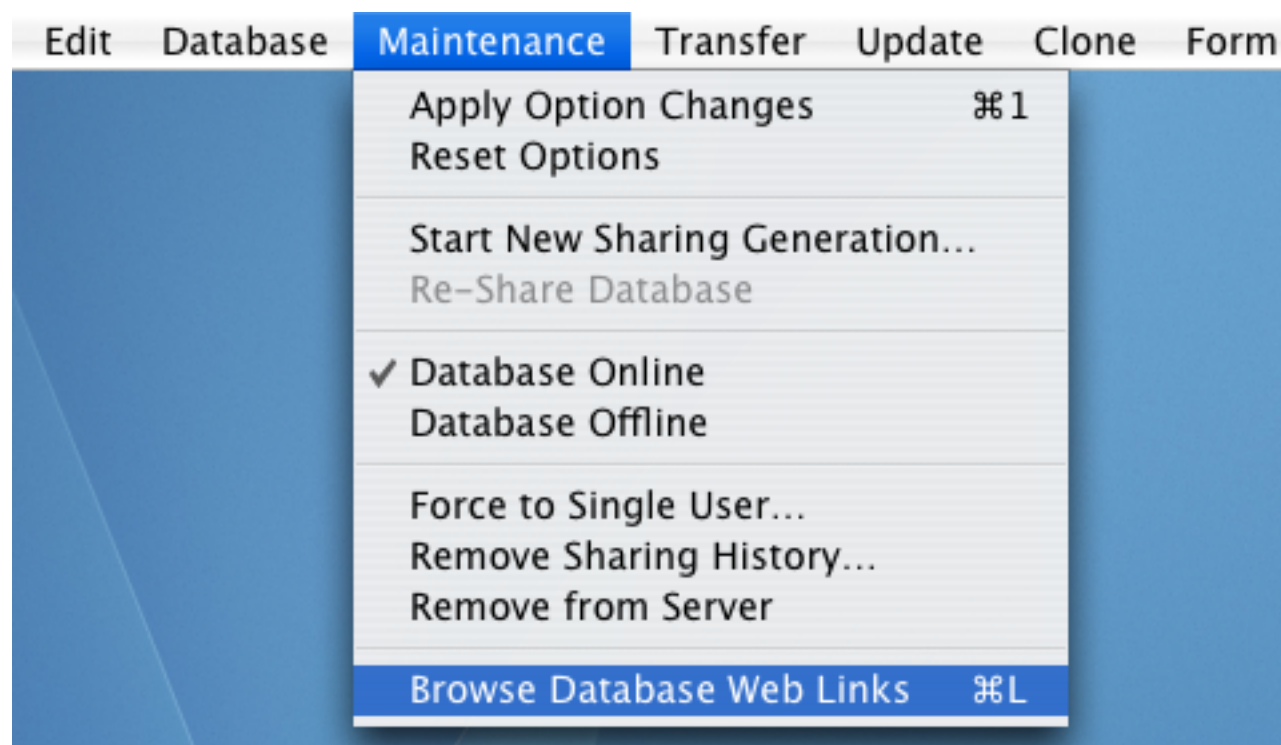
| Database | Forms | Procedures |
|---|---|--|
| HobbyShopCatalog • 96 records • 9 fields | Add Catalog Item Catalog Items Catalog Search Display Catalog Item Edit Catalog Item Search Results ShoppingCart T&E WebZ | (1) AddIndex CloseWindow Close DeleteLine DeleteItem DynamicProc FeatureItemLine Initiate LinkQuery MainMenu OpenItemLine OpenItem SearchLog SaveIndex CanPurchase CanDisplay CanEmpire CanRecommendations Calculate InitiateMatrix Lookup ListChoices MoveItemImage menuItem menuItem RemoveNewDemoItems Search Upper/ll Z ZZ |

The web links page lists all of the forms and procedures for that database. From this page you can click on any web form to open that form (see “[Testing a Web Form](#)” on page 204), or click on any procedure in the database to trigger the procedure (see “[Testing a Web Procedure](#)” on page 206). (If the web links page doesn’t appear, see “[Debugging Web Link Page Problems](#)” on page 207).

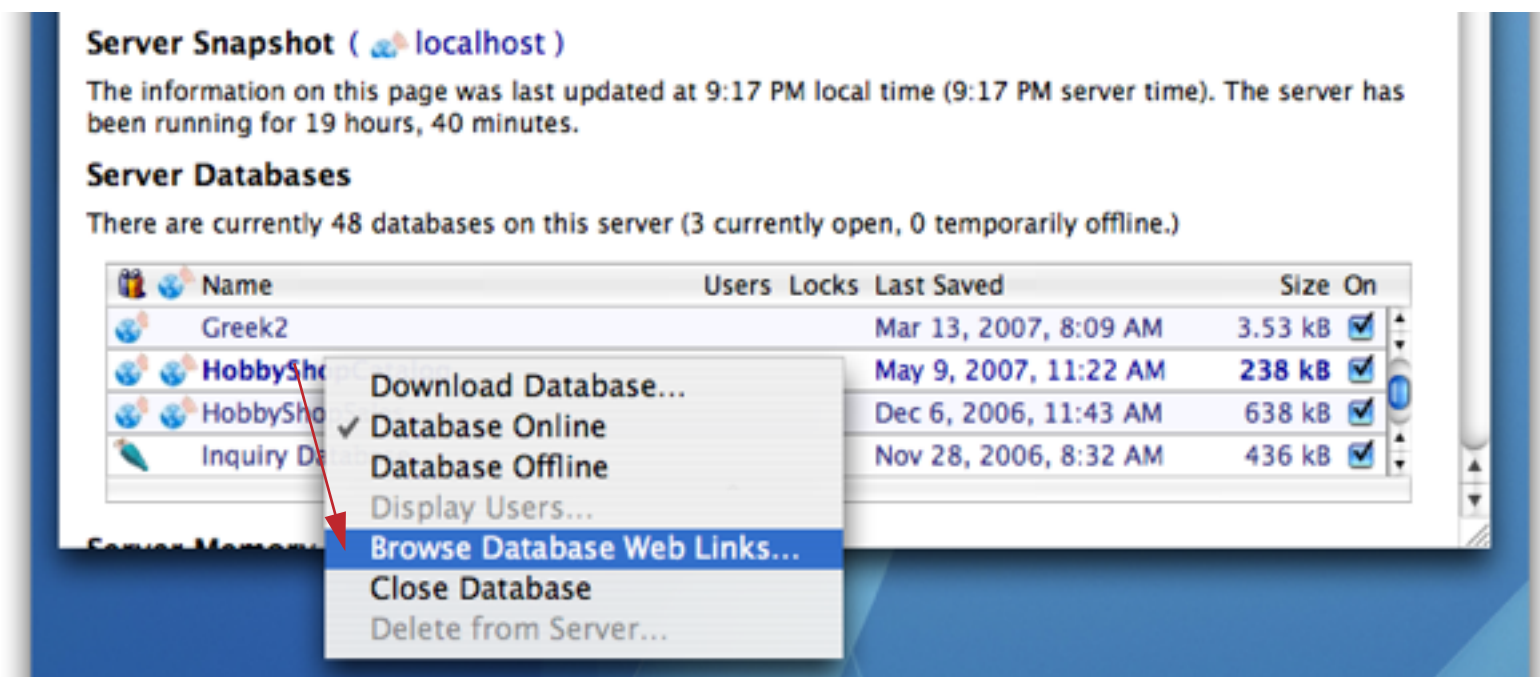


This page is divided into three columns. The first column displays basic information about the database. The second column lists all of the web forms in the database (only Panorama forms that have been specifically converted to web forms will be listed, see “[Converting a Panorama Form into a Web Form](#)” on page 231). The third column lists all of the procedures in the database.

Note: In addition to opening the Database Web Links page from the main server status page you can also open it using the Database Sharing Options wizard. Choose the **Browse Database Web Links** command from the **Maintenance** menu.



You can also open this Web Links page from the Server Admin wizard. Hold down the **Control** key and click on the database name to see the pop-up menu with the **Browse Database Web Links** command:



You can also open this page by simply typing in the url yourself in the format shown below, substituting the actual **domain**, **database name** and **password**.

<http://www.yourserverdomain.com/cgi-bin/panorama.cgi?database~admin@password>

If you do type in the url manually and have a *Web Admin Password* set up, you must include this password at the end of the url, as shown above. See "[Changing the Auxiliary Passwords](#)" on page 80 for more information on this password.

Testing a Web Form

To test a web form simply click on the name of the form. For example, clicking the [Advanced Search](#) link will display the corresponding form:

The screenshot shows a web browser window titled 'Advanced Search'. The address bar contains the URL 'http://localhost/cgi-bin/Panorama.cgi?Customers~for'. The page features a logo for 'World's Greatest Hobby' and a search form with the following fields:

- First Name:
- Last Name:
- Address:
- City:
- State:
- Zip Code:
- Phone:
- Email:
- Scale:
- Era:
- Railroad:
- Notes:

Forms accessed directly from the admin page will always be displayed as blank forms, with no data filled in. This allows you to start a process such as a search or adding a new record. In this case we can fill in the city to start a search:

The screenshot shows a web form with the following fields and values:

- Address:
- City:
- State:
- Zip Code:
- Phone:
- Email:
- Scale:
- Era:
- Railroad:
- Notes:

Buttons: Search, Reset

Pressing the **Search** button starts the search process, which in this case displays the first 15 records that match.

The screenshot shows a web browser window titled "Search Results" with the URL `http://localhost/cgi-bin/Panorama.cgi?Customers~qu`. The page displays the "World's Greatest Hobby" logo and the text "Search Results... 34 of 12,125 records selected from Customers." Below this is a search bar and a "Search" button. The results are shown in a table with the following columns: Name, City, Scale, Era, and Railroad.

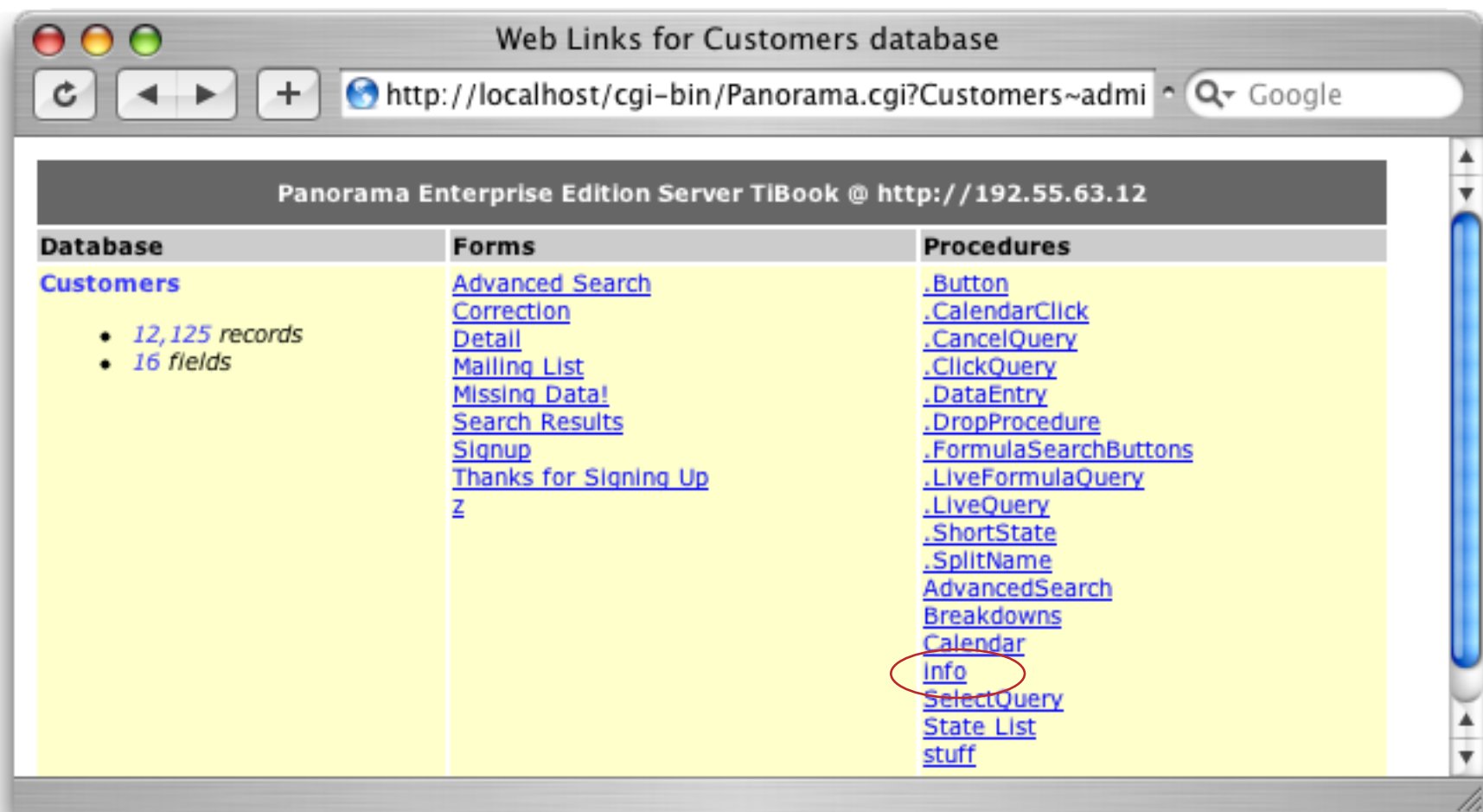
| Name | City | Scale | Era | Railroad |
|-----------------------------------|----------------|-------|------------|-------------------------------|
| Charles Middleton | Long Beach, CA | G | Modern | Santa Fe (ATSF) |
| Edward Weisgrau | Long Beach, CA | N | Transition | Amtrak |
| Thomas Horner | Long Beach, CA | G | Transition | Pennsylvania |
| Phillip Dalton | Long Beach, NY | N | Steam | Great Northern (GN) |
| Carl Anderson | Long Beach, CA | N | Modern | Union Pacific (UP) |
| John Marks | Long Beach, CA | HO | Modern | Conrail |
| Robert Bergin | Long Beach, NY | G | Modern | Santa Fe (BNSF) |
| Herbert Knight | Long Beach, CA | HO | Transition | Santa Fe (ATSF) |
| Joseph Hartman | Long Beach, CA | HO | Transition | Santa Fe (ATSF) |
| Henry Evans | Long Beach, MS | HO | Modern | Union Pacific (UP) |
| Russell Hulen | Long Beach, CA | G | Modern | Southern Pacific (SP) |
| Harry Day | Long Beach, CA | N | Transition | Norfolk & Western |
| Steven Life | Long Beach, CA | G | Transition | Western Pacific (WP) |
| Douglas Hillman | Long Beach, CA | O | Modern | Seabome Coast Line |
| Henry Haeler | Long Beach, CA | HO | Modern | Chicago & North Western (CNW) |

Navigation links: 1 2 3 NEXT

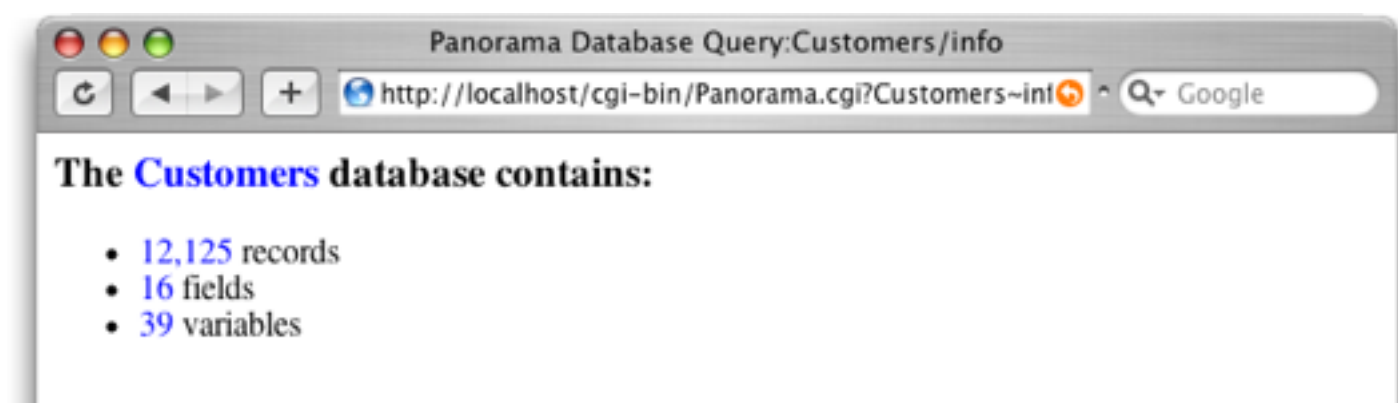
To learn more about how to set up a search process like this see “[Standard Form Action — QUERY](#)” on page 249.

Testing a Web Procedure

A **web procedure** is a procedure that is designed to produce a web page. To test a web form simply click on the name of the procedure from the third column of the web links page. For example clicking on the [info](#) link will trigger the info procedure. **WARNING:** The admin page lists ALL of the procedures in the database, whether they have been designed for producing web pages or not. Before you click on a procedure name be sure that you know that it is a web procedure. Clicking on an procedure that is not designed as a web procedure can cause unpredictable results, including the possibility of data loss or crashing Panorama. So be careful out there!



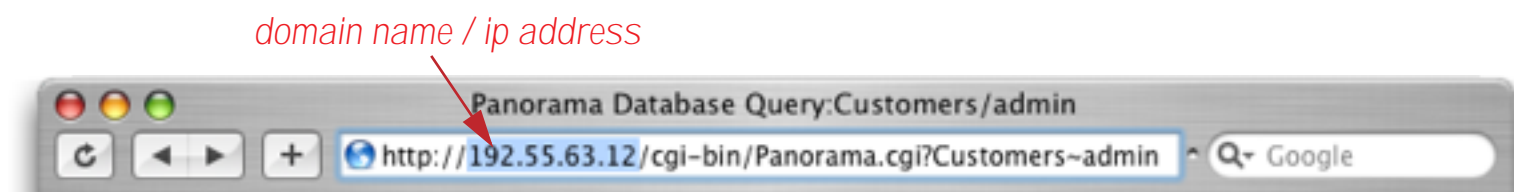
The [info](#) procedure is a simple procedure that displays a few facts about the database.



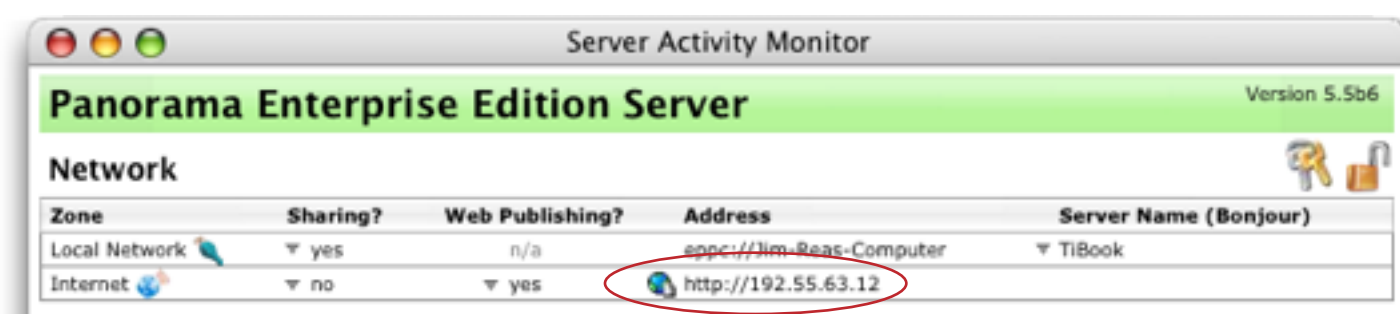
To learn more about how to write web procedures see “[Web Programming 101](#)” on page 317.

Debugging Web Link Page Problems

If the server is set up properly, you should have no problem getting the database's web link page to appear (see "[Testing a Web Database](#)" on page 199). If it doesn't appear the most likely problem is an incorrect domain name/ip address. Check the browser's address bar to make sure the domain name/ip address is correct.



The Database Sharing Options wizard gets the domain name/ip address from the server itself, so if this address is incorrect you'll need to adjust this on the server.



To fix this, you'll first need to unlock the server by clicking on the lock icon (see "[Unlocking the Server Configuration](#)" on page 49) and then click the globe icon to change the domain name/ip address (see "[Setting the IP Address/Domain Name](#)" on page 56).

Once the domain name/ip address has been fixed on the server you must open the Available Servers wizard and refresh the Server List.



Now you should be able to go back to the Database Sharing Options wizard's and select the **Browse Database Web Links** from the **Maintenance** menu to open the admin page in your browser (see "[Testing a Web Database](#)" on page 199).

Disabling the Web Link Page

If you want to completely disable the web link page (so others can't see the list of your forms and procedures even if they have the password) simply add a procedure named `admin` to the database, with a single line.

```
cgiHTML="Unauthorized."
```

You can change the message to whatever you want to appear if someone tries to access this page.

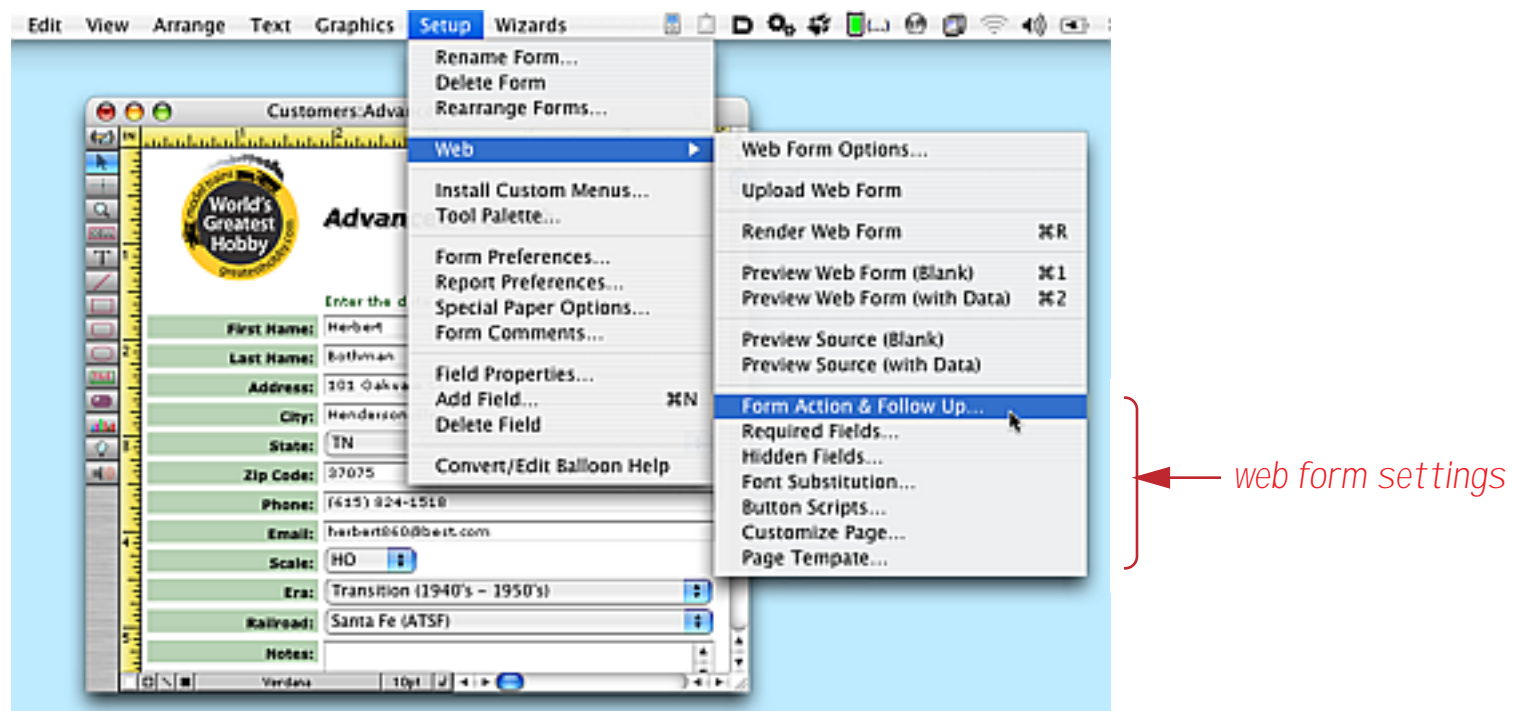
Modifying a Web Published Database

You can modify a web published database at any time. To do so you'll need the original copy of the database. No matter what you are modifying, the basic sequence is the same — modify the original database, then upload the change to the server.

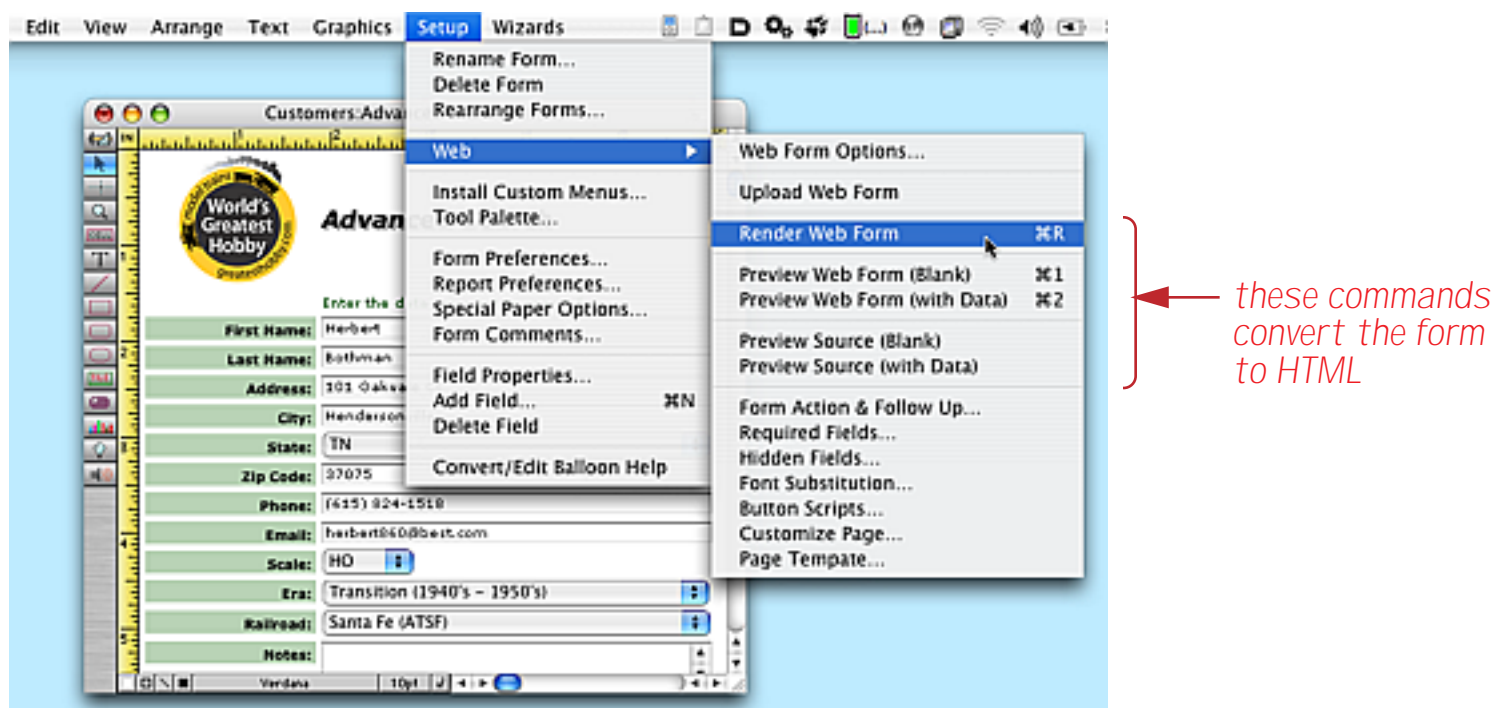
Updating a Web Form/Adding a new Web Form

This manual includes an entire chapter about setting up forms for the web (see “[Web Forms](#)” on page 231). But the short story for updating a form is very simple:

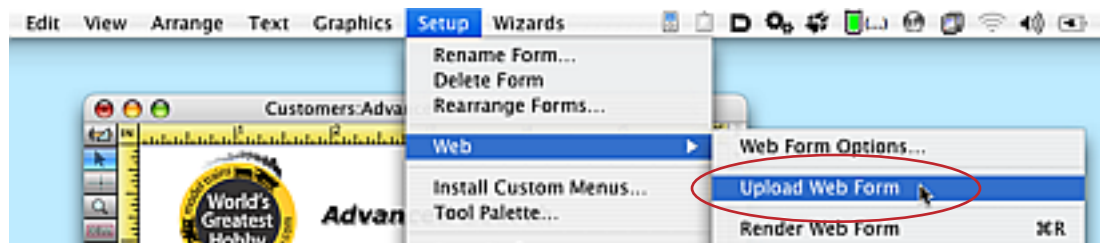
- If this is a new form, create the form.
- Make whatever changes are necessary to the form (create/move objects, etc.).
- Using the commands at the bottom of the **Web** submenu in the **Setup** menu, make any changes that are necessary to the web form settings (see “[Preparing a Form](#)” on page 236).



- Using the **Render** or **Preview** commands in the **Web** submenu in the **Setup** menu, convert the revised form to HTML (also called rendering the form, see “[Converting a Panorama Form into a Web Form](#)” on page 231).



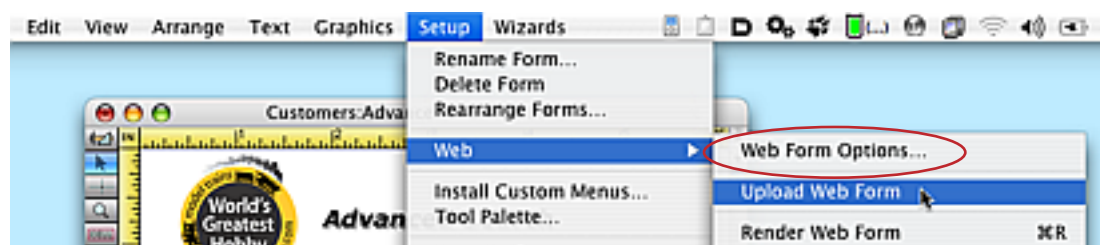
- Use the **Upload Web Form** command to update the server.



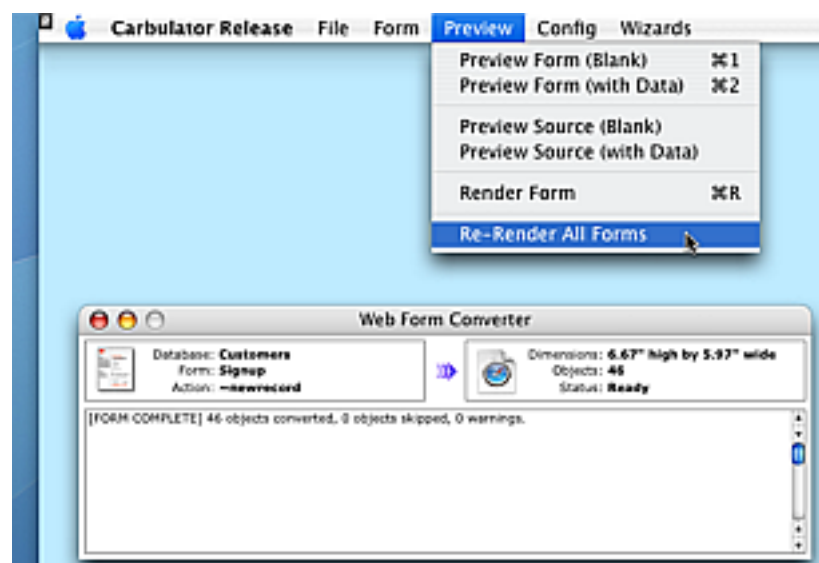
The final step is to test the revised/new form (see [“Testing a Web Database”](#) on page 199).

Updating All Web Forms

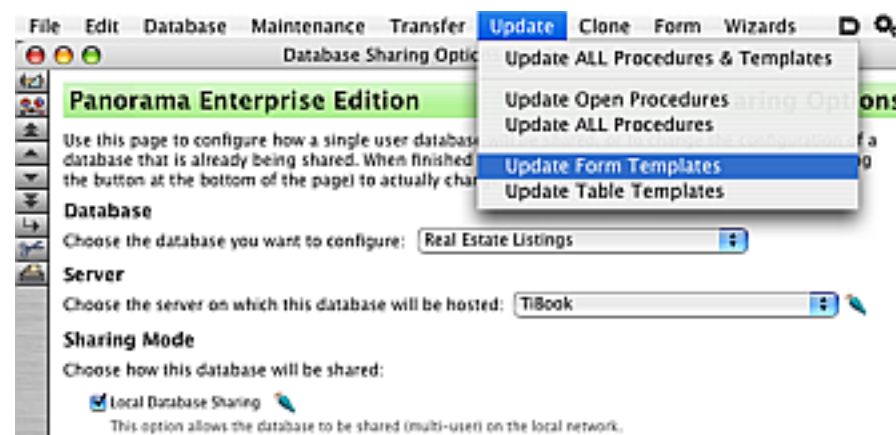
In rare circumstances you may need to re-render and re-upload all of the forms in a database. (For example, all the forms need to be re-rendered if you change the name of the database on the server (this includes making a new copy of the database with a different name.) To re-render all of the forms, first open the **Web Form Converter** wizard using the **Web Form Options** command.



Now choose the **Re-Render All Forms** command from the **Preview** menu.



Depending on how many forms are in your database this process may take a while. Only forms that have already been rendered to HTML already will be re-rendered. When the process is complete, open the **Database Sharing Options** wizard and choose the **Update Form Templates** command from the **Update** menu.

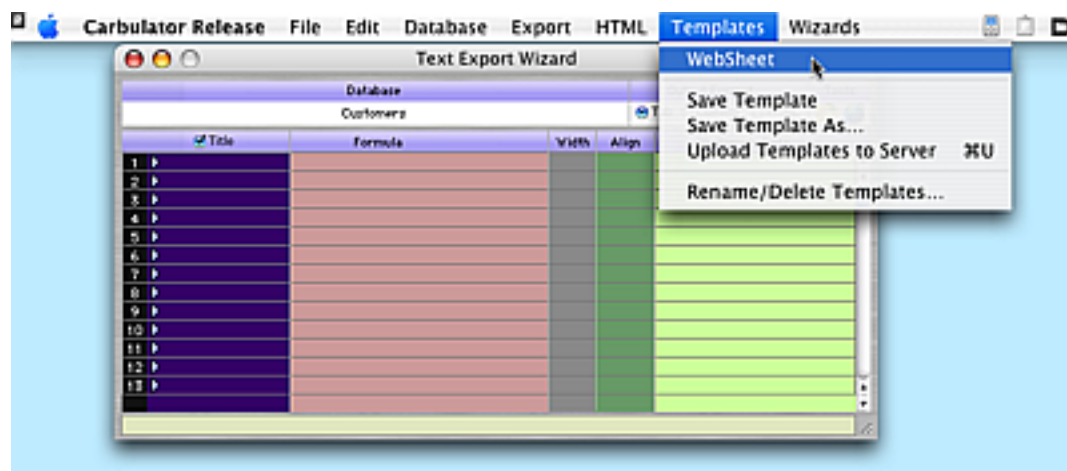


You can also choose the **Update All Procedures & Templates** command from the **Transfer** menu, which also uploads all of the procedures and web tables.

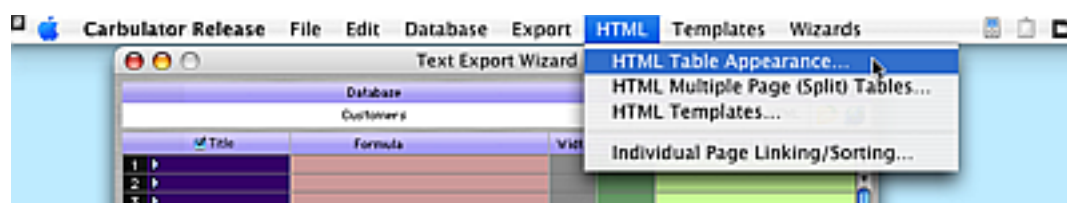
Updating a Table/Adding a new Web Table Template

This manual includes an entire chapter about setting up web table templates (see “[Web Tables](#)” on page 277). But the short story for updating a template is very simple:

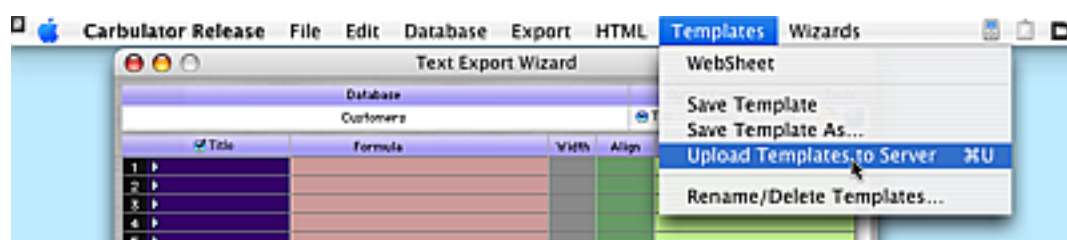
- Open the **Text Export** wizard.
- If this is a new template, create the template. Otherwise select the template from the **Template** menu.



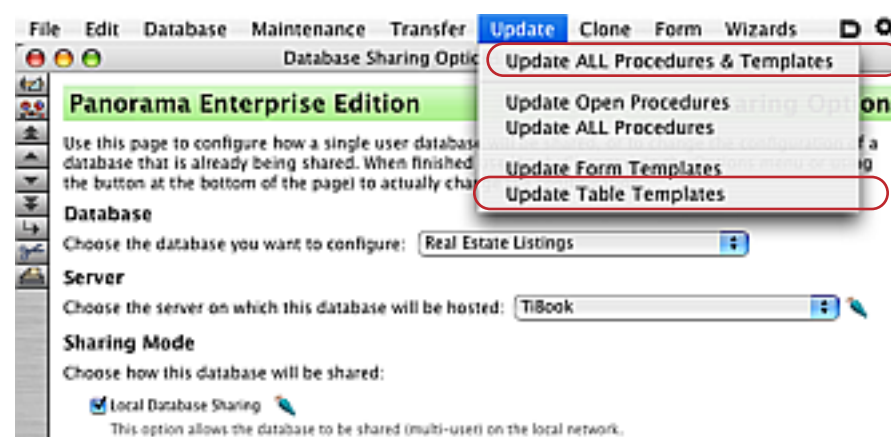
- Use the dialogs in the **HTML** menu to customize the template (see “[Customizing the Table Appearance](#)” on page 283).



Choose **Upload Templates to Server** from the **Templates** menu (this actually uploads all of the web table templates in this database).



- You can also upload web table templates from the **Update** menu of the **Database Sharing Options** wizard.

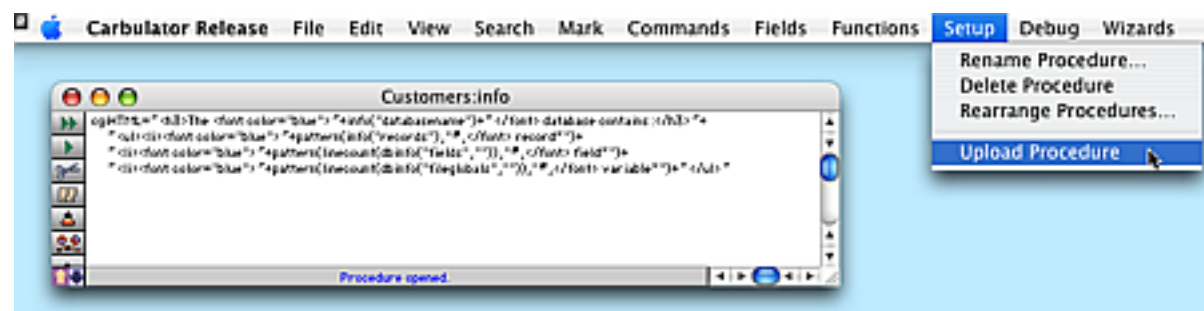


The final step is to test the revised/new web table (see “[Testing a Web Database](#)” on page 199).

Updating a Procedure/Adding a new Procedure

This manual includes an entire chapter about writing and debugging procedures for performing web actions (see “[Web Programming 101](#)” on page 317). But the short story for updating a procedure is very simple:

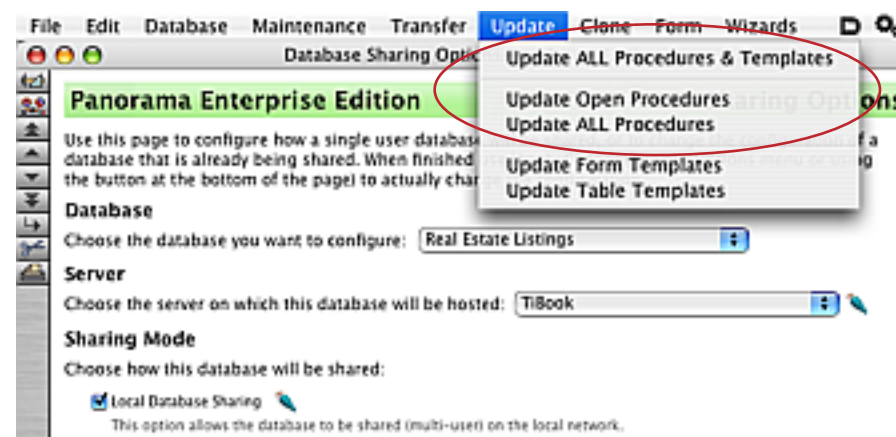
- Open the procedure, or create the new procedure.
- Make any changes that are necessary.
- Choose **Upload Procedure** from the **Setup** menu.



That's all there is to it! The final step is to test the revised/new web table (see “[Testing a Web Database](#)” on page 199).

Uploading Multiple Procedures

If you make changes to more than one procedure you can either upload each one separately (as described in the previous section) or you can use the **Update** menu in the **Database Sharing Options** wizard to upload a bunch of procedures at once.



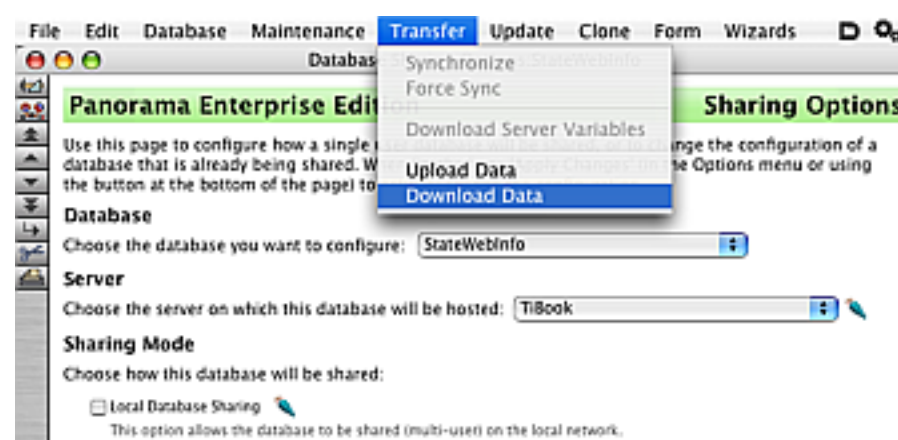
To update all of the procedures that are currently open (visible on the screen) use the **Update Open Procedures** command. To update all of the procedures in the database, whether they are open or not, use the **Update ALL Procedures** command or the **Update ALL Procedures & Templates** command. (Note: When uploading is complete, these commands will list all of the procedures that have been updated. Only procedures that actually changed will be listed! If no procedures have changed, you'll see that zero procedures have been updated.)

Synchronizing data between the original copy and the server

If you've set up your web published database with database sharing turned on (see "[Database Sharing and Web Publishing](#)" on page 195) then you don't have to worry about synchronizing data with the server. Just open your local copy of the database and Panorama takes care of everything for you. If you didn't turn on database sharing then follow the steps in one of the sections below to either download data from or upload data to the server.

Downloading Data from the Server

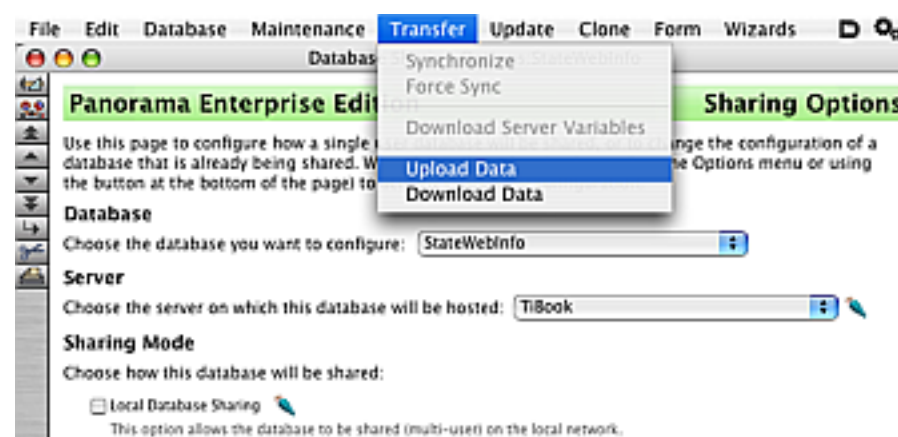
If you are collecting data on the server (for example a guest book) then you can periodically download the data to your local copy of the database for further analysis, etc. Start by opening the **Database Sharing Options** wizard, then choose **Download Data** from the **Transfer** menu (if the database is shared this menu option will be disabled, use **Synchronize** instead).



WARNING! When you use the **Download Data** command any data that is in the local copy of the database is erased, and cannot be recovered. The local data is completely replaced by the data downloaded from the server. If you want to make changes both using web browsers and using Panorama itself we recommend enabling the database sharing option.

Uploading Data to the Server

If you are updating the local Panorama database (for example making changes to a product catalog) then you can periodically upload the data to the server (in other words publish the changes). Start by opening the **Database Sharing Options** wizard, then choose **Upload Data** from the **Transfer** menu (if the database is shared this menu option will be disabled, use **Synchronize** instead).



WARNING! When you use the **Upload Data** command any data that is in the server copy of the database is erased, and cannot be recovered. The data on the server is completely replaced by the data uploaded from the local copy of the database. If you want to make changes both using web browsers and using Panorama itself we recommend enabling the database sharing option.

Removing a Database from the Server

There are three methods for removing a database from the server.

Method 1 - Open the original database on your client computer, then open the **Database Sharing Options** wizard. Then choose **Remove from Server** from the **Maintenance** menu.

Method 2 - Open the **Server Administration** wizard. Find the database you want to delete in the list of server databases. Hold down the **Control** key while you click on the database name, then select **Database Offline** from the pop-up menu.

Server Databases

There are currently 14 databases on this server (1 currently open, 0 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|-----------------------------|-------|-------|-----------------------|---------|-------------------------------------|
| Offline Mailing List | | | Nov 4, 2005, 12:40 PM | 21.0 KB | <input checked="" type="checkbox"/> |
| Real Estate Listings | | | Feb 12, 2006, 2:35 PM | 48.9 KB | <input checked="" type="checkbox"/> |
| StateAbbreviationsTi | | | Sep 1, 2005, 4:26 PM | 2.2 KB | <input checked="" type="checkbox"/> |
| Test X Price List | | | Nov 11, 2005, 4:21 PM | 6.7 KB | <input checked="" type="checkbox"/> |

The database will go dim indicating that is now offline (can no longer be accessed).

Server Databases

There are currently 14 databases on this server (1 currently open, 1 temporarily offline.)

| Name | Users | Locks | Last Saved | Size | On |
|----------------------|-------|-------|------------------------|----------|-------------------------------------|
| Forum | | | Nov 4, 2005, 1:04 PM | 16.6 KB | <input checked="" type="checkbox"/> |
| HobbyShopCatalog | | | 12, 2006, 7:08 PM | 150.9 KB | <input type="checkbox"/> |
| Invoices | | | Jul 27, 2005, 11:19 AM | 48.5 KB | <input checked="" type="checkbox"/> |
| Offline Mailing List | | | Nov 4, 2005, 12:40 PM | 21.0 KB | <input checked="" type="checkbox"/> |

database is offline

Finally, hold down the **Control** key, click on the database name and choose **Delete from Server** from the pop-up menu.

Method 3 - You shouldn't have any trouble with the first two techniques, but if you do, here is a sure fire method. Go to the server computer and shut down the server (see "[Shutting Down the Server](#)" on page 45). Open the **Public Databases** folder (see "[Public Databases](#)" on page 494) and erase the database file and the **database.cfd** file. (Note: If the database is shared then the database file will be named **database.ees**. In that case you should also erase the journal file, if any, which is named **database.jnl**.) Then restart the server.

Associating a Database with Multiple Servers (Clones)

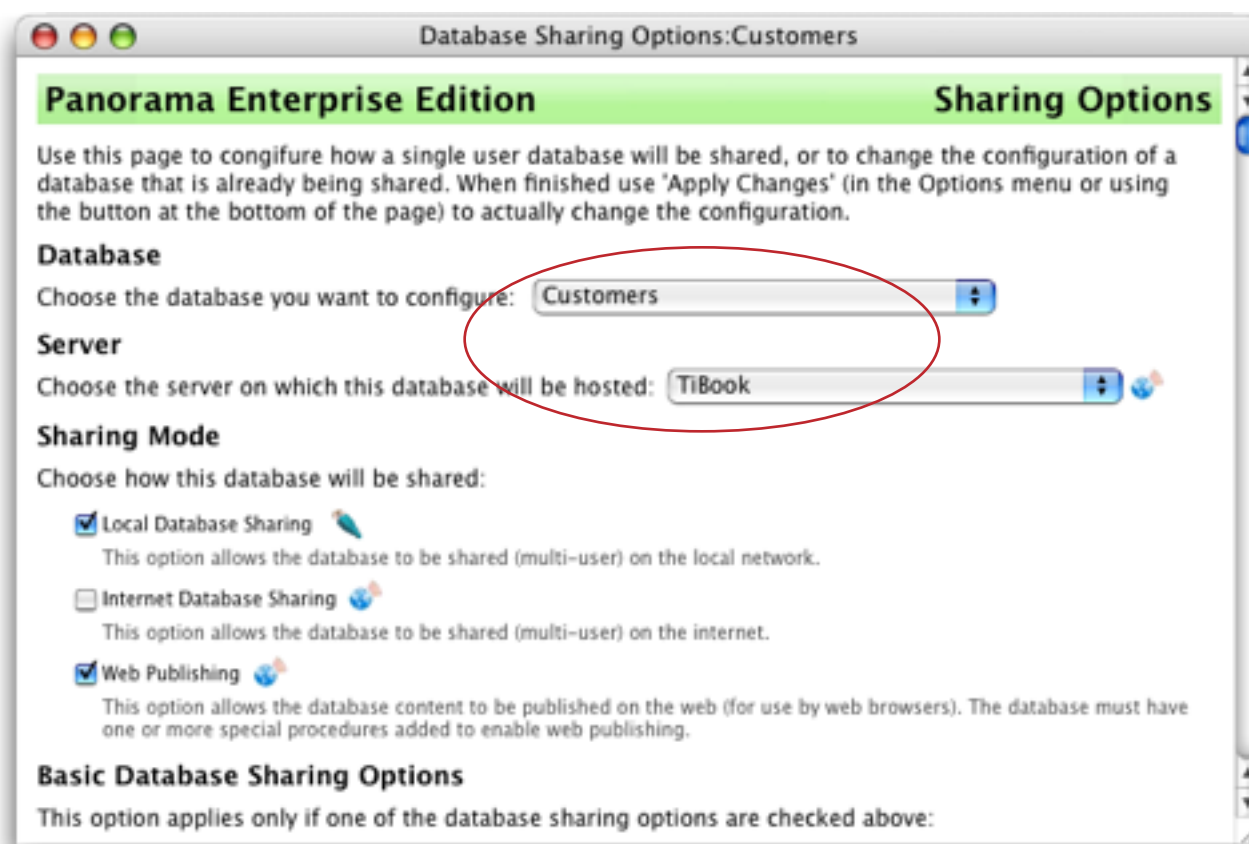
Normally a database is only associated with a single server. However there are situations where you might want to have a single database with clones on two or more servers:

| | |
|-----------------------------|--|
| Test vs. Production Servers | In many situations you may want to run an internal test server for debugging as well as a separate production server for public access. This allows you to do in-house testing on your local network, resolving any problems with procedures or forms before actually deploying them on the public server. |
| “Canned” Web Applications | If you are a developer or web consultant that creates standard web applications that are sold to multiple customers, you can create clones on each customer’s server. As you make updates to your application, it is very easy to update each of the clones on all of your customer’s servers. |

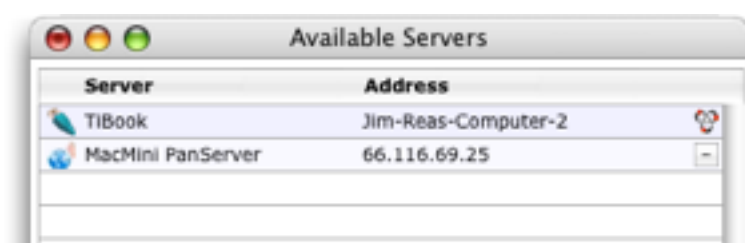
Clones share procedures and forms, but not data. Each clone has its own separate data, and there is no synchronization or linkage of data between the different clone databases.

Creating a Clone

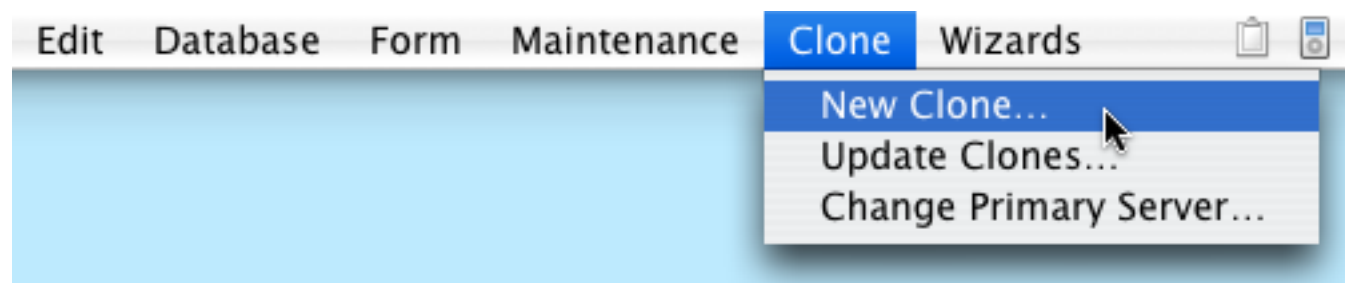
To illustrate creating a clone we’ll start with a database named **Customers** which is already on a server named **TiBook**.



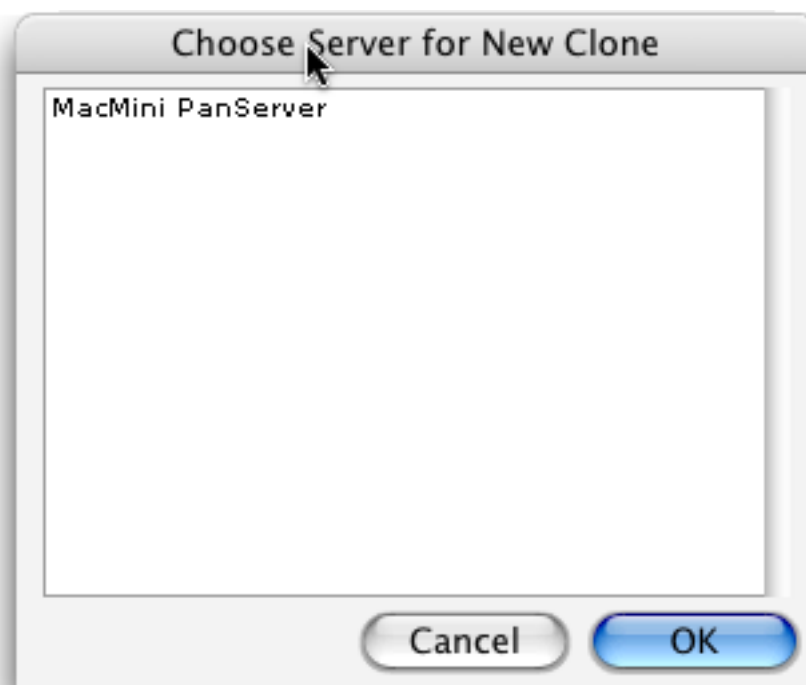
Checking our available servers, we have one more server available (**MacMini PanServer**):



To make a clone of the Customer database on the [MacMini PanServer](#) server use the Database Sharing Options wizard's Clone menu.



This command will display a list of available servers. (Any server that already contains a copy of this database is excluded from the list, which is why [TiBook](#) doesn't show up.)



Choose the server you want to use, then press **OK** (or double click on the server name). Panorama will upload a copy of the database to the new server (in this case [MacMini PanServer](#)) where it can then be accessed from web browsers.

Updating a Clone Database's Procedures and Forms

After you've created one or more clones, you can continue to use and modify the original database. Whenever you update procedures or forms (see "[Updating a Procedure/Adding a new Procedure](#)" on page 211 and "[Updating a Web Form/Adding a new Web Form](#)" on page 208) the original master database will be updated — the clones on other servers aren't touched. When you are ready to update one or more clones open the **Database Sharing Options** wizard and use the **Update Clones** command in the **Clone** menu.

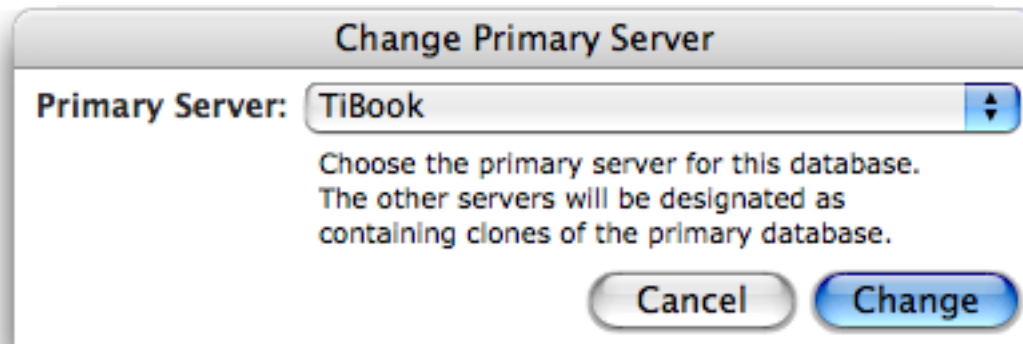


The dialog lists all of the clones for this database (in this case there is only one). Select the clones that you want to update, then press the **Update** button. Panorama will update the procedures and forms on the clone databases. (Note: You don't need to take a database offline to update it.) When it's done, it will display a list of the forms and procedures that it has changed. (Note: Only procedures that have actually changed will be listed.)

Remember that each clone has its own separate data, so updating does not transfer any data.

Changing the Primary Server

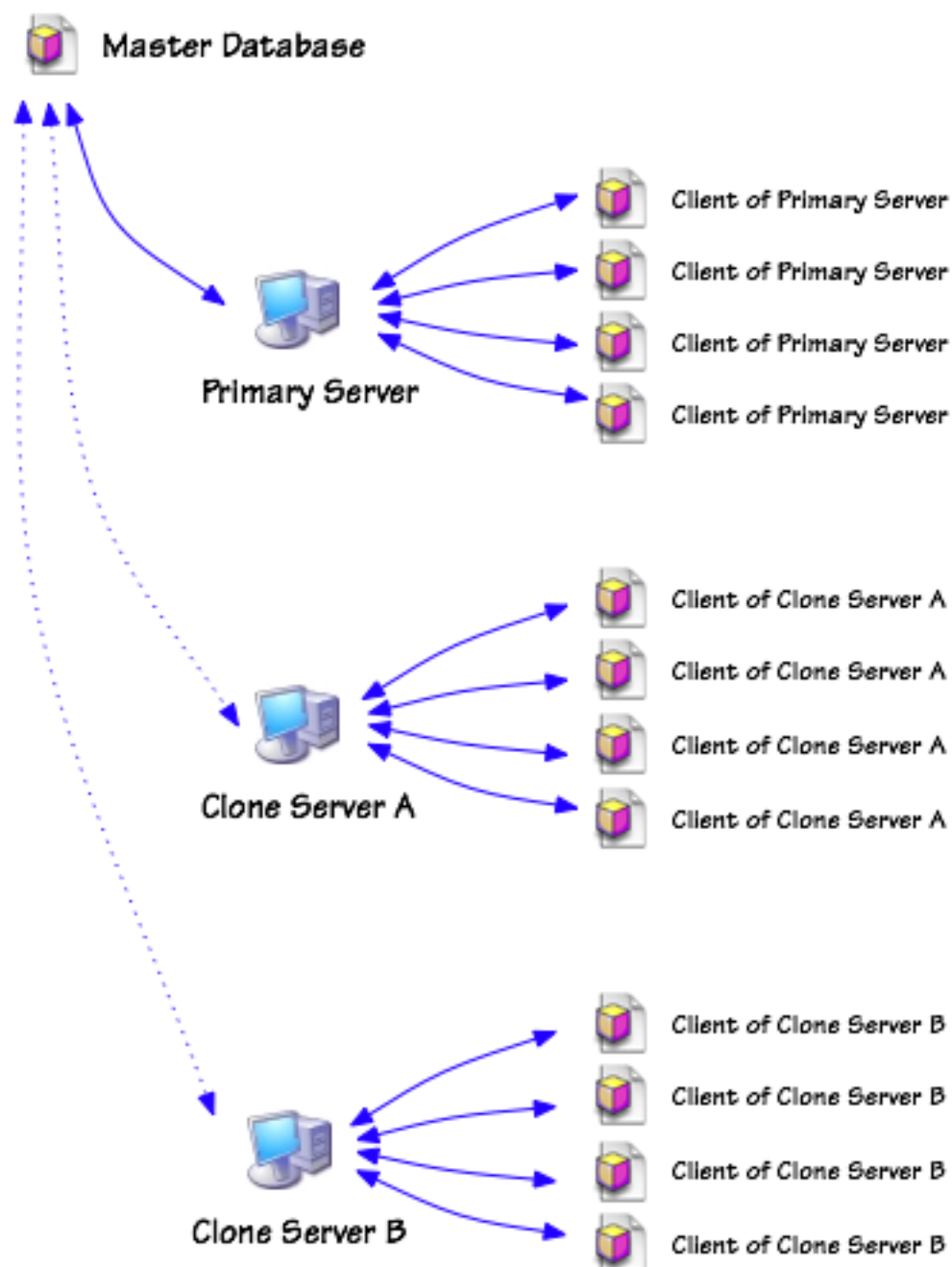
Though you can easily update any clones you've created with the **Update Clones** command (see above) the primary server remains the one you originally uploaded the database to. If necessary, however, you can change the primary server at any time using the **Change Primary Server** command in the **Clone** menu of the **Database Sharing Options** wizard.



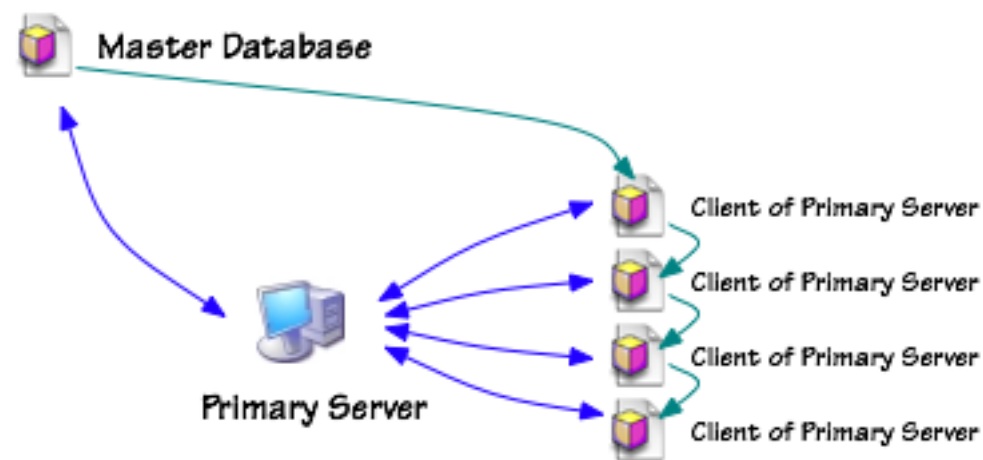
Use the pop-up menu to select the new primary server. If this is a shared database, the wizard will force synchronize the local client database with the new primary server. In other words, all of the data in the local client will be discarded and the data from the new primary server downloaded into the client.

Distributing Shared Clients for Clone Servers

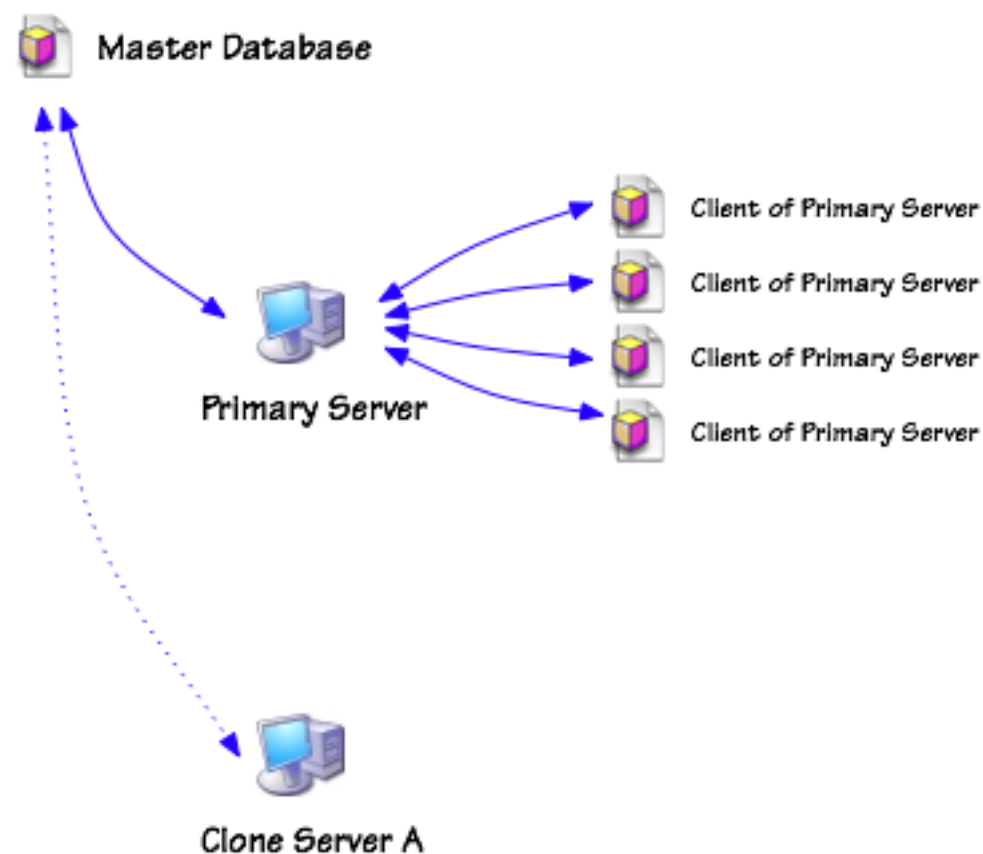
When a shared database is cloned, each server will have its own set of clients linked to it. There is no data sharing between the different servers or the clients of different servers.



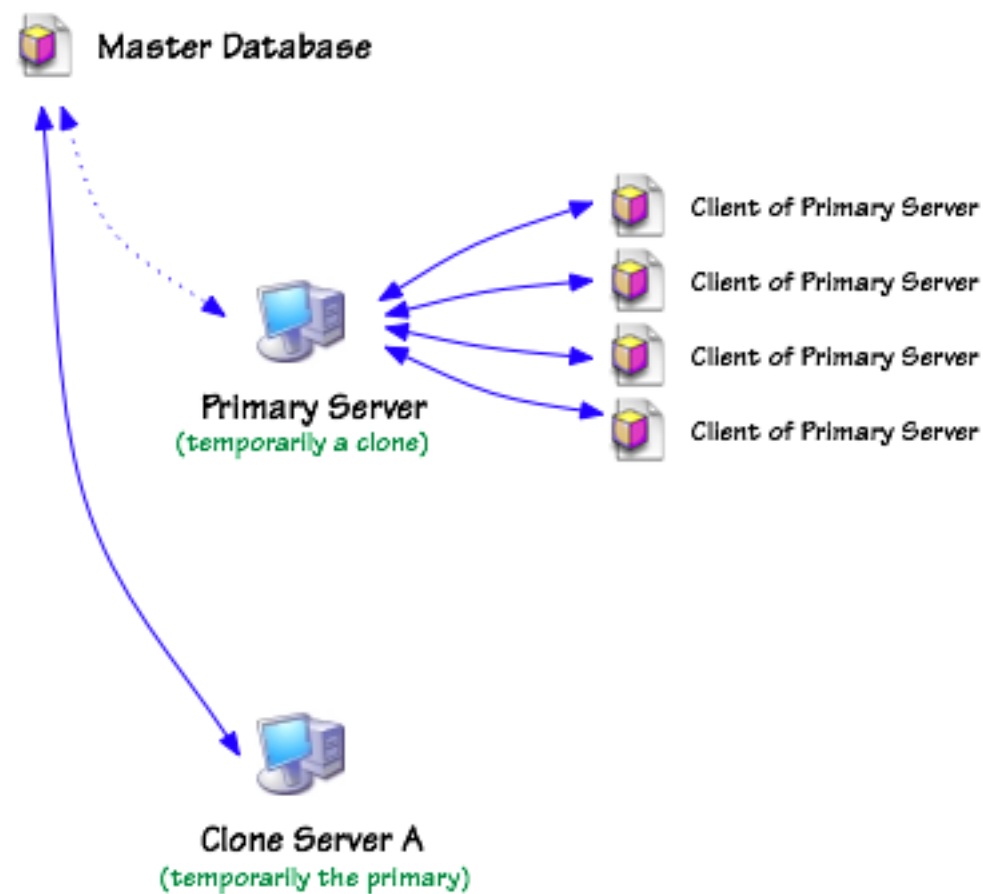
To set up these sets of servers you'll need to at least temporarily change the primary server. Start by setting up the primary server and its clients as described in Chapter 3 (see "[Creating a Shared Database](#)" on page 103). The green arrows show how copies of the master database are distributed from client to client, either by copying in the Finder, e-mail, USB drive, or some other file transfer mechanism.



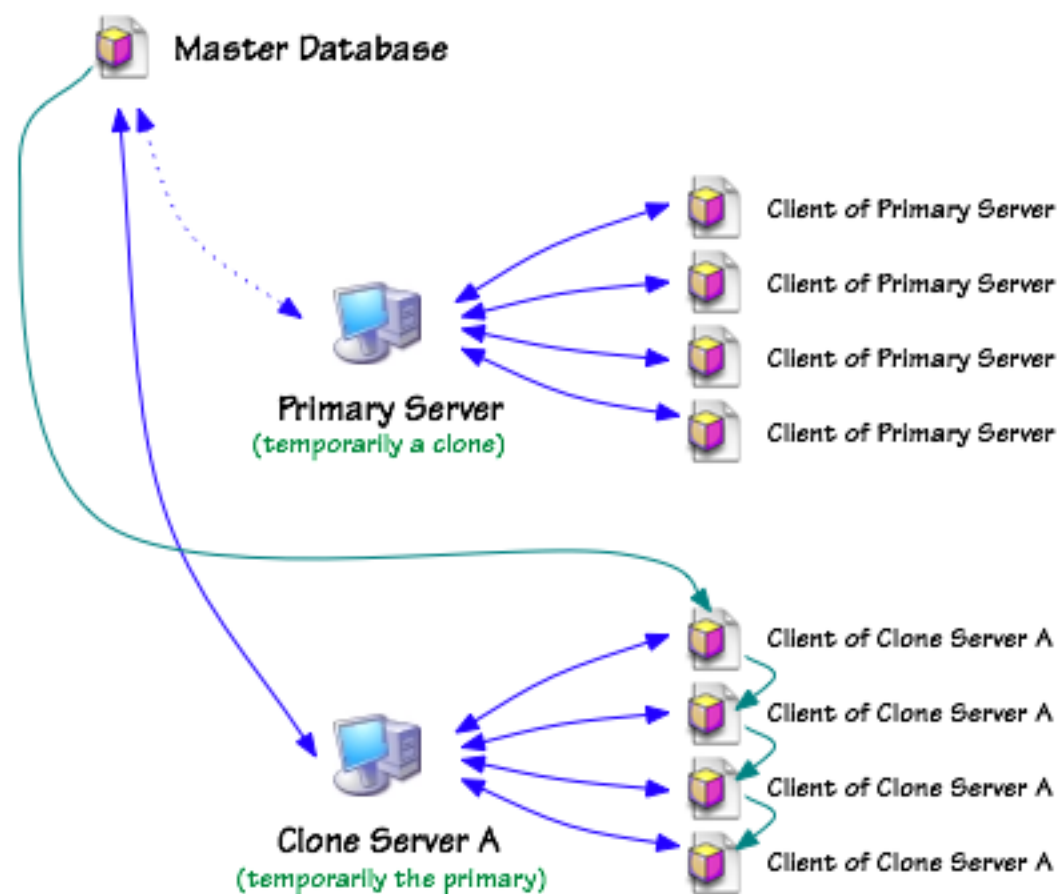
Next, use the **New Clone** command to set up a new clone of the database (see "[Creating a Clone](#)" on page 214).



Then change the primary server so that the clone is now the primary, and the original server is now a clone (see “[Changing the Primary Server](#)” on page 217).



Now distribute copies of the master database to the clients of the clone server, again using finder copying, e-mail, usb drives, or whatever other method you want to use to copy files. (Once you have at least one copy you can always make more duplicates later.)

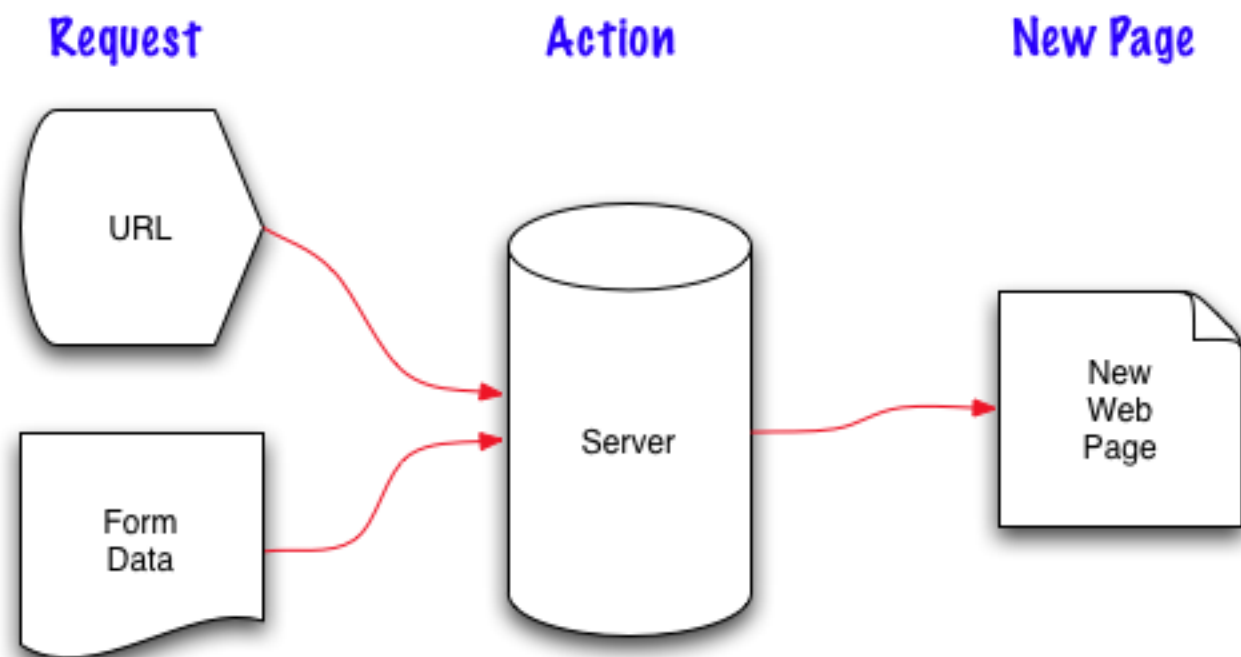


Repeat for any additional clone servers. When you're done, switch the primary back to the original server, and all of your servers and clients are ready to use.

Designing Your Web Database Application

So far we've discussed the nuts and bolts of working with the server, while skipping over the most important part — designing the actual web database application. A web application is divided into discrete steps or actions. Each step begins with someone using a web browser making a request to the server. The server processes the request and generates a new custom web page which it sends back to the web browser.

The diagram below illustrates the operation of a single step in a web application.



The original request is triggered by typing in a URL, clicking on a link, or pressing the **Submit** button on a web form. Each request may have multiple components. Every request has a URL. The request may also contain data that has been submitted from a form.

Web Database URL Format

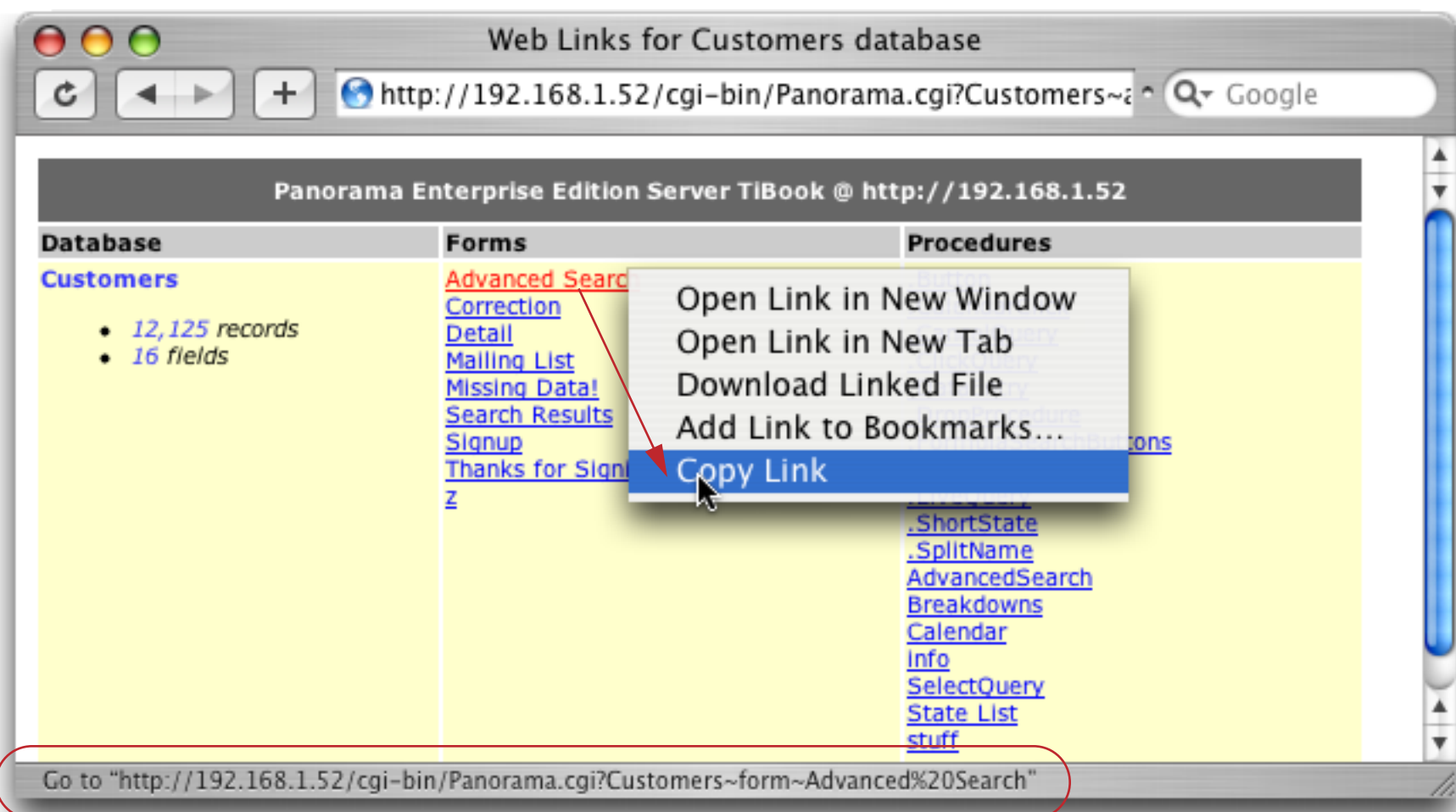
When the server receives a request it starts by analyzing the URL. The information in the URL tells the server what sort of action it should take. You can type the URL into the web browser, but more commonly you'll include the URL's in links from other web pages (using the `` tag). The URL for accessing a Panorama database is always in the format shown below.



The [http:](#) and [/cgi-bin/Panorama.cgi?](#) portions of the URL are always the same. The rest of the URL is divided into three sections.

| Section | Description | Examples |
|----------|---|--|
| Domain | This is the overall location of the entire web site. If you don't know the domain you'll need to contact your ISP or IT department. If this is a public web site then this will be a name like www.mycompany.com , etc. If this site doesn't have a name you can use the ip address (192.168.1.50), or you can use localhost if the server is on the same computer as the browser (for testing) | www.acme.com 192.169.1.100 localhost |
| Database | This is the name of the database being accessed. Be sure to use the name on the server (if it is different than the original name). | Contacts Invoices |
| Action | This is the action to be performed by this URL. The Panorama server has several built in actions (for displaying forms, performing queries, and other common tasks) or any procedure in the database can be used as an action. The action is always separated from the database name by a ~ character. Sometimes the action may include one or more extra parameters, in that case these extra parameters are separated from the main action name by a ~ character. | form~DataEntry updaterecord displayshoppingcart newitem~4~widgets |

You can use the database web link page to generate and copy a URL without typing (see “[Testing a Web Database](#)” on page 199). This illustration shows how to generate a URL for the [Advanced Search](#) form.



Once the URL is in the clipboard it can easily be copied to BBEdit, Dreamweaver, or the web authoring program you prefer.

Standard Actions

The Panorama Enterprise Server has a handful of standard built-in actions that perform common actions like displaying a form (see “[Form Actions and Sequence](#)” on page 241), performing a query (see “[Standard Form Action — QUERY](#)” on page 249), adding a new record (see “[Standard Form Action — NEWRECORD](#)” on page 243), updating an existing record (see “[Standard Form Action — UPDATERECORD](#)” on page 253) and displaying submitted data (see “[Standard Form Action — FORMDUMP](#)” on page 242). Standard actions can also be connected together to perform more complicated sequences, as will be described in a moment.

Custom Actions

Custom actions are created by writing procedures. Using custom actions gives you the greatest flexibility and power, but has the disadvantage of requiring programming skills. To be used as a custom action a procedure must fill the global variable `cgiHTML` with the HTML text to be displayed on a browser. A very simple custom action procedure might contain only one line. This example displays the current time.

```
cgiHTML="<font color=green>The time is <b>"+timepattern(now(),"hh:mm am/pm")+<b></font>"
```

Of course most custom actions are much more complicated than this. To learn more about writing custom actions see “[Web Programming 101](#)” on page 317.

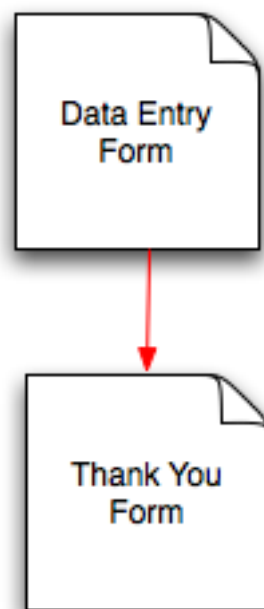
Action Sequences

A single server action can be used all by itself, but more often each action will be part of a larger sequence. A typical sequence might be displaying a search form, then displaying a list of results, then displaying a detailed form of a particular item. From the users point of view, the sequence of actions flows from page to page to page.

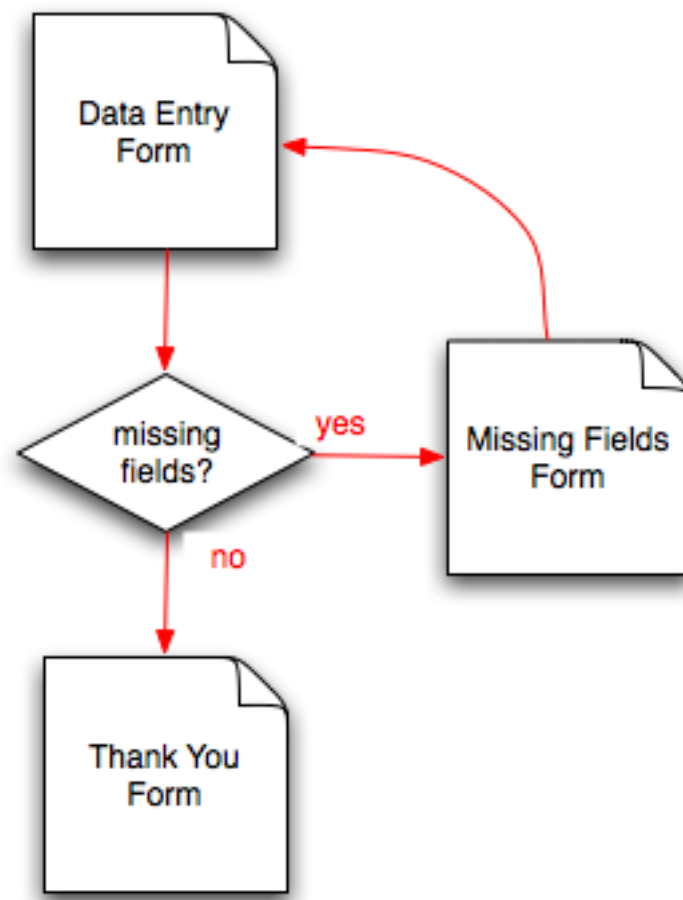
The Panorama Enterprise Server’s standard actions have been designed so that they can easily be connected into larger sequences. To set up these standard sequences you simply fill in dialog boxes specifying how the sequence flows from page to page. In this chapter you’ll learn about the overall theory and operation of these standard sequences. The nitty gritty details of setting up a sequence are covered in detail in the following chapters.

Standard Data Entry Sequence

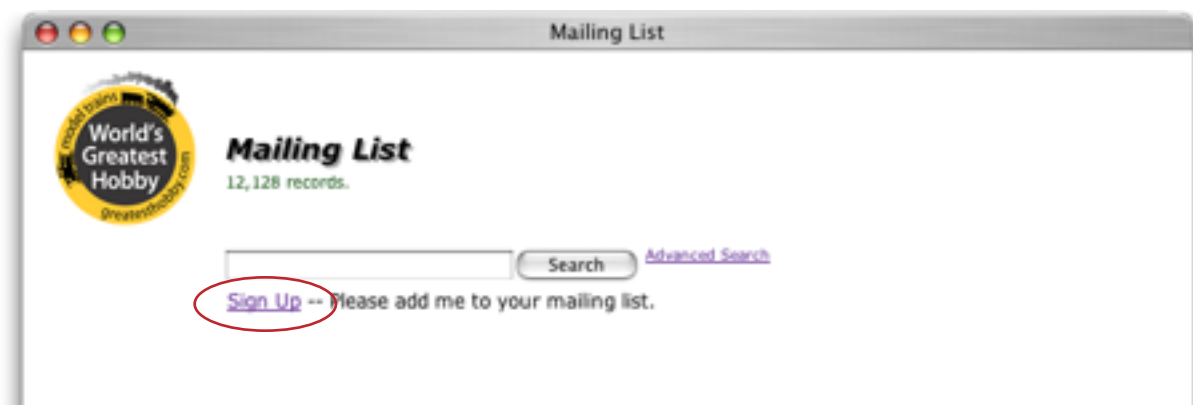
The simplest standard sequence is the **data entry** sequence. This sequence allows the user to fill in a form and add the new data to a database. In it’s most basic form this sequence has only two elements: a data entry form and a thank you form.



You can also configure this sequence to check for missing fields, in which case the sequence is slightly more complicated.



Let's look at a typical data entry sequence in action. The sequence starts by clicking a link from another page on the site.



This link brings up the data entry form, which generally starts out blank.

World's Greatest Hobby
Sign Up for Mailing List

First Name:

Last Name:

Address:

City:

State: AL

Zip Code:

Phone:

Email:

Scale: G O S HO N Z

Era: Early (1840's - 1910's)

Railroad: Amtrak

Notes:

Sign Up Reset

Once the form is filled out press the **Submit** button to complete the sequence.

Panorama Database Query: MailingList/newrecord

The following required fields are missing:

- City
- Zip
- Email

Please press the Back button on your browser to re-edit the form and input the missing fields.

Oops — the sequence is not complete because some required fields were missing. In this case the server is displaying the default missing fields page, but later you'll learn how to customize this page any way you want. For now we'll simply press the browser's back button to get back to the data entry form.

World's Greatest Hobby
greaterhobby.com

Sign Up for Mailing List

First Name: John
Last Name: Wilson
Address: 992 Olive Avenue
City: Long Beach
State: CA
Zip Code: 98621
Phone: (562) 555-1212
Email: jwilson@bigisp.net
Scale: G O S HO N Z
Era: Modern (1960's - Present)
Railroad: Western Pacific (WP)
Notes:

Sign Up Reset

Once the missing fields are entered press **Submit** again to actually add a new record to the database on the server with the new information.

World's Greatest Hobby
greaterhobby.com

Thanks for Signing Up!

Thanks for signing up for our mailing list. Please check the information below. Click [here](#) if you need to make any changes.

Address: John Wilson
 992 Olive Avenue
 Long Beach, CA 98621
Phone: (562) 555-1212
Email: jwilson@bigisp.net
Scale:
Era: Modern (1960's - Present)
Railroad: Western Pacific (WP)
Notes:

This final form is called the “thank-you” form. Once again you can customize it any way you want — it can even be set up to allow the user to edit the newly entered data. The new record has been added to the database on the server.

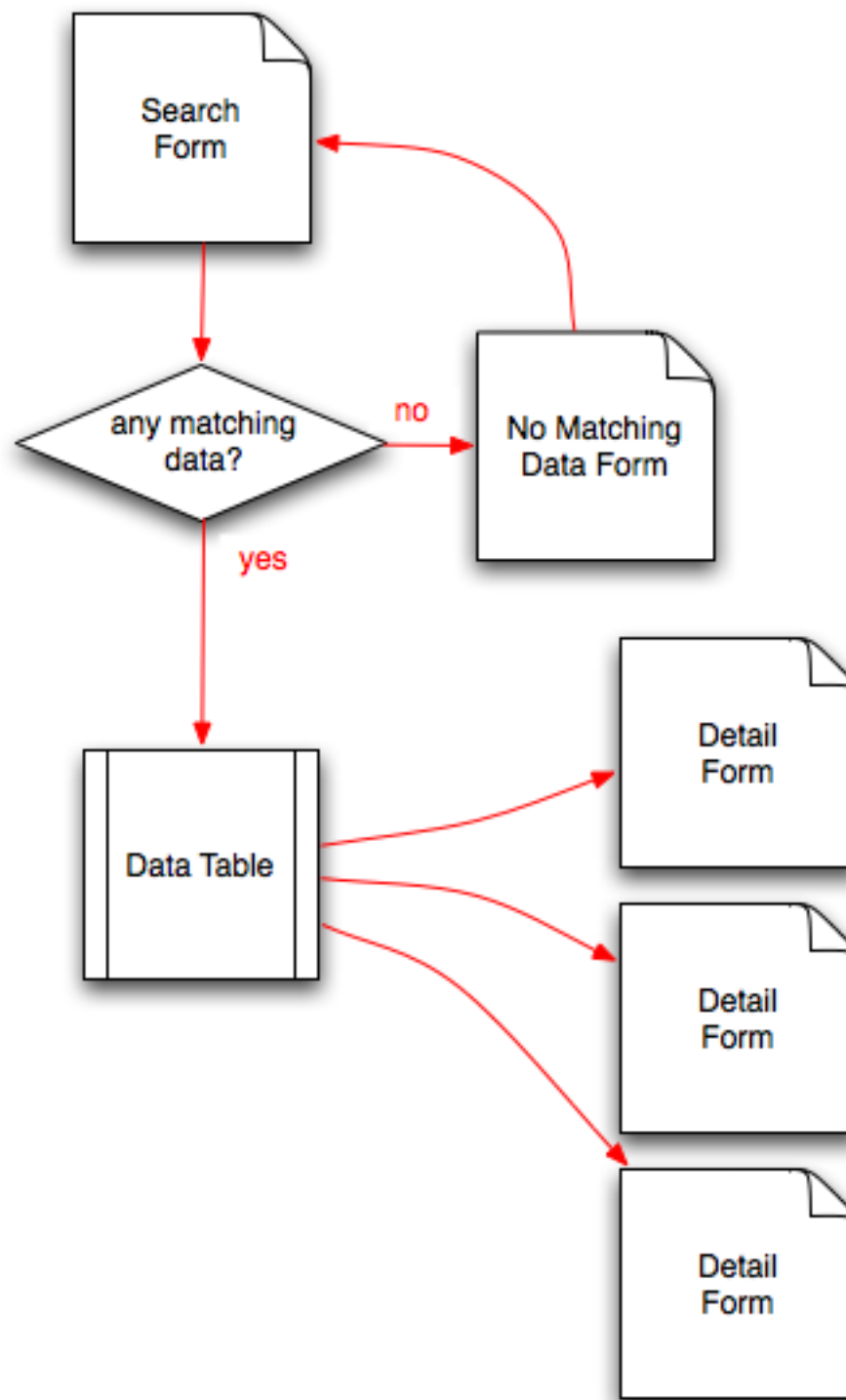
| | | | | | | | | | | | |
|--------|----------|---------------------|------------------|----|-------|---------------------------|---------------------------|--------------------------|-----------------------------|----------------|--------------|
| Thomas | McDaniel | 212 Sagamore Road | Hillburn | NJ | 07041 | G | Early (1840's - 1910's) | Nickel Plate (NKP) | thomas_mcdaniel@micronet.se | (212) 310-4886 | Jun 26, 2006 |
| Alfred | Softon | 487 Starboard Lndg | Fernandina Beach | FL | 32084 | N | Steam (1920's - 1930's) | Great Northern (GN) | alfred96@ic.net.com | (703) 242-5125 | Jun 26, 2006 |
| Pat | Furnare | 40210 Balhazrol Cir | Palm Desert | CA | 92211 | HO | Steam (1920's - 1930's) | Western Pacific (WP) | mwilson@fastisp.net | (760) 345-4890 | Jun 15, 2006 |
| John | Kuttel | 17298 Hampton Ln | Huntington Beach | CA | 92649 | HO | Modern (1960's - Present) | Florida East Coast (FEC) | 125 | (714) 846-6119 | Jul 5, 2006 |
| John | Wilson | 992 Olive Avenue | Long Beach | CA | 98621 | Modern (1960's - Present) | Western Pacific (WP) | Western Pacific (WP) | jwilson@bigisp.net | (562) 555-1212 | Jul 11, 2006 |

12128 visible/12128 total

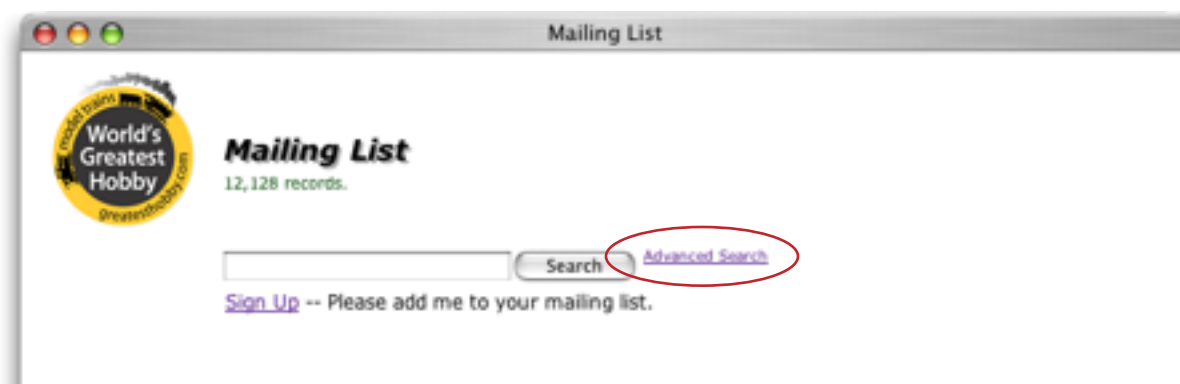
This entire sequence can be set up without any programming simply by setting up forms and filling in a few dialogs.

Search → List → Detail Sequence

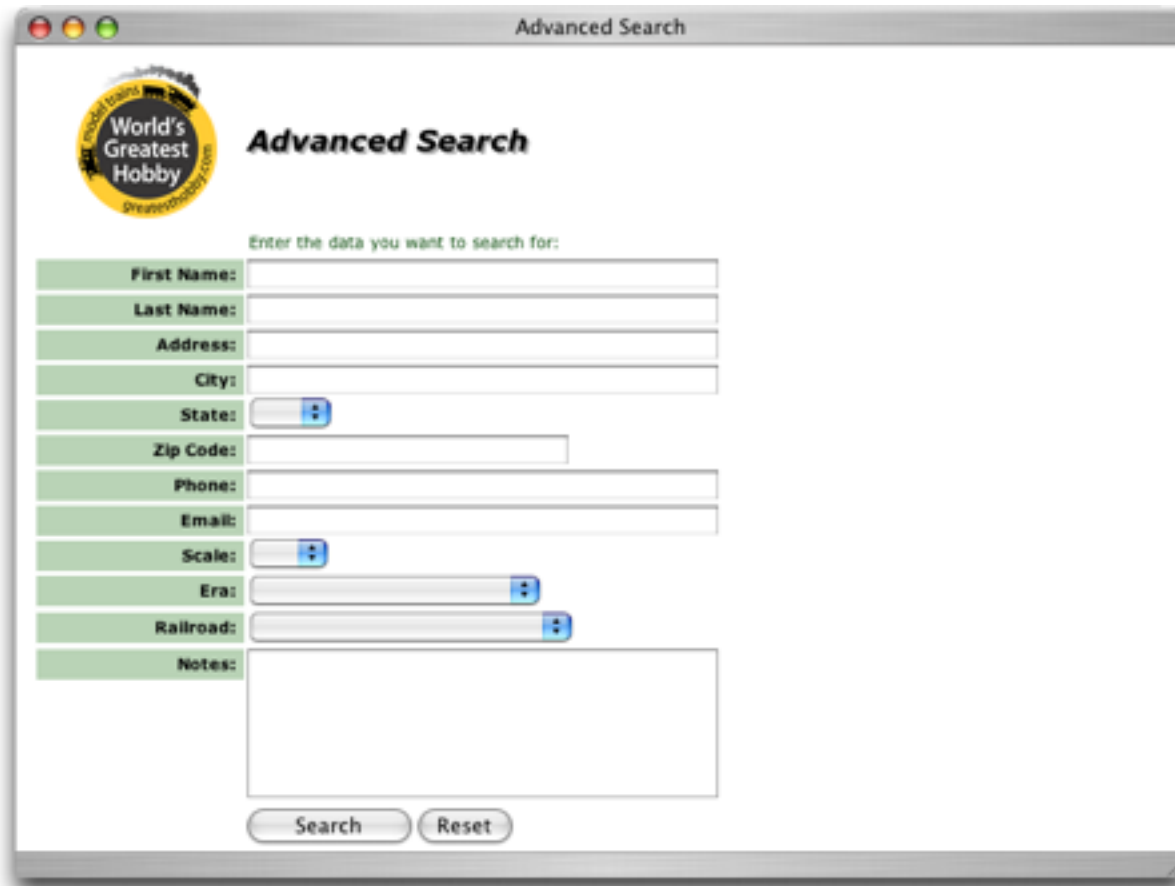
This standard sequence allows a web user to search an online database and display a table listing the matching records. The sequence can stop there, or each table entry can include an automatically generated link that flows to a detailed form for that record. The detailed form can simply display data or it can be designed to allow the user to modify the record. With some additional programming the user can request that the server perform other operations on the data (for example add the item to a shopping cart). Here is the flow chart for this standard sequence.



Let's look at a typical **search-list-detail** sequence in action. The sequence starts by clicking a link from another page on the site.



This link brings up the search form, which generally starts out blank.



Advanced Search

World's Greatest Hobby

Advanced Search

Enter the data you want to search for:

First Name:

Last Name:

Address:

City:

State:

Zip Code:

Phone:

Email:

Scale:

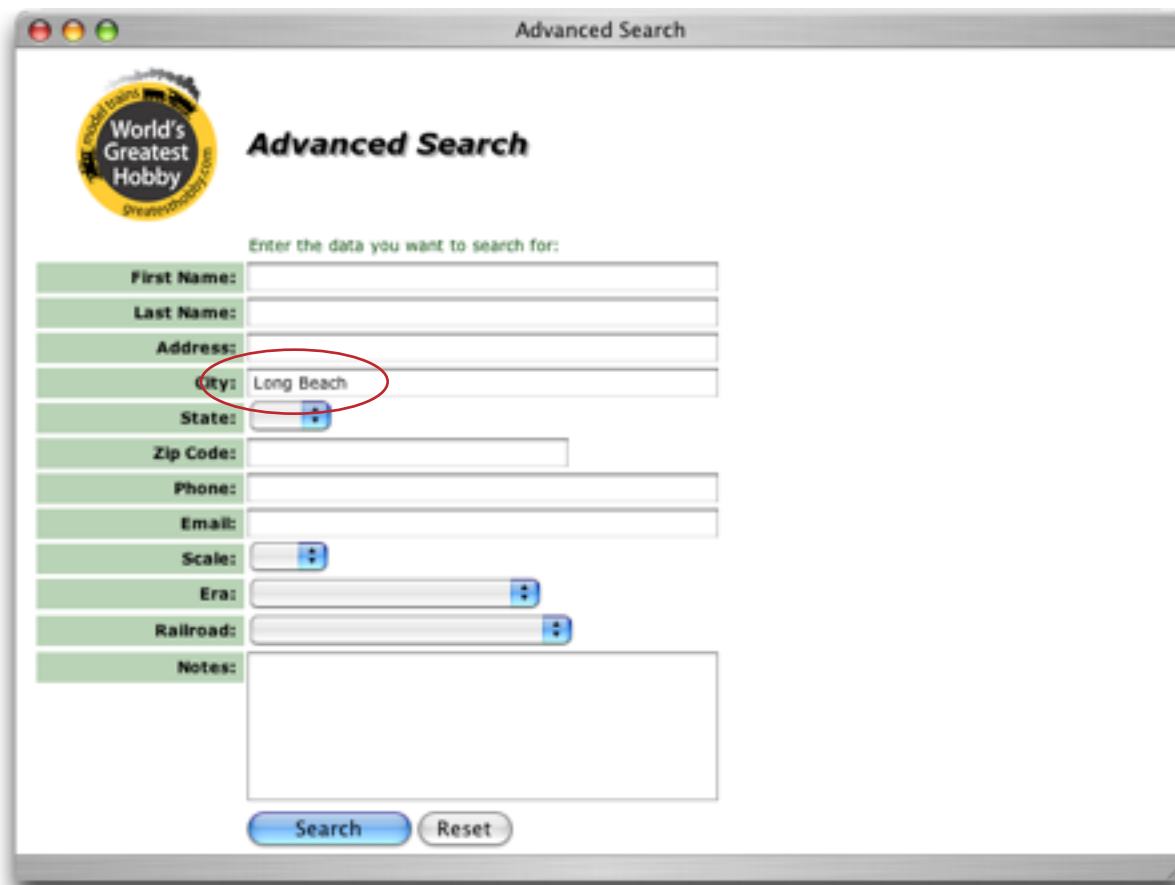
Era:

Railroad:

Notes:

Search Reset

Fill out one or more fields to specify what data to search for, then press the **Submit** button.



Advanced Search

World's Greatest Hobby

Advanced Search

Enter the data you want to search for:

First Name:

Last Name:

Address:

City: Long Beach

State:

Zip Code:

Phone:

Email:

Scale:

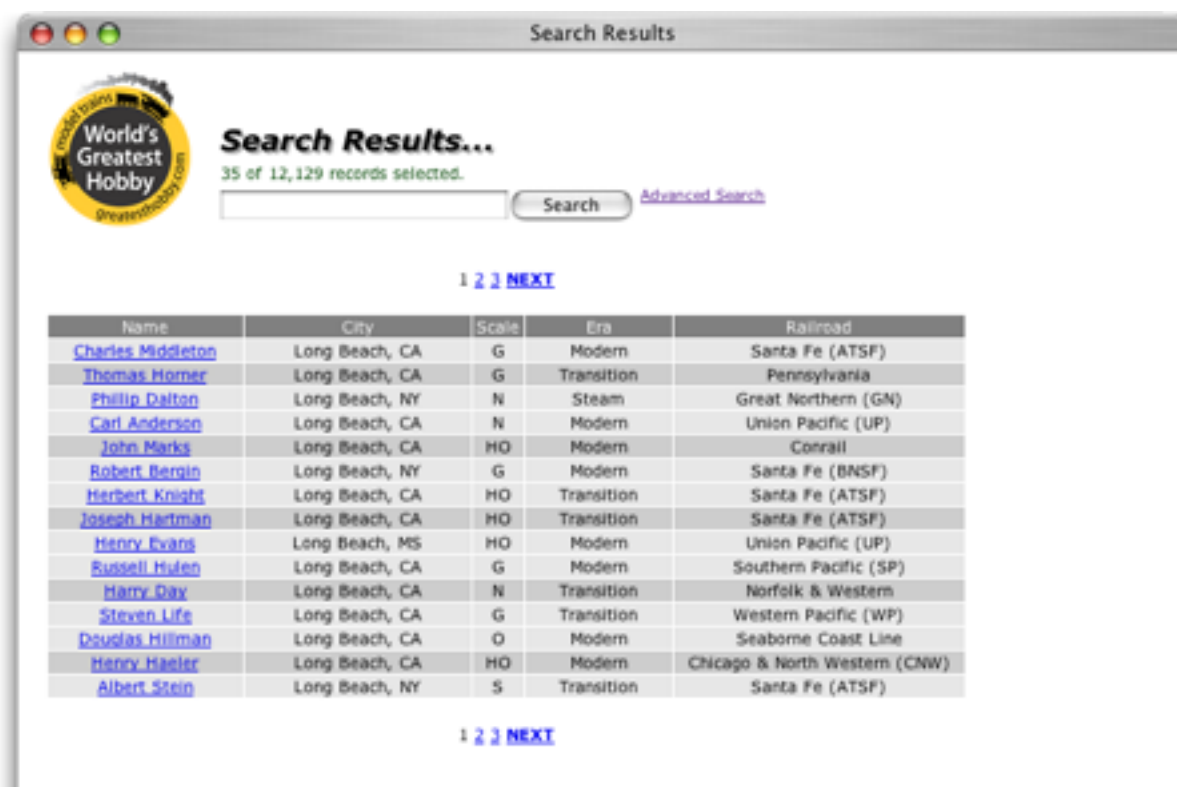
Era:

Railroad:

Notes:

Search Reset

The server will search the database and display a list of all records that match the requested search criteria. (If no records match, a customizable form appears.)



Search Results...
35 of 12,129 records selected.

[Search](#) [Advanced Search](#)

[1](#) [2](#) [NEXT](#)

| Name | City | Scale | Era | Railroad |
|-----------------------------------|----------------|-------|------------|-------------------------------|
| Charles Middleton | Long Beach, CA | G | Modern | Santa Fe (ATSF) |
| Thomas Horner | Long Beach, CA | G | Transition | Pennsylvania |
| Phillip Dalton | Long Beach, NY | N | Steam | Great Northern (GN) |
| Carl Anderson | Long Beach, CA | N | Modern | Union Pacific (UP) |
| John Marks | Long Beach, CA | HO | Modern | Conrail |
| Robert Bergin | Long Beach, NY | G | Modern | Santa Fe (BNSF) |
| Herbert Knight | Long Beach, CA | HO | Transition | Santa Fe (ATSF) |
| Joseph Hartman | Long Beach, CA | HO | Transition | Santa Fe (ATSF) |
| Henry Evans | Long Beach, MS | HO | Modern | Union Pacific (UP) |
| Russell Hulen | Long Beach, CA | G | Modern | Southern Pacific (SP) |
| Harry Day | Long Beach, CA | N | Transition | Norfolk & Western |
| Steven Life | Long Beach, CA | G | Transition | Western Pacific (WP) |
| Douglas Hillman | Long Beach, CA | O | Modern | Seaboard Coast Line |
| Henry Haeler | Long Beach, CA | HO | Modern | Chicago & North Western (CNW) |
| Albert Stein | Long Beach, NY | S | Transition | Santa Fe (ATSF) |

[1](#) [2](#) [NEXT](#)

This table has been configured to display a maximum of 15 records per page. Since there are 35 records selected, the Panorama server automatically splits the display into three pages. You can navigate between the pages using the automatically generated links. Click [NEXT](#) to display the next 15 records.

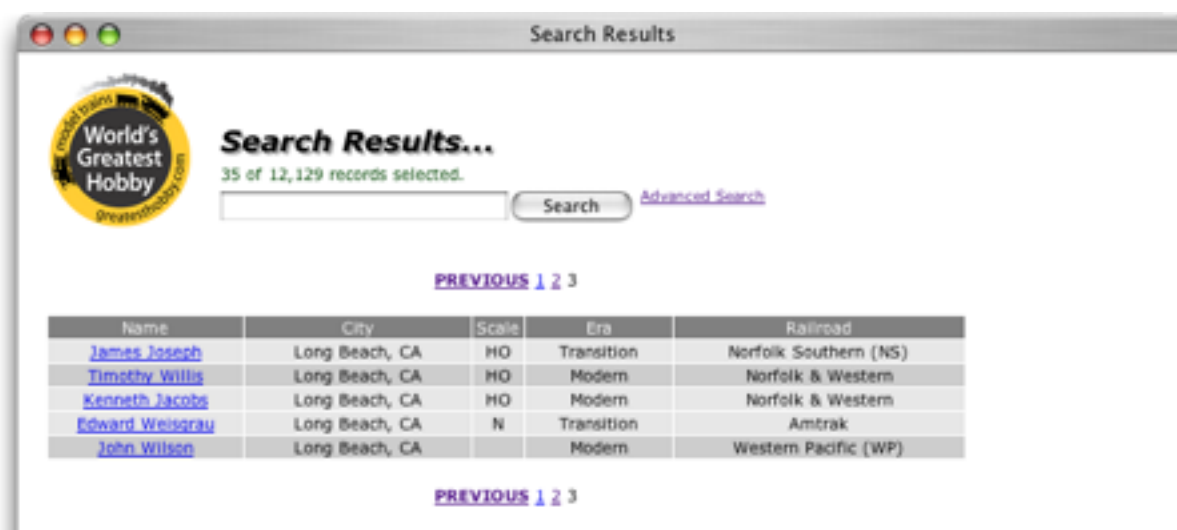


[PREVIOUS](#) [1](#) [2](#) [NEXT](#)

| Name | City | Scale | Era | Railroad |
|-----------------------------------|----------------|-------|------------|-----------------------|
| Timothy Schroeder | Long Beach, CA | N | Steam | Seaboard Coast Line |
| Anthony Gorsky | Long Beach, CA | HO | Transition | Santa Fe (ATSF) |
| Harry Lauer | Long Beach, CA | HO | Transition | Boston & Maine |
| Allen Livingston | Long Beach, CA | N | Transition | Santa Fe (BNSF) |
| Herbert Andrews | Long Beach, CA | G | Transition | Western Pacific (WP) |
| Donald Haas | Long Beach, CA | HO | Transition | Southern Pacific (SP) |
| Jerome Harding | Long Beach, NY | HO | Steam | Union Pacific (UP) |
| Anthony Haas | Long Beach, CA | G | Transition | Great Northern (GN) |
| Jack Richter | Long Beach, CA | N | Steam | Milwaukee (MILW) |
| Gerald Wade | Long Beach, CA | N | Transition | Santa Fe (ATSF) |
| Gary Pearson | Long Beach, CA | N | Transition | Great Northern (GN) |
| Brian Terry | Long Beach, CA | HO | Modern | Nickel Plate (NKP) |
| Peter McLaughlin | Long Beach, CA | HO | Transition | Alaska Railroad |
| William Luff | Long Beach, CA | HO | Transition | Union Pacific (UP) |
| Warren Hackett | Long Beach, CA | O | Steam | Union Pacific (UP) |

[PREVIOUS](#) [1](#) [2](#) [NEXT](#)

Click [NEXT](#) again to display the final 5 records (or you can simply press [3](#) to go right to the end).



[PREVIOUS](#) [1](#) [2](#) [3](#)

| Name | City | Scale | Era | Railroad |
|---------------------------------|----------------|-------|------------|-----------------------|
| James Joseph | Long Beach, CA | HO | Transition | Norfolk Southern (NS) |
| Timothy Willis | Long Beach, CA | HO | Modern | Norfolk & Western |
| Kenneth Jacobs | Long Beach, CA | HO | Modern | Norfolk & Western |
| Edward Weisgrau | Long Beach, CA | N | Transition | Amtrak |
| John Wilson | Long Beach, CA | | Modern | Western Pacific (WP) |

[PREVIOUS](#) [1](#) [2](#) [3](#)

This table has been configured to display links for each record (links are underlined above). Clicking a link brings up a detail form for this record. For example, clicking [John Wilson](#) displays the detail form for this record.

World's Greatest Hobby
Sign Up for Mailing List

Make any necessary changes and press the **Submit** button.

| | |
|-------------|--|
| First Name: | John |
| Last Name: | Wilson |
| Address: | 992 Olive Avenue |
| City: | Long Beach |
| State: | CA |
| Zip Code: | 98621 |
| Phone: | (562) 555-1212 |
| Email: | jwilson@bigisp.net |
| Scale: | <input type="radio"/> G <input type="radio"/> O <input type="radio"/> S <input type="radio"/> HO <input type="radio"/> N <input type="radio"/> Z |
| Era: | Modern (1960's - Present) |
| Railroad: | Western Pacific (WP) |
| Notes: | |

Submit Reset

This form displays the record and in this case also allows the user to modify the data. Pressing the Submit button updates the database on the server with the new data and displays a customizable “thank-you” form (similar to the data entry sequence). This entire sequence can be set up without any programming simply by setting up forms and filling in a few dialogs.

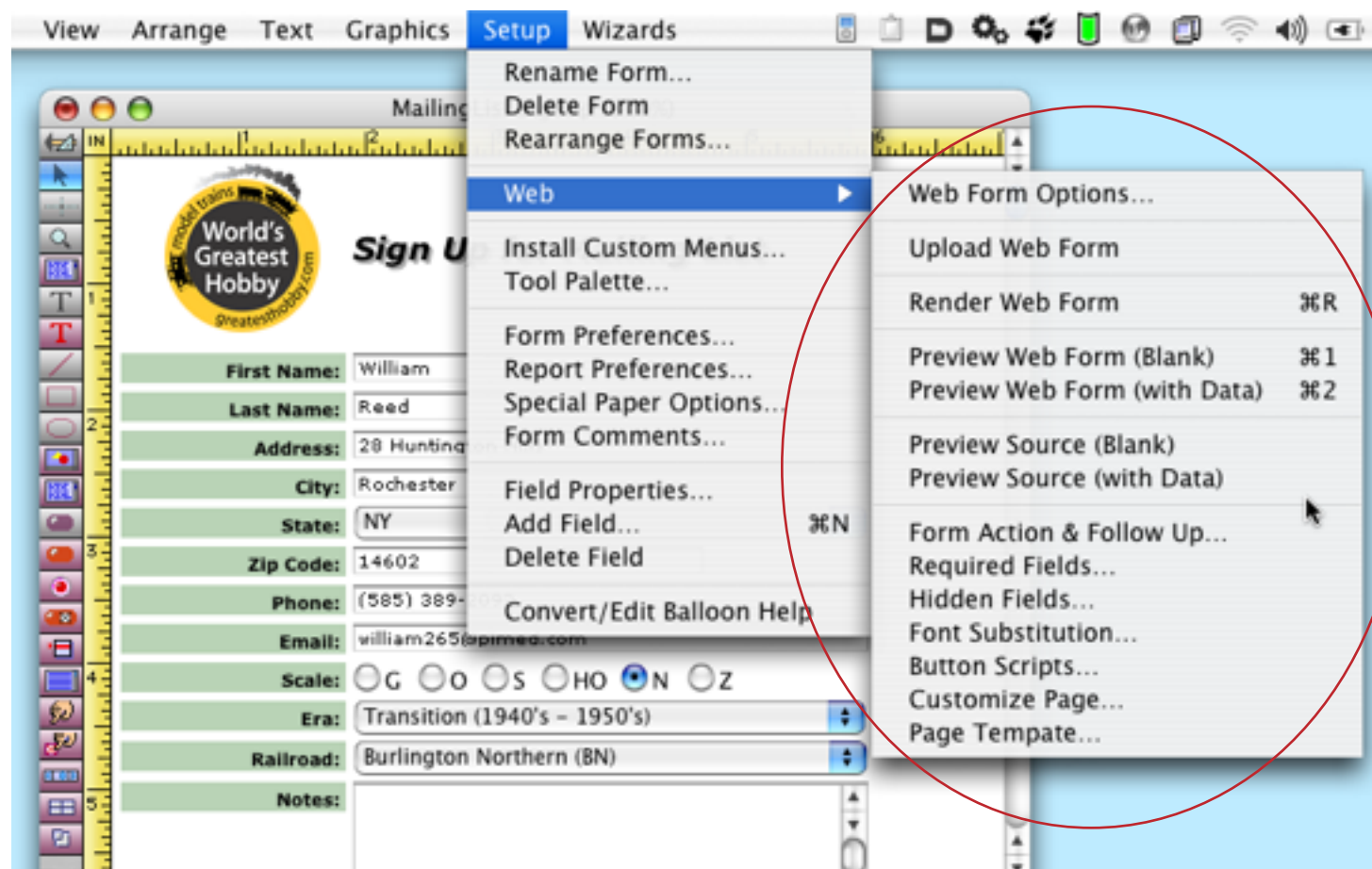
Chapter 6: Web Forms



The Panorama Enterprise Edition Server can publish any kind of HTML web page, but there are two basic building blocks that are the most common elements of any dynamic site — forms and tables. This chapter describes forms, while the next chapter describes tables.

Converting a Panorama Form into a Web Form

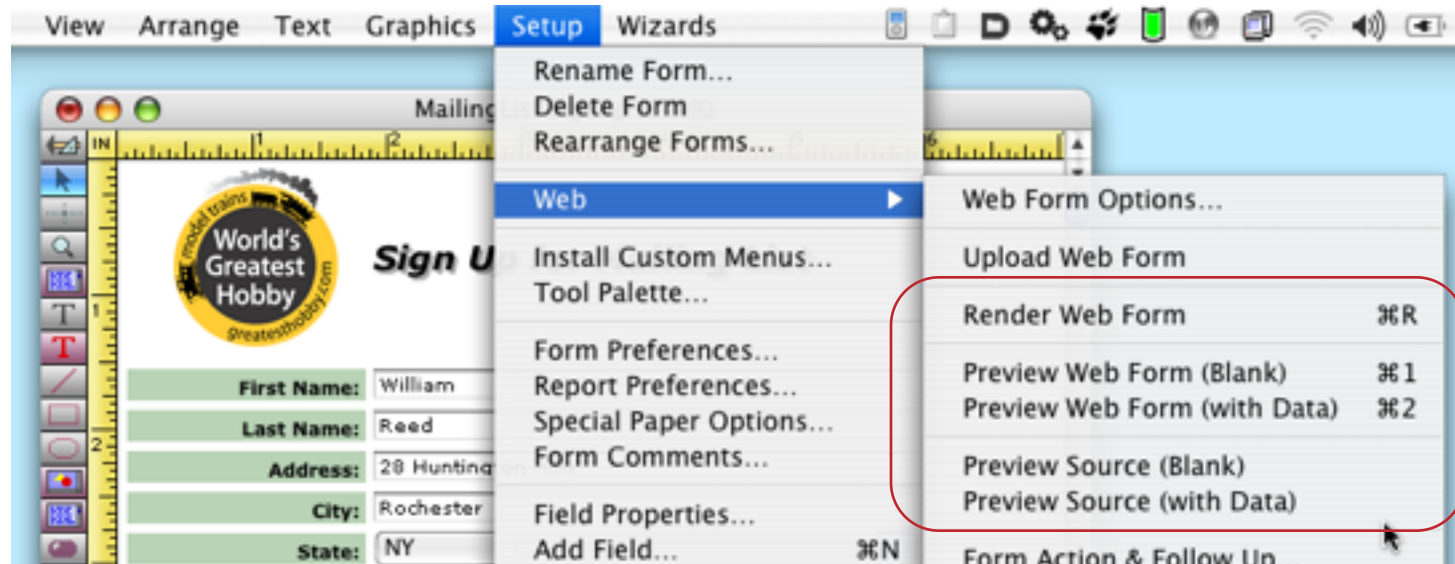
If you've been using Panorama for a while you are probably familiar with creating forms using Panorama's graphics tools. Starting with version 5.5 Panorama has a special wizard that allows you to convert forms created in Panorama into forms that can be used on the web. You can access this wizard directly, but usually you'll access it's features directly through the Web submenu of the Setup menu while in graphics mode.



The commands in the Web menu are used to convert Panorama forms into web page forms and to customize how that conversion is performed.

Converting a Panorama Form into a Web Page Form

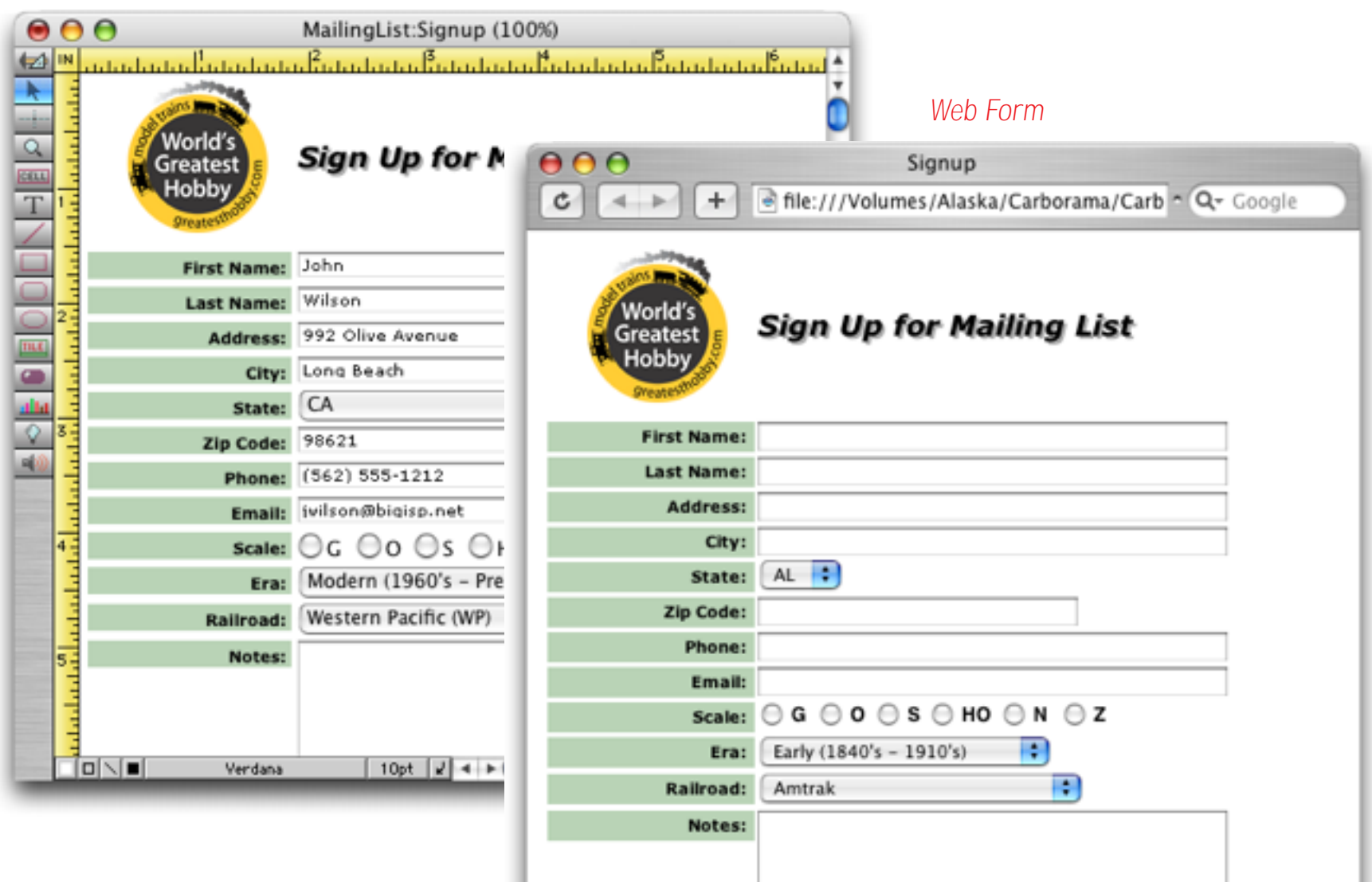
Panorama can convert a form for web use directly from graphics mode (this is sometimes called “rendering” the form). Use the **Web** submenu of the **Setup** menu to convert the current form. (Note: The current database does not have to be installed on a server to convert the form, though of course you cannot actually use the HTML form unless the database has been published for the web, see “[Uploading the Database to the Server](#)” on page 194. In other words, you can set up your HTML forms while the database is still in single user mode.)



The **Render Web Form** command performs the conversion to HTML format. Use this command if you don't need to see what the finished HTML page will look like.

The **Preview Web Form (Blank)** command renders (converts) the form to HTML, then opens your web browser and previews what the form will look like without any data (use this option if the form is to be used for data entry or searching) (Note: This previewed version of the form is not functional, but it does show you exactly what this form will look like in this browser).

Panorama Form



The **Preview Web Form (with Data)** command renders (converts) the form to HTML, then opens your web browser and previews what the form will look like with data (the data in the current record is used for the preview). (Note: This previewed version of the form is not functional, but it does show you exactly what this form will look like in this browser).

Panorama Form

A screenshot of a Panorama Form window titled "MailingList:Signup (100%)". The form is displayed in a window with a ruler at the top and a toolbar on the left. The form contains the following data:

| | |
|-------------|---|
| First Name: | John |
| Last Name: | Wilson |
| Address: | 992 Olive Avenue |
| City: | Long Beach |
| State: | CA |
| Zip Code: | 98621 |
| Phone: | (562) 555-1212 |
| Email: | jwilson@bigisp.net |
| Scale: | <input type="radio"/> G <input type="radio"/> O <input type="radio"/> S <input type="radio"/> H |
| Era: | Modern (1960's - Pres |
| Railroad: | Western Pacific (WP) |
| Notes: | |

Web Form

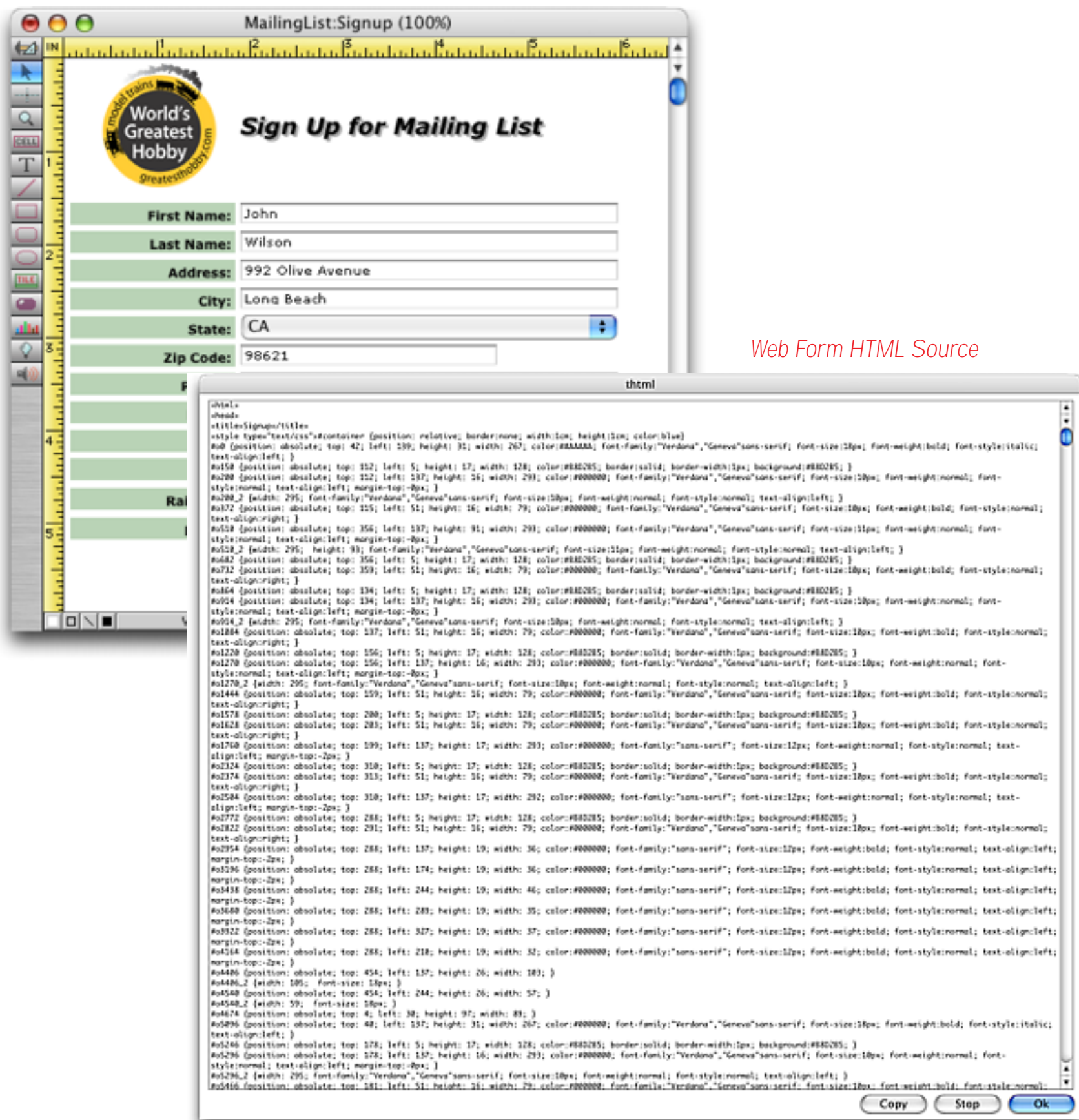
A screenshot of a Web Form window titled "Signup". The form is displayed in a browser window with a search bar at the top. The form contains the following data:

| | |
|-------------|--|
| First Name: | John |
| Last Name: | Wilson |
| Address: | 992 Olive Avenue |
| City: | Long Beach |
| State: | CA |
| Zip Code: | 98621 |
| Phone: | (562) 555-1212 |
| Email: | jwilson@bigisp.net |
| Scale: | <input type="radio"/> G <input type="radio"/> O <input type="radio"/> S <input type="radio"/> HO <input type="radio"/> N <input type="radio"/> Z |
| Era: | Modern (1960's - Present) |
| Railroad: | Western Pacific (WP) |
| Notes: | |

Buttons: Sign Up, Reset

The Preview Web Form (Blank) and Preview Web Form (with Data) commands render the form to HTML, then open a window to display the actual HTML source code. You can examine this code or even copy it to a separate text editor.

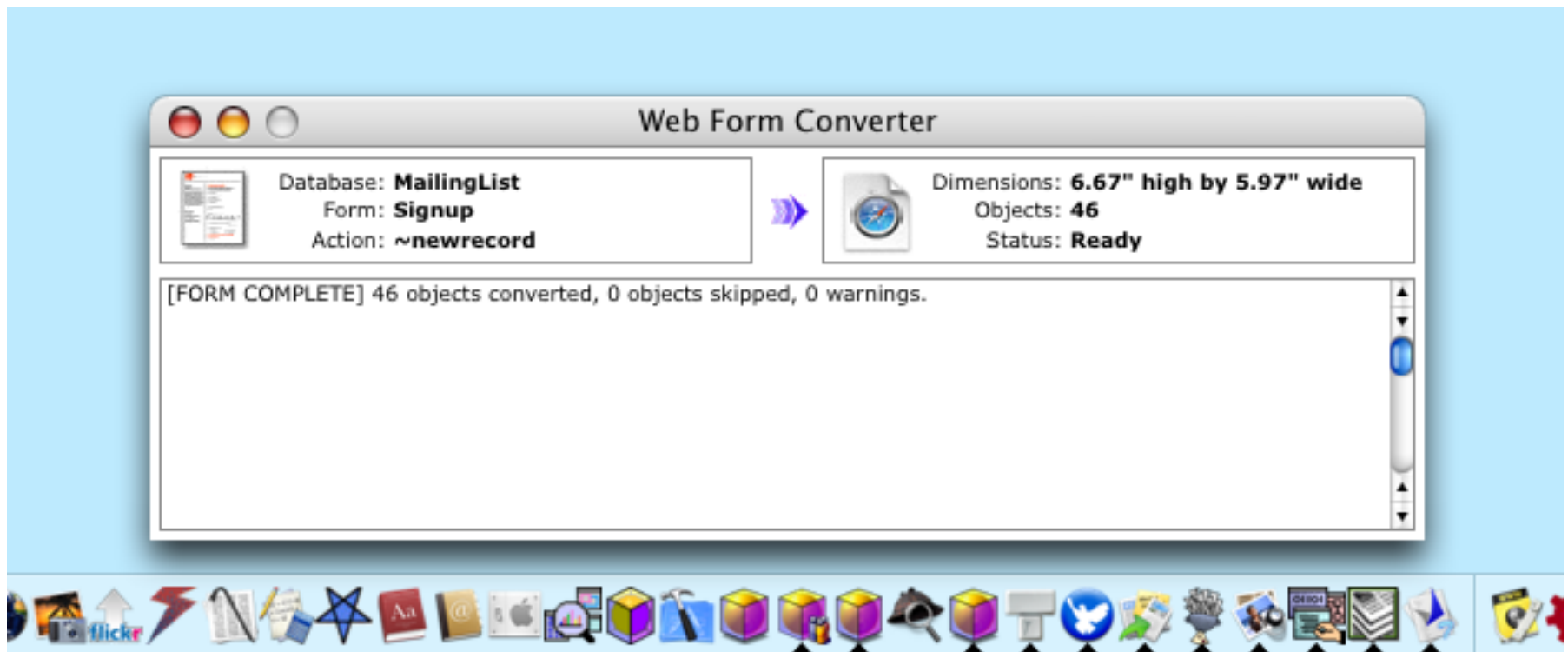
Panorama Form



Web Form HTML Source

The Web Form Converter Wizard

Whenever you render a form to HTML Panorama will automatically open the Web Form Converter wizard. This wizard will appear centered at the bottom of the screen.



The wizard displays the status of last form that was rendered, including any errors or warnings (see “[Conversion Limitations](#)” on page 236). You can close this wizard any time you want — it will re-open the next time you render a form to HTML. You can also manually open this wizard using the **Web Form Options...** command in the **Web** submenu of the **Setup** menu.

Preparing a Form

Although any form can be converted for use on the web, you'll usually want to build forms specifically designed for the web — just as you probably build forms specifically for use as printed reports. As described below, some Panorama features cannot be converted for the web, so these elements of the form will not be included in the converted web version of the form. You may also want to adjust fonts and images for the best presentation on web browsers.

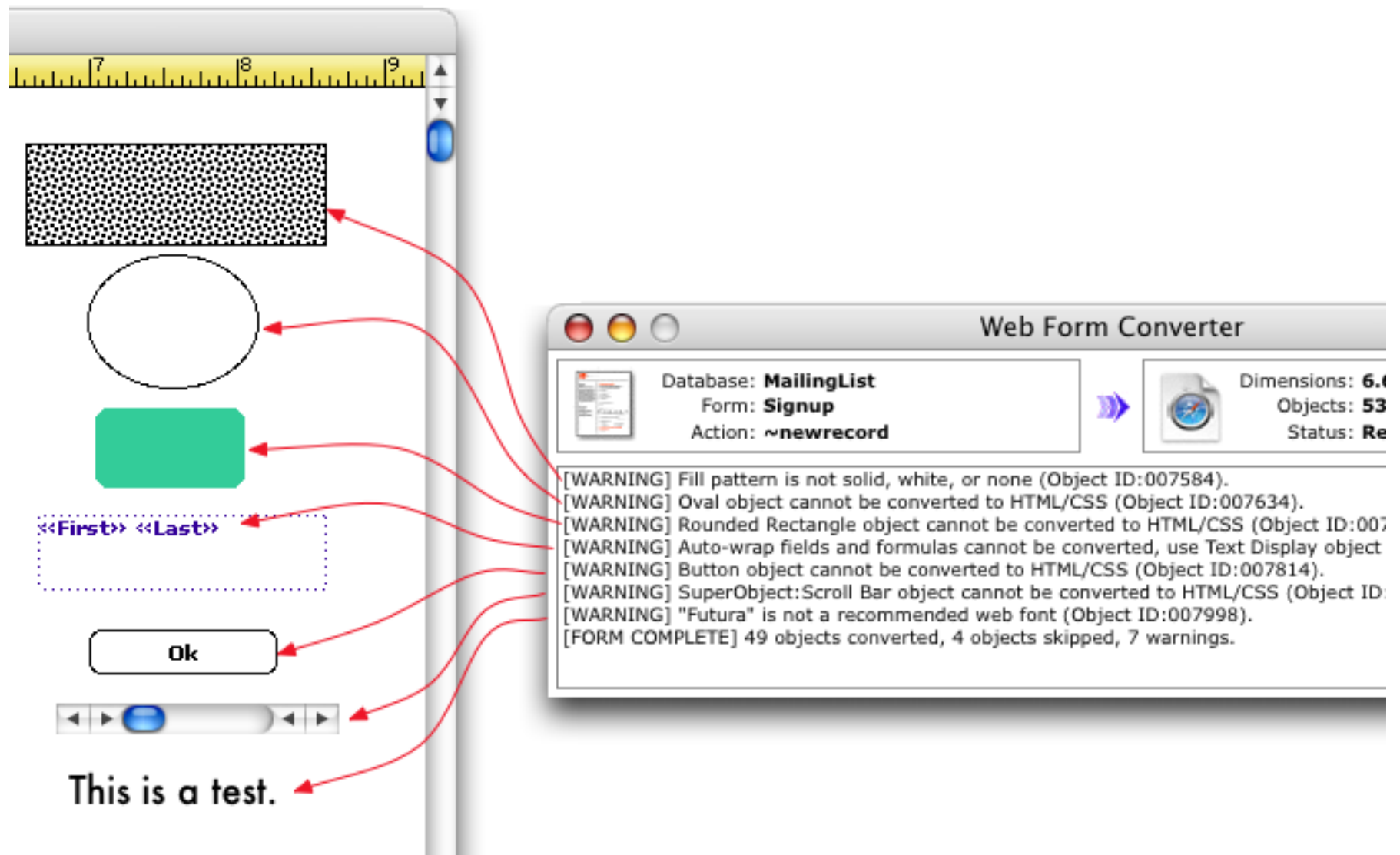
Conversion Limitations

The table below lists types of Panorama form objects that can be converted to equivalent web entities, along with any limitations. In general, Panorama includes many options for customizing buttons, pop-up menus, text editing, etc., but most of these don't work with web browsers. If you stick to standard appearance objects you will usually be ok.

| Object | Limitations |
|-------------------|---|
| Rectangle | Fill pattern must be solid, outline or hollow, pen pattern must be solid or none. |
| Line | Horizontal lines work very well (they are implemented with the <code><hr></code> tag). Vertical lines are possible, but will be twice the width specified (the minimum width of a vertical line is two pixels). Diagonal lines are not supported. |
| Click Text | See " Font selection " on page 255. |
| Autowrap Text | Cannot contain embedded fields or formulas (use Text Display SuperObject instead). Also see " Font selection " on page 255. |
| Text Display | Cannot display a scroll bar, also see " Font selection " on page 255. |
| Data Cell | Uses default browser appearance for text editing objects (see " Data Cells and Text Editor SuperObjects " on page 238), |
| Text Editor | Uses default browser appearance for text editing objects, cannot control borders, scroll bars, etc. (see " Data Cells and Text Editor SuperObjects " on page 238), |
| Push Button | Buttons cannot trigger a procedure, a button either submits the form data or resets the form data (if it is called Reset). |
| Data Button | Only standard checkbox or radio button styles are supported, cannot trigger a procedure. |
| Pop-Up Menu | Only standard pop-up appearance is supported. No multi-column menus. |
| List Super Object | Cannot trigger a procedure. Also see " Font selection " on page 255. |
| Super Flash Art | Can only display images from web, not from local hard drive. No scaling, alignment must be to upper left, no scrolling. See " Displaying Images in a Web Form " on page 264. |

Object types not listed in the table above cannot be converted, and will be skipped completely when converting a form to HTML. This include tiles, ovals, round rectangles, super matrix, auto grow, "classic" button, word processor, charts, "classic" flash art, pictures, flash art buttons, sticky push buttons and scroll bars.

If a form contains one or more objects that cannot be fully converted, these objects will be listed in the **Web Form Converter** wizard when the form is rendered. For example, consider this form, which has several objects that cannot be converted or can only be partially converted.

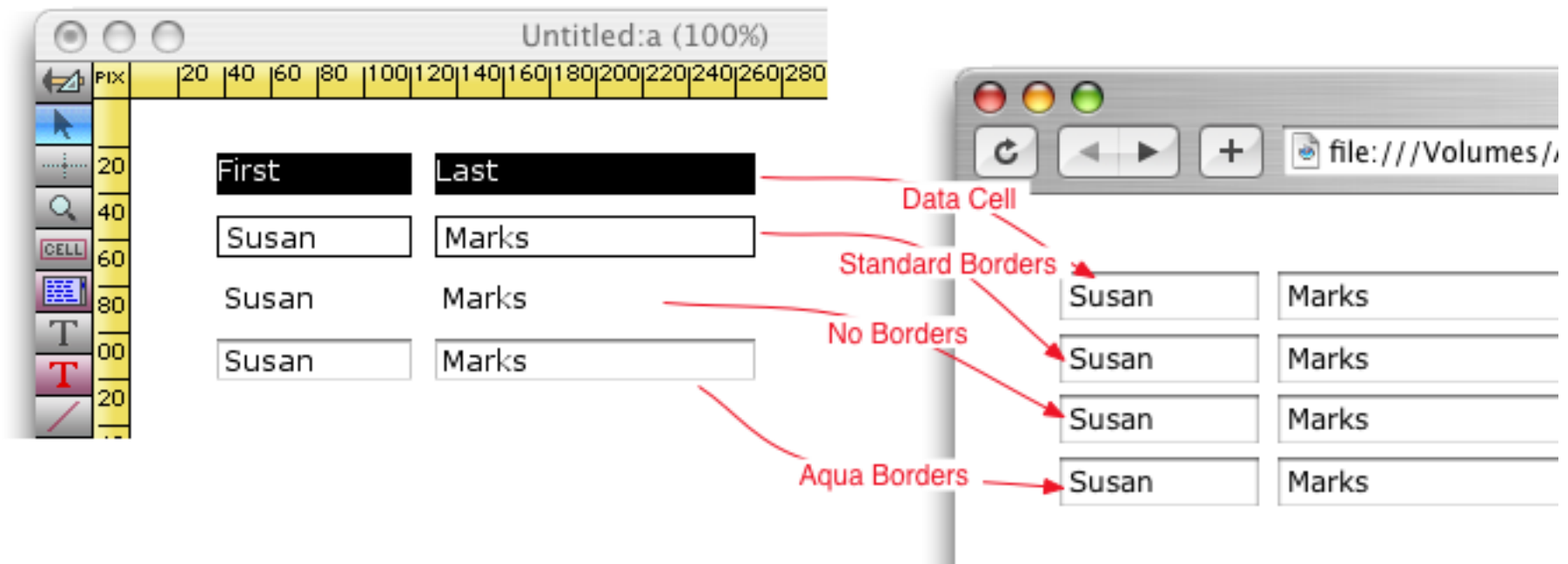


(Of course the objects with problems will not usually line themselves up neatly on the edge of the form! If you have trouble determining which objects are causing the problems you can locate the objects by object ID using the **Form Explorer** wizard.)

Fields and Variables in Web Forms

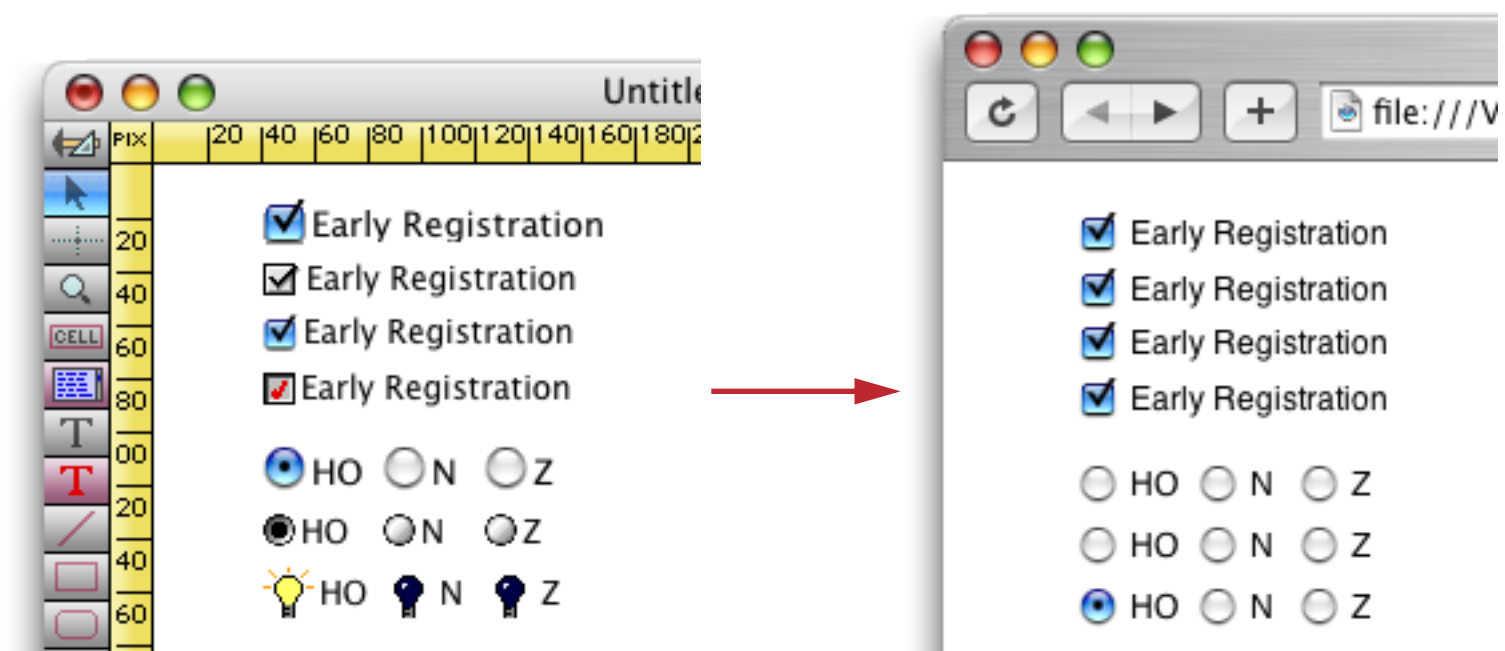
Most forms handle data in fields and variables. Data can be displayed using formulas in Text Display SuperObjects. Data editing is done with data cells, Text Editor SuperObjects, checkboxes, radio buttons and pop-up menus.

Data Cells and Text Editor SuperObjects. Panorama has many formatting options for editing data. You can use data cells, or Text Editor SuperObjects with various types of borders. Web browsers, however, only have a single formatting option for editing data. No matter what option you use in Panorama, the result looks the same on the web browser. The illustration below shows four different data editing styles on a Panorama form on the left. The right side of the illustration shows these same four sets of data editing objects rendered as a web page — now they all look identical.



The moral of the story? Go ahead and choose any style you like for your Panorama form, but know that when the form is rendered as a web page it may look completely different (and may look different in different browsers and on different platforms, as well).

Data Buttons (Checkboxes and Radio Buttons). Panorama has many display options for checkboxes and radio buttons. Web browsers, however, support only plain buttons. No matter what option you select for your radio button or checkbox in Panorama, it will always be converted to a generic button when the form is rendered as a web page.

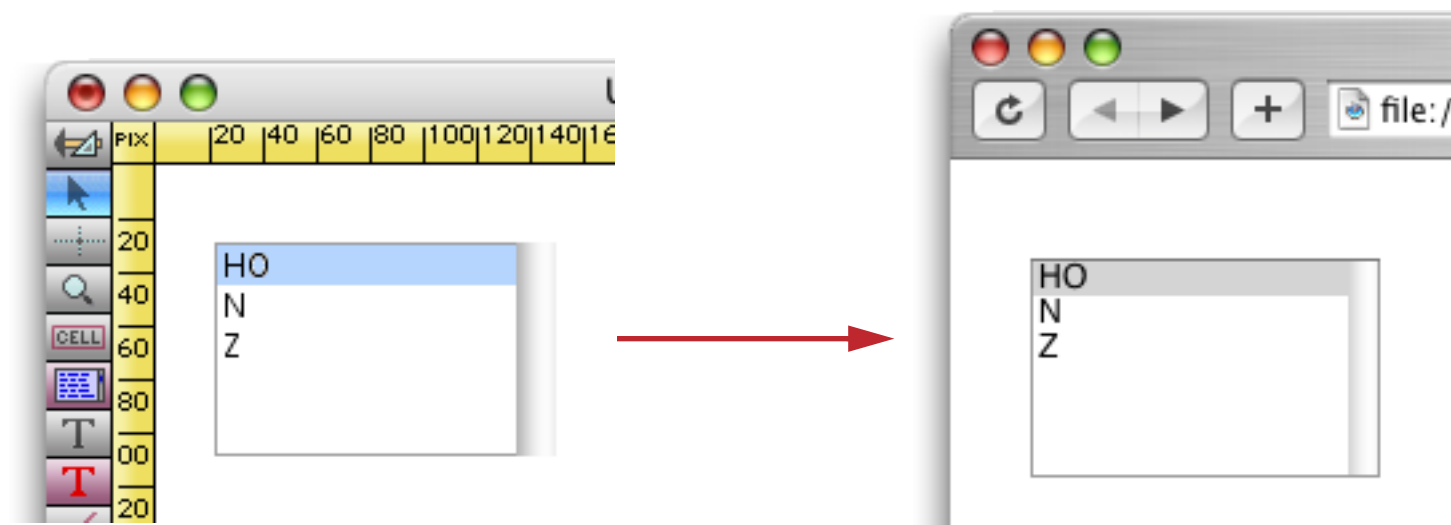


Panorama supports multiple checkboxes assigned to a single field or variable, allowing a single field or variable to contain multiple values (pizza toppings, for instance). Web browsers do not support this, so a Panorama form that uses this feature will not work properly if converted to a web page. Instead you must have a separate field or variable for each option.

Pop-Up Menus. Like the other data entry objects, Panorama has many options for pop-up menus. As you can imagine, none of these are supported by web browsers, so all pop-up menus are converted to generic system standard pop-up menus. In addition, the width of the pop-up menu is determined automatically from the length of the longest pop-up option. So you may find that your pop-up menu gets narrower or wider when the form is rendered as a web page.

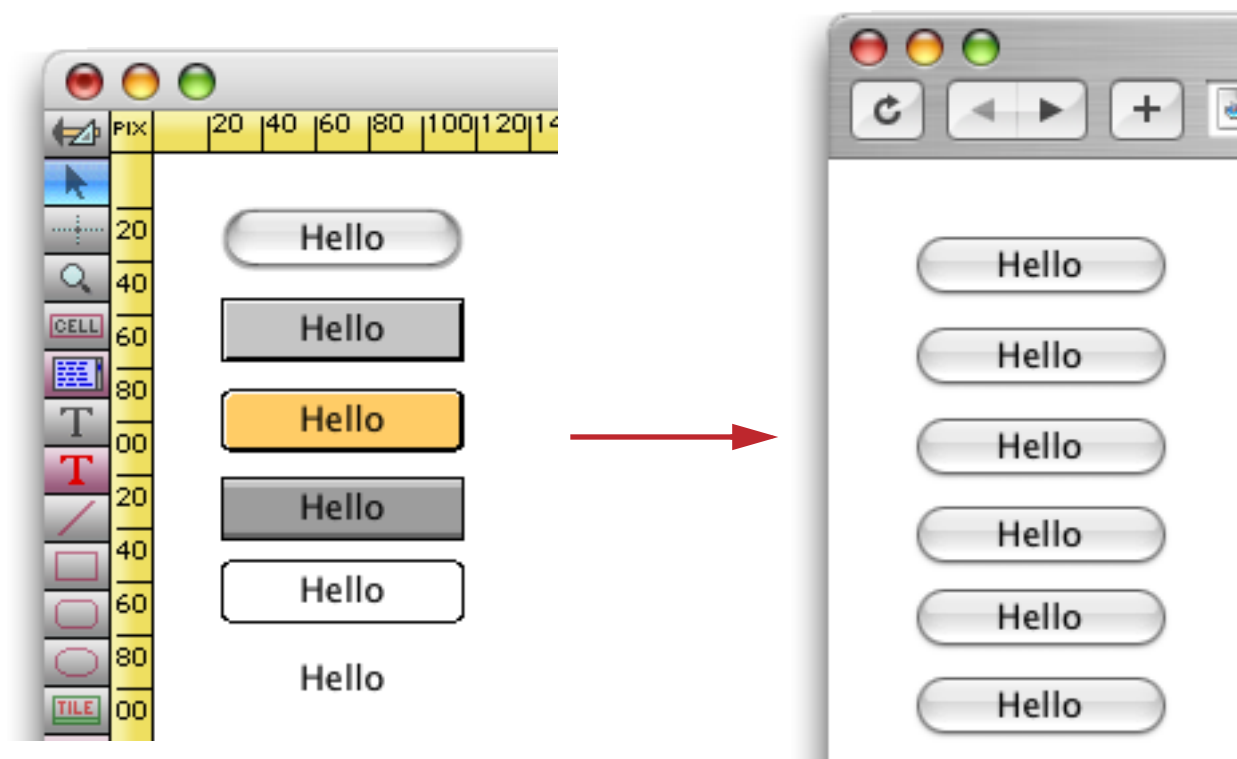
When used in a Panorama form the pop-up menu option list can be calculated on the fly when the menu is clicked on. This allows the menu to change depending on current conditions. Web pop-up menus don't allow this, so you should not use any fields or variables in the formula used to calculate pop-up menus.

Lists. Lists generally look pretty similar when converted from a Panorama form to a web form.



When used in a regular form Panorama allows the list of items to be included in the list to be calculated on the fly, or even to be created by scanning the database. These options are not allowed in a web form — the list must be a fixed list of options with no fields or variables.

Push Buttons. In Panorama forms push buttons can come in many sizes, shapes, styles and colors — even transparent! Push buttons on web pages come in only one size and style (though you can make buttons from an image, see “[Making an Image Link to Another Page](#)” on page 267). No matter what option you select for your push button in Panorama, it will always be converted to a standard generic button when the form is rendered as a web page.



Panorama buttons normally trigger procedures, but in a web form they normally submit the contents of the form to the server. Unless you specify otherwise (as described in a moment), any button on a form will act as a **Submit** button. There are two exceptions. If the button title is **Reset**, the button will clear the contents of the form (actually set the form back to its default state.) The second exception involves the **Button Scripts** dialog, which allows one or more buttons to trigger JavaScript code. See “[Triggering JavaScript with a Button](#)” on page 270 for more information.

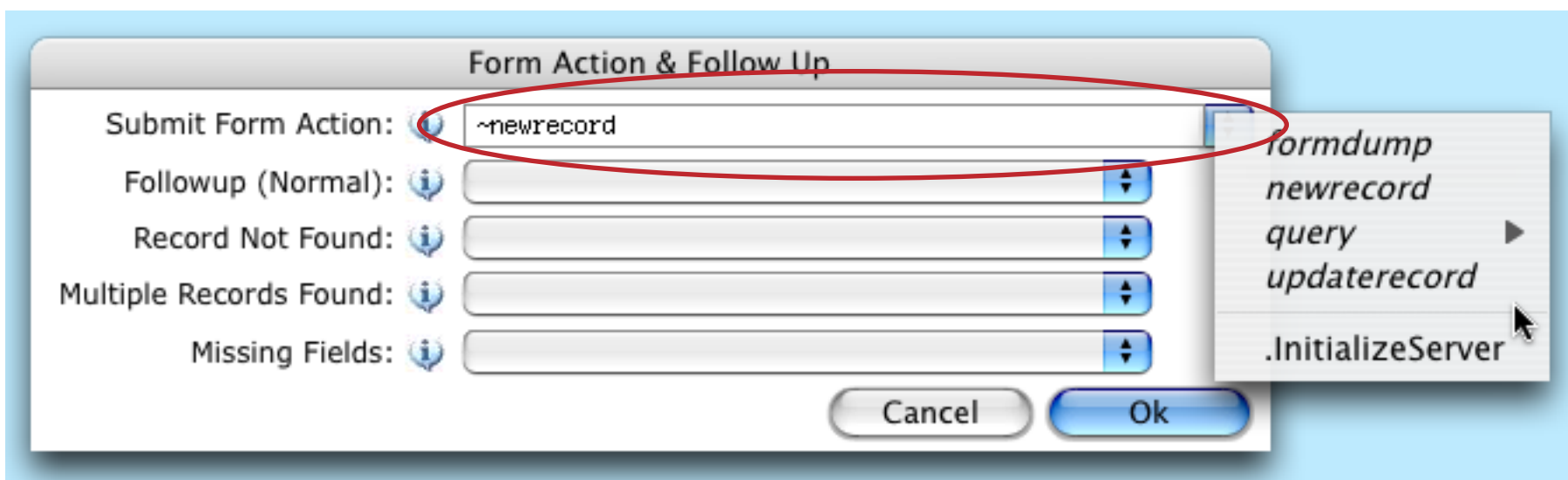
Form Actions and Sequence

On the web, a form doesn't exist in a vacuum, but is part of a larger sequence. The basic sequence is:

- Form is displayed.
- User fills in form
- User submits form data to server
- Server processes the submitted data and displays the next page

The following section discusses the last step in this sequence — processing the form data and displaying the next page. This processing is called the **form action**. The Panorama server has several built in standard actions that you can use (for entering new data, querying data, etc.) or you can create your own custom action by writing a program.

To specify what action should be taken when the form is submitted, use the **Form Action & Follow Up** command in the **Web** submenu of the **Setup** menu (see “[Converting a Panorama Form into a Web Form](#)” on page 231).



The first option in this dialog is the **Submit Form Action**. You can type in the name of the action you want to use, but usually you'll select the action you want from the pop-up menu on the right. This menu lists the 4 standard built in actions along with all of the procedures in this database. Whatever you select (standard action or procedure) will be triggered when the form is submitted to process the data and display the next page. (The other four options in this dialog are used with various standard actions, and are described in the following sections.)

Standard Form Action — FORMDUMP

The **formdump** standard action simply takes the data submitted and generates a web page that lists all of the values that were submitted (including hidden values, see “[Hidden Data](#)” on page 271). This is primarily useful for debugging — you wouldn’t normally use this action in a production web site. For example, suppose the form below has been set up to use the **formdump** action.

World's Greatest Hobby
Sign Up for Mailing List

| | |
|-------------|---|
| First Name: | Scott |
| Last Name: | August |
| Address: | 1010 Wilshire Blvd |
| City: | Los Angeles |
| State: | CA |
| Zip Code: | 91206 |
| Phone: | (323) 555-1212 |
| Email: | scottaugust@cedarmesa.com |
| Scale: | <input type="radio"/> G <input type="radio"/> O <input type="radio"/> S <input type="radio"/> HO <input checked="" type="radio"/> N <input type="radio"/> Z |
| Era: | Transition (1940's - 1950's) |
| Railroad: | Santa Fe (ATSF) |
| Notes: | |

Sign Up Reset

When the **Sign Up** button is pressed the form is submitted and the fields and their values are displayed.

Panorama Database Query:MailingList/formdump

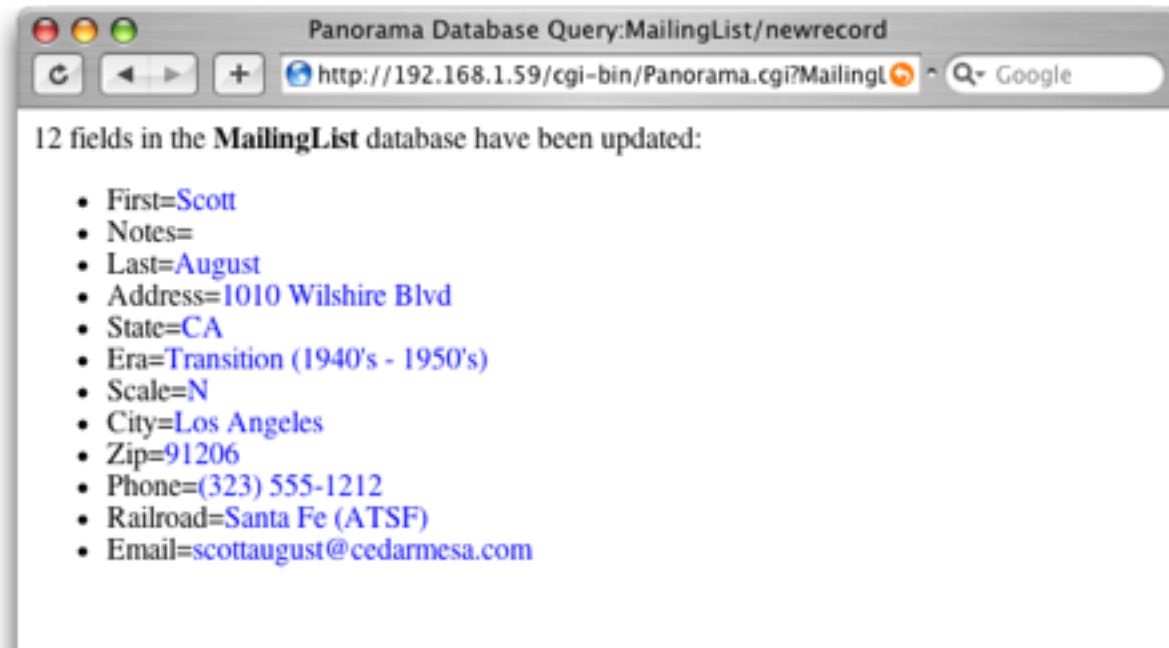
12 fields submitted to the *MailingList* database:

- First=Scott
- Notes=
- Last=August
- Address=1010 Wilshire Blvd
- State=CA
- Era=Transition (1940's - 1950's)
- Scale=N
- City=Los Angeles
- Zip=91206
- Phone=(323) 555-1212
- Railroad=Santa Fe (ATSF)
- Email=scottaugust@cedarmesa.com

Note: If you don’t assign an action to a form the **formdump** action is assigned by default.

Standard Form Action — NEWRECORD

The **newrecord** standard action takes the submitted data and adds it to the database (in a new record). It then displays a “thank you” page that by default looks like this:



Setting Up a Custom Response Page. For most applications you will probably want to design your own “thank you” page. To do this, create another Panorama form and convert it to a web page. Here’s an example of a typical “thank you” form. This form uses Text Display SuperObjects to display the newly added data.

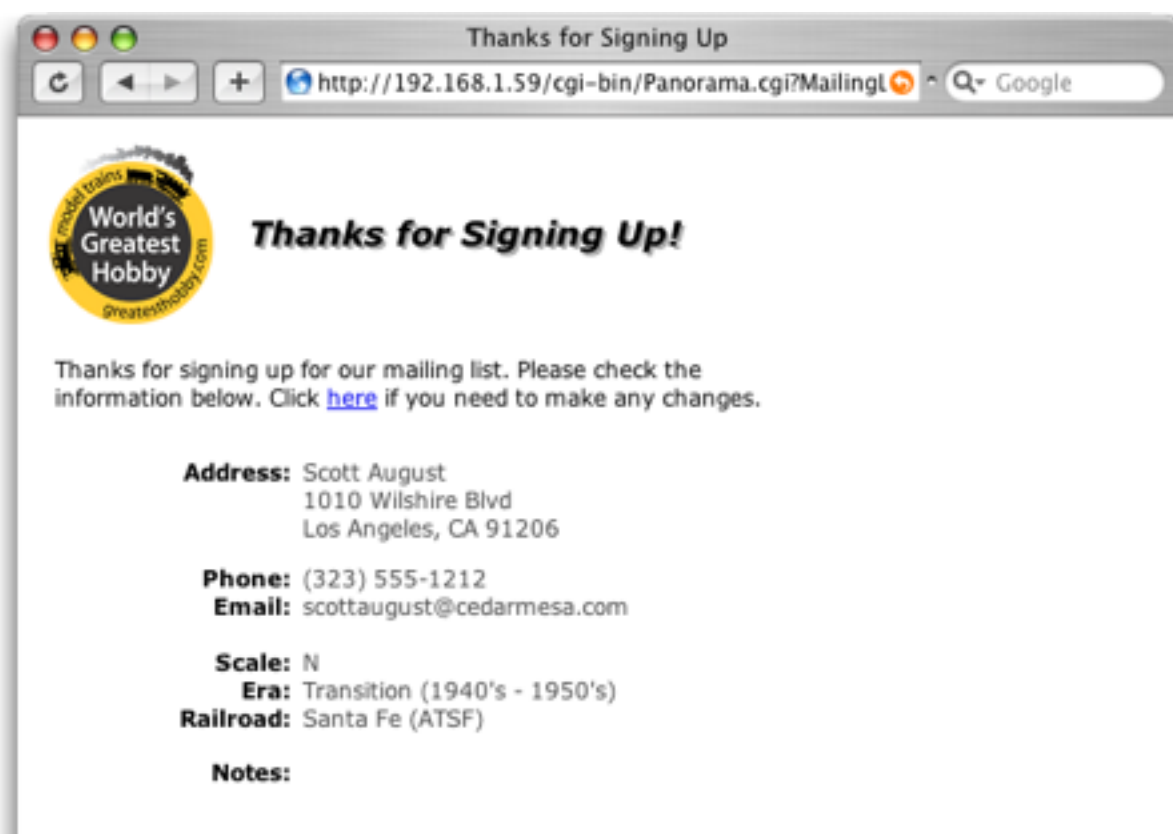


To tell the server to display this “thank you” form use the **Form Action & Follow Up** dialog in the **Web** sub-menu of the **Setup** menu (see “[Converting a Panorama Form into a Web Form](#)” on page 231). Select the form from the **Followup (Normal)** pop-up menu.



click here for pop-up list of forms in this database

Now when new data is entered the server will respond with this custom “thank you” page.

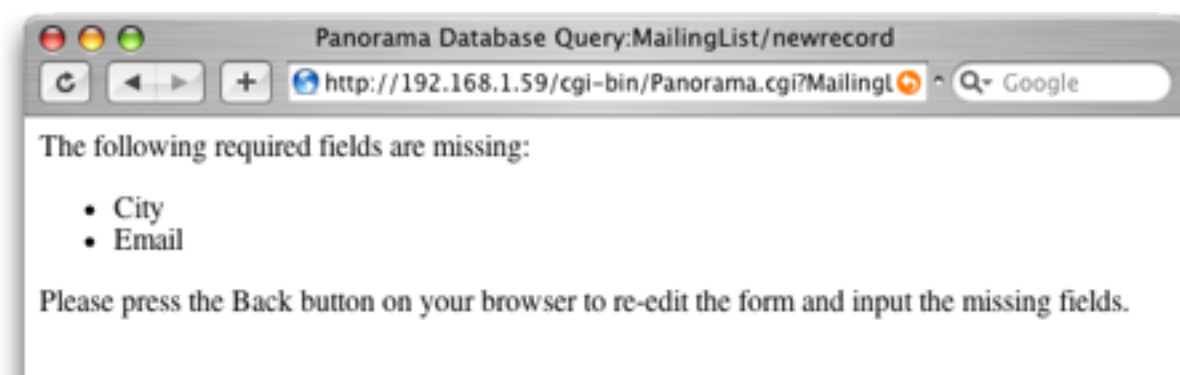


Note: If you are making this change to a database that has already been uploaded to the server then you'll need to re-render and upload the form (see “[Updating a Web Form/Adding a new Web Form](#)” on page 208).

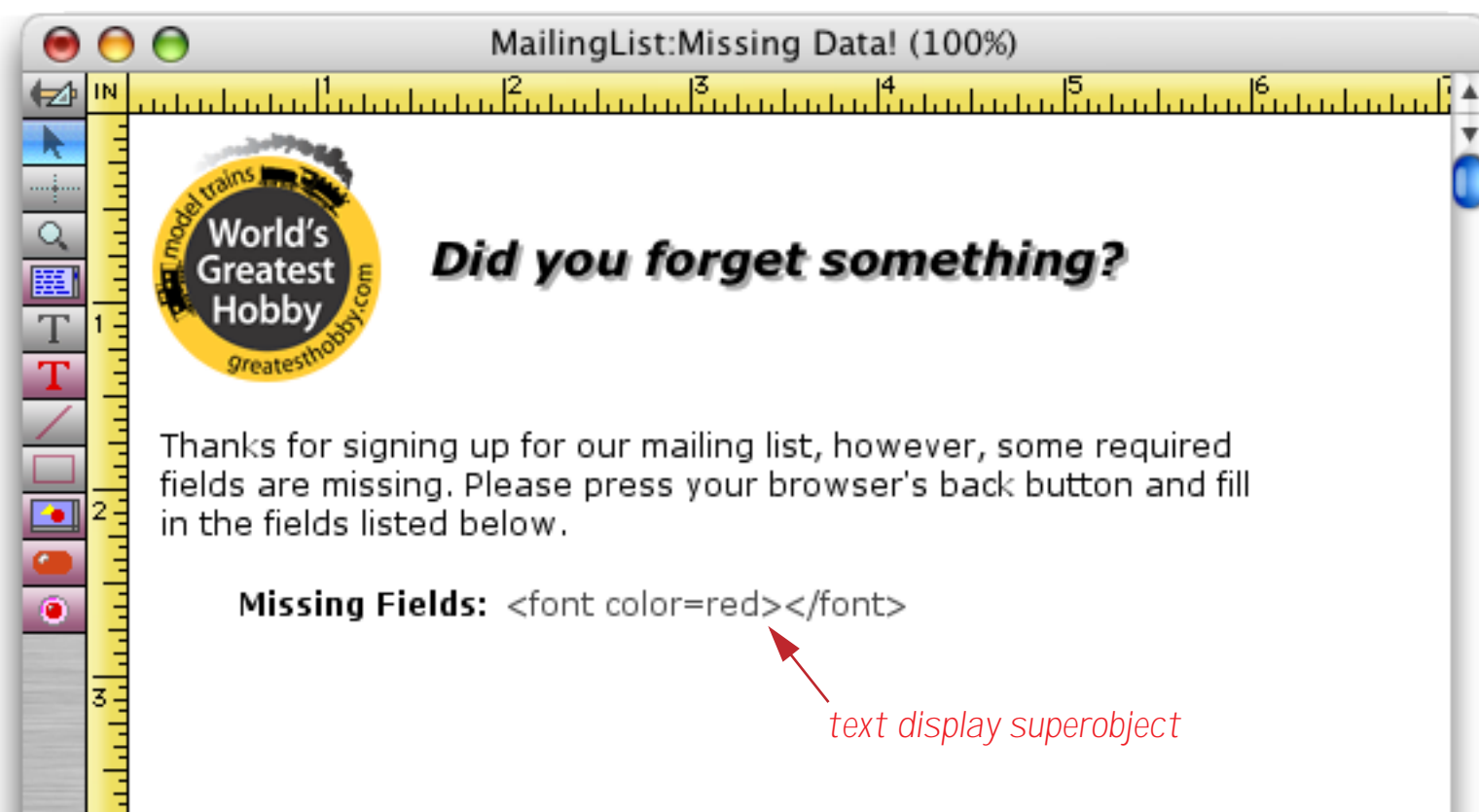
Checking for Required Fields/Preventing Missing Fields. When entering new records into the database you may want the server to make sure that some fields are filled in (not left empty). To enable this feature, use the **Required Fields** dialog in the **Web** submenu of the **Setup** menu (see “[Converting a Panorama Form into a Web Form](#)” on page 231). Simply check each field that is required, as shown here. (In addition to database fields this list also shows permanent variables.)



Before the new record is added to the database, the server checks to make sure that all these fields are present. (Note: The required field feature will not work with checkboxes or radio buttons on the form, only with Text Editor, Pop-up Menu and List objects.) If any are missing then a special “missing fields” page is displayed, like this:



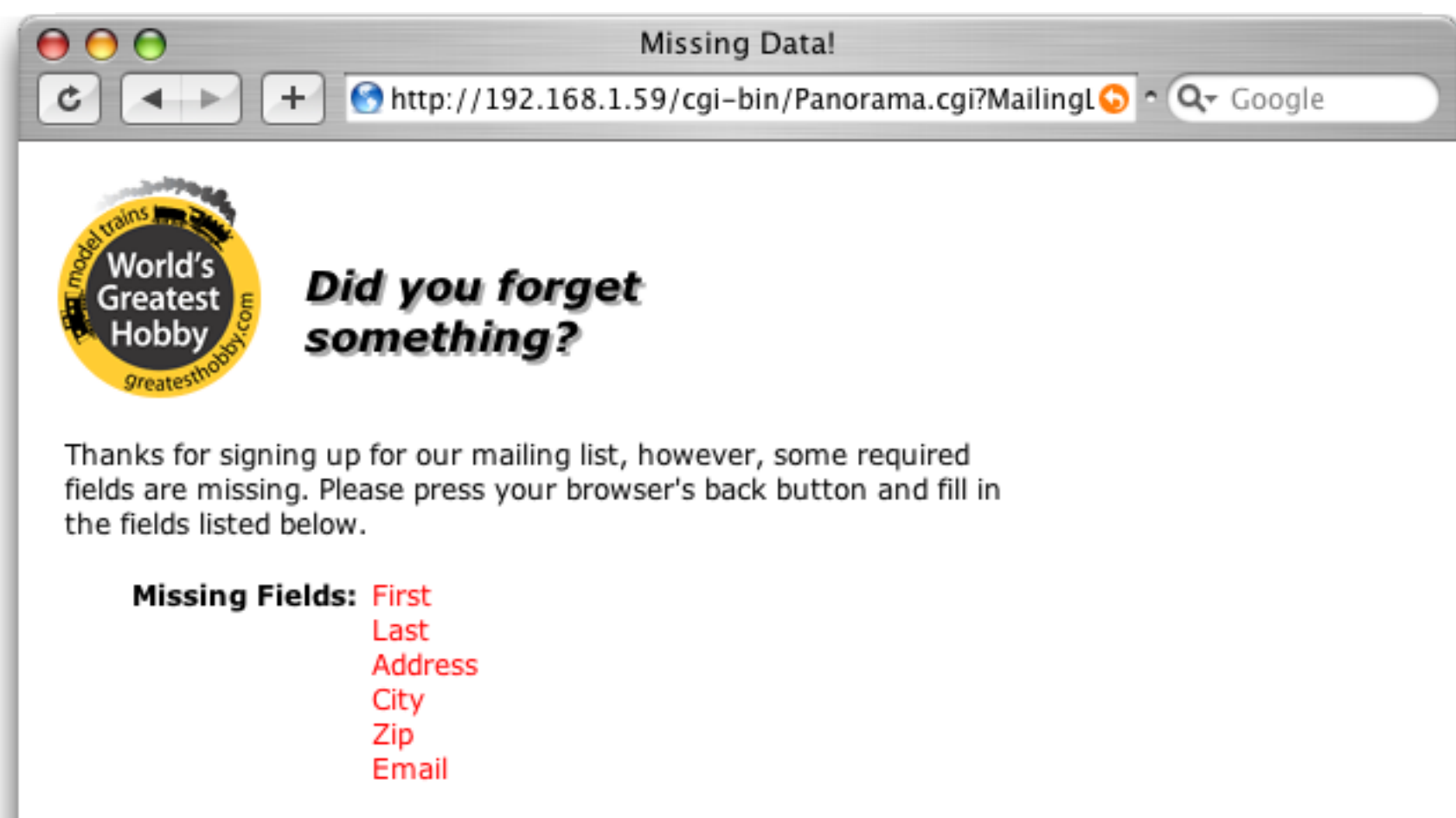
Is this page too plain for you? Then set up your own! Start by creating a Panorama form with the layout you want.



The Text Display SuperObject uses the `webformmissingfields()` function to list the fields that are missing.

```
"<font color=red>" + webformmissingfields("<field><br>") + "</font>"
```

This function has one parameter, a template that specifies how each field name should be listed. The field name itself is specified by the `<field>` tag, while other text, punctuation and tags can be added to control the formatting. In this example each field name is displayed on a separate line (because of the `
` tag). Here's how this page will actually look when there are missing fields (but first you must set up the **Form Action & Follow-Up Dialog**, described later in this section).



Here are several additional examples of how the text display formula can be adjusted to produce different formats for the missing fields.

```
"<font color=red>" + trim(webformmissingfields("<field>,"),1) + "</font>"
```

Missing Fields: First, Last, Address, City, Zip, Email

```
"<font color=red><ul>" + webformmissingfields("<li><field>") + "</ul></font>"
```

Missing Fields:

- First
- Last
- Address
- City
- Zip
- Email

```
"<font color=red><ol>" + webformmissingfields("<li><field>") + "</ol></font>"
```

Missing Fields:

1. First
2. Last
3. Address
4. City
5. Zip
6. Email

```
"<table border=0 cellspacing=2 width=150>" + webformmissingfields({<tr><td bgcolor="#CCCCCC" align=center><font size=-1 color=red><field></font></td></tr>} + "</table>"
```

Missing Fields:

| |
|---------|
| First |
| Last |
| Address |
| City |
| Zip |
| Email |

To tell the server to use your new “missing fields” form, go back to the original form (in this case the Signup form) and use the **Form Action & Follow Up** dialog in the **Web** submenu of the **Setup** menu (see “[Converting a Panorama Form into a Web Form](#)” on page 231). Select the form from the **Missing Fields** pop-up menu.

| Setting | Value |
|-------------------------|-----------------------|
| Submit Form Action: | ~newrecord |
| Followup (Normal): | Thanks for Signing Up |
| Record Not Found: | |
| Multiple Records Found: | |
| Missing Fields: | Missing Data! |

[click here for pop-up list of forms in this database](#)

Now if data is missing, the server will respond with your new custom “missing fields” page, as shown earlier in this section. Note: If you are making this change to a database that has already been uploaded to the server (see “[Uploading the Database to the Server](#)” on page 194) then you’ll need to re-render and upload the form (see “[Updating a Web Form/Adding a new Web Form](#)” on page 208).

Standard Form Action — QUERY

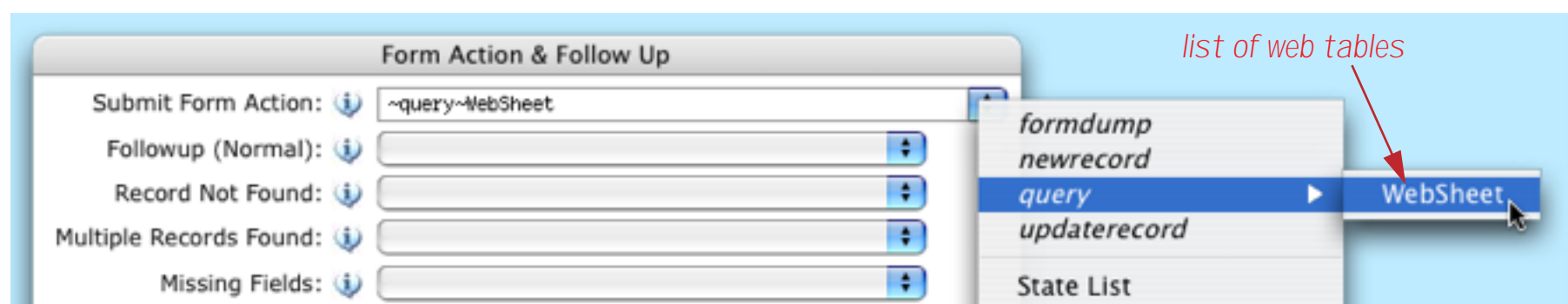
The **query** standard action selects one or more records in the database and displays a table showing the selected records. It's sort of the web equivalent of Panorama's **Find/Select** command.

The query process starts with a blank form which has one or more fields. When the form is submitted, the server scans the database to find all of the records that match all of the fields that have been entered (for example, if the last name and city have been entered then only records with the same last name *and* city will be listed).

An alternate query technique starts with a blank form with just one field, which must be set up to edit a variable named **SearchAllFields**. When the form is submitted, the server scans the database to find records with any field that contains the submitted text. We'll come back to this alternate method later.

Here's an example of a typical search form.

As you can see, this form is very similar to a data entry form. What makes it a search form is the fact that the action has been set to **query** instead of **newrecord** in the **Form Action & Follow Up** dialog in the **Web** submenu of the **Setup** menu (see “[Converting a Panorama Form into a Web Form](#)” on page 231).



When the query option is selected in the pop-up menu, an additional menu appears listing all of the web tables that have been set up for this database. (A web table is a template that controls the appearance of a table of records, see “[Web Tables](#)” on page 277.) In this example there is only one web table, **WebSheet**. Choose the web table you want to use with this query.

That's all there is to setting up a query! Make sure the form is rendered (converted to HTML) and uploaded to the server and you can use the query. For example, you could use this form to locate everyone named **Wilson** in **California**.

Advanced Search

World's Greatest Hobby

Enter the data you want to search for:

First Name:

Last Name:

Address:

City:

State:

Zip Code:

Phone:

Email:

Scale:

Era:

Railroad:

Notes:

Search Reset

Pressing the **Search** button shows that there are 4 Wilson's in this state.

Search Results

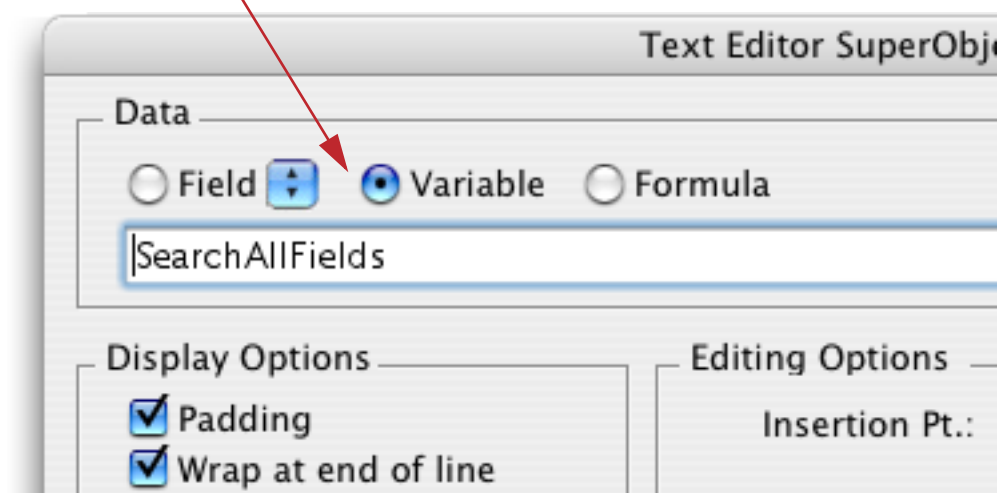
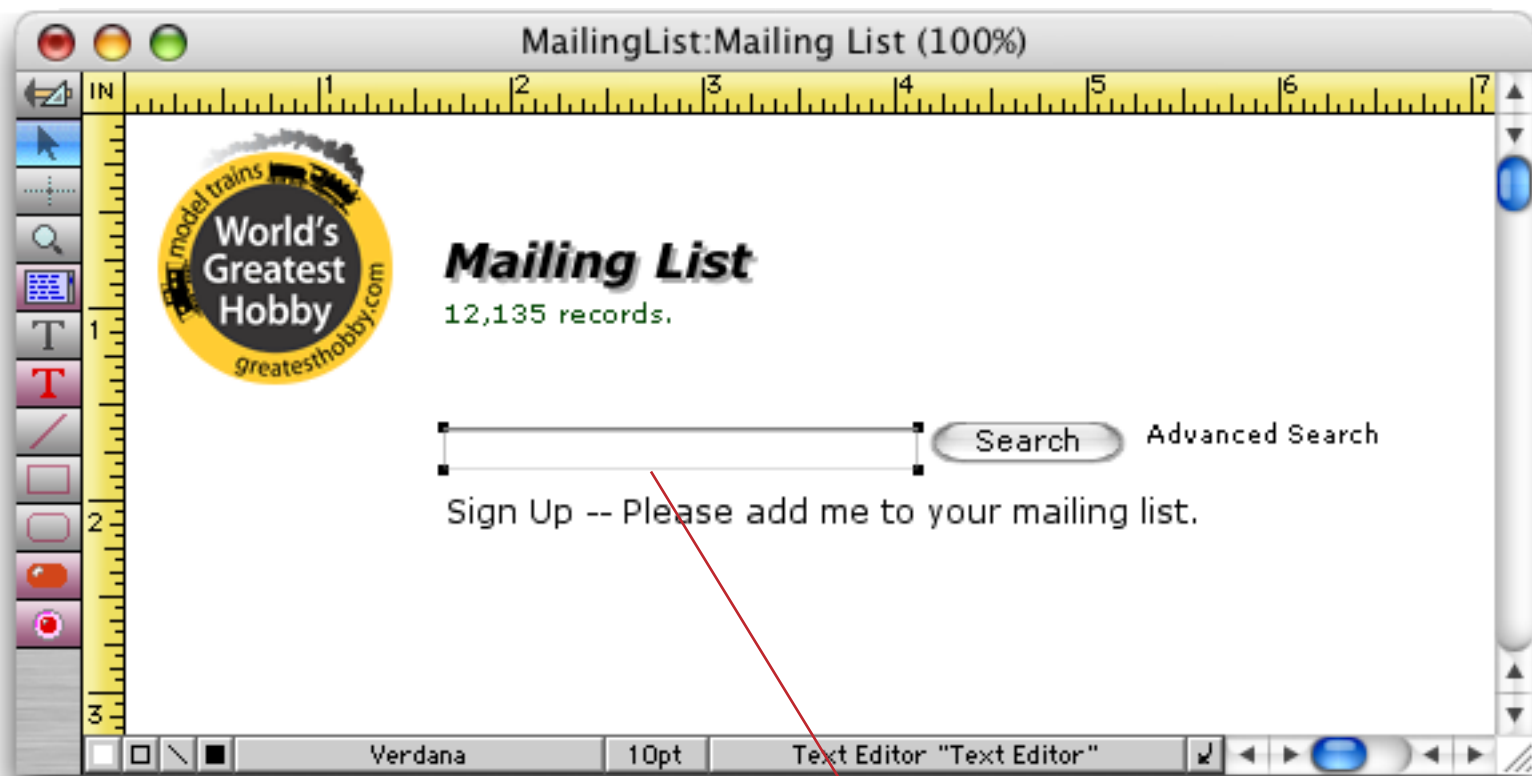
World's Greatest Hobby

Search Results...
4 of 12,147 records selected.

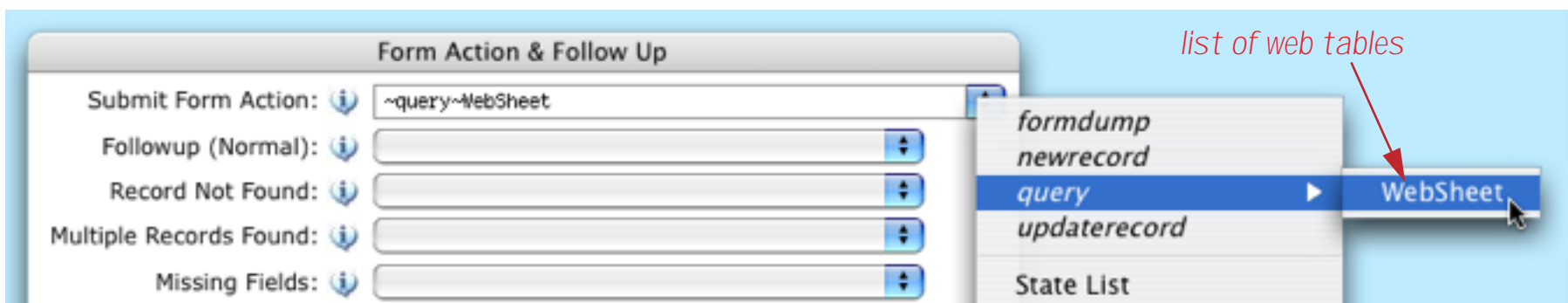
Search [Advanced Search](#)

| Name | City | Scale | Era | Railroad |
|--------------------------------|-------------------|-------|------------|------------------------|
| William Wilson | Campbell, CA | G | Transition | New York Central (NYC) |
| Scott Wilson | San Diego, CA | H0 | Modern | Southern |
| Mathew Wilson | San Francisco, CA | G | Transition | Santa Fe (BNSF) |
| John Wilson | Long Beach, CA | | Modern | Western Pacific (WP) |

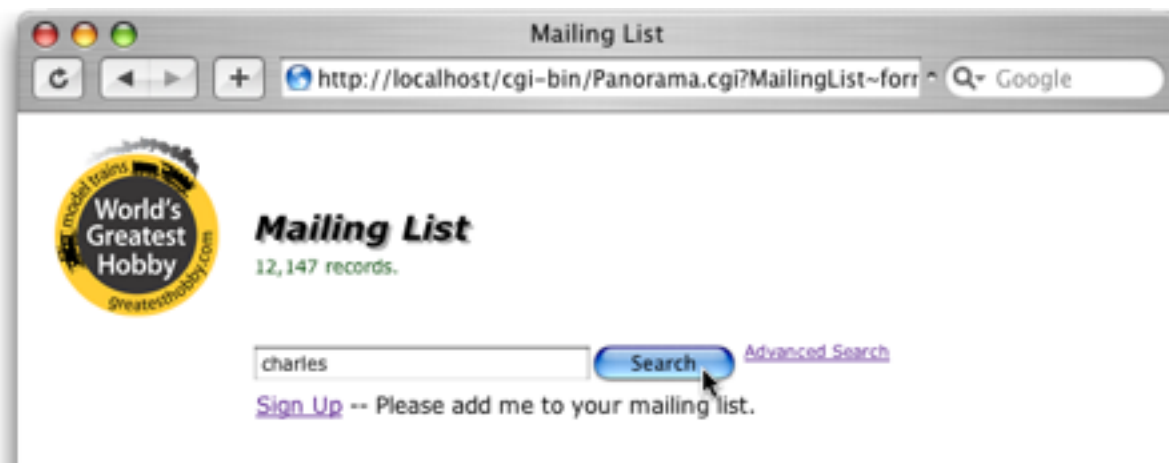
Searching All Fields. To search all fields simultaneously create a form with a single search field that edits a variable named `SearchAllFields`.



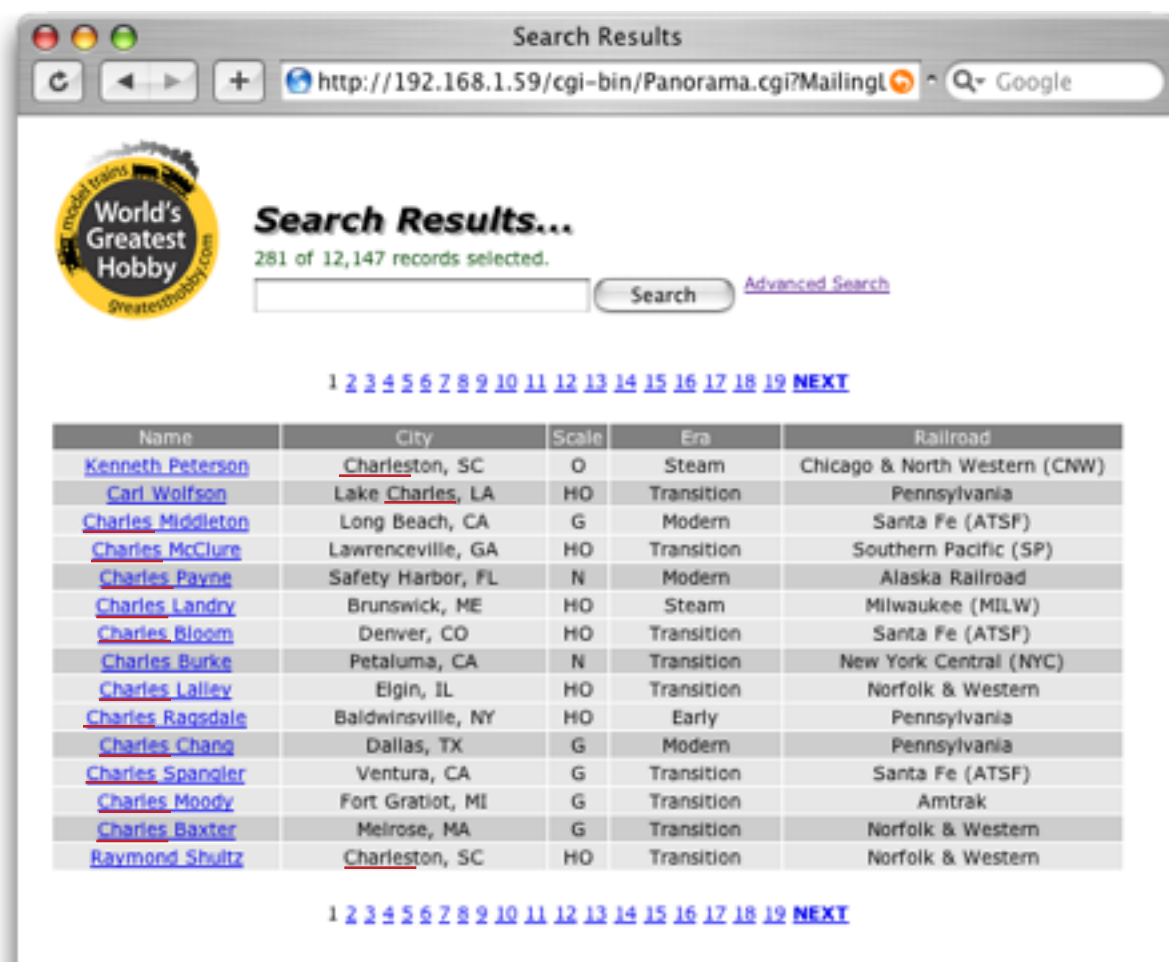
Just like a regular search form, set the form action to `query` and select the web table using the **Form Action & Follow Up** dialog in the **Web** submenu of the **Setup** menu (see "[Converting a Panorama Form into a Web Form](#)" on page 231).



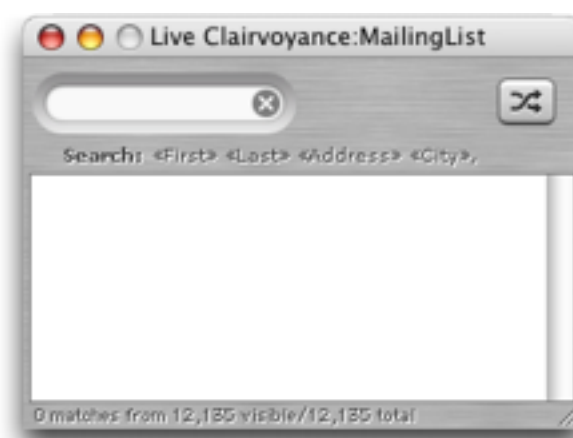
Make sure the form is rendered (converted to HTML) and uploaded to the server and you can use the query. For example you could use this query form to locate every occurrence of **charles** in the database.



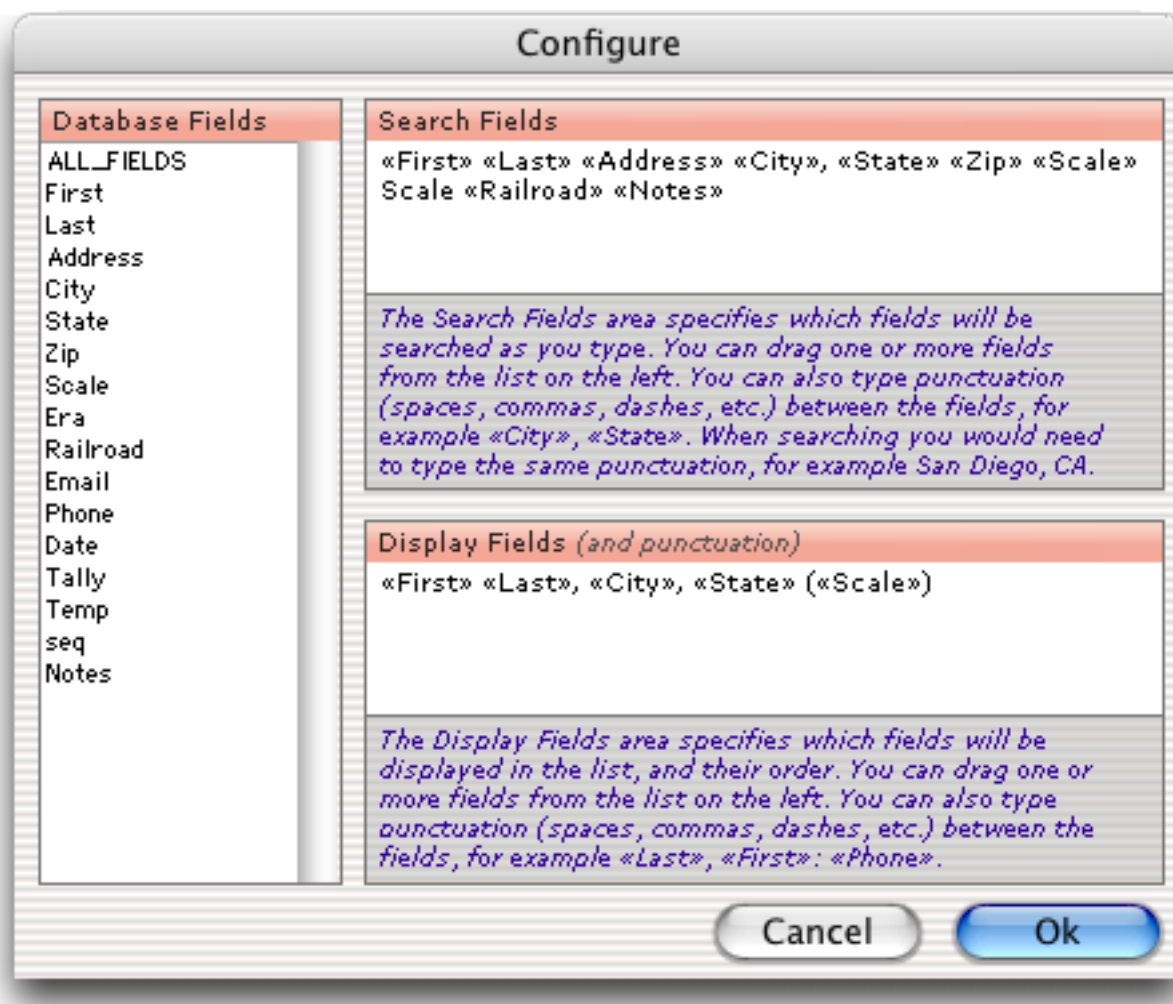
This search will find **charles** no matter what field it is in.



Searching Multiple Fields instead of All Fields. Using the Live Clairvoyance wizard you can customize the all field search to include only a subset of the fields in the database. First make sure the database you want to modify is active, then open the Live Clairvoyance wizard.



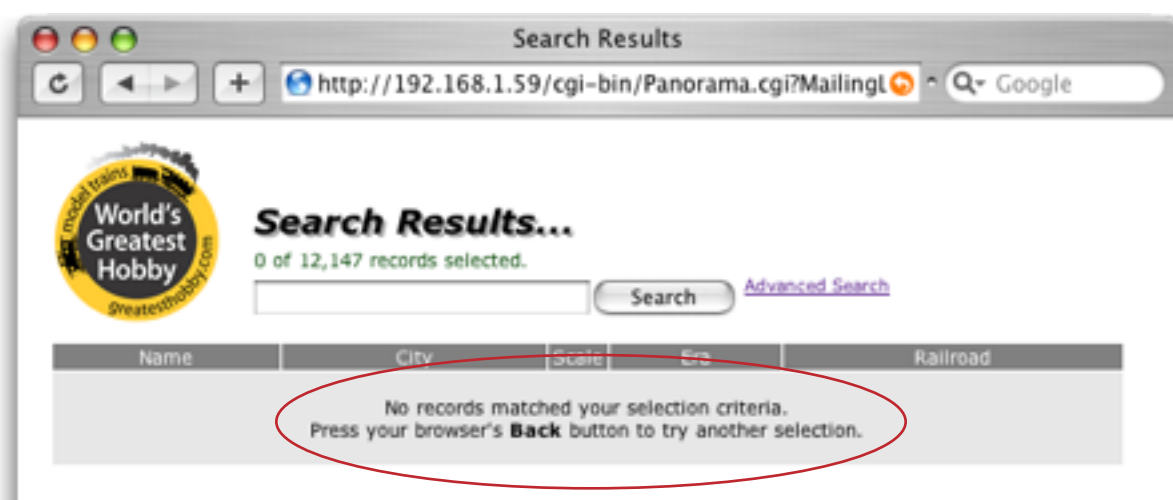
Now press the  button to configure Live Clairvoyance. The configuration dialog will appear:



(Note: You may wonder why the search fields section contains «Scale» **Scale**. This allows searches to be made for **HO Scale**, **N Scale**, etc., which would be a common way for model railroaders to specify a search.)

Follow the on-screen instructions to set up the fields to be searched, then press the **Ok** button to close the dialog. Close the wizard and save the database, then if necessary re-upload the form to the server. Instead of searching all fields, the search will use the configuration you have just set up.

Handling Failed Searches. What if a search query turns up no results? For example, suppose you search for **zumi**. There is no **zumi** in the database, so the server displays an empty table.



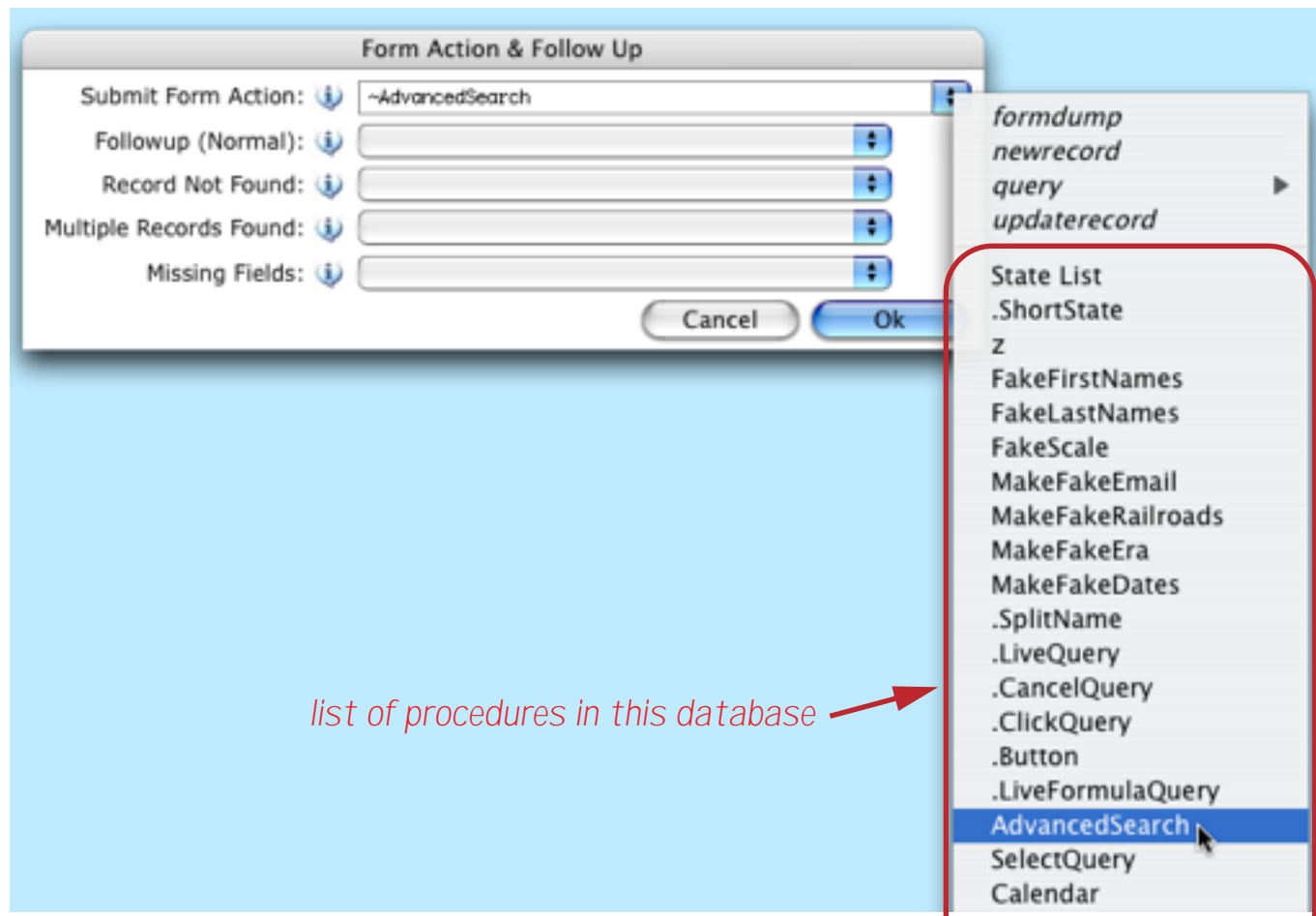
The server's response to a failed search can be modified with a custom program, see "[Displaying an Empty Table](#)" on page 426.

Standard Form Action — UPDATERECORD

The **update** standard action is similar to **newrecord** but instead of adding a new record it modifies an existing record. This action cannot be used by itself but must be part of a sequence that displays the form from the database. See "[Editing/Updating a Record](#)" on page 313 for more details on this process.

Custom Form Actions

Any action other than the four standard form actions discussed above (formdump, newrecord, query and update) is a custom form action. Custom actions are created by writing a procedure to process the form input (see “[Web Programming 101](#)” on page 317). To specify that a procedure should be triggered when the form is submitted, use the **Form Action & Follow Up** dialog in the **Web** submenu of the **Setup** menu (see “[Converting a Panorama Form into a Web Form](#)” on page 231).



The first option in this dialog is the **Submit Form Action**. You can type in the name of the procedure preceded by the ~ character, but usually you’ll select the procedure you want from the pop-up menu on the right. This menu lists the 4 standard built in actions along with all of the procedures in this database. Whatever you select (standard action or procedure) will be triggered when the form is submitted to process the data and display the next page.

Advanced Form Techniques

This section describes advanced techniques for customizing forms, including techniques for embedding HTML directly into your form.

Font selection

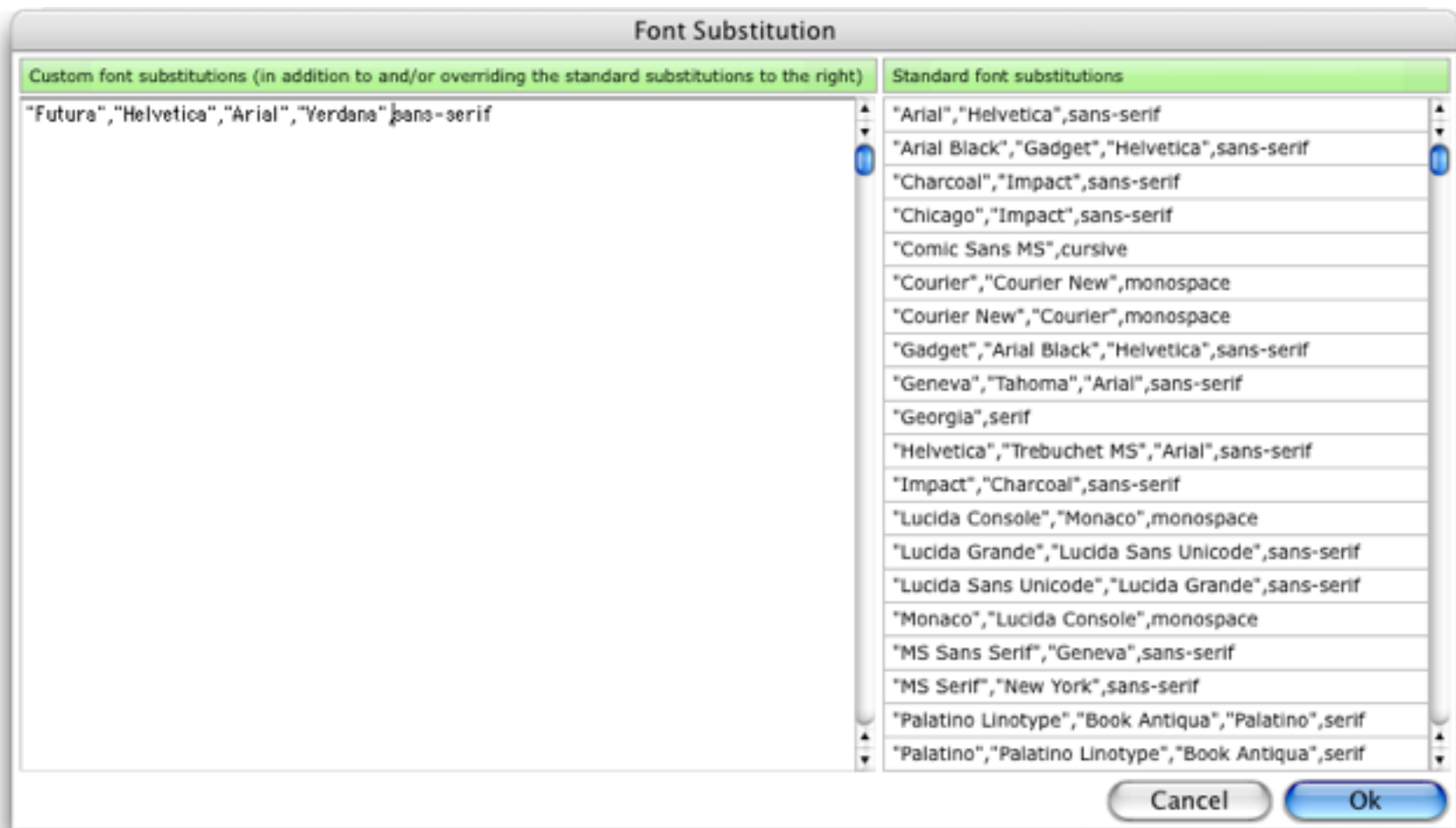
On your own computer you can use any font you want because you have complete control over what fonts are installed and available for use. A web page, however, is the opposite situation since you have zero control over what fonts may be available when the page is viewed. Because of this it is usually best to pick from a restricted subset of fonts that are almost universally available. The fonts that are most commonly available are: *Arial*, *Comic Sans MS*, *Courier*, *Georgia*, *Helvetica*, *Times* and *Verdana*.

You may want to use fonts beyond this very restricted list. If you use one of the primary fonts listed below, the web browser will automatically substitute the first alternate font if the primary font is not available. If the first alternate is not available, the second alternate will be used. If even that is not available, the web browser will select a generic font based on the final alternate column.

| Primary Font | Alternate 1 | Alternate 2 | Final Alternate |
|---------------------|---------------------|--------------|-----------------|
| Arial | Helvetica | | sans-serif |
| Arial Black | Gadget | Helvetica | sans-serif |
| Charcoal | Impact | | sans-serif |
| Chicago | Impact | | sans-serif |
| Comic Sans MS | | | cursive |
| Courier | Courier New | | monospace |
| Courier New | Courier | | monospace |
| Gadget | Arial Black | Helvetica | sans-serif |
| Geneva | Tahoma | Arial | sans-serif |
| Georgia | | | serif |
| Helvetica | Trebuchet MS | Arial | sans-serif |
| Impact | Charcoal | | sans-serif |
| Lucida Console | Monaco | | monospace |
| Lucida Grande | Lucida Sans Unicode | | sans-serif |
| Lucida Sans Unicode | Lucida Grande | | sans-serif |
| Monaco | Lucida Console | | monospace |
| MS Sans Serif | Geneva | | sans-serif |
| MS Serif | New York | | sans-serif |
| Palatino Linotype | Book Antiqua | Palatino | serif |
| Palatino | Palatino Linotype | Book Antiqua | serif |
| Tahoma | Geneva | Arial | sans-serif |
| Times | Times New Roman | | serif |
| Times New Roman | Times | | serif |
| Trebuchet MS | Helvetica | | sans-serif |
| Symbol | | | symbol |
| Verdana | Geneva | | sans-serif |
| Webdings | | | fantasy |

For example, suppose that you have specified that a text object be displayed using the *Verdana* font. If *Verdana* is not available, then *Geneva* will be used. If *Geneva* is not available then a generic sans-serif font will be used. (Of course this means that your form should not rely on the dimensions of a certain font for exact form layout.)

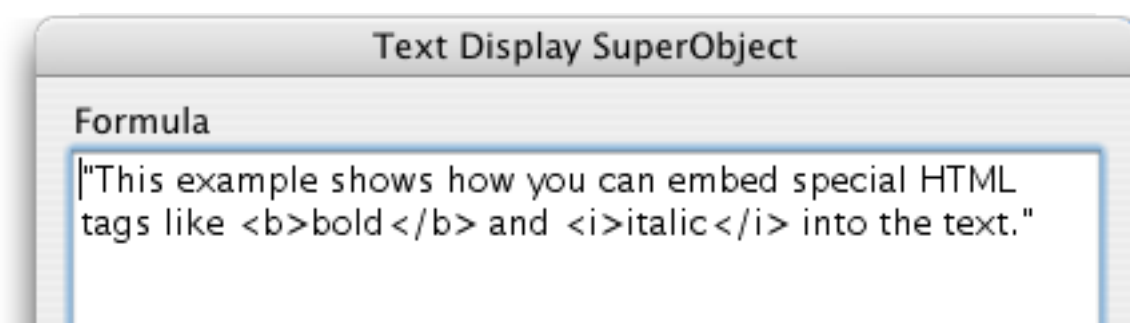
If you use a font that is not on the list above, then all bets are off. If that font is not available when the web page is displayed, the results are unpredictable. You can, however, use the **Font Substitution** dialog (in the **Web** submenu of the **Setup** menu, see “[Converting a Panorama Form into a Web Form](#)” on page 231) to set up custom substitutions. The example below shows how to set up substitutions for the *Futura* font.



This example sets up four alternates for the *Futura* font. Notice that when actual font names are specified they should be specified in quotes, while generic font names (*serif*, *sans-serif*, *monospace*, *cursive* or *fantasy*) should not be enclosed in quotes.

Embedding HTML in a Text Display SuperObject

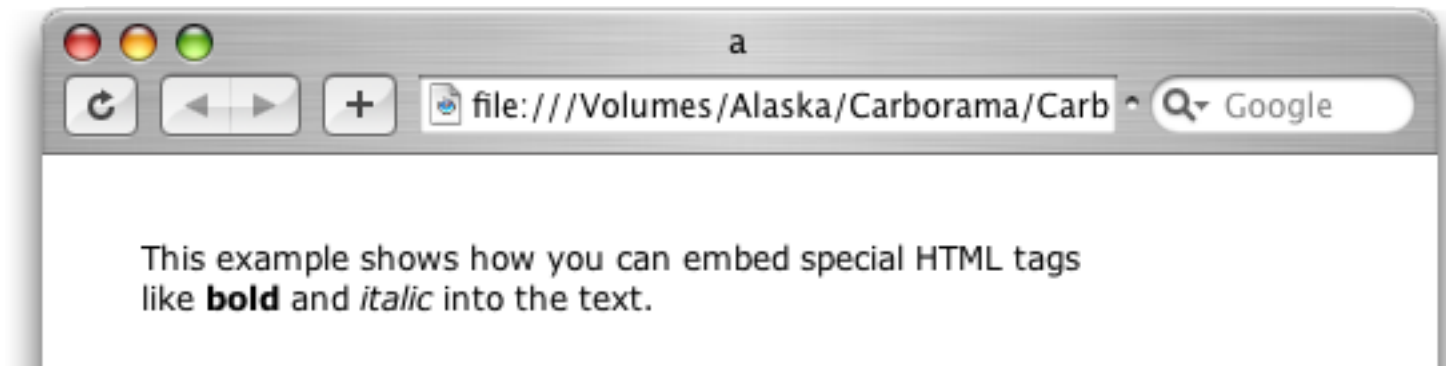
If you want to display text in a Text Display SuperObject, Panorama normally takes care of all the details for you. If you are familiar with HTML you know that special characters like `<`, `>`, and `&` normally require special processing, but Panorama takes care of that for you. There is one exception to this rule, however. If the text you are displaying contains HTML tags, then Panorama does not special processing. This allows you to place custom HTML tags (bold, italic, links, etc.) into the final page. For example, here is a formula that will display some text in bold and italic.



In the Panorama form you see the actual tags.



Here's what this form looks like when it is converted to HTML:



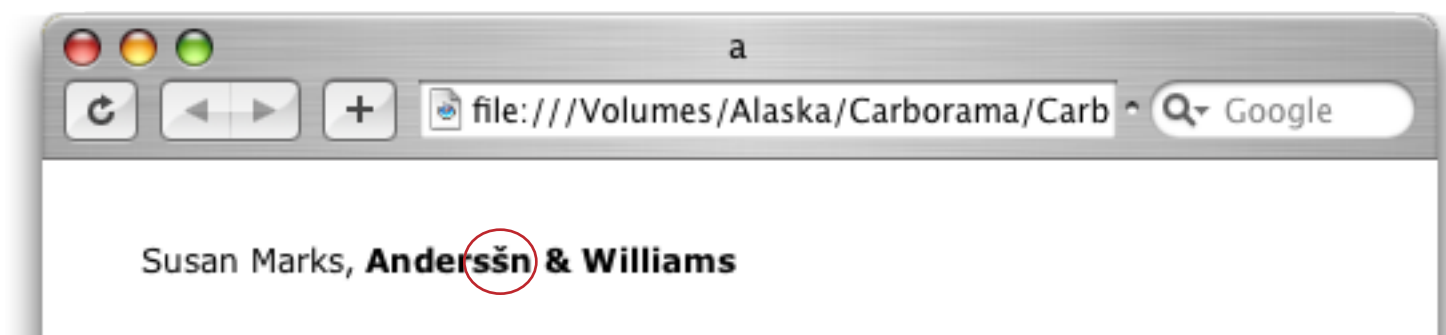
If you embed HTML tags into the Text Display formula you must take care to make sure that any data included in the formula is converted to web format. This can be done with the `webtext()` function. Suppose, for example, you want to display a person's name and company, with the company name in bold. You might think to use a formula like this:



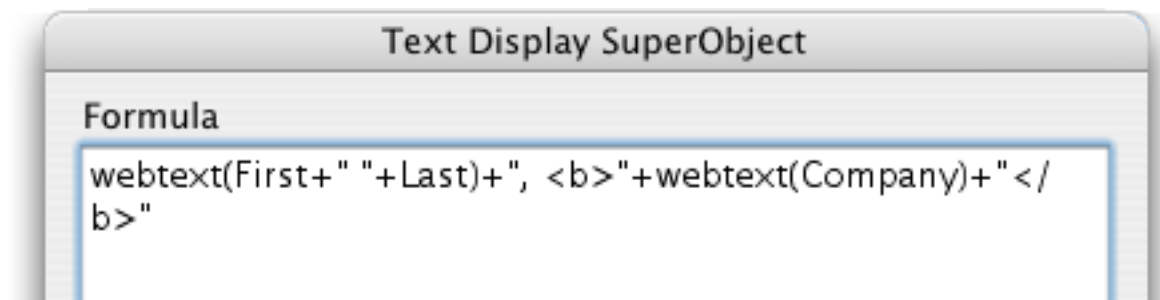
But what if the data contains a special character, like this:



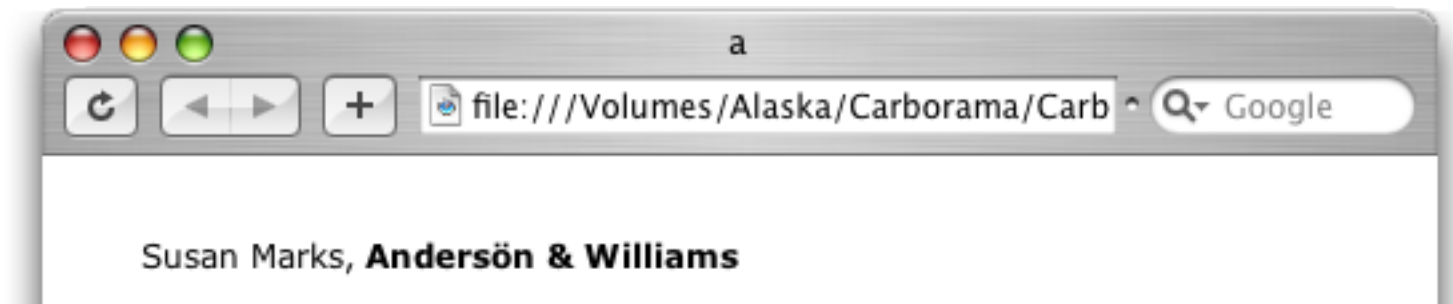
This special character will not display properly on the browser without conversion:



Adding the `webtext()` function around each field fixes the problem.



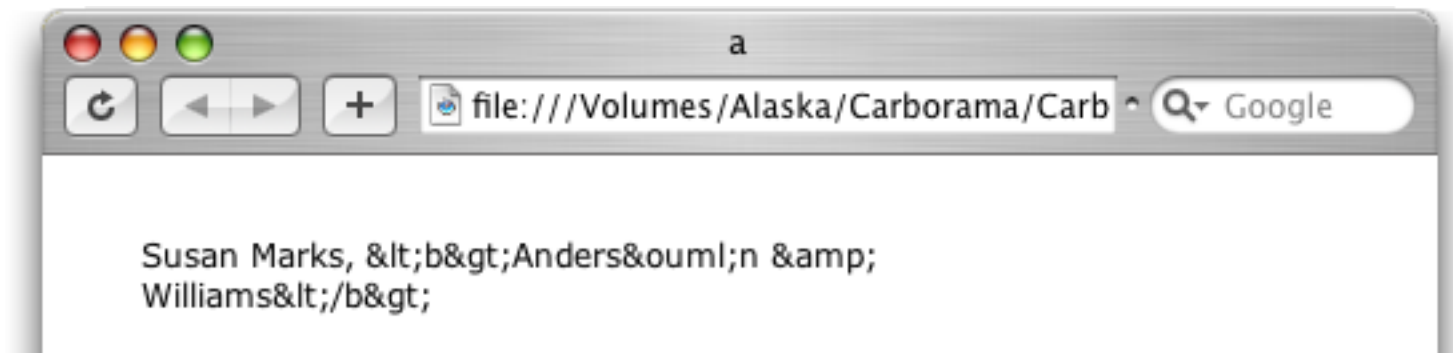
Here's the finished result, with the correct accent character:



A common mistake is to enclose the entire formula with the `webtext()` function, like this:



As you can see below, this results in a mess. Don't include any HTML tags within the `webtext()` function, only data (or text that does not include tags).



Remember, the `webtext()` function is only necessary if your formula contains HTML tags. If the result of your formula doesn't contain any tags then the `webtext()` function should not be used. (Note: Panorama assumes the text contains HTML tags if the first character is `<` and the last character is `>`, or if the text contains `</` anywhere within the text.)

Linking to Other Web Pages

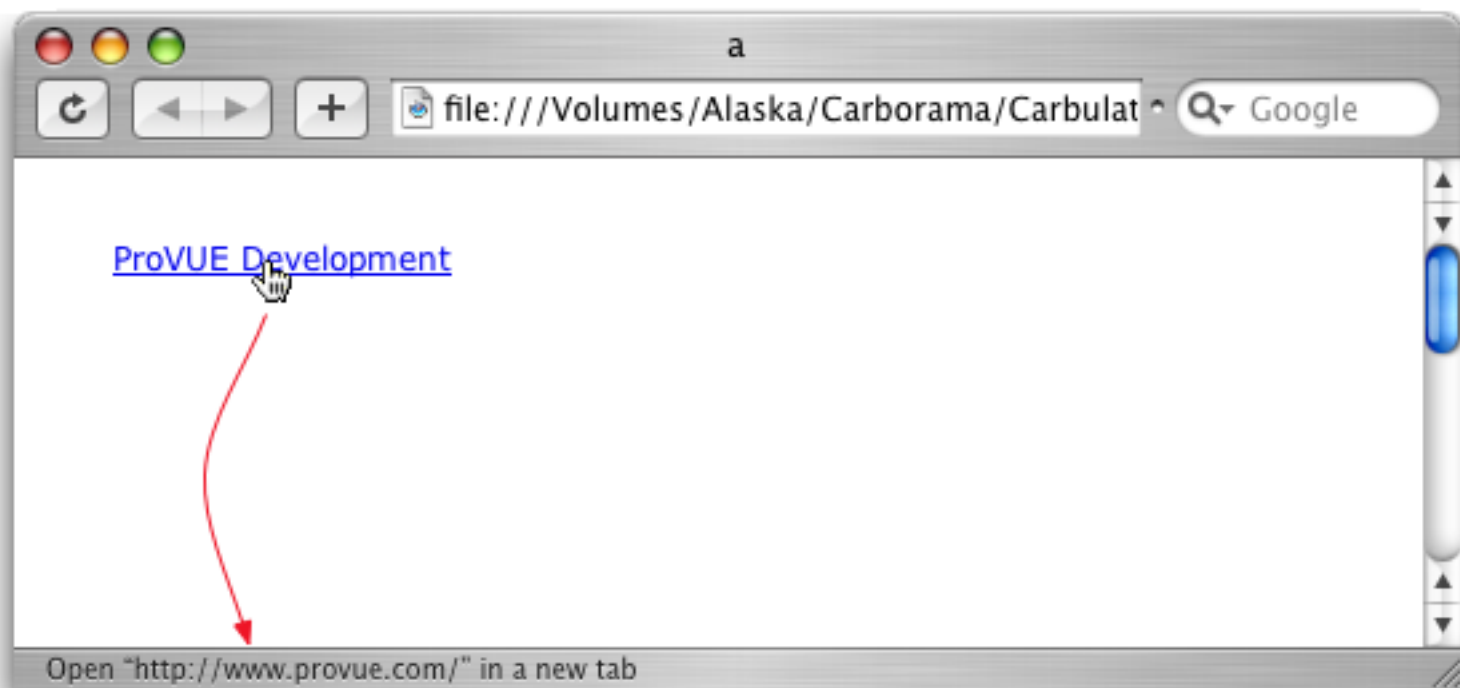
To link to another web page, use the `weblink()` function. This function has two parameters, the URL to link to and the caption for the link. Here's an example that will create a link to the ProVUE Development home page.



In the Panorama form only the caption will appear.



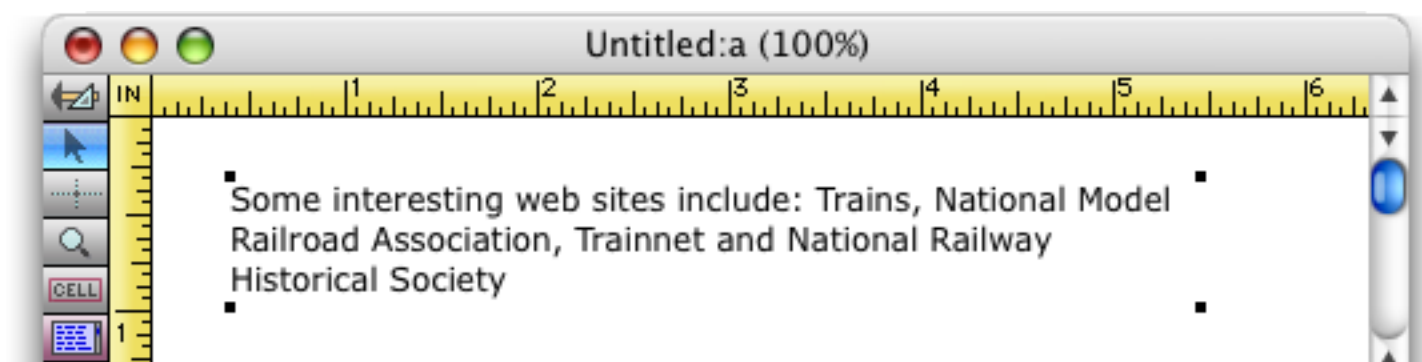
When the form is converted to a web page, the text becomes a link to the ProVUE Development home page. Clicking on the link will jump to that page.



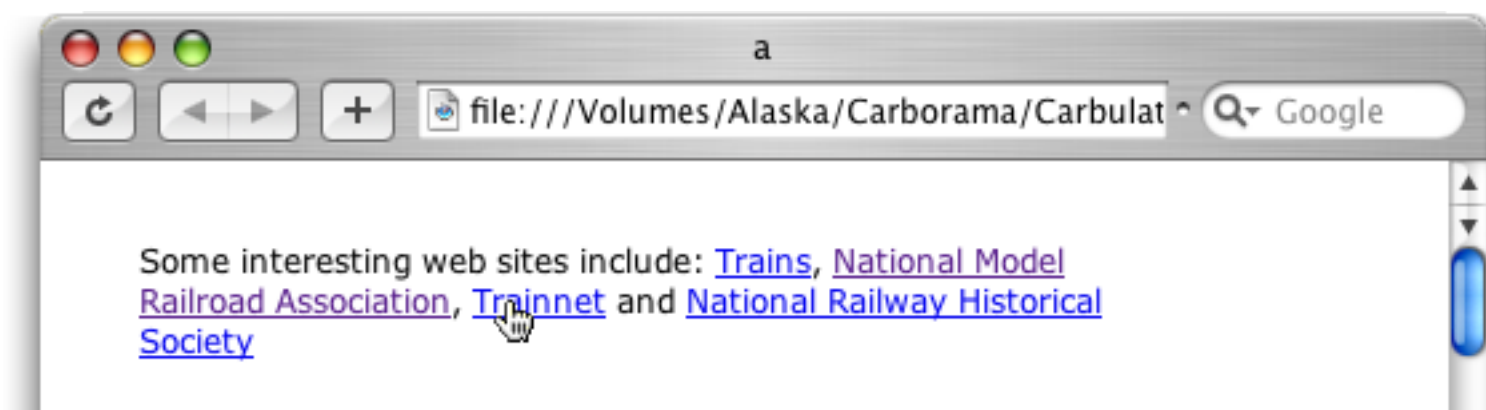
You can combine a link with other text, or even put multiple links in a single Text Display object.



The Panorama form will look like this:



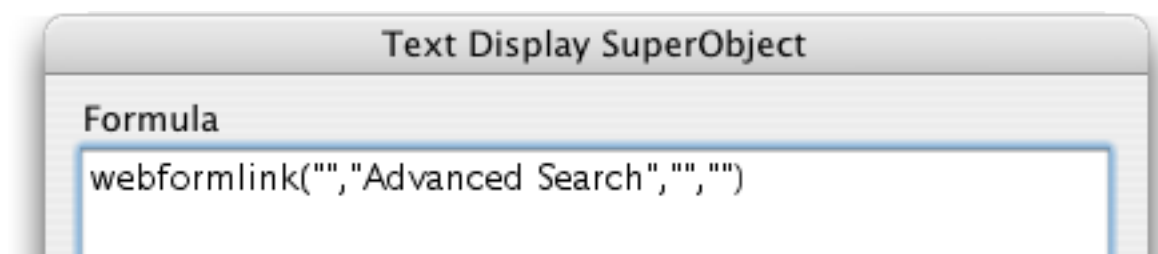
and the web page like this:



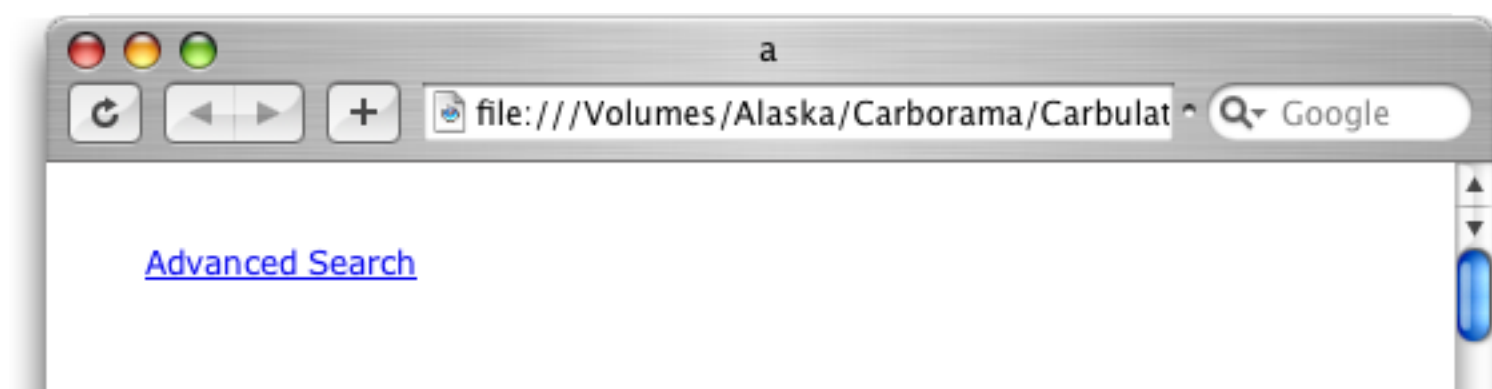
The `weblinknewwindow()` function works exactly the same except that the new web page will automatically be opened in a new window (instead of in the same window).

Linking to Blank Panorama Forms (from a Panorama Form)

To link to a blank Panorama form (for example to add a new record or start a search) use the `webformlink()` function. For example, suppose you have created a form called `Advanced Search` in the same database. This formula will link to the search page:



In the Panorama form, this object will simply say `Advanced Search`. When rendered to HTML this becomes a link.



The `webformlink()` function actually has four parameters (though as we have seen, often only one is needed).

```
webformlink(database, form, record, caption)
```

The **database** parameter specifies the database that contains the requested form. This should be the name of the database on the server. If the form is in the same database as the current form, the database parameter can be left blank.

The **database** parameter specifies the form being linked to.

The **record** parameter must be left blank to create a blank form. (If non-blank it specifies a record to be used to populate the form, see “[What Record Are We Talking About?](#)” on page 395.)

The **caption** parameter specifies the text that will be displayed for this link (the text that the user sees). If left blank the form name will automatically be used as the caption.

Like the **weblink()** function, the **webformlink()** function can be used as part of a more complex formula that may include multiple links, tags, etc. There is also a **webformlink()** function that causes the blank form to open in a new browser window.

Linking to Blank Panorama Forms (from a standard web page)

The illustration below shows how to build a URL to a Panorama form from a standard web page (a web page built with a text editor or web design program like Dreamweaver). The black parts of the URL never change, while the blue sections must be adjusted for your site, database, and form.

http://www.mysite.com/cgi-bin/Panorama.cgi?MyDatabase~Form~MyForm

The database name and form name must be exact, including upper and lower case. If the database name or form name contain any special characters they must be converted to % notation, for example a space becomes %20. The example below shows how to build a link to an Advanced Search form in the Registration database.

```
<a href="http://www.acme.com/cgi-bin/Panorama.cgi?Registration~Form~Advanced%Search">
Advanced Search</a>
```

Linking to Panorama Procedures (from a Panorama Form)

In a later chapter you'll learn how to write a Panorama procedure that creates a web page (see “[Generating HTML](#)” on page 369). To link to such a procedure use the **cgilink()** function. The **cgilink()** function has three parameters:

```
cgilink(database,urlsuffix,caption)
```

The **database** parameter specifies the database that contains the requested procedure. This should be the name of the database on the server. If the procedure is in the same database as the current form the database parameter can be left blank.

The **urlsuffix** parameter specifies the form being linked to, along with any additional parameters. If there are any parameters they must be separated from the procedure name with the ~ character.

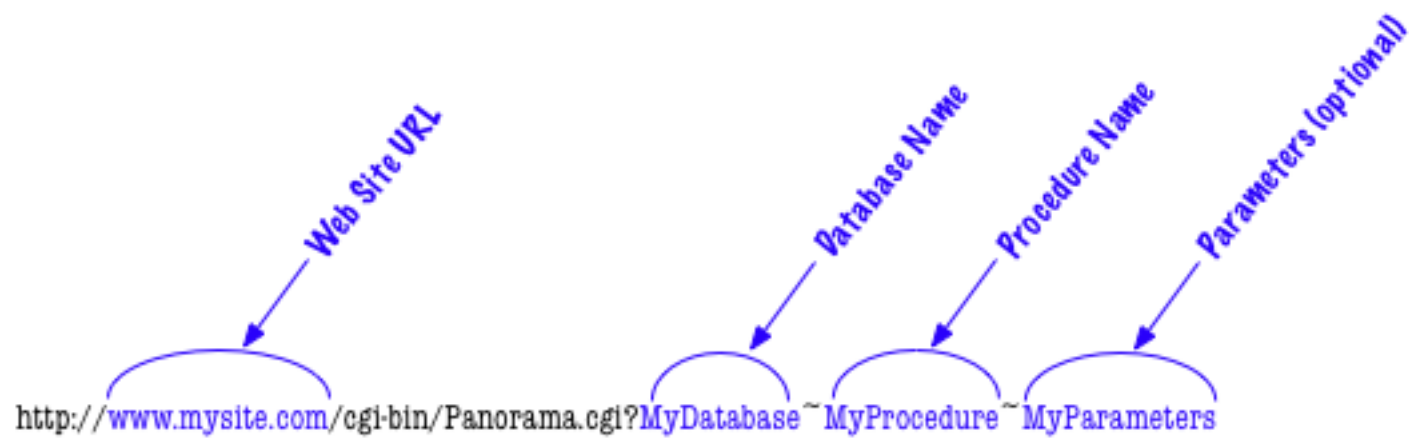
The **caption** parameter specifies the text that will be displayed for this link (the text that the user sees).

Here is an example that will display statistics from 2006 (assuming, of course, that the Statistics procedure has been written to process an extra parameter this way.

| Formula |
|--|
| cgilink("", "Statistics~2006", "Stats 2006") |

Linking to Panorama Procedures (from a standard web page)

The illustration below shows how to build a URL to a Panorama form from a standard web page (a web page built with a text editor or web design program like Dreamweaver). The black parts of the URL never change, while the blue sections must be adjusted for your site, database, and procedure.

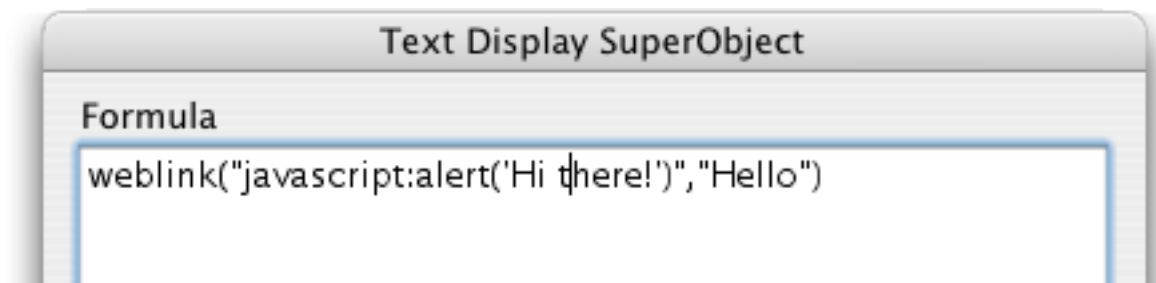


The database name and procedure name must be exact, including upper and lower case. If the database name or procedure name contain any special characters they must be converted to % notation, for example a space becomes %20. The example below shows how to build a link to a statistics procedure in the Registration database.

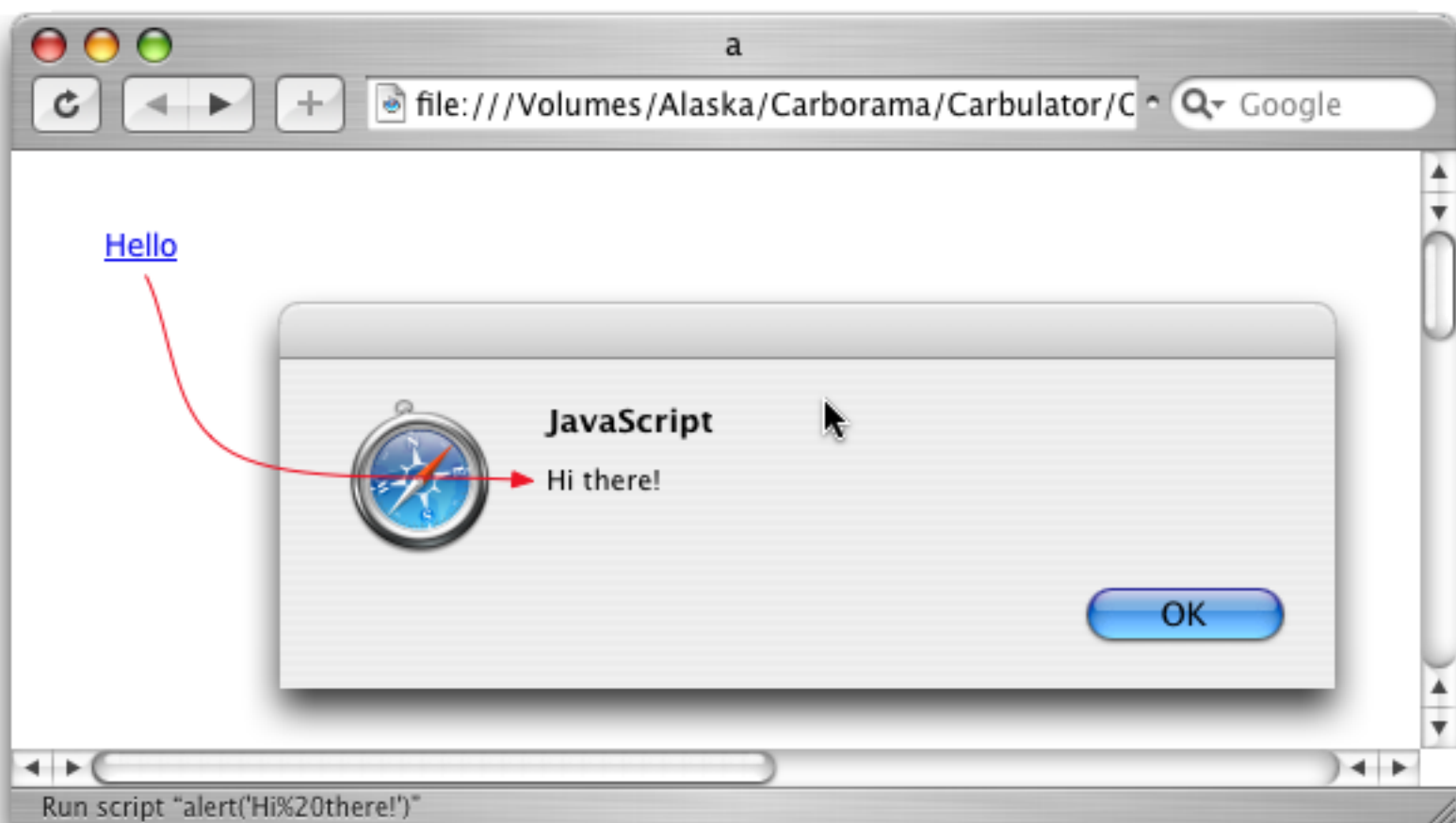
```
<a href="http://www.acme.com/cgi-bin/Panorama.cgi?Registration~Statistics~2006">Stats</a>
```


Linking to a JavaScript Script

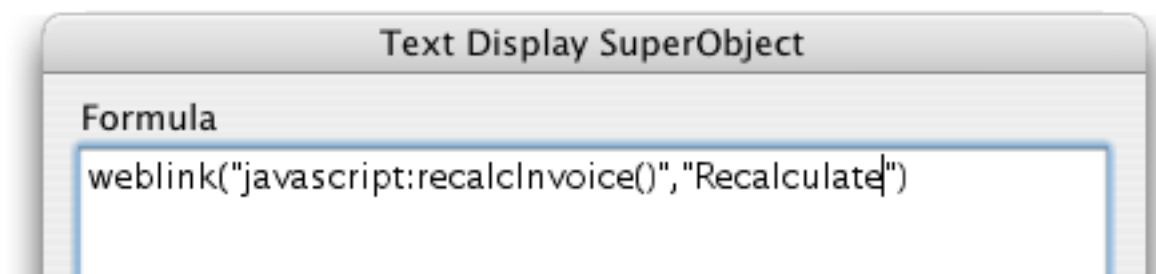
The `weblink()` function can also be used to trigger a JavaScript script. Simply start the URL with `javascript:` instead of `http:`, as shown below. The script itself must not contain any `"` characters, and should not return a value.



In the web browser clicking on the word [Hello](#) will display a short message.



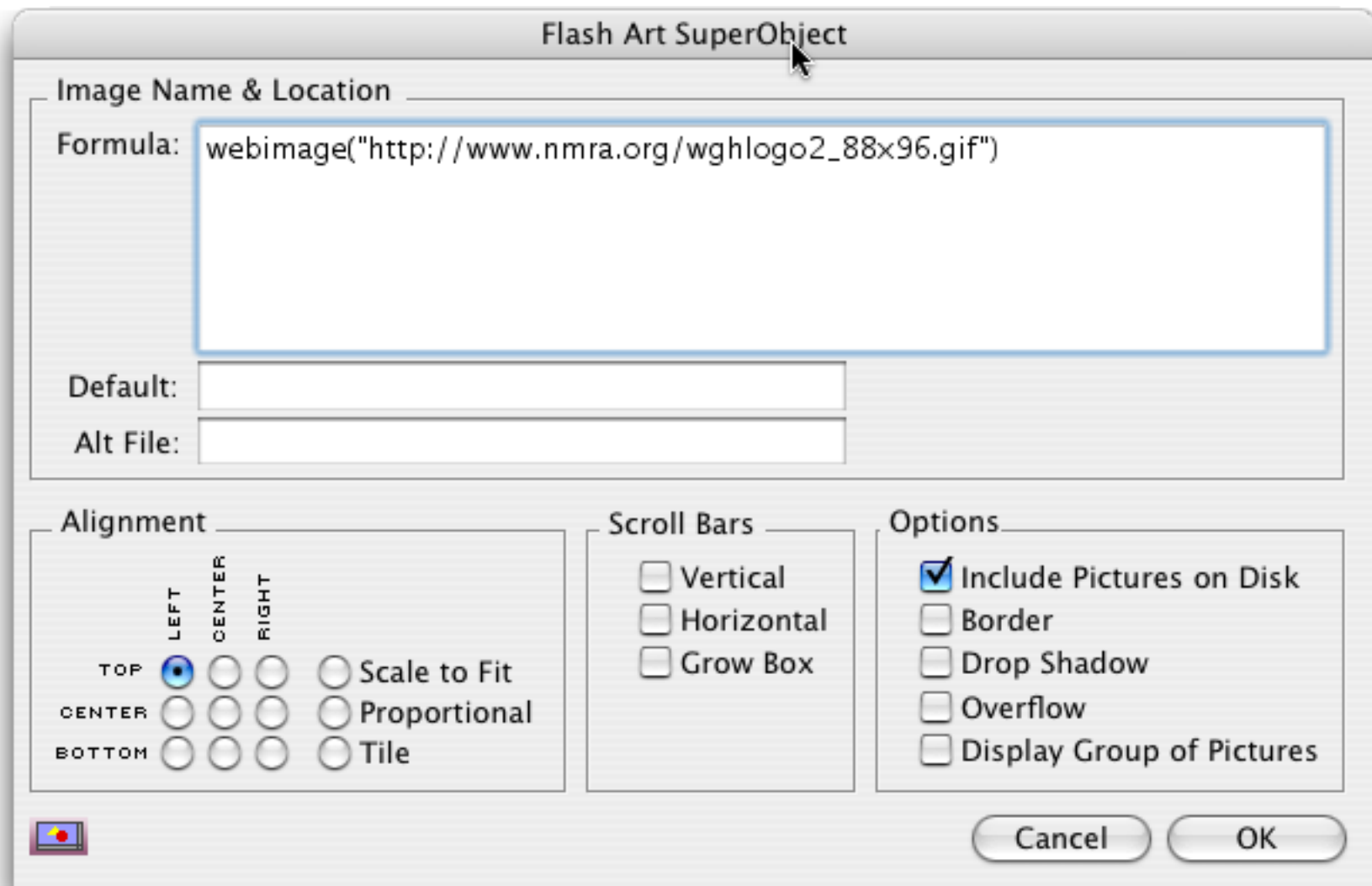
If the script you want to run has more than a couple of statements in it, it's usually best to define a custom JavaScript function separately (see "[Adding JavaScript to a Page](#)" on page 275) and simply trigger the function with the `weblink()`.



Displaying Images in a Web Form

Many forms include images. Images may be fixed, like a logo, or variable, like a catalog photo that changes for each record. In a regular Panorama form, images can be displayed by pasting them directly into the form, they can be displayed from the Flash Art catalog or from a file on disk, or they can be displayed directly from the web. When a form is exported for the web, however, only the last option is available. If a form contains images that are pasted in or stored locally these images must be converted into web images before they can be displayed on a web form.

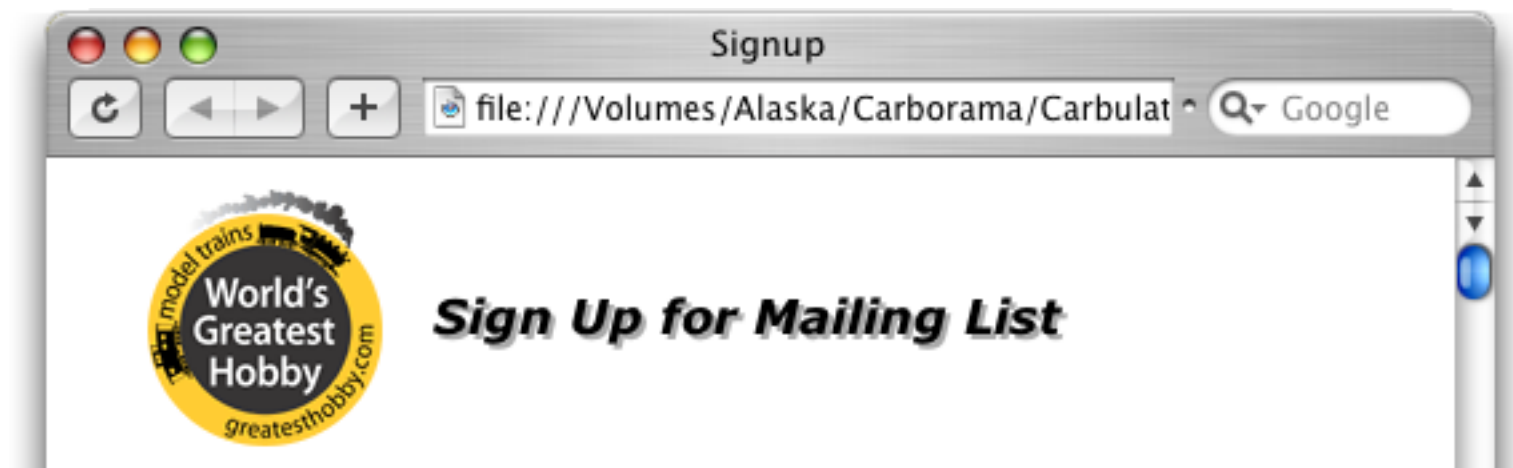
To display an image from the web create a Flash Art SuperObject (see Chapter 16 of the Panorama Handbook) and use the `webimage()` function to specify the URL of the image you want to display (usually a GIF or JPEG image).



Be sure that the **Include Pictures on Disk** and **Top/Left** options are checked. None of the other options will work when the page is converted to HTML. When the Panorama form is displayed, the image will be fetched over the internet and displayed (assuming that the computer is connected to the internet!). There may be a delay the first time the image is displayed.

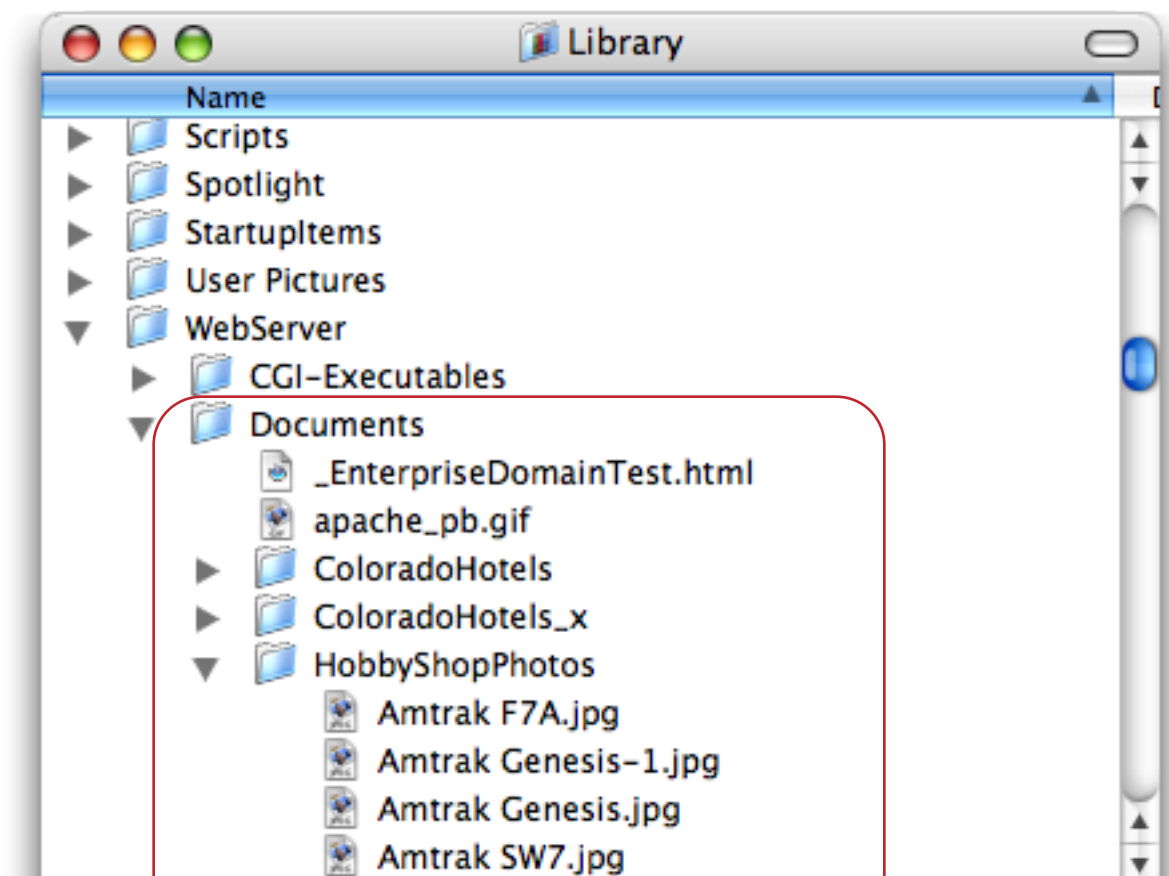


When the form is converted to HTML, the image will be displayed just as it was on the Panorama form:

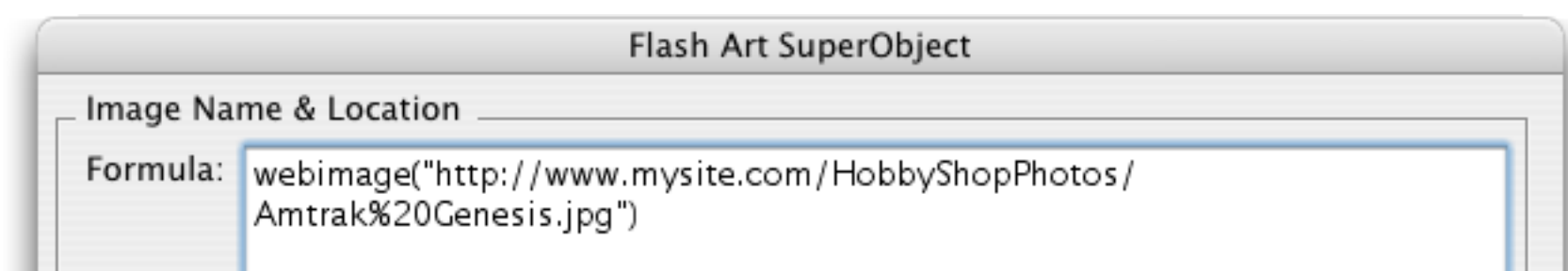


There's one problem with this example — it's displaying an image from somebody else's web site. In many cases this may not be appreciated. First of all, you are poaching someone else's bandwidth. Secondly, you may be violating the copyright of the person or organization that created the images. (However, it may be ok to directly access photo sharing sites like flickr if you are accessing your own images or images that are in the public domain — check the terms of service for the site in question).

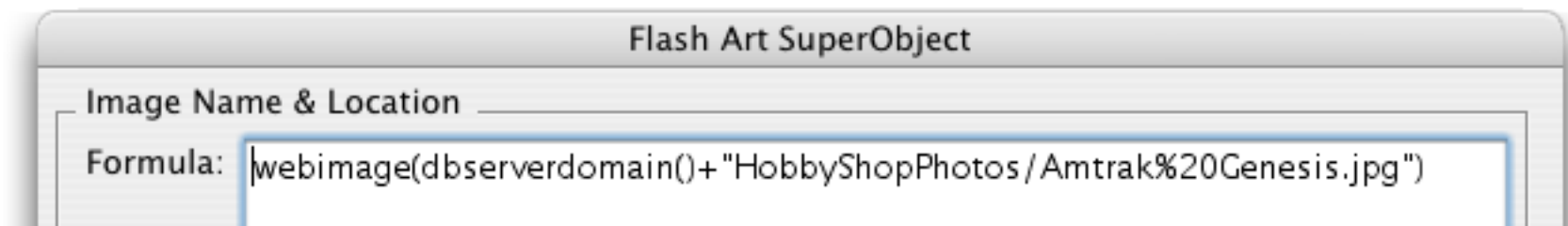
Because of the issues described in the previous paragraph, you'll usually want to create your own images and upload them to your own web site. If you're using Mac OS X with Apache then you'll want to put the images in a subfolder of the [WebServer Documents](#) folder, which you'll find inside the Library folder on the main hard drive of the computer. Usually you'll want to put the images in a subfolder, like the [HobbyShop-Photos](#) folder shown below.



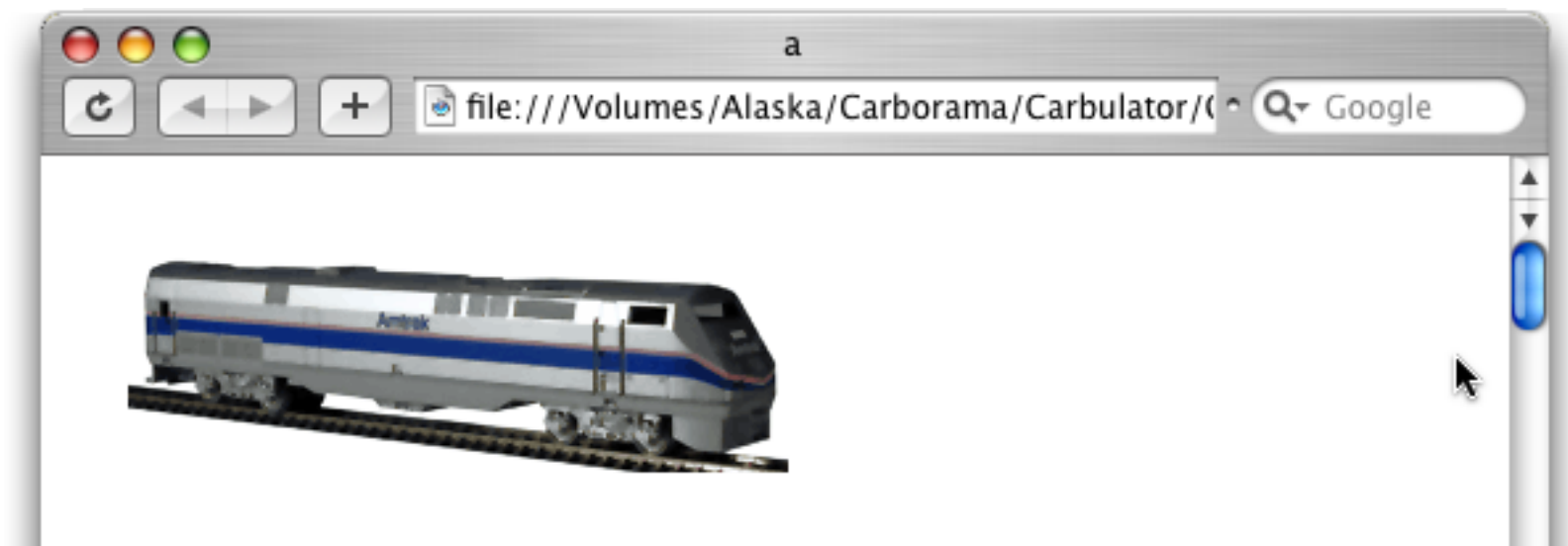
Here's the formula for displaying the image of the Amtrak Genesis locomotive:



But what if you don't know the URL for your web site yet, or perhaps you want to first test your database on a separate local server before deploying it on your live web server. By adjusting the formula to use the `dbServerDomain()` function, the form can automatically display the image from whatever server is being used to host the database itself. This function automatically generates the first part of the URL (`http://` and the domain name).



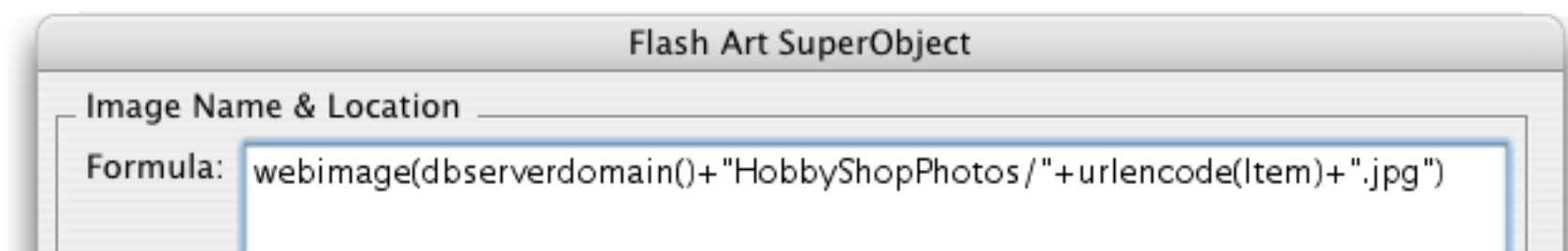
You can move the database from server to server and the images will continue to display correctly (of course you'll need to move the images also).



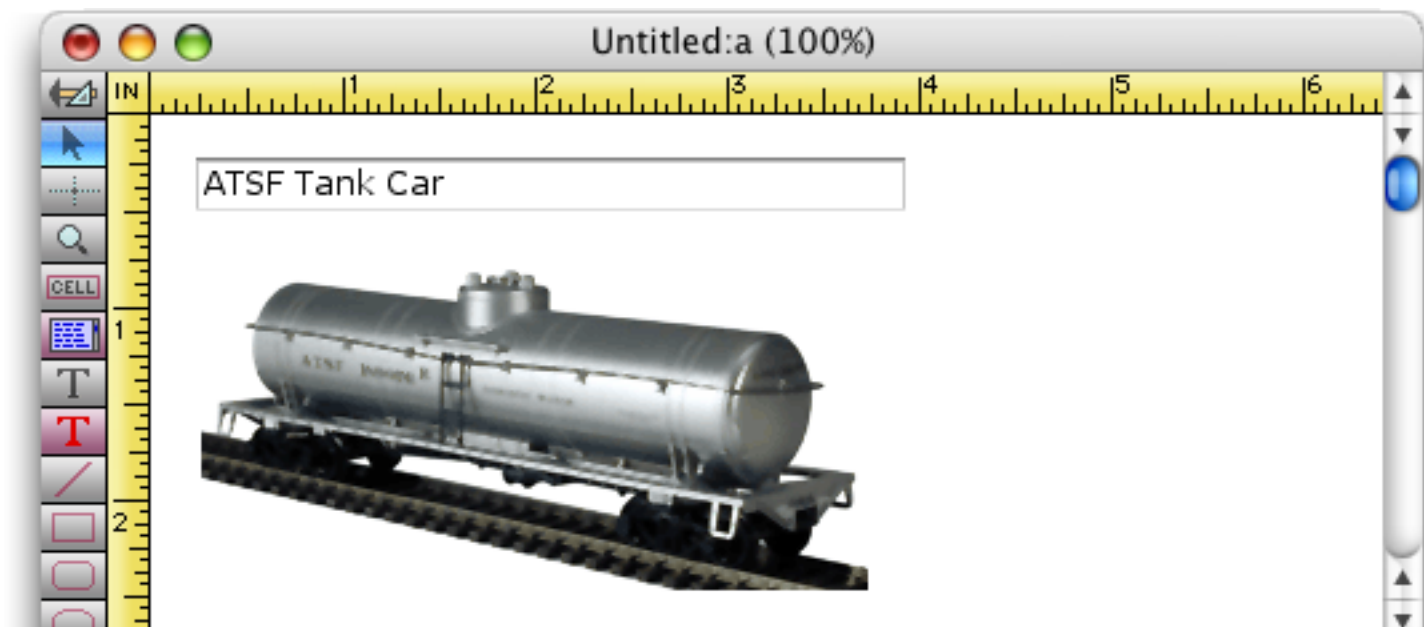
Note: The `dbserverdomain()` function does not work until you have assigned the database to a server with the Database Sharing Options wizard (see "[Uploading the Database to the Server](#)" on page 194).

Displaying Images Based on a Field or Variable

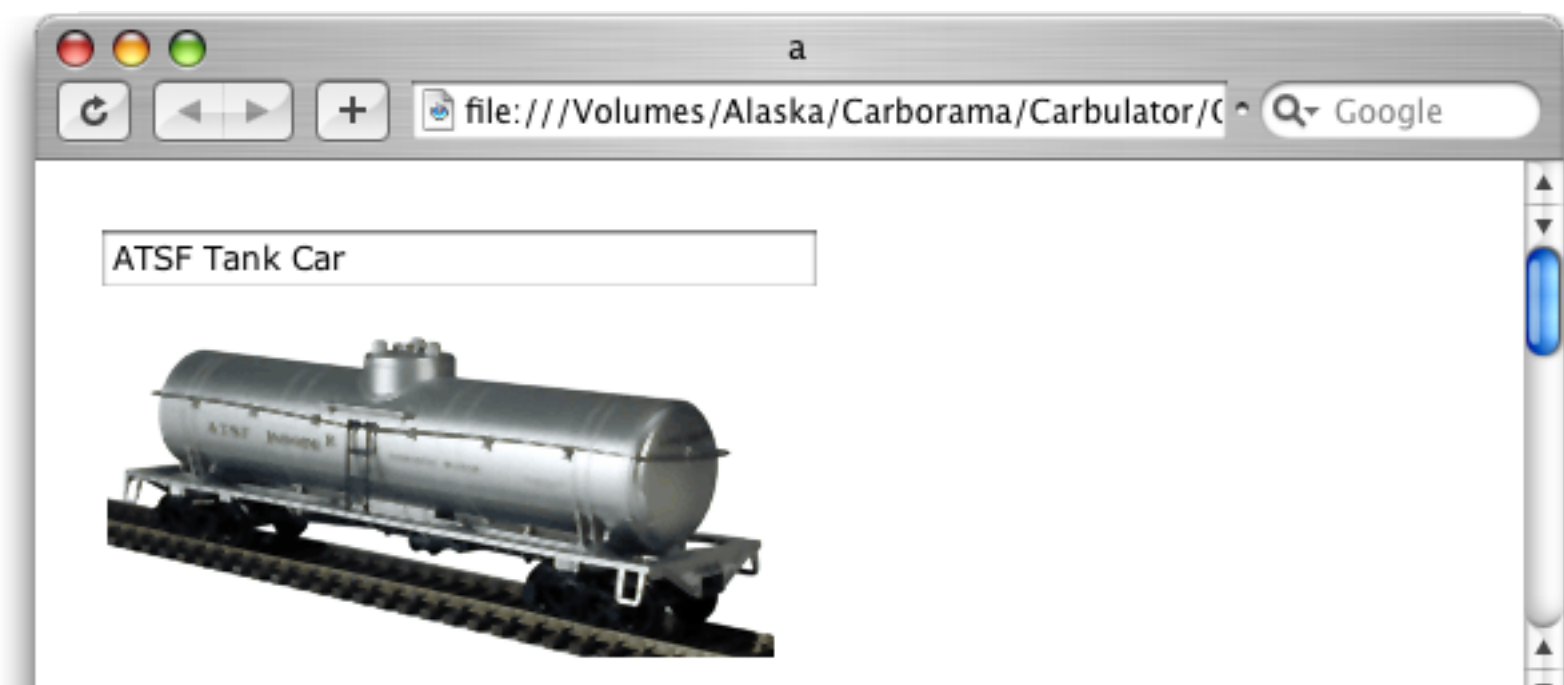
To display an image based on a field or variable simply include that field or variable in the formula. In this example the `Item` field contains image names. The `urlencode()` function is necessary to make any special characters in the image name URL friendly, for example spaces become `%20`. However, the image name should not contain the `\` character and should not contain any accented characters or other characters that use the upper 128 positions of the ASCII character set.



The Panorama form will display the image (if any) that matches the data in the Item field:

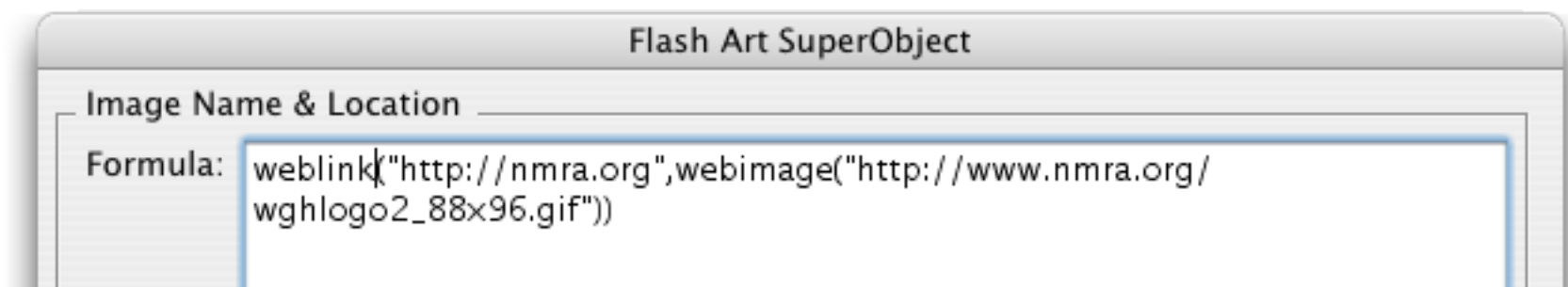


and so will the converted web form.



Making an Image Link to Another Page

To make an image a clickable link to another web page, combine the `webimage()` function with the `weblink()` function (see "[Linking to Panorama Procedures \(from a Panorama Form\)](#)" on page 261). Use the `webimage()` function as the caption parameter for the `weblink()` function.



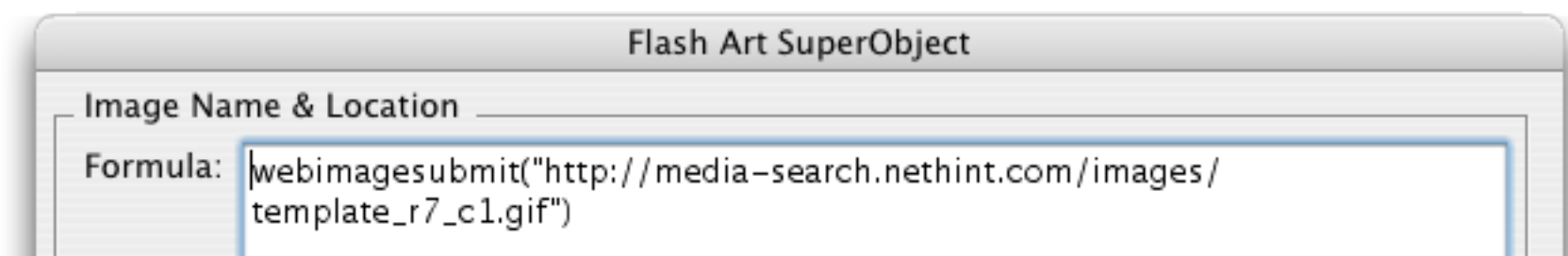
The addition of the `weblink()` function turns the image into a link to the <http://www.nmra.org> web page.



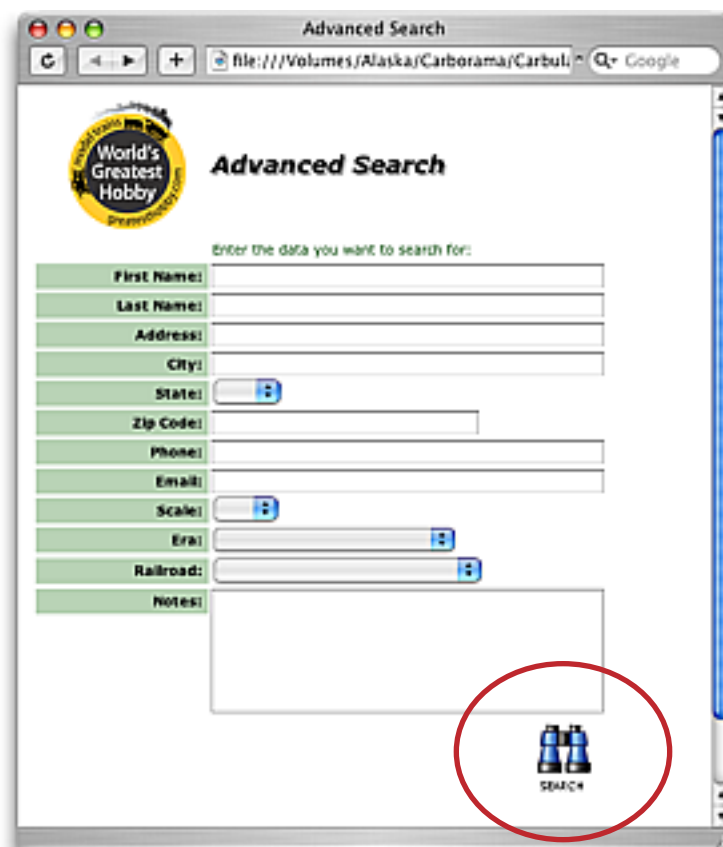
You can also use the `weblinknewwindow()`, `webformlink()`, `webformlinknewwindow()`, `cgilink()` and `cgilinknewwindow()` functions in an image formula.

Making an Image a Submit Button

To make an image act as if it is a submit button use the `webimagesubmit()` function instead of `webimage()`.

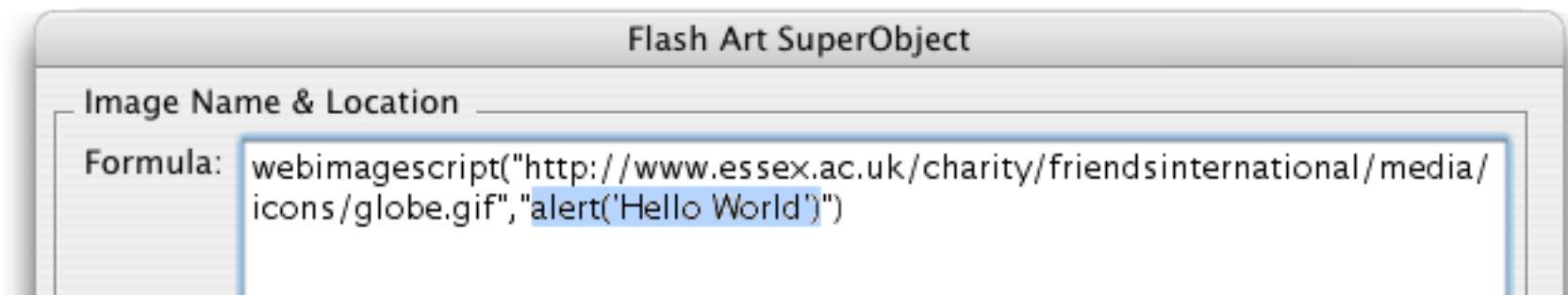


The image displays normally, but can also be clicked to submit the form.

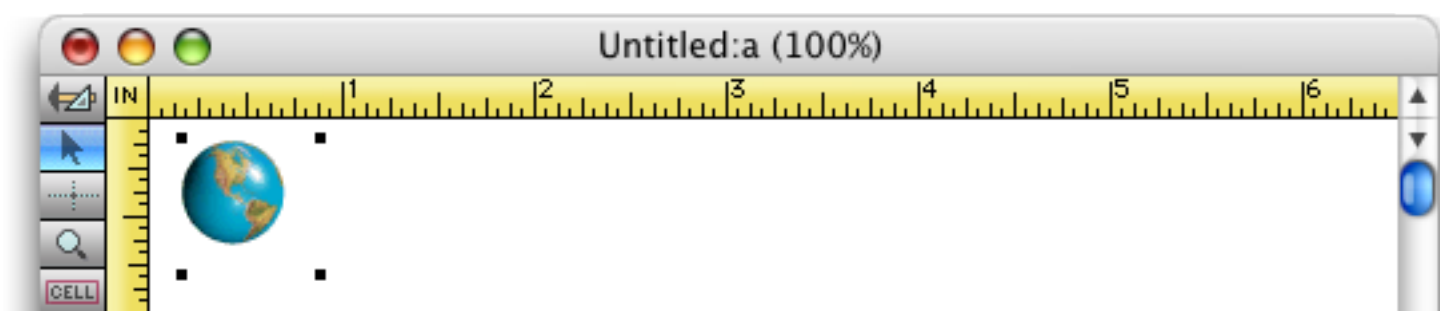


Making an Image a JavaScript Button

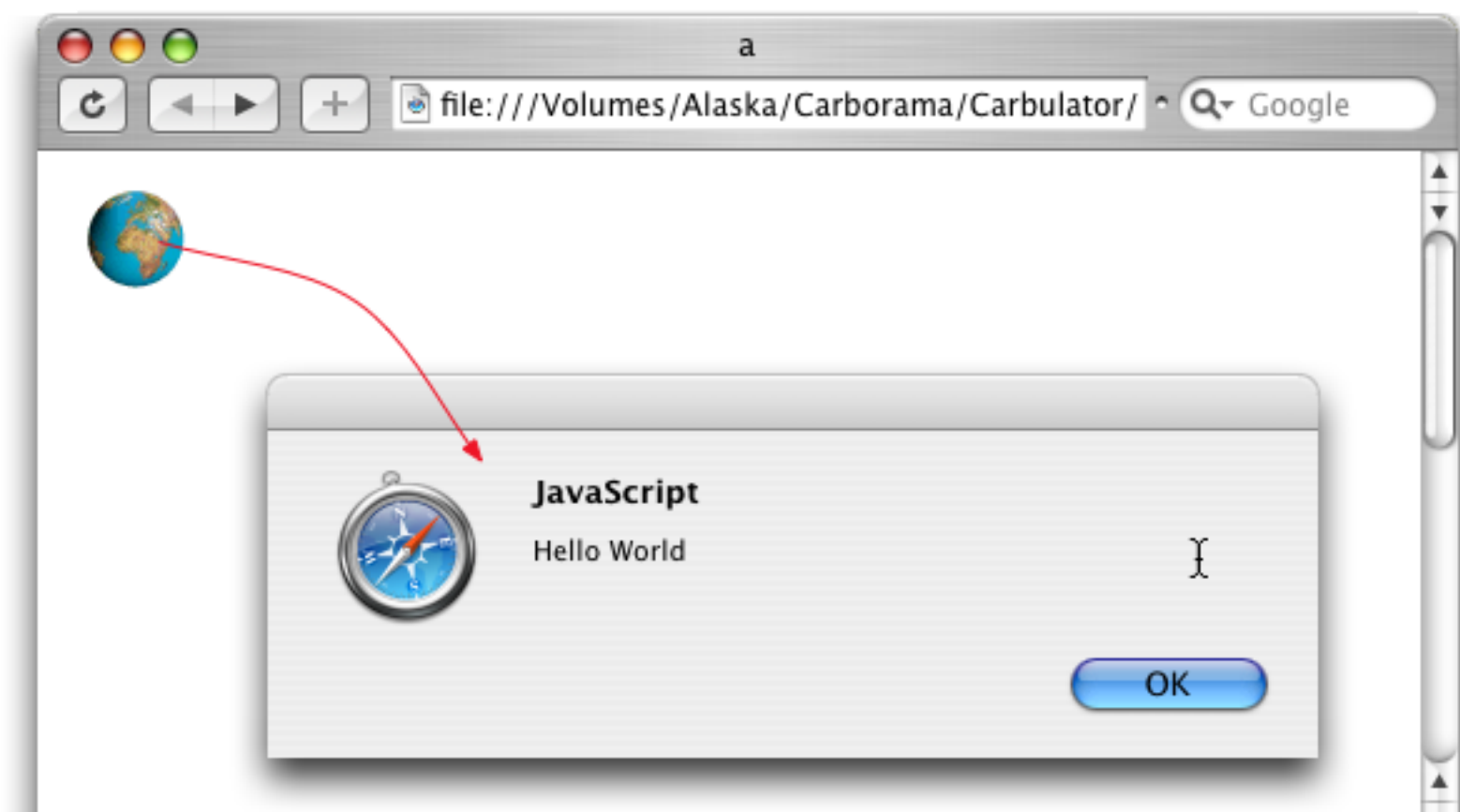
If you use the `webimagescript()` function clicking on the image will trigger a JavaScript. The second function parameter is the text of the script. Important: The script text **must not** contain the `"` character. In the example below the text of the JavaScript script is highlighted, in this case a simple script that displays an alert.



In the Panorama form, the image is displayed normally. (However, since Panorama does not have a built-in JavaScript interpreter, clicking on the link does not trigger the JavaScript.)



When the form is rendered as a web page the JavaScript becomes activated. Clicking on the image triggers the script, in this case displaying a message.



If the script you want to run has more than a couple of statements it's usually best to define a JavaScript function separately (see "[Adding JavaScript to a Page](#)" on page 275) and simply trigger this function with the `webimagescript()`.

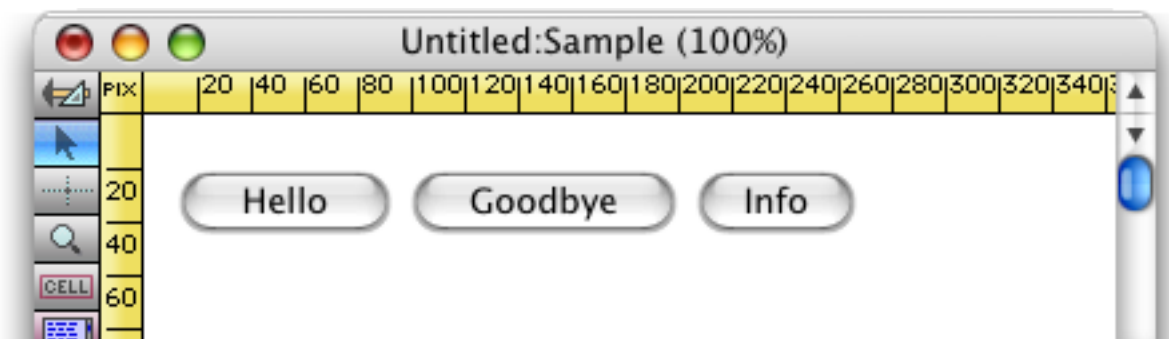
Triggering JavaScript with a Button

Buttons are normally used to submit the form, or to reset the form data (see “[Push Buttons](#)” on page 240). They can also be used to trigger JavaScript. To set this up, use the **Button Scripts** dialog (in the **Web** submenu of the **Setup** menu).



To use this dialog enter one or more button javascripts, one per line. Each line starts with the button title, then an equals sign (=), followed by the script (which must not include the " character).

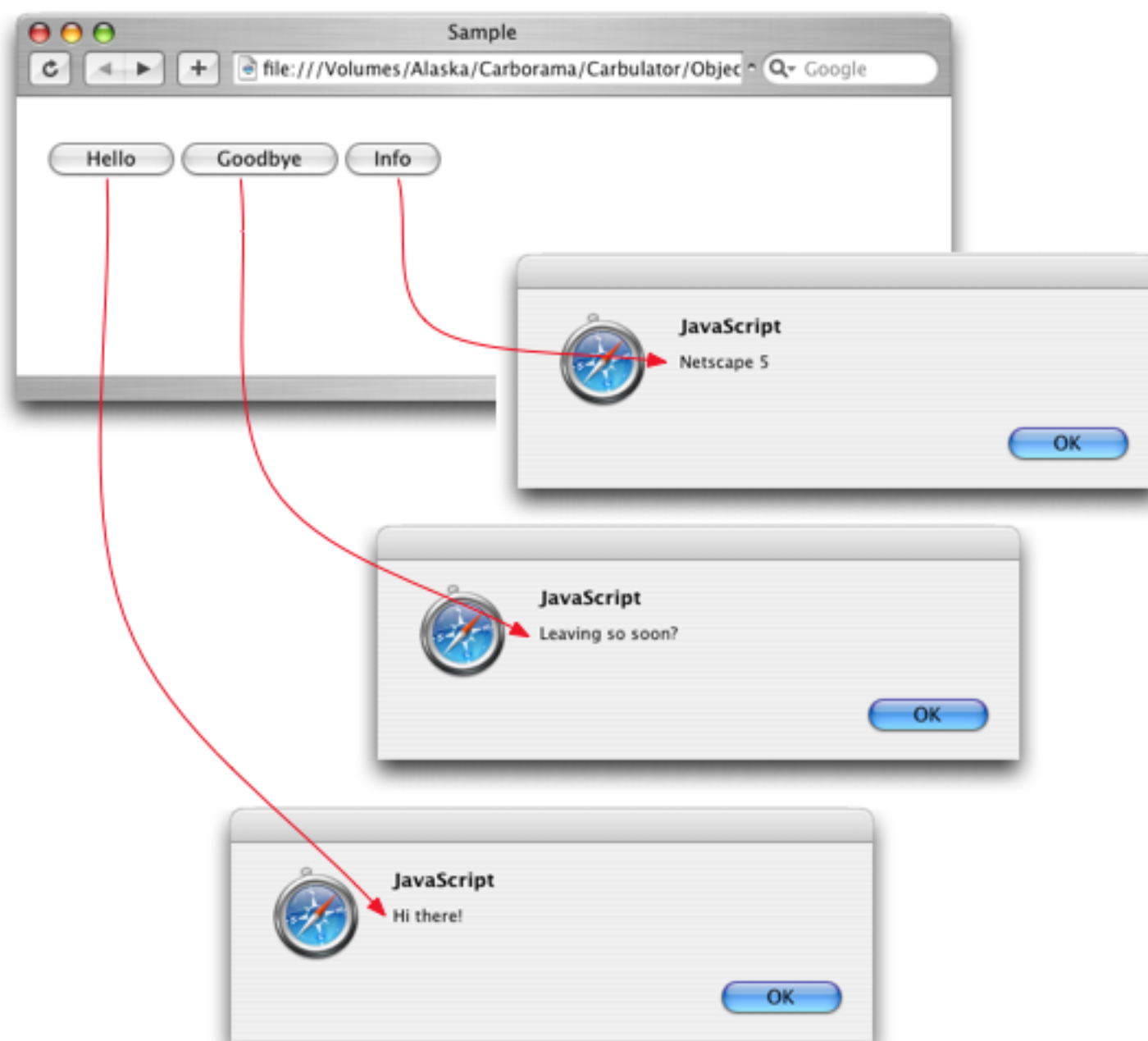
For example, suppose you have created a form with several buttons, like this:



Once the buttons are set up, open the **Button Scripts** dialog and type in the JavaScript for each button.



Now when the form is converted to a web page pressing these buttons triggers the corresponding JavaScript, in this case displaying a message.



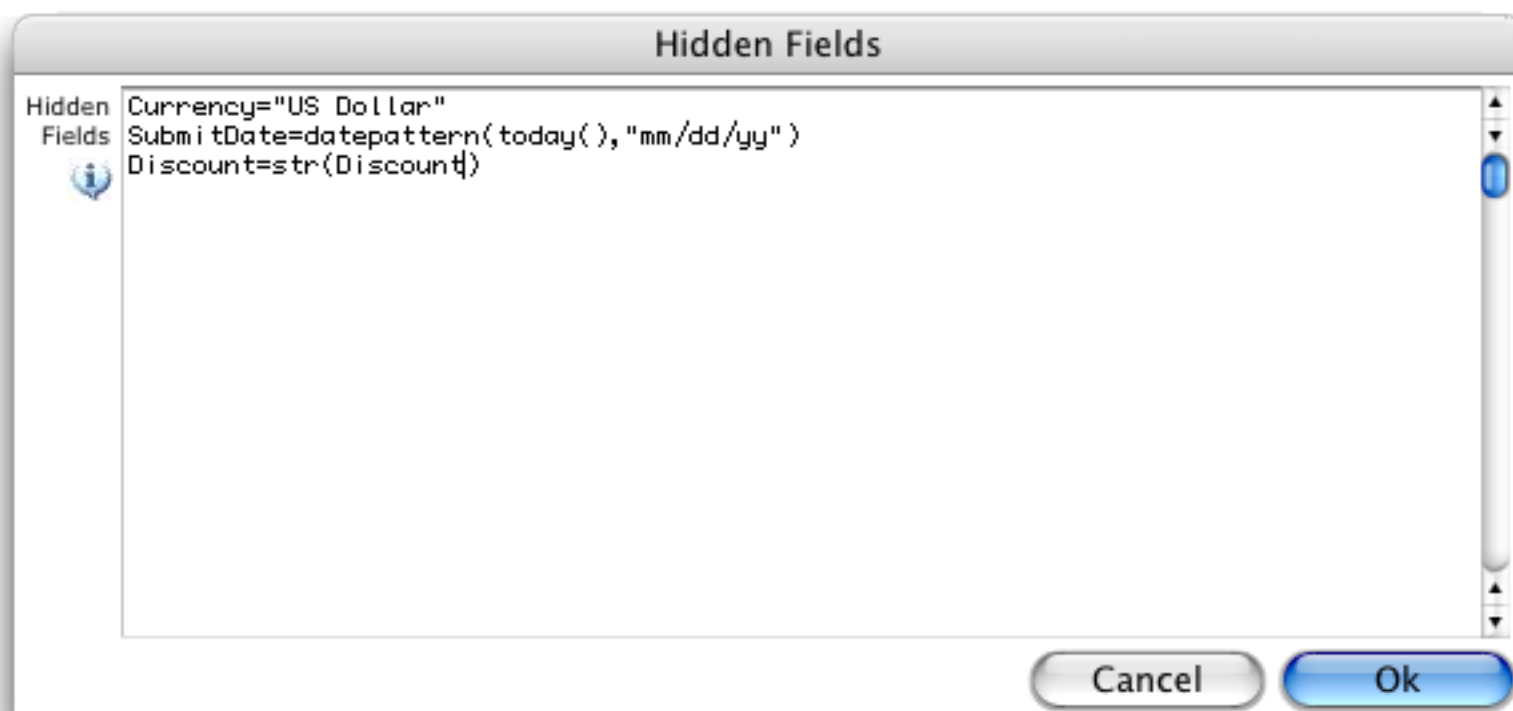
To learn more about how to add JavaScript to a form see “[Adding JavaScript to a Page](#)” on page 275.

Hidden Data

When the **Submit** button is pressed, the web browser collects all of the data that has been entered on the form and submits it to the web server (hence the name of the button). The web server then processes this data to perform the requested operation — searching, entering data, etc. In addition to normal visible data values (text editing cells, checkboxes, radio buttons, etc.) web browsers also support hidden values. These values are generated when the web form is displayed, then stored invisibly within the web form and submitted with the

other data values when the **Submit** button is pressed. Essentially these hidden values allow the server to pass values to itself, allowing it to remember a value or calculation from a previous interaction with the server. For example a hidden value could contain the time the form was displayed, allowing the server to calculate how long it took you to fill in the form.

To set up one or more invisible values choose the **Hidden Fields** command in the **Web** submenu of the **Setup** menu (see “[Converting a Panorama Form into a Web Form](#)” on page 231):



Using this dialog you can enter one or more hidden field assignments, one per line. Each line starts with the name of the hidden field, then an equals sign (=), followed by a formula that specifies the value for that hidden field. The formula must calculate a text result (not numeric or date). In the example above three hidden values are created — **Currency**, **SubmitDate** and **Discount**.

To learn how to process hidden values see “[Accessing Form Item Values in a Formula](#)” on page 435.

Customizing the form HTML (Advanced)

If you are into HTML, you can customize the web pages Panorama generates. Otherwise you can simply leave these options alone and let Panorama do the heavy lifting for you!

Customize Page Dialog

Open this dialog from the **Web** submenu of the **Setup** menu (see “[Converting a Panorama Form into a Web Form](#)” on page 231).

This dialog allows you to customize seven aspects of the generated HTML page.

Web Page Title. Enter the title of the web page here. The title will appear at the top of the browser window.

If you leave the title blank the Panorama form name will be used.

Form Tag Parameters. The `action=` and `method=` parameters are inserted into the form tag automatically. Enter any additional parameters that you want included in the form tag here. Typical parameters that you might want to add to the form tag include `name=`, `onsubmit=`, and `onreset=`.

Form Prefix. This text will be placed at the top of the form, right after the `<form>` tag and before any objects generated from the Panorama form.

Form Suffix. This text will be placed at the end of the form, right before the `</form>` tag and after any objects generated from the Panorama form.

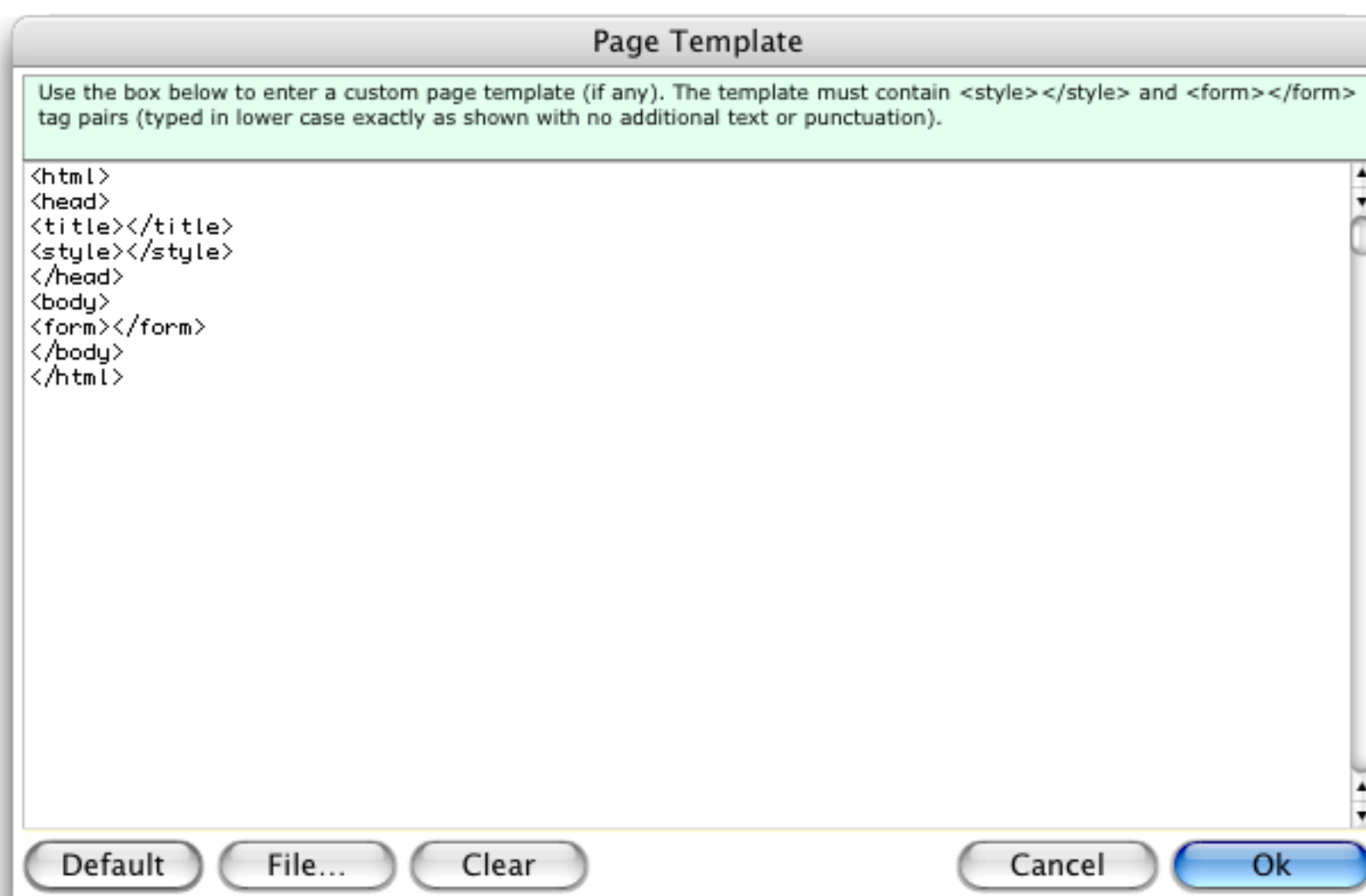
Style Tag Parameters. Enter any additional parameters that you want included in the style tag here (for example `media=` parameters).

Style Prefix. This text will be placed at the top of the style section of the page, right after the `<style>` tag and before any styles generated from the Panorama form. This is where you should put `@import` elements to import external style sheets, if necessary.

Style Suffix. This text will be placed at the bottom of the style section of the page, right before the `</style>` tag and before any styles generated from the Panorama form.

Page Template Dialog

This dialog modifies the template Panorama uses for generating the HTML page. Using this dialog you can add additional items to the `<head>` section of the page (for example JavaScript), modify the `<body>` tag to change the background color, etc. The illustration below shows this dialog with the default HTML template (which was inserted into the dialog by pressing the **Default** button).

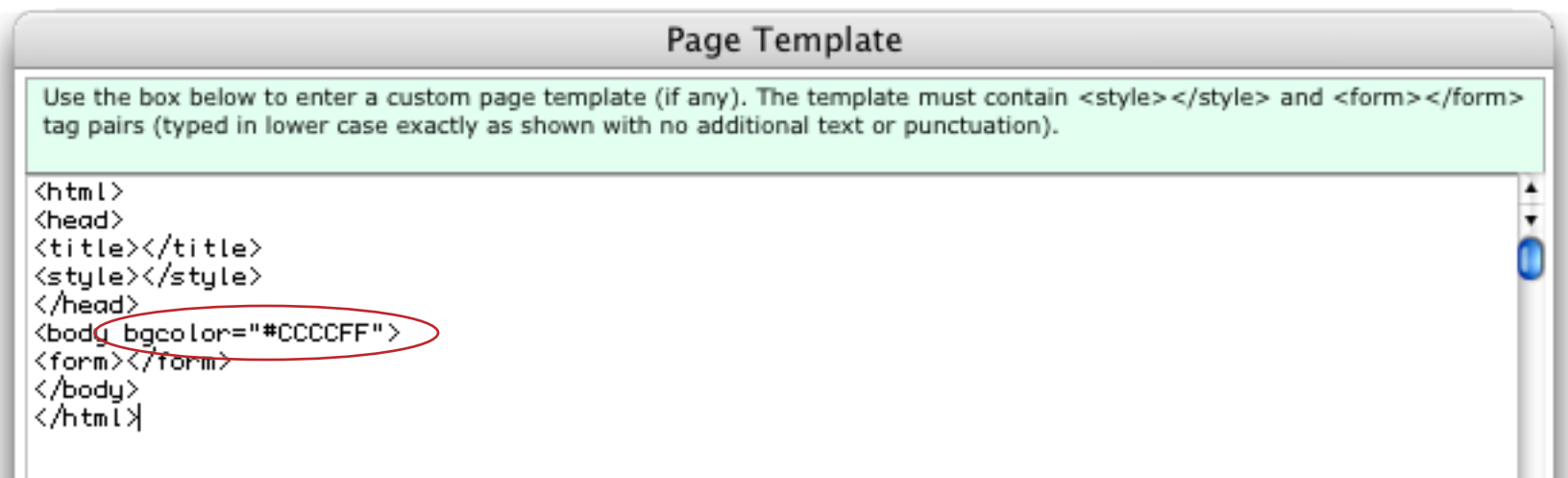


The **Default** button inserts the default page template into the dialog. This is usually the first step, from there you can modify the template for your specific needs.

The **File** button allows you to select an external text file to use as the template. This allows you to use a page built by an external program like *Dreamweaver* or *Freeway*.

The **Clear** button erases the text in the dialog. If the text is empty, the default template will be used.

Changing the Page Background Color. In this example the `<body>` tag has been modified to change the background color of the page to pale blue.

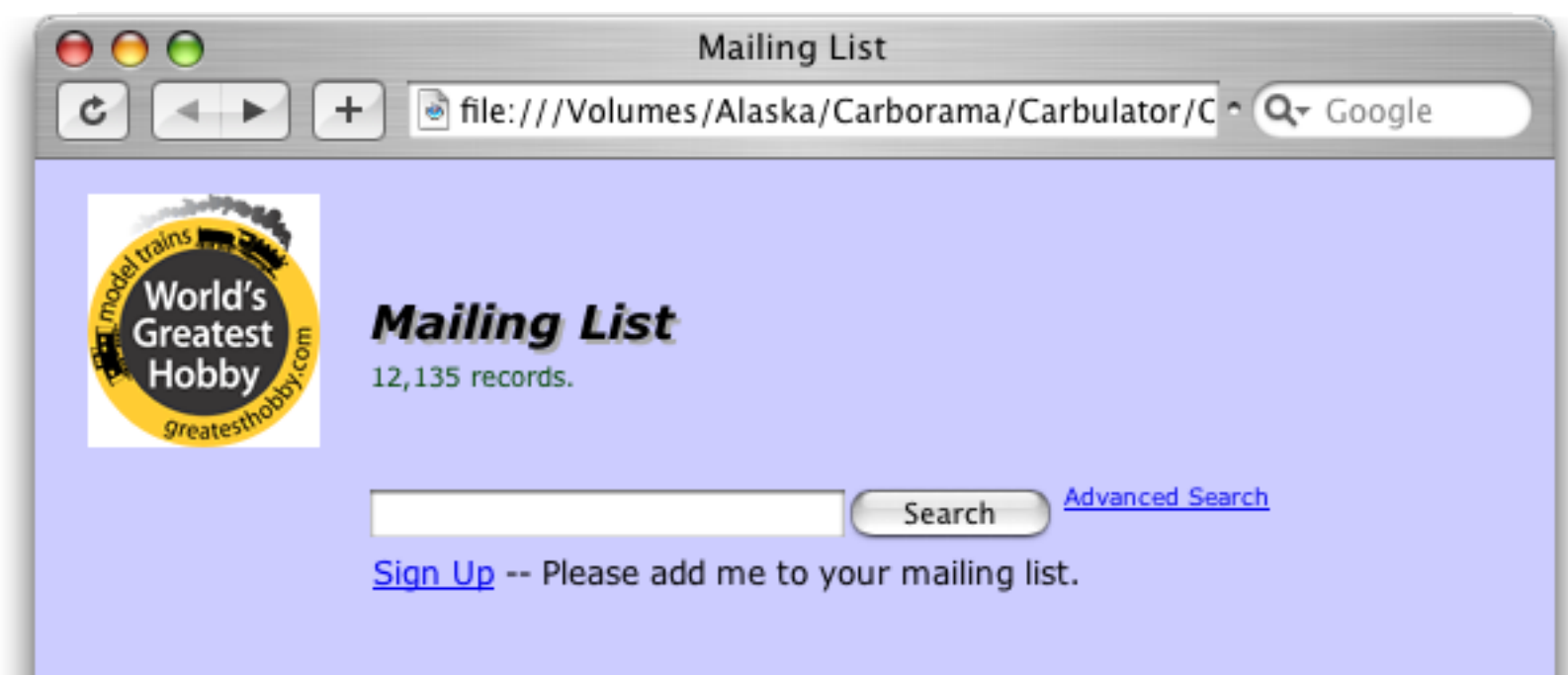


```

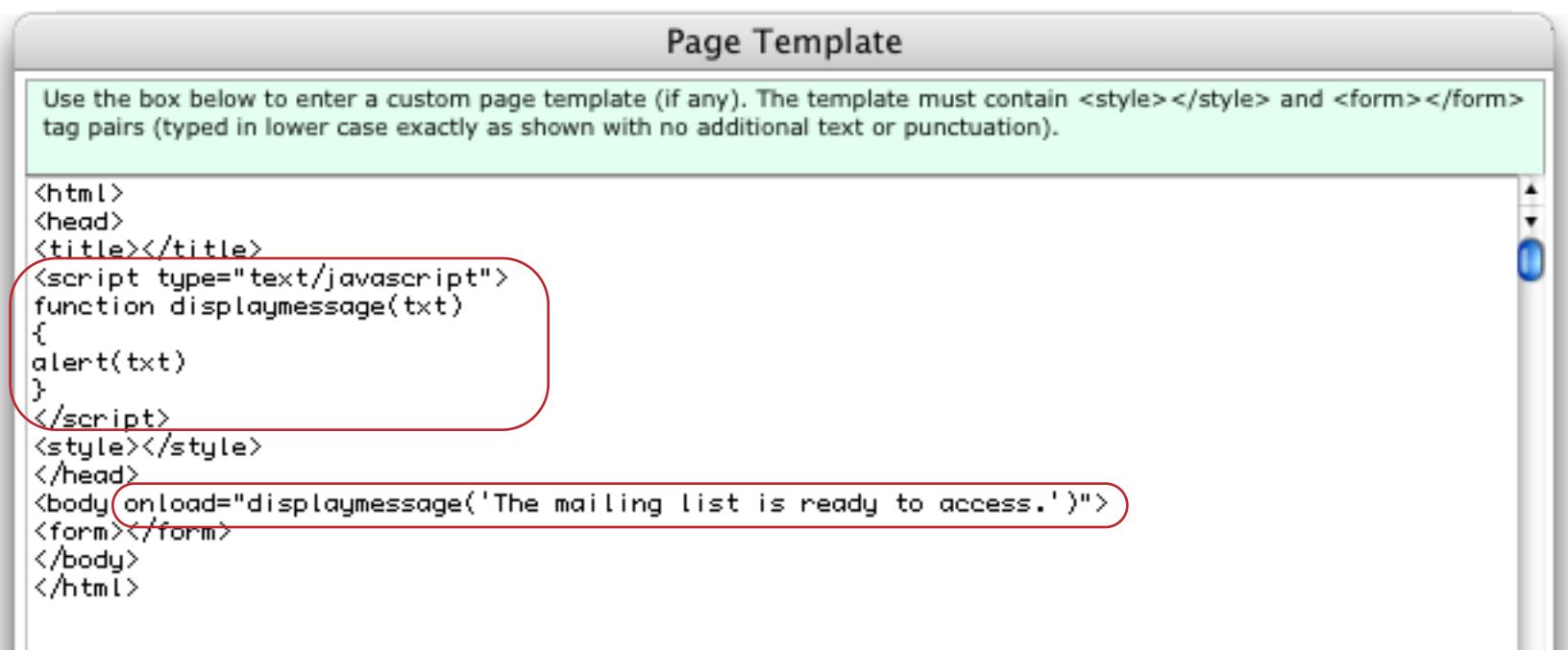
<html>
<head>
<title></title>
<style></style>
</head>
<body bgcolor="#CCCCFF">
<form></form>
</body>
</html>

```

Here's what a page might look like if rendered with this template:



Adding JavaScript to a Page. In this example a JavaScript function has been added to the header of the HTML template.

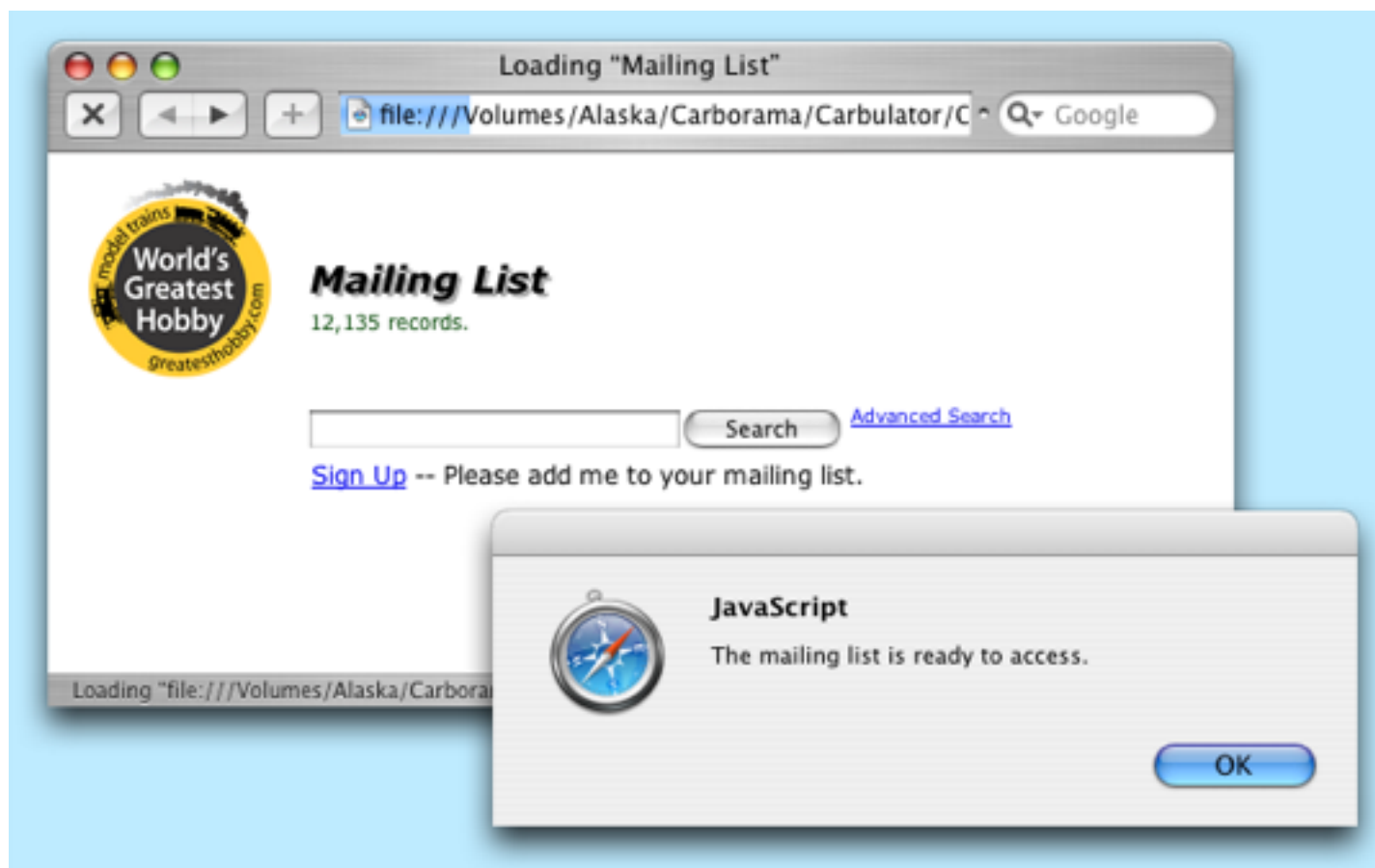


```

<html>
<head>
<title></title>
<script type="text/javascript">
function displaymessage(txt)
{
alert(txt)
}
</script>
<style></style>
</head>
<body onload="displaymessage('The mailing list is ready to access.')">
<form></form>
</body>
</html>

```

This function is triggered by the `onload=` parameter of the `<body>` tag so the message will appear automatically when the page finishes loading from the server.



Building a form in an external program. There are many web development programs that can create web forms (Dreamweaver, GoLive, etc.). When working with Panorama databases, it's usually much simpler to generate the web form from a Panorama form, because then all of the details are taken care of for you. However, if you want to build a web form using an external editor you can. If the page template doesn't contain the tags `<style></style>` and `<form></form>` then the page template becomes the actual finished form (any Panorama objects on the Panorama form are ignored). If you do this, you can embed Panorama formulas into the form by including the formula between `~{` and `}~` tags (see "[Rendering Using an External Text File as a Template](#)" on page 483). You also need to make sure that any `<input>`, `<textarea>` and other form item tags use the special format required by Panorama's form processing actions. See "[Rendering Using an External Text File containing a Form as a Template](#)" on page 484 for more information about these special formats.

Chapter 7: Web Tables



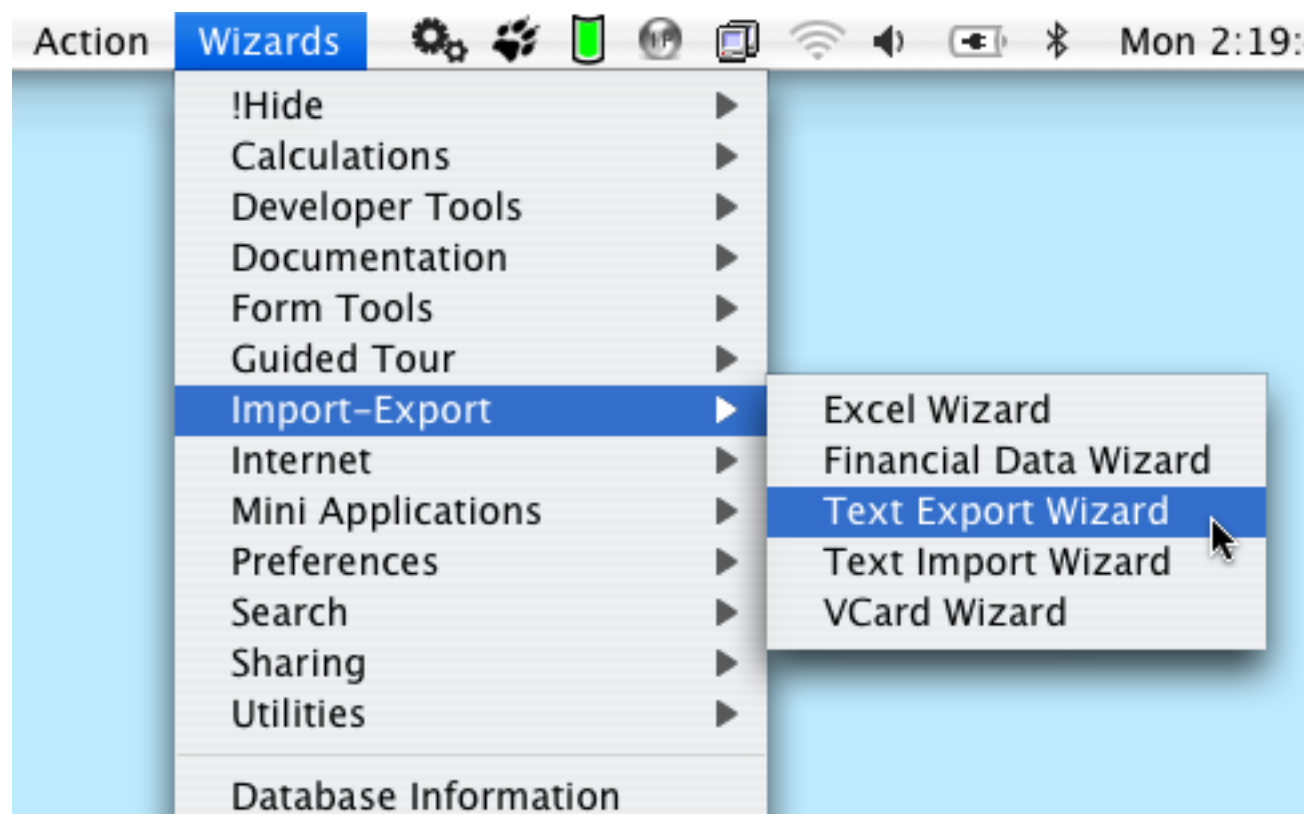
The Panorama Enterprise Edition Server can publish any kind of HTML web page, but there are two basic building blocks that are the most common elements of any dynamic site — forms and tables. This chapter describes tables (see the previous chapter to learn how to create Panorama web forms).

Web Tables

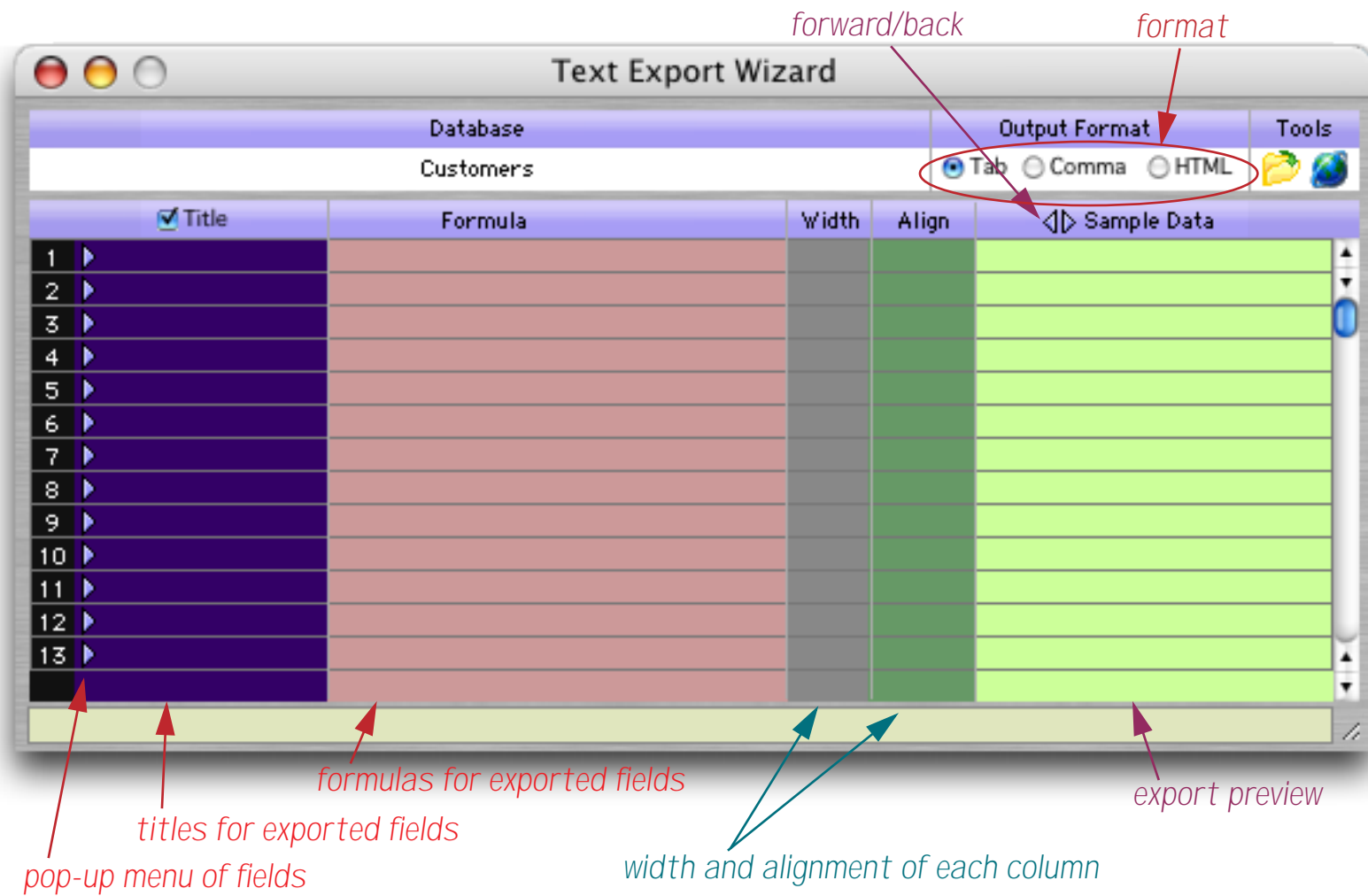
Web table templates are created and modified with the **Text Export** wizard. This may seem like an odd choice until you remember that web pages are simply text pages with HTML tags in them. Before the Panorama Enterprise Edition Server was created, the **Text Export** wizard already had the capability to create web pages from a database, so we simply added the options necessary to integrate this feature with the server.

Text Export Wizard Refresher

Haven't used the **Text Export** wizard lately? Don't worry, we'll cover the portions of this wizard you need to know for setting up server web tables right now. For complete documentation of this wizard see Chapter 1 of the Panorama Handbook. You'll find the **Text Export** wizard in the **Import-Export** submenu of the **Wizards** menu.

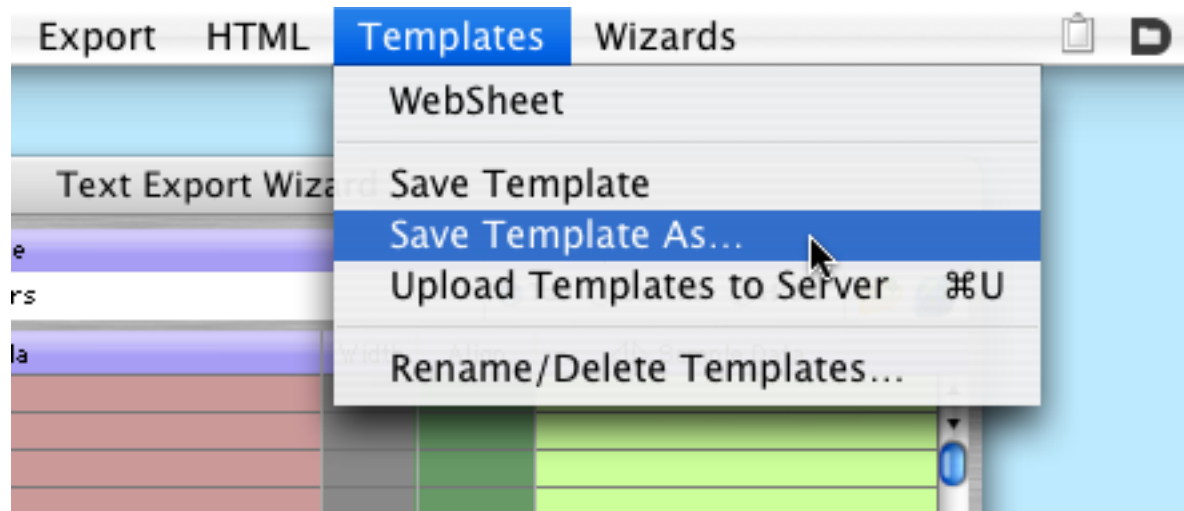


The wizard starts out looking like this.

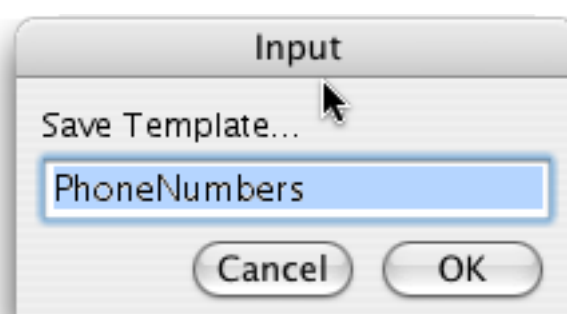


Creating a Template

If you are setting up a table for use with the server the first thing you'll need to do is create a new template (or select a template you've created previously). To create a new template use the **Save Template As...** commands in the **Template** menu.

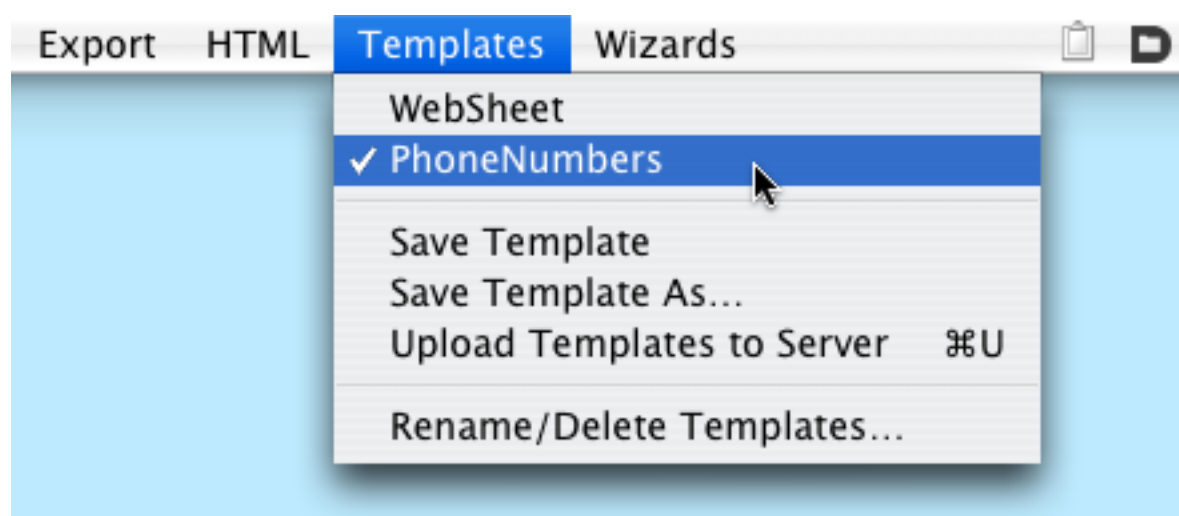


The wizard will prompt you to type in a name for the new template (the default is the name of the text file being exported).



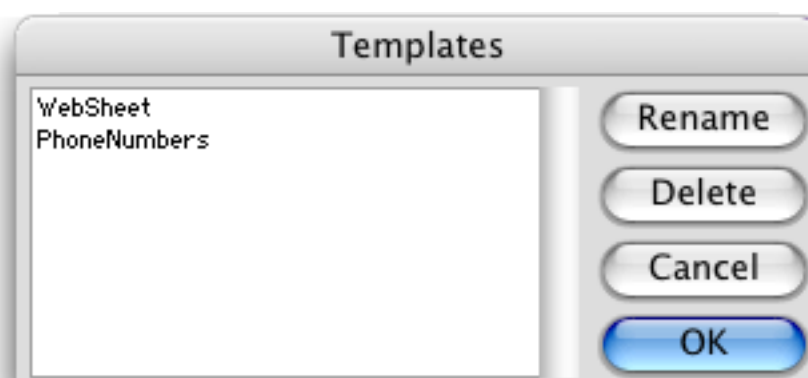
Using an Existing Template

Once a template has been saved you can open it again by selecting it from the list at the top of the **Template** menu.



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



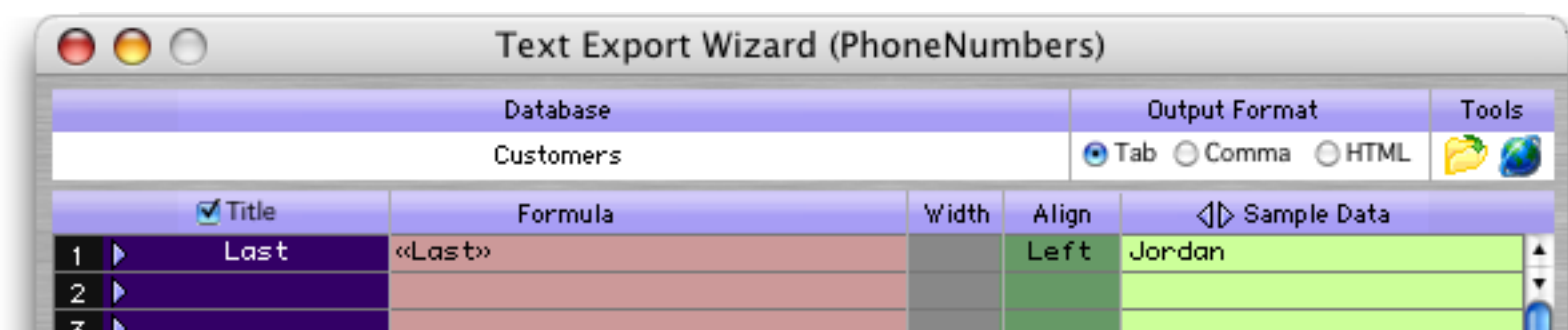
When you are done press the **OK** button.

Configuring the Table Columns (Title, Formula, Width and Alignment)

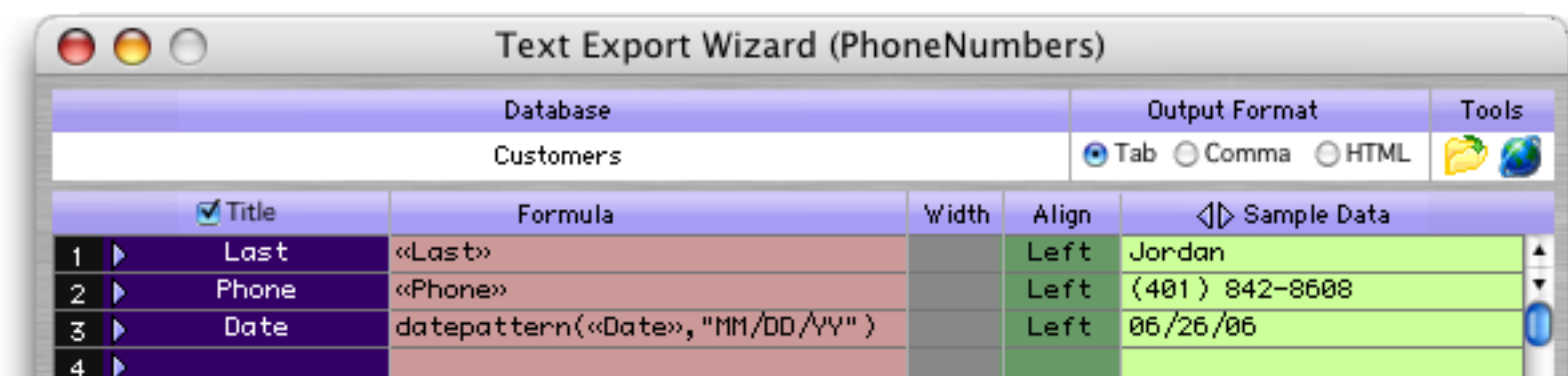
The main body of the wizard is used to configure the columns that appear in the table. To set up a field to be exported, simply choose that field using the pop-up menu



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

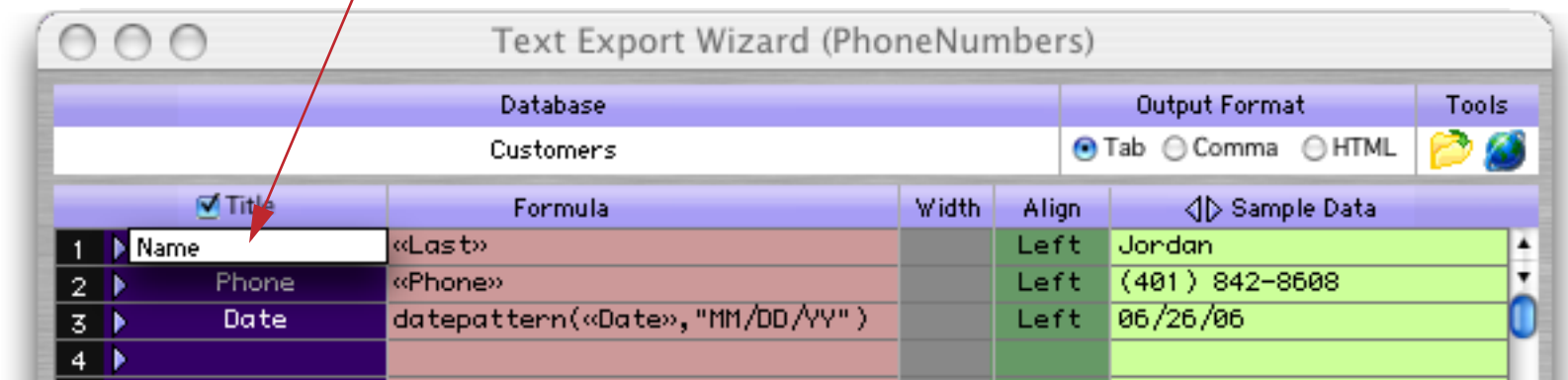


Repeat the process to select all of the fields you want to export. If a numeric field is selected, the wizard will use the `pattern()` function to convert the number into text, as shown below. If a date field is selected, the `datepattern()` function will be used to perform the conversion.



To edit a column title simply double click on it.

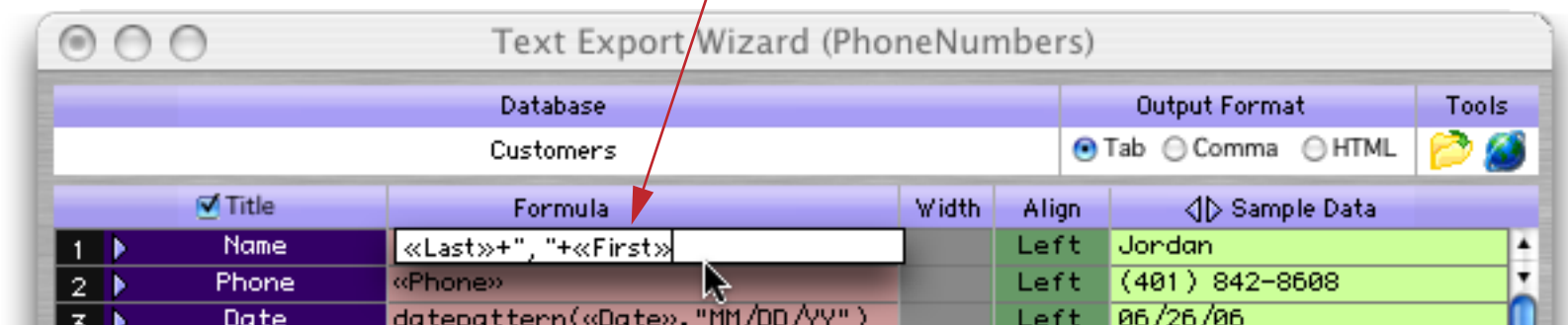
double click, then type in new name, then press ENTER



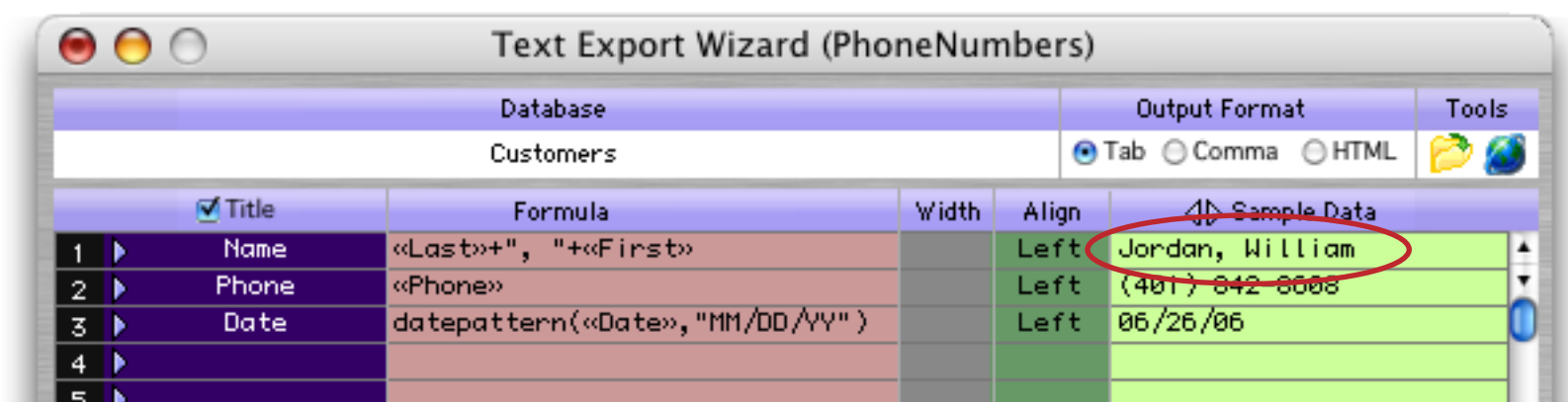
Type in the new title, then press the **Enter** key.

The formula for the column can also be edited by double clicking. For example you can use a formula to combine two database fields into one column in the web table, like this.

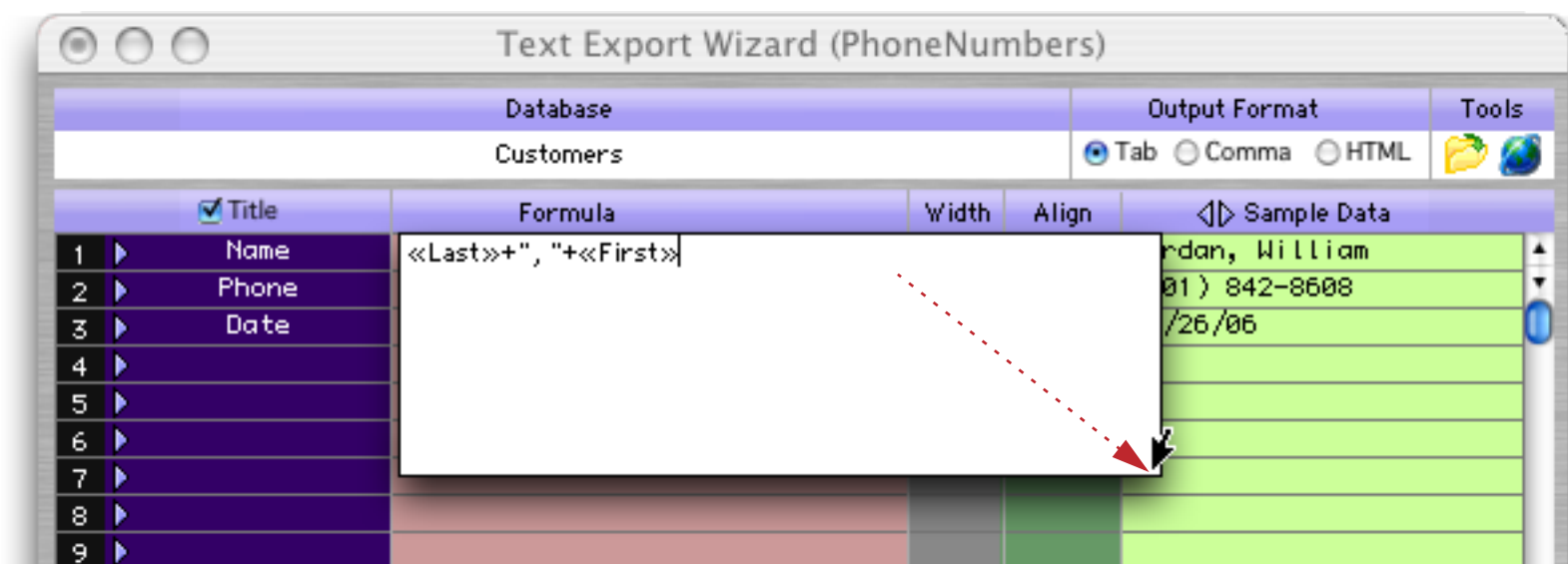
double click, type in new formula, then press ENTER



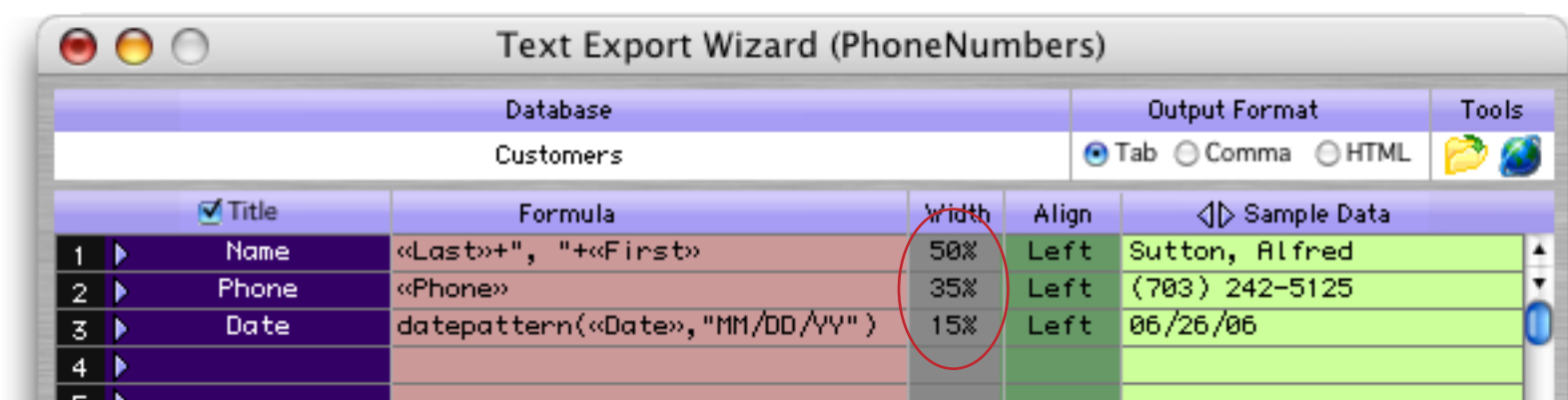
When you press the **Enter** key, the **Sample Data** will update to show what the data in this column will look like.



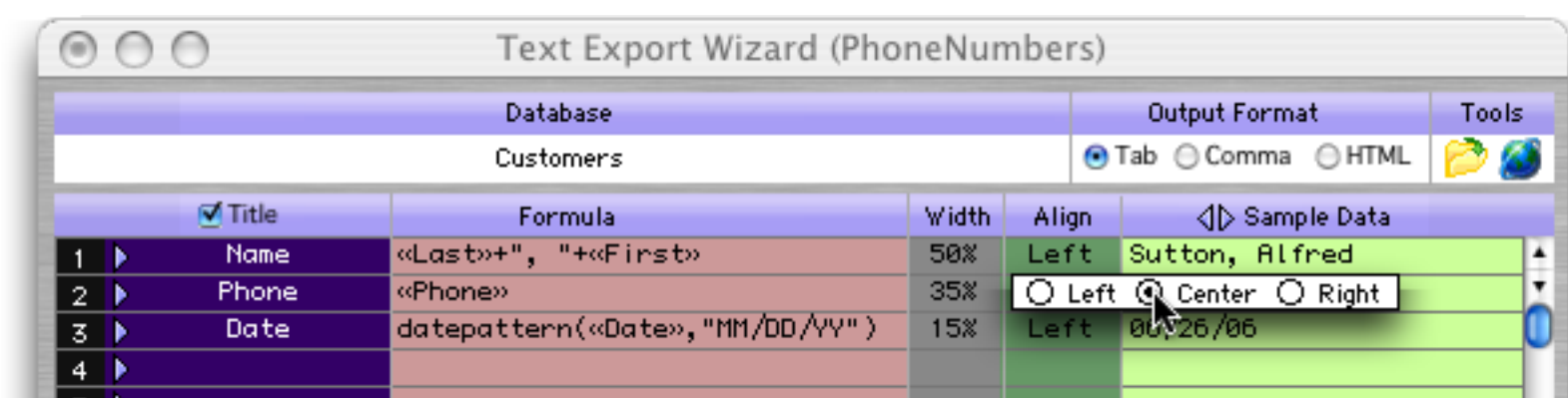
You can make the formula as complicated as you like. To make the editing area larger just drag on the bottom right corner like you would for a data sheet data cell.



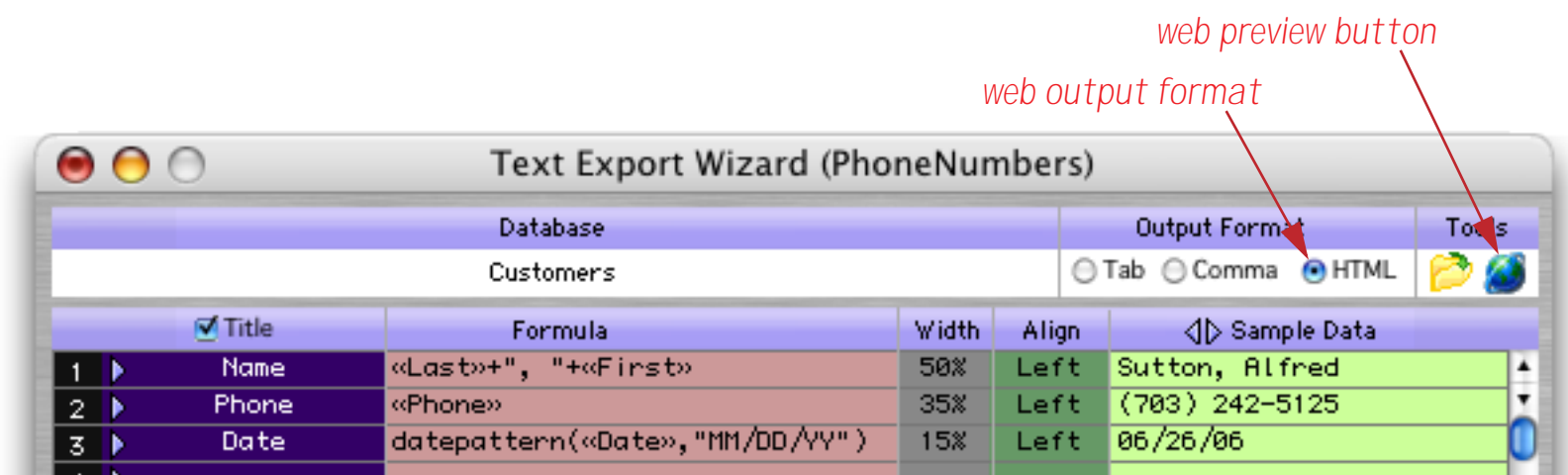
If you leave the width blank, the web browser will automatically assign the width of each column (based on the width of the widest item in the column). You can also fill in a specific value for the column width, either in pixels (for example 180) or as a percentage (for example 35%). If you specify a width for one column you should specify a width for all of them.



The final option is the alignment of the text with each column, which may be left, center or aligned on the right.



Click the **HTML** radio button to specify that this template should be exported in HTML format. Then press the **web preview** button to see what your table is going to look like in the web browser.



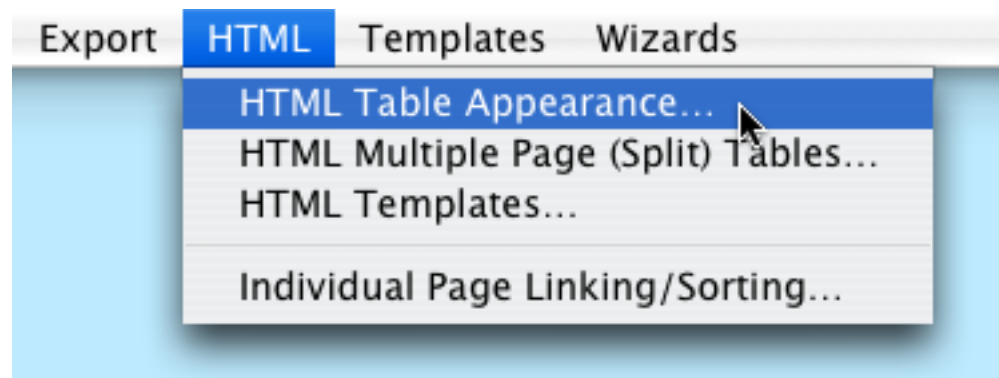
Here's what this table looks like so far.



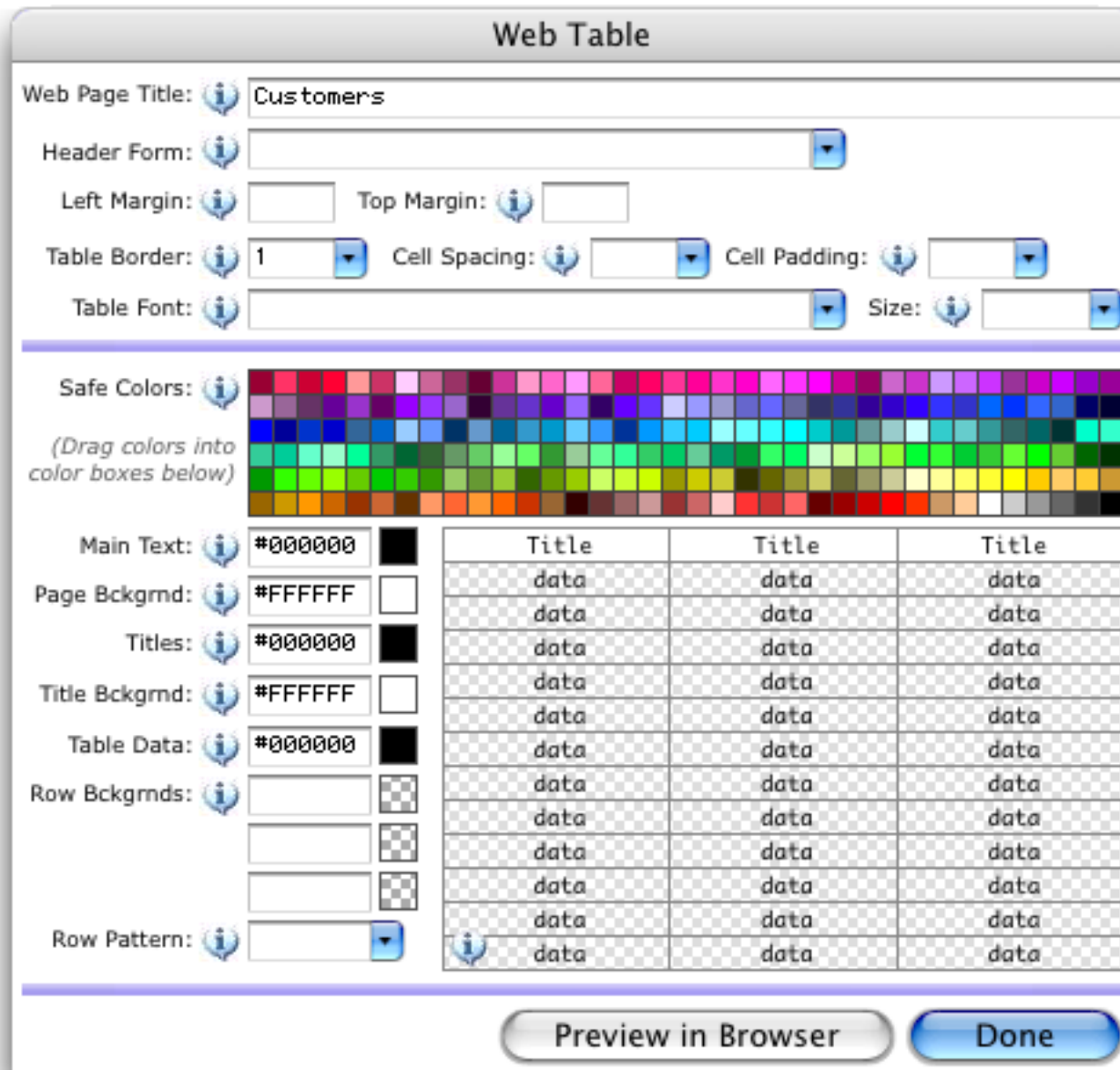
Not very exciting, but functional. We'll look at how to dress up the table in the next section.

Customizing the Table Appearance

The HTML Table Appearance dialog gives you control over many aspects of the table format.

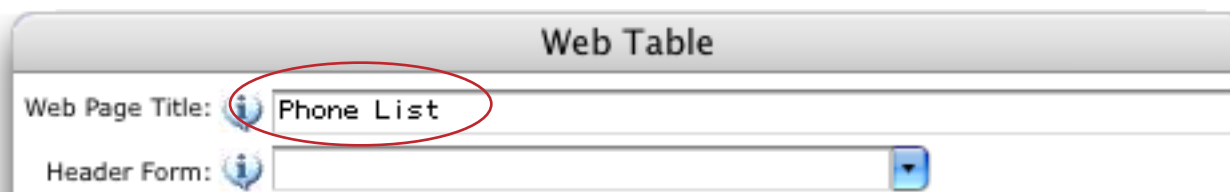


This dialog allows you to control the title, margins, font and color of the table.



Web Page Title

The **Web Page Title** is the title that appears in the title bar of the browser window when this table is displayed. It defaults to the database name. For example, suppose you set the title to **Phone List**.



Here's the table with title displayed in a browser.

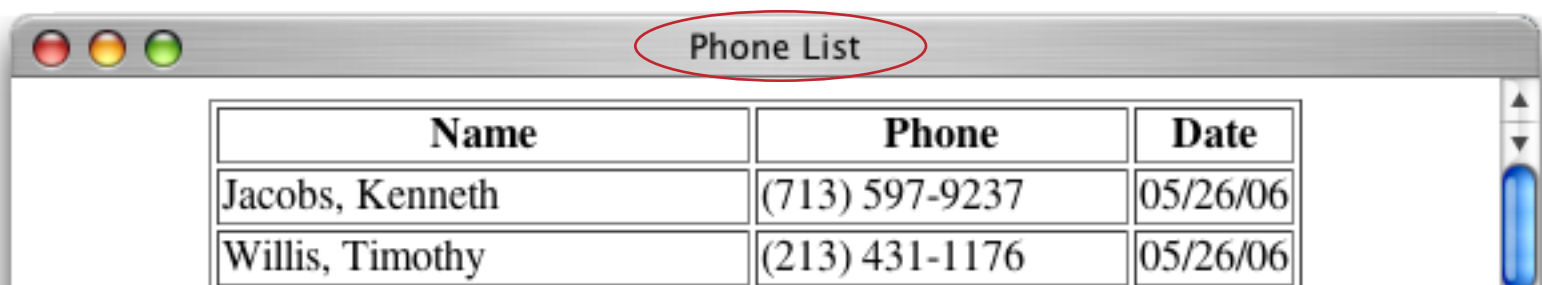
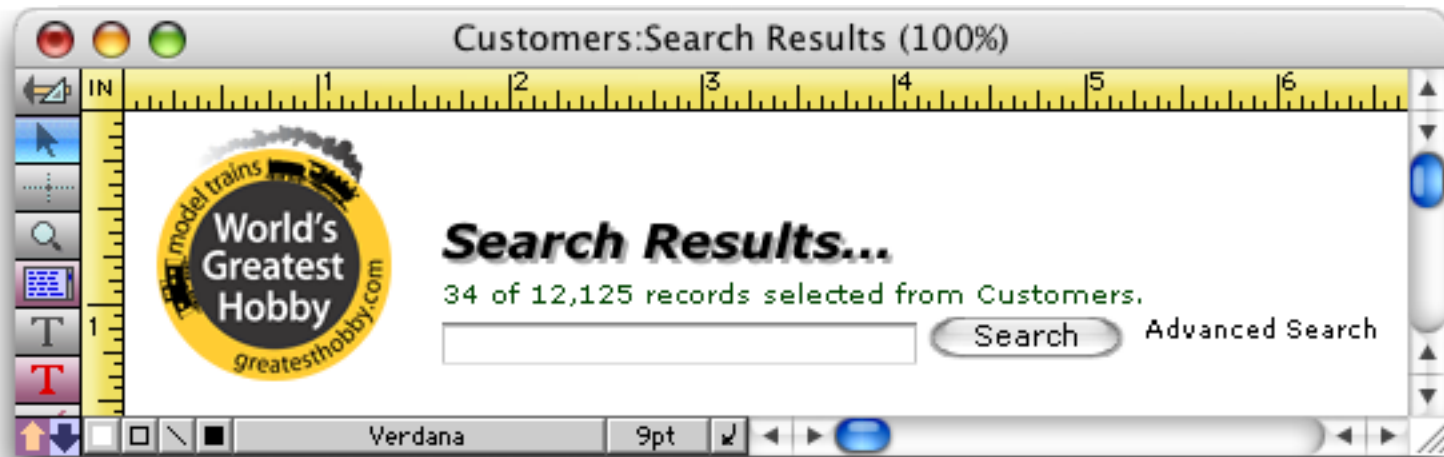
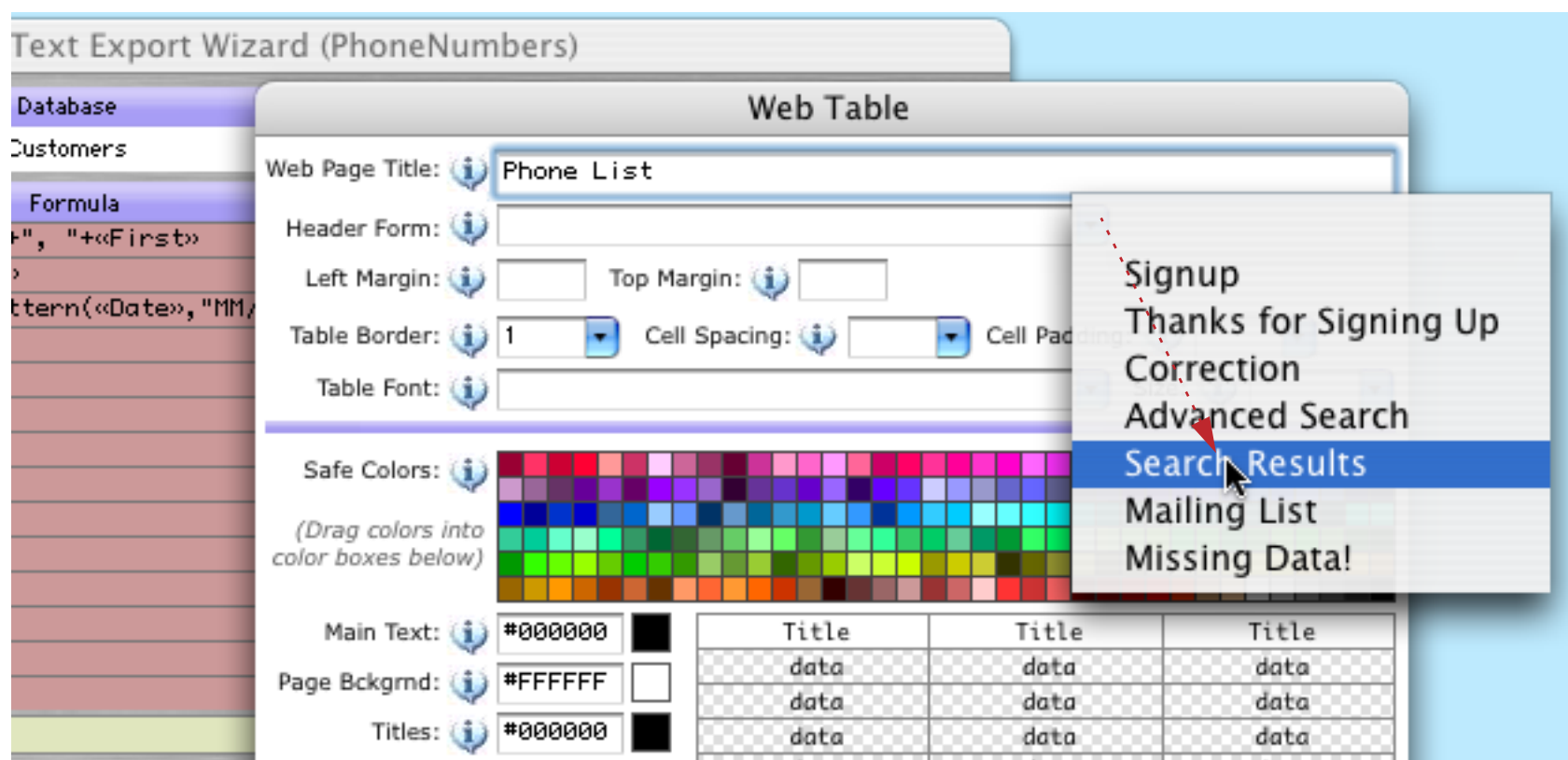


Table Header Form

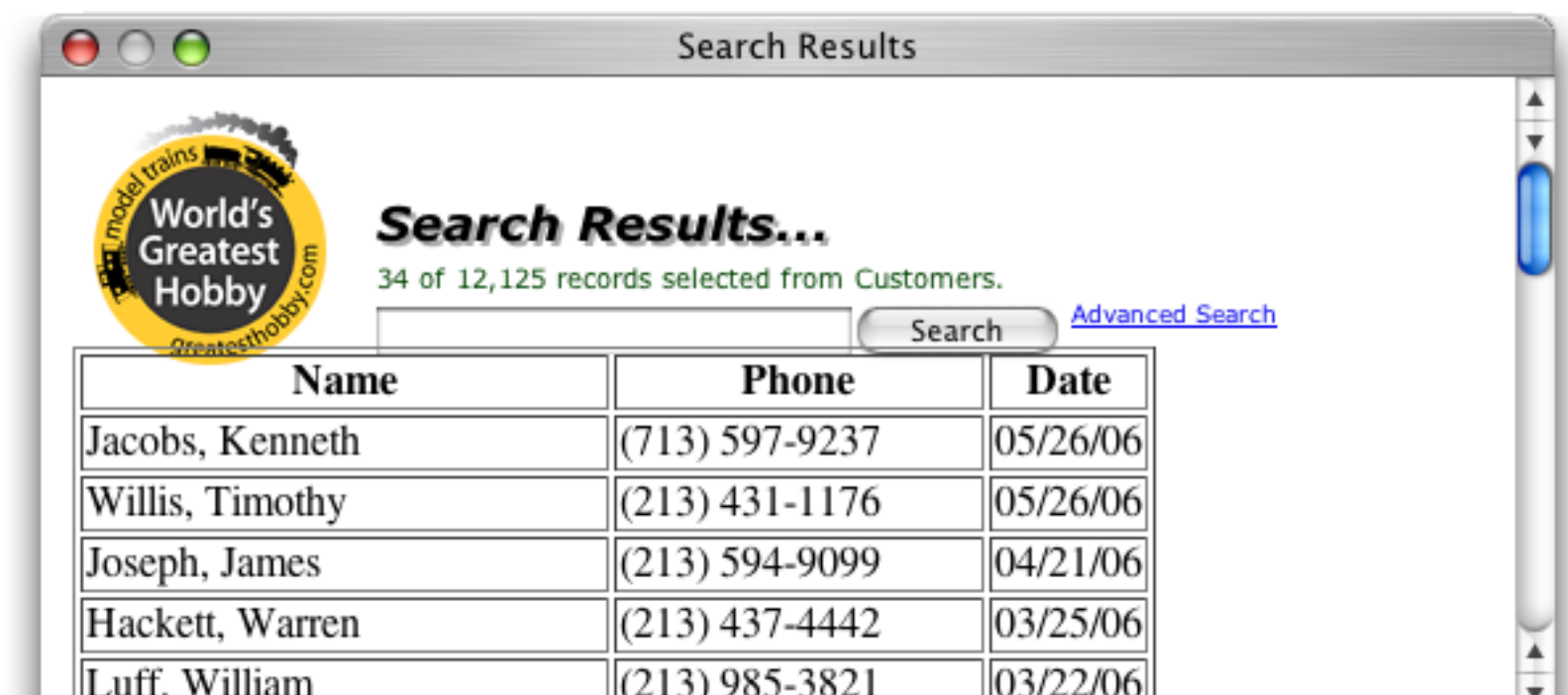
The **Header Form** option allows you to “stick” a form on top of the table. For example, suppose you’ve created a Panorama form that looks like this:



To use this as a table header, first render the form as an HTML page, as described in the last chapter (see “[Converting a Panorama Form into a Web Page Form](#)” on page 232). Then you can select this form from the pop-up list of potential header forms.



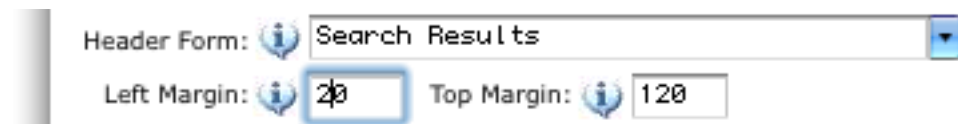
When you table is previewed in a web browser the [Search Results](#) form appears at the top of the page.



As you can see the server isn't always 100% successful at positioning the table relative to the form. In the next section you'll learn how to adjust the table position.

Table Margins

The **Left Margin** and **Top Margin** allow the table to be precisely positioned relative to the upper left hand corner of the page.






The margins are specified in pixels (1 pixel = 1/72th of an inch). The margins shown above move the table down and to the right.



You may need to "twiddle" with the numbers a bit to get the position you want. Just press the **Preview in Browser** button to see what the current margins look like.

Table Border

The Table Border is a numeric value that specifies the thickness (in pixels) of the border around each cell in the table. If the value is set to 0 then there is no border at all. Here are some examples of different border settings.

| Border | Example |
|--------|--|
| 0 |  |
| 1 |  |
| 2 |  |

When combined with the cell spacing, cell padding, and cell color options there are literally dozens of options for customizing the table appearance.

Cell Spacing

This option controls the space between each cell in the table (in pixels).



It's easier to see the cell spacing if the table cells have a colored background. In this example there are 4 pixels between each cell.

| Name | Phone | Date |
|-----------------|----------------|----------|
| Jacobs, Kenneth | (713) 597-9237 | 05/26/06 |
| Willis, Timothy | (213) 431-1176 | 05/26/06 |
| Joseph, James | (213) 594-9099 | 04/21/06 |
| Hackett, Warren | (213) 437-4442 | 03/25/06 |
| Luff, William | (213) 985-3821 | 03/22/06 |

Cell Padding

This option controls the space between the contents of each cell (text or image) and the edge of each cell (in pixels).

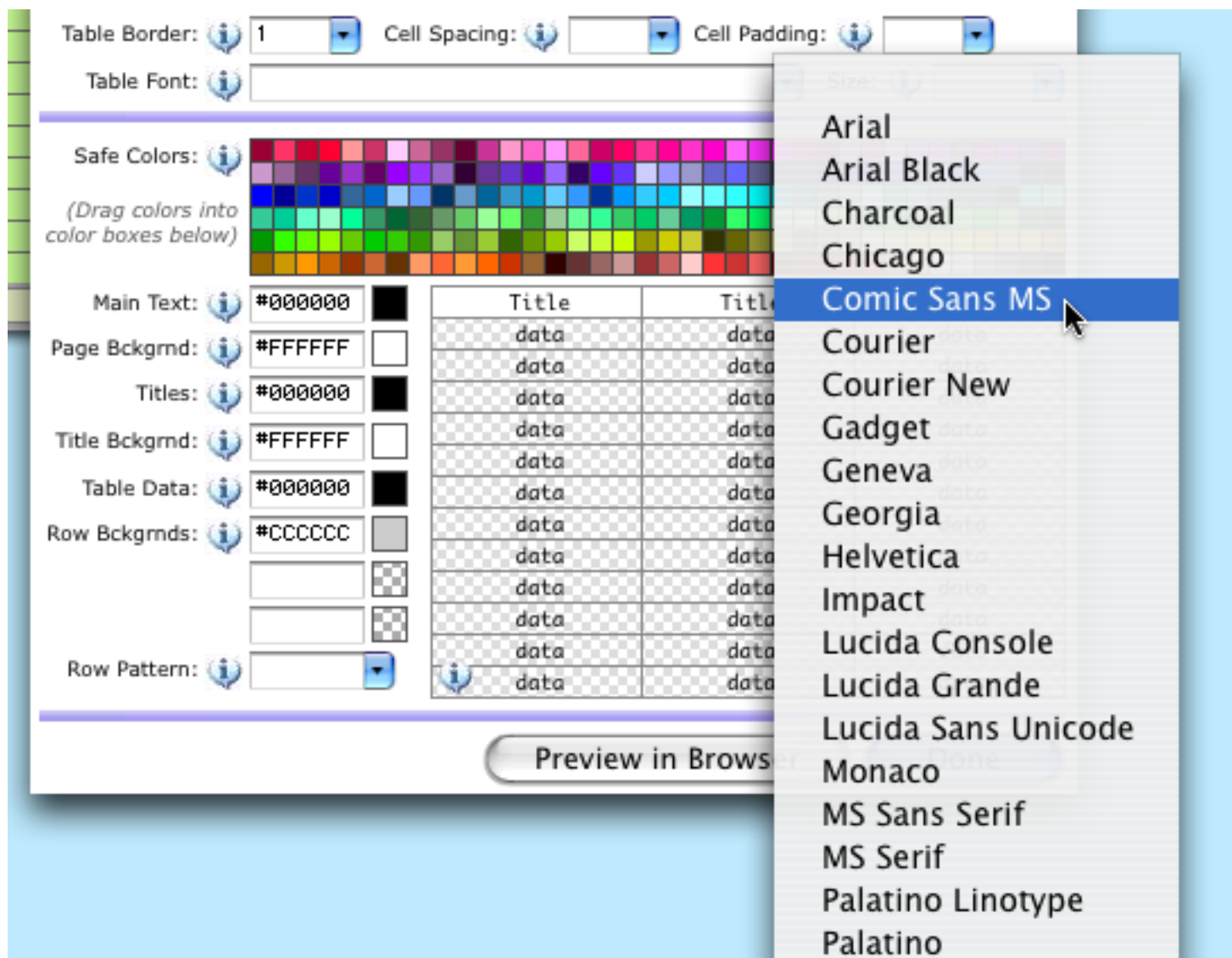


In this example there are 6 pixels of padding around each cell.

| Name | Phone | Date |
|-------------------|----------------|----------|
| Jacobs, Kenneth | (713) 597-9237 | 05/26/06 |
| Willis, Timothy | (213) 431-1176 | 05/26/06 |
| Joseph, James | (213) 594-9099 | 04/21/06 |
| Hackett, Warren | (213) 437-4442 | 03/25/06 |
| Luff, William | (213) 985-3821 | 03/22/06 |
| McLaughlin, Peter | (562) 439-2472 | 03/06/06 |

Table Font

This option specifies the font to use to display the table. (If left blank, the browser's default font will be used.) You can either type in the font name or choose from a pop-up menu listing the most common web fonts.

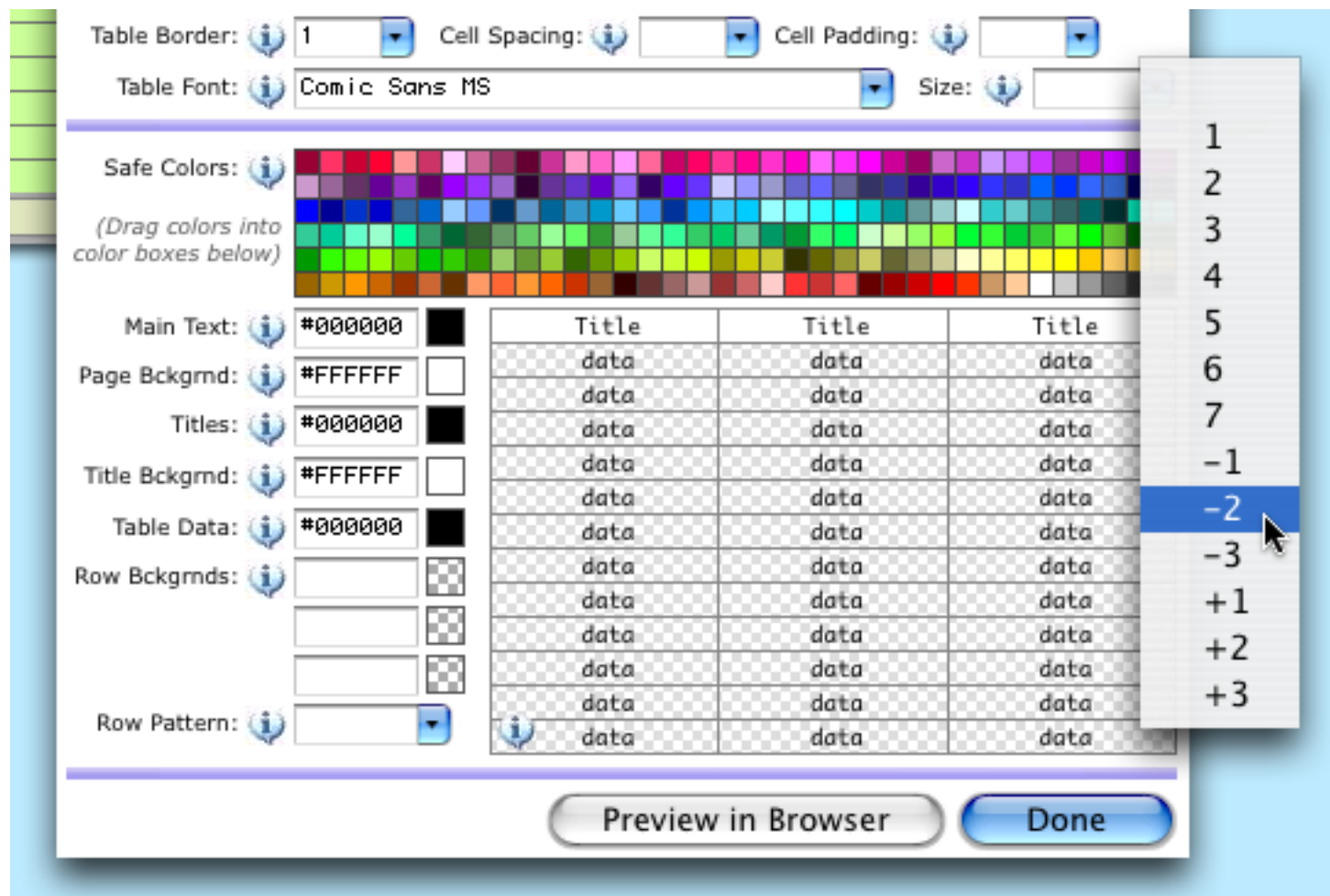


On your own computer you can use any font you want because you have complete control over what fonts are installed and available for use. A web page, however, is the opposite situation since you have zero control over what fonts may be available when the page is viewed. Because of this, it is usually best to pick from a restricted subset of fonts that are almost universally available. The fonts that are most commonly available are: *Arial*, *Comic Sans MS*, *Courier*, *Georgia*, *Helvetica*, *Times* and *Verdana*. Here's what the phone number table looks like using *Comic Sans*.

| Name | Phone | Date |
|------------------|----------------|----------|
| Sutton, Alfred | (703) 242-5125 | 06/26/06 |
| McDaniel, Thomas | (212) 310-4886 | 06/26/06 |
| Weber, Samuel | (201) 431-5137 | 06/26/06 |
| Rowe, Douglas | (609) 490-1972 | 06/26/06 |
| Swartz, Daniel | (212) 556-6657 | 06/26/06 |
| Alexander, Allen | (914) 967-7684 | 06/26/06 |
| Dannell, Herbert | (914) 793-5017 | 06/26/06 |

Text Size

This option specifies the size of the text to be used for the table. But take note — the size is not specified in pixels. Instead it is specified using a special HTML text size specification. Absolute values (1-7) specify a fixed font size from extremely small (1) to huge (7). Negative values specify sizes smaller than the text size in the rest of the page, for example -2 is two sizes smaller than normal font. Positive values specify sizes larger than the text size in the rest of the page, for example +2 is two sizes larger than normal font. You can type in the size or select from a pop-up menu of common sizes.

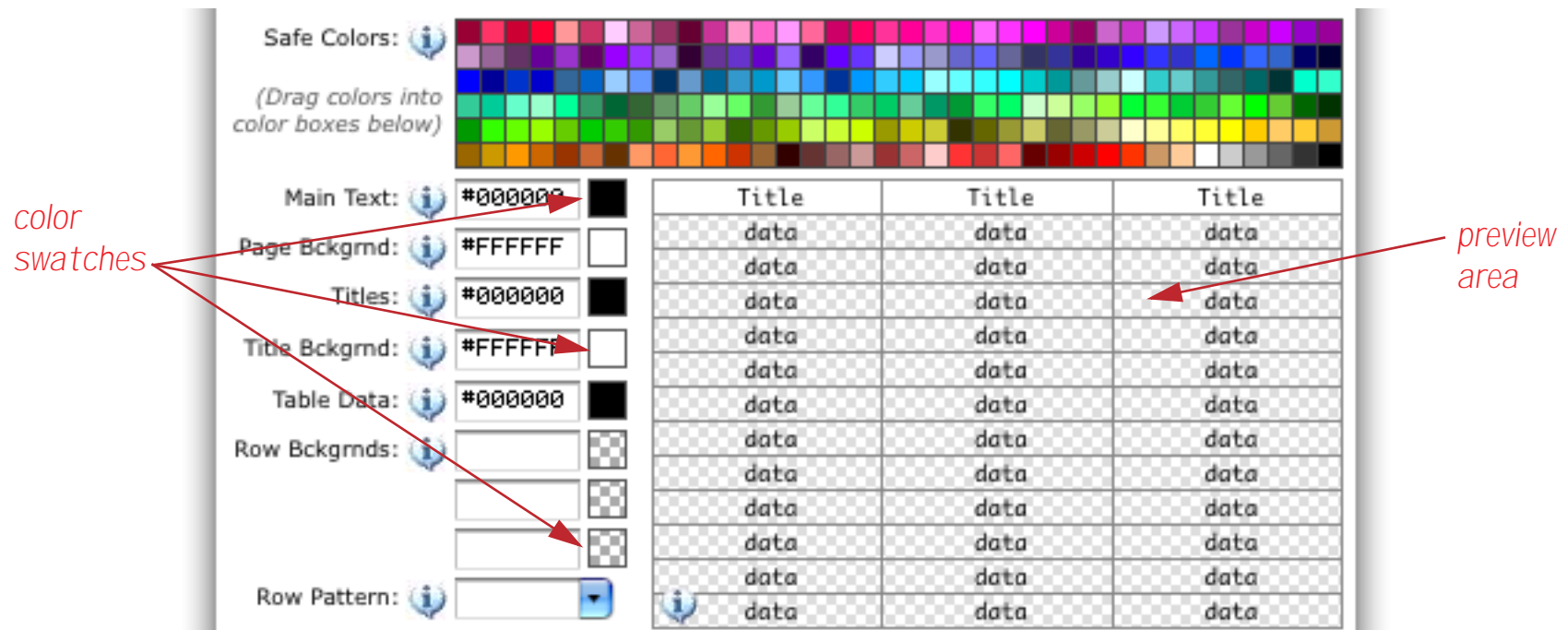


Here's the same table as the previous example but with the text displayed in a smaller (-2) size.

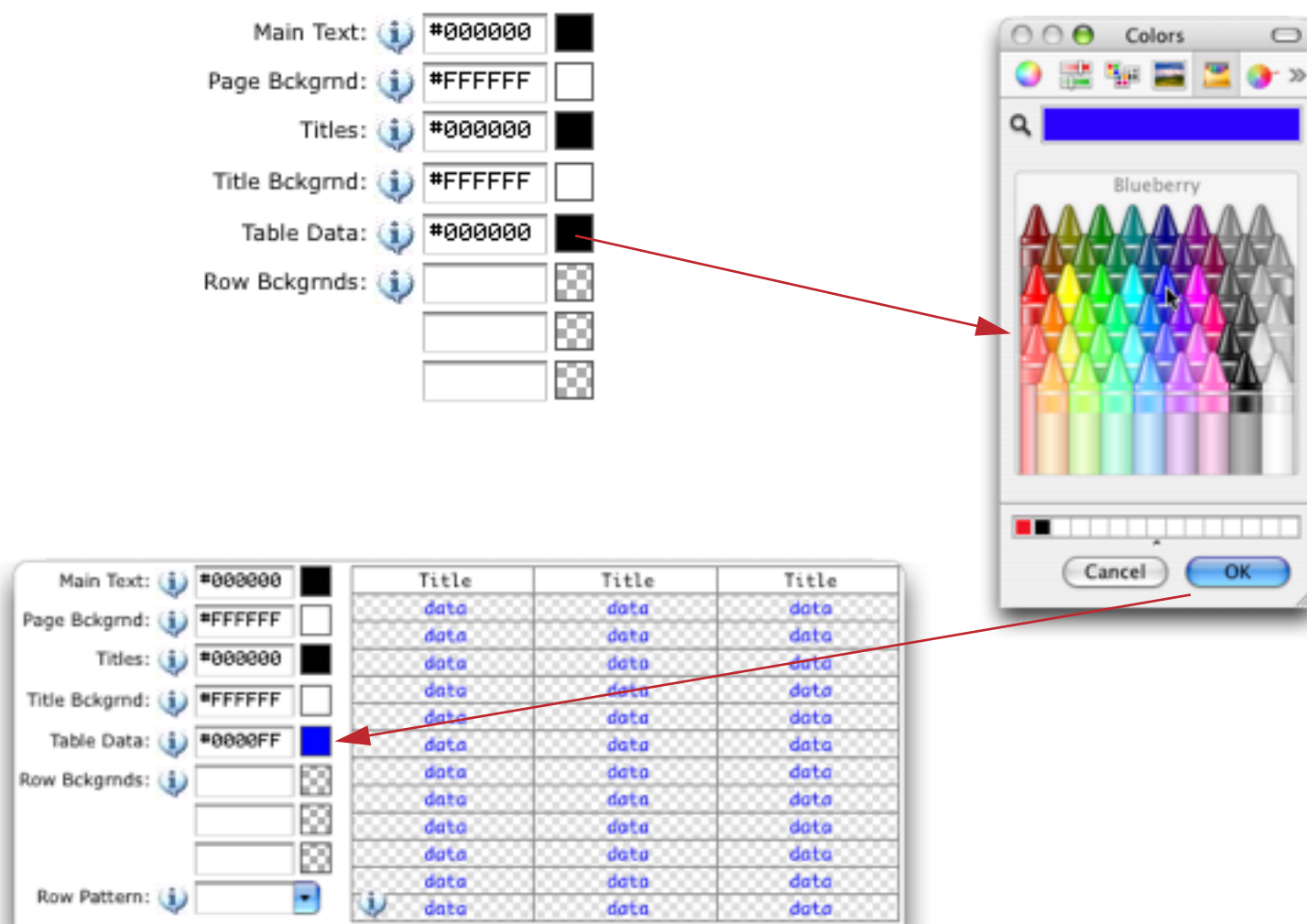
| Name | Phone | Date |
|------------------|----------------|----------|
| Sutton, Alfred | (703) 242-5125 | 06/26/06 |
| McDaniel, Thomas | (212) 310-4886 | 06/26/06 |
| Weber, Samuel | (201) 431-5137 | 06/26/06 |
| Rowe, Douglas | (609) 490-1972 | 06/26/06 |
| Swartz, Daniel | (212) 556-6657 | 06/26/06 |
| Alexander, Allen | (914) 967-7684 | 06/26/06 |
| Donnell, Herbert | (914) 793-5017 | 06/26/06 |
| Swift, Anthony | (215) 925-6820 | 06/26/06 |
| Booth, Thomas | (215) 942-1183 | 06/26/06 |
| Kuo, Joseph | (202) 806-4622 | 06/26/06 |
| Cline, Samuel | (301) 761-8958 | 06/26/06 |

Text and Background Colors

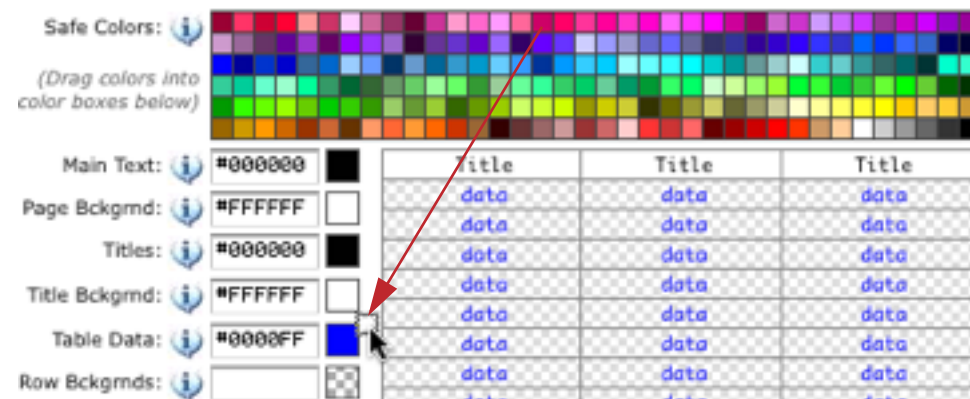
By default Panorama web tables use black text on a transparent background. The bottom half of the dialog allows you to specify any custom color combination you want. There are 8 color swatches on the left hand side, and a preview area that simulates the selected color combinations on the right (only the color selections are previewed, this area does not show borders, cell spacing, padding, font or size).



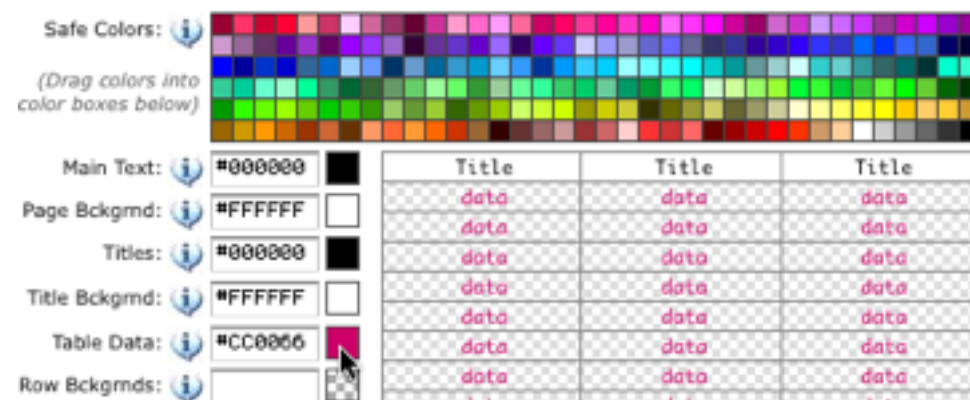
Color Selection Techniques. There are three ways to change a color swatch in this dialog. The simplest method is to simply click on a swatch. This opens the system's standard color picker dialog. When you select the color, the swatch and preview will update.



The second method is using the palette of 216 “web safe” colors. A few years ago, when most computers supported only 256 different colors, this list of 216 “web safe” colors was suggested as a standard. (The reason there are only 216 safe colors, not 256, is because the Microsoft and Mac operating system used 40 different “reserved” fixed system colors (about 20 each).) Since most computers can now display millions of colors it’s no longer all that important to use colors from this palette, but the palette is still available for your use. To use one of these colors, simply drag it from the palette to one of the eight swatches.



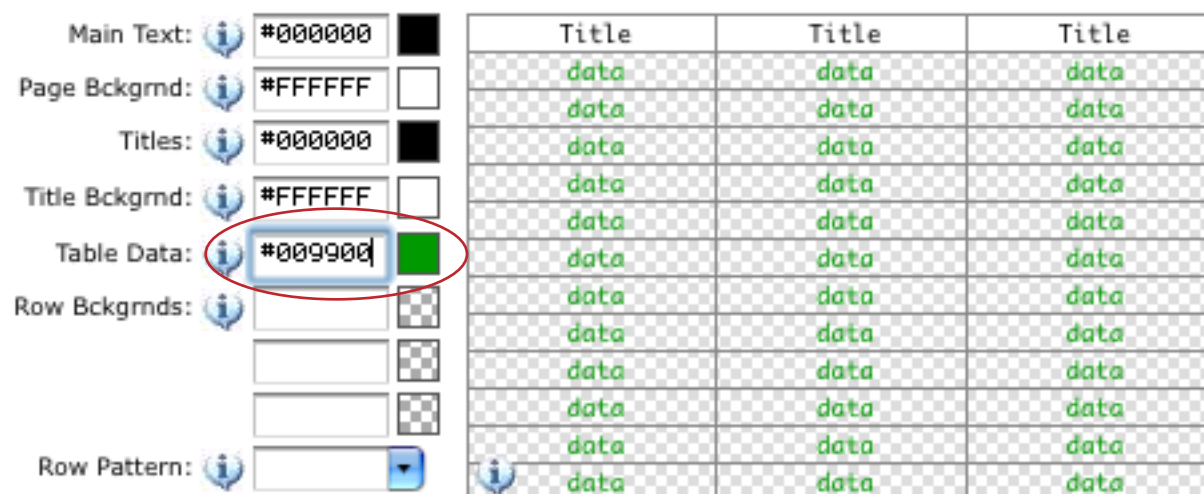
When the color is dropped onto the swatch the swatch changes to the selected color.



The third method for setting the color is to type in the color using the HTML color notation. HTML colors are defined using a hexadecimal notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex #00). The highest value is 255 (hex #FF). For example #0000FF represents pure blue, #00FF00 represents pure green, and #FF0000 represents pure red. Using this notation, you can specify millions of different colors. A few of these are shown in the table below.

| | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|
| | FFFFFF | FFCCFF | FF99FF | FF66FF | FF33FF | FF00FF | |
| | FFFACC | FFCCCC | FF99CC | FF66CC | FF33CC | FF00CC | |
| | FFFF99 | FFCC99 | FF9999 | FF6699 | FF3399 | FF0099 | |
| EEEEEE | FFFF66 | FFCC66 | FF9966 | FF6666 | FF3366 | FF0066 | 00FF00 |
| DDDDDD | FFFF33 | FFCC33 | FF9933 | FF6633 | FF3333 | FF0033 | 00EE00 |
| CCCCCC | FFFF00 | FFCC00 | FF9900 | FF6600 | FF3300 | FF0000 | 00DD00 |
| BBBBBB | CCFFFF | CCCCFF | CC99FF | CC66FF | CC33FF | CC00FF | 00CC00 |
| AAAAAA | CCFFCC | CCCCCC | CC99CC | CC66CC | CC33CC | CC00CC | 00BB00 |
| 999999 | CCFF99 | CCCC99 | CC9999 | CC6699 | CC3399 | CC0099 | 00AA00 |
| 888888 | CCFF66 | CCCC66 | CC9966 | CC6666 | CC3366 | CC0066 | 009900 |
| 777777 | CCFF33 | CCCC33 | CC9933 | CC6633 | CC3333 | CC0033 | 008800 |
| 666666 | CCFF00 | CCCC00 | CC9900 | CC6600 | CC3300 | CC0000 | 007700 |
| 555555 | 99FFFF | 99CCFF | 9999FF | 9966FF | 9933FF | 9900FF | 006600 |
| 444444 | 99FFCC | 99CCCC | 9999CC | 9966CC | 9933CC | 9900CC | 005500 |
| 333333 | 99FF99 | 99CC99 | 999999 | 996699 | 993399 | 990099 | 004400 |
| 222222 | 99FF66 | 99CC66 | 999966 | 996666 | 993366 | 990066 | 003300 |
| 111111 | 99FF33 | 99CC33 | 999933 | 996633 | 993333 | 990033 | 002200 |
| 000000 | 99FF00 | 99CC00 | 999900 | 996600 | 993300 | 990000 | 001100 |
| FF0000 | 66FFFF | 66CCFF | 6699FF | 6666FF | 6633FF | 6600FF | 0000FF |
| EE0000 | 66FFCC | 66CCCC | 6699CC | 6666CC | 6633CC | 6600CC | 0000EE |
| DD0000 | 66FF99 | 66CC99 | 669999 | 666699 | 663399 | 660099 | 0000DD |
| CC0000 | 66FF66 | 66CC66 | 669966 | 666666 | 663366 | 660066 | 0000CC |
| BB0000 | 66FF33 | 66CC33 | 669933 | 666633 | 663333 | 660033 | 0000BB |
| AA0000 | 66FF00 | 66CC00 | 669900 | 666600 | 663300 | 660000 | 0000AA |
| 990000 | 33FFFF | 33CCFF | 3399FF | 3366FF | 3333FF | 3300FF | 000099 |
| 880000 | 33FFCC | 33CCCC | 3399CC | 3366CC | 3333CC | 3300CC | 000088 |
| 770000 | 33FF99 | 33CC99 | 339999 | 336699 | 333399 | 330099 | 000077 |
| 660000 | 33FF66 | 33CC66 | 339966 | 336666 | 333366 | 330066 | 000066 |
| 550000 | 33FF33 | 33CC33 | 339933 | 336633 | 333333 | 330033 | 000055 |
| 440000 | 33FF00 | 33CC00 | 339900 | 336600 | 333300 | 330000 | 000044 |
| 330000 | 00FFFF | 00CCFF | 0099FF | 0066FF | 0033FF | 0000FF | 000033 |
| 220000 | 00FFCC | 00CCCC | 0099CC | 0066CC | 0033CC | 0000CC | 000022 |
| 110000 | 00FF99 | 00CC99 | 009999 | 006699 | 003399 | 000099 | 000011 |
| | 00FF66 | 00CC66 | 009966 | 006666 | 003366 | 000066 | |
| | 00FF33 | 00CC33 | 009933 | 006633 | 003333 | 000033 | |
| | 00FF00 | 00CC00 | 009900 | 006600 | 003300 | 000000 | |

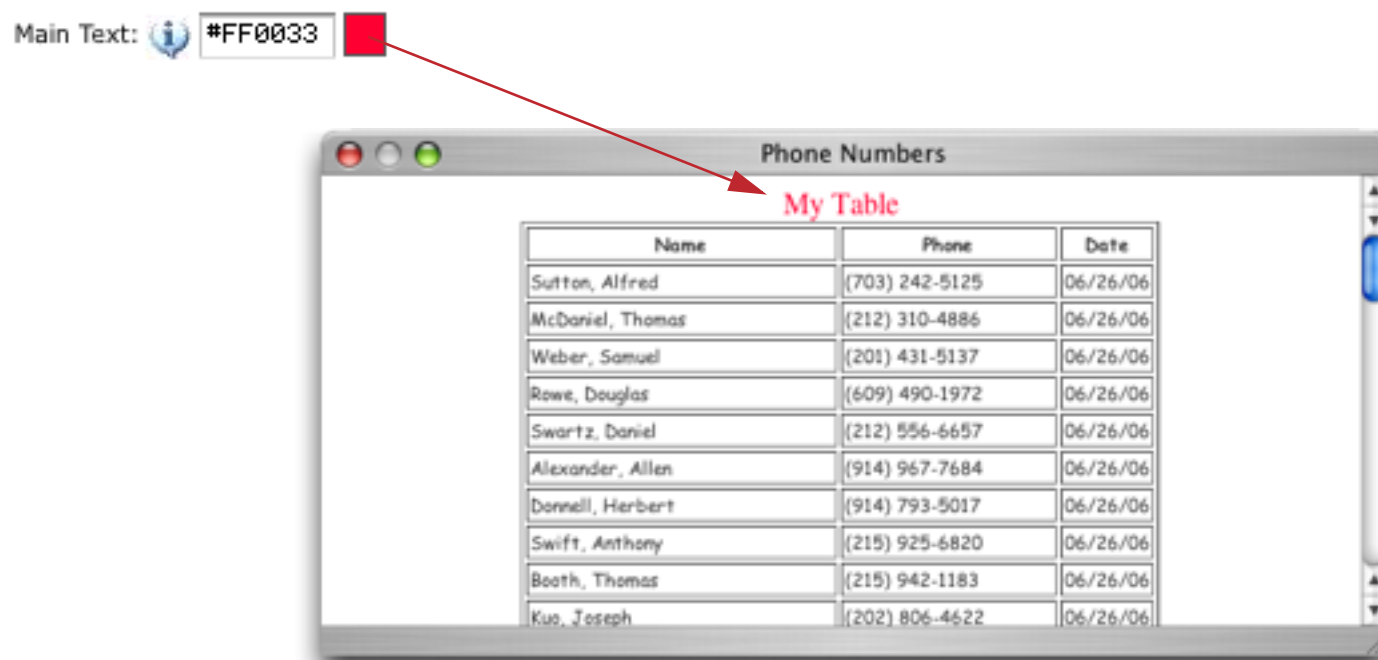
The illustration shows how the table data can be displayed in dark green by typing #009900 into the Table Data color.



The color will appear in the swatch and preview when you click on another area of the dialog.

You've probably noticed that some swatches and areas of the preview are shown with a light gray checkerboard pattern. This indicates that these colors have not been set and will be transparent.

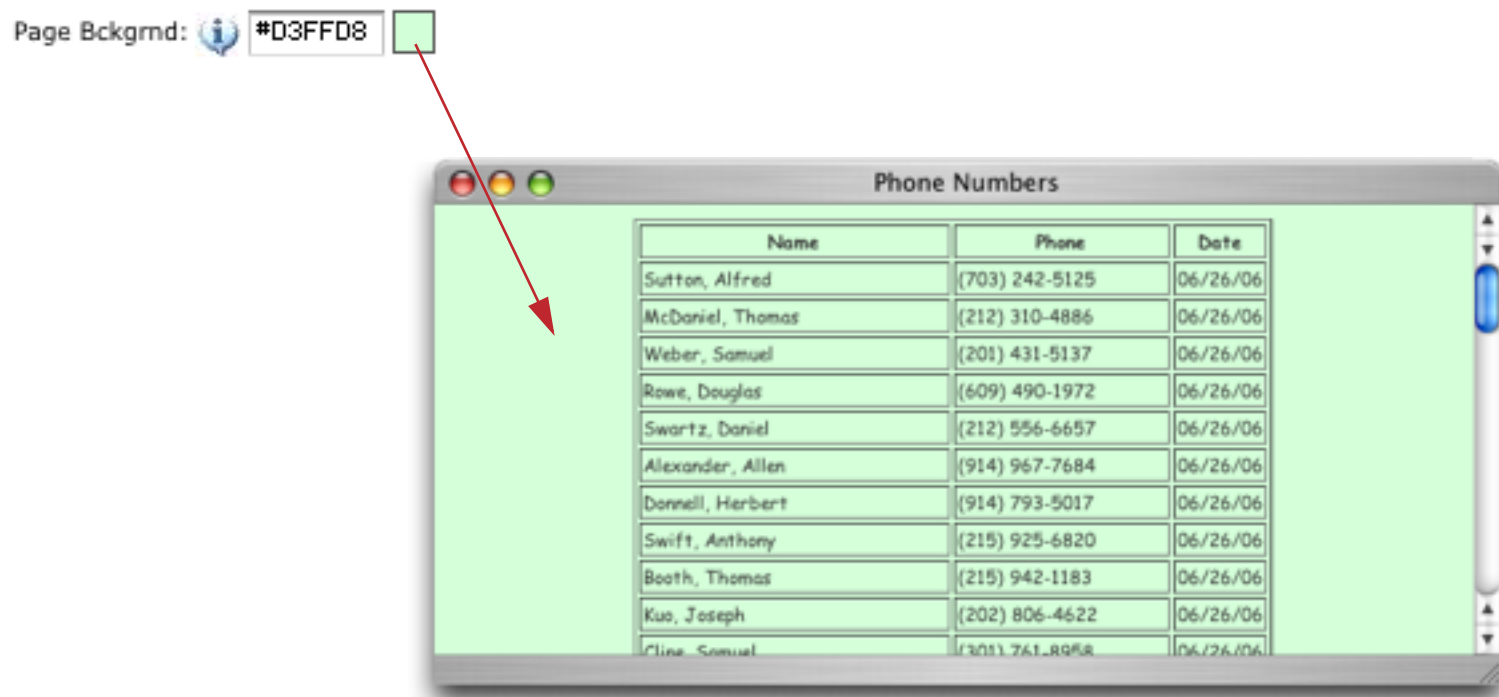
Main Text Color. This is the default text color for the entire page, including any text outside the table. Usually there isn't any text outside the table, but you can add extra text (headers, footers, etc.) using the **HTML Templates** dialog (see "[Customizing the table HTML \(advanced\)](#)" on page 315). Here's a simple example with an extra header at the top of the page.



Technical note: What this color actually does is change the `text` parameter in the `<body>` tag of the page.

```
<body bgcolor="#FFFFFF" text="#FF0033" link="0000FF" vlink="336666">
```

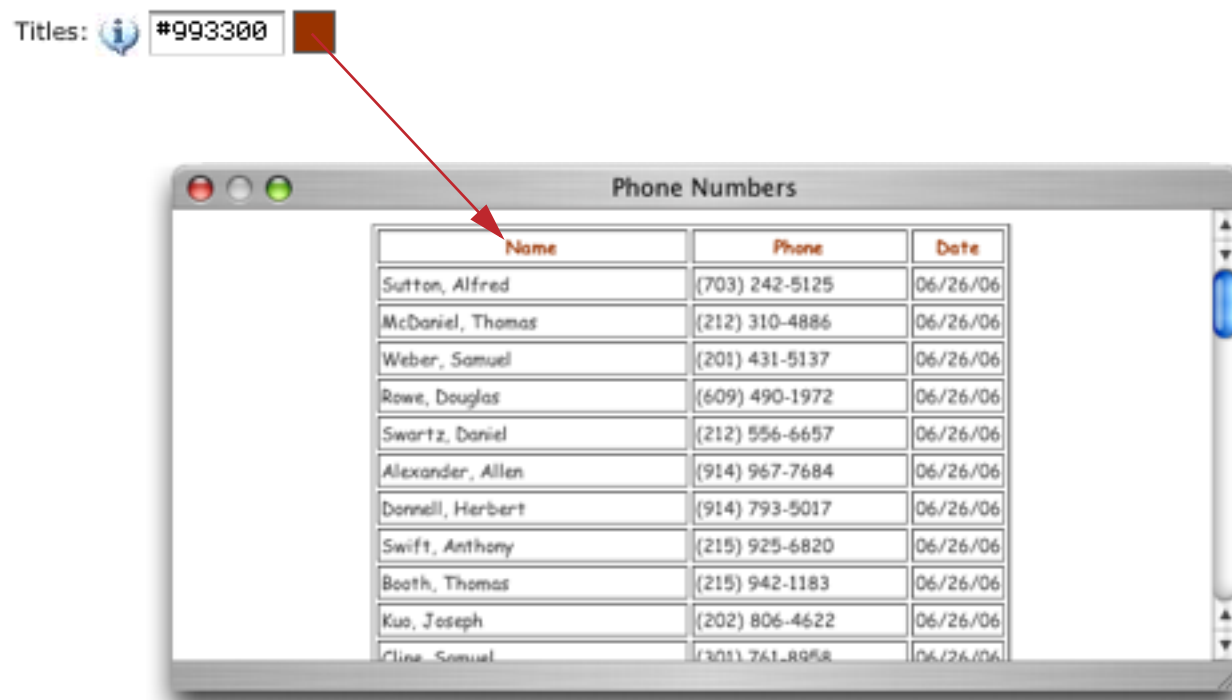

Page Background Color. This is the default background color for the entire page (not just the table). Here is a typical example.



Technical note: What this color actually does is change the `bgcolor` parameter in the `<body>` tag of the page.

```
<body bgcolor="#D3FFD8" text="#000000" link="#0000FF" vlink="#336666">
```

Title Color. This is the color for the column titles.



Title Background Color. This is the background color for the column titles. If you pick a dark background color then it's best to pick a light title color, or vice versa.

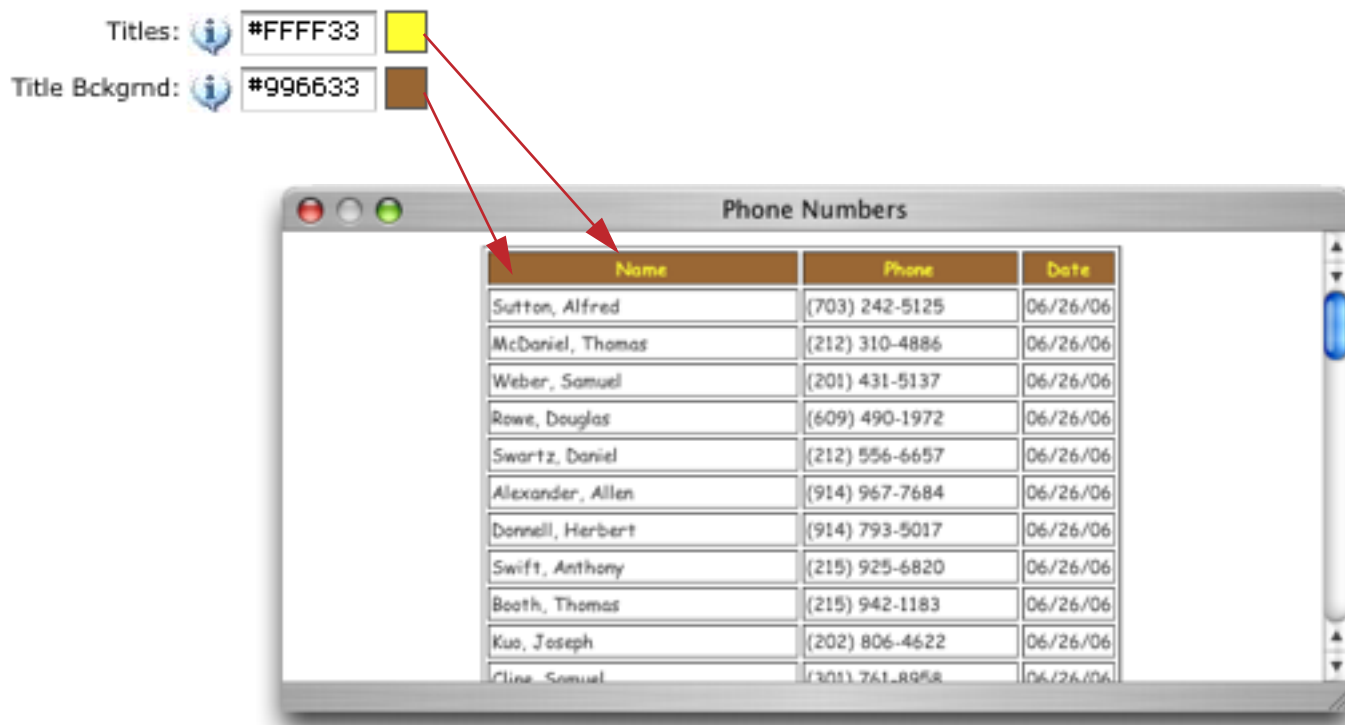
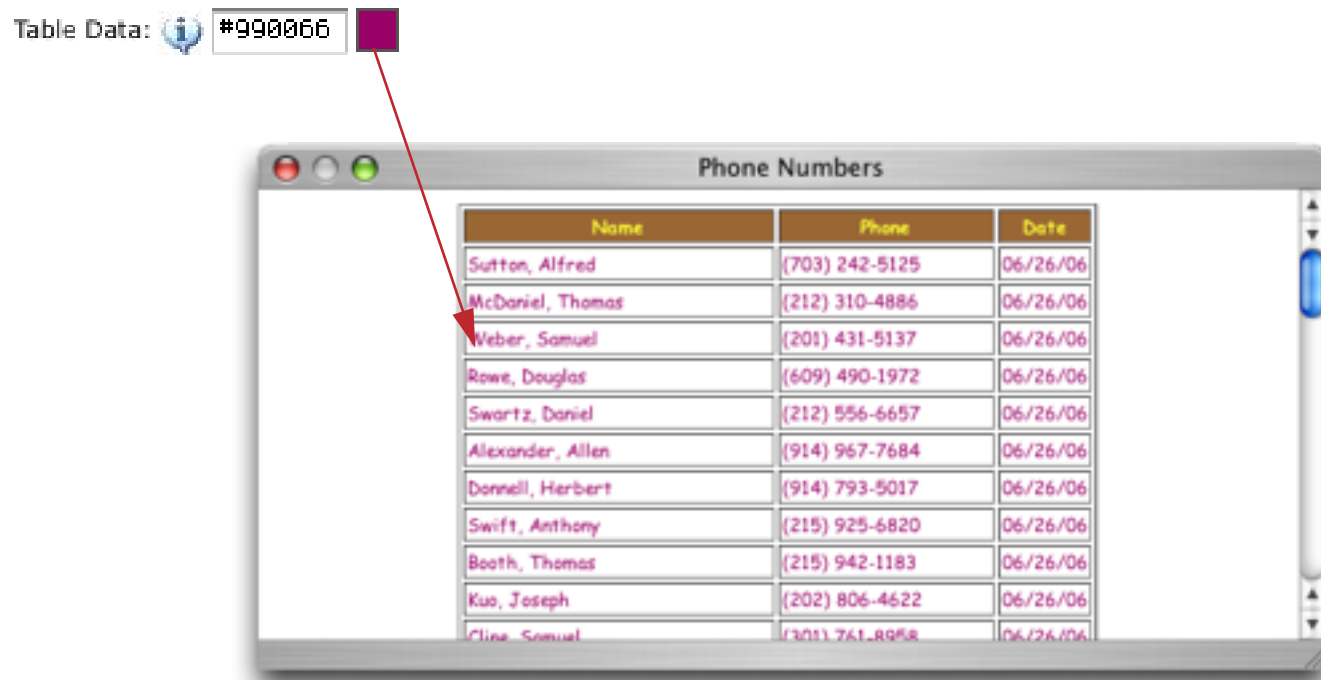




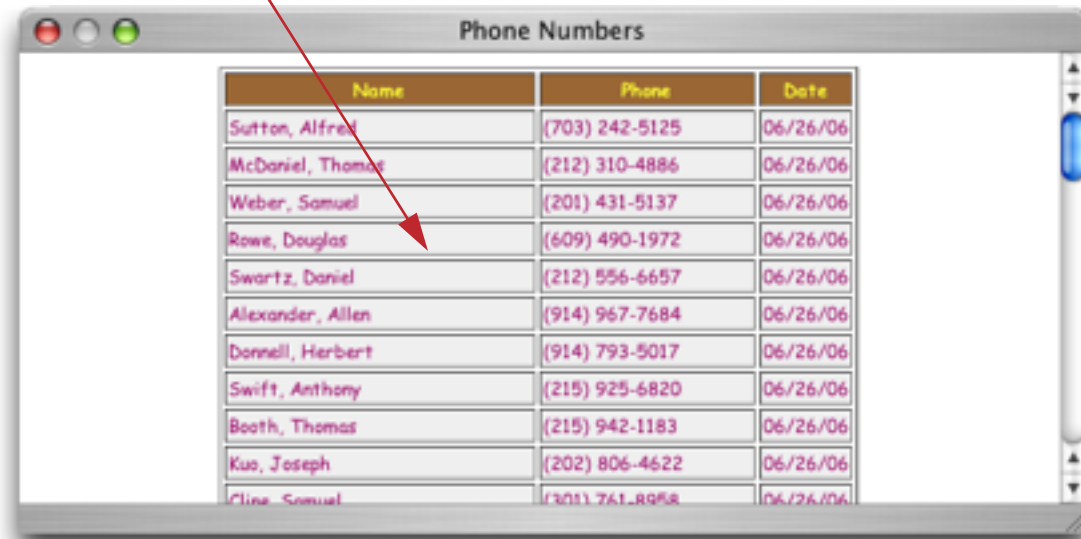


Table Data Color. This is the color for the text in the table.






Row Background Color. This is the background color for the body of the table. If you pick a dark background color then it's best to pick a light title color, or vice versa.

Table Data:  #990066 
 Row Bckgrnds:  #EEEEEE 



| Name | Phone | Date |
|------------------|----------------|----------|
| Sutton, Alfred | (703) 242-5125 | 06/26/06 |
| McDaniel, Thomas | (212) 310-4886 | 06/26/06 |
| Weber, Samuel | (201) 431-5137 | 06/26/06 |
| Rowe, Douglas | (609) 490-1972 | 06/26/06 |
| Swartz, Daniel | (212) 556-6657 | 06/26/06 |
| Alexander, Allen | (914) 967-7684 | 06/26/06 |
| Donnell, Herbert | (914) 793-5017 | 06/26/06 |
| Swift, Anthony | (215) 925-6820 | 06/26/06 |
| Booth, Thomas | (215) 942-1183 | 06/26/06 |
| Kuo, Joseph | (202) 806-4622 | 06/26/06 |
| Cline, Samuel | (301) 761-8958 | 06/26/06 |

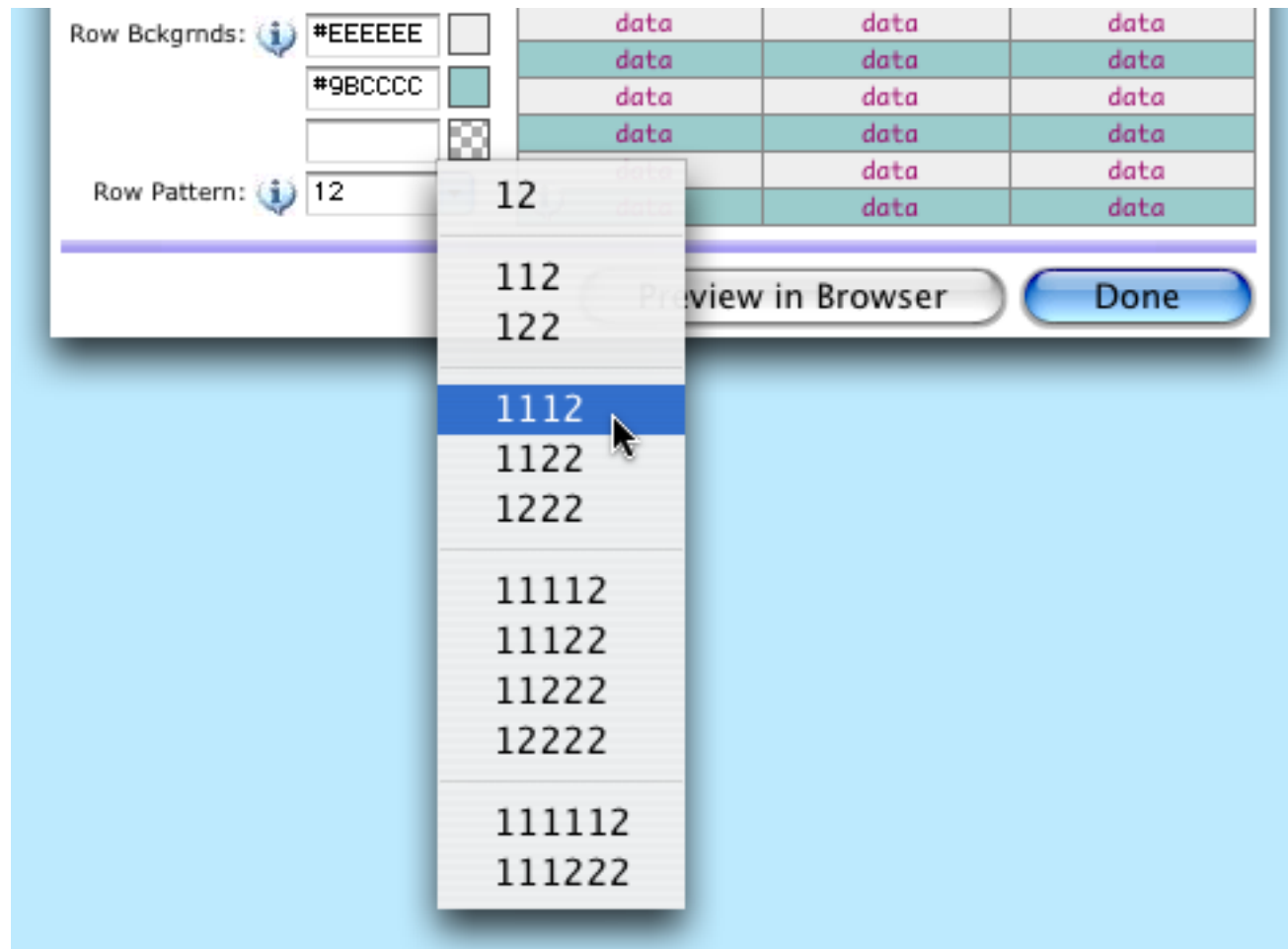
Multiple Background Colors. You can actually specify up to three **row background colors**. If you specify more than one background color, the server will automatically alternate them on successive rows.

Row Bckgrnds:  #EEEEEE 
 #9BCCCC 



| Name | Phone | Date |
|------------------|----------------|----------|
| Sutton, Alfred | (703) 242-5125 | 06/26/06 |
| McDaniel, Thomas | (212) 310-4886 | 06/26/06 |
| Weber, Samuel | (201) 431-5137 | 06/26/06 |
| Rowe, Douglas | (609) 490-1972 | 06/26/06 |
| Swartz, Daniel | (212) 556-6657 | 06/26/06 |
| Alexander, Allen | (914) 967-7684 | 06/26/06 |
| Donnell, Herbert | (914) 793-5017 | 06/26/06 |
| Swift, Anthony | (215) 925-6820 | 06/26/06 |
| Booth, Thomas | (215) 942-1183 | 06/26/06 |
| Kuo, Joseph | (202) 806-4622 | 06/26/06 |
| Cline, Samuel | (301) 761-8958 | 06/26/06 |

The default pattern is for the colors to alternate every other row — 12121212. The row pattern allows you to change the way the colors alternate. You can either type in a pattern, for example 11221122, or you can select from common patterns from the pop-up menu.



With this pattern, the table will have a grey background for three rows (color 1), then a blue background (color 2), then three grey rows, one blue, etc.

| Name | Phone | Date |
|------------------|----------------|----------|
| Sutton, Alfred | (703) 242-5125 | 06/26/06 |
| McDaniel, Thomas | (212) 310-4886 | 06/26/06 |
| Weber, Samuel | (201) 431-5137 | 06/26/06 |
| Rowe, Douglas | (609) 490-1972 | 06/26/06 |
| Swartz, Daniel | (212) 556-6657 | 06/26/06 |
| Alexander, Allen | (914) 967-7684 | 06/26/06 |
| Dannell, Herbert | (914) 793-5017 | 06/26/06 |
| Swift, Anthony | (215) 925-6820 | 06/26/06 |
| Booth, Thomas | (215) 942-1183 | 06/26/06 |
| Kuo, Joseph | (202) 806-4622 | 06/26/06 |
| Cline, Samuel | (201) 761-8958 | 06/26/06 |

Don't forget that you can adjust the font, borders and cell spacing with the other options described earlier in this section.



| Name | Phone | Date |
|------------------|----------------|----------|
| Sutton, Alfred | (703) 242-5125 | 06/26/06 |
| McDaniel, Thomas | (212) 310-4886 | 06/26/06 |
| Weber, Samuel | (201) 431-5137 | 06/26/06 |
| Rowe, Douglas | (609) 490-1972 | 06/26/06 |
| Swartz, Daniel | (212) 556-6657 | 06/26/06 |
| Alexander, Allen | (914) 967-7684 | 06/26/06 |
| Donnell, Herbert | (914) 793-5017 | 06/26/06 |
| Swift, Anthony | (215) 925-6820 | 06/26/06 |
| Booth, Thomas | (215) 942-1183 | 06/26/06 |
| Kuo, Joseph | (202) 806-4622 | 06/26/06 |
| Cline, Samuel | (301) 761-8958 | 06/26/06 |
| Wylie, Phillip | (301) 831-1370 | 06/26/06 |
| Rumsey, Aaron | (703) 671-5889 | 06/26/06 |

When you are finished customizing the appearance, press the **Done** button to close the dialog. The changes are automatically saved to the template.

Linking a Table with a Query Form

Once a web table is set up, it can be displayed by a custom web procedure using the `htmldatatable` statement (see “[Generating an HTML Table or List from Multiple Records](#)” on page 413). However, the most common method for displaying a table is in response to a search form (a form that uses the standard query action). Setting up a search form is described in detail in “[Standard Form Action — QUERY](#)” on page 249, but we'll review the basics here.

To start you'll need a Panorama form. This form should have one or more database fields, or one field set up to edit a variable named `SearchAllFields`.



World's Greatest Hobby
greatesthobby.com

Phone Search

Enter the data you want to search for:

First Name:

Last Name:

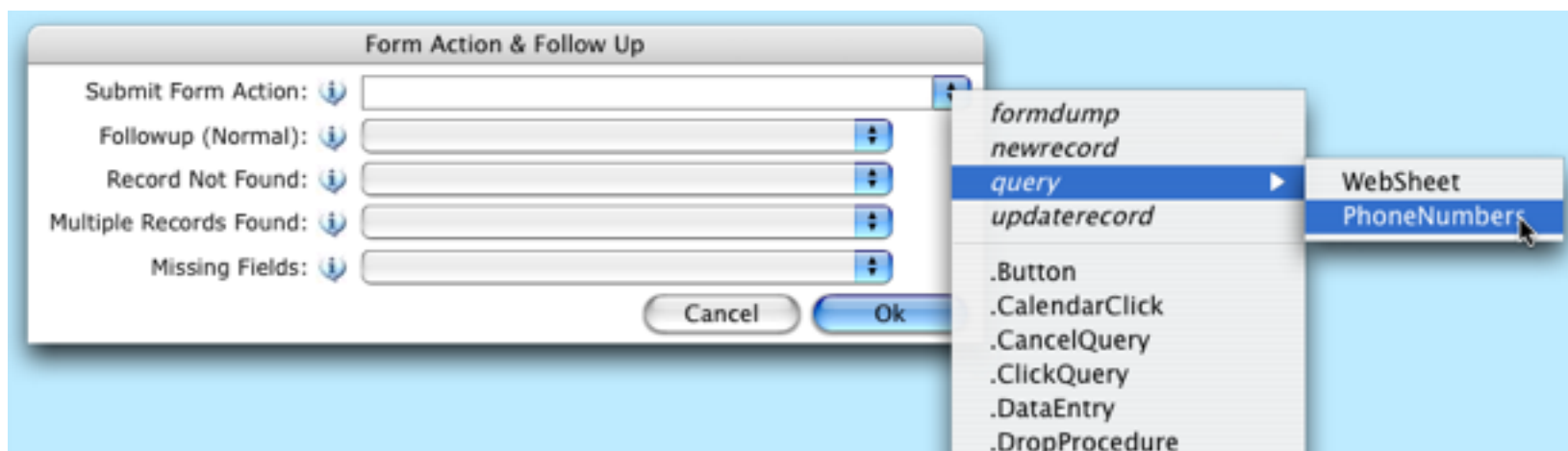
Address:

City: State: Zip:

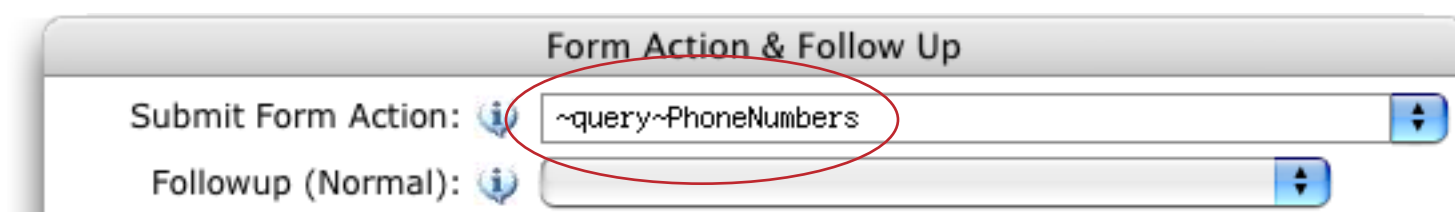
Phone:

Lucida Grande 12pt

In the **Form Action & Follow Up** dialog (in the **Web** submenu of the **Setup** menu) use the pop-up menu to specify the web sheet from the query submenu.



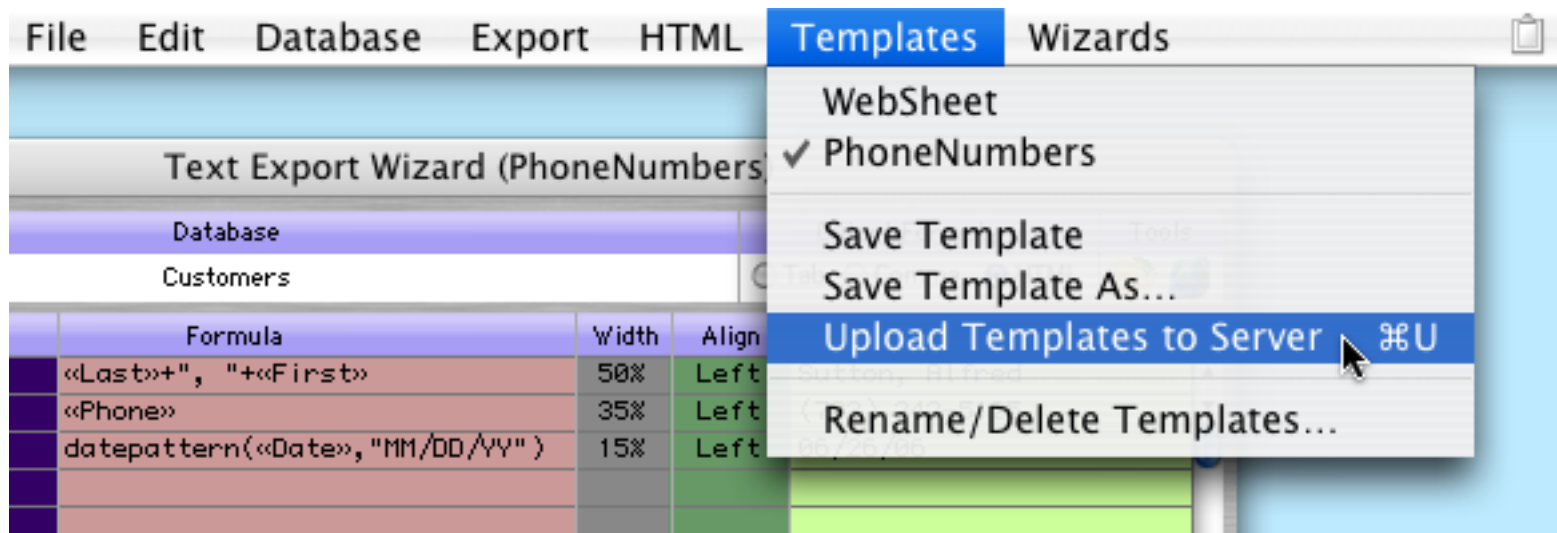
The submit form action should look something like this:



Press the **Ok** button and then upload the form to the server (see “[Updating a Web Form/Adding a new Web Form](#)” on page 208).

Uploading a Table to the Server

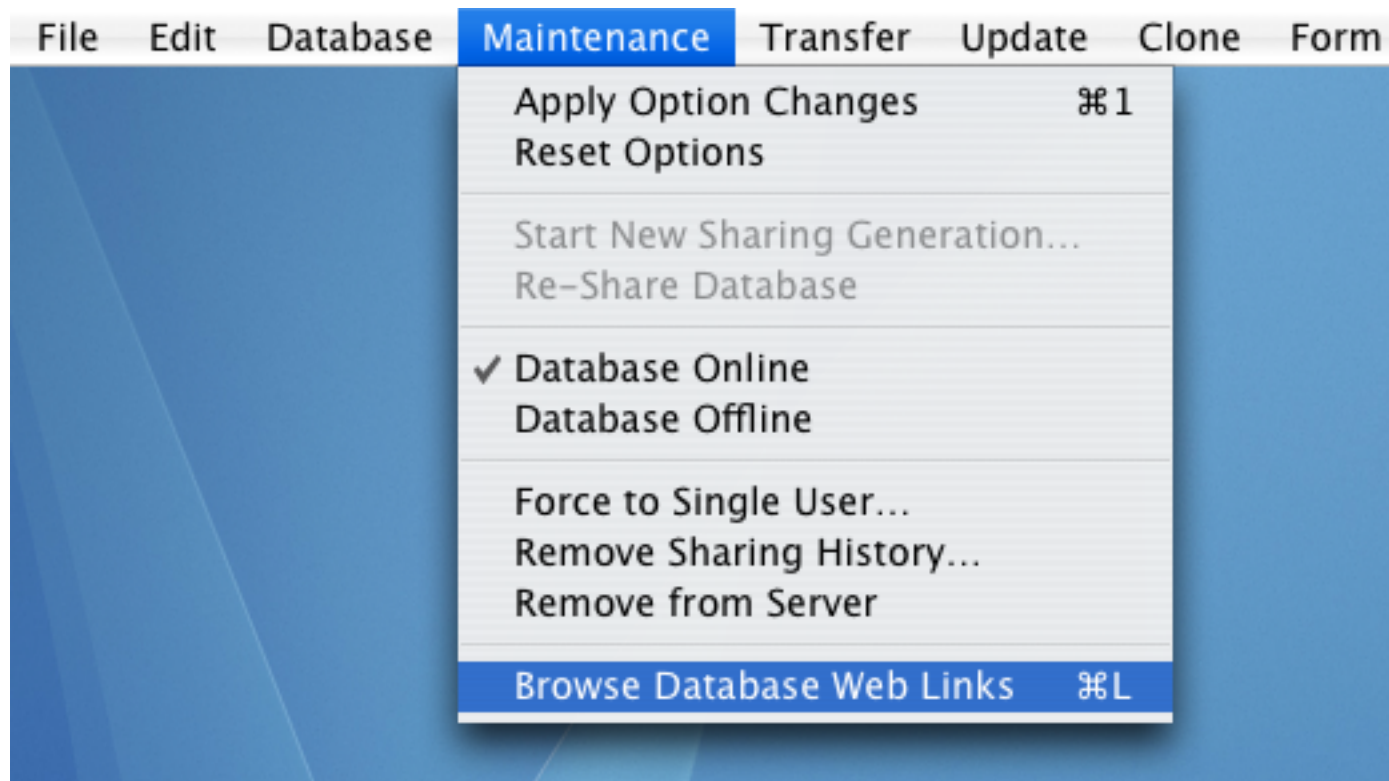
Once a web table has been set up (or modified) it must be uploaded to the server. Usually the easiest way to do that is with the **Upload Templates to Server** command in the **Text Export Wizard’s Templates** menu. (Note: This command uploads all of the templates to the server, not just the current one.)



You can also upload web table templates using the Maintenance menu of the **Database Sharing Options** wizard (see “[Updating a Table/Adding a new Web Table Template](#)” on page 210).

Testing the Query and Table

To test your new web table you could simply open your web browser and type in the url for the query form you just created. An easier method, however, is to select the **Browse Database Web Links** from the **Database Sharing Options** wizard's **Maintenance** menu.



This opens your web browser and displays a page listing all of the web forms and procedures in the database (if this page doesn't appear, see "[Debugging Web Link Page Problems](#)" on page 207).



As circled above, click on the name of the new query form you just created (in the **Forms** column).

Phone Search

World's Greatest Hobby
greatesthobby.com

Enter the data you want to search for:

First Name:

Last Name:

Address:

City: State: Zip:

Phone:

Search Reset

Fill in one or more fields to search for, then press the **Search** button. The table will appear.

Panorama Database Query:Customers/query

http://192.168.1.52/cgi-bin/Panorama.cgi?Custc Google

| Name | Phone | Date |
|--------------------|----------------|----------|
| Middleton, Charles | (562) 938-9850 | 06/09/00 |
| Welsgrau, Edward | (562) 428-4209 | 07/11/00 |
| Horner, Thomas | (562) 431-9792 | 08/17/01 |
| Dalton, Phillip | (516) 931-2202 | 09/27/01 |
| Anderson, Carl | (310) 439-5574 | 10/13/01 |
| Marks, John | (310) 927-8257 | 02/04/02 |
| Bergin, Robert | (516) 432-4252 | 05/23/02 |
| Knight, Herbert | (310) 438-5465 | 12/26/02 |
| Hartman, Joseph | (310) 439-7841 | 05/08/03 |
| Evans, Henry | (601) 867-7935 | 07/22/03 |
| Hulen, Russell | (310) 520-9859 | 08/13/03 |
| Day, Harry | (310) 599-7873 | 01/23/04 |
| Life, Steven | (310) 606-8886 | 02/20/04 |
| Hillman, Douglas | (310) 498-7564 | 02/25/04 |
| Haeler, Henry | (310) 431-6748 | 04/09/04 |

The table appears using whatever customization options (font, colors, spacing, etc.) that were set the last time you uploaded the table template. (The rest of this chapter will discuss more customization options for your web tables.)

Splitting a Long Table into Multiple Pages

By default a web table displays all of the selected records in the database. If there are hundreds or thousands of records selected this can make the table very unwieldy (and slow to load). Fortunately the Panorama Enterprise Edition Server can automatically split a large table into manageable pages, with links so you can navigate to different sections of the table. Here's an example of a long table that has been split into multiple pages. A search for cities containing **boulder** in the name has turned up 38 matching records. This table has been configured to display a maximum of 15 records so the table is split into three separate pages.

Loading "Search Results"

Search Results...
38 of 12,125 records selected from Customers.

[Advanced Search](#)

1 2 3 [NEXT](#)

| Name | City | Scale | Era | Railroad |
|-----------------------------------|------------------|-------|------------|-------------------------------|
| Joseph Irwin | Boulder, CO | N | Modern | Southern Pacific (SP) |
| Harold Donnell | Boulder, CO | HO | Transition | Southern Pacific (SP) |
| Dale McGuire | Boulder, CO | HO | Transition | Santa Fe (ATSF) |
| Craig Ott | Boulder, CO | HO | Transition | New York Central (NYC) |
| Harry Dickson | Boulder, CO | HO | Transition | Conrail |
| Peter Rodriguez | Boulder, CO | HO | Steam | Norfolk Southern (NS) |
| Lawrence Goodrich | Boulder, CO | G | Transition | Santa Fe (ATSF) |
| Marcus Silva | Boulder, CO | HO | Transition | Illinois Central (IC) |
| Michael Allen | Boulder, CO | G | Transition | Union Pacific (UP) |
| Joseph Cohen | Boulder, CO | O | Modern | Chicago & North Western (CNW) |
| Mark Fong | Boulder, CO | HO | Transition | Southern |
| George Derewlanko | Boulder, CO | HO | Transition | Nickel Plate (NKP) |
| Alfred Lamb | Boulder City, NV | N | Transition | Conrail |
| Douglas Dockery | Boulder, CO | HO | Transition | Illinois Central (IC) |
| Steven Hagen | Boulder, CO | HO | Modern | Southern Pacific (SP) |

1 2 3 [NEXT](#)

The Panorama server has automatically generated a “navigation bar” at the top and bottom of the table that allows you skip from page to page.

Search Results

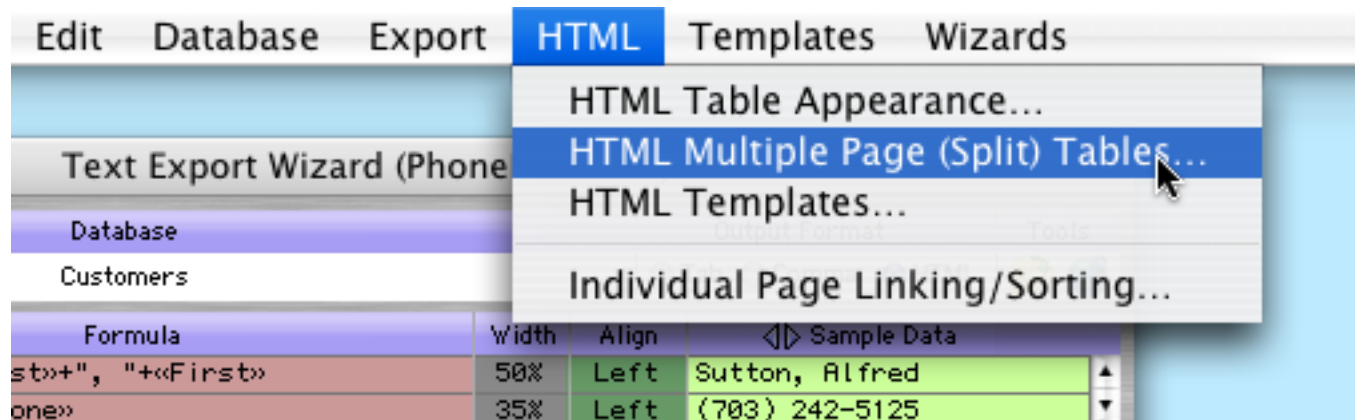
Loading "Search Results"

Loading "Search Results"

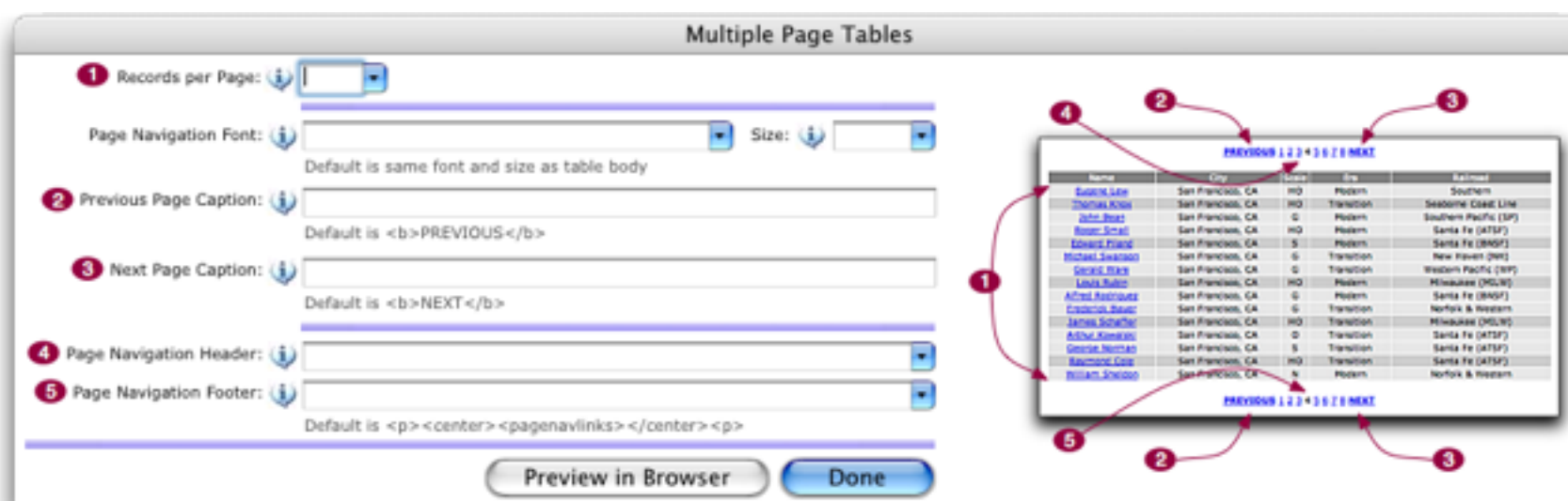
In the following sections you'll learn how to set up and customize the navigation bar.

The Multiple Page Table Dialog

To customize how your web table will be split into multiple pages open the **Text Export Wizard** and select the **HTML Multiple Page (Split) Tables** command from the **HTML** menu.

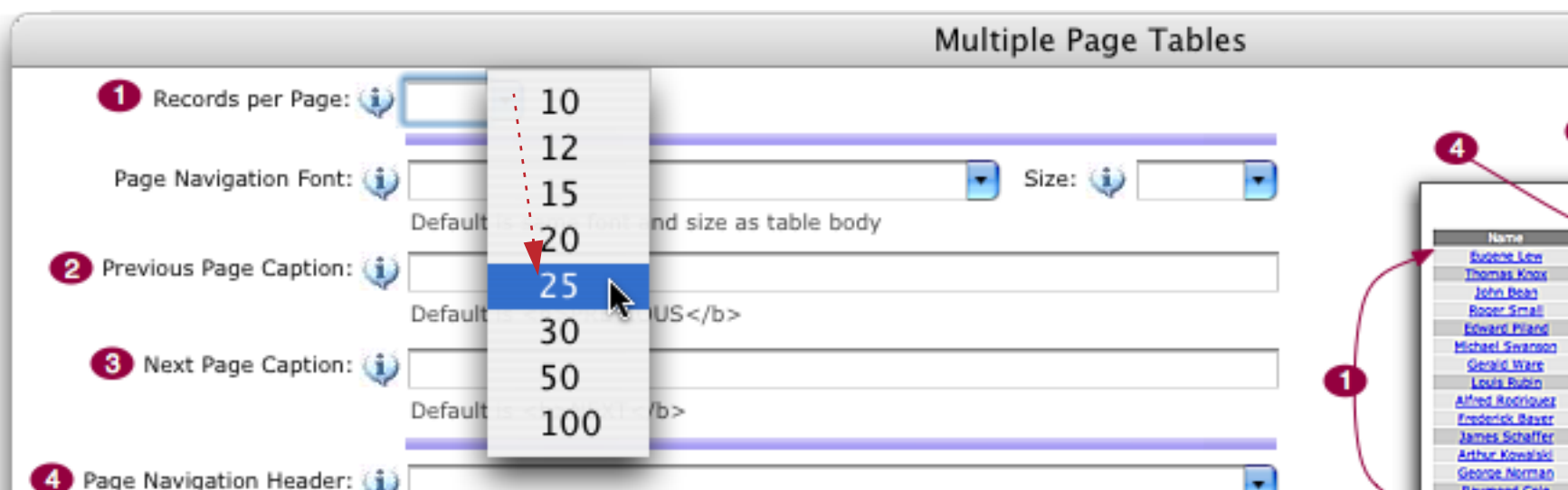


This very wide dialog has all of the options for splitting pages and automatic generation of the intra-page navigation bar.



After you finish setting up the options in this dialog, press **Done** and then upload the template to the server.

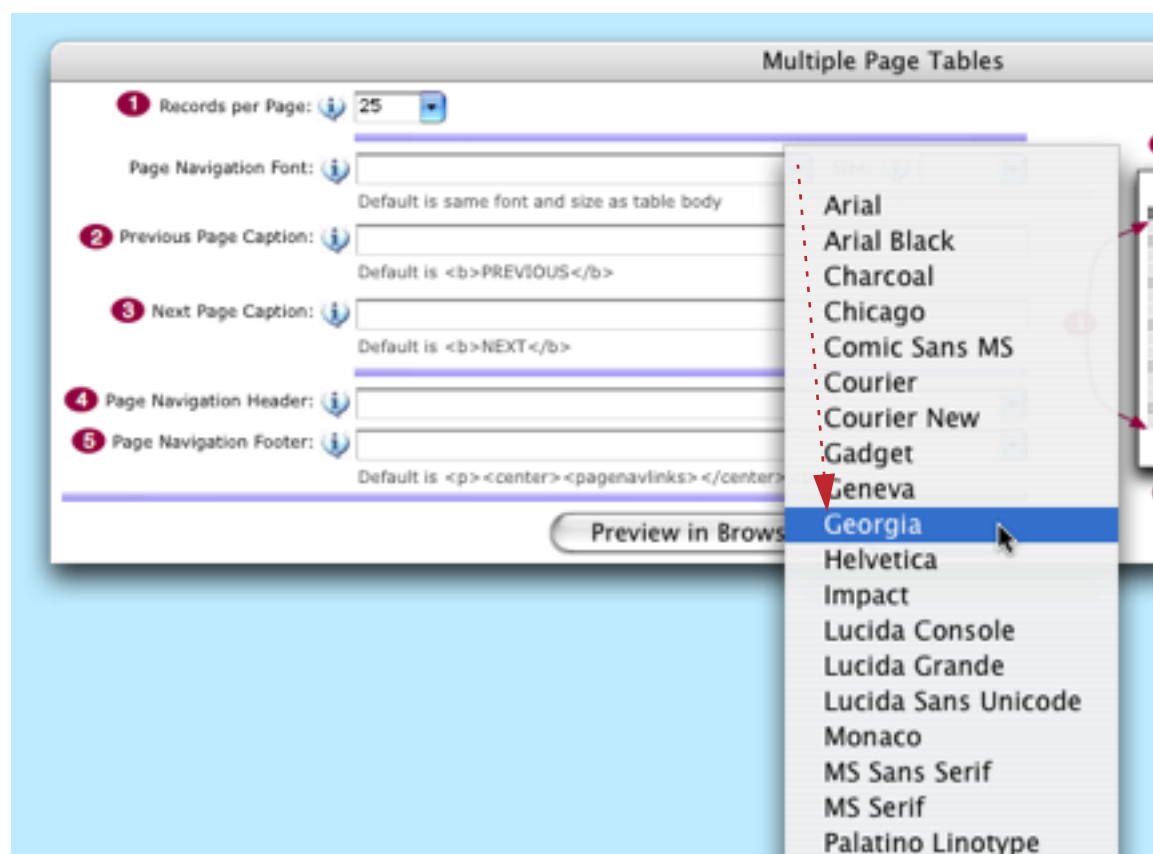
Records per Page. Set this to the maximum number of records that will be displayed in a single page. You can either type in a value or select from the pop-up menu.



If this option is left blank then the Panorama Server will not automatically split the table into multiple pages and the rest of the options in this dialog will be ignored (because no page navigation bar will be generated).

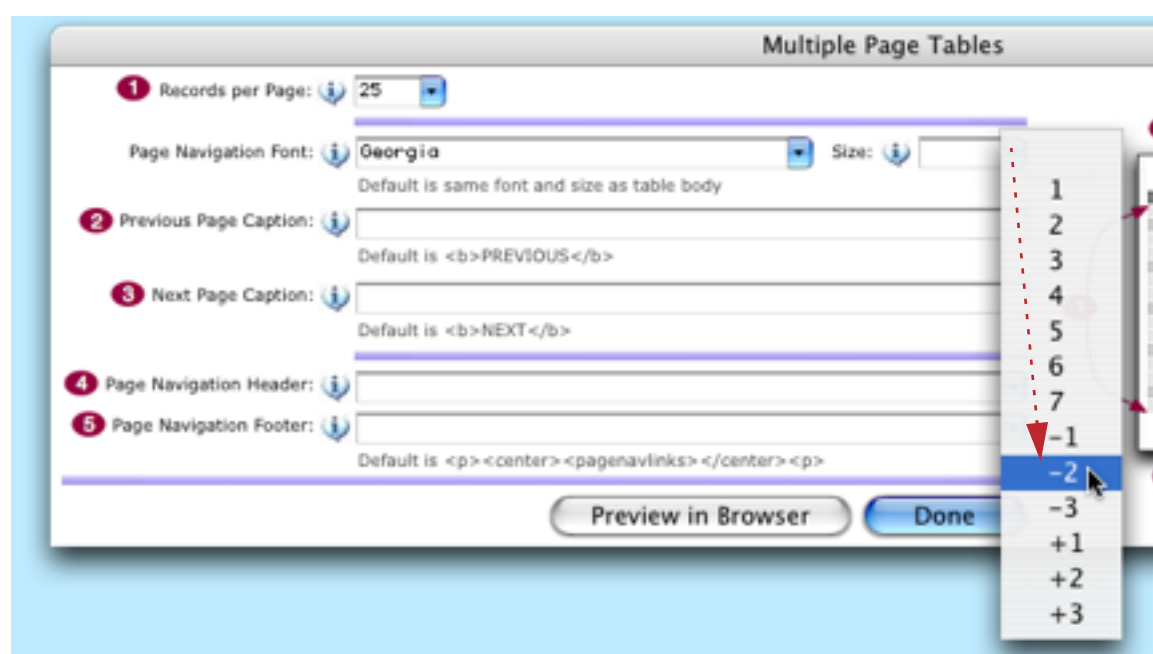
Note: If you specify a maximum number of records, we **highly recommend** that you also set up a specific **sort order** for this table (see “[Sort by](#)” on page 312). This will make sure that the same order is used for each page when navigating from page to page to page.

Page Navigation Font. This option specifies the font to be used for the page navigation bar. (If left blank, the browser's default font will be used.) You can either type in the font name or choose from a pop-up menu listing the most common web fonts.

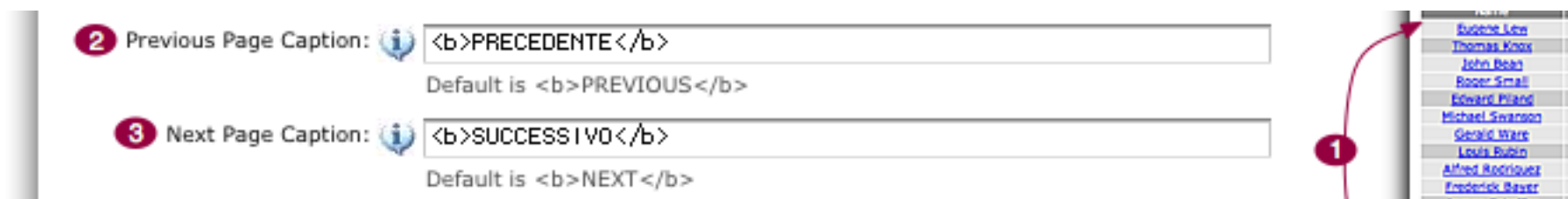


On your own computer you can use any font you want because you have complete control over what fonts are installed and available for use. A web page, however, is the opposite situation since you have zero control over what fonts may be available when the page is viewed. Because of this, it is usually best to pick from a restricted subset of fonts that are almost universally available. The fonts that are most commonly available are: *Arial*, *Comic Sans MS*, *Courier*, *Georgia*, *Helvetica*, *Times* and *Verdana*.

Text Size. This option specifies the size of the text to be used for the page navigation bar. But take note — the size is not specified in pixels. Instead it is specified using a special HTML text size specification. Absolute values (1-7) specify a fixed font size from extremely small (1) to huge (7). Negative values specify sizes smaller than the text size in the rest of the page, for example -2 is two sizes smaller than normal font. Positive values specify sizes larger than the text size in the rest of the page, for example +2 is two sizes larger than normal font. You can type in the size or select from a pop-up menu of common sizes.



Previous and Next Page Caption. The first and last links in the page navigation bar are usually [PREVIOUS](#) and [NEXT](#), displayed in bold. Use these options to change the text of these links. For example, if your web site is in Italian you might change these captions to [PRECEDENTE](#) and [SUCCESSIVO](#), as shown below.



You can put any HTML tags you want into the caption. If you want to display an image for these links (for example forward/back arrows simply use an `` tag for the caption.



Page Navigation Header and Footer. The page navigation bar is usually centered below the bottom of the table. However you can move the navigation bar to the top of the table, or include the navigation bar at both the top and bottom of the table. You can also customize the HTML tags that are used to separate the page navigation bar from the table and the rest of the page. You can either type in the header and/or footer or select from a pop-up menu of common formats.



The header and/or footer can include any HTML tags you want. Whatever you type in must, however, include the special tag `<pagenavlinks>` (this must be all in lower case). When the page is displayed the `<pagenavlinks>` tag will be replaced with the actual tags and text for page navigation bar. (If the `<pagenavlinks>` tag is missing then the navigation bar will be missing also!)

If both the header and footer are left blank then a default footer will be used. This default centers the page navigation bar below the table. If either the header or footer is filled in then the default won't be used. So if you want the navigation bar to appear above the table but not below, simply fill in the **Page Navigation Header** but leave the **Page Navigation Footer** blank.

Linking Individual Table Rows to a Detail Form

In many applications, a table is a jumping off point for additional pages. For example a phone list might link to individual address pages, while a product catalog table would probably have links to detail pages for each product listed. This section explains how to set up links from a table to individual forms.

The image illustrates a web application interface for signing up for a mailing list. It consists of a search results page and three detail forms.

Search Results Page: The page displays a table of search results. The table has columns for Name, City, State, Scale, Era, and Railroad. The data is as follows:

| Name | City | State | Scale | Era | Railroad |
|--------------------------------|-------------------|-------|-------|------------|-----------------------|
| Richard Chavez | Boulder, CO | CO | G | Transition | Santa Fe (BNSF) |
| Carl Hunter | Boulder Creek, CA | CA | N | Transition | Santa Fe (ATSF) |
| Ronald Mandell | Boulder, CO | CO | G | Transition | Southern Pacific (SP) |
| Daniel Day | Boulder, CO | CO | O | Transition | Illinois Central (IC) |
| Sean Shaw | Boulder, CO | CO | N | Transition | Norfolk Southern (NS) |
| Paul Barnes | Boulder, CO | CO | HO | Transition | Santa Fe (ATSF) |
| Andrew Gilman | Boulder, CO | CO | HO | Transition | Santa Fe (ATSF) |
| Darryl Moran | Boulder, CO | CO | N | Modern | Milwaukee (MLW) |

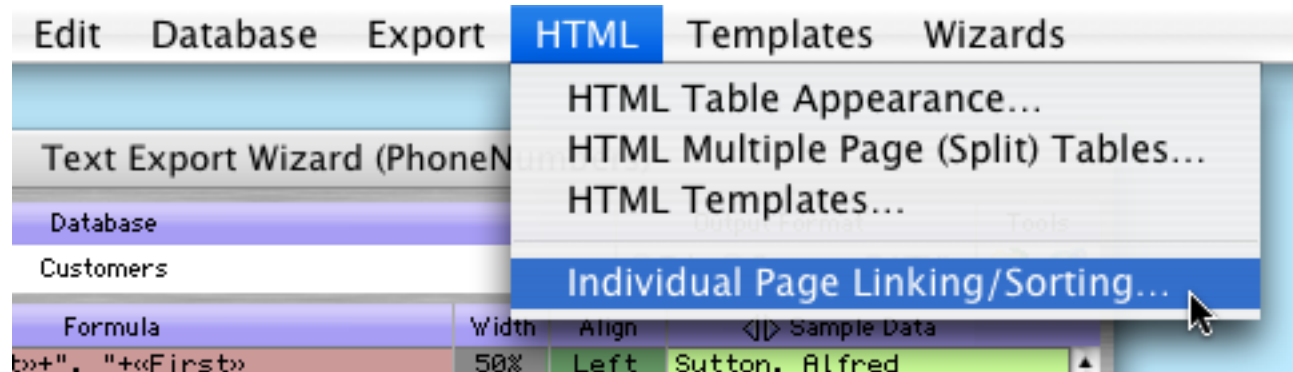
Detail Forms: Three detail forms are shown, each titled "Sign Up for Mailing List". Each form contains a "Correction" section with the following fields:

- First Name: [Text Input]
- Last Name: [Text Input]
- Address: [Text Input]
- City: [Text Input]
- State: [Dropdown Menu]
- Zip Code: [Text Input]
- Phone: [Text Input]
- Email: [Text Input]
- Scale: [Radio Buttons]
- Era: [Dropdown Menu]
- Railroad: [Dropdown Menu]

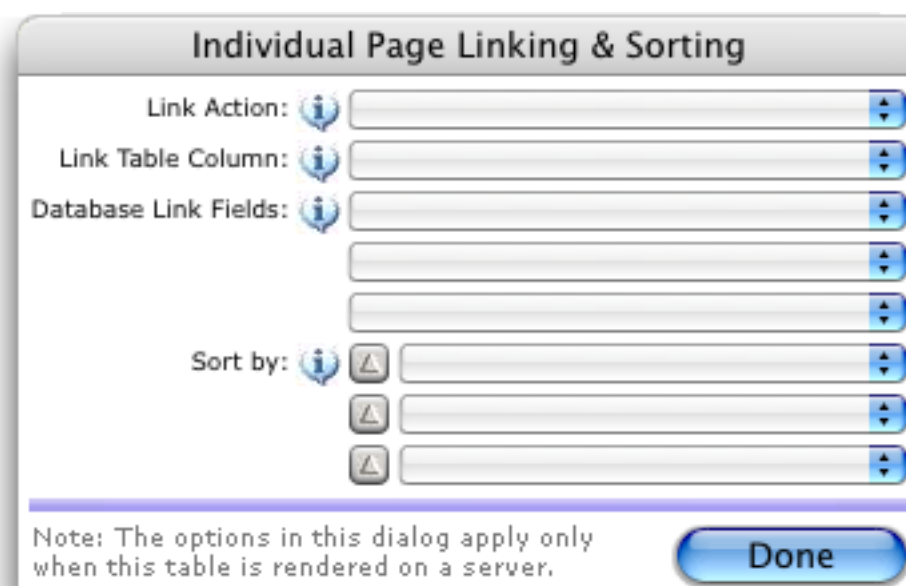
Red arrows indicate the flow from the search results table to the detail forms. The first arrow points from the "Richard Chavez" row to the first detail form. The second arrow points from the "Carl Hunter" row to the second detail form. The third arrow points from the "Darryl Moran" row to the third detail form.

The Individual Page Linking & Sorting dialog

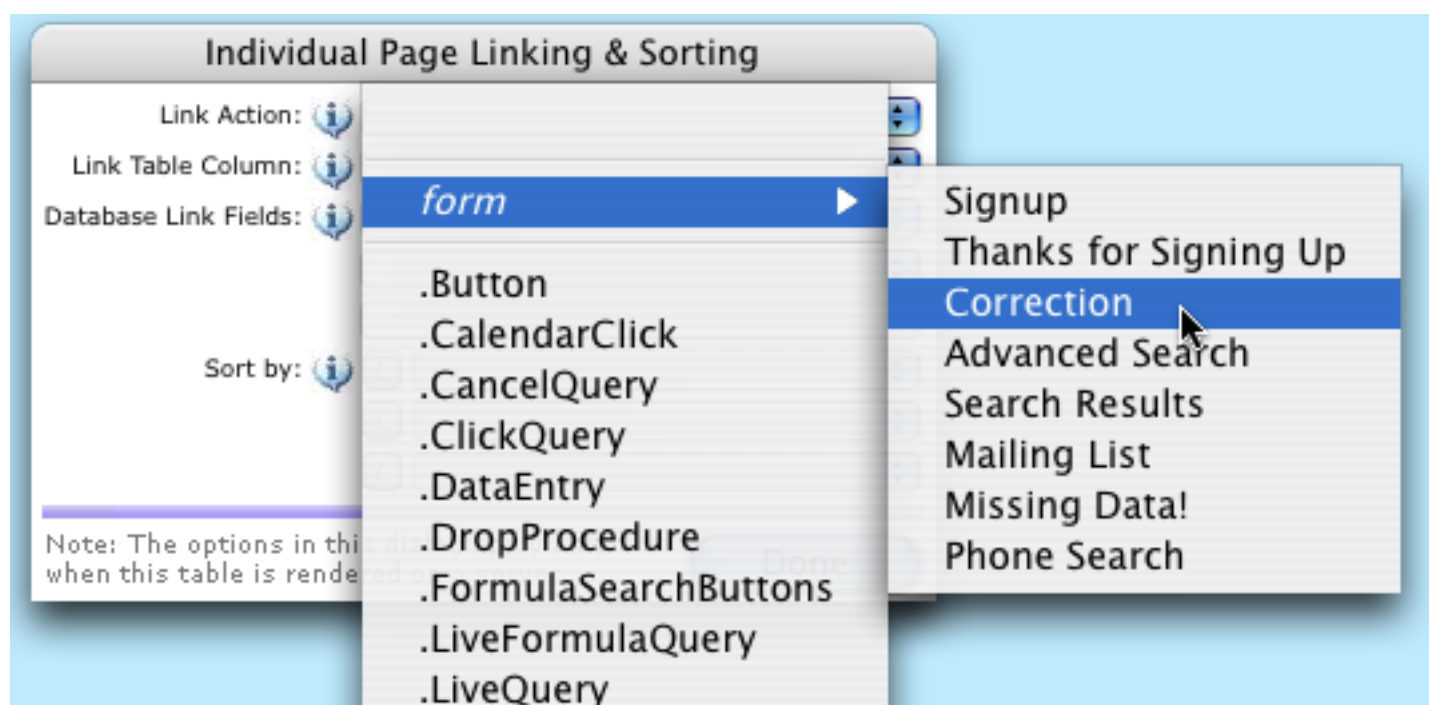
To set up linking from table rows to separate forms, open the Text Export Wizard and select the **Individual Page Linking/Sorting** command from the HTML menu.



The dialog contains just over a half dozen pop-up menus for configuring the links.



Link Action. This option specifies what should happen when the link is clicked. Any procedure can be used as an action, or you can select from the list of web forms that have been set up for this database.



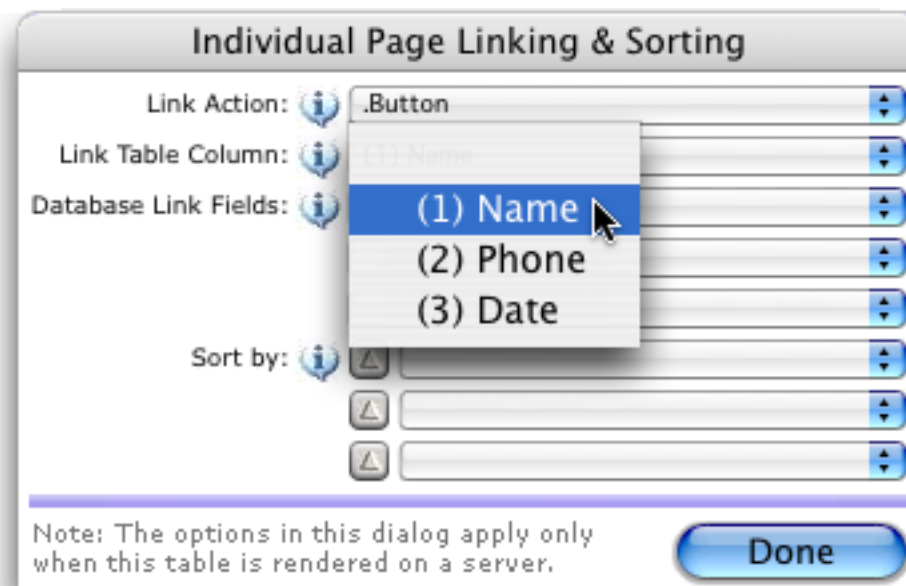
If the link action is a form, everything is taken care of for you. If the link action is a procedure, that procedure must start with this code:

```
weburlselect cgiExtraParameters
```

This line will automatically select the record in the database that is linked to (see “[What Record Are We Talking About?](#)” on page 395).

Link Table Column. This option specifies the column within the web table that will contain the links.

Columns are numbered from left to right (starting with 1). The pop-up menu lists the column numbers and column titles.



In this example, the links are in the first column of the table.

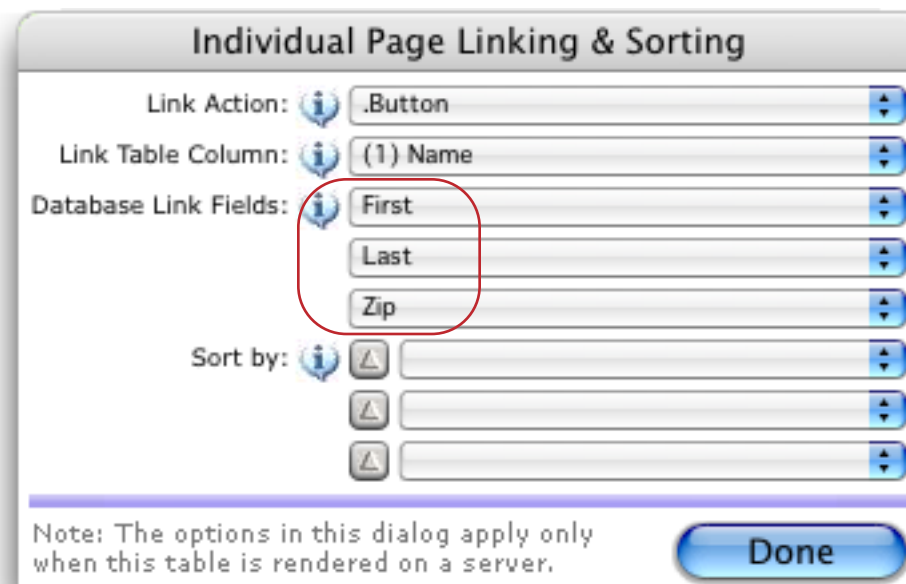
The screenshot shows a browser window titled "Panorama Database Query:Customers/query". It displays a table with three columns: "Name", "Phone", and "Date". The "Name" column contains blue underlined links for each row. A red circle highlights the "Name" column.

| Name | Phone | Date |
|------------------------------------|----------------|----------|
| Middleton, Charles | (562) 938-9850 | 06/09/00 |
| Welsgrau, Edward | (562) 428-4209 | 07/11/00 |
| Horner, Thomas | (562) 431-9792 | 08/17/01 |
| Dalton, Phillip | (516) 931-2202 | 09/27/01 |
| Anderson, Carl | (310) 439-5574 | 10/13/01 |
| Marks, John | (310) 927-8257 | 02/04/02 |
| Bergin, Robert | (516) 432-4252 | 05/23/02 |
| Knight, Herbert | (310) 438-5465 | 12/26/02 |
| Hartman, Joseph | (310) 439-7841 | 05/08/03 |
| Evans, Henry | (601) 867-7935 | 07/22/03 |
| Hulen, Russell | (310) 520-9859 | 08/13/03 |
| Day, Harry | (310) 599-7873 | 01/23/04 |
| Life, Steven | (310) 606-8886 | 02/20/04 |

Only one column in a table can contain the links.

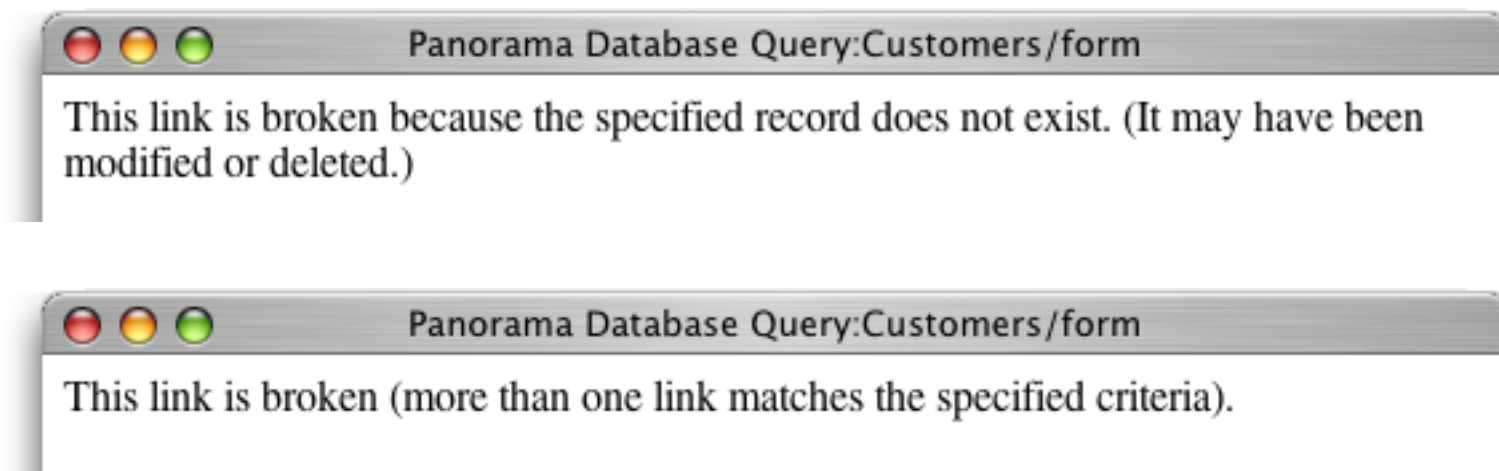
Database Link Fields. In order to complete each link, the Panorama Server must be able to uniquely identify each record in the database. If the database is configured for sharing this is easy because each record contains a unique ID number. If the database is shared you should simply leave all three of the link pop-up menus blank, and you can skip the rest of this section.

If the database is not shared you'll need to use the pop-up menus to specify from one to three fields that when combined will uniquely identify each record (the order in which the fields are chosen does not matter).

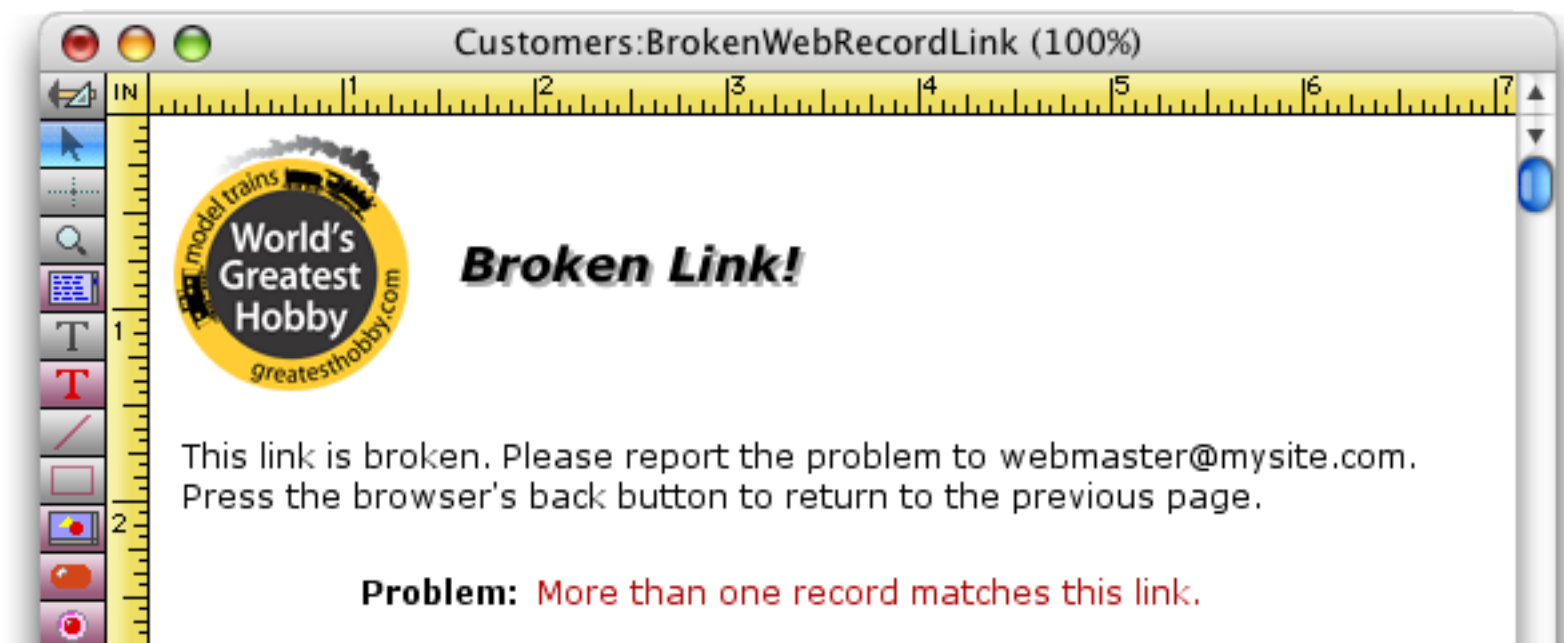


In this example three fields have been specified – First, Last and Zip. For many databases these three fields would be sufficient to uniquely identify each record, but what if two people in a zip code have the same name? Or what if some records don't have any name entered at all? Analyze your data carefully to make sure that the fields that you choose really do uniquely identify a single record. If there is no combination of fields that will uniquely identify each record then you will need to add a new field and make sure that it gets filled with a unique value when each record is created. Or, better, yet, just use a shared database and let Panorama take care of the unique ID for you.

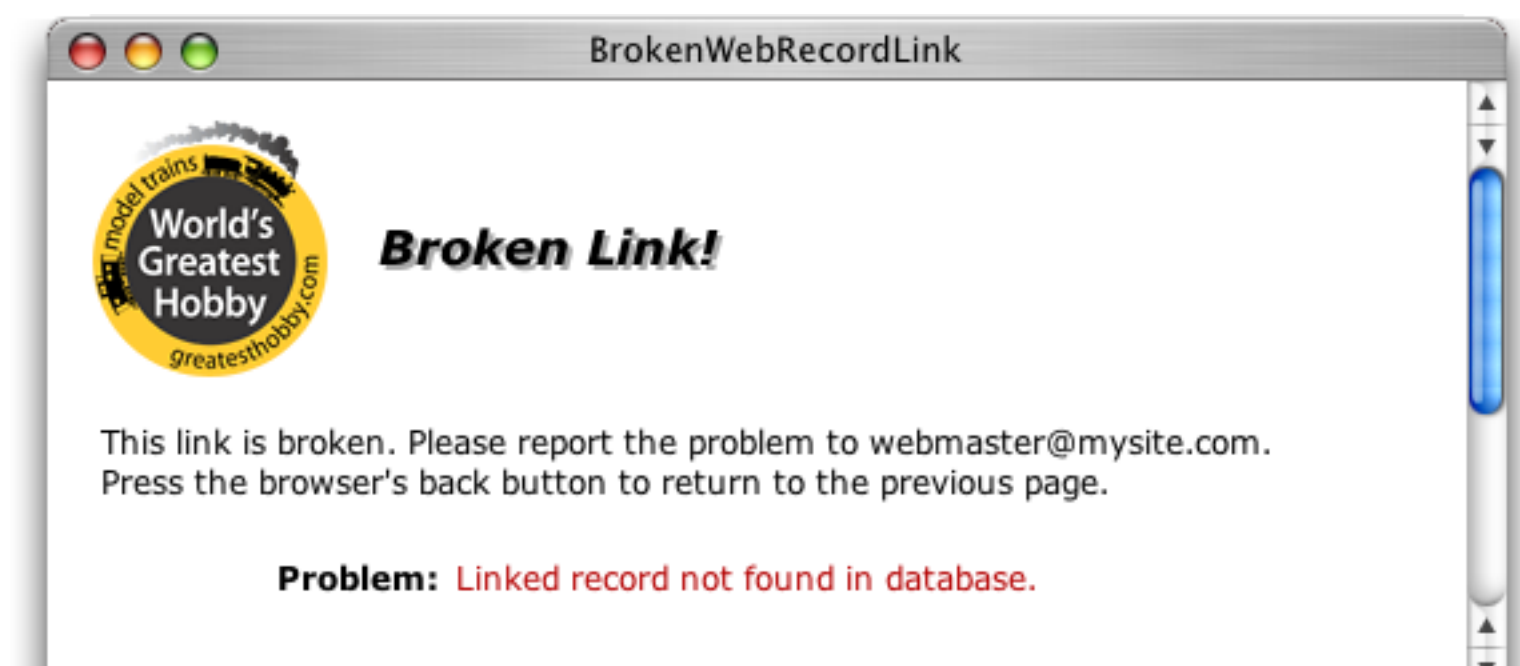
Broken Links. What if the link fields you set up don't always specify a unique record? For example, a link might be ambiguous and match more than one record. Or if the database has been modified since the table was displayed a link could be completely broken, with no records matching the link. In that case the Panorama Server will normally display an automatic error message on the browser.



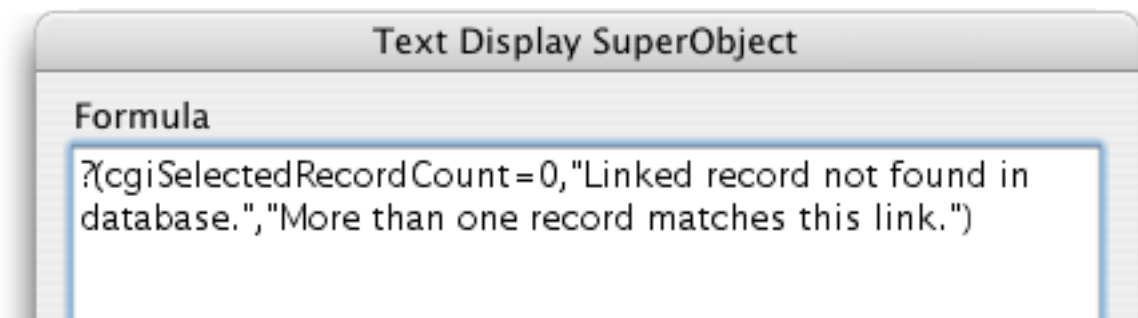
If you don't like this rather plain message, you can set up a form to be displayed when this situation occurs. Create a web form in the database named **BrokenWebRecordLink**.



Then render and upload this form to the server. Now when a broken link occurs, your form will be displayed instead of the generic error message.

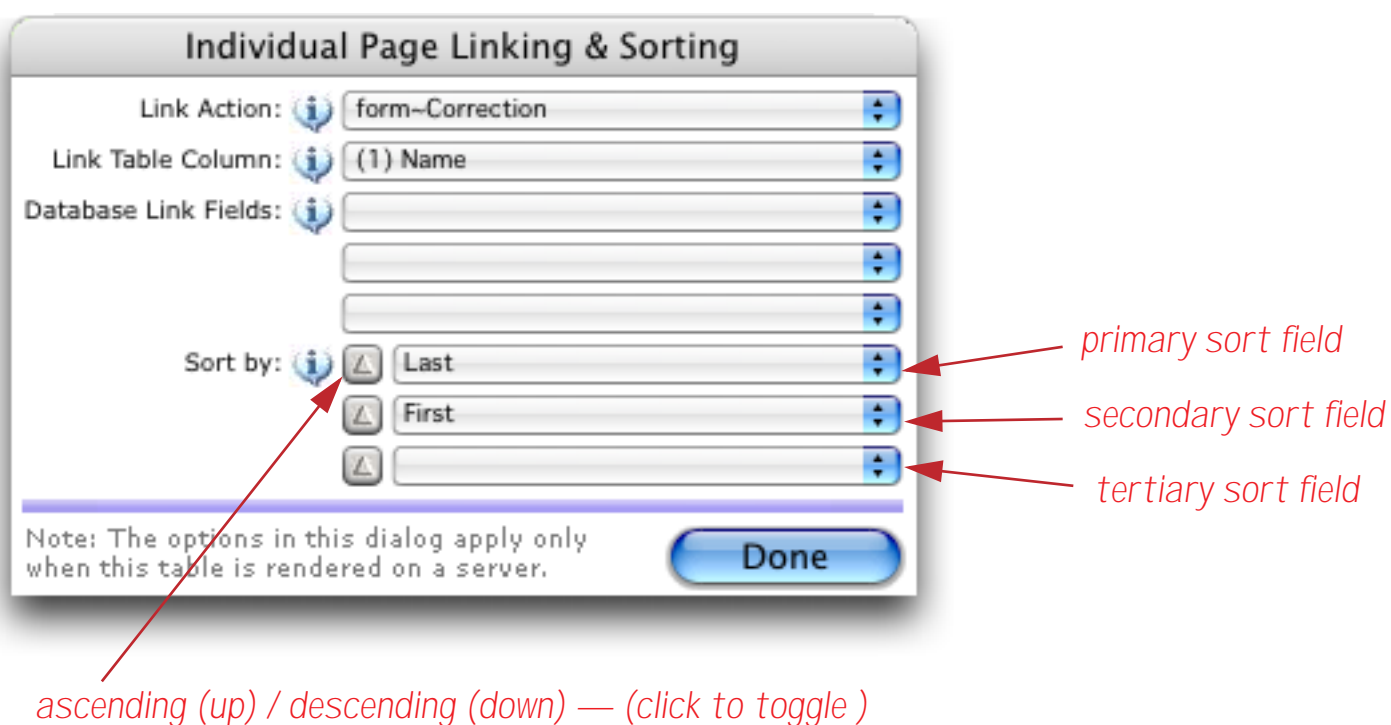


The form can use the `cgiSelectedRecordCount` variable to determine the cause of the broken link. If this variable is zero then the record was not found. If it is two or greater then more than one record matches the link. In our example form, the text in red is displayed based on the value of the `cgiSelectedRecordCount` variable using the `?()` function to display one of two messages.



A second method for handling broken links is to set up a procedure named `.BrokenWebRecordLink`. If a database contains a procedure with this name, it will be automatically called if a broken link occurs (this overrides any `BrokenWebRecordLink` form that may be in the database). The `.BrokenWebRecordLink` procedure must create the text for a web page in the `cgiHTML` global variable (just like any other custom web procedure, see “[Generating HTML](#)” on page 369).

Sort by. If you want the rows in your table to be in a particular order then specify the fields to sort by here. You can specify up to three fields, and each may be sorted in either ascending (up) or descending (down) order.



If the table is designed to automatically split into separate pages if there are too many records (see “[Splitting a Long Table into Multiple Pages](#)” on page 303) then the table must be sorted for proper page navigation.

Editing/Updating a Record

The previous section explained how to link the rows in a table to individual forms. If the form linked to allows editing (if it includes data cells, super text editing objects, checkboxes, radio buttons, etc.) then this form can be used to edit the actual data in the Panorama database. The sequence works like this: 1) Search the database and display the form, 2) click on a link, 3) edit the data, 4) press the Submit button to update the database, 5) the server displays a “thank you” page confirming that the database has been updated.

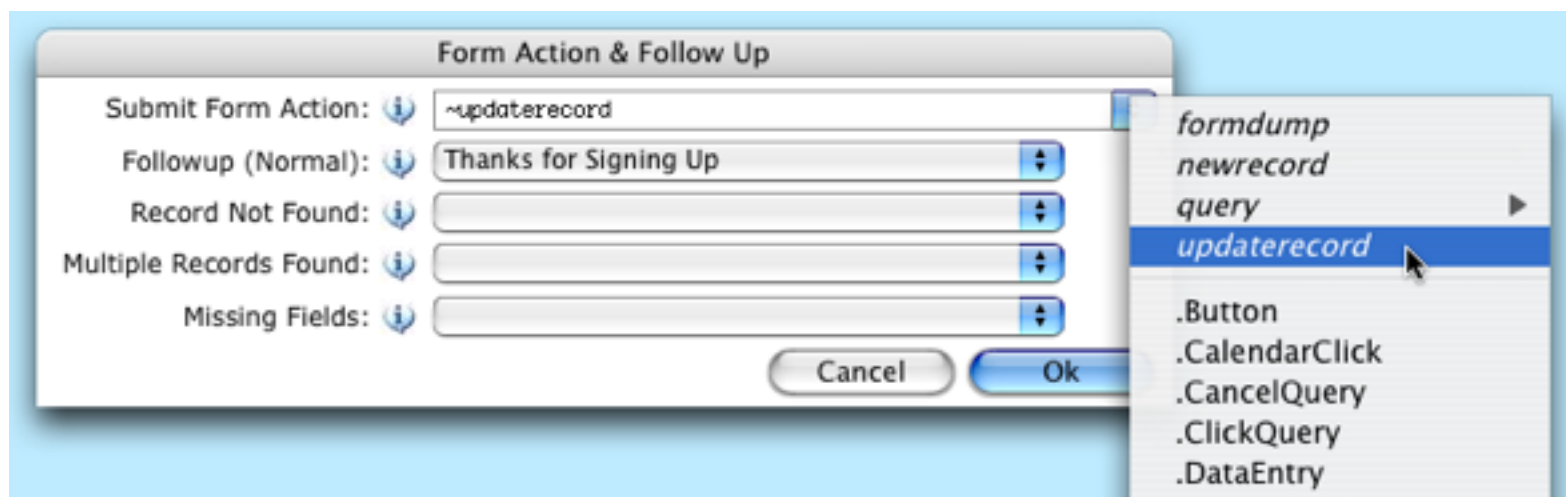
The sequence of screenshots illustrates the process of editing a record in a database:

- Search Results:** A browser window showing search results for customers. A table lists customer details including Name, City, State, Scale, Era, and Railroad. A red circle with the number '1' is placed over the search bar.
- Clicking a Link:** A red circle with the number '2' is placed over a link in the table, with an arrow pointing to the next screenshot.
- Correction Form:** A browser window titled 'Correction' showing a form for editing a record. The form includes fields for First Name, Last Name, Address, City, State, Zip Code, Phone, Email, Scale, Era, Railroad, and Notes. A red circle with the number '3' is placed over the Address field, with an arrow pointing to the next screenshot.
- Submit Button:** A red circle with the number '4' is placed over the 'Submit' button, with an arrow pointing to the next screenshot.
- Thanks for Signing Up:** A browser window titled 'Thanks for Signing Up' showing a confirmation message and the updated record details. A red circle with the number '5' is placed over the 'Thanks for Signing Up!' heading, with an arrow pointing to the previous screenshot.

| Name | City | State | Scale | Era | Railroad |
|-----------------------------------|----------------|-------|-------|------------|----------------------------|
| Charles Middleton | Long Beach, CA | CA | G | Modern | Santa Fe (ATSF) |
| Edward Weisbrau | Long Beach, CA | CA | N | Transition | Amtrak |
| Thomas Horner | Long Beach, CA | CA | G | Transition | Pennsylvania |
| Willa Carter | Long beach, NY | NY | N | Steam | Great Northern (GN) |
| Carl Anderson | Long Beach, CA | CA | N | Modern | Union Pacific (UP) |
| John Miller | Long Beach, CA | CA | HO | Modern | Conrail |
| Robert Bergin | Long Beach, NY | NY | G | Modern | Santa Fe (BNSF) |
| Herbert Knisht | Long Beach, CA | CA | HO | Transition | Santa Fe (ATSF) |
| Joseph Hartman | Long Beach, CA | CA | HO | Transition | Santa Fe (ATSF) |
| Henry Evans | Long Beach, MS | MS | HO | Modern | Union Pacific (UP) |
| Russell Hulm | Long Beach, CA | CA | G | Modern | Southern Pacific (SP) |
| Harry Gray | Long Beach, CA | CA | N | Transition | Norfolk & Western |
| Shawn Life | Long Beach, CA | CA | G | Transition | Western Pacific (WP) |
| Douglas Hillman | Long Beach, CA | CA | O | Modern | Seaboard Coast Line |
| Henry Mastler | Long Beach, CA | CA | HO | Modern | Chicago & North Western (C |

Preparing a Database Update Form

A database update form is created much like any other web form. In fact, the only difference is in how the **Form Action & Follow Up** dialog is set up. Simply use the pop-up menu to set the **Submit Form Action** to `updaterecord`.



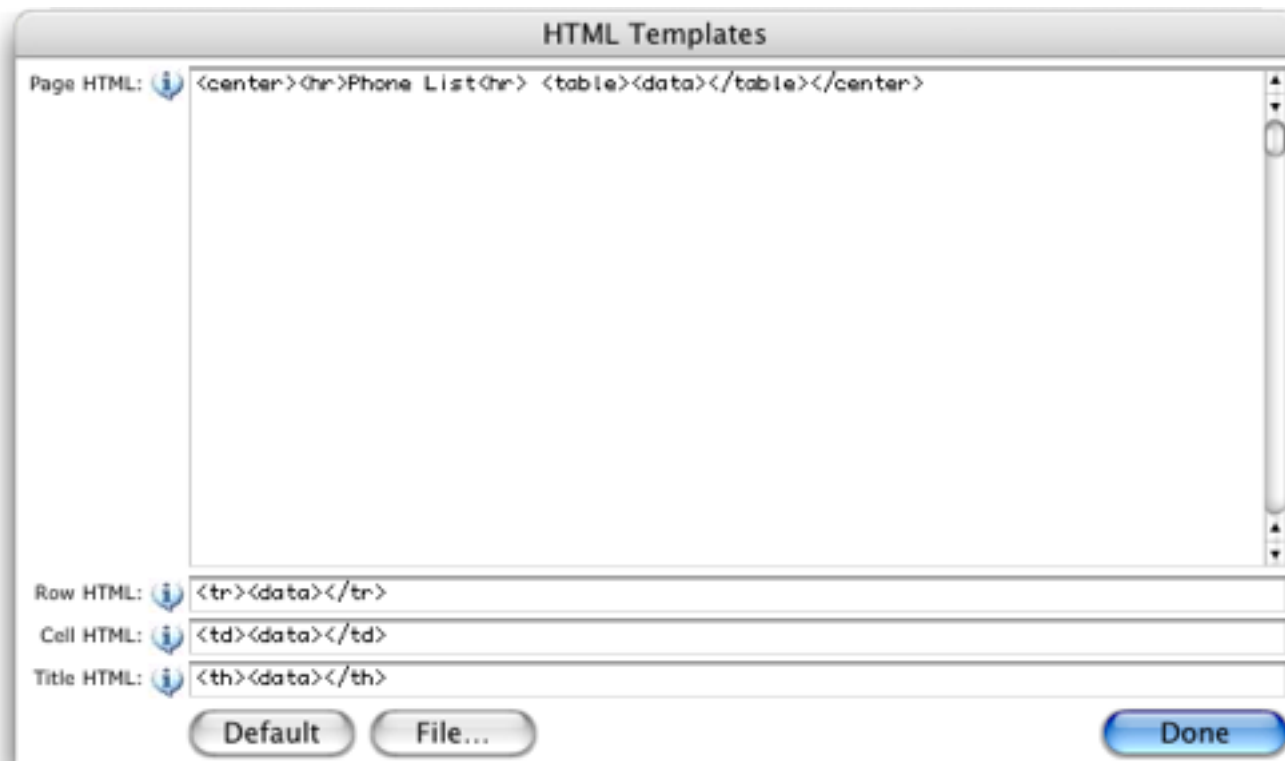
You may also want to set up the **Followup (Normal)** and **Missing Fields** options (see “[Standard Form Action — NEWRECORD](#)” on page 243 and “[Checking for Required Fields/Preventing Missing Fields](#)” on page 245). Then press the Ok button, render and upload the form, and you are ready to edit your database online! That’s all there is to it.

Broken Record Identification. When the Panorama Server displays a web form from a table link, it invisibly embeds identifying information for the record into the form. This identifying information is based on the table’s link fields (see “[Database Link Fields](#)” on page 310). But what if the link fields you set up don’t always specify a unique record? For example, a link might be ambiguous and match more than one record. Or if the database has been modified since the table was displayed, a link could be completely broken, with no records matching the link. In that case when you press the **Submit** button, the Panorama Server will normally display an automatic error message on the browser. This situation is very rare because it can only happen if the record in question has been modified between the time the form was displayed and the **Submit** button is pressed, but it can happen (if the database is a shared database, it can only happen if the record has been deleted by another user).


If you don’t want the user to see Panorama’s generic error message in this situation, you can use the **Form Action & Follow Up** dialog to specify a web form to display if these errors occur. The **Record Not Found** form will be displayed if the record being updated no longer exists. The **Multiple Records Found** form will be displayed if there is now more than one record that matches the identifying information for the database (this can never happen if the database is shared).

Customizing the table HTML (advanced)

The Panorama Server normally generates the HTML for a web table page automatically, possibly with the help of a web form (see “[Table Header Form](#)” on page 285). You can, however, use the HTML Templates dialog (in the Text Export Wizard’s HTML menu) to customize the HTML Panorama generates.



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

| Option | Examples | Description |
|------------|---|---|
| Page HTML | <pre><center><hr>Phone List<hr> <table><data></table></center></pre>  | <p>This is the template for the overall HTML of the body of the page (the server will create the header and footer automatically). At a minimum it should include the tags <code><table><data></table></code>. The wizard will automatically replace the <code><data></code> tag with the body of table generated from the database.</p> |
| Row HTML | <pre><tr valign=center><data></tr></pre> | <p>This is the template for each line in the table. At a minimum it should include the tags <code><tr><data></tr></code>. The wizard will automatically replace the <code><data></code> tag with the body of the line (which has been built up from individual cells, see the next entry).</p> |
| Cell HTML | <pre><td><i><data></i></td></pre> | <p>This is the template for each cell in the table. At a minimum it should include the tags <code><td><data></td></code>. The wizard will automatically replace the <code><data></code> 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.</p> |
| Title HTML | <pre><td><data></td></pre> | <p>This is the template for each title in the table. At a minimum it should include the tags <code><th><data></th></code> or <code><td><data></td></code>. The wizard will automatically replace the <code><data></code> tag with the column title as specified in the wizard.</p> |

Chapter 8: Web Programming 101

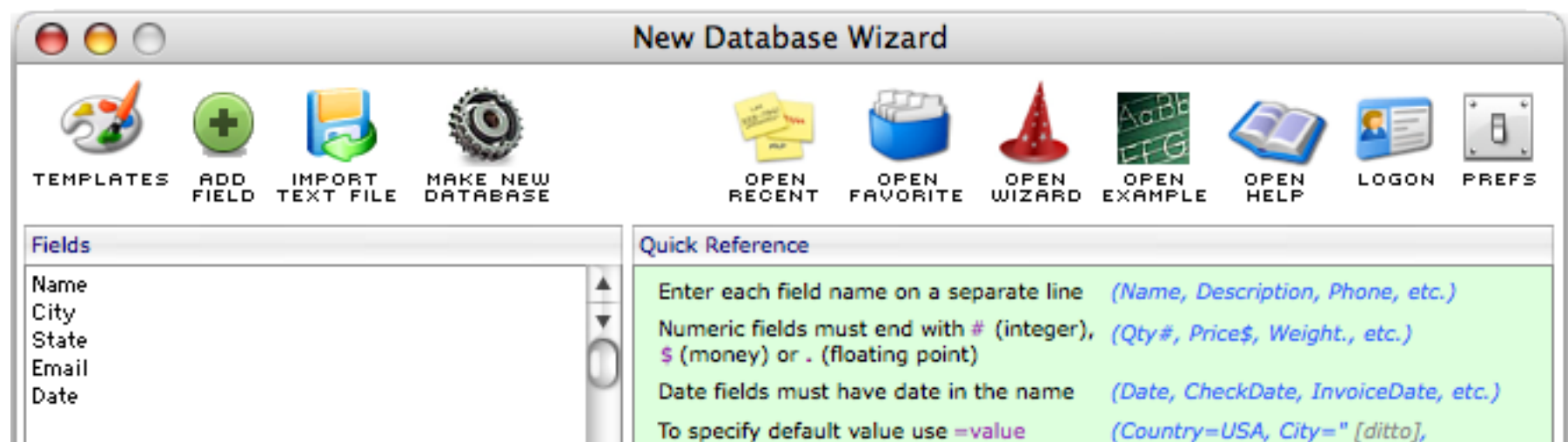


Basic web database applications can be created using the form and table templates described in the previous chapters. More advanced applications are created by writing custom web action procedures (programming), sometimes called **CGI procedures** (CGI stands for **Common Gateway Interface**, and refers to a standard protocol for interfacing external application software, like Panorama, with web servers). A CGI procedure takes a request passed to it from the web server and responds with an HTML page which is passed back to the client's browser. Because a CGI procedure can be programmed using all of the power and flexibility of Panorama's programming language a CGI procedure can do just about anything you could want.

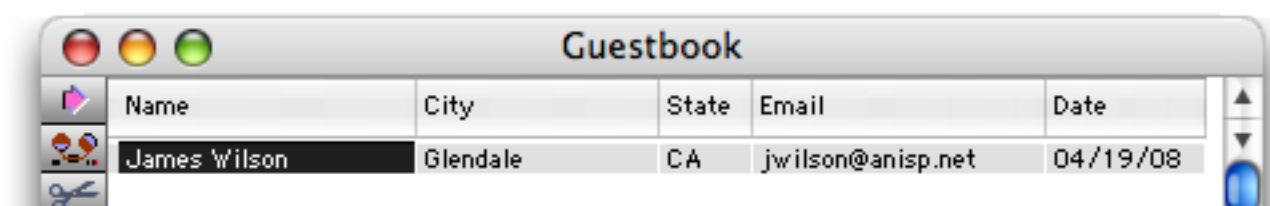
The next four chapters explain how to create, test and deploy your own web procedures. These chapters assume that you are already familiar with writing custom procedures in single user Panorama databases. If you need a refresher, see Chapters 24 (Procedures) and 25 (Programming Techniques) of the **Panorama Handbook** (you may also want to review Chapter 23, Formulas).

Creating a Simple Guestbook Web Database

To illustrate basic web programming techniques we'll start by creating, uploading and testing a simple database that includes custom programming. The first step is to create a standard Panorama database, which we'll do with the **New Database** wizard (see Chapter 1 of the Panorama Handbook for further details).



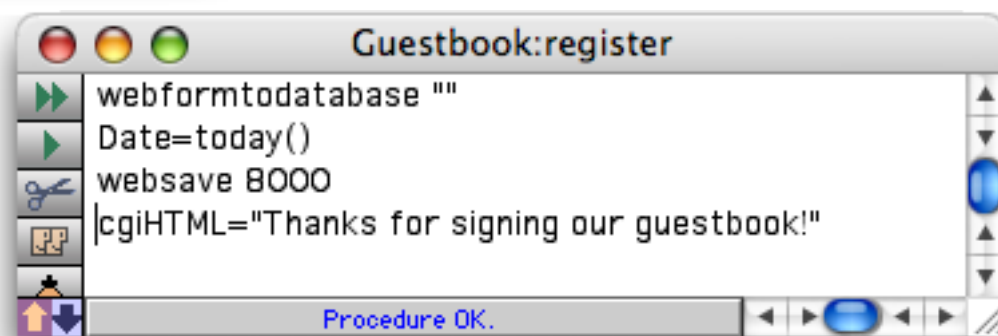
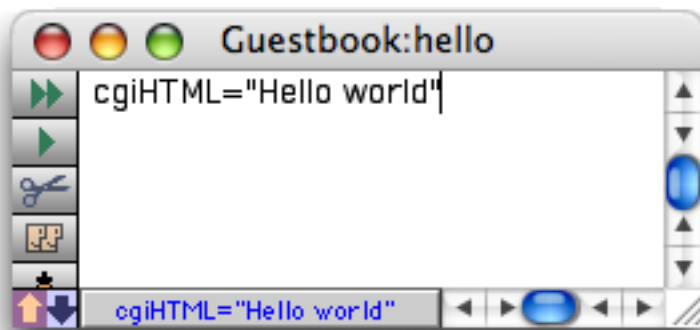
As you can see, the database contains five fields.



All of these are text fields except for the **Date** field. I've pre-filled in the first record just to get things started.

Creating Web Procedures

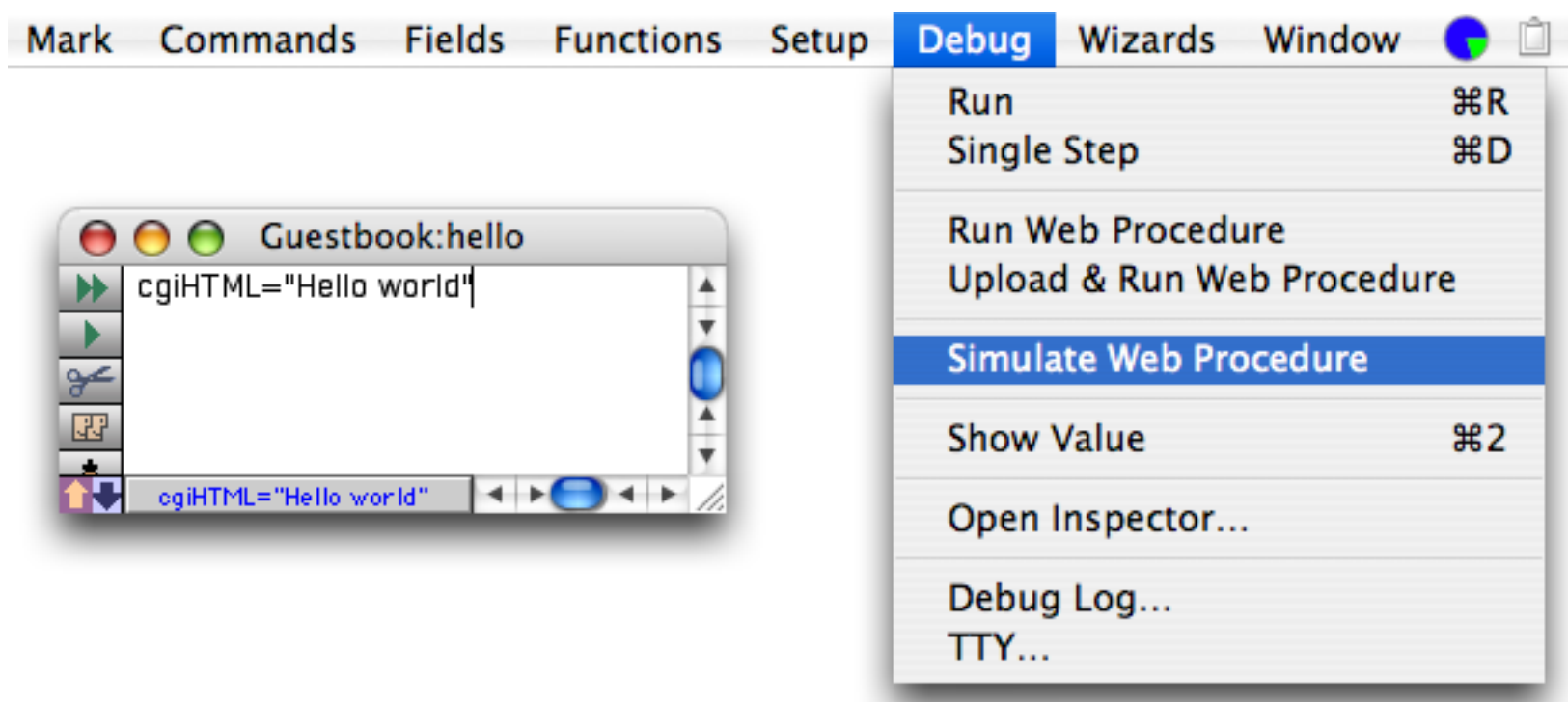
Before uploading the database to the server I'll add two web procedures — `hello` and `register` (web procedures can also be added after uploading the database, as you'll learn later in this chapter).



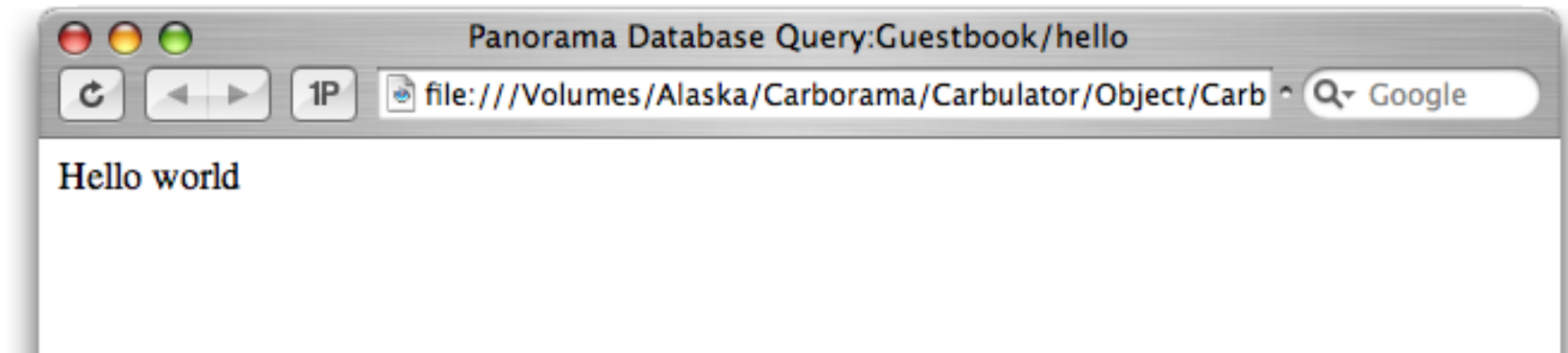
These procedures are created with the **New Procedure** command in the **View** menu, just like any other procedures. If you are following along you can simply type in the procedure contents shown above — we'll explain what the code in these procedures does later.

Testing a Procedure in Advance

Before going any further let's test the `hello` procedure. This can be done right on your computer (in fact, you don't even need a server to run this test). Start by making sure the `hello` procedure is open and is the topmost window, then choose **Simulate Web Procedure** from the **Debug** menu.



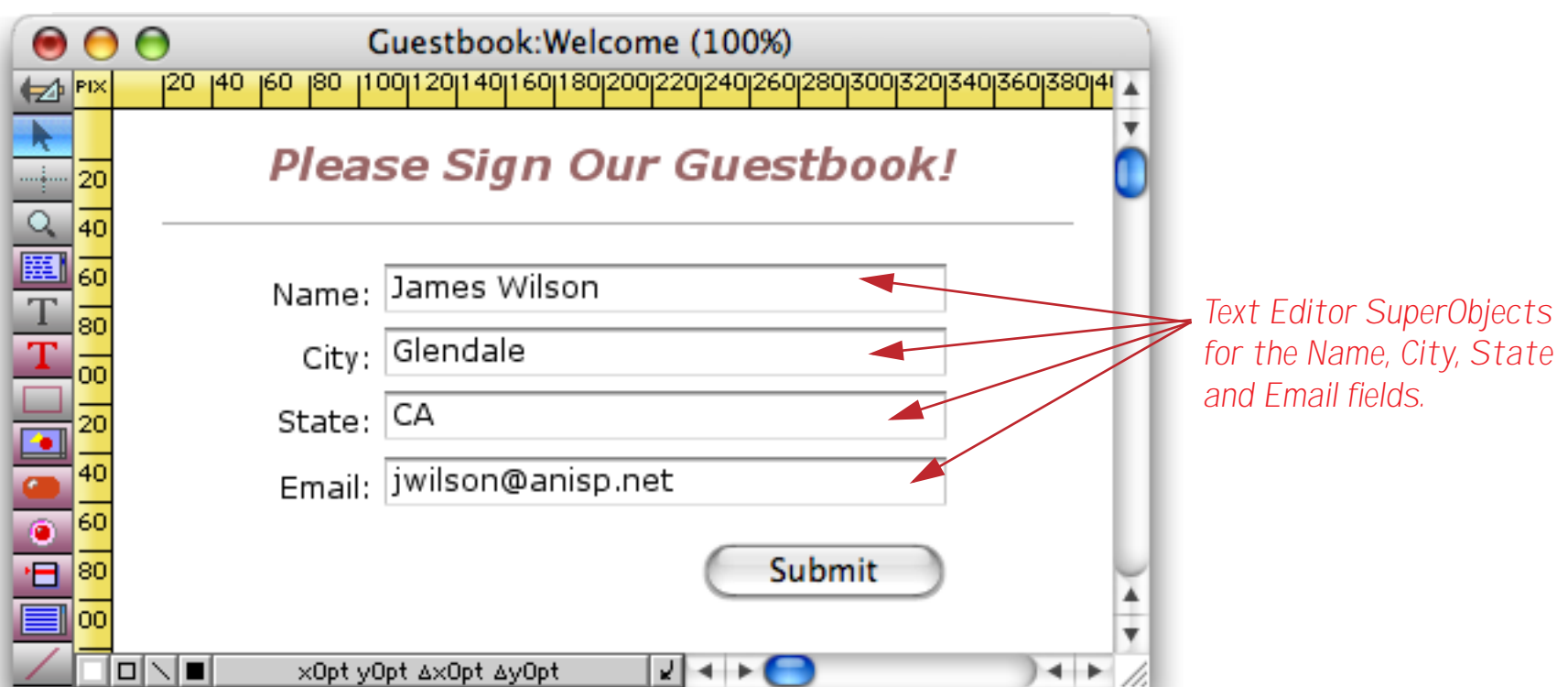
Your web browser will open and a new window will appear with the results of the procedure.



Since this procedure is only one line long, it's not surprising that it worked perfectly the first time. Of course more complex procedures often don't work perfectly the first time, so later in this chapter you'll learn how the simulator can help you find and fix bugs.

Create a Web Form

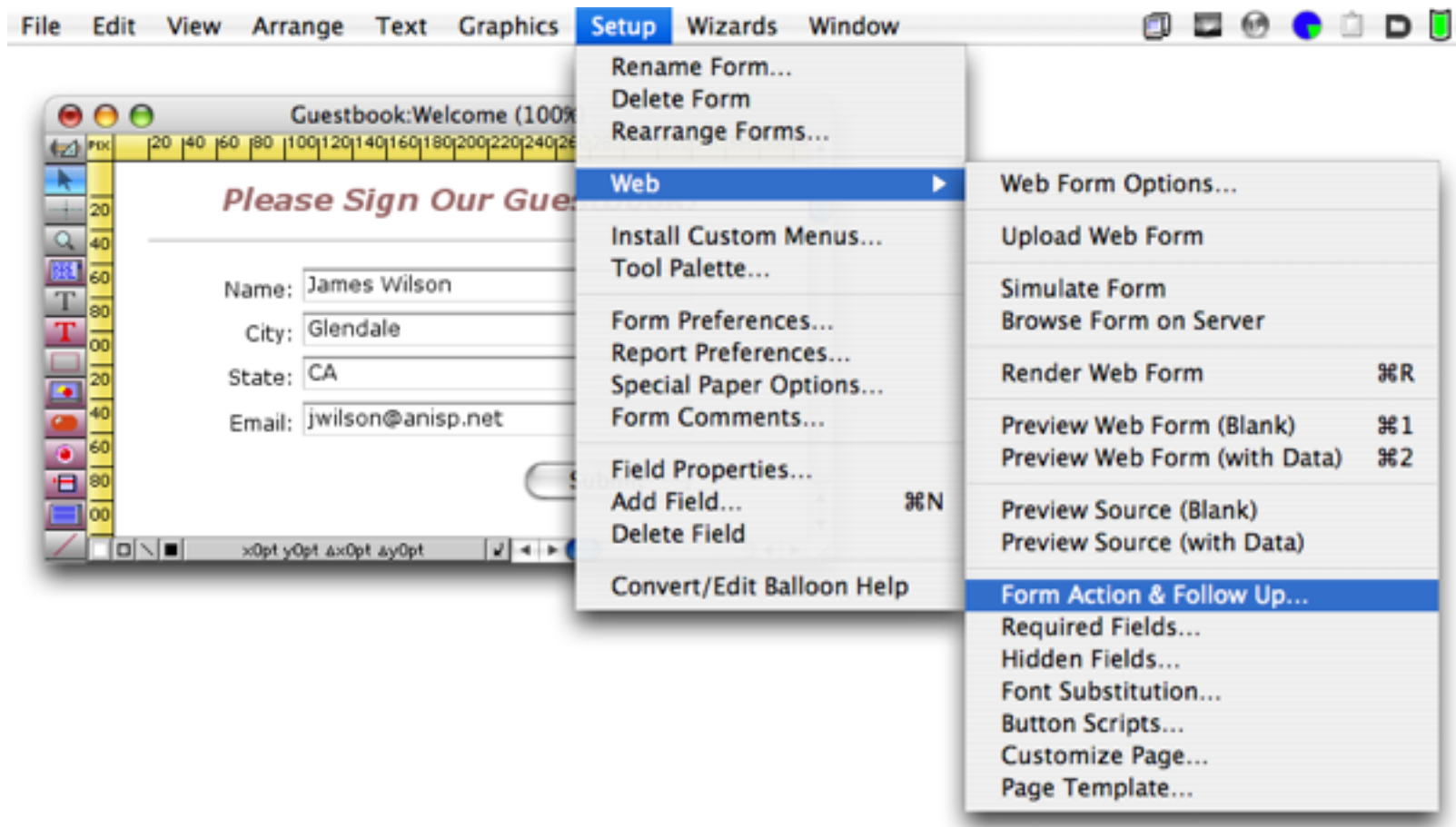
Next I want to test the [register](#) procedure, but to do that, I need to create a new web form.



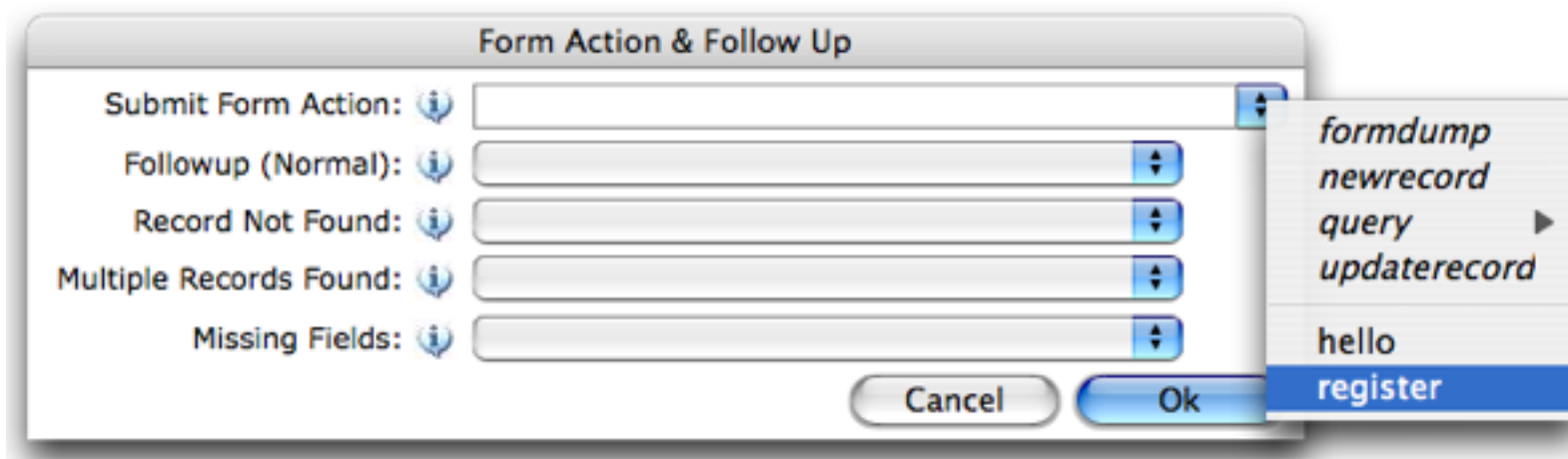
See "[Converting a Panorama Form into a Web Form](#)" on page 231 if you need to review the techniques for making a web form.

Assigning a Procedure to the Form

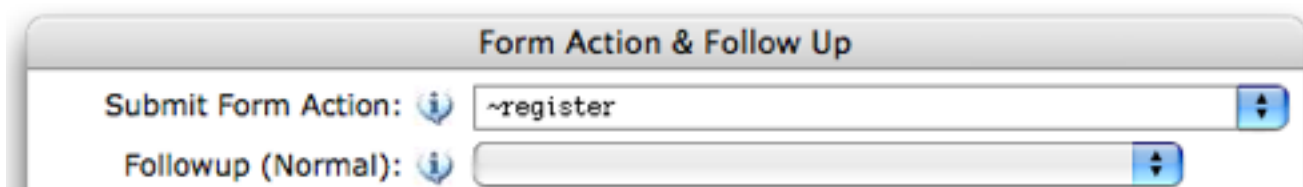
Once the form is complete you need to assign the **register** procedure to it. This tells Panorama Server to run the **register** procedure when the submit button is pressed, allowing the procedure to process whatever the user typed into the form. To make this assignment first make sure the form is open and in graphics mode, then choose **Form Action & Follow Up** from the **Web** submenu of the **Setup** menu.



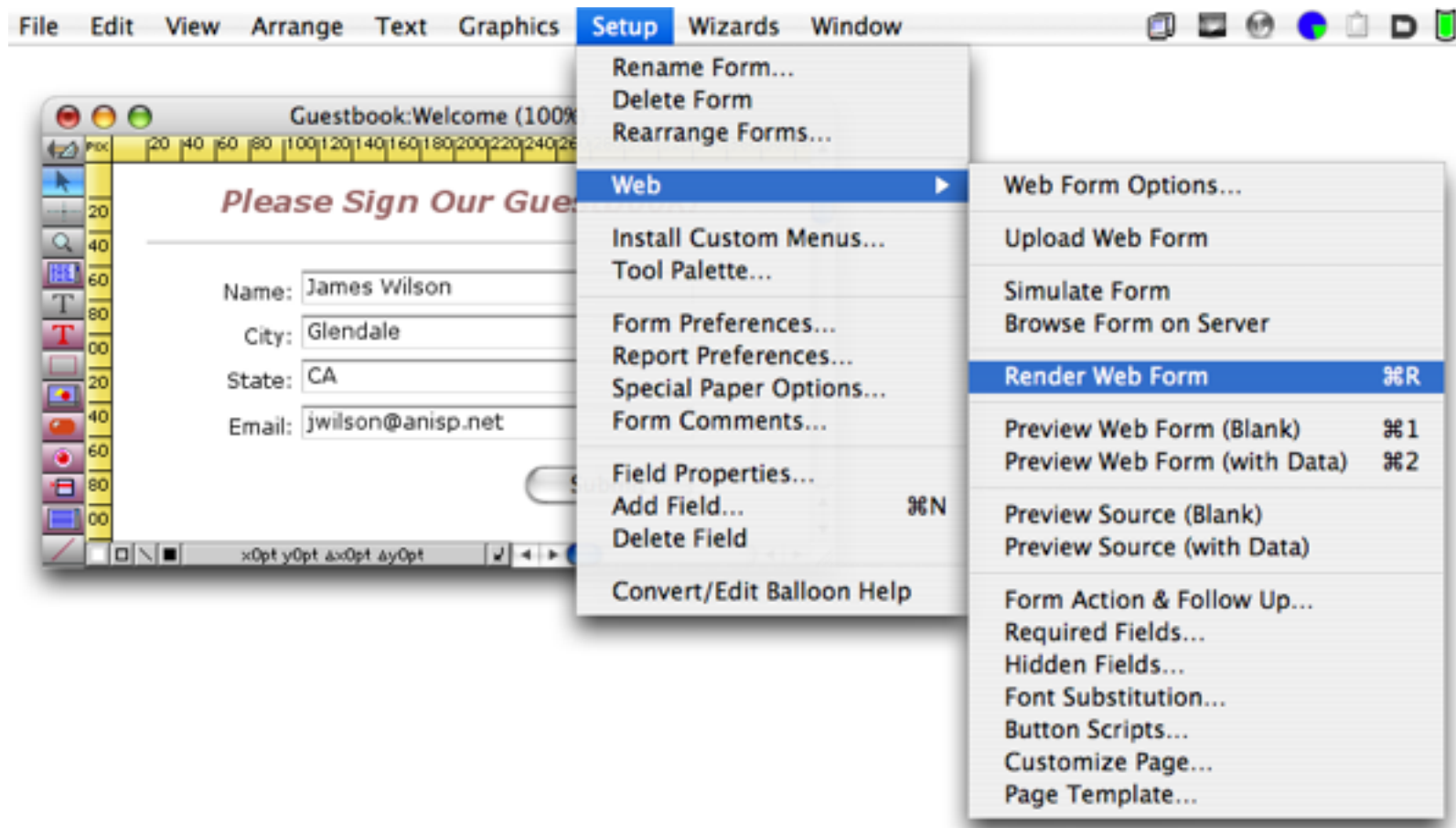
The **Form Action & Follow Up** dialog will appear. The first item in this dialog is **Submit Form Action**, with a pop-up menu on the right. Choose the procedure you want to assign to the form (in this case **register**) from this pop-up menu.



The procedure name (in this case **register**) will appear in the **Submit Form Action** area, preceded by a **~** character.



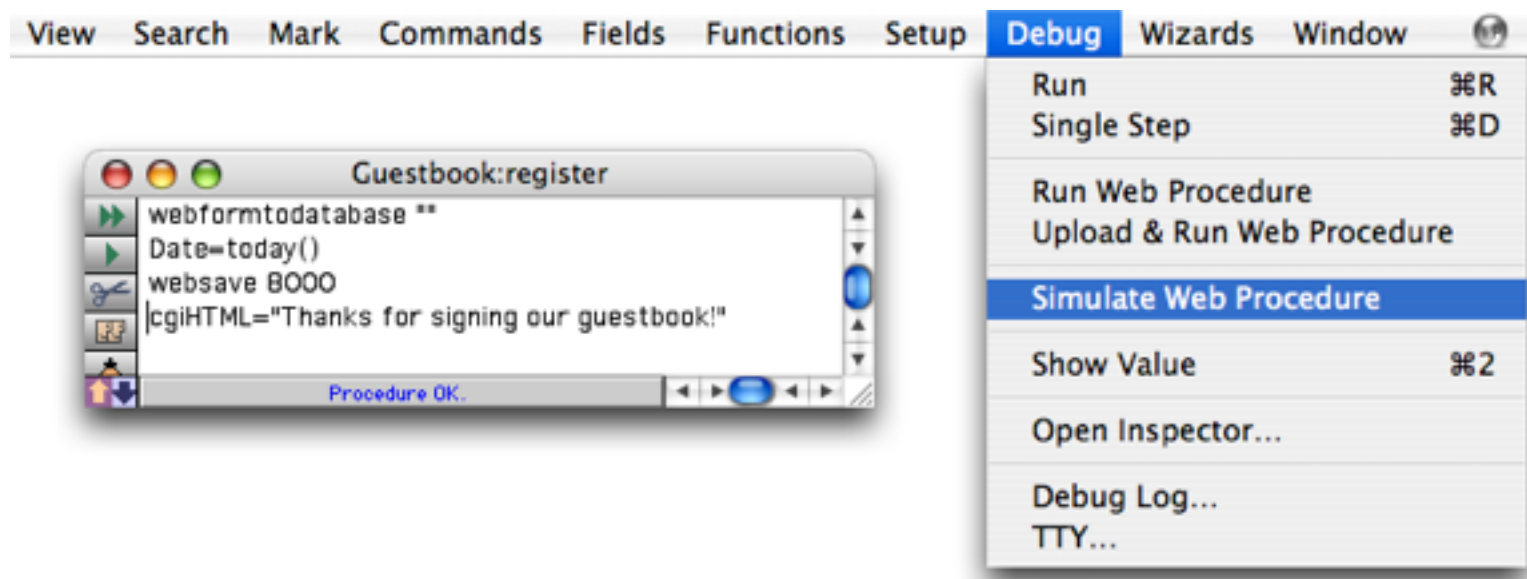
Press **Ok** to close the dialog, then re-render the form by choosing **Render Web Form** from the **Web** submenu of the **Setup** menu. (If this database had already been uploaded to the server, you would also need to upload the web form to the server after rendering it.)



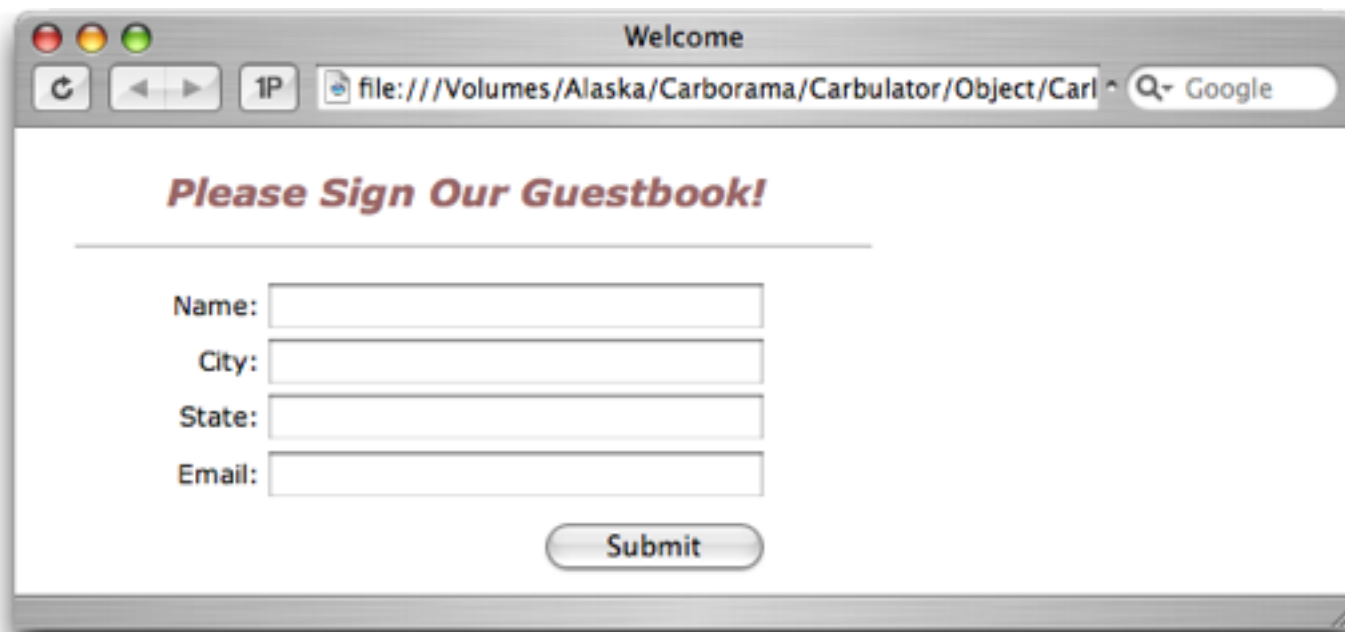
The **register** procedure is now assigned to this form.

Testing the Form and Procedure using Simulation

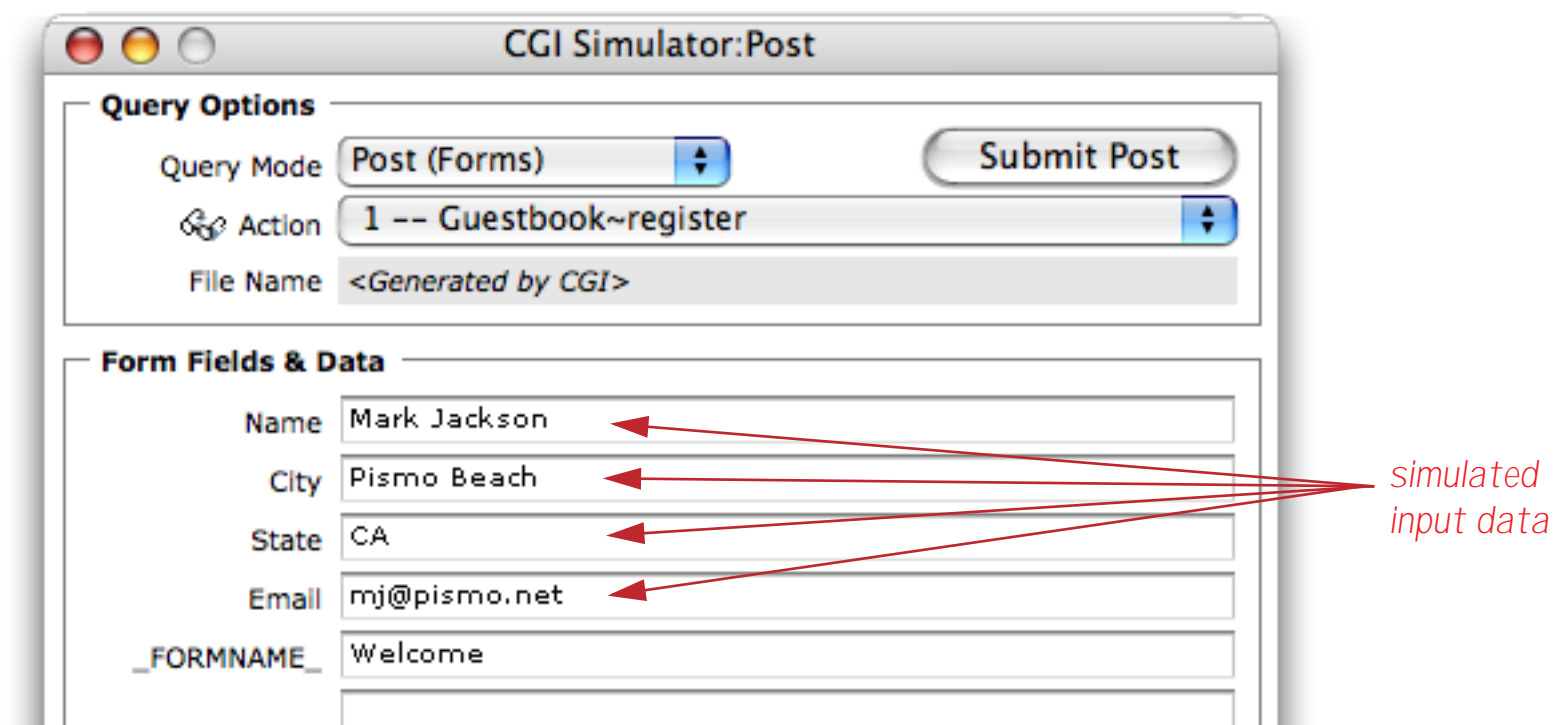
Just like the simple hello procedure, the **Welcome** form/**register** procedure combination can be tested by simulation before you upload the database. To test, make sure that procedure is the top window then choose **Simulate Web Procedure** from the **Debug** menu. (Another option is to open the form in Graphics mode and choose **Simulate Form** from the **Web** submenu of the **Setup** menu.)



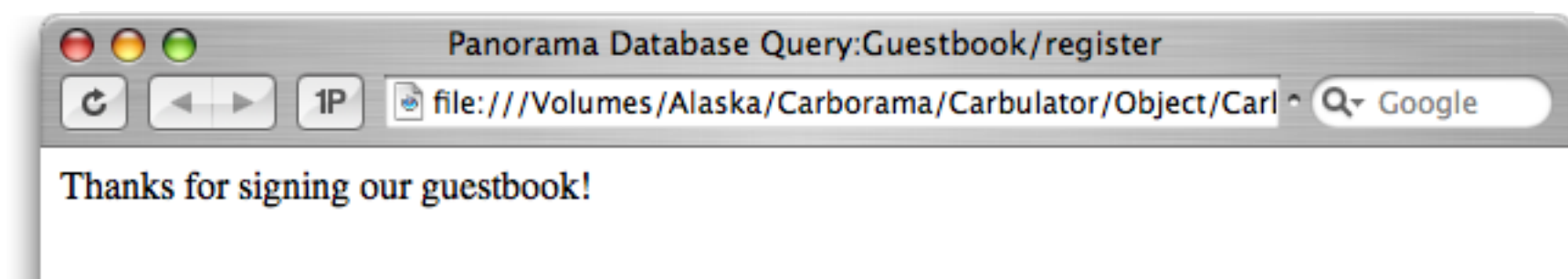
Running this simulation makes the form appear in your web browser. Don't fill out this form, though. Since you're not actually running a web server, the form in your web browser won't work.



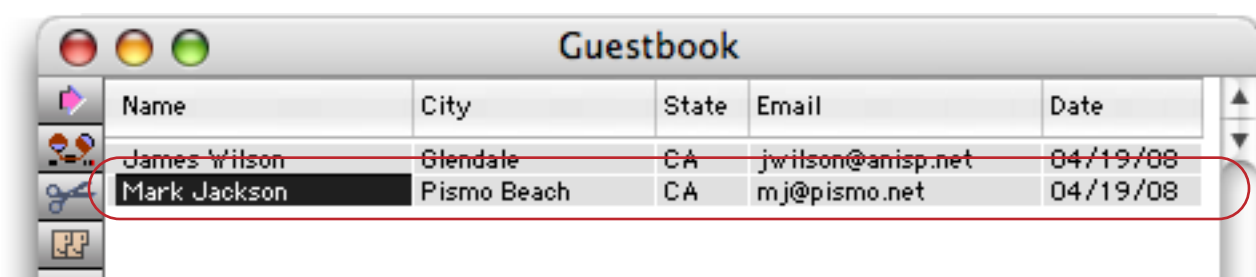
To continue with the test, click back to Panorama (a convenient way to do this is to click on Panorama's dock icon). You'll see that a new **CGI Simulator:Post** window has appeared. This window simulates the web form. It doesn't show any of the graphics or layout of the form, but it does allow you to enter simulated data into any fields on the web form.



Once the simulated data is entered press the **Submit Post** button. This triggers the procedure assigned to this form, just like pressing the **Submit** button on the web form will when this database is running on the server. The result will appear in a new browser window.



If you go back to Panorama you'll see that the data input into the **CGI Simulator:Post** window has been added to the **Guestbook** database, along with the date the data was entered.

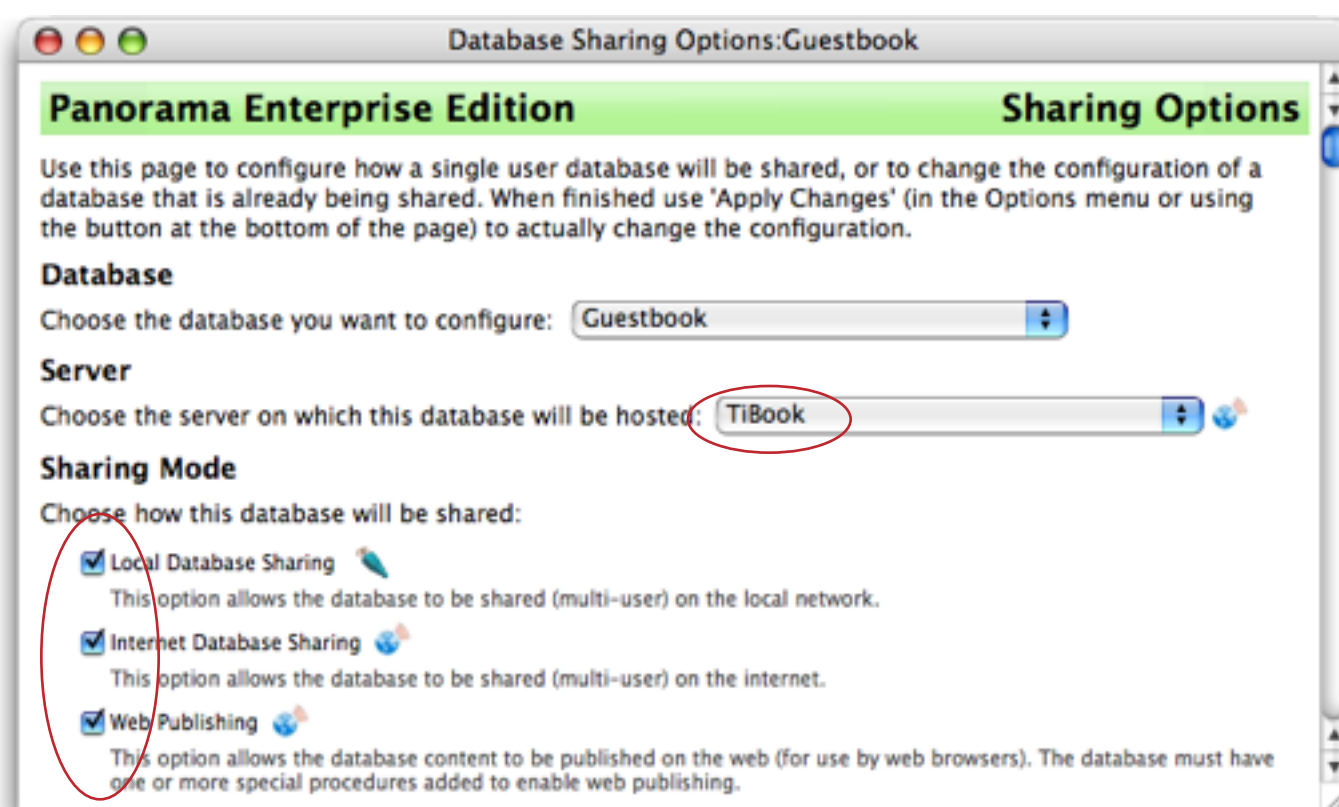


| Name | City | State | Email | Date |
|--------------|-------------|-------|-------------------|----------|
| James Wilson | Glendale | CA | jwilson@anisp.net | 04/19/08 |
| Mark Jackson | Pismo Beach | CA | mj@pismo.net | 04/19/08 |

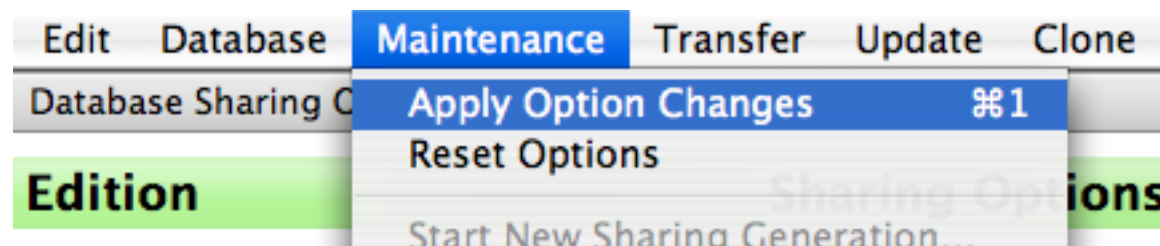
Now that I've verified that the procedures in the **Guestbook** works properly the next step is to upload the database to a server.

Uploading the Guestbook Database

The details of uploading a database for web publishing were covered in Chapter 5 (see "[Uploading the Database to the Server](#)" on page 194). To share this **Guestbook** database I'll open the **Database Sharing Options** wizard, choose the server (**TiBook**) and check all three sharing mode options (**Local Database Sharing**, **Internet Database Sharing** and **Web Publishing**).



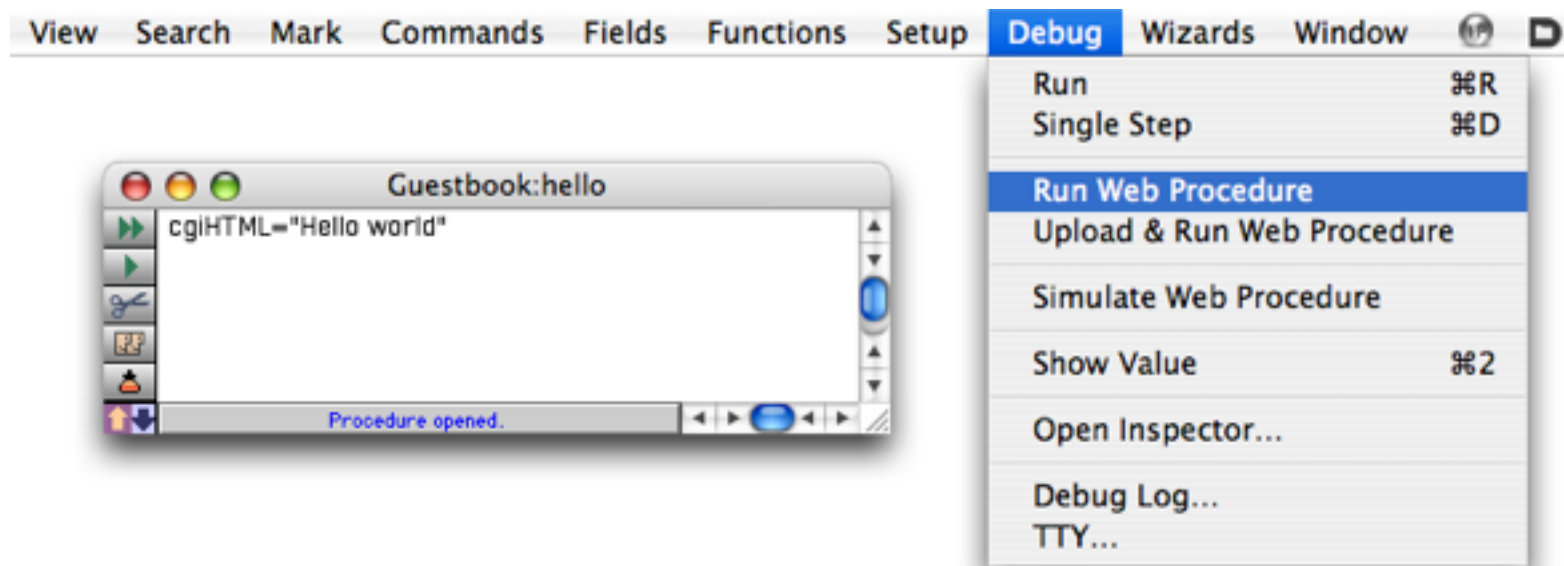
Once these options are set I complete the upload by choosing **Apply Changes** from the **Maintenance** menu (or pressing **Command-1**). (Or I can scroll to the bottom of the wizard and press the **Apply Changes** button.



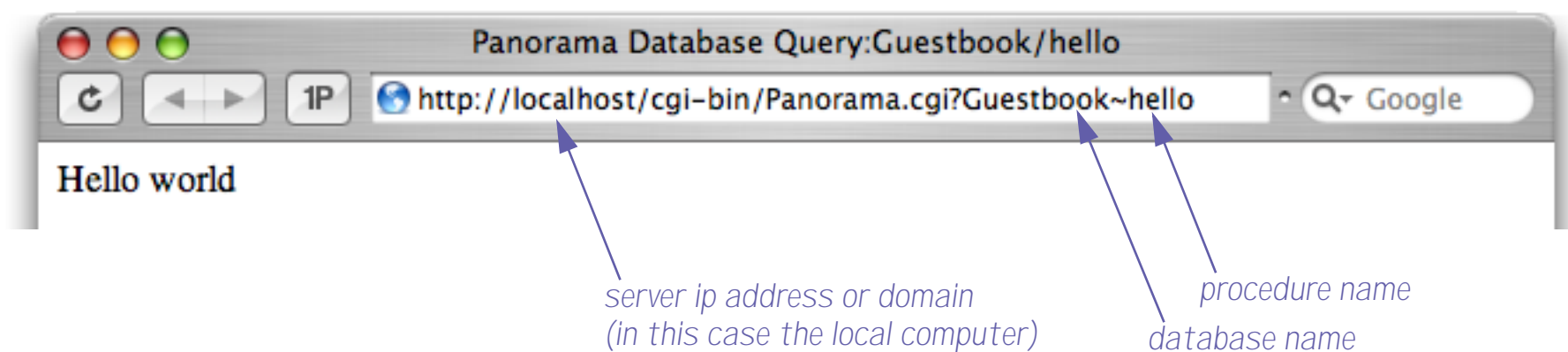
After a confirmation dialog the database will be uploaded to the server.

Testing Web Procedures on the Server

Once the database is uploaded to the server you'll want to test the procedures. There are several ways to do this but the easiest is to open a client copy of the database on your local machine (if it's not already open), open the procedure you want to try, and then choose **Run Web Procedure** from the **Debug** menu.



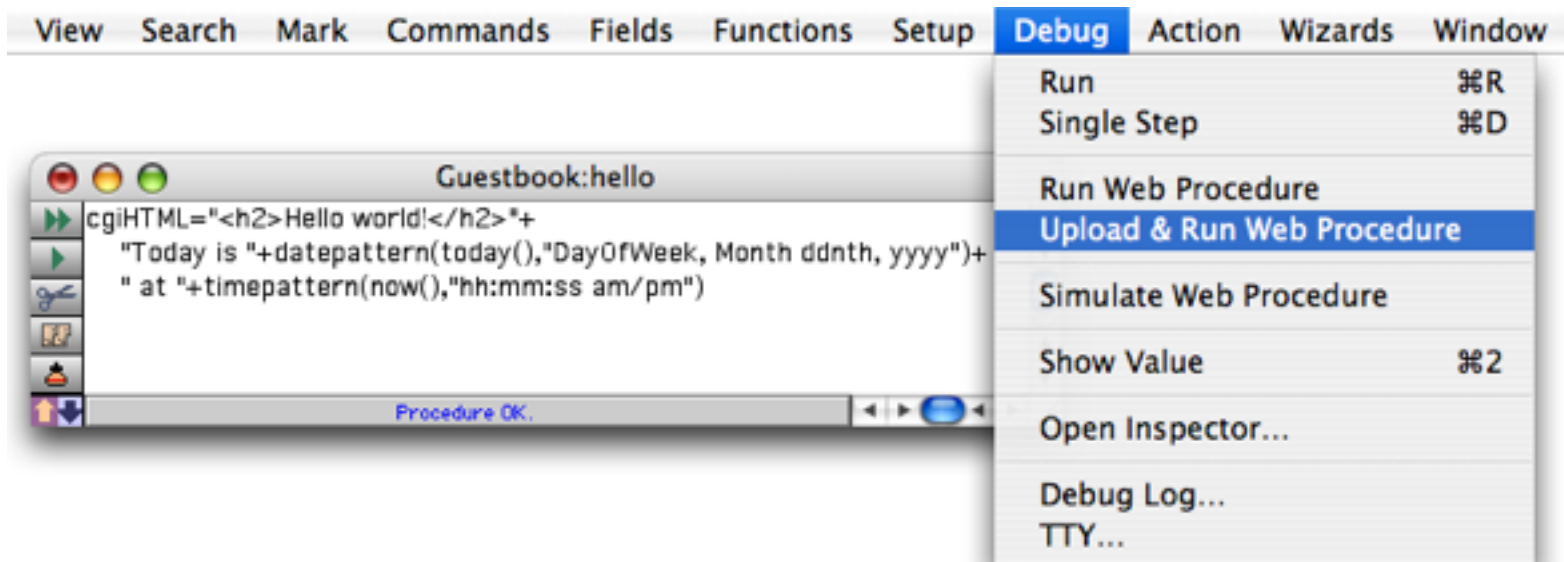
This command will open your web browser and trigger the URL necessary to test the procedure. The results will appear in a new browser window.



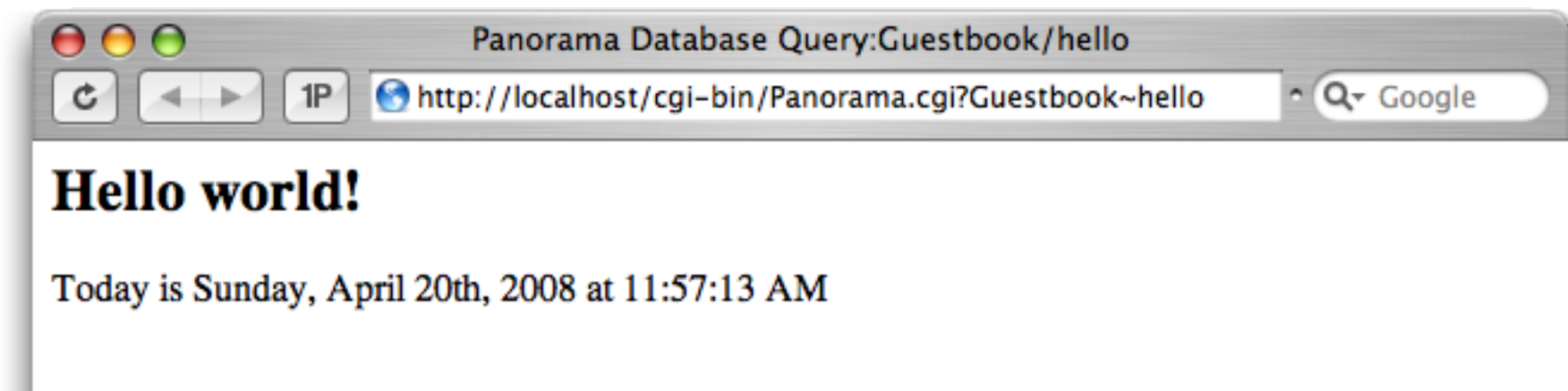
As you can see the URL contains the server ip address (or domain name), the database name, and the procedure name. The exact details of Panorama Server URL's will be discussed later in the next chapter (see "[What Records are we Talking About? \(The WebSelect Statement\)](#)" on page 413).

Modifying a Web Procedure

To modify a web procedure start by modifying the local client copy of the procedure. When your changes are done choose **Upload & Run Web Procedure** from the **Debug** menu.



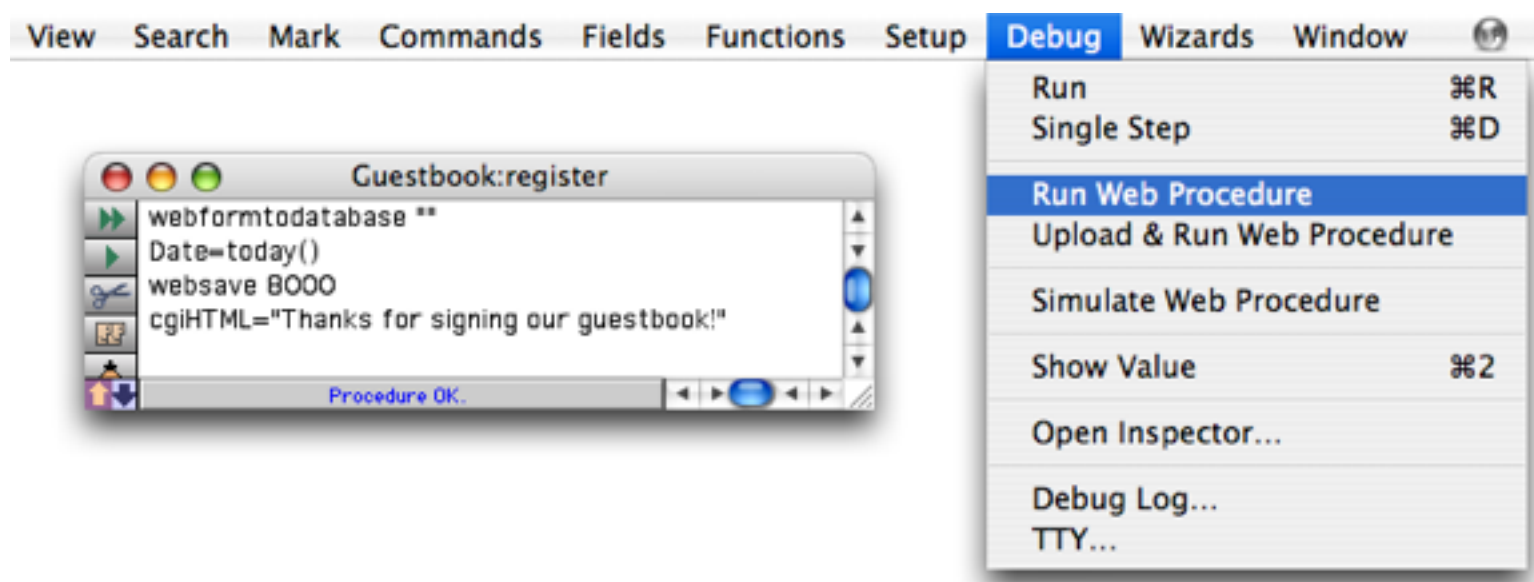
Panorama will upload the revised procedure to the server, then open your web browser and trigger the URL necessary to test the procedure. Once again, the results will appear in a new browser window.



Note: If you want to upload a revised procedure to the server without testing it, use the **Upload Web Procedure** command from the **Setup** menu.

Testing a Procedure Assigned to a Web Form

To test a procedure assigned to a web form, first open the procedure, then choose **Run Web Procedure** from the **Debug** menu.



Instead of running the procedure directly, Panorama will open the web form in a new browser window. (This is because the procedure has been assigned to the form, so it doesn't make sense to run the procedure directly.)

Welcome

http://localhost/cgi-bin/Panorama.cgi?Guestbook~form~We

Please Sign Our Guestbook!

Name:

City:

State:

Email:

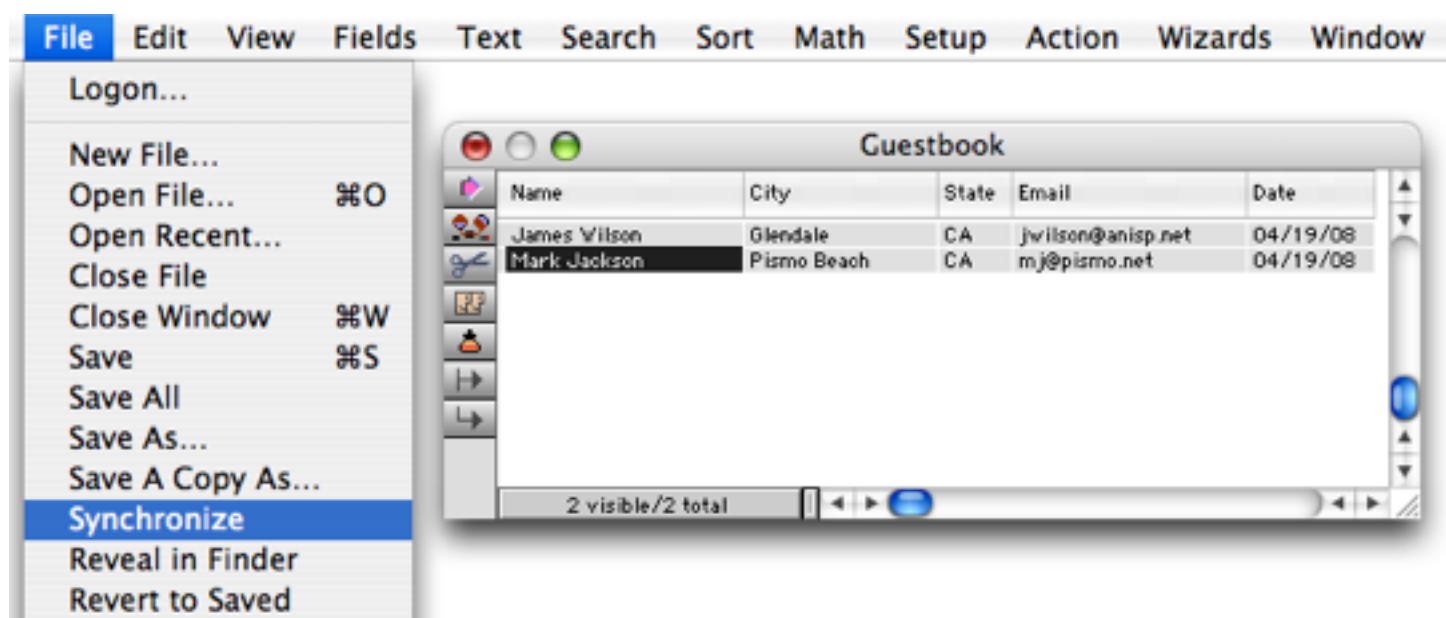
Once the form appears, fill in the blanks (as shown above) and press **Submit** to complete the test. This will trigger the web procedure assigned to the form (in this case the [register](#) procedure). The results are displayed in your web browser.

Panorama Database Query:Guestbook/register

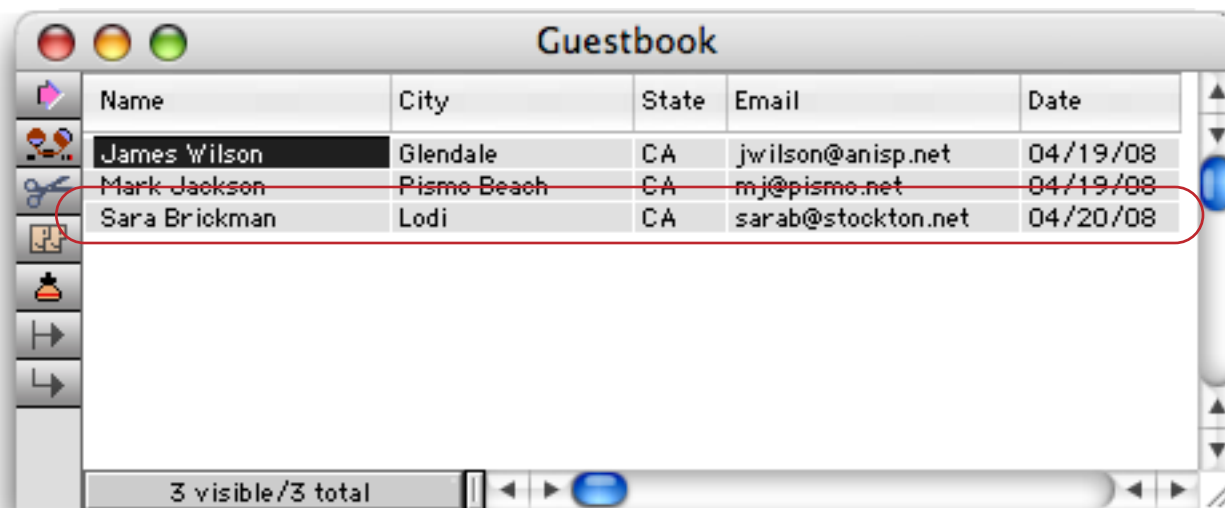
http://localhost/cgi-bin/Panorama.cgi?Guestbook~regist

Thanks for signing our guestbook!

In this case the web procedure has added a new record to the server database. To see this new record, bring up the client copy of the Panorama database and choose **Synchronize** from the File menu.



The new record will appear at the end of the data sheet view. (The same process can work in reverse, allowing you to create and edit data using Panorama that then shows up on the web. Synchronization is not required to allow Panorama edited data to appear in the web interface — any changes made using Panorama will immediately show up in new web pages.)

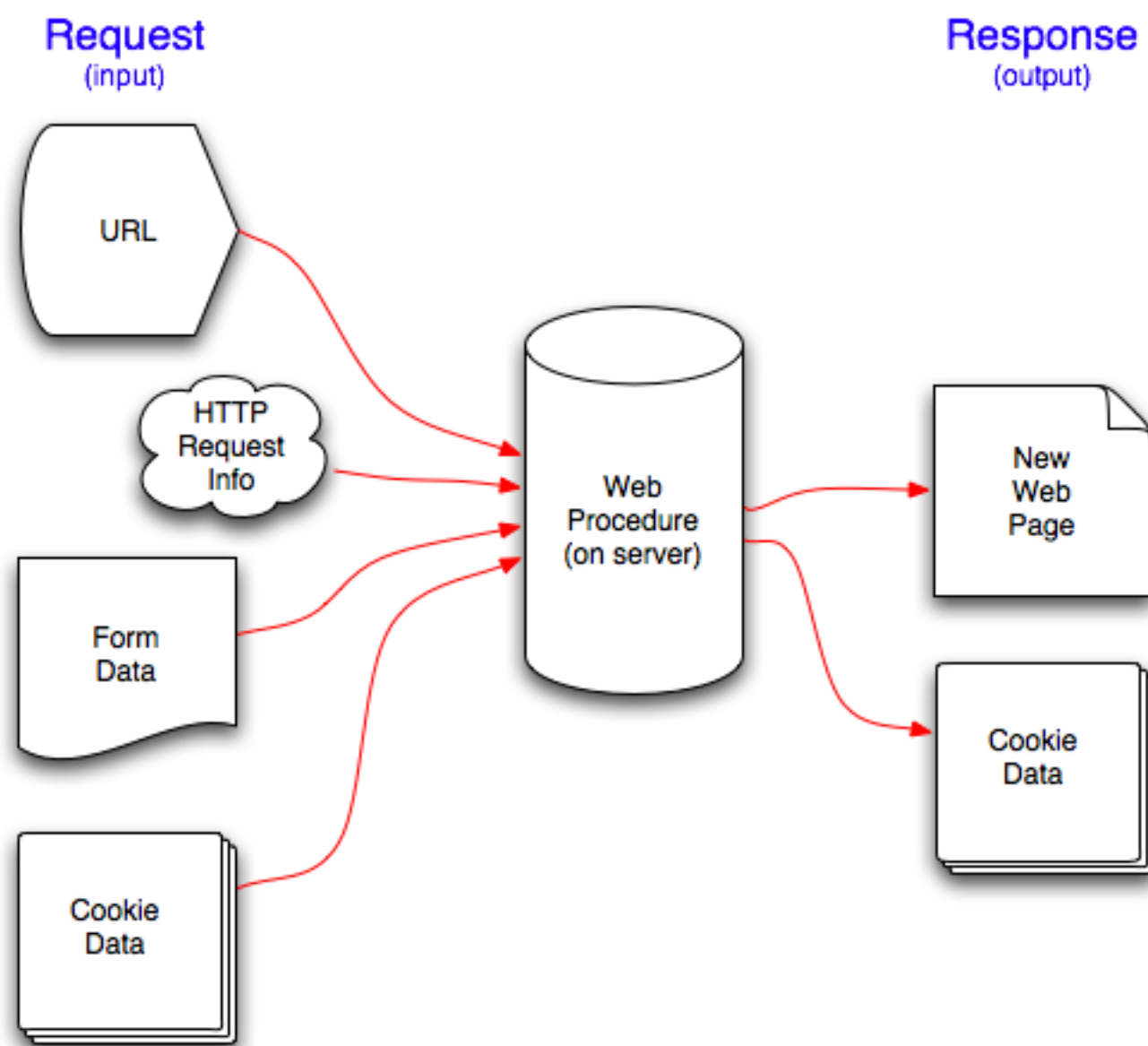


| Name | City | State | Email | Date |
|---------------|-------------|-------|--------------------|----------|
| James Wilson | Glendale | CA | jwilson@anisp.net | 04/19/08 |
| Mark Jackson | Pismo Beach | CA | mj@pismo.net | 04/19/08 |
| Sara Brickman | Lodi | CA | sarab@stockton.net | 04/20/08 |

Now that you've got a taste of how Panorama Server web procedures work we'll turn to a more detailed explanation of the nuts and bolts of creating and deploying these procedures.

Web Procedure Inputs and Outputs

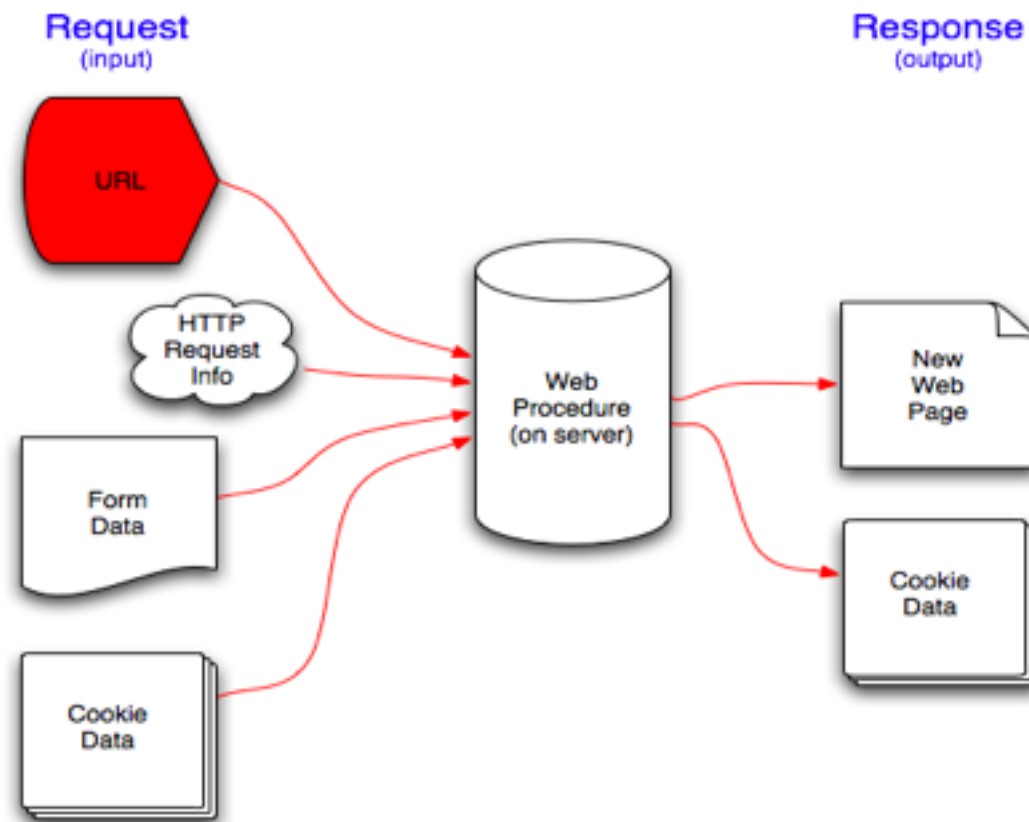
The primary job of a web procedure is to take a request from a web browser and produce a response that is sent back to the web browser (the web procedure may perform other jobs as well, such as modifying a data-base). As shown in the diagram below, the request may consist of up to four different components: the URL, HTTP request information, form data, and cookie data. The response may consist of up to two components, a web page, and cookie data (the response can also be an error).



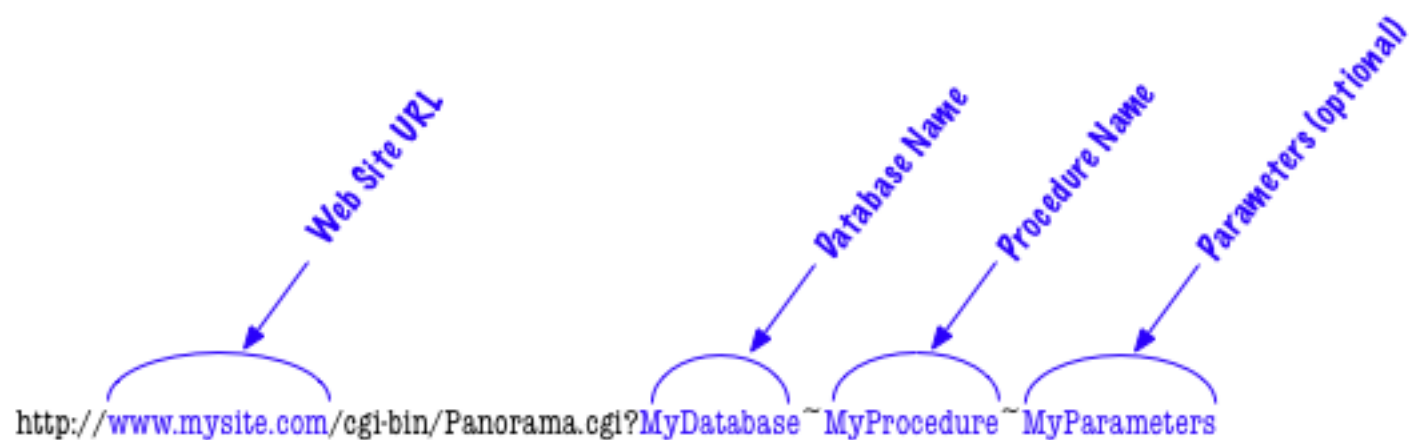
The following sections will introduce you to each of these six components in greater detail.

Web Procedure URLs

The primary input to any web procedure is the **URL**. This input component is required — without a URL your web procedure will never run.



Each database you upload to your Panorama Server may contain one or more procedures. Each of these procedures is specified by its own unique URL. The URL contains the location of the server (ip address or domain name), the database name, and the name of the procedure. The URL can also contain additional information that the procedure can access. Panorama Server URL's must always be written using the format shown in this diagram:



The text in black is always the same. The colored sections change depending on what procedure you want to trigger. Here are some examples of typical URL's for web procedures:

```
http://www.mydomain.com/cgi-bin/panorama.cgi?Checkbook~Balance
http://66.245.219.3/cgi-bin/panorama.cgi?GuestBook~RecentVisitors
http://192.168.1.12/cgi-bin/panorama.cgi?Shopping~displaycart
```

Upper and Lower Case Characters in Procedure Names and Extra Parameters

For the most part you don't need to worry about upper vs. lower case in the URL. The domain name, cgi-bin/panorama.cgi, and the database name may all be any mixture of upper and lower case. However, the procedure name in the URL must exactly match the procedure name in the database. For example, consider the second URL in the examples from the previous section. Assuming the procedure is named **RecentVisitors**, all of these URL's would work fine to specify this procedure:

```
http://66.245.219.3/cgi-bin/panorama.cgi?GuestBook~RecentVisitors
http://66.245.219.3/cgi-bin/panorama.cgi?guestbook~RecentVisitors
http://66.245.219.3/CGI-BIN/PANORAMA.CGI?GUESTBOOK~RecentVisitors
```

These URLs, however, will not work.

```
http://66.245.219.3/cgi-bin/panorama.cgi?GuestBook~recentvisitors
http://66.245.219.3/CGI-BIN/PANORAMA.CGI?GUESTBOOK~RECENTVISITORS
```

Be careful that the case of the procedure name in the URL exactly matches the actual case of the procedure name.

In many cases any extra parameters used after the procedure name may also be case sensitive. This depends on how the procedure uses the extra parameters (see “[URL Extra Parameters](#)” on page 332). It's usually best to assume that anything after the database name is case sensitive.

Database and Procedure Names Containing Spaces

What if the database name or procedure name has one or more spaces? It turns out that the web doesn't like URL's with spaces in them. To get around this, use **%20** (the hexadecimal representation of a space) anywhere you would normally put a space. For example, suppose you have a database called **Real Estate** with a procedure called **New Listings**. To access this procedure from a web browser you would use a URL like this:

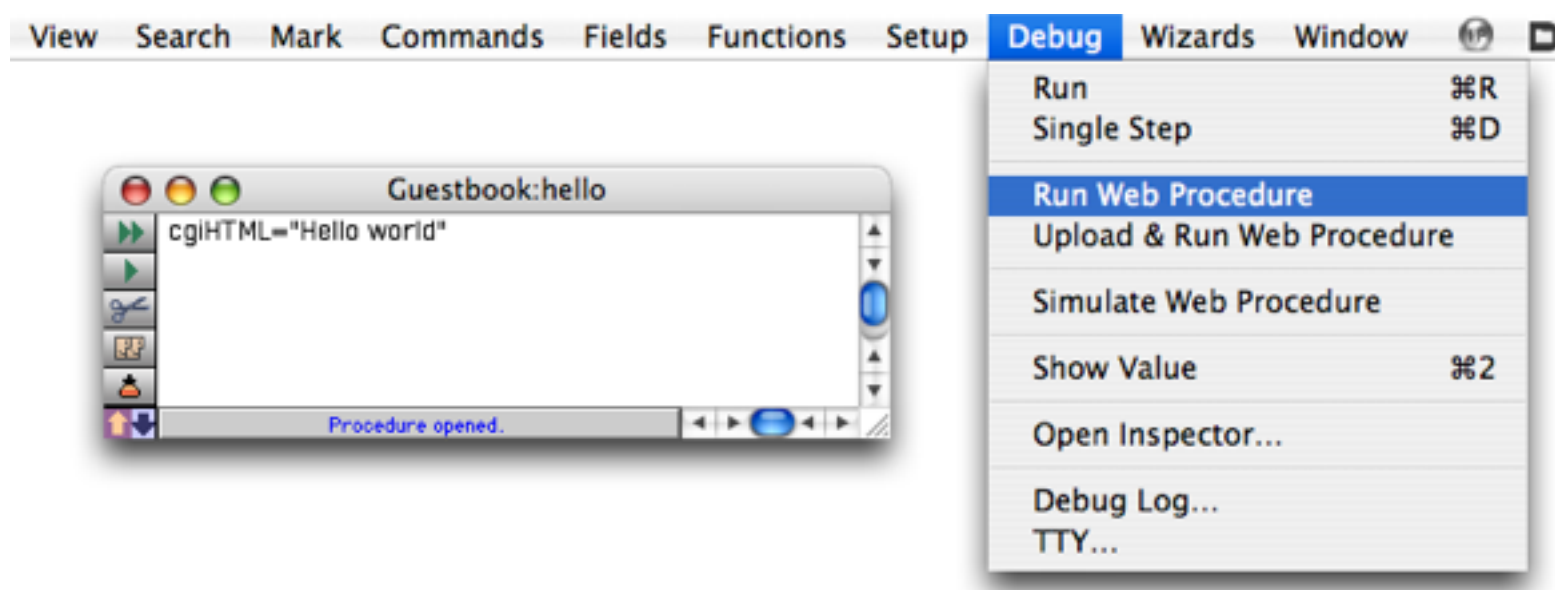
```
http://www.mydomain.com/cgi-bin/panoarma.cgi?Real%20Estate~New%20Listings
```

Regular text can automatically be converted into this special format by using the **urlencode()** function. Use this function if you need to turn text in a field or variable into a URL — it will automatically convert any spaces to **%20** for you. This is especially useful for constructing URL extra parameters (see “[URL Extra Parameters](#)” on page 332).

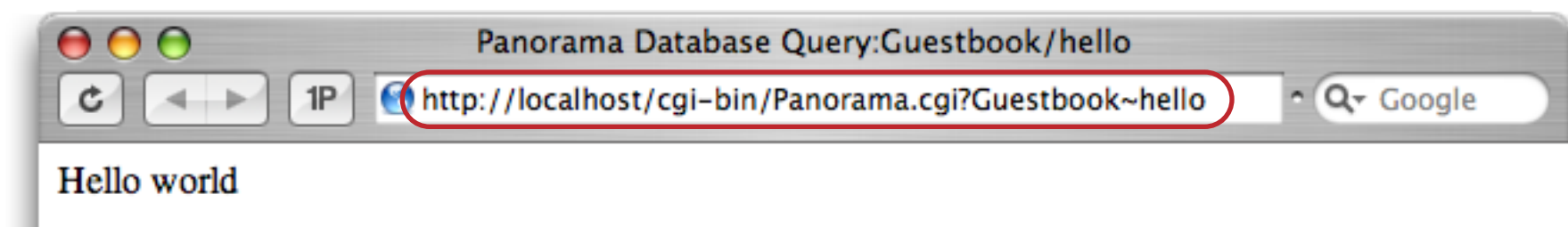
Generating a Web Procedure URL Without Typing

You can create a Web Procedure URL by manually typing it using the guidelines above, but that's the hard way. The easy way is to let Panorama figure out the link for you. There are a couple of ways to do that.

Run Web Procedure in the Debug menu. If you have a client copy of the web database on your computer the simplest technique is to open the procedure and then choose **Run Web Procedure** from the **Debug** menu.



This command will open your web browser and fill in the URL for that procedure (it will also run the procedure).



Now you can copy the URL into the clipboard or save it as a bookmark.

Server Status Page. Another way to get the URL is to use the Server Status Page from your web browser. This can be done from any computer since it doesn't require that you have a client copy of the database. See "[The Server Status Page](#)" on page 200 for details on using the status page.

Linking to a Web Procedure from Regular Web Pages

You'll often need to include a link to a Panorama Server web procedure from other pages in your web site. Like any other link, this is done with the `<a>` tag. Here are some examples, which could be embedded in any web page using the design tool of your choice — DreamWeaver, GoLive, RapidWeaver, BBEdit, etc.:

```
<a href="http://www.mydomain.com/cgi-bin/panorama.cgi?Checkbook~Balance">Balance</a>
<a href="http://66.245.219.3/cgi-bin/panorama.cgi?GuestBook~RecentVisitors">Recent</a>
<a href="http://192.168.1.12/cgi-bin/panorama.cgi?Shopping~displaycart">Shopping Cart</a>
```

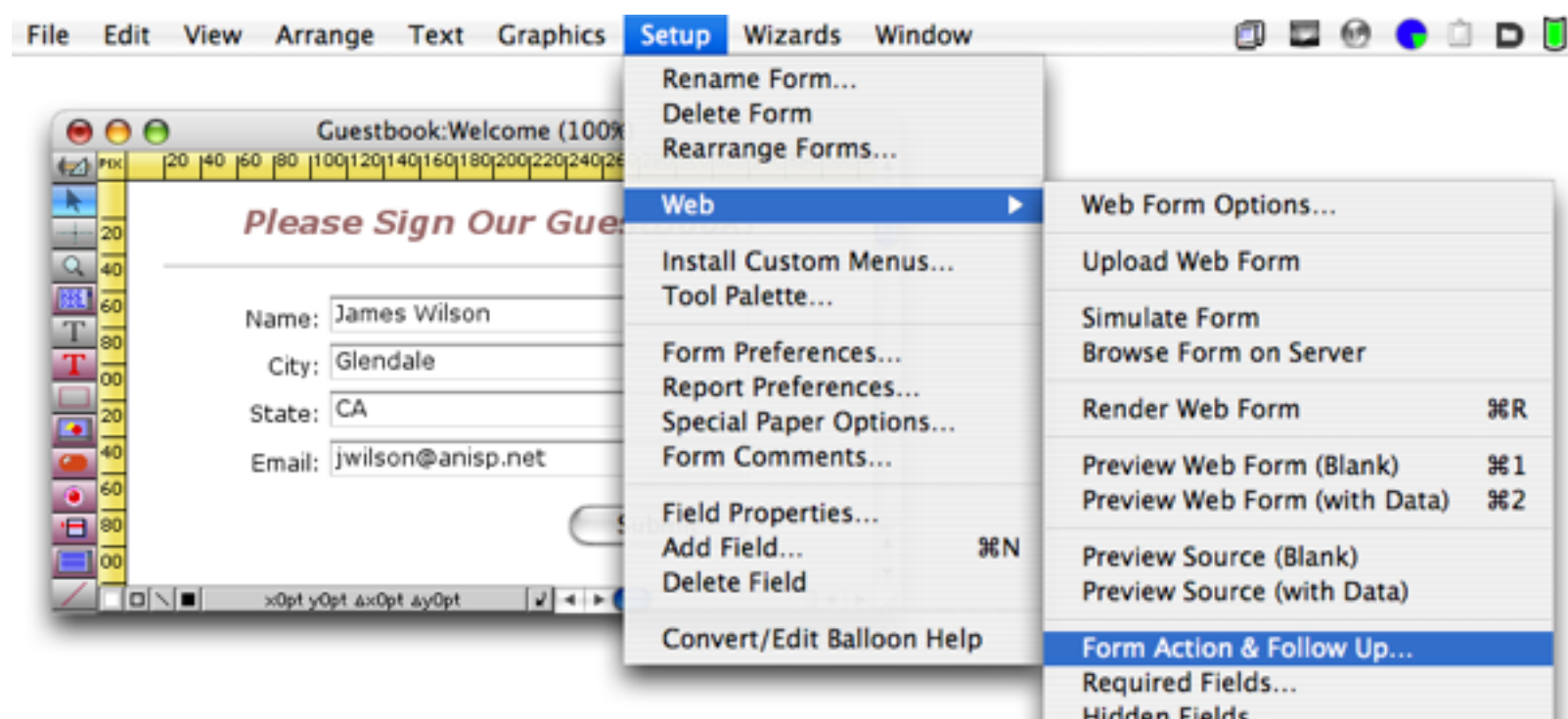
If you want to avoid typing the URL you can generate it using the techniques described in the previous section.

Linking to a Web Procedure from another Web Procedure

It's frequently necessary for a page generated by one web procedure to link to other web procedures. Your web procedure code can manually generate an `<a>` tag as described in the previous section, but an even better idea is to use the special `cgilink()` and `webformlink()` functions that will automatically generate the `<a>` tag for you. See "[Links to Other Web Pages](#)" on page 378 for information on how to use these functions.

Linking to a Web Procedure from a Form

Panorama web procedures can be used to process the data when a form's submit button is pressed. To do this the HTML code for the form must specify the Panorama web procedure as the designated action for the form. If you are using a Panorama web procedure this is taken care of when you use the **Form Action & Follow Up** command to assign the procedure to the form (see "[Assigning a Procedure to the Form](#)" on page 320).



If you want to use a form developed in some other tool (DreamWeaver, BBEdit, etc.) then you'll need to install the link in the **action** parameter of the `<form>` tag, like this:

```
<form action="http://192.168.1.7/cgi-bin/Panorama.cgi?Guestbook~register" method=post>
```

If the form page will be hosted on the same server as the Panorama database you can omit the server location at the beginning of the URL, as shown here:

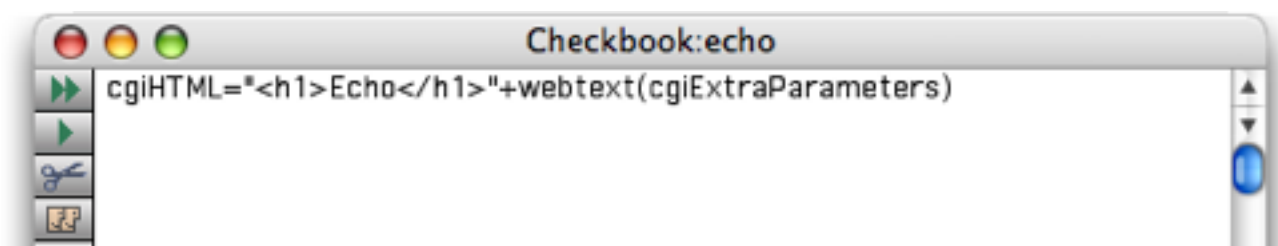
```
<form action="/cgi-bin/Panorama.cgi?Guestbook~register" method=post>
```

URL Extra Parameters

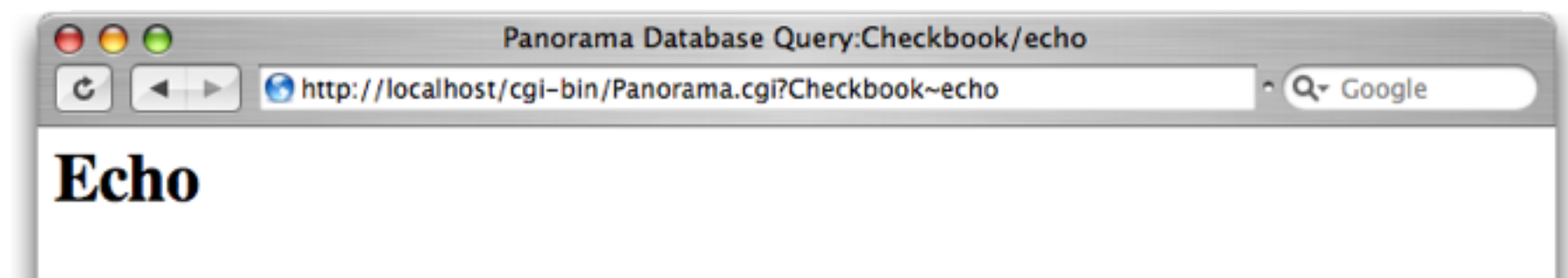
In addition to the database and procedure name, the URL can also contain additional information at the end. This extra information must be separated from the procedure name with the `~` character. The extra information, also called extra parameters, is completely ignored by the Panorama Enterprise server. Here are some examples of URLs with extra parameters (for illustration purposes the extra parameters are shown in orange).

```
http://66.245.219.3/cgi-bin/panorama.cgi?GuestBook~RecentVisitors~2weeks
http://66.245.219.3/cgi-bin/panorama.cgi?GuestBook~RecentVisitors~45days
http://66.245.219.3/cgi-bin/panorama.cgi?Orders~ShowInvoice~78219
http://66.245.219.3/cgi-bin/panorama.cgi?Orders~ShipInvoices~78219~78232~78237
```

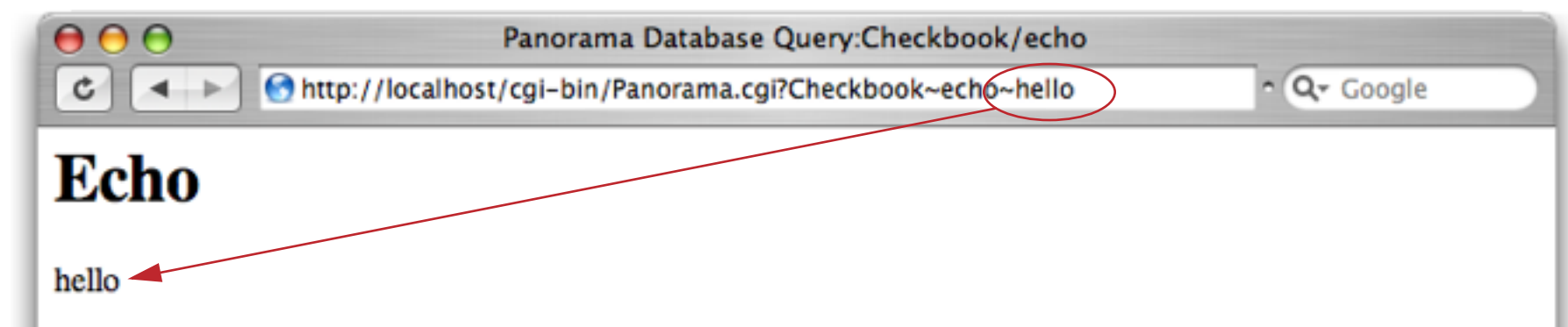
Your web procedure can access this information (it is in the `cgiExtraParameters` variable) and use it any way it you need it to. Here is a very simple web procedure that simply repeats the extra parameter data to the web page.



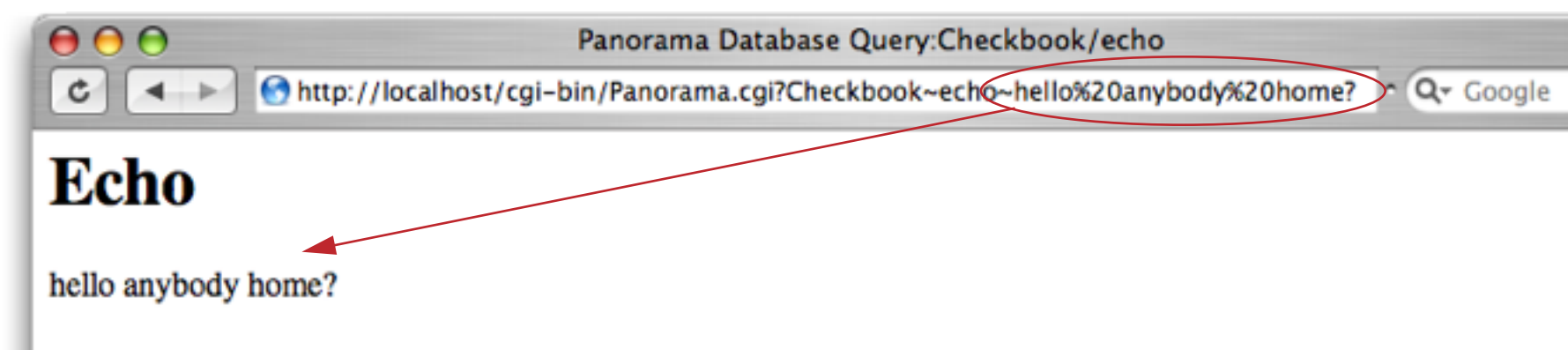
Choose the **Upload & Run Procedure** command in the **Debug** menu to try out this procedure. Since no additional parameters were supplied on the end of the URL nothing appears after the word **Echo**.



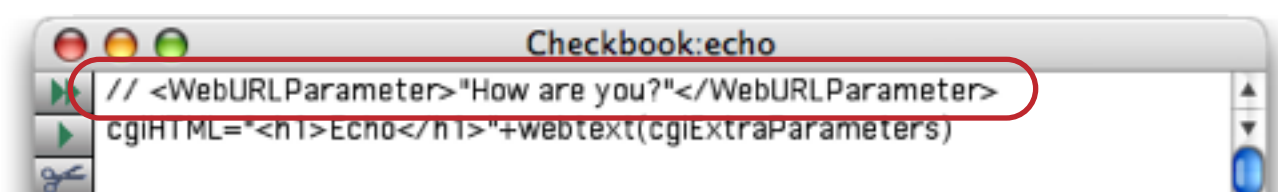
Now let's add some text to echo. First add a `~` symbol, then the text to repeat. Press **Enter** to run the procedure again.



If you want to include spaces or other punctuation you must escape them using standard URL hex notation (see “[Database and Procedure Names Containing Spaces](#)” on page 330).

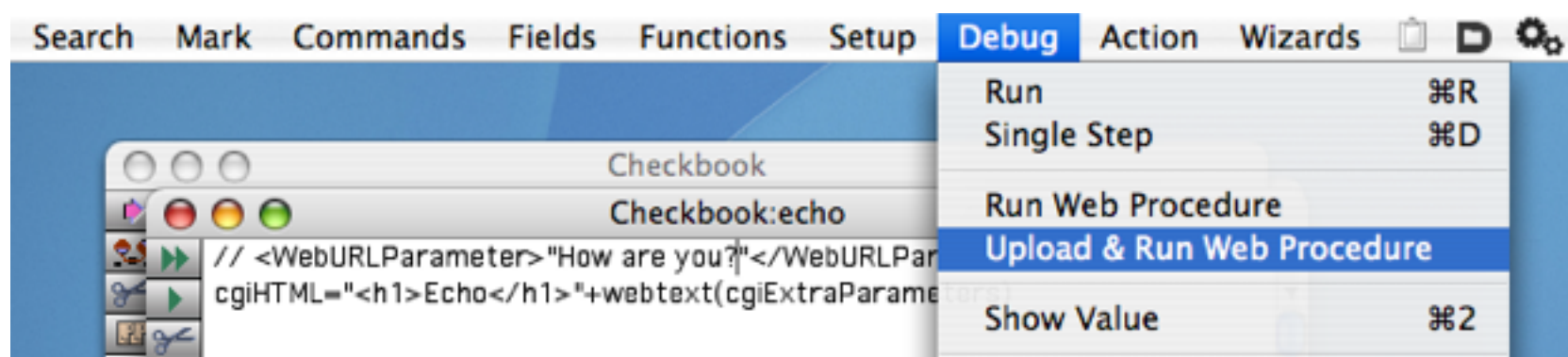


Testing Procedures with Extra URL Parameters. Earlier in this chapter you learned how to test procedures with the **Debug** menu, both with the simulator and on a live server. These same **Debug** menu commands can be used to test web procedures that require extra parameters. To do this you’ll need to embed the extra URL data you want to test with into a comment on the first line of the procedure. This comment must contain a `<WebURLParameter></WebURLParameter>` tag, with a formula in between the tags. Here’s a modified version of the echo procedure with an embedded tag.

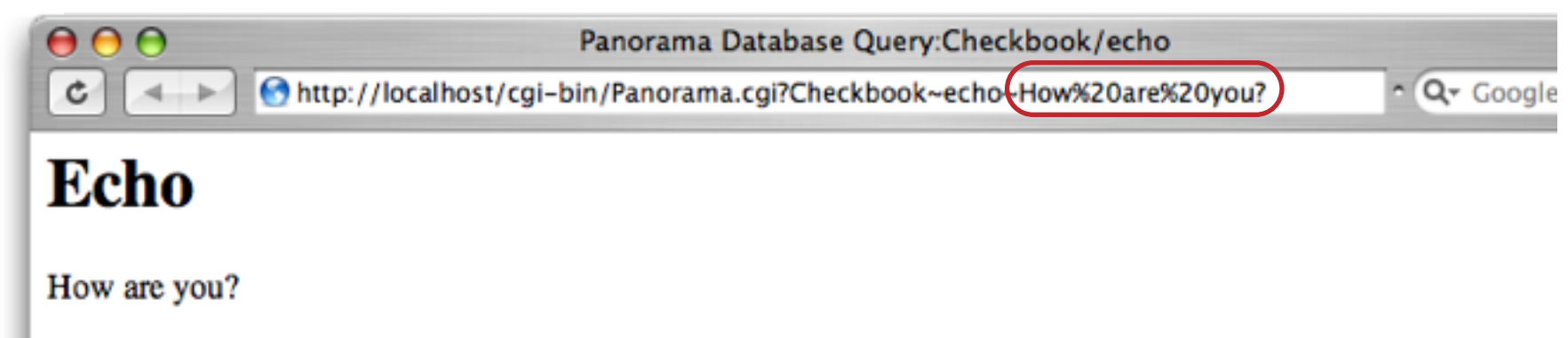


When you use the **Run Web Procedure**, **Upload & Run Web Procedure** or **Simulate Web Procedure** commands in the **Debug** menu Panorama will check for the presence of this special comment and tag. If it finds such a comment it evaluates the formula in the tag and uses that as the extra URL parameter for the purposes of the test. (You don’t have to worry about spaces in the parameter, Panorama automatically converts them to `%20` for you.)

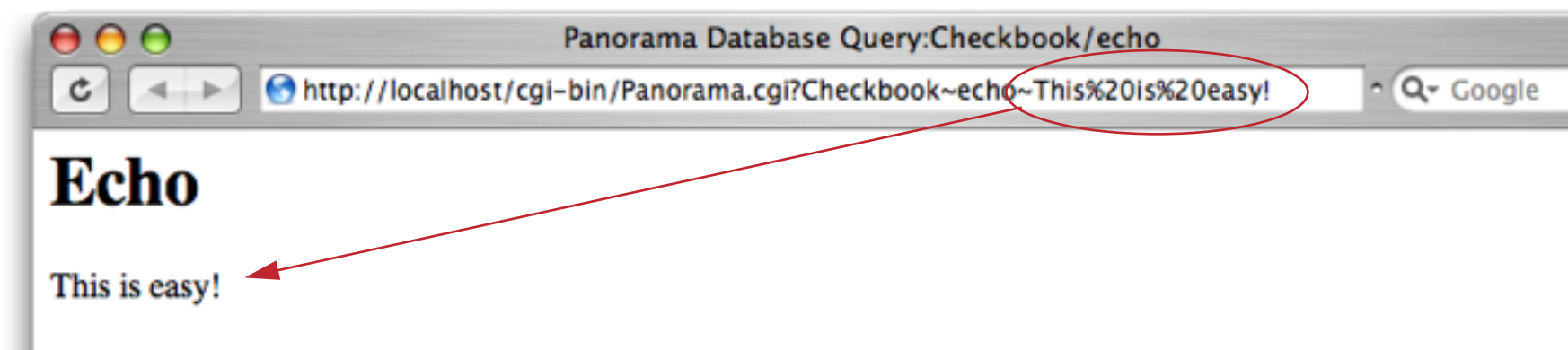
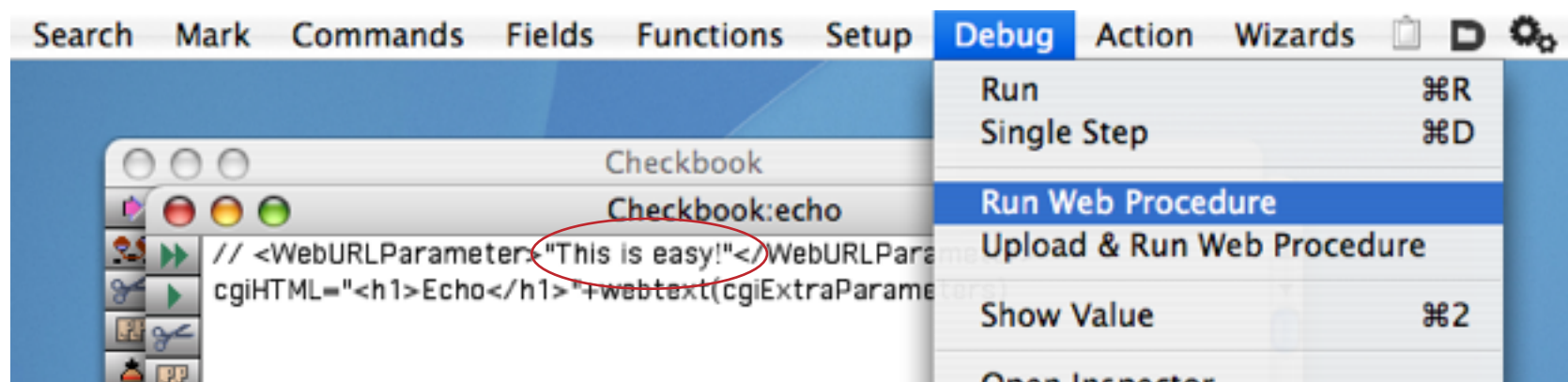
For example, suppose you use the **Upload & Run Web Procedure** command in the **Debug** menu to test this echo procedure.



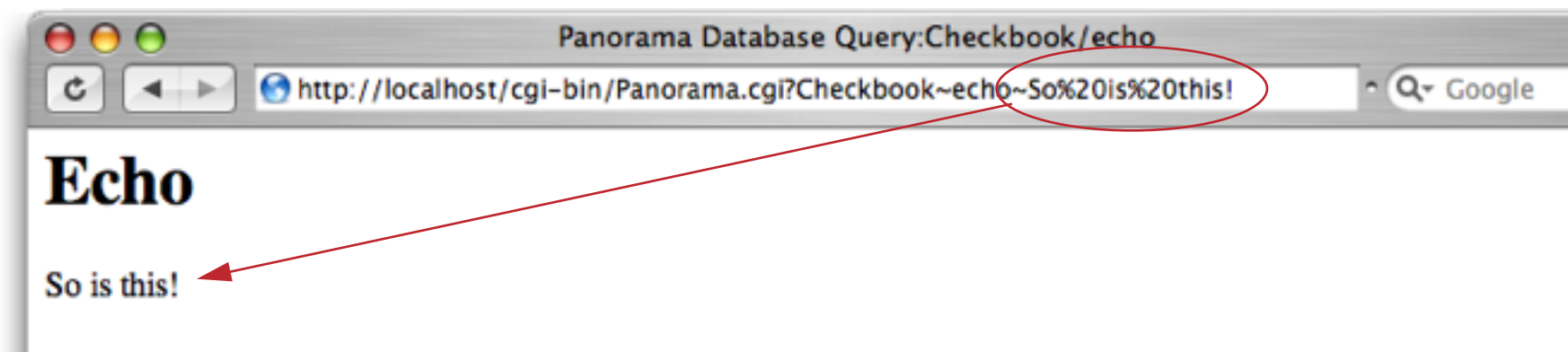
Panorama will automatically add `How are you?` to the end of the URL. (Notice that the spaces are automatically encoded in hex URL format for you.)



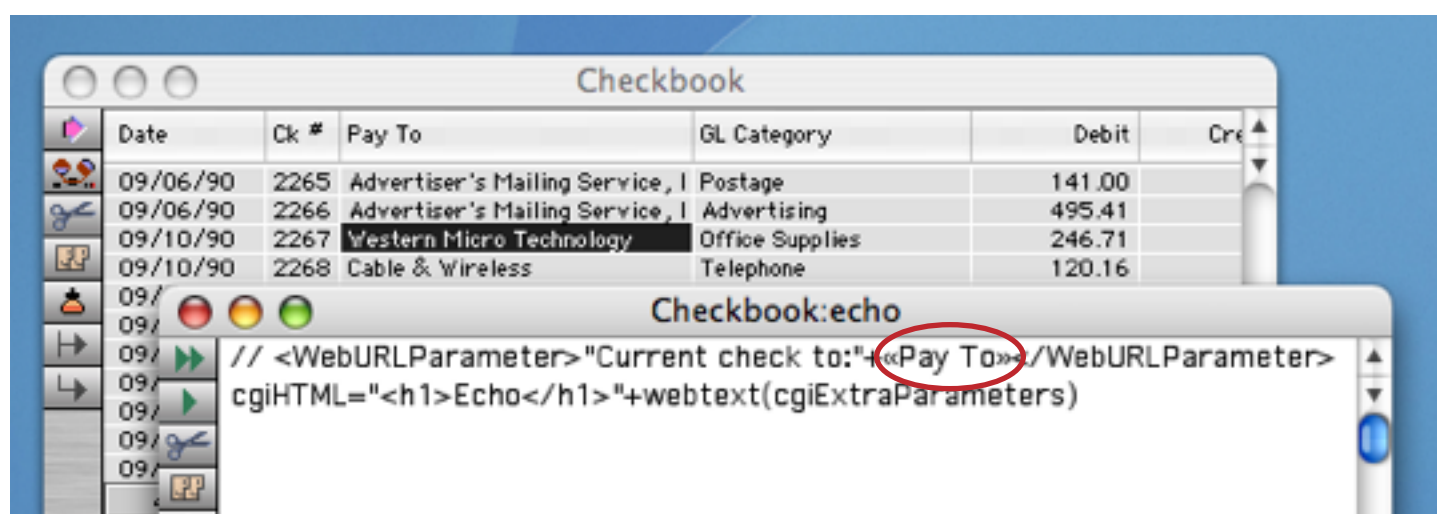
If you change the embedded parameter but nothing else you don't need to re-upload the procedure — just choose Run Web Procedure from the Debug menu.



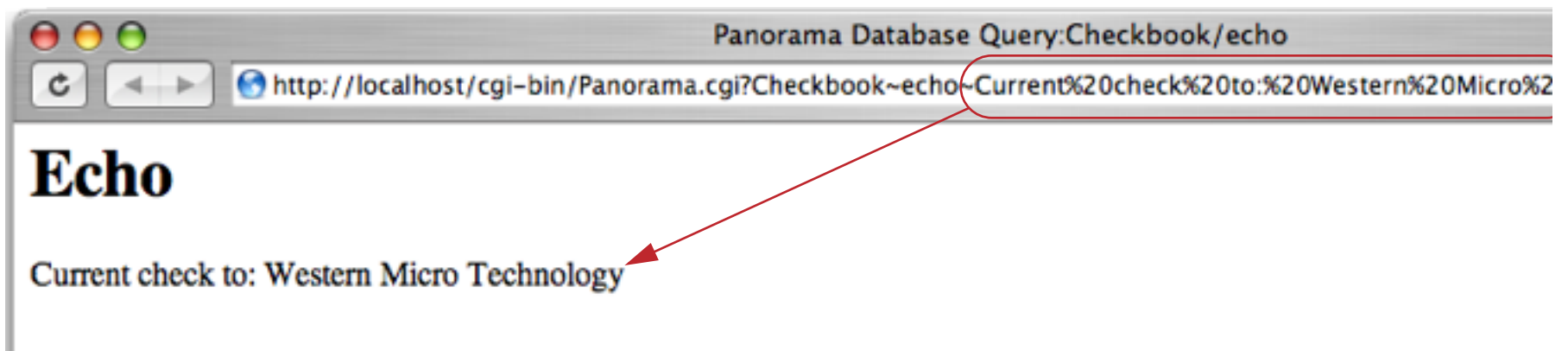
Of course you can also type in whatever you want directly into the URL bar of the browser (the procedure ignores the embedded parameter in this case).



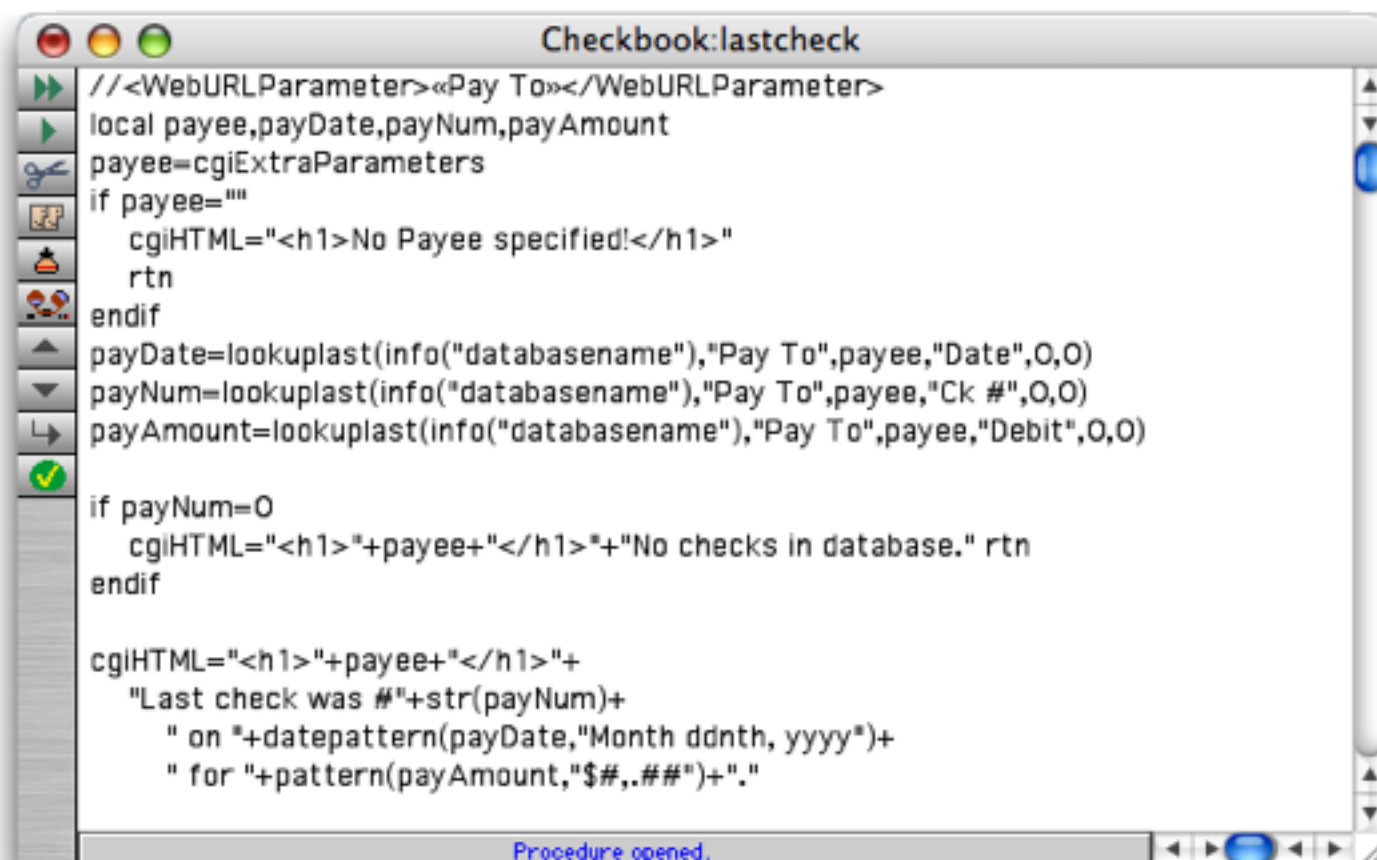
You are not limited to constant values for the extra URL parameter, in fact, you can use any valid Panorama formula. The example below includes a field from the database (**Pay To**) in the formula.



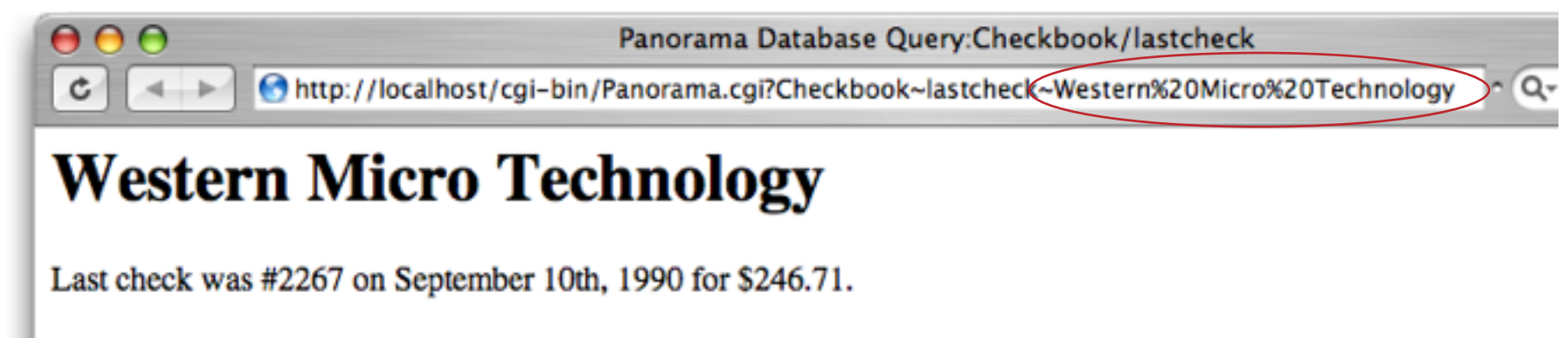
Choose **Run Web Procedure** from the **Debug** menu to see the result. Notice that we haven't actually changed the procedure itself, just the embedded formula for testing (so you don't need to upload to server).



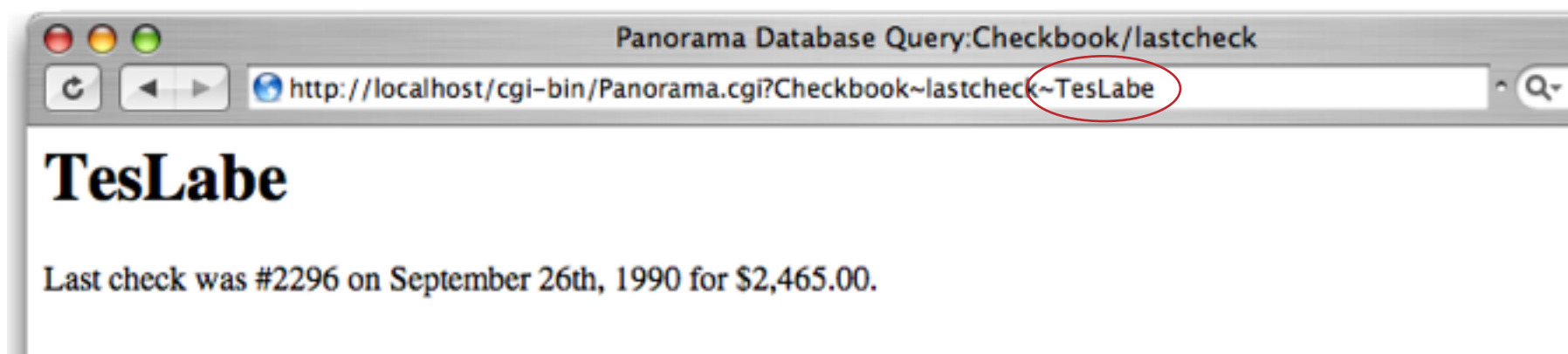
Before leaving this topic let's look at a more realistic example.



Choose **Upload & Run Web Procedure** from the **Debug** menu to see the result of this web procedure.



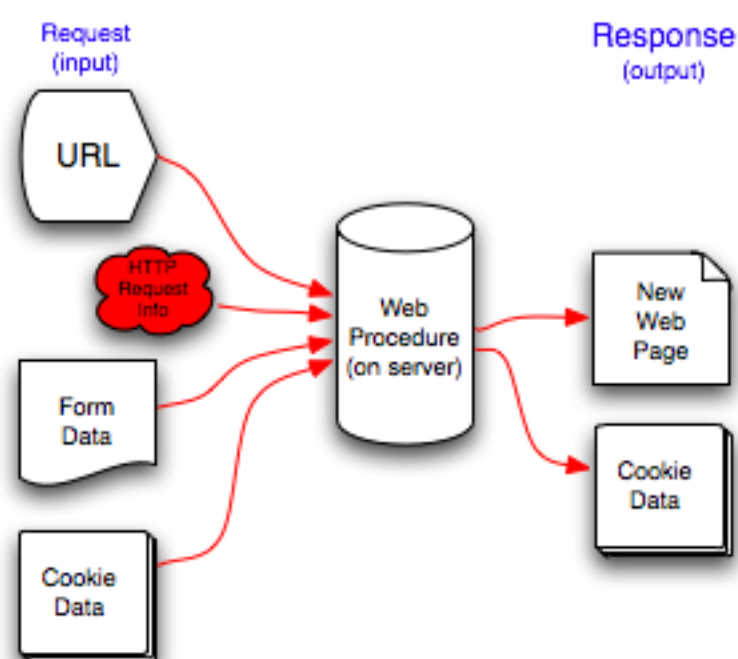
The previous example used the embedded formula in the procedure, but of course you can also type the parameter directly into the URL bar in your browser.



These examples have used the **Run Web Procedure** command, but this embedded extra parameter trick works with the **Simulate Web Procedure** command as well.

HTTP Request Information

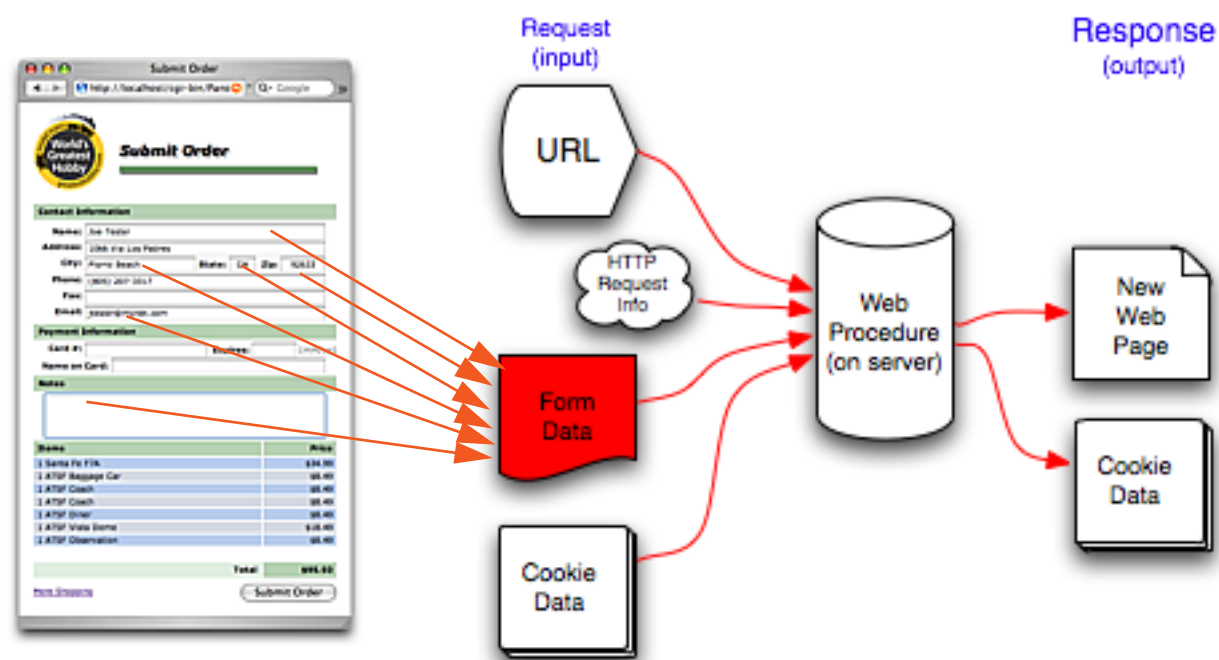
A second component of the request from the web browser is the **HTTP Request Information**. This information includes a variety of items, including the type of web browser the user is using, the users IP address and port, what language they prefer (English, French, German, etc.), and much more.



The web procedure can access all of this information using the `webserverinfo` statement. For all of the details see "[Accessing Additional Web Server Information](#)" on page 460.

Form Input Data

If a web procedure is assigned to a web form, any input data in the form will be passed as input to the web procedure.



An HTML web form contains one or more items that the user can fill in. Each item has two components, a **name** and a **value**. Each item's **name** is set up by the form designer and usually describes what the item is for (for example Company, State, Zip, Phone, etc.). When you are designing a form to use with a Panorama database it's usually a good idea to give each form item the same name as the corresponding field in the database. Each item's **value** is the text or number the user enters into the item, so the value will change each time the form is used.

When the user presses the **Submit** button the browser collects all the form items (name/value pairs) and sends them to the server. On the server, a web procedure can access the form items (name/value pairs) with the `webformitem`, `webformitemnames`, `webformselection` and `webformtodatabase` statements.

Accessing Form Item Values in a Formula

The `webformitemvalue()` function gets the value of any form item. It has one parameter, the name of the form item.

```
webformitemvalue(item)
```

This function can be used in any Panorama formula in your web procedure.

```
cgiHTML=webformitemvalue("Name")+" lives in "+
webformitemvalue("City")+", "+webformitemvalue("State")
```

The output of this function is always text.

Assigning a Form Item Value into a Field or Variable

The `webformitem` statement gets the value of any form item, then assigns that value to a variable or a database field.

```
webformitem item,fieldorvariable
```

The *name* parameter is the name of the form item you want. The *fieldorvariable* parameter is the name of a field or variable where the value of this item should be stored. For example here is a simple procedure for adding a new record to a web based guestbook.

```
addrecord
webformitem "Name",Name
webformitem "City",City
webformitem "State",State
webformitem "Country",Country
webformitem "Email",Email
webformitem "Name",Name
```

See “[Assigning Multiple Items into Multiple Fields and/or Variables](#)” on page 435 for an even better way to write this procedure.

Note: The `webformitem` statement will automatically convert data into the proper format for storage into a text, numeric or date field. If the data submitted in the form is not the proper format (for example if the user types `tree` for a number or date field) `webformitem` will simply leave the field or variable empty.

Getting a List of Form Item Names

The `webformitemnames` statement returns a list of the names of all of the form items submitted to the server.

```
webformitemnames names
```

The *names* parameter is the name of a field or variable where the list should be placed. The list is a carriage return separated array. Here is a code fragment that calculates how many form items were submitted.

```
local itemCount,itemNames
webformitemnames itemNames
itemCount=arraysize(itemNames,1)
```

You probably won't use the `webformitemnames` statement much because you usually design the forms yourself and already know the names of the form items.

What Form Is This?

It's possible to write a single web procedure that is assigned to more than one web form. In that case the procedure will need to know what web form is being submitted. If the web form was created in Panorama (see “[Converting a Panorama Form into a Web Form](#)” on page 231) then the procedure can find out what form was submitted using the `webformname()` function.

```
if webformname() = "New"
...
endif
if webformname() = "Renewal"
...
endif
```

If you did not use Panorama to create your forms you'll need to come up with some other method to identify which form is being submitted, perhaps with a hidden field (see below).

Hidden Form Items

In addition to normal visible data values (text editing cells, checkboxes, radio buttons, etc.) web browsers also support hidden values. These values are generated when the web form is displayed, then stored invisibly within the web form and submitted with the other data values when the **Submit** button is pressed. Essentially these hidden values allow the server to pass values to itself, allowing it to remember a value or calculation from a previous interaction with the server. For example a hidden value could contain the time the form was displayed, allowing the server to calculate how long it took you to fill in the form.

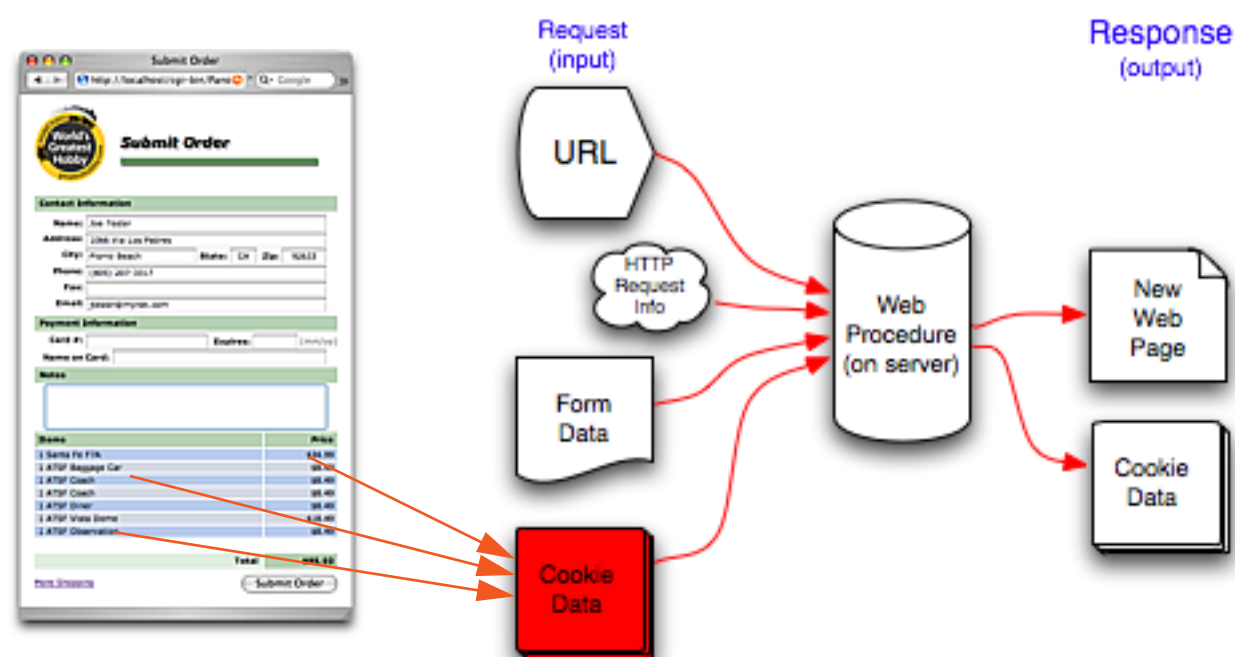
If you are building your web form in Panorama you can use the **Hidden Fields** command in the **Web** sub-menu of the **Setup** menu to create one or more hidden form items (see “[Hidden Data](#)” on page 271). If you are building your web form using some other tool you’ll need to use an `<INPUT>` tag for each hidden item. The type of the tag must be set to hidden, and it must include the name and value of the item. Here are a couple of examples.

```
<INPUT TYPE=hidden NAME=recipient VALUE="mjd@help.com">
<INPUT TYPE=hidden NAME=subject VALUE="Feedback on your help system">
```

Processing Hidden Form Items in a Procedure. In a web procedure, hidden form items are handed exactly like regular visible form items. In fact, there is no way for a web procedure to determine if a form item is visible or hidden. The procedure can use the `webformitemvalue()` function and `webformitem` statement to get the value of any hidden items. The `webformitemnames` statement will return the names of all form items, both visible and invisible.

Cookies

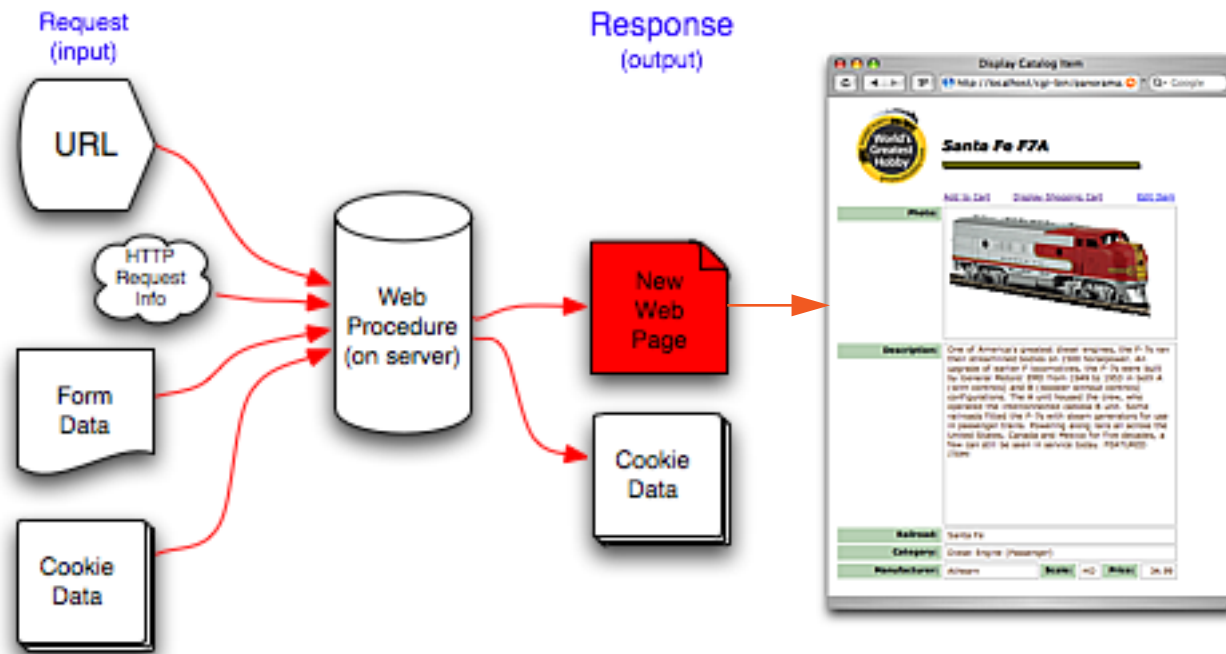
An **HTTP cookie**, or a **Web cookie** (also called simply a **cookie**), is a parcel of text sent by a server to a web browser and then sent back unchanged by the browser each time it accesses that server. HTTP cookies are used for authenticating, tracking, and maintaining specific information about users, such as site preferences and the contents of their electronic shopping carts. Without cookies, each retrieval of a web page or component of a web page is an isolated event, mostly unrelated to all other views of the pages of the same site. By returning a cookie to a web server, the browser provides the server a means of connecting the current page view with prior page views. (This paragraph was adapted from material found on Wikipedia. You can find a lot more at http://en.wikipedia.org/wiki/HTTP_cookie).



The diagram above shows a typical usage of a cookie for a shopping cart. As items are added to the cart they are stored in the cookie on the user’s computer. It’s important to keep in mind that cookie data is stored on the users’s computer, not on the web server. Whenever a form is submitted to the server the web browser also submits all of the cookie data that was stored by that user. The web procedure can access or ignore this data as needed. For more information about accessing cookie data see “[Working with Cookies](#)” on page 453.

Generating a Web Page

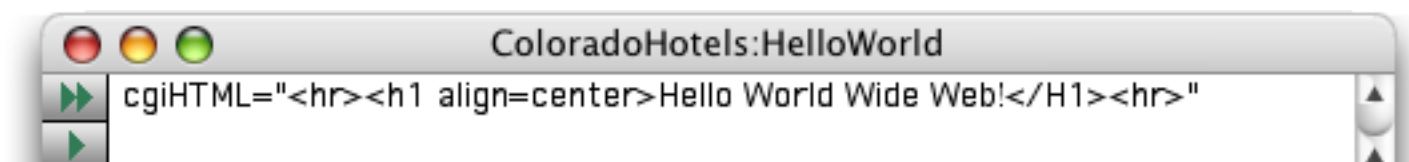
The primary job of any web procedure is to produce an HTML web page that will be displayed on the users web browser.



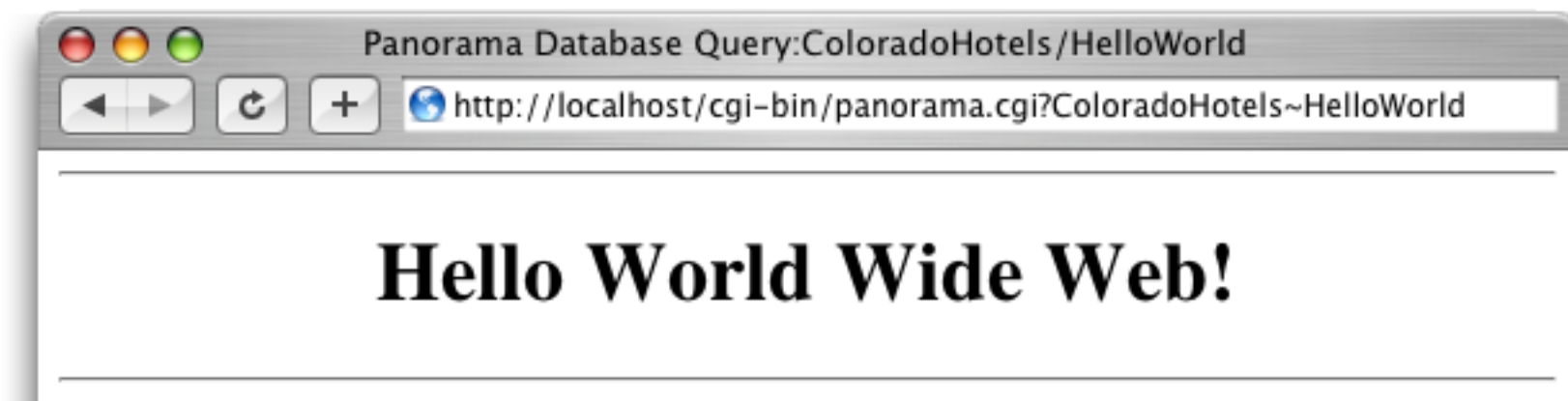
A procedure may do other things as well (for example modify the content of a database) but all web procedures are required to produce a page, or else (see Chapter 10 “[Generating HTML](#)” on page 369)!

The cgiHTML Global Variable

When a request comes in from the web Panorama sets up a global variable named `cgiHTML`. This variable is initially empty (`""`). Whatever text you put into this variable will be passed back to the client web browser. The most basic web procedure can simply consist of one line that assigns some text to this variable.



Here's the result.



Of course most web procedures are more complex than this. In fact, a web procedure can take advantage of all of the features Panorama makes available for writing complex programs, including subroutines, formulas, custom statements, etc. (In fact, Panorama Enterprise includes a number of custom statements and functions to assist in creating complex pages, including statements for automatically generating HTML tables from database fields and arrays. See Chapter 10 “[Generating HTML](#)” on page 369 for more information on these special capabilities.)

HTML (HyperText Markup Language)

Whatever text your procedure puts in the `cgiHTML` variable will be passed back to the users web browser for display. Unless you've specified otherwise, the web browser will interpret and display the text based on the HTML markup language (it is possible to tell the browser to display the text as-is with no markup, see "[Non HTML Content Types \(text/plain\)](#)" on page 461). HTML uses embedded tags to control the appearance of the text, for example `` to start bold text and `` to end bold text.

Depending on the browser on the user's computer there are dozens of different tags available that can be included in web pages generated by Panorama web procedures. Knowledge of at least basic HTML is a prerequisite for effectively writing web procedures. There is a wealth of learning and reference material about HTML available both on-line and in books, so this material is not covered here. On the web, a good place to start is:

<http://www.w3schools.com/html/default.asp>

Another site we've found useful is:

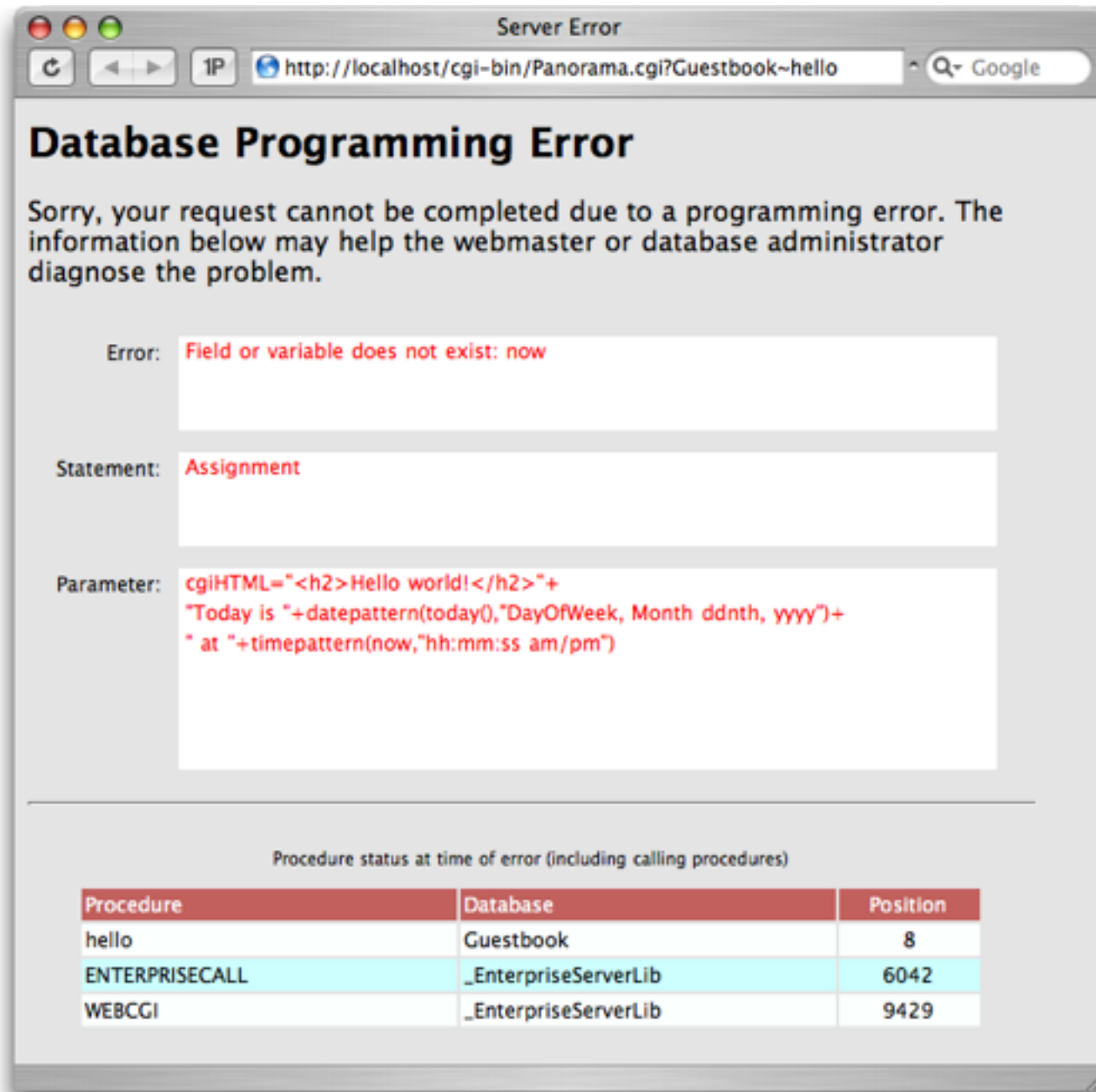
<http://www.htmlhelp.com/>

You can find much more simply by using Google to search for `html`. Or, if you'd rather read a book, search Amazon for `html` and you'll find well over 100 titles.

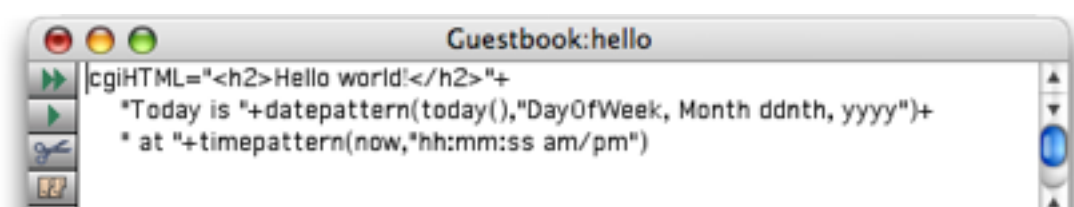
JavaScript and CSS. You can put any kind of text that a browser will understand into the `cgiHTML` variable. This means that web procedures aren't limited generating HTML, but can also include "web 2.0" technologies like JavaScript and CSS. In general these technologies aren't covered in this manual, but there is lots of material available in books and on the web.

Web Procedure Errors

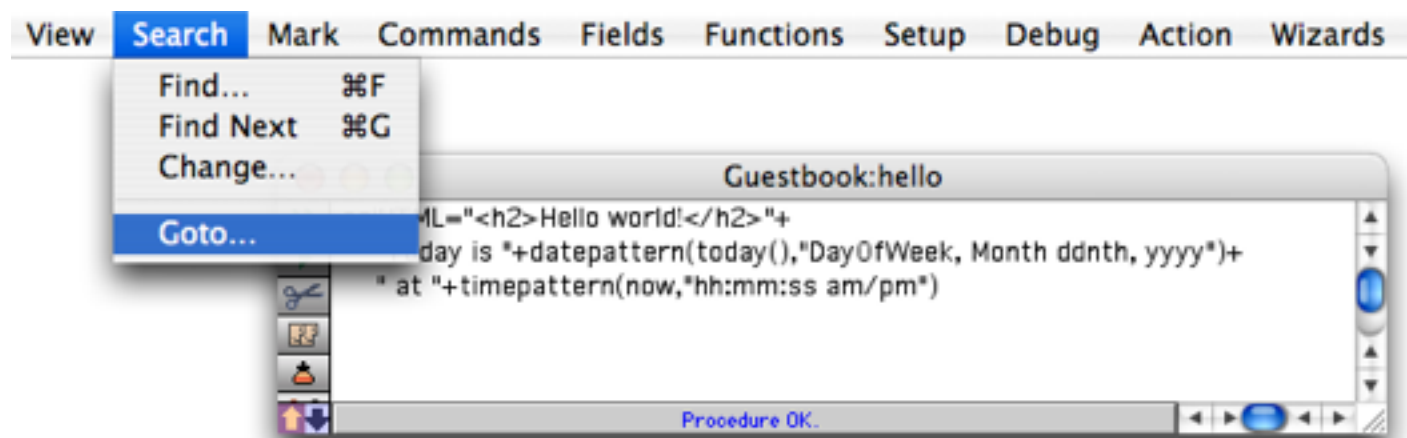
There are three possible types of errors that can occur when running a Panorama web procedure. 1) An error will occur if the database specified by the URL doesn't exist on the server, 2) An error will occur if the procedure specified by the URL doesn't exist on the server, or 3) The procedure itself contains a programming error (for example it tries to access a variable that doesn't exist). If any of these situations occurs the server will respond with a web page describing the error. Here is an example:



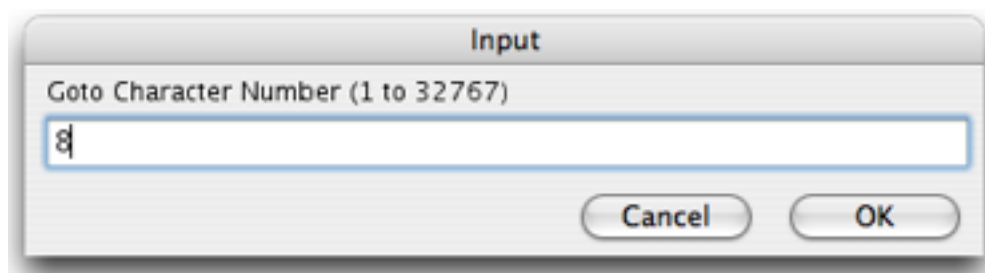
The information displayed on this page can be used to help track down the problem. The table at the bottom of the page shows the procedures that were running at the time the error occurred. The bottom two lines of this table will always be **WEBCGI** and **ENTERPRISECALL**, which are part of Panorama Enterprise itself. The lines above this are your code. In this case the error occurred in the **hello** procedure in the **Guestbook** database, so open this procedure to see what is inside.



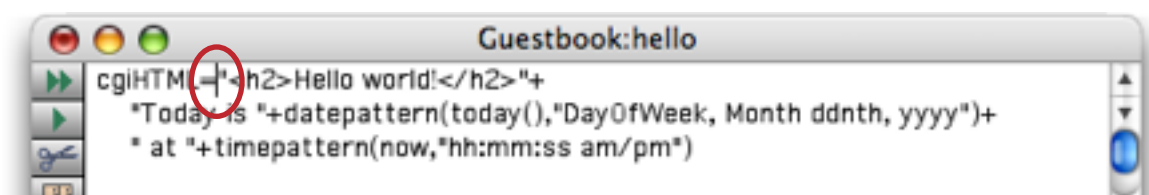
The table in the error page indicates that the problem is at position 8 in this procedure. To find this spot use the Goto command in the Search menu.



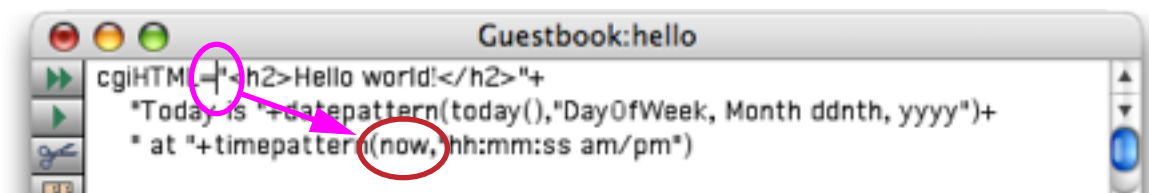
Enter the position you are looking for, in this case 8.



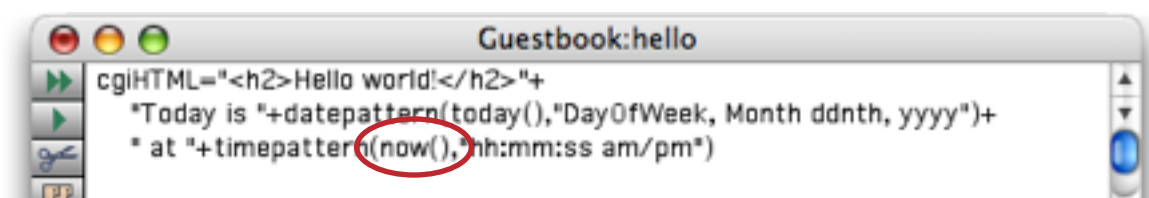
Press OK to see where the problem occurred.



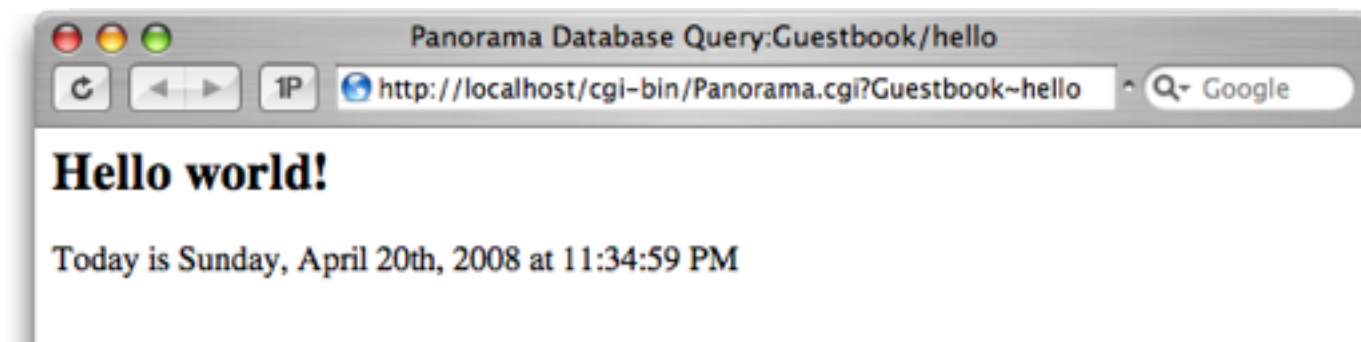
The problem location is often only approximate. In this case it is identifying the beginning of the formula that contains the error. The actual error is a couple of lines down in this formula. The error message itself provides an important clue — it says that a field or variable named `now` does not exist. Looking through the formula shows where the problem is.



Aha! This isn't supposed to be a variable, it is supposed to be the `now()` function. Adding the parentheses fixes the problem.



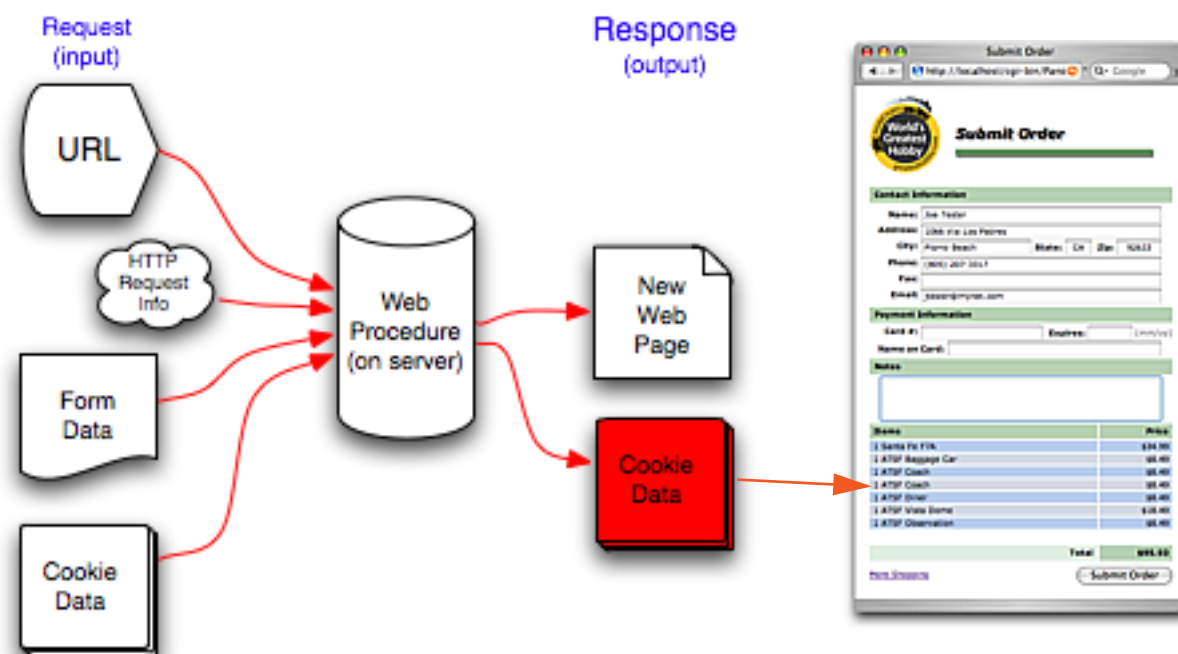
With this fix the web procedure gives the desired result:



The standard error display page is helpful for tracking down problems, but you may not want to display that much information on a production web site. You can customize the way Panorama Server responds when an error occurs to display as much or as little information as you want. To learn more about this see “[Custom Handling of Programming Errors](#)” on page 462.

Cookie Output

Cookies, a mechanism for storing and retrieving data on the client machine, were introduced earlier in this chapter (see “[Cookies](#)” on page 339). If a web procedure needs to save one or more items of data for later use (for example the contents of a shopping cart) it needs to write that data into a cookie.



The diagram above kinds of makes it look like the cookie data is displayed on the web page, but that is not correct. Cookies are stored invisibly on the client user’s computer, then sent back to the server when a later request is made. To learn more about how to use cookies with Panorama Enterprise server see “[Working with Cookies](#)” on page 453.

Getting Procedures onto the Server

There are two methods that can be used to get one or more procedures onto the server computer: 1) Prepare the procedures in advance, then upload them with the **Database Sharing Options** wizard when the database itself is uploaded to the server (or re-uploaded via a new sharing generation), or 2) Prepare (or modify) the procedure(s) after the database has been uploaded, then upload the procedures individually with the **Setup** or **Debug** menu, or as a group with the **Database Sharing Options** wizard.

Uploading All Procedures when the Database is Uploaded

All of the procedures in a database will automatically be uploaded when the database itself is uploaded to the server. You can test the procedures in advance with the simulator (see “[Testing Web Procedures with the CGI Simulator Wizard](#)” on page 348) before uploading the database. When everything is ready, use the **Database Sharing Options** wizard to upload the database (see “[Uploading the Database to the Server](#)” on page 194).

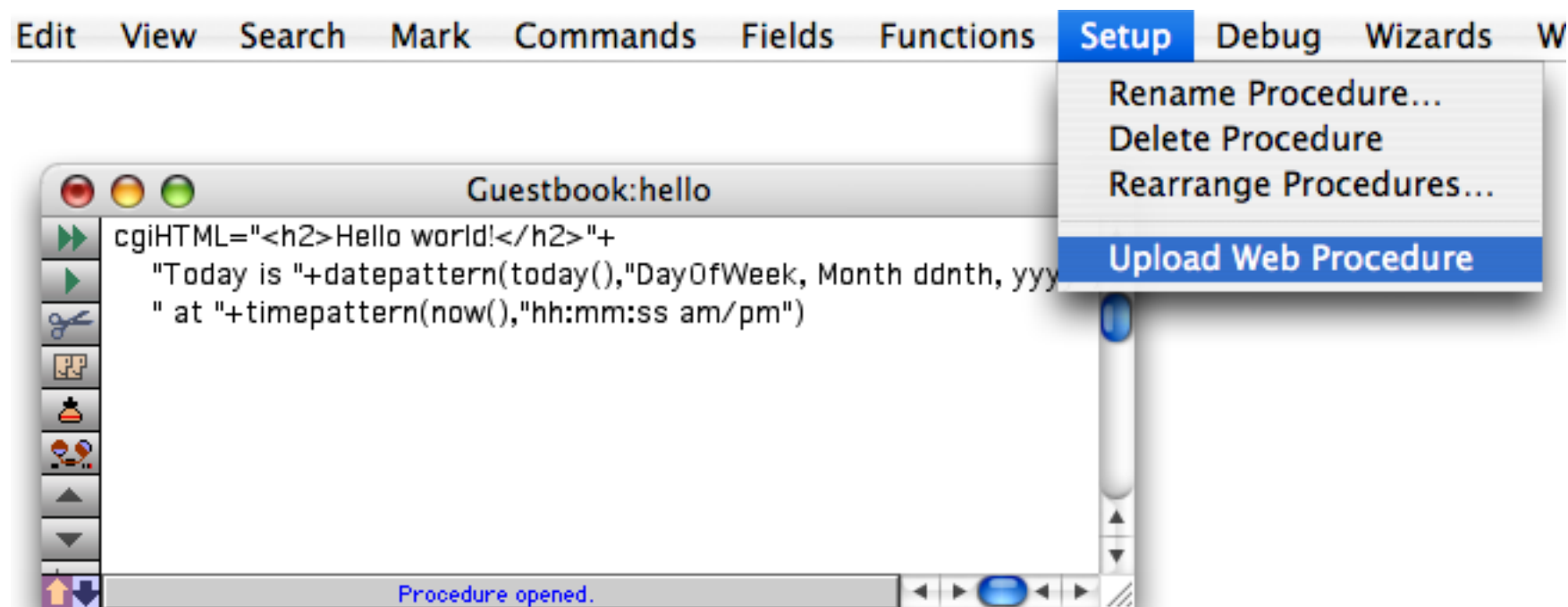
Uploading All Procedures via a New Sharing Generation

Once a database has been uploaded to the server you can re-upload it by asking the **Database Sharing Options** wizard to create a new sharing generation (see “[Sharing “Generations”](#)” on page 119). This will update all of the procedures on the server with the revised procedures on your local machine (it will also update the forms). This can be handy if you have made revisions to a lot of procedures and/or forms and want to upload them all at once.

Updating a Single Procedure/Adding a new Procedure

If you just need to upload a single web procedure you can do so right from the **Setup** menu (the **Database Sharing Options** wizard is not involved). Here’s how to do it.

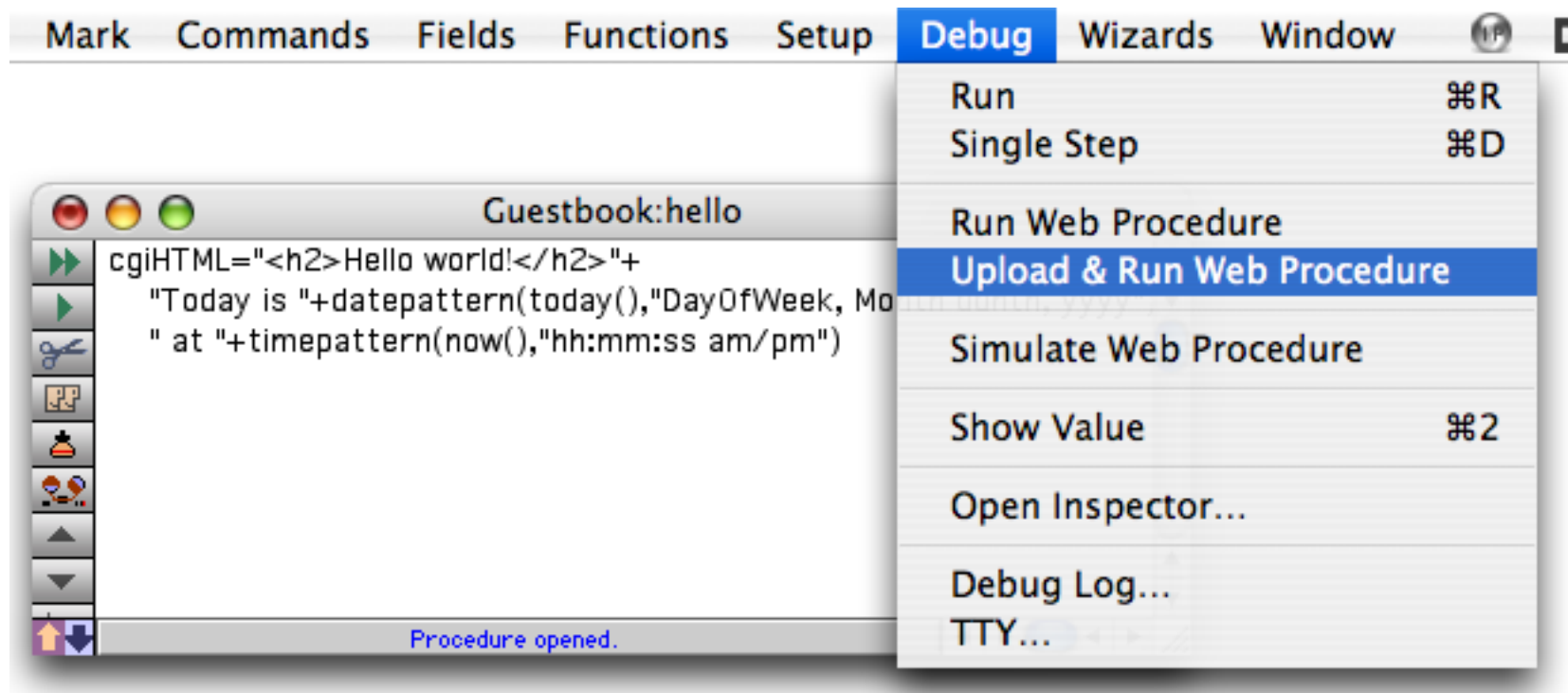
- Open the procedure on your local client (or create the new procedure).
- Make any changes that are necessary.
- Choose **Upload Web Procedure** from the **Setup** menu.



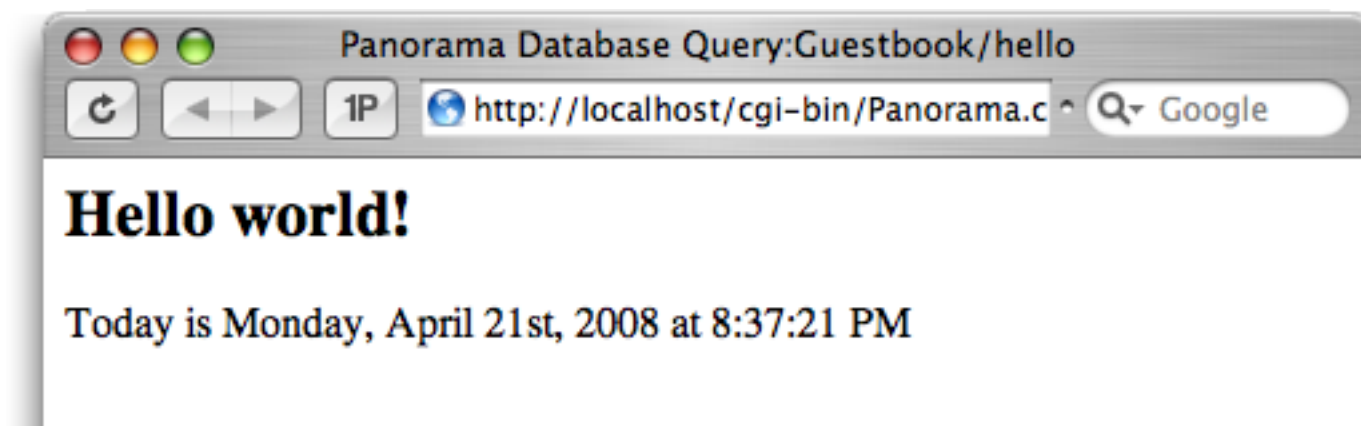
That’s all there is to it! The changes are effective immediately.

Uploading and Testing a Web Procedure

You can upload and test a web procedure all in one step. Just follow the same instructions as in the last section, but choose **Upload & Run Web Procedure** from the **Debug** menu instead of **Upload Web Procedure** from the **Setup** menu.

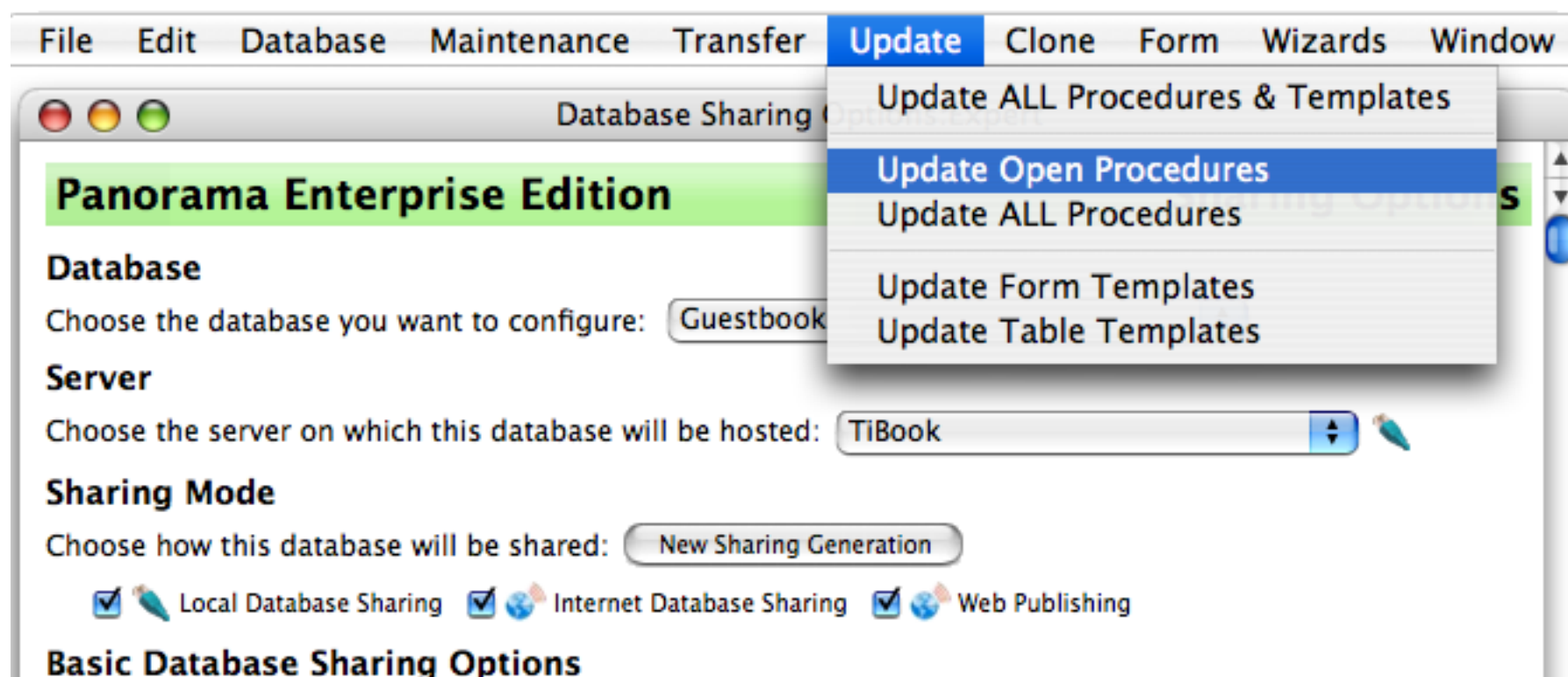


The procedure will be uploaded and then immediately run in your web browser.



Uploading Multiple Procedures with the Database Sharing Options Wizard

If you make changes to more than one procedure you can either upload each one separately (as described in the previous section) or you can use the **Update** menu in the **Database Sharing Options** wizard to upload a bunch of procedures at once.



To update all of the procedures that are currently open (visible on the screen) use the **Update Open Procedures** command. To update all of the procedures in the database, whether they are open or not, use the **Update ALL Procedures** command or the **Update ALL Procedures & Templates** command. (Note: When uploading is complete, these commands will list all of the procedures that have been updated. Only procedures that actually changed will be listed! If no procedures have changed, you'll see that zero procedures have been updated.)

Testing Web Procedures with the CGI Simulator Wizard

Since every copy of Mac OS X has a built-in web server you can try out your Panorama web interfaces on your development computer without disturbing your public web server. Debugging a Panorama procedure on a working web server isn't easy, however. Because Panorama must run in the background you can't use the normal debugging tools you've grown accustomed to — breakpoints, single stepping, even displaying an alert. This can make procedure debugging a frustrating guessing game.

To help making it easier to debug web procedures we've developed a **CGI Simulator** that simulates the operation of your web browser and server. The simulator allows you to try out your procedures in Panorama's normal environment. All of Panorama's normal debugging tools are available, so you can get your procedures up and running quickly. (Of course we do recommend that you perform a final test with the actual web server.)

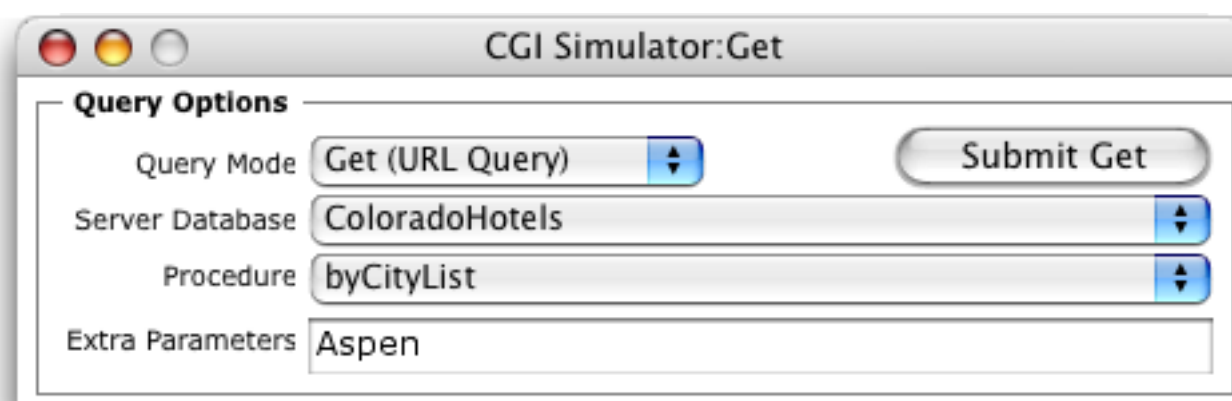
As far as your web procedures are concerned the **CGI Simulator** does a 100% perfect job of simulating the operation of the web server. From your point of view, however, the simulator is not able to accurately simulate the user interface. At each step of the simulation you'll see the web pages in your browser, but the buttons and links in these simulated pages are not functional (the browser only knows how to work with a real server). The simulator provides an alternate user interface that allows you to simulate entering data and clicking on links, buttons and pop-up menus. To be honest this alternate user interface is a bit clunky, but it's often well worth the inconvenience if you're having a hard time debugging a web procedure.

To open the **CGI Simulator** simply select it from the **Developer Tools** submenu of the **Wizard** menu.


Query Mode (get vs. post)

The **CGI Simulator** has two distinct modes of operation: **Get** and **Post**. These correspond to the two different types of HTTP web requests.

Get requests consist of just a URL. In *Get* mode the simulator simply allows you to create and submit a URL.



Put requests contain both a URL and form data. In *Put* mode the simulator allows you to fill in form data and click on any links in the form.



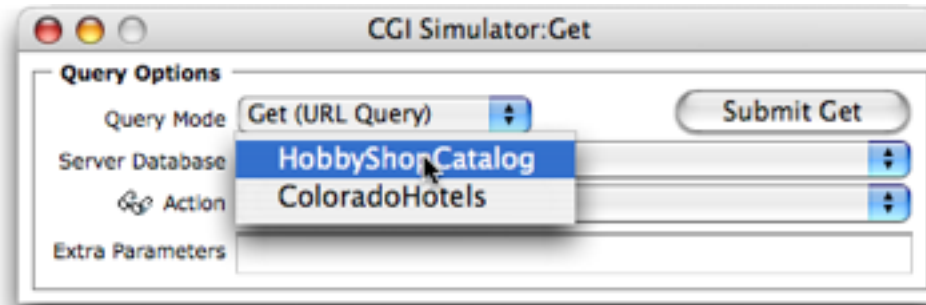
The image shows a window titled "CGI Simulator:Post" with three sections:

- Query Options:** Contains a "Query Mode" dropdown menu set to "Post (Forms)", a "Submit Post" button, an "Action" dropdown menu set to "1 -- ColoradoHotels~NewHotel", and a "File Name" text field containing "NewHotelForm.html".
- Form Fields & Data:** A list of form fields with their current values:
 - Hotel: Hotel Supremico
 - City: Aspen
 - Phone: 549-0987
 - Rate: 400.00
 - Units: 12
 - Stars: 5
 - Description: This is the best hotel in town!
- Links (click to simulate):** A list of empty links for simulation.

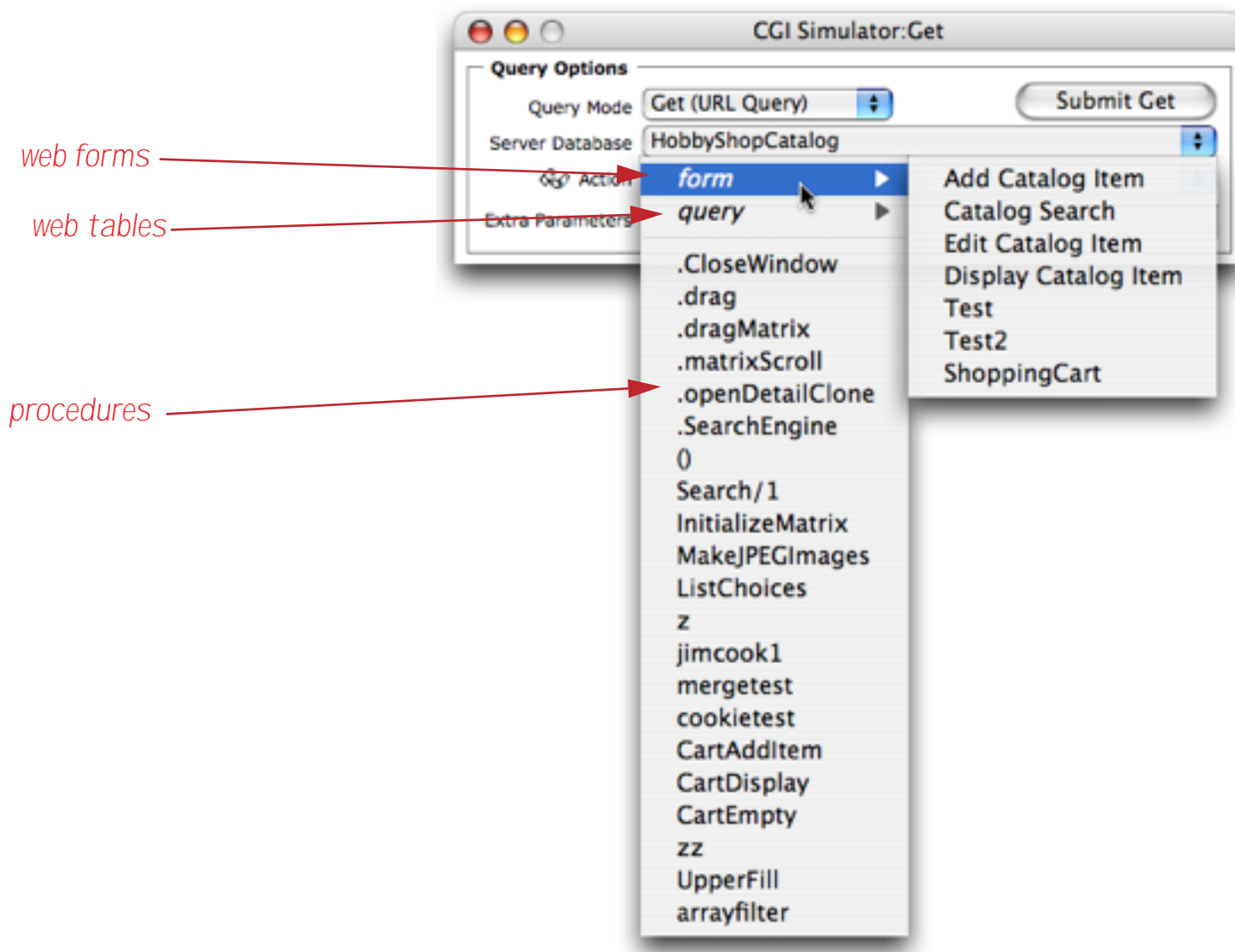
You can use the **Query Mode** pop-up menu to switch between *Get* and *Put* mode. The simulator will also sometimes switch modes automatically, for example switching from *Get* to *Put* mode if you drop a form on the simulator or if the result of a *Get* request is a form.

Testing Get Queries

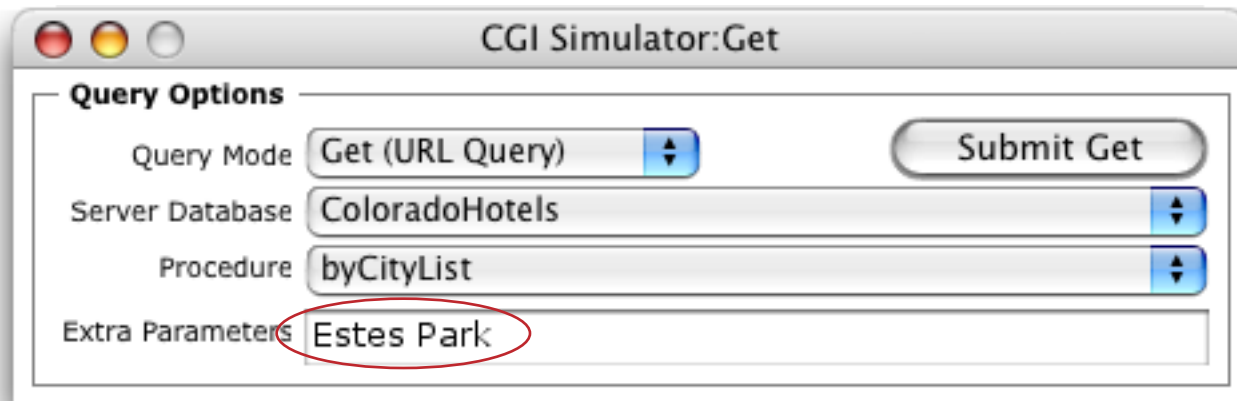
To test a get query you simply set up the URL and then submit. Start by selecting the database. The pop-up menu will show all of open databases that contain a web form or contain a procedure that uses the `cgiHTML` variable.



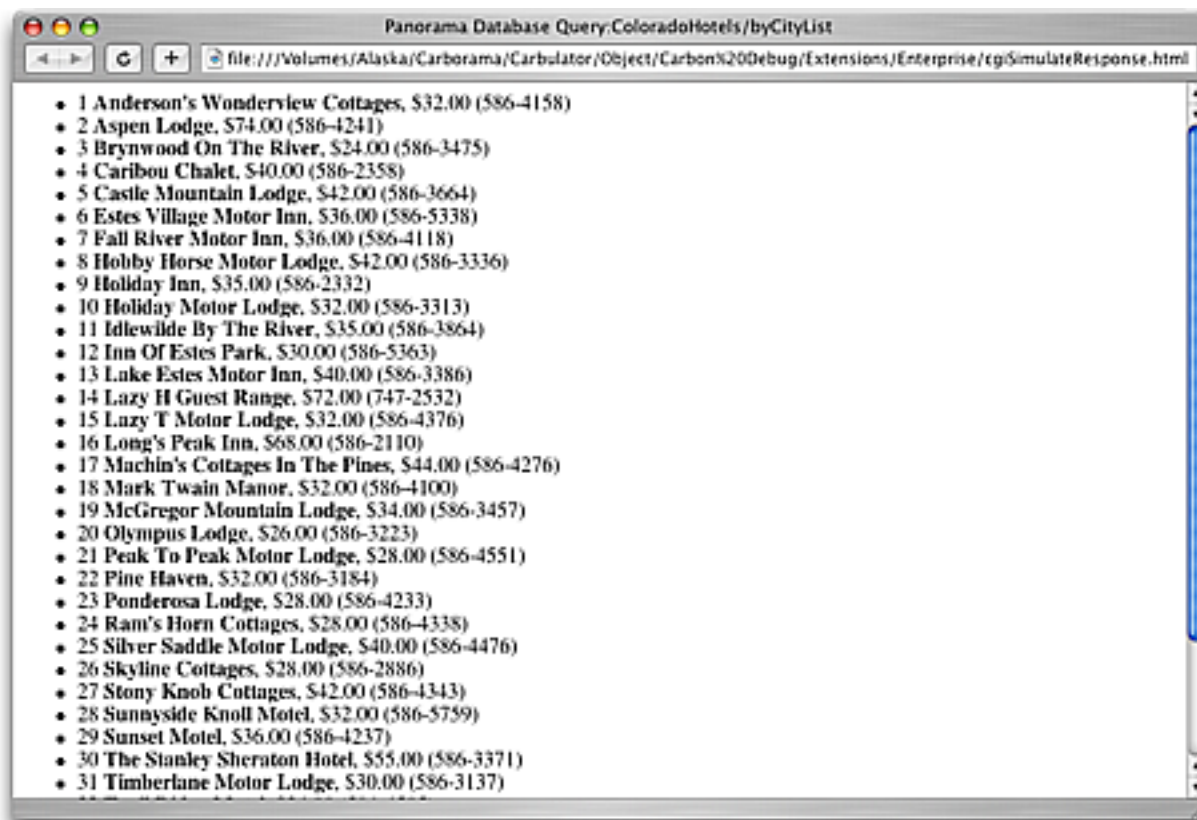
Next, select the action you want to test. You can choose a web form, a table, or any procedure in the database.



If the procedure requires any extra parameters type them into the box at the bottom of the window.



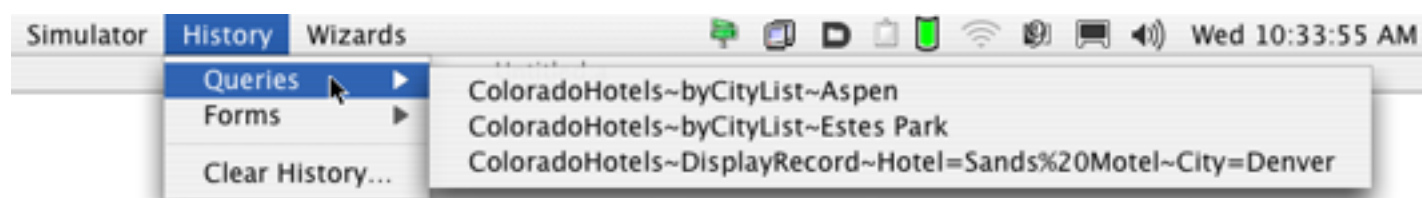
When everything is set up press the **Submit Get** button. The simulator will set up Panorama's variables and environment just like Apache would, then calls the procedure. If the procedure works correctly it will render an HTML page. The simulator takes this page and displays it in your default web browser.



If the procedure encounters an error it will stop and display an error message. If you can figure out the problem from the error message then fix it and try again. If needed you can set a breakpoint and submit the query again (if nothing has changed just press **Submit Get** again). The procedure will stop when it reaches the breakpoint and you can single step from that point to isolate the error.

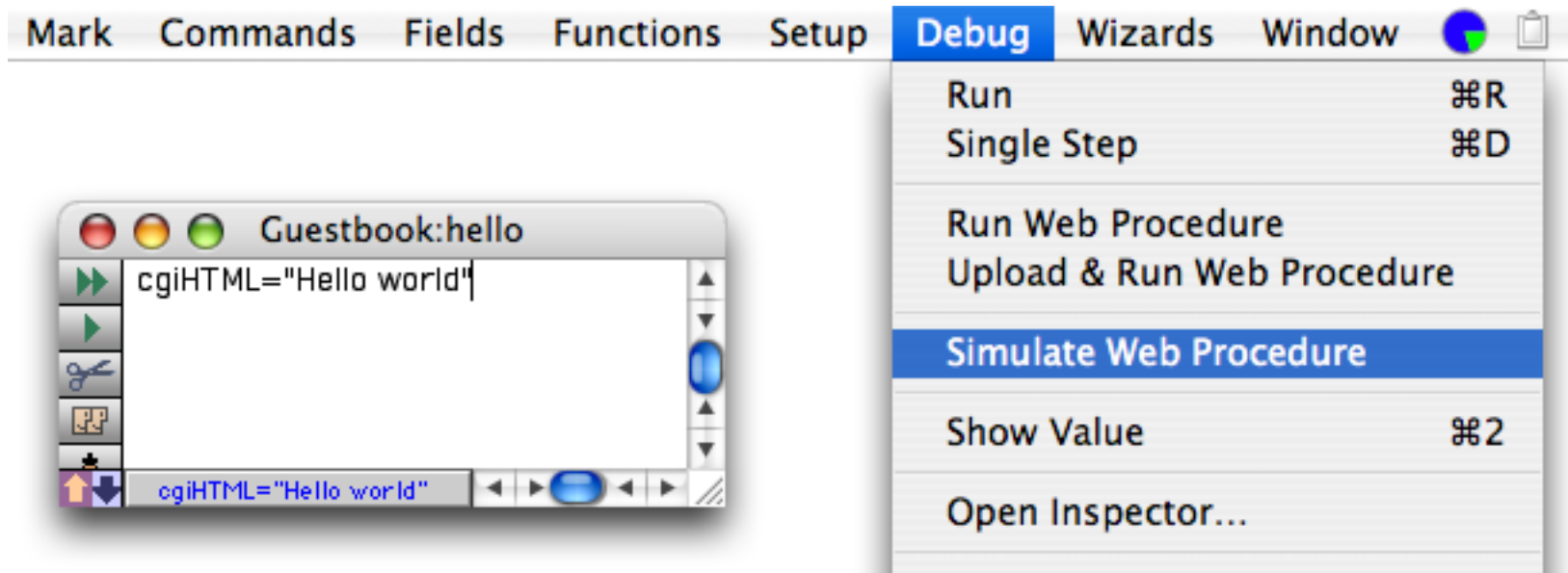
Repeating a Previous Query

The simulator keeps a record of each URL you submit. These previous URL's are visible in the **Queries** sub-menu of the **History** menu. To re-submit any previous query simply select it from the submenu, make any changes if necessary and then press the **Submit Get** button.

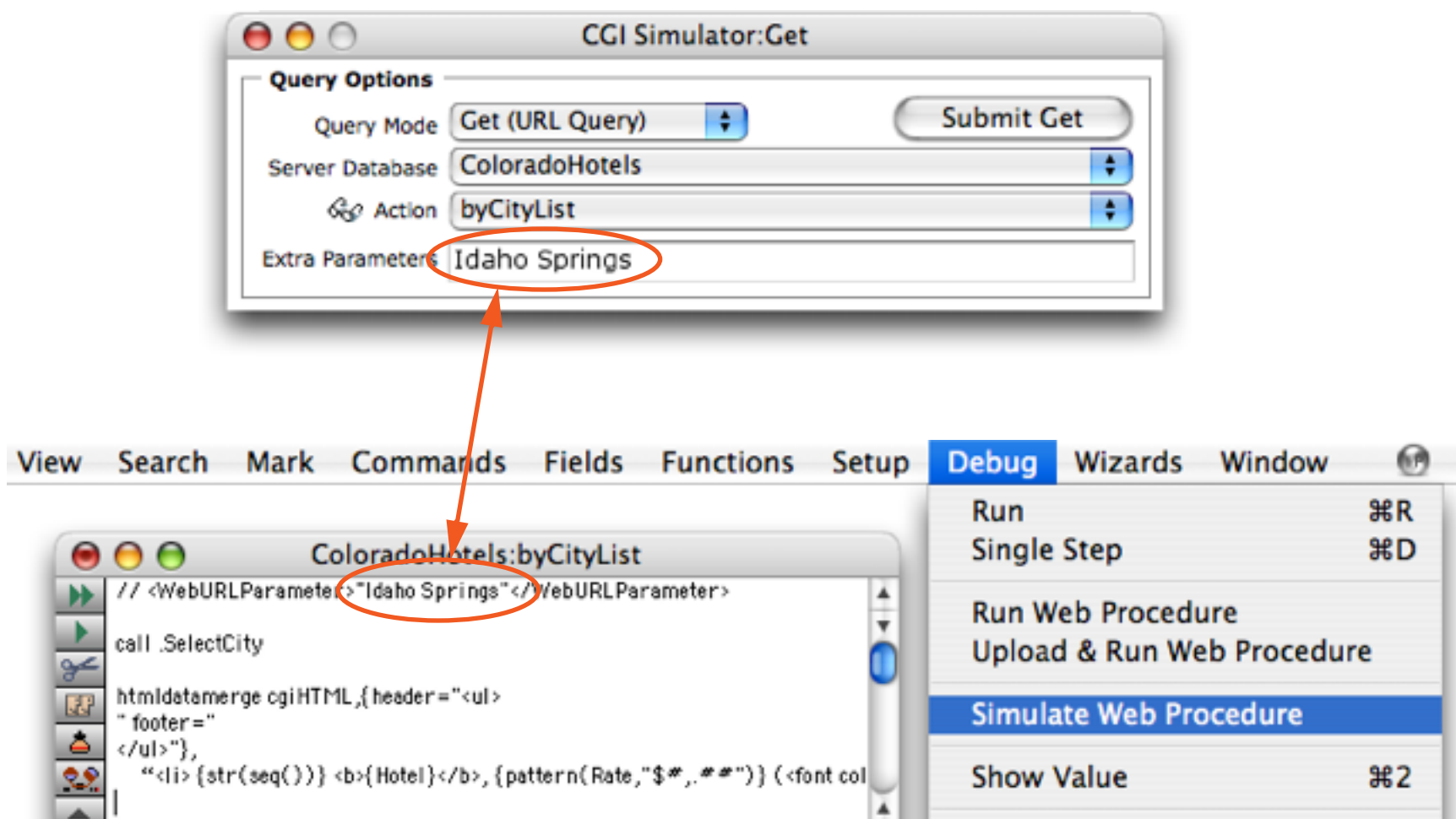


Testing Get Queries from the Debug Menu

The previous section described how to simulate a web procedure by opening the **CGI Simulator** wizard and selecting the database and procedure from pop-up menus. If the procedure is already open there is a much simpler method — just choose the **Simulate Web Procedure** command from the **Debug** menu. Choosing this item will automatically open the **CGI Simulator** wizard, fill in the pop-up menus and press the **Submit Get** button.

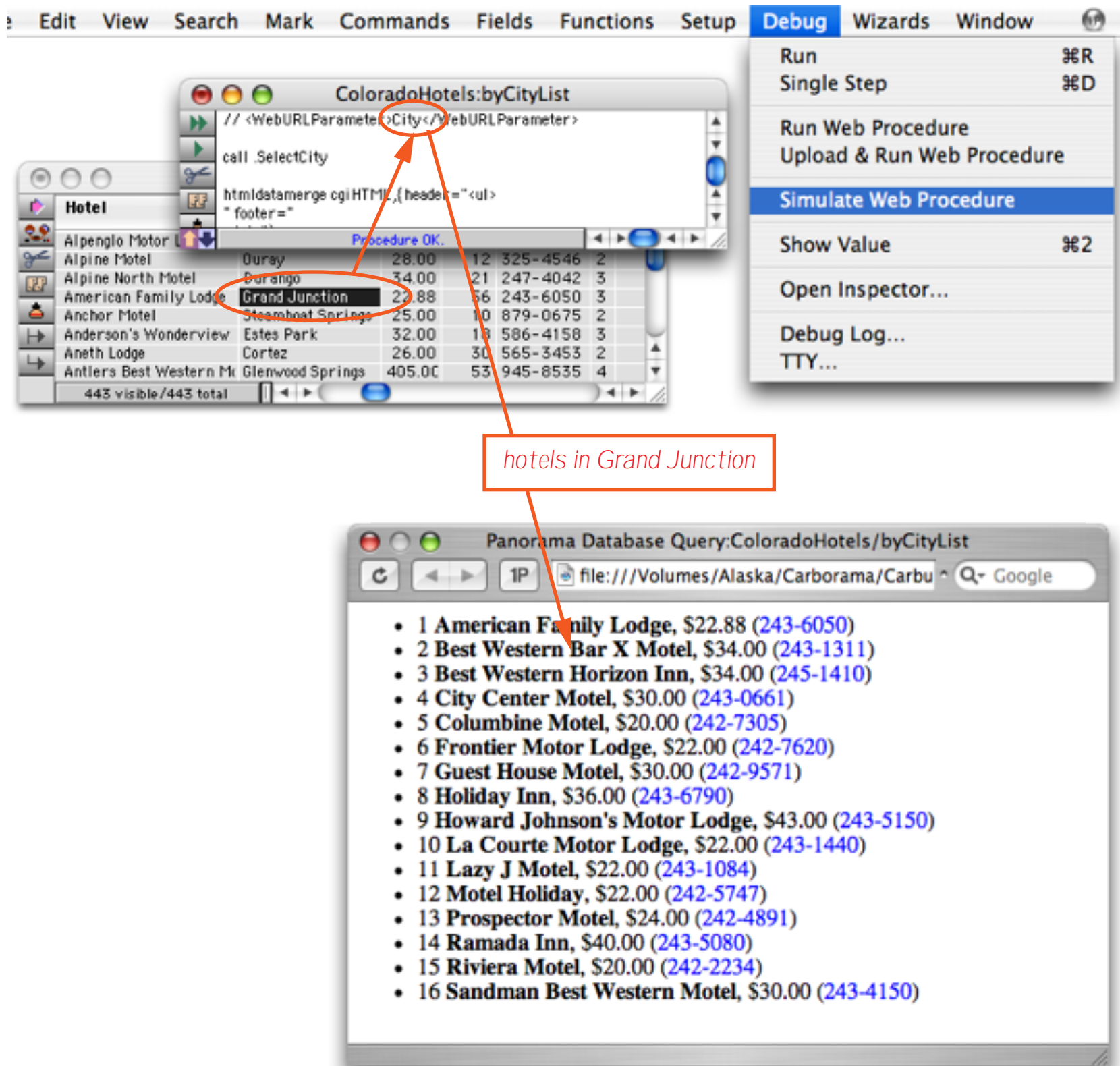


The **CGI Simulator** wizard allows you to enter any extra parameters needed at the end of the URL. When you use the **Simulate Web Procedure** command you bypass this window, so you don't get a chance to enter any extra parameters needed. Instead, you can embed the extra parameter portion of the URL directly into a comment on the first line of the procedure itself. This comment must contain a `<WebURLParameter></WebURLParameter>` tag, with a formula in between the tags. The illustration below shows how this is done using the **CGI Simulator** wizard vs. using an embedded tag.



When you use the **Simulate Web Procedure** command Panorama will check for the presence of this special comment and tag. If it finds such a comment it evaluates the formula in the tag and uses that as the extra URL parameter for the purposes of the test. (You don't have to worry about spaces in the parameter, Panorama automatically converts them to `%20` for you.)

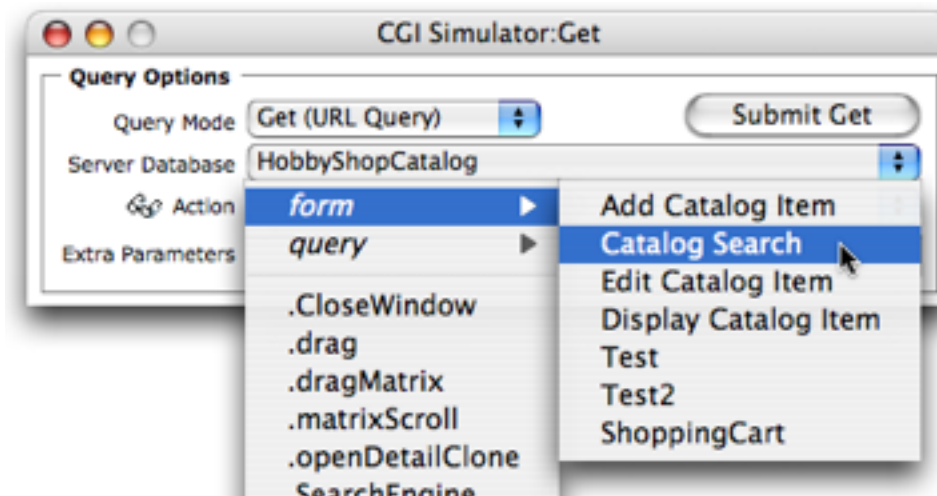
You can use any valid formula inside the `<WebURLParameter></WebURLParameter>` tag, including database fields and variables (if you use a variable, make sure it is a global or fileglobal variable that is already defined before the procedure runs). For example, this illustration shows how a database field can be used to generate the extra parameter.



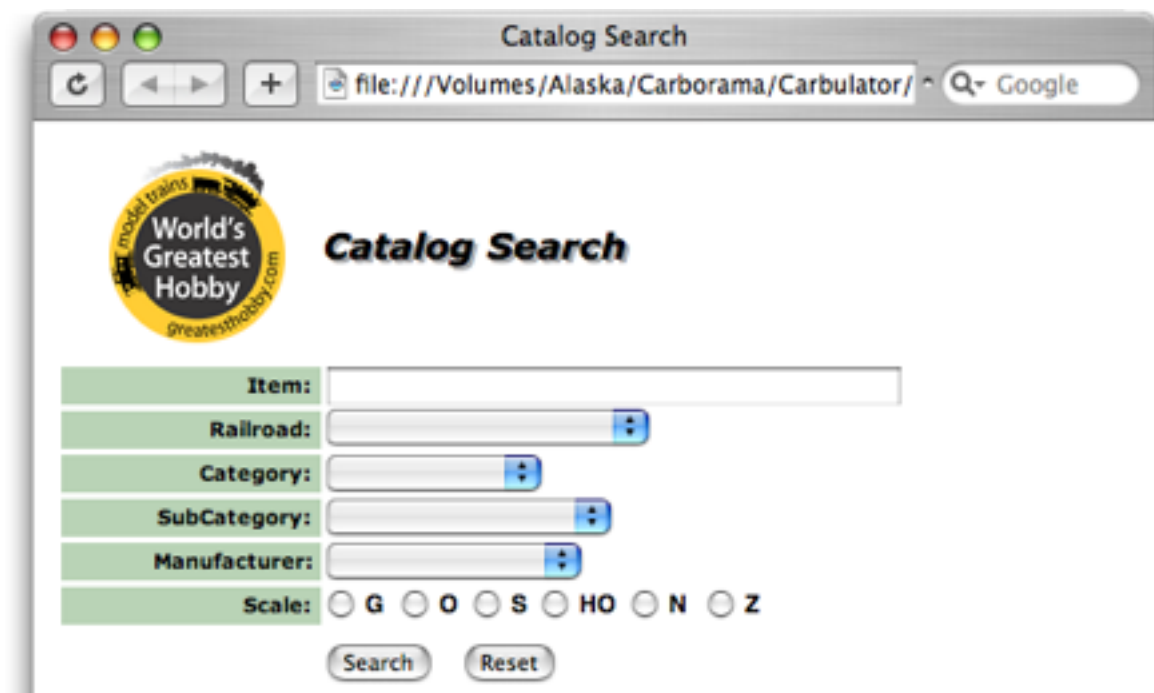
In this example the formula is simply the `City` field. Since the current city is `Grand Junction`, simulating the web procedure will list all hotels in Grand Junction.

Testing Post Queries (Forms)

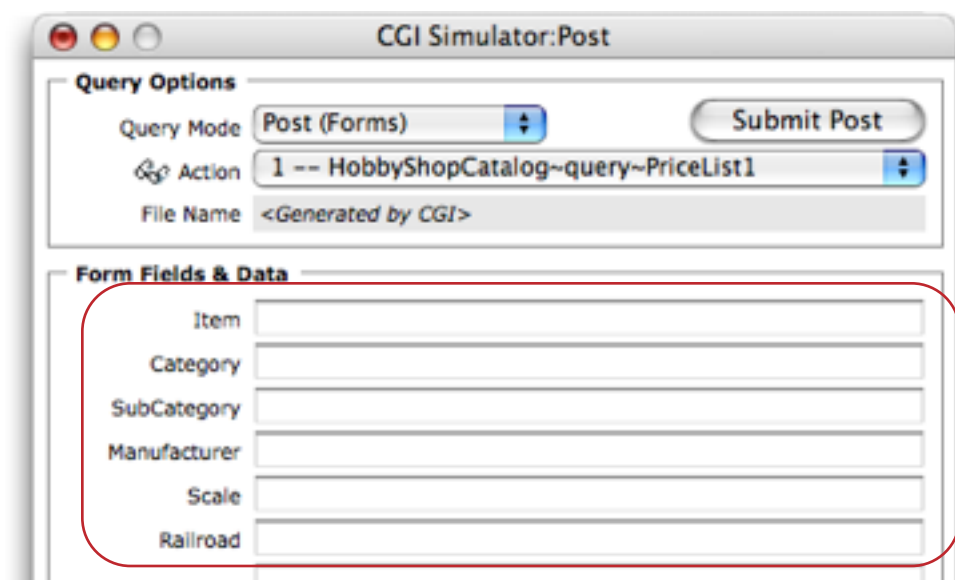
Many web procedures are designed to process data submitted from forms. The simulator can be used to test and debug these procedures as well. The first step is to set up the data to be submitted. You can't use the web browser for this, since the web browser only knows how to submit data to a real server. So the simulator includes a simple data entry ability. If you are using a web form based on a Panorama form you can simply select the form from the submenu in the Action pop-up menu.



When you press the **Submit Get** button the simulator displays the form in your default web browser.



As mentioned earlier, you can't simply fill in this form in the web browser. However, when you bring Panorama back to the front you'll see that the simulator has expanded and shows all of the fields in the form:



Fill in the fields as required for your testing. In this case we'll search for items from the [Santa Fe](#) railroad. (As you can see, the simulator requires that all fields be filled in by typing. There are no pop-up menus, checkboxes or radio buttons. It's up to you to know what each field is and what it should contain. If the form contains any "hidden" fields these are also shown, and can be modified.)

The screenshot shows a window titled "CGI Simulator:Post". Under the "Query Options" section, the "Query Mode" is set to "Post (Forms)" and the "Action" is "1 -- HobbyShopCatalog--query--PriceList1". A "Submit Post" button is visible. The "Form Fields & Data" section contains several input fields: "Item", "Category", "SubCategory", "Manufacturer", "Scale", and "Railroad". The "Railroad" field is highlighted with a red circle and contains the text "Santa Fe".

Once you've entered the necessary data press the **Submit Post** button. The simulator will trigger the actual server code used to process data submitted from this form. The database will be searched, selected and/or modified as specified by the code you have written. When the code is complete the simulator will display the result in the browser. In this example, a list of items matching [Santa Fe](#) are listed.

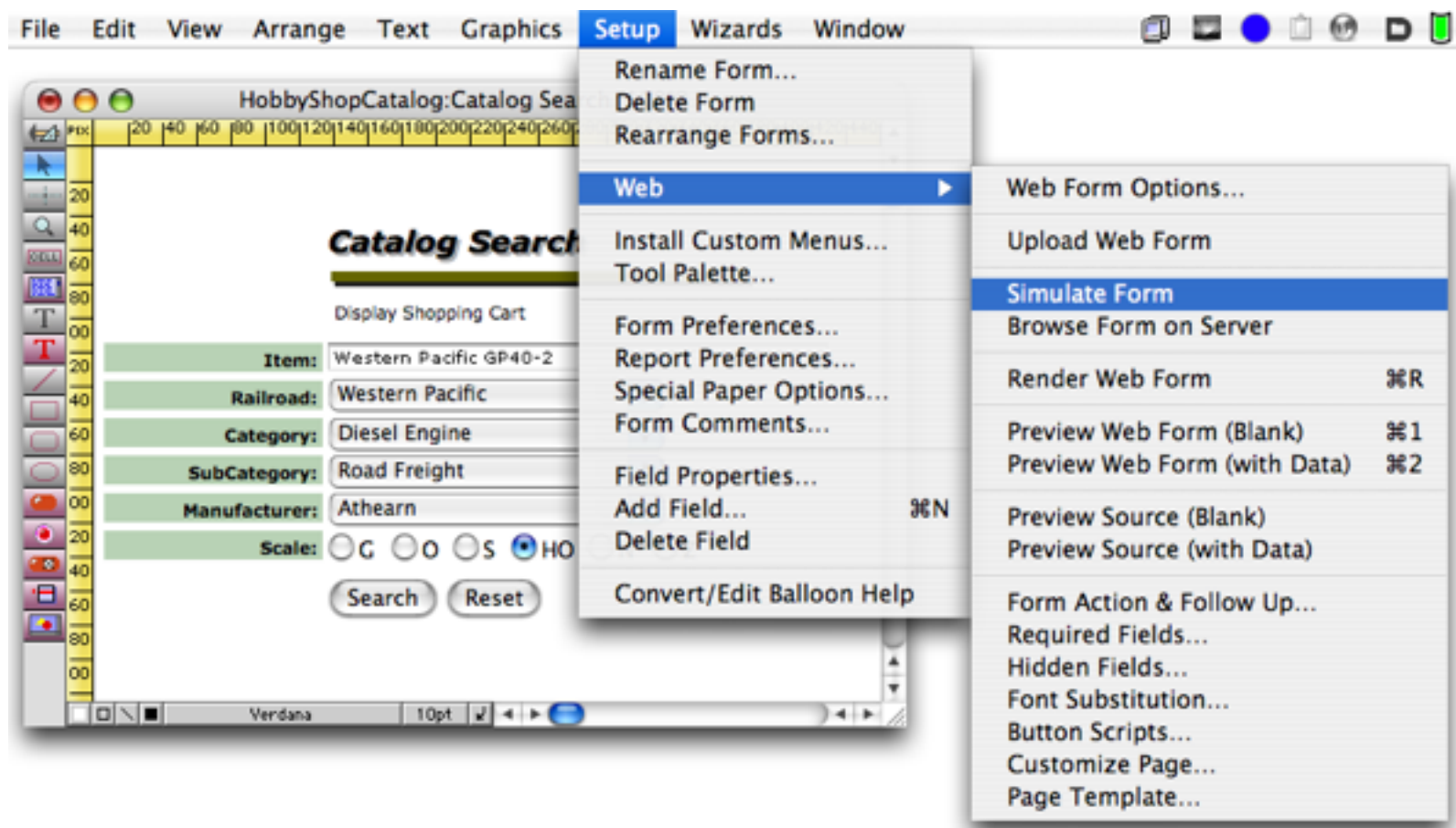
The screenshot shows a web browser window titled "Hobby Shop Catalog". The address bar shows a file path. The main content is a table with the following data:

| Scale | Manufacturer | Item | Price |
|-------|--------------|---|-------|
| HO | Athearn | Santa Fe F7A | 34.99 |
| HO | Athearn | Santa Fe F7B | 15.99 |
| HO | Athearn | Santa Fe SD40-2 | 22.49 |
| HO | Athearn | Santa Fe F45 | 29.99 |
| HO | Athearn | Santa Fe 40' Reefer | 14.99 |
| HO | Athearn | Santa Fe 40' Pulp Flat | 4.99 |
| HO | Athearn | Santa Fe WideVision Caboose | 5.49 |
| HO | Athearn | Santa Fe GP60 | 34.99 |
| HO | Athearn | Santa Fe Caboose | 5.49 |
| HO | Athearn | Santa Fe GP50 | 34.99 |
| HO | Athearn | ATSF 50' Box (Grand Canyon) | 4.49 |
| HO | Athearn | ATSF 50' Box (Chief) | 4.49 |
| HO | Athearn | ATSF Tank Car | 4.49 |
| HO | Life-Like | ATSF 4427 Covered Hopper | 20.79 |

If the procedure encounters an error it will stop and display an error message. If you can figure out the problem from the error message then fix it and try again. If needed you can set a breakpoint and submit the query again (if nothing has changed just press **Submit Post** again). The procedure will stop when it reaches the breakpoint and you can single step from that point to isolate the error.

Testing Post Queries Directly from a Panorama Form

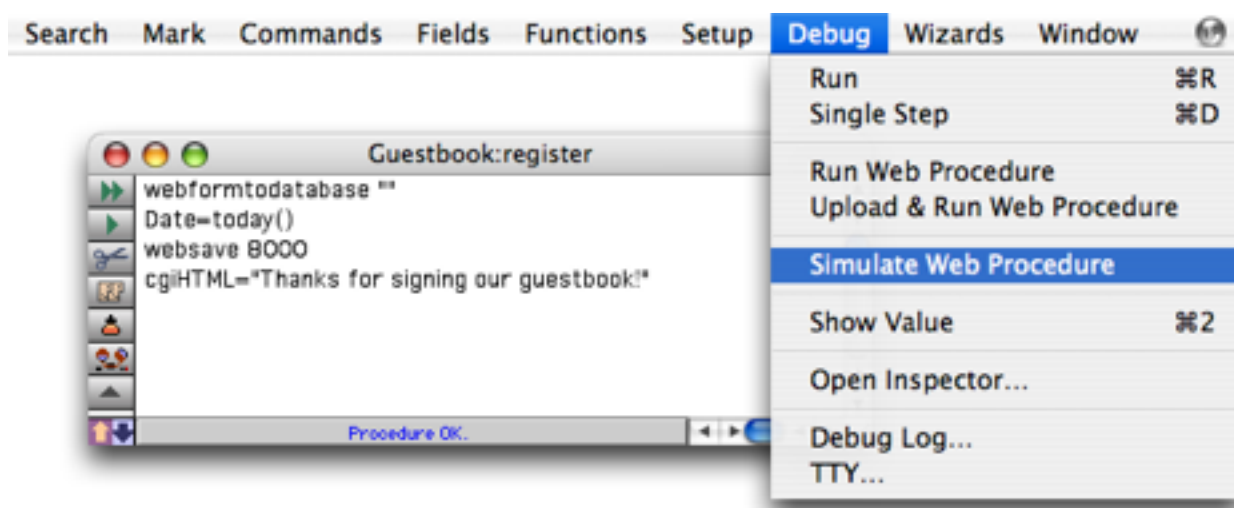
The previous section described how to simulate a web procedure by opening the CGI Simulator wizard and selecting the database and form from pop-up menus. If the form is already open there is a much simpler method — just choose the **Simulate Form** command from the **Web** submenu of the **Setup** menu.



From this point on the simulator works just as described in the previous section.

Testing Post Queries Directly from a Panorama Procedures

If the form you want to test is already open then the **Simulate Form** command is very handy. But often you'll have the procedure assigned to the form open, but not the form itself. No worries, simply choose the **Simulate Web Procedure** command from the **Debug** menu.

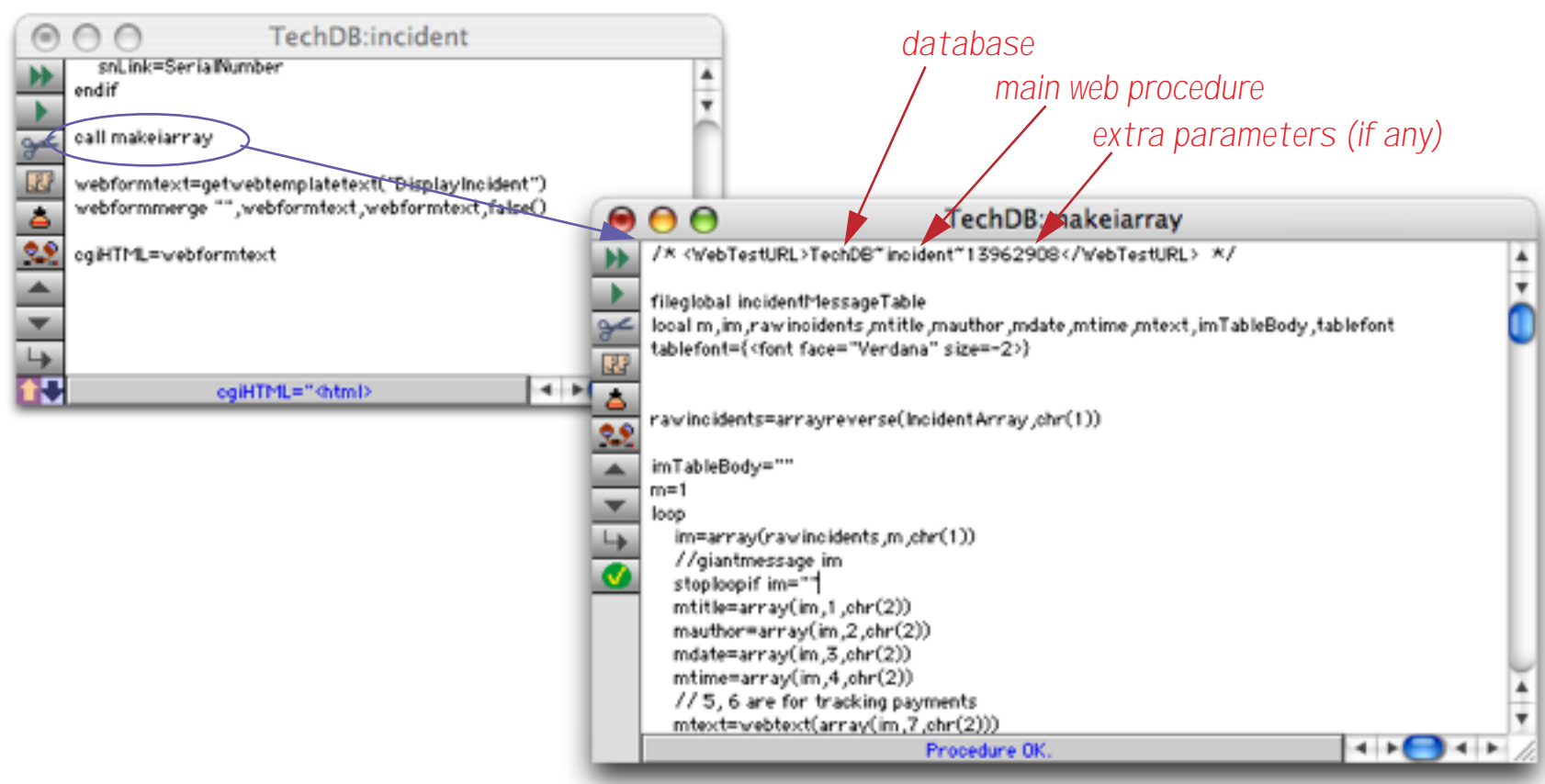


Panorama will check to see if the procedure is attached to a web form in the current database, and if so, it will simulate the form rather than trying to immediately simulate the procedure (which won't work because the procedure needs form input. From this point on the simulator works just as described in "[Testing Post Queries Directly from a Panorama Form](#)" on page 356.

Testing Post Queries from a Subroutine

For complex applications you may break up web procedures into multiple subroutines. If you're working on the subroutine using **Simulate Web Procedure** won't work, because you really want to simulate the main web procedure, which then calls the subroutine. One solution is to leave the main procedure open and always use **Simulate Web Procedure** with that window on top. However, by embedding a special comment in the subroutine you can tell Panorama that it should simulate the main web procedure instead of the subroutine when **Simulate Web Procedure** is used. Simply embed a comment at the top of the subroutine that contains the tag `<WebTestURL></WebTestURL>`. The contents of this tag must contain the name of the main web procedure, the procedure that calls this subroutine.

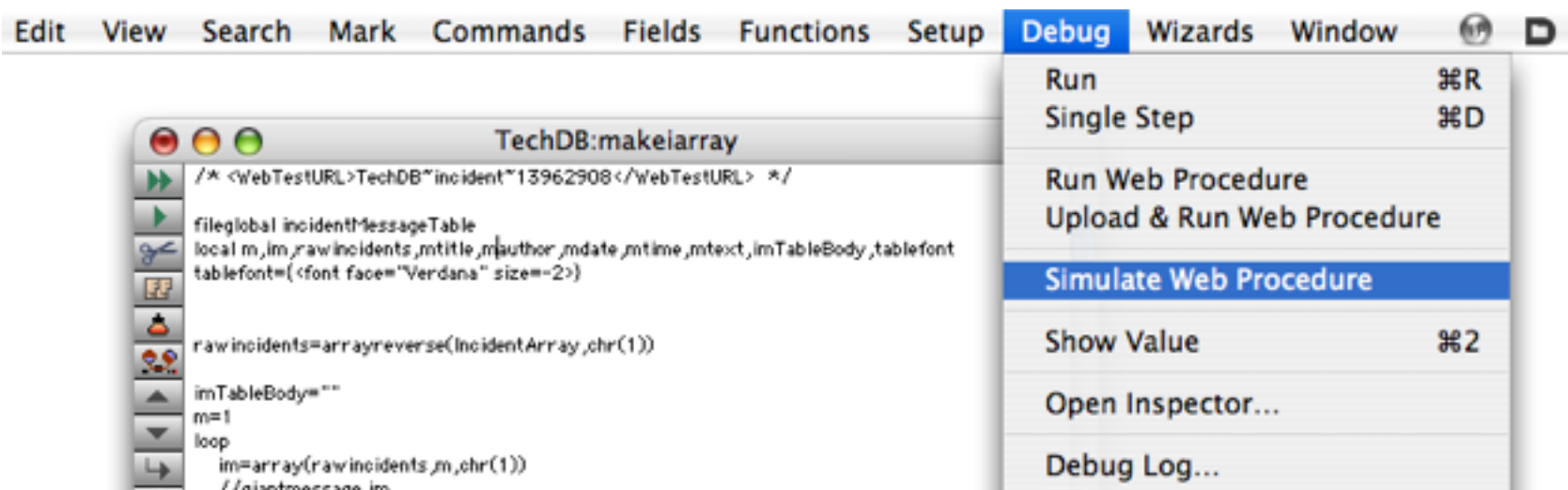
The illustration below shows how this is done. The main web procedure is called `incident`. This is a long procedure, only the last few lines are shown. Towards the end it calls a subroutine named `makeiarray`, another long procedure. If you are working on debugging `makeiarray` you really need to simulate `incident`.



The first line contains the comment needed to make this happen:

```
/* <WebTestURL>TechDB~incident~13962908</WebTestURL> */
```

The `<WebTestURL>` tag contains the Panorama portion of the URL you want to simulate: the database, procedure name, and any extra parameters. With this tag you can simulate even if the incident procedure isn't currently open.



Because of the `<WebTestURL>` tag the Simulate Web Procedure command will actually simulate the **incident** procedure, which then calls `makeiarray`. Here's the final result.



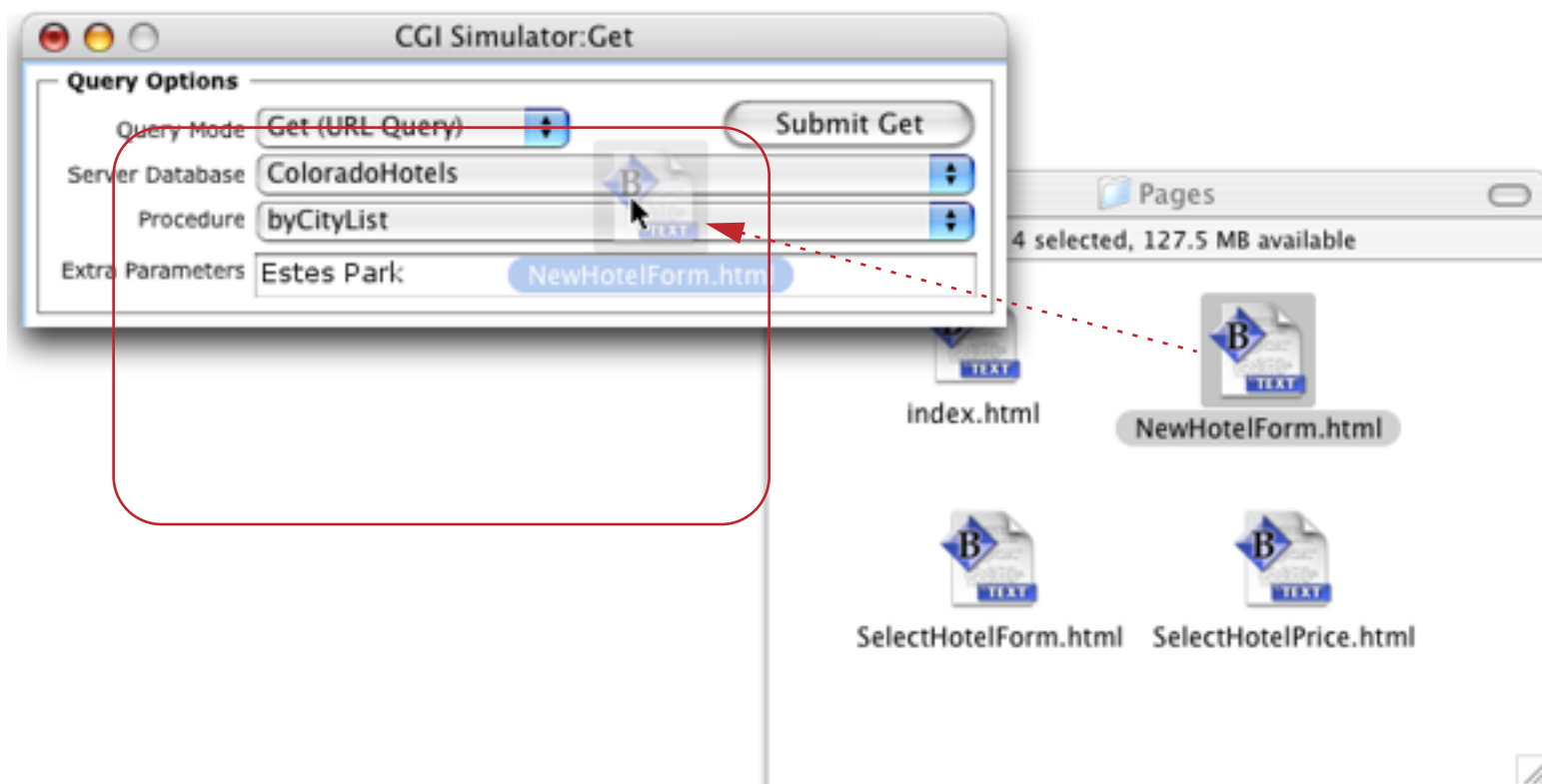
← this portion of the page was generated by the `makeiarray` subroutine

The `<WebTestURL>` tag also works with the Run Web Procedure and Upload & Run Web Procedure commands.

Testing Forms in Separate HTML Files

In addition to creating web forms within Panorama you can also use forms created in separate files (forms with BBEdit, Dreamweaver, etc.). To work with Panorama Enterprise these forms must contain an action tag that invokes a Panorama procedure (see "[Linking to a Web Procedure from a Form](#)" on page 331).

To test an external HTML file, simply drag the file from the Finder onto the simulator.



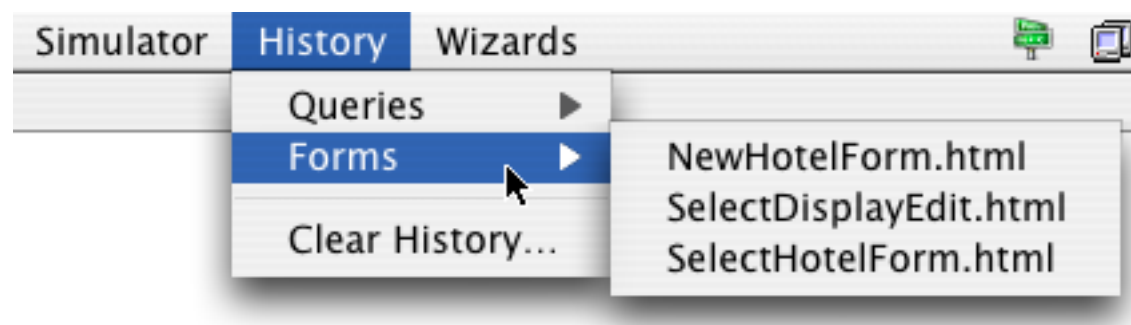
If the simulator is in *Get* mode (as shown above) it will automatically switch to *Post* mode. It will also analyze the form and display the action URL for the form and the form items.

The screenshot shows the 'CGI Simulator:Post' window. It has three main sections:

- Query Options:** Contains a 'Query Mode' dropdown set to 'Post (Forms)', a 'Submit Post' button, an 'Action' dropdown set to '1 -- ColoradoHotels~NewHotel', and a 'File Name' text field containing 'NewHotelForm.html'.
- Form Fields & Data:** A list of form fields: 'Hotel', 'City', 'Phone', 'Rate', 'Units', 'Stars', and 'Description'. Each field has a corresponding empty text input box.
- Links (click to simulate):** A list of empty text input boxes for simulating links.

From this point on the simulator works exactly the same as when using a Panorama based form. Simply fill in the fields and press the **Submit Post** button.

If dragging the HTML file onto the form is inconvenient you can also select the file using the **Load Form** command in the **Simulator** menu. You can also select any external form you've used previously from the **Forms** submenu of the **History** menu.

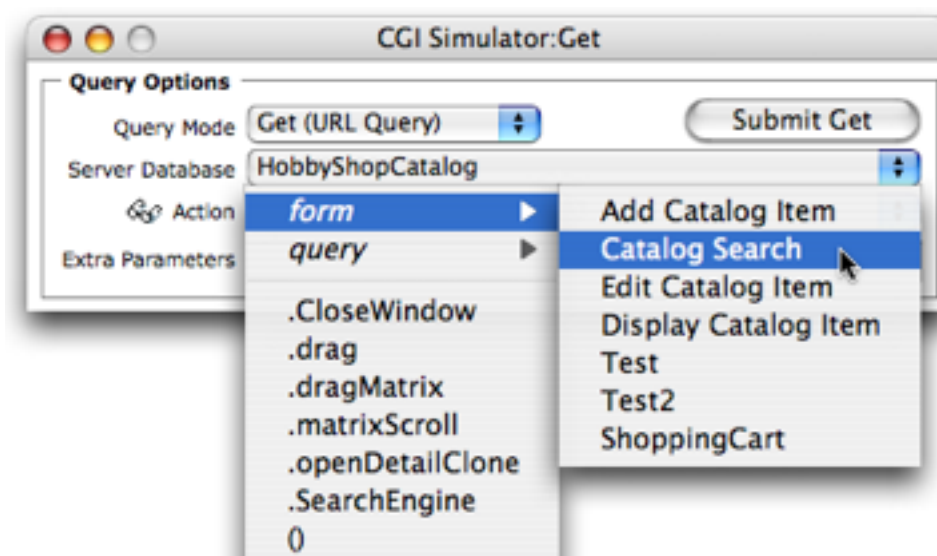


Simulating Multiple Request Sequences

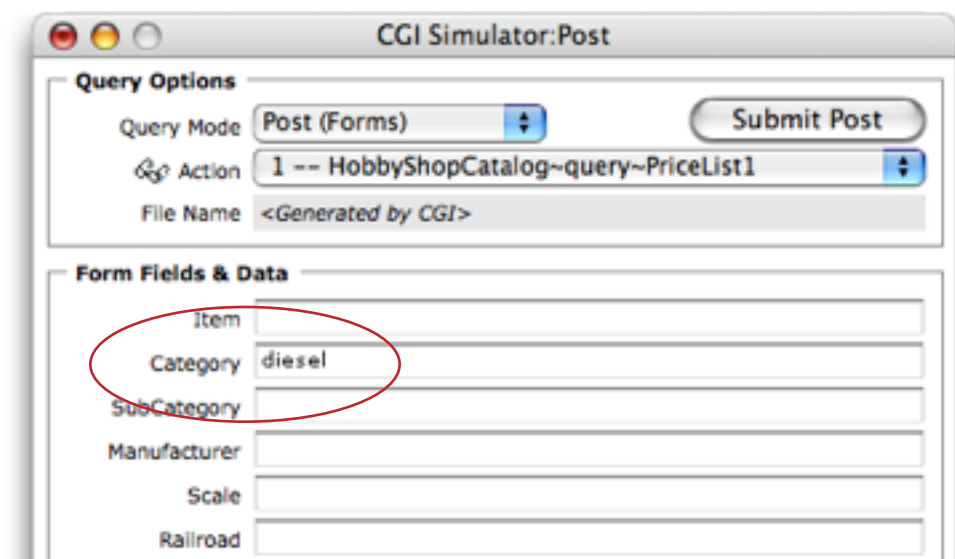
So far we've assumed that all interaction with a database is handled with a single request. The user makes a request, the server responds, we're done. However, many real world applications are more complicated. After the user makes his or her initial request, the server may respond with a form that allows a further request to be submitted, or with a list of links to further requests. These requests can ping pong back and forth between user and server until the overall process is complete.

Each time the simulator processes a request it checks the resulting HTML to see if it contains a form or any links that would allow the user to continue interacting with the server. If so, it updates the Post window to allow you to continue the session by filling in the new form or selecting a link.

The best way to understand a multiple request session is to see one in action. We'll start the session with the [Catalog Search](#) form. Open the form with the **form** submenu of the **Action** pop-up menu (or, for external forms, by dragging the form onto the simulator).



Start by filling in the criteria you want to search for. We'll search for [diesel](#) engines.



When the **Submit Post** button is pressed the simulator does its thing, producing a list of diesel engines in the catalog.

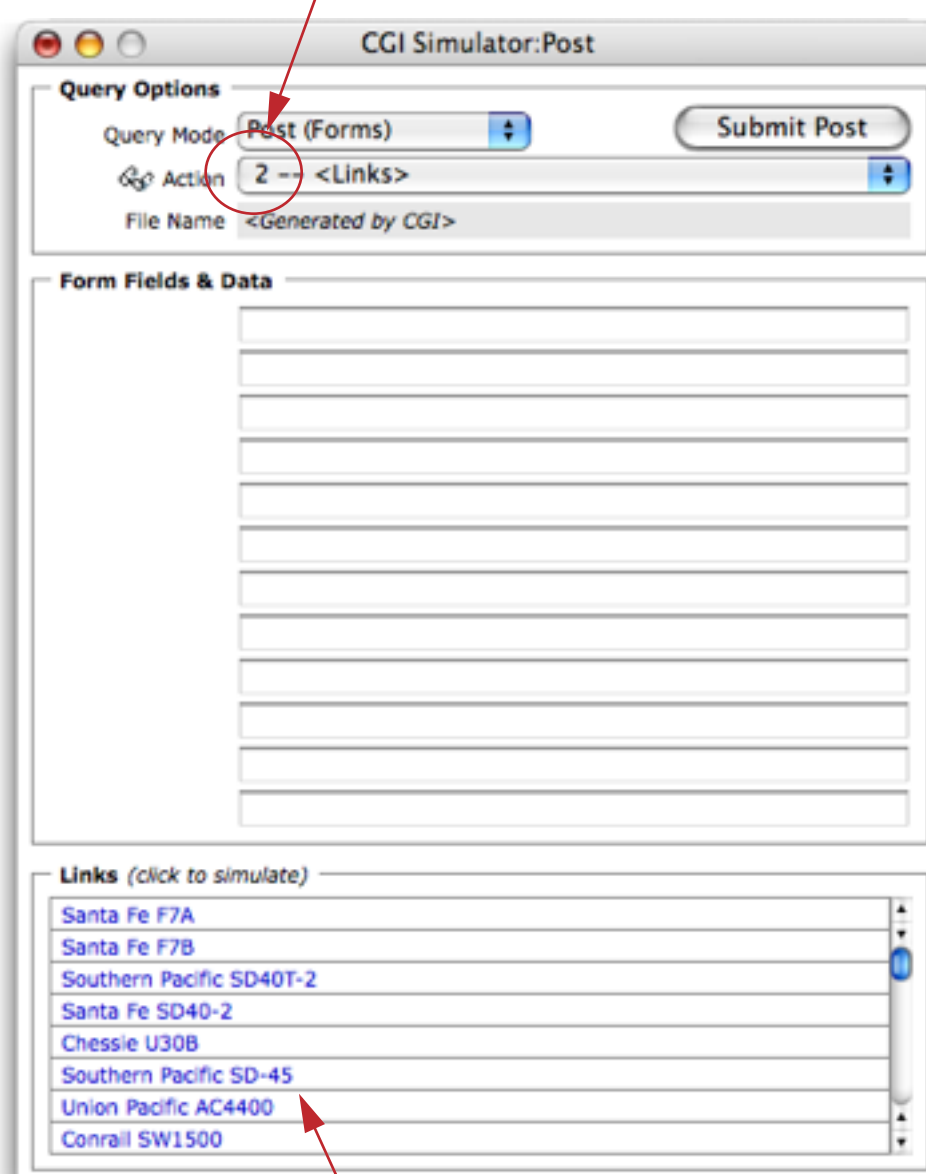


| Scale | Manufacturer | Item | Price |
|-------|--------------|--|-------|
| HO | Athearn | Santa Fe F7A | 34.99 |
| HO | Athearn | Santa Fe F7B | 15.99 |
| HO | Athearn | Southern Pacific SD40T-2 | 37.99 |
| HO | Athearn | Santa Fe SD40-2 | 22.49 |
| HO | Athearn | Chessie U30B | 28.99 |
| HO | Athearn | Southern Pacific SD-45 | 29.99 |
| HO | Athearn | Union Pacific AC4400 | 50.49 |

Notice that each hotel name is underlined to indicate that this is a link. If this page had been generated by a real web server instead of the simulator you could click on the link to see and edit the detailed information for that hotel. However this page is really a static web page (text file) generated by the simulator so the links don't work.

It is possible to continue the session, however. To do that, click back to the **CGI Simulator**, which now looks like this:

Second page of this session



Query Options

Query Mode: **Post (Forms)**

Action: **<Links>**

File Name: <Generated by CGI>

Form Fields & Data

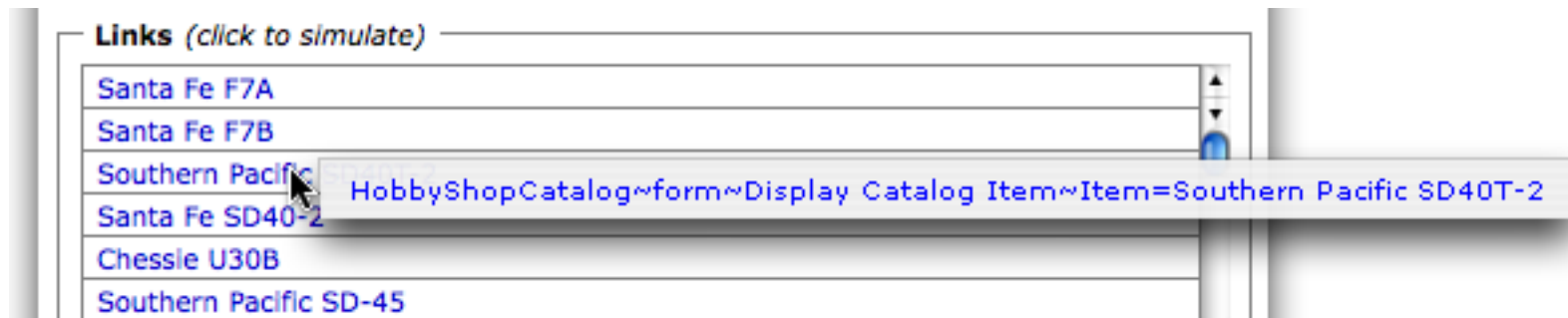
Links (click to simulate)

- [Santa Fe F7A](#)
- [Santa Fe F7B](#)
- [Southern Pacific SD40T-2](#)
- [Santa Fe SD40-2](#)
- [Chessie U30B](#)
- [Southern Pacific SD-45](#)
- [Union Pacific AC4400](#)
- [Conrail SW1500](#)

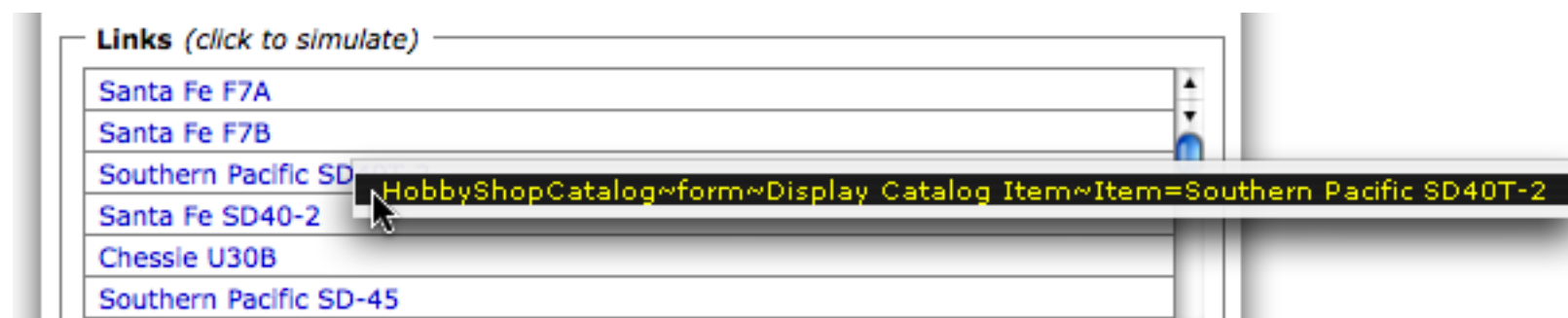
Links to additional pages

Near the top of this window the **Action** pop-up menu starts with 2. This indicates that this is the second page of this session. This page doesn't have a form, so the **Form Fields & Data** section is blank. The page does contain links, however, which are listed in the bottom section of the window. All of the links that will trigger another CGI request are listed. (Links to other web pages or web sites are not included).

When you click on a link a pop-up menu shows the exact URL for the link.



To simulate this URL select this URL in the pop-up menu.

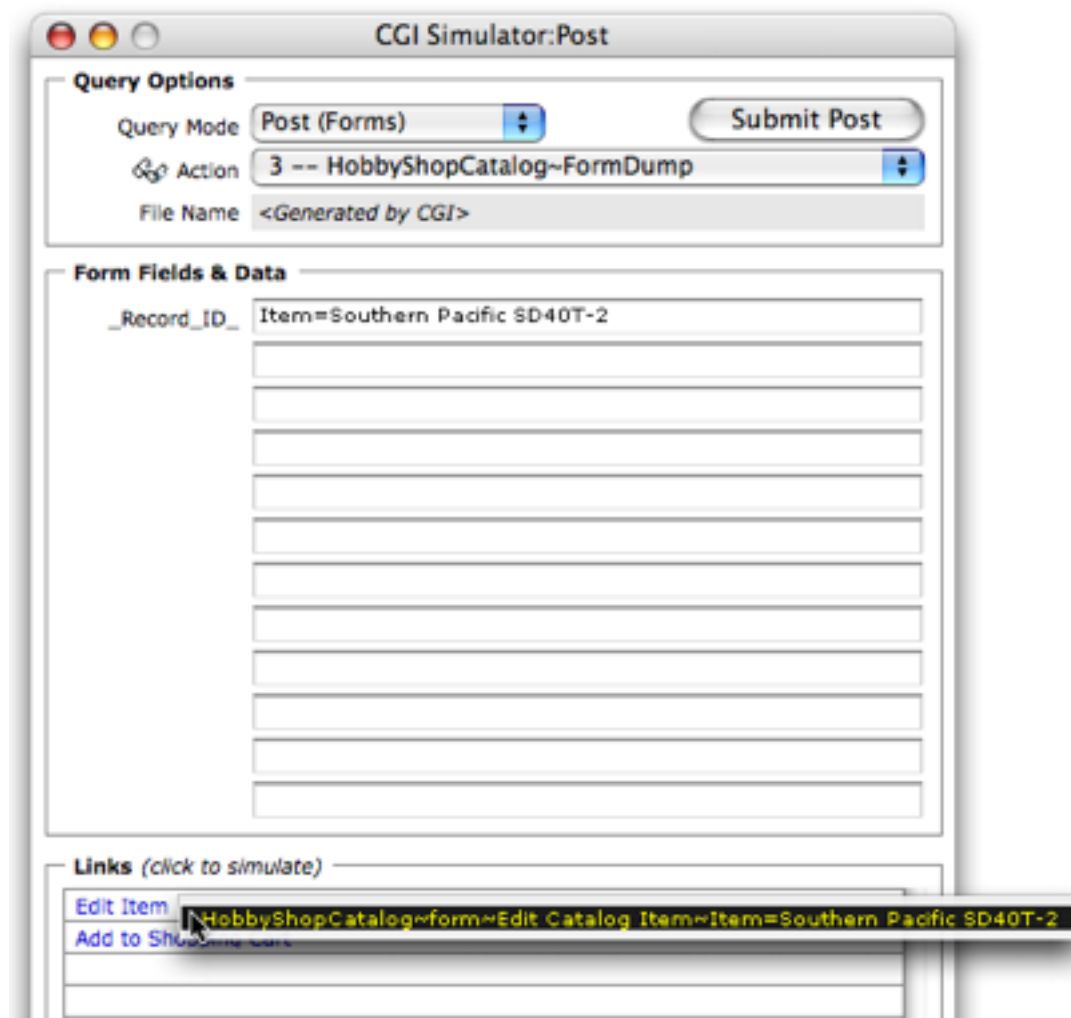


When you release the mouse the simulator will run. In this case the result is a page that shows the detail information for this diesel engine.

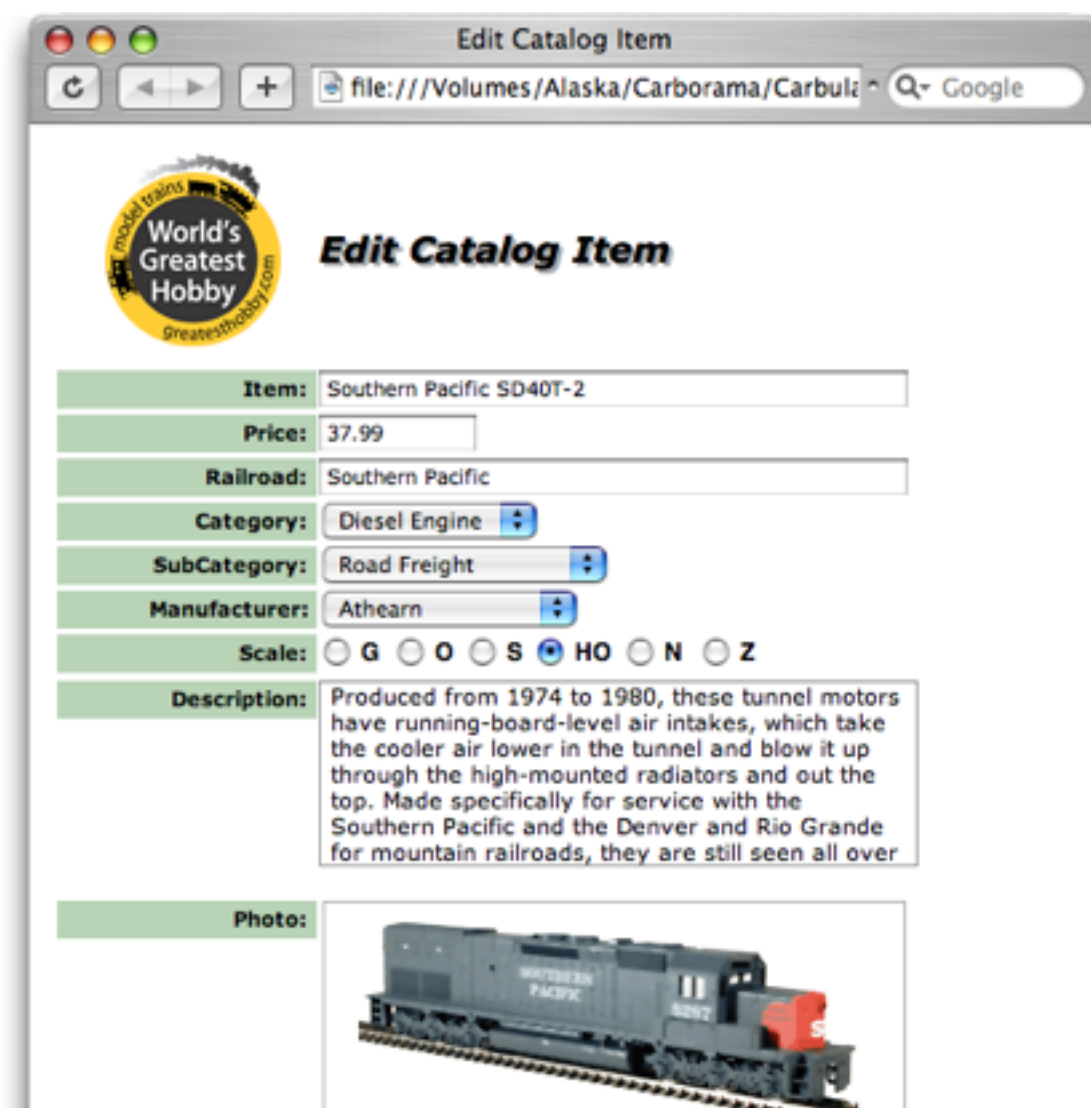


This form has a couple of links to additional pages — **Add to Shopping Cart** and **Edit Item**. Again, since this is just a simulation these links don't work in the browser.

But when you switch back to Panorama you'll see that these links appear in the simulator. You can click on the [Edit Item](#) link to open the form for editing this data.



When you release the mouse Panorama simulates the operation of the link, in this case generating a form for editing the corresponding record in the database.



If this form had been served from a web server you could edit the data in the browser and press **Submit** to update the database. However this page is really a static web page (text file) generated by the simulator so the **Submit** button doesn't work.

To continue the session click back to the **CGI Simulator** window in Panorama.

The screenshot shows a window titled "CGI Simulator:Post". It contains a "Query Options" section with a "Query Mode" dropdown set to "Post (Forms)", a "Submit Post" button, an "Action" dropdown set to "4 -- HobbyShopCatalog--updaterecord", and a "File Name" field containing "<Generated by CGI>". Below this is a "Form Fields & Data" section with several input fields:

| | |
|--------------|--|
| Item | Southern Pacific SD40T-2 |
| Description | Produced from 1974 to 1980, these tunnel motors have |
| Price | 37.99 |
| Railroad | Southern Pacific |
| Category | Diesel Engine |
| SubCategory | Road Freight |
| Manufacturer | Athearn |
| Scale | HO |
| _Record_ID_ | Item=Southern Pacific SD40T-2 |

Now the Action pop-up menu shows that we are on the fourth page of this session. All of the fields are displayed and you can edit them. We'll change the price to \$50.00 and submit.

The screenshot shows a browser window titled "Panorama Database Query:HobbyShopCatalog/updaterecord". The address bar shows a file path: "file:///Volumes/Alaska/Carborama/Carbula...". The main content area displays the following message:

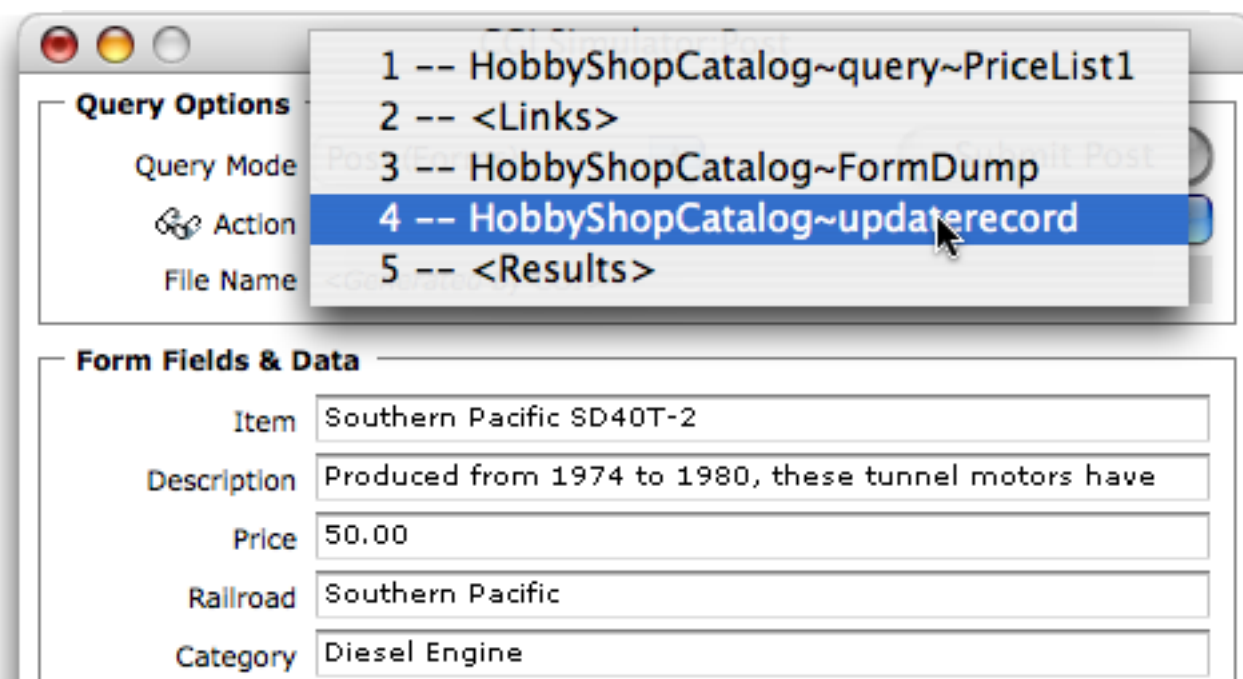
8 fields in the **HobbyShopCatalog** database have been updated:

- Item=[Southern Pacific SD40T-2](#)
- Description=[Produced from 1974 to 1980, these tunnel motors have running-board-level air intakes, which take the cooler air lower in the tunnel and blow it up through the high-mounted radiators and out the top. Made specifically for service with the Southern Pacific and the Denver and Rio Grande for mountain railroads, they are still seen all over the country.](#)
- Price=[50.00](#)
- Railroad=[Southern Pacific](#)
- Category=[Diesel Engine](#)
- SubCategory=[Road Freight](#)
- Manufacturer=[Athearn](#)
- Scale=[HO](#)

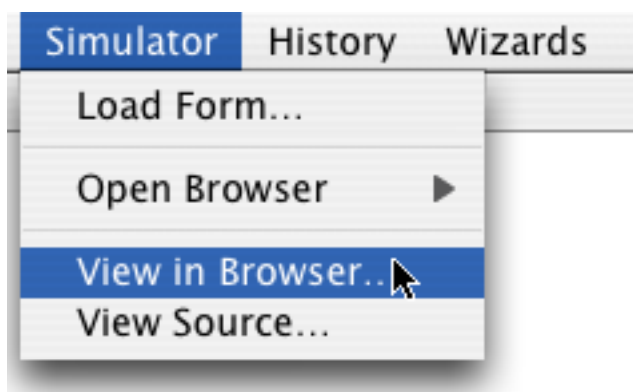
The session can continue until it reaches a dead end (no more forms or links) or you simply get tired of it.

Navigating Within a Session

Using the **Action** pop-up menu you can navigate to any page within the current session.



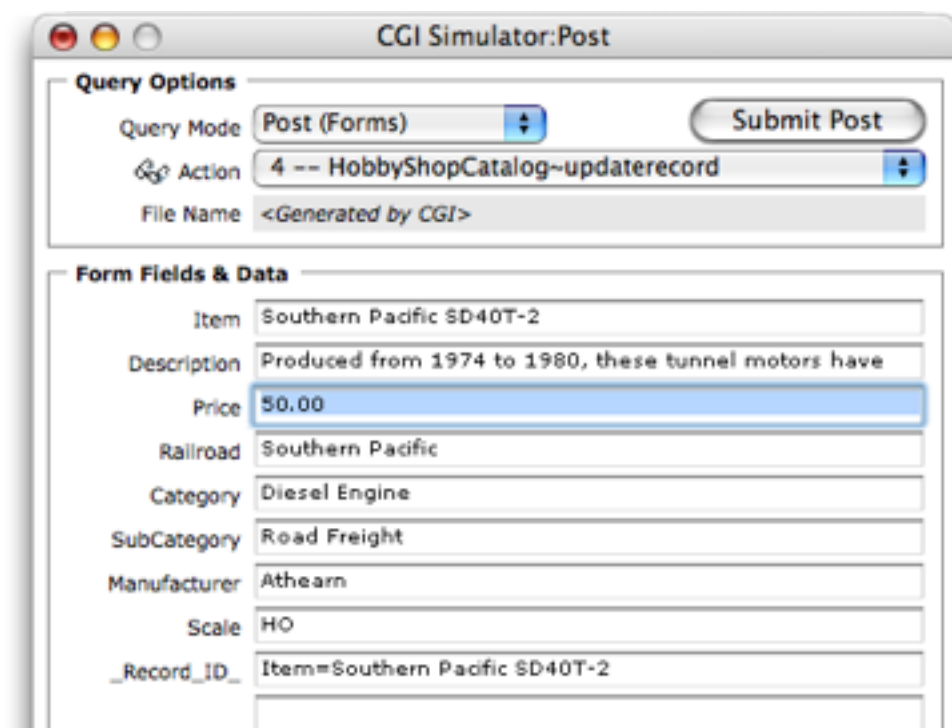
If you want to review what a page looked like use the **View in Browser** command in the **Simulator** menu.



The simulator keeps a copy of each page that is generated during a session, so it doesn't need to re-simulate the query to show you the page. You can also use the **View Source** command to see the pages actual HTML.

Re-Simulating a Previous Query

You can go back to any previous page in the session and re-run the simulation, perhaps with different data. For example, suppose you need to test the HobbyShop update procedure again. Simply use the Action pop-up menu to go back to page 4, make any changes necessary for the new test, then press **Submit Post** again.



When you re-simulate a previous page in a session any pages after that will be lost. For example, suppose you have gotten up to page 7 of a session, then go back and re-simulate page 3. The simulator will drop pages 4 through 7 of the session before running the simulation. The simulation will then create a new page 4, so the session will wind up with four pages.

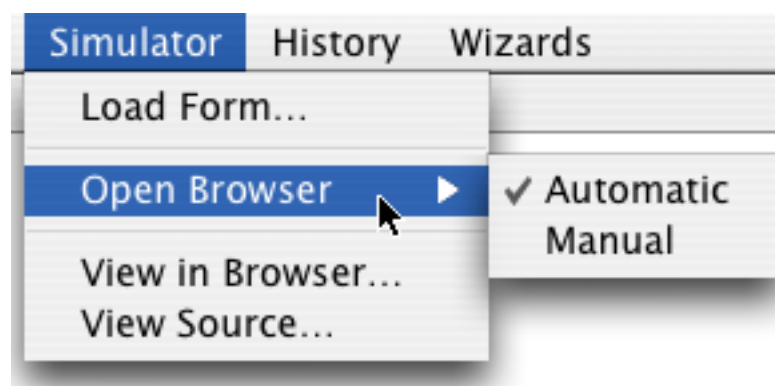
Starting a New Session

To start a new session simply change the **Query Mode** back to **Get (URL Query)**. The new session doesn't actually start until you press the **Submit Get** button, until then you can switch back to **Post (Forms)** and jump to any spot in the session.

You can also start a new session by dragging an external form onto the simulator (see "[Testing Forms in Separate HTML Files](#)" on page 358), selecting it from the **History** menu or using the **Load Form** command. Any method of loading a new form erases the previous session.

Disabling Automatic Browser Preview

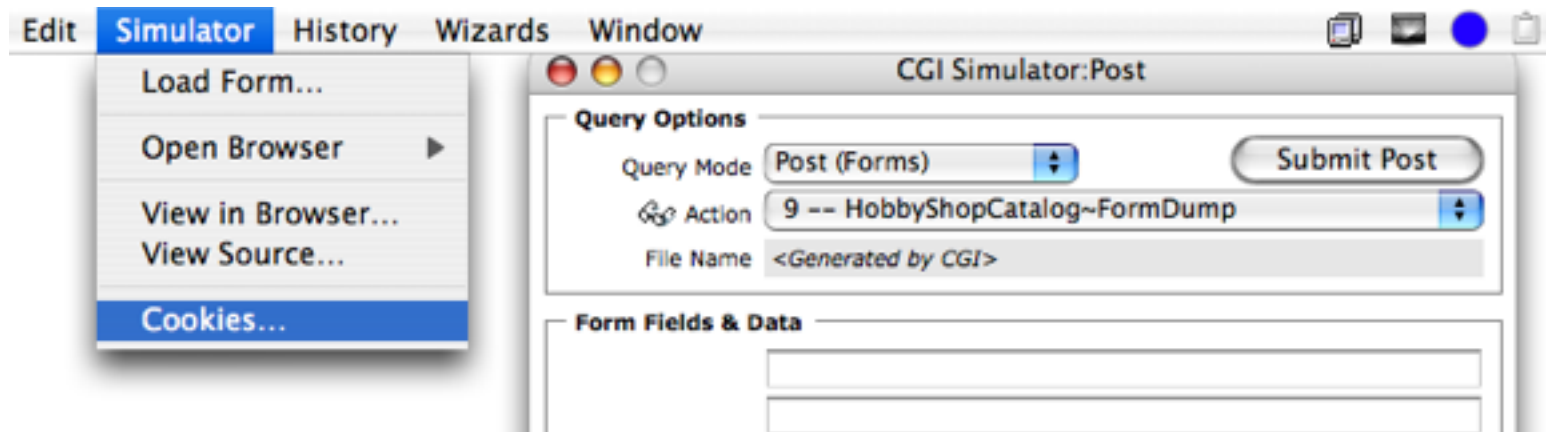
The simulator usually opens your browser to display the web page that is generated by any query. Sometimes when you are debugging a session this web page preview just slows things down. To turn off the automatic preview choose **Manual** from the **Open Browser** submenu.



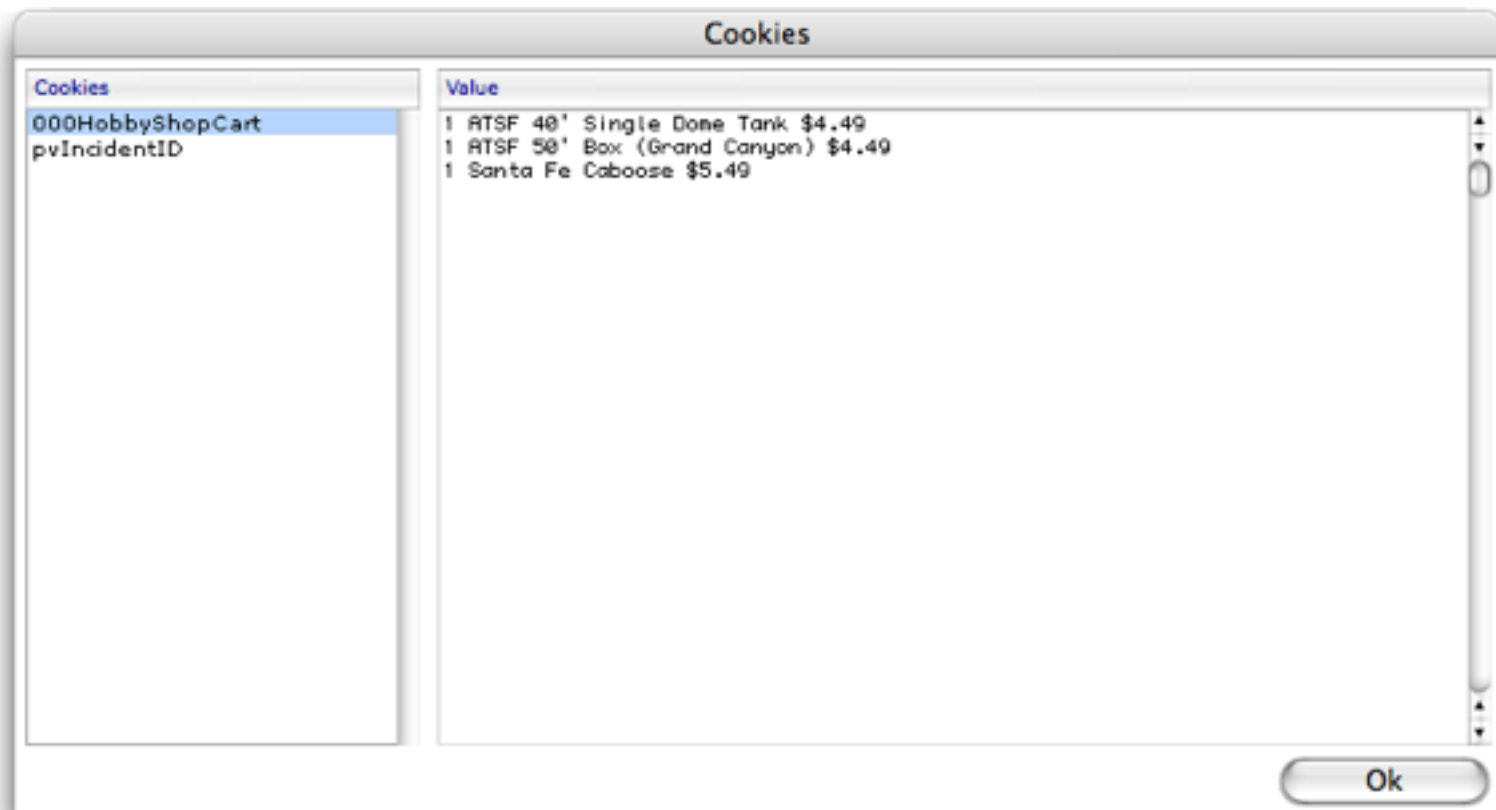
If you have disabled the automatic preview you can always preview the page manually by choosing the **View in Browser** command.

Simulating Cookies

The CGI Simulator wizard will simulate the action of a web browser for storing cookies (see “[Working with Cookies](#)” on page 453). At any time you can examine what is in your cookies by choosing **Cookies** from the Simulator menu.



This command opens a dialog that displays a list of all of the simulated cookies on the left hand side. To see the contents of a particular cookie, click on the cookie name in the list.

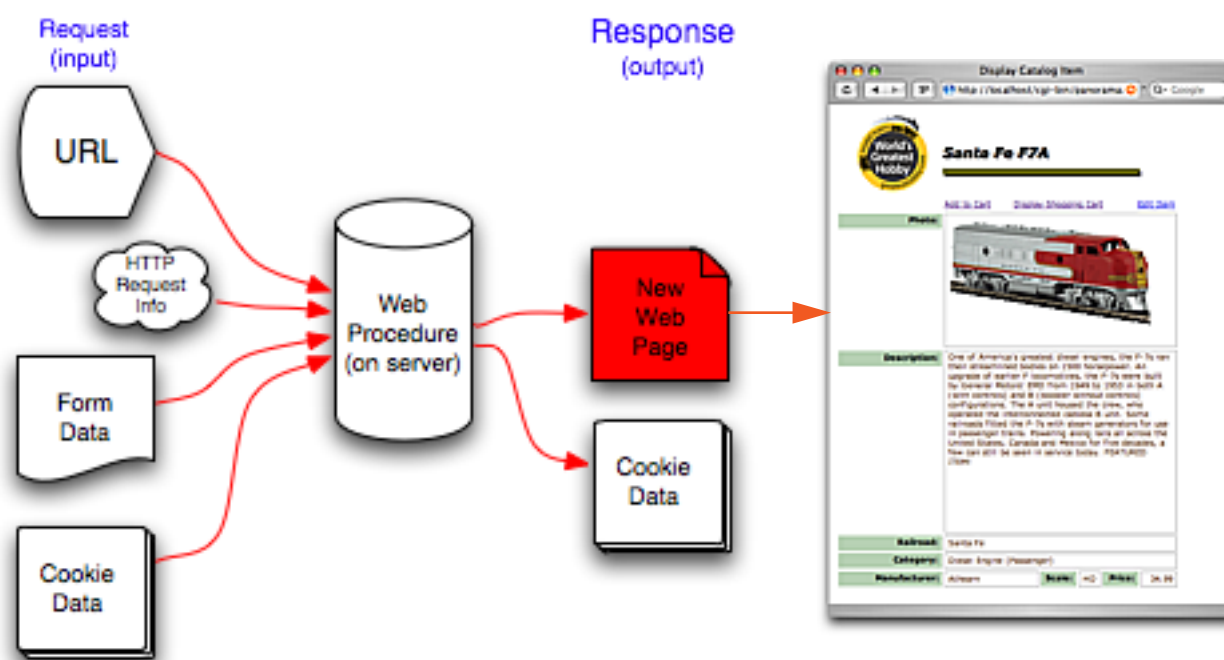


You can also click on the right hand side and edit the cookie value. When your are done, press the **Ok** button.

Chapter 9: Generating HTML



The primary job of a web procedure is to take a request from a web browser and produce an HTML page that is sent back to the web browser (the web procedure may perform other jobs as well, such as modifying a database).



This chapter covers the programming techniques available for generating an HTML page.

What is HTML?

An HTML file is a text file containing small markup tags. The markup tags tell the web browser how to display the page. If you aren't already familiar with HTML tags then stop immediately, do not pass GO, and go find your self a book or web resource for learning HTML. Your local bookstore should have a shelf full, or you can find over a hundred by searching Amazon for [html](#). Don't come back until you understand basic HTML tags like ``, `<i>`, ``, `
`, `<p>`. It would also be good to be familiar with more advanced tags like `<a>`, ``, `<form>`, `<input>` and `<table>`. The concepts are really quite simple, so it shouldn't take you very long to get the hang of it. We'll wait for you until you get back.

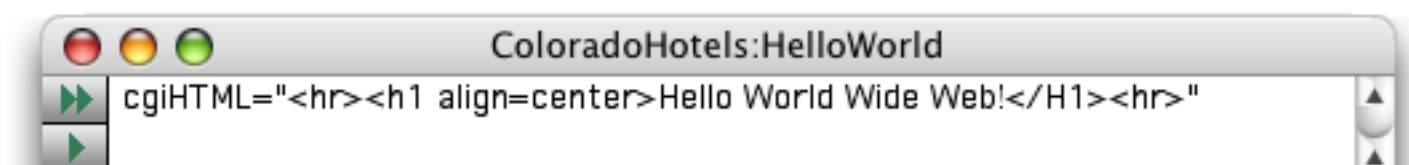
From Panorama's point of view the most important aspect of HTML is that it is made up of regular text. Panorama is really good at manipulating regular text, so it is also very good at manipulating HTML. To make it even better, Panorama Enterprise includes special functions that automatically create big chunks of HTML for you, for example automatically outputting database information as an HTML table. These special functions are described later in this chapter.

Other Web Languages (JavaScript, CSS)

HTML is the basic language of the web, but many modern web pages use additional languages including JavaScript (a programming language that runs inside web browsers) and CSS (a language for describing the appearance of a web page). From Panorama's point of view these languages all have one common attribute — they all are composed from regular text. So just like HTML, Panorama is very good at manipulating source code in these languages. Most of the techniques described in this chapter will work equally well with JavaScript, CSS, and any other web language you come across as they do with HTML. For the most part, however, we'll leave this up to you. With a few exceptions, we won't mention these additional languages again, but keep in mind that Panorama Enterprise allows them to be used anywhere HTML is used.

Directly Generating an HTML Page

When a request comes in to the server from a web browser, Panorama sets up a global variable named `cgiHTML`. This variable is initially empty (`""`). Whatever text you put into this variable will be passed back to the client web browser. The most basic web procedure can simply consist of one line that assigns some text to this variable.



Here's the result.



Of course most web procedures are more complex than this. In fact, a web procedure can take advantage of all of the features Panorama makes available for writing complex programs, including subroutines, formulas, custom statements, etc.

The previous example created a static, unchanging page. This isn't a very good use of Panorama Enterprise — after all you could simply create a .html text file to display a fixed message. Here's a more complicated procedure that displays different data depending on changing conditions.



```

local count, names

/* note: the ¶ characters are just to make it look pretty if you View Source in the browser */

cgiPageTitle=info("databasename")+ " Database Info"
cgiHTML="<h2>"+info("databasename")+"</h2>"+¶+
"Database information as of "+datepattern(today(),"Month ddnth, yyyy")+ " at "+
timepattern(now(),"hh:mm:ss am/pm")+"<p>"+¶+
pattern(info("selected"),"#, visible")+"/"+pattern(info("selected"),"#, total record~")+ "<p>"+¶

names=dbinfo("fields","")
count=arraysize(names,¶)
arrayfilter names,names,¶,"<li>"+import()
cgiHTML=cgiHTML+str(count)+" fields:"+¶+"<ul>"+¶+names+¶+"</ul>"+¶

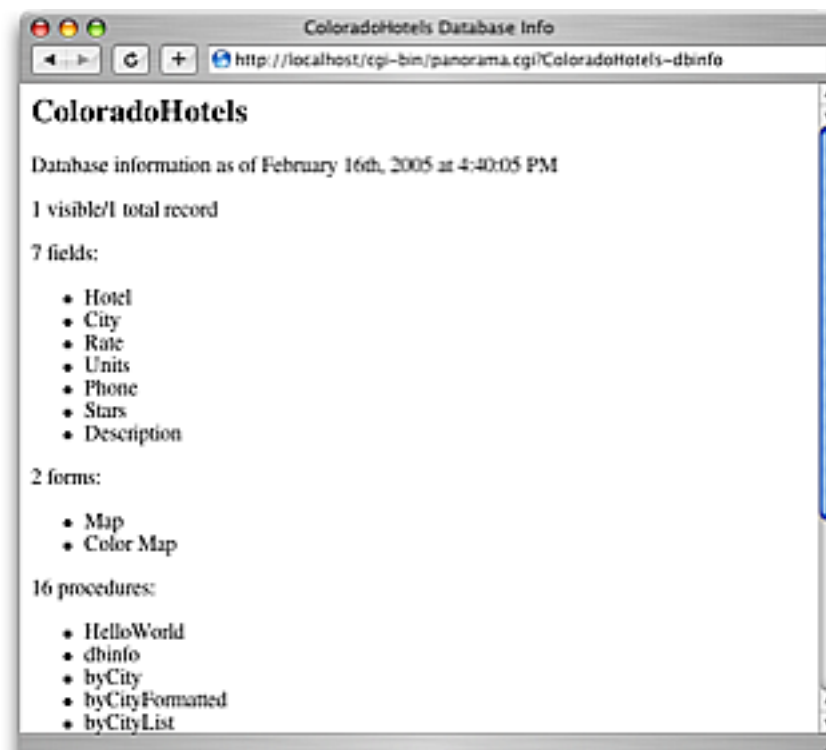
names=dbinfo("forms","")
count=arraysize(names,¶)
arrayfilter names,names,¶,"<li>"+import()
cgiHTML=cgiHTML+str(count)+" forms:"+¶+"<ul>"+¶+names+¶+"</ul>"+¶

names=dbinfo("procedures","")
count=arraysize(names,¶)
arrayfilter names,names,¶,"<li>"+import()
cgiHTML=cgiHTML+str(count)+" procedures:"+¶+"<ul>"+¶+names+¶+"</ul>"+¶

```

Procedure opened.

This procedure builds lists of fields, forms, and procedure and then uses a formula to merge them together to display all of this information on the web page:



The HTML for this page is really pretty simple. The procedure is shown on the left, with the HTML source for the page on the right. The arrows show which sections of code render which sections of the HTML. Basically each section of the page is generated as an array, which is then concatenated to the end of the page (in the `cgiHTML` variable)

The diagram illustrates the mapping between Panorama code and HTML output. The left window shows the code for 'ColoradoHotels:dbinfo' with variables like `count`, `names`, and `cgiHTML`. The right window shows the resulting HTML source with sections for fields, forms, and procedures. Blue arrows connect the code to the HTML output.

```

local count, names

/* note: the ¶ characters are just to make it look pretty if you View Source in the browser

cgiPageTitle=info("databasename")+" Database Info"
cgiHTML="<h2>" + info("databasename") + "</h2>" + ¶ +
  "Database information as of " + datepattern(today(), "Month ddth, yyyy") + " at " +
  timepattern(now(), "hh:mm:ss am/pm") + " <p>" + ¶ +
  pattern(info("selected"), "#", visible) + "/" + pattern(info("selected"), "#", total record"

names=dbinfo("fields", "")
count=arraysize(names, ¶)
arrayfilter names, names, ¶, "<li>" + import()
cgiHTML=cgiHTML+str(count)+" fields:" + ¶ + "<ul>" + ¶ + names + ¶ + "</ul>" + ¶

names=dbinfo("forms", "")
count=arraysize(names, ¶)
arrayfilter names, names, ¶, "<li>" + import()
cgiHTML=cgiHTML+str(count)+" forms:" + ¶ + "<ul>" + ¶ + names + ¶ + "</ul>" + ¶

names=dbinfo("procedures", "")
count=arraysize(names, ¶)
arrayfilter names, names, ¶, "<li>" + import()
cgiHTML=cgiHTML+str(count)+" procedures:" + ¶ + "<ul>" + ¶ + names + ¶ + "</ul>" + ¶

```

```

<html>
<head>
<title>ColoradoHotels Database Info</title>
</head>
<body bgcolor="FFFFFF" text="000000" text="0000FF" text=
<h2>ColoradoHotels</h2>
Database information as of February 16th, 2005 at 4:40:
1 visible/1 total record<p>
7 fields:
<ul>
<li>Hotel
<li>City
<li>Rate
<li>Units
<li>Phone
<li>Stars
<li>Description
</ul>
2 forms:
<ul>
<li>Map
<li>Color Map
</ul>
16 procedures:
<ul>
<li>HelloWorld
<li>dbinfo
<li>byCity
<li>byCityFormatted
<li>byCityList
<li>byCityDescriptions
<li>byCityCentered
<li>byCityDescriptionTable
<li>.SelectCity
<li>NewHotel
<li>UpdateAddHotel
<li>SelectHotels
<li>SelectInPriceCategory
<li>DisplayRecord
<li>SelectDisplayEdit
<li>ServerParameters
</ul>

</body>
</html>

```

If you are not already familiar with Panorama's array statements and functions you should review the sections of the Panorama Handbook that describe these features. Arrays are often very useful when rendering HTML code.

Customizing the HTML Page Header

HTML pages normally have a header and footer that specifies things like the page title, background color, meta tags, etc. Here's a typical example of what an HTML page header looks like:

```
<html>
<head>
<title>Colorado Hotels Database Information</title>
</head>
<body bgcolor="FFFFFF" text="000000" link="0000FF" vlink="336666">
```

Here's what a footer looks like:

```
</body>
</html>
```

If you've been following closely you may have noticed that the HTML pages created by the web procedures in the previous section DID NOT generate this header and footer — they only generated the contents of the body of the page (the stuff between the `<body>` and `</body>` tags). This is ok because if the header and footer are missing, Panorama Enterprise will automatically create them for you and attach them to the HTML body that you place in the `cgiHTML` variable. One less thing for you to worry about!

Of course, sometimes you want to worry about the header and footer sections because you want to customize them. Perhaps you want to set up a custom page title, or include some JavaScript in the header. There are two ways to do this. One is to simply go ahead and include the header and footer sections in the `cgiHTML` variable. If you do this, Panorama Enterprise won't create its own header and footer sections, it will just pass your page as is back to the web browser.

In most cases, however, it's easier to let Panorama Enterprise create the header and footer sections for you and customize by modifying up to seven special variables that control how the header is created. These special variables are listed in the table below.

| Variable | Description |
|---------------------------------|--|
| <code>cgiPageTitle</code> | This variable contains the page title. This title will appear in the drag bar of the browser window. |
| <code>cgiBackColor</code> | This variable controls the background color. The color may be set to a named color ("white", "blue", etc.) or to an HTML format hexadecimal format ("CCFFFF", "996633", etc.) |
| <code>cgiTextColor</code> | This variable controls the text color. The color may be set to a named color ("white", "blue", etc.) or to an HTML format hexadecimal format ("CCFFFF", "996633", etc.) |
| <code>cgiLinkColor</code> | This variable controls the color of any links on the page. The color may be set to a named color ("white", "blue", etc.) or to an HTML format hexadecimal format ("CCFFFF", "996633", etc.) |
| <code>cgiVisitedColor</code> | This variable controls the color of any links on the page that have already been visited. The color may be set to a named color ("white", "blue", etc.) or to an HTML format hexadecimal format ("CCFFFF", "996633", etc.) |
| <code>cgiExtraHeader</code> | The contents of this variable will be inserted inside the header after the <code><title></code> tag. This allows you to add additional tags in the header, for example for meta tags or for loading a javascript file. |
| <code>cgiExtraBodyOption</code> | The contents of this variable will be inserted inside the <code><body></code> tag. this allows you to add additional options into the body tag, for example <code>onload=<javascript></code> etc. |

Here is a revised version of the dbinfo procedure that customizes the page title, background color and text color (the assignments to these variables can occur anywhere in the procedure):

```

local count, names

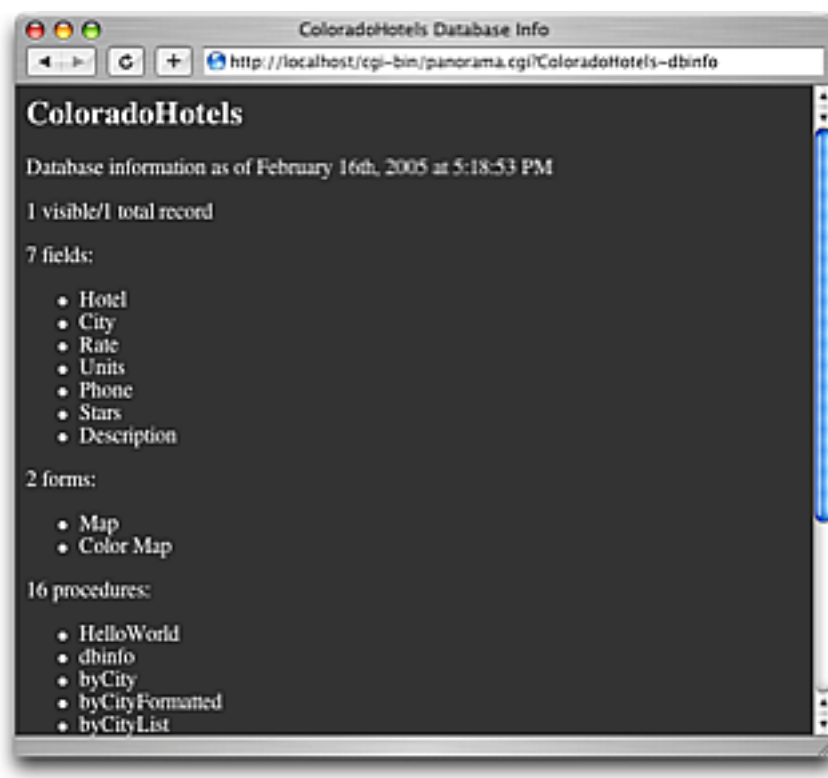
/* note: the ¶ characters are just to make it look pretty if you View Source in the browser

cgiPageTitle=info("databasename")+ " Database Info"
cgiBackColor="333333"
cgiTextColor="FFFFFF"

cgiHTML="<h2>"+info("databasename")+"</h2>"+¶+
"Database information as of "+datepattern(today(),"Month ddnth, yyyy")+ " at "+

```

This revised web procedure displays the same information but with white text on a dark gray background.



The “extra” variables, `cgiExtraHeader` and `cgiExtraBodyOption` allow you to insert just about anything you want into the header without having to actually code your own header and footer. This is especially handy if you want to use JavaScript functions in your page — the definitions for the functions should be put in the `cgiExtraHeader` variable.

Placing Fields and Variables into the HTML Page

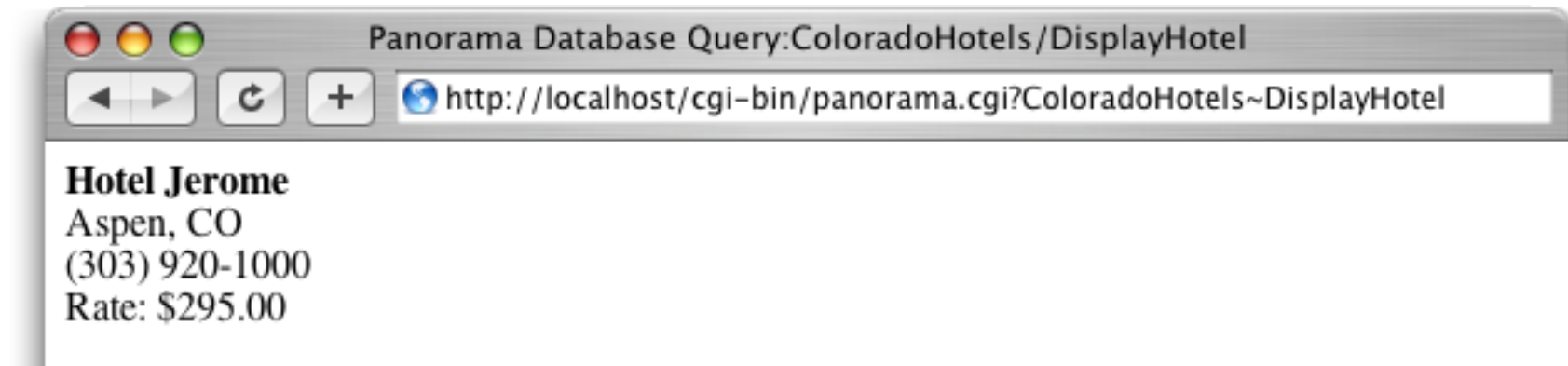
To place fields or variables into the page simply create a formula that combines the HTML tags you need with variables and fields from the database. If a variable or field contains numeric data you must convert it to text with the `str()` or `pattern()` function, as shown below for the `Rate` field. Here is a simple procedure that displays the current record. (Of course this example is really a bit too simple, in any real application you would want to specify what record to display rather than simply displaying whatever happened to be the current record – see “[What Record Are We Talking About?](#)” on page 395.)

```

cgiHTML="<b>"+Hotel+"</b><br>"+
City+", CO<br>( 303) "+Phone+"<br>"+
"Rate: $"+str(Rate)

```


Here's what this page will look like in the browser.



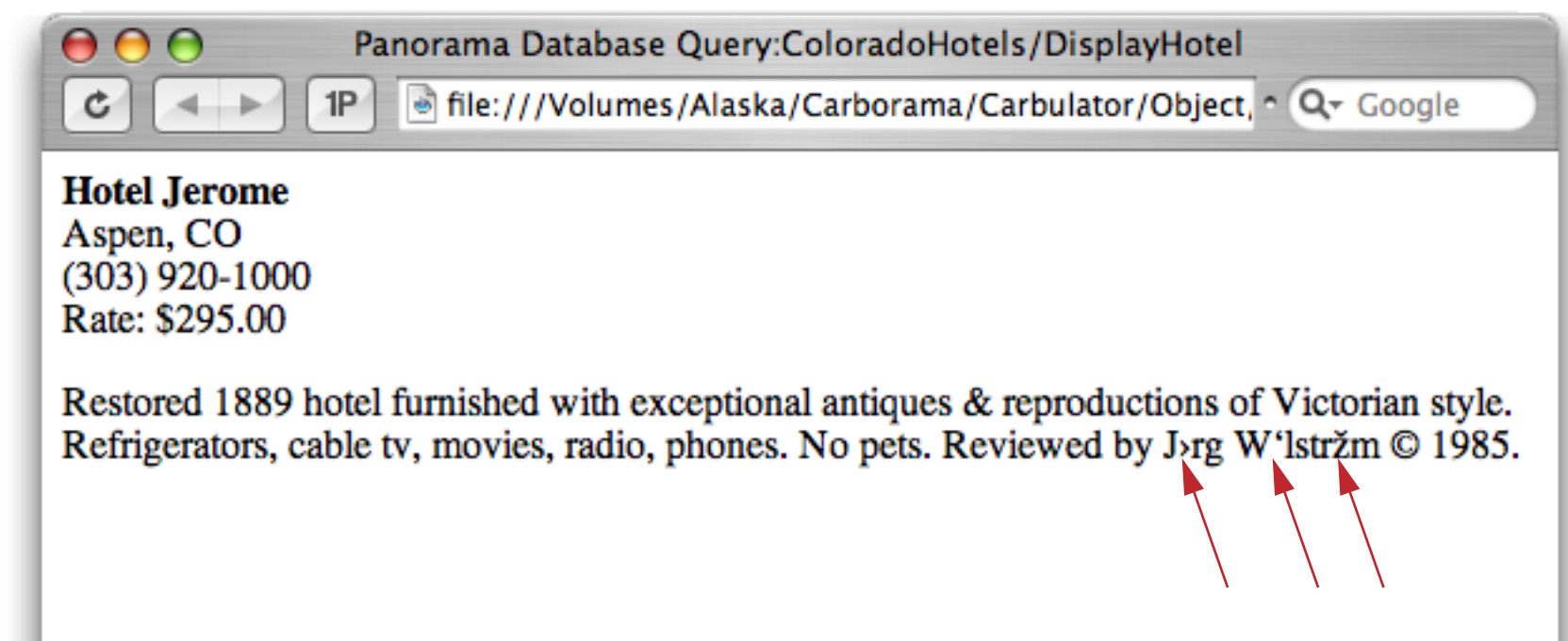
Of course this example is really a bit too simple — it doesn't actually specify what record to display. It blithely goes ahead and displays whatever record the server happens to be sitting on at the moment. In a real application the procedure would first locate the record to display using the `find`, `findbackwards`, `weburlfind`, `select`, `webselect` or `weburlselect` statements. For more on specifying the record to display see “[What Record Are We Talking About?](#)” on page 395.

Fields or Variables with Special Characters

If a field or variable might contain “special” characters you must convert the text to html with the `webtext()` function. You'll need to use this function if the data might contain accented characters or special characters like `<`, `>`, `©`, `®`, `¶`, `§` etc. When in doubt it's best to use the `webtext()` function. For example, suppose the `Description` field of a database contains accented and special characters, like this:

| ColoradoHotels | | | | | |
|----------------|--------|-------|----------|----|--|
| City | Rate | Units | Phone | St | Description |
| Aspen | 295.00 | 94 | 920-1000 | 3 | Restored 1889 hotel furnished with exceptional antiques |
| Denver | 129.50 | 34 | 555-9876 | 2 | & reproductions of Victorian style. Refrigerators, cable |
| Aspen | 59.00 | 38 | 925-3221 | 4 | tv, movies, radio, phones. No pets. Reviewed by Jörg |
| Alamosa | 67.00 | 34 | 555-1212 | 3 | Wëlstrôm © 1985. |

Without the `webtext()` function, these characters will not look right on a web browser



To fix this, use the `webtext()` function to convert the `Description` field to proper HTML format.

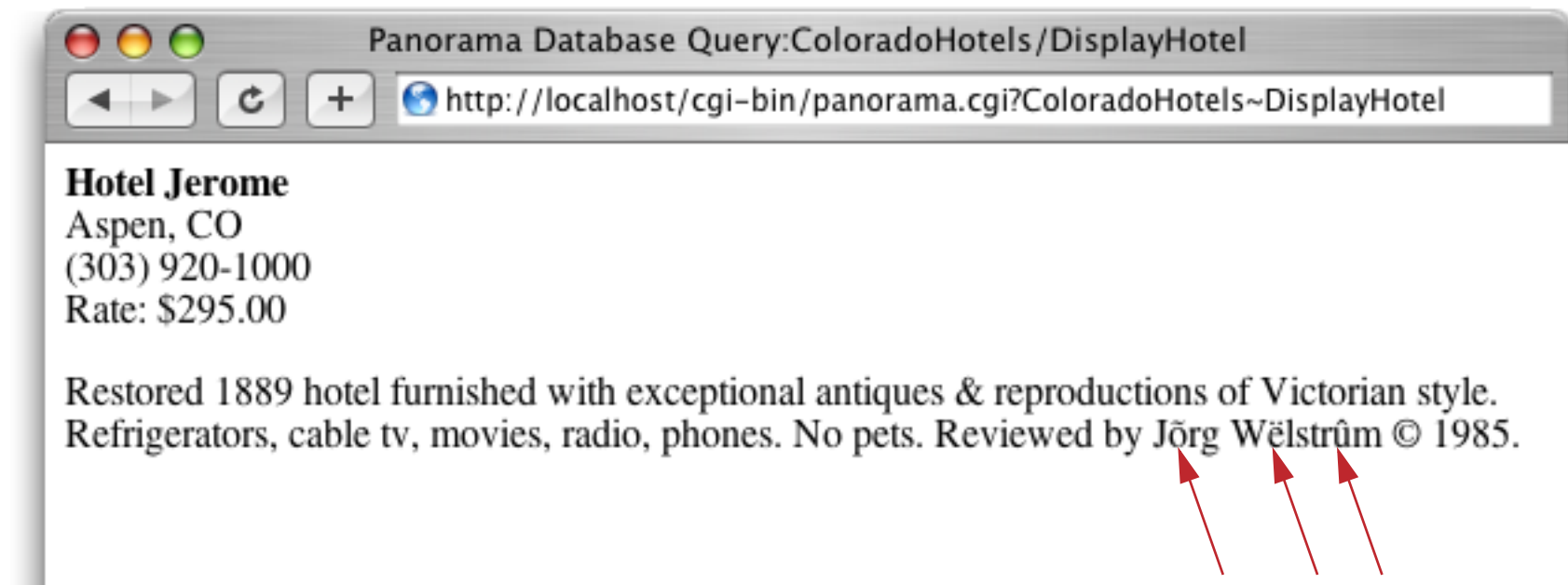


```

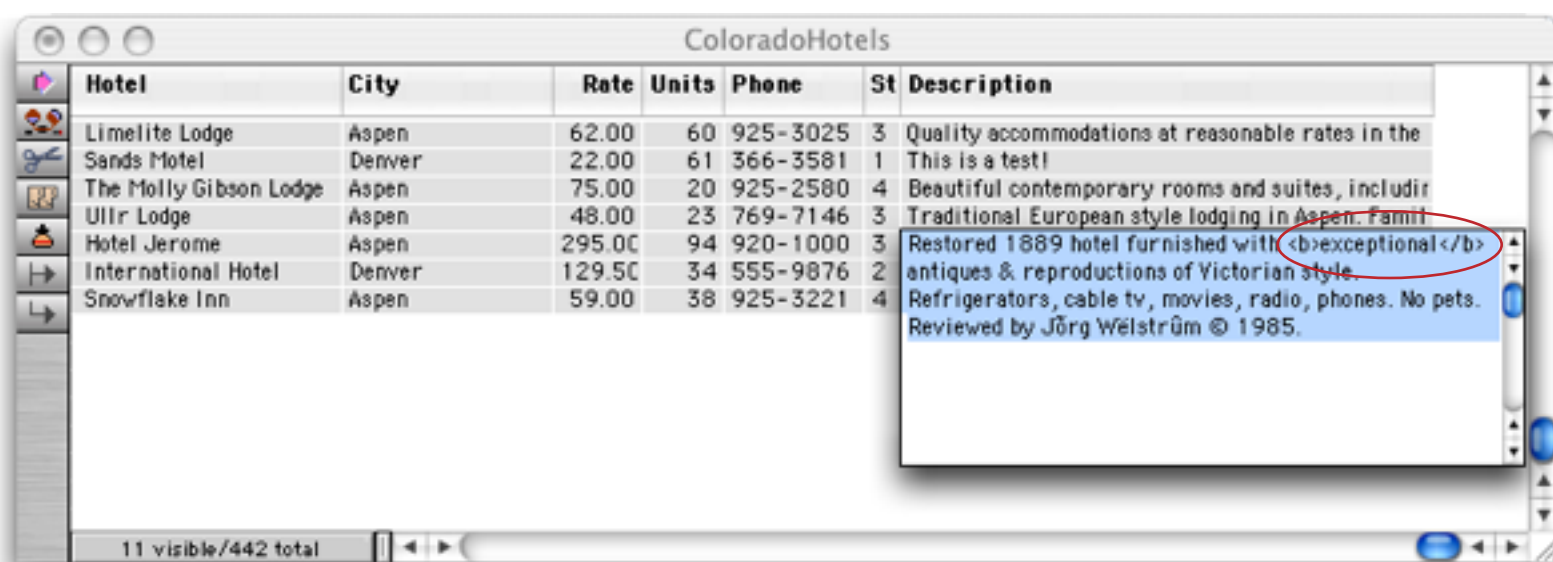
cgiHTML="<b>"+Hotel+"</b><br>"+
City+", CO<br>( 303) "+Phone+"<br>"+
"Rate: $"+str(Rate)+
+"<p>"+webtext(Description)

```

Now the accented and special characters to display correctly, as shown below.

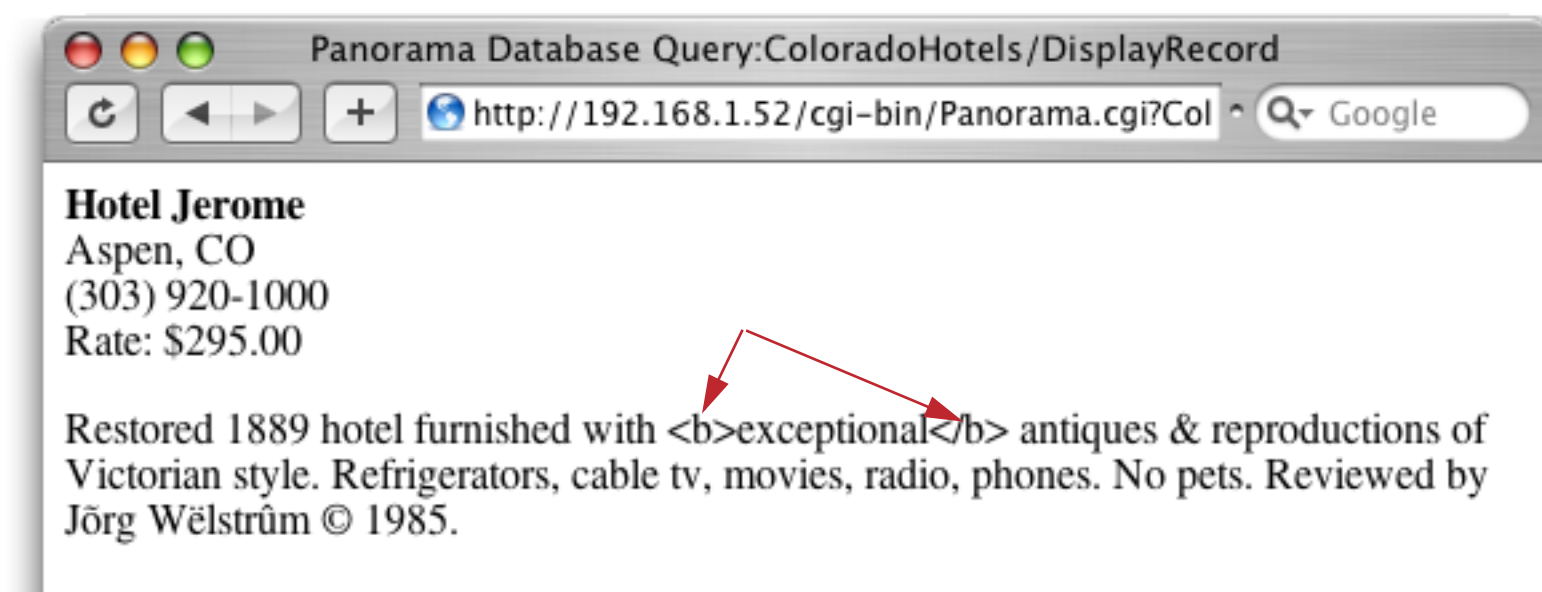


The `webtext()` function will disable any HTML tags in the text being converted. For example, suppose the `Description` field in the previous example contained tags to make the word **exceptional** bold.

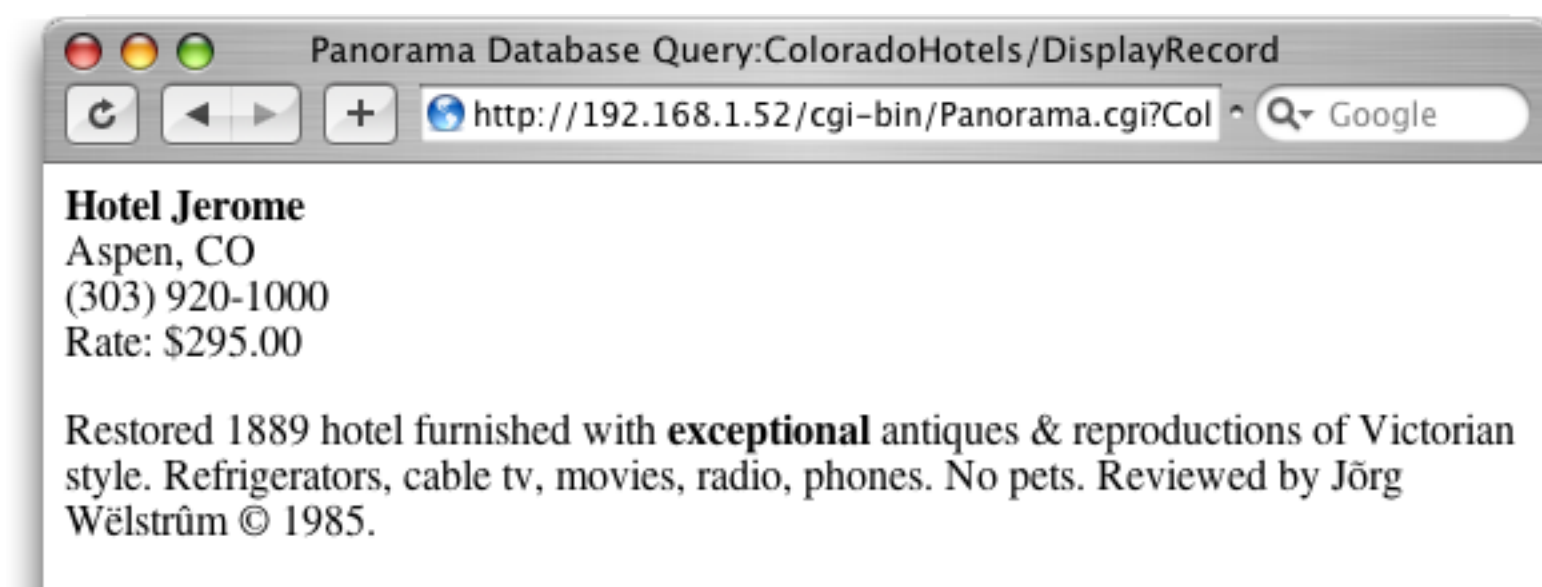


| Hotel | City | Rate | Units | Phone | St | Description |
|------------------------|--------|--------|-------|----------|----|---|
| Limelite Lodge | Aspen | 62.00 | 60 | 925-3025 | 3 | Quality accommodations at reasonable rates in the |
| Sands Motel | Denver | 22.00 | 61 | 366-3581 | 1 | This is a test! |
| The Molly Gibson Lodge | Aspen | 75.00 | 20 | 925-2580 | 4 | Beautiful contemporary rooms and suites, includir |
| Ullr Lodge | Aspen | 48.00 | 23 | 769-7146 | 3 | Traditional European style lodging in Aspen. Famil |
| Hotel Jerome | Aspen | 295.00 | 94 | 920-1000 | 3 | Restored 1889 hotel furnished with exceptional |
| International Hotel | Denver | 129.50 | 34 | 555-9876 | 2 | antiques & reproductions of Victorian style. |
| Snowflake Inn | Aspen | 59.00 | 38 | 925-3221 | 4 | Refrigerators, cable tv, movies, radio, phones. No pets. Reviewed by Jörg Wëlstrûm © 1985. |

The `webtext()` function will convert the `<` and `>` characters to `<` and `>`, so these characters will be displayed rather than being interpreted as tags.



If you want to be able to embed HTML tags in the text use the `htmlencode()` function instead of the `webtext()` function. As shown below, the `htmlencode()` function will allow most special characters to display correctly but leaves the `<` and `>` characters alone so they can be used for HTML tags. The only problem with this function is that `<` and `>` characters that are not part of tags will not display correctly, for example if the text says `2006 sales > 2005 sales` the `>` character will not display (the text would have to be edited to `2006 sales > 2005 sales` to make it display correctly in the browser).



Another approach is to use the `webtagortext()` function instead of the `webtext()`. The `webtagortext()` function works just like the `webtext()` function unless the data contains tags. If the data does contain tags, no conversion at all is done, and it's up to the person doing the data entry to make sure that the text will display correctly (for example converting special characters to HTML entities). The advantage of the `webtagortext()` function is that for regular data entry (no tags) everything is done automatically, and no knowledge of HTML is required. If someone wants to enter tags into their data then it is assumed that they know what they are doing and that they will do all necessary HTML conversion themselves. So both novices and power users get what they want.

Links to Other Web Pages

In HTML hypertext links to other web pages are made with the `<a>` tag. This tag can be included in the rendered web page like any other text.

```
cgiHTML = ... + "<a href='http://www.somesite.com/somepage.html'>Some Caption</a>" + ...
```

The `weblink()` function provides an alternative method to generate an hypertext link (`<a>` tag). This function has two parameters, `url` and `caption`. The `url` is simply the URL of the page this tag will link to. The `caption` is the text that appears on the browser for this link. (If the caption is blank `""`, the URL will be used as the caption.) Here is revised code that uses the `weblink()` function to create the same hypertext link as the example above.

```
cgiHTML = ... + weblink("http://www.somesite.com/somepage.html","Some Caption") + ...
```

The `cgilink()` function generates a hypertext link to a Panorama Server action on this server. In addition to being more compact than writing the link by hand, this function also automatically adjusts the generated URL to reference the current server. That way if the database is later moved to a different server any links generated by this function will continue to work. The `cgilink()` function has three parameters:

```
cgilink(database,action,caption)
```

The `database` parameter is the name of the database to be accessed.

The `action` parameter is action portion of the url (see “[Web Database URL Format](#)” on page 220). This could be a standard action (form, addrecord, etc.) or a procedure name, optionally followed by `~` and any extra parameters.

The `caption` parameter is simply the text that will be displayed on the browser for this link.

This example shows how the `cgilink()` function can be used. This code generates a hypertext link to the `AddItemToCart` procedure in the `ShoppingCart` database.

```
cgiHTML = ... + cgilink("ShoppingCart","AddItemToCart","Add To Cart")
```

If your web site is `www.somesite.com` then the formula above is equivalent to the formula below. Much simpler, don't you think!

```
cgiHTML = ... +
  "<a href='http://www.somesite.com/cgi-bin/Panorama.cgi?ShoppingCart~AddItemToCart'>Add
  To Cart</a>" + ...
```

The `webformlink()` function generates a hypertext link to a Panorama form on this server. Like the `cgilink()` function, this function also automatically adjusts the generated URL to reference the current server. The `webformlink()` function has four parameters:

```
webformlink(database,form,record,caption)
```

The `database` parameter is the name of the database to be accessed.

The `form` parameter is the name of the web form to link to. This must be a Panorama form that has been converted to an HTML form (see “[Web Forms](#)” on page 231).

The `record` parameter specifies what record (if any) to display in the linked form. If this parameter is left empty (`""`) the form will be blank (for example for adding a new record or searching). Otherwise this parameter is passed internally to the `renderwebform()` function or `weburlselect` statement which locates the requested record. See “[What Record Are We Talking About?](#)” on page 395 for more information about how to use this parameter.

The `caption` parameter is simply the text that will be displayed on the browser for this link.

This example shows how the `webformlink()` function can be used. This code generates a hypertext link to the [Full Search](#) procedure in the [Catalog](#) database.

```
cgiHTML = ... + webformlink("Catalog","Full Search","", "Advanced Search")
```

The `weblink()`, `cgilink()`, and `webformlink()` functions generate links that normally open in the same browser window. In other words, the new page appears in the same window as the current page. If you want the new page to appear in a new, separate window you can use the alternate `weblinknewwindow()`, `cgilinknewwindow()`, and `webformlinknewwindow()` functions instead.

Images

In HTML images are displayed with the `` tag. This tag can be included in the rendered web page like any other text.

```
cgiHTML = ... + "<img src='http://www.somesite.com/someimage.jpg'>" + ...
```

If the image is on the same server as the database you can use the `dbserverdomain()` function instead of hard coding the domain name. This function returns the domain name of the current server.

```
cgiHTML = ... + "<img src='"+dbserverdomain()+"/someimage.jpg'>" + ...
```

Using the `dbserverdomain()` function allows the image to be displayed correctly if both it and the database are moved to a different server.

Generating a Page Using a Panorama Form Template

If you want to, you can create web pages of unlimited complexity simply by string together complicated formulas. Doing it this way gives you full control over every character in the final HTML page, but it's a lot of work. It also requires a lot of imagination, since you can't see the page as you are building it. The only way to see the result is to run (or simulate) your procedure.

There is another way. Instead of generating the HTML page entirely on your own from scratch, start with a Panorama web form (see "[Web Forms](#)" on page 231). This allows you to layout your page with Panorama's graphic tools. Your web procedure can use the web form as a template, which it can then customize further depending on the particular application. This technique can often reduce page of code to just a few lines, and it makes it much easier to modify the layout of your data as needed.

The first step is to create and render the web form for use with the procedure. Creating web forms is covered in detail in Chapter 6. When you are creating a standalone web form it will usually only include database fields, but a form web form designed for use with a procedure will often include variables in Text Editor SuperObjects, Text Display SuperObjects, and Flash Art SuperObjects. More on this in a moment (see "[Using Variables to Customize a Web Form on the Fly](#)" on page 383).

The RenderWebForm(Function

To display a Panorama web form use the `renderwebform()` function in the formula that creates the HTML page. This function has two parameters.

```
renderwebform(formname,recordid)
```

The first parameter is the name of the form, which must be in the same database as the procedure, and must already have been converted to HTML (see "[Converting a Panorama Form into a Web Form](#)" on page 231).

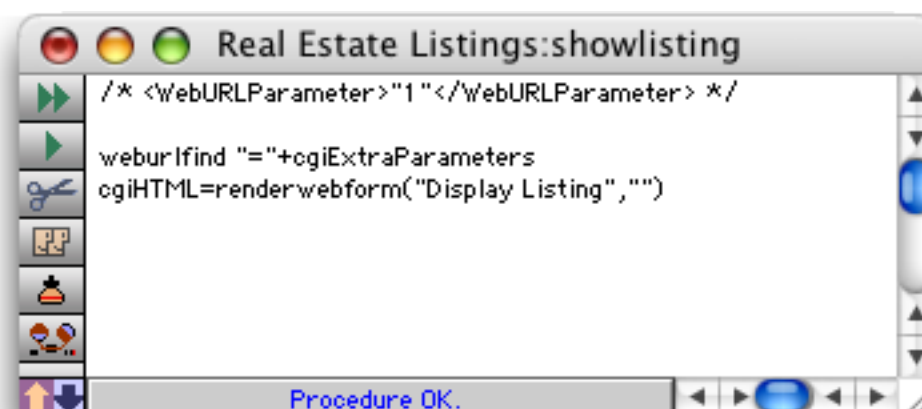
The second parameter is the record identifier. If this is empty ("") then a blank form will be produced. If any data is displayed on the form it will come from the current record. If the second parameter is not empty it must be a record identification string compatible with the `weburlselect` statement. We'll talk more about this parameter in a moment (see "[What Record Are We Talking About?](#)" on page 395).

Using a Web Form to Display Data

The simplest application for the `renderwebform()` function is to display a form that simply displays data and does not contain any editable items. In other words, the form only uses Text Display SuperObjects and Flash Art SuperObjects, and does not contain any Data Cells, Text Editor SuperObjects, checkboxes, radio buttons, pop-up menus, etc. Here's an example.



Here is a simple procedure that will display a record using this form.



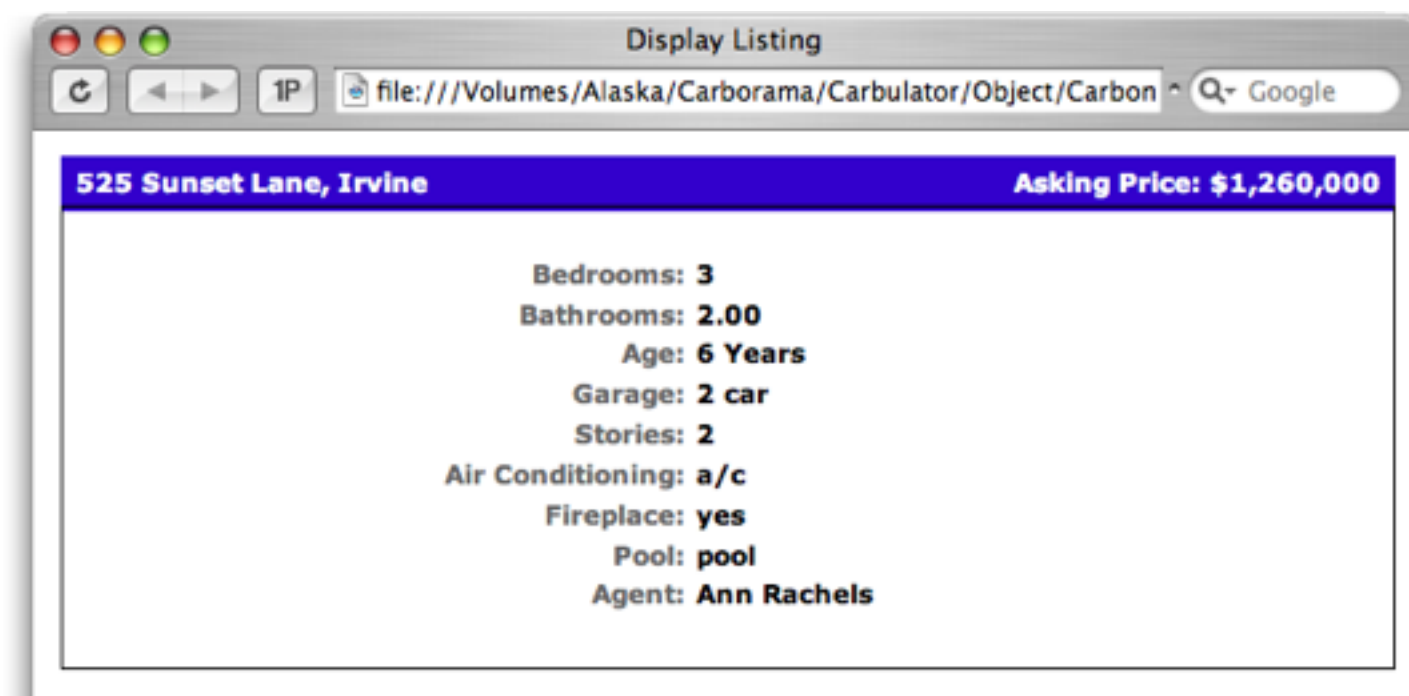
The first non-comment line in this procedure:

```
weburlfind "="+cgiExtraParameters
```

determines what record will be displayed. We'll talk more about that later (see the next section), in fact, we'll show how this line can be eliminated. The second line:

```
cgiHTML=renderwebform("Display Listing", "")
```

actually combines the data from the database with the web form template to create the final HTML page. Here's the result displayed in a web browser (either by using **Simulate Web Procedure** or actually running this procedure on your web server):



I think you'll agree that creating this web page this way is much easier than generating all of the HTML by hand!

Specifying the Record to Display

In the previous example the second parameter to the `renderwebform()` function was left blank.

```
cgiHTML=renderwebform("Display Listing","")
```

This tells the function to display the contents of the current record, whatever that is. It's up to you to make sure that the correct record is already "spotted" (either by finding or selecting it) before the `renderwebform()` function is used. In the previous example this was done by using the `weburlfind` statement first, before the `renderwebform()` function.

```
weburlfind "="+cgiExtraParameters
cgiHTML=renderwebform("Display Listing","")
```

We'll talk more about the `weburlfind` statement a bit later (see "[Using WebURLFind For Navigation in Shared Databases](#)" on page 398), but for now all you need to know is that this statement takes the extra parameter from the URL and uses it to find a specific record in the database. There's one other thing you should know — the `weburlfind` statement is built into the `renderform()` function! This means that you can take the parameter from the `weburlfind` statement and move it to the second parameter of the `renderwebform()` function, then get rid of the `weburlfind` statement! Now the entire procedure is only one line long.

```
cgiHTML=renderwebform("Display Listing","@")
```

Note: The specific use of the "@" format for this parameter will only work if the database is shared as well as web published. If the database is web published only you'll need to use a different format, see "[Using WebURLFind for Navigation in Non-Shared Databases](#)" on page 401.

Customizing the Web Form's HTML Header (Page Title, etc.)

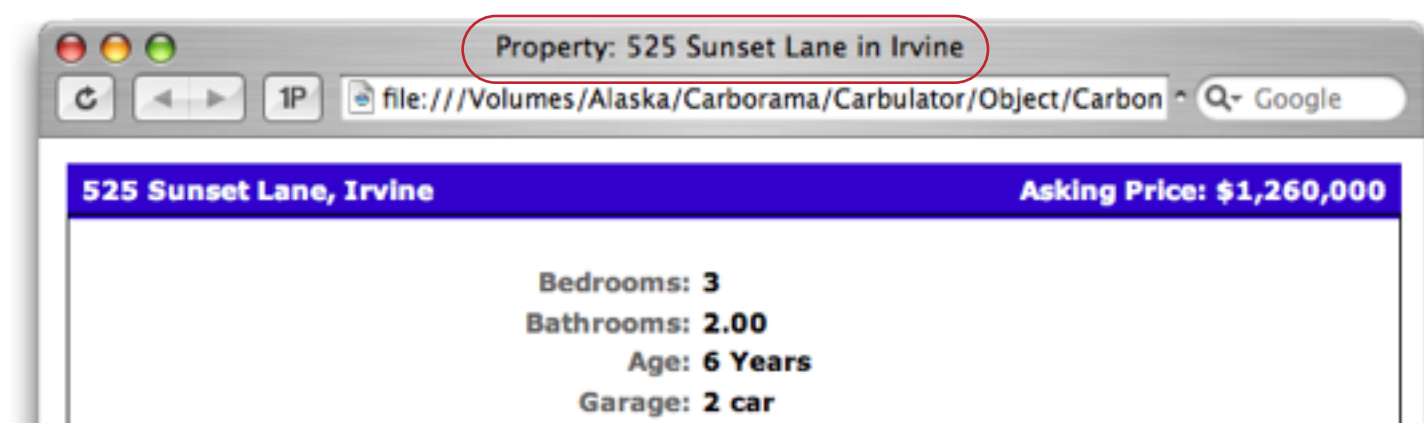
When a Panorama web form is rendered it sets up the HTML header (page title, background color, etc.) up for you. A procedure can, however, override some of these options by this using the variables described in “[Customizing the HTML Page Header](#)” on page 373. This must be done *before* the `renderwebform()` function is used, not after. For example, here is a modified version of the `showlisting` procedure that displays the property address in the browser title bar.

```

/* <WebURLParameter>"1"</WebURLParameter> */
weburlselect "+cgiExtraParameters
cgiPageTitle="Property: "+Address+" in "+City
cgiHTML=renderwebform("Display Listing", "")

```

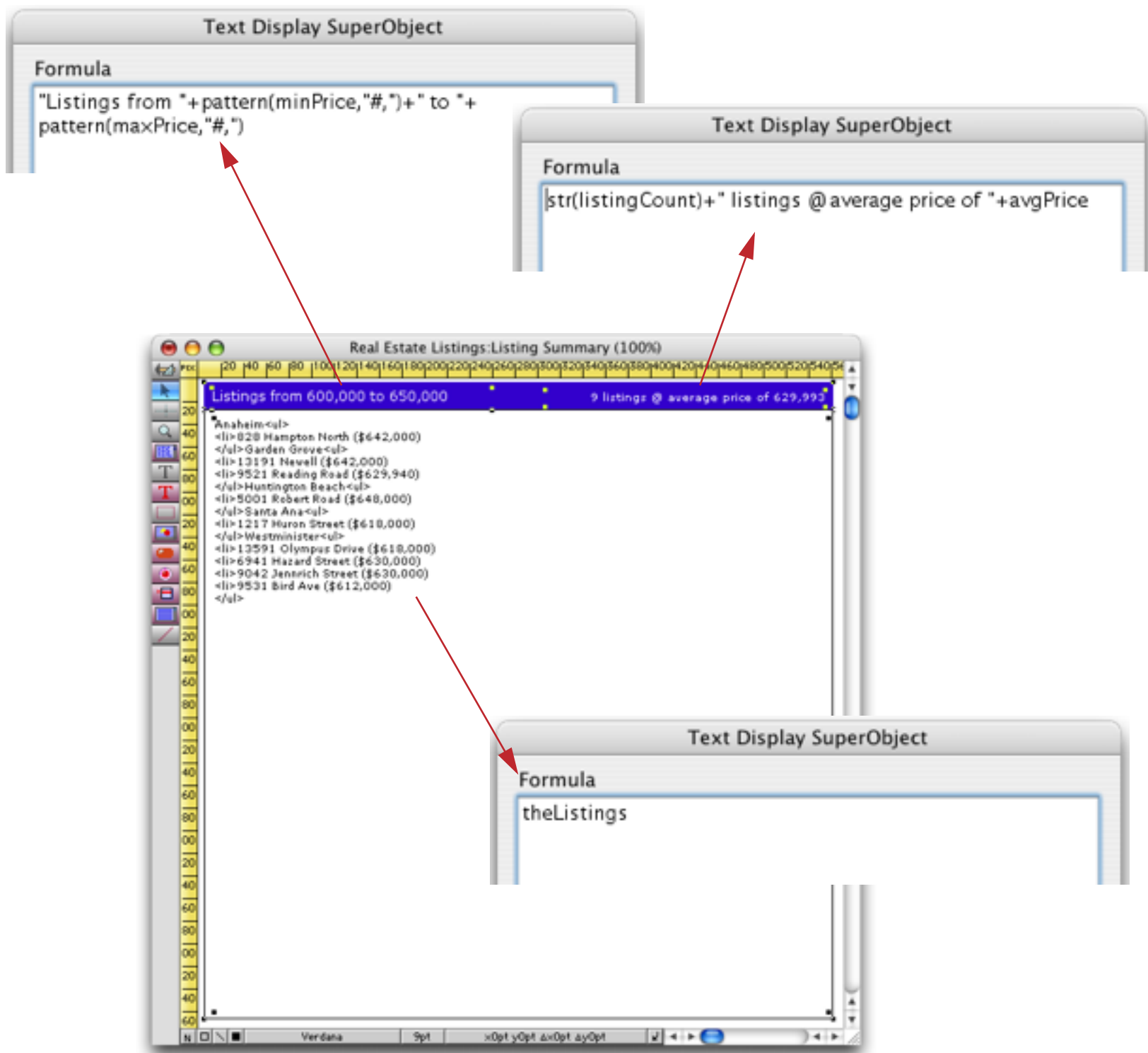
Here is the result in the browser.



In addition to the page title you can also set the `cgiExtraHeader` and `cgiExtraBodyOption` variables. However you cannot modify the text and background colors, as those are preset by the form template.

Using Variables to Customize a Web Form on the Fly

One easy way to customize a form with a procedure is to set up the form with Text Display SuperObjects and Flash Art SuperObjects that display one or more variables. The procedure sets up the variable values before rendering the form. Here is a form that contains three Text Display SuperObjects. As you can see, the procedure that displays this form will need to calculate five variables before rendering the form: `minPrice`, `maxPrice`, `listingCount`, `avgPrice` and `theListings`.



Here is the web procedure that calculates these variables.

```

Real Estate Listings:listings
/* <WebURLParameter>"600000~650000"</WebURLParameter> */
fileglobal minPrice,maxPrice,avgPrice,listingCount,theListings
local templist,n,city,lastcity,tempprices,at,ac
minPrice=val(array(cgiExtraParameters,1,""))
maxPrice=val(array(cgiExtraParameters,2,""))
select Asking>=minPrice and Asking<=maxPrice
arrayselectedbuild templist,cr(),"","fixedwidth(City,20)+tab()+Address+tab()+str(Asking)
arraysort templist,templist,cr()
arrayfilter templist,tempprices,cr(),array(import(),3,tab())
at=float(arraynumerictotal(tempprices,cr()))
listingCount=float(arraysize(tempprices,cr()))
avgPrice=pattern(at/listingCount,"#,"")

n=1 lastcity=""
theListings=""<ul>
loop
  city=strip(array(nthline(templist,n),1,tab()))
  stoploopif city=""
  if city<>lastcity
    theListings=theListings+ "</ul>" + city + "<ul>" + cr()
    lastcity=city
  endif
  theListings=theListings+
    "<li>" + array(nthline(templist,n),2,tab()) +
    "(" + pattern(val(array(nthline(templist,n),3,tab())),"$#,"") + ")" + cr()
  n=n+1
while forever
theListings=theListings+ "</ul>"
theListings=replace(theListings,"<ul></ul>","")
cgiHTML=renderwebform("Listing Summary","")

selectall
    
```

Here's the final result that combines the calculations with the web form.

Listings from 600,000 to 650,000 9 listings @ average price of 629,993

- Anaheim
 - 828 Hampton North (\$642,000)
- Garden Grove
 - 13191 Newell (\$642,000)
 - 9521 Reading Road (\$629,940)
- Huntington Beach
 - 5001 Robert Road (\$648,000)
- Santa Ana
 - 1217 Huron Street (\$618,000)
- Westminister
 - 13591 Olympus Drive (\$618,000)
 - 6941 Hazard Street (\$630,000)
 - 9042 Jennrich Street (\$630,000)
 - 9531 Bird Ave (\$612,000)

Web Forms display Text Display SuperObjects with the `webtagortext()` function, so variables displayed in your form can contain HTML tags (see “[Fields or Variables with Special Characters](#)” on page 375). For example the procedure could build an array then convert it to an HTML table using the `htmlarraytable` statement (see “[Generating an HTML Table from a Panorama Array](#)” on page 403). This table (for example line items in a shopping cart) can then be displayed in a Text Display SuperObject.

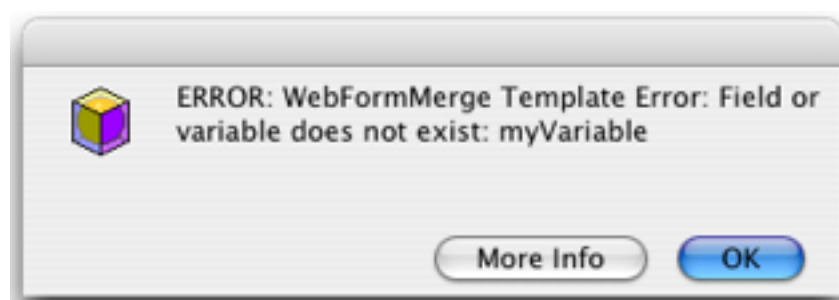
Note: If your variable does contain HTML tags, it’s up to you to make sure that any special characters in the text be converted to HTML format. This can be done with the `webtext()` or `htmlencode()` functions.

Which Came First, the Chicken (Web Template) or the Egg (Variables)?

When creating a web form that displays variables you may run into a problem — you can’t preview the form until the variables are created! For example, suppose I add a Text Display SuperObject to my form that displays a variable named `myVariable`. So far, however, I haven’t defined this variable in a procedure.



When I try to preview the form, an error message appears, and I can’t see what the form will look like in a browser.



The only solution is to define the variable in a procedure and run the procedure. You may want to create a special procedure just for initializing variables, perhaps called `.InitializeVariables`.

```
makefileglobals myVariable="",myOtherVariable=""
```

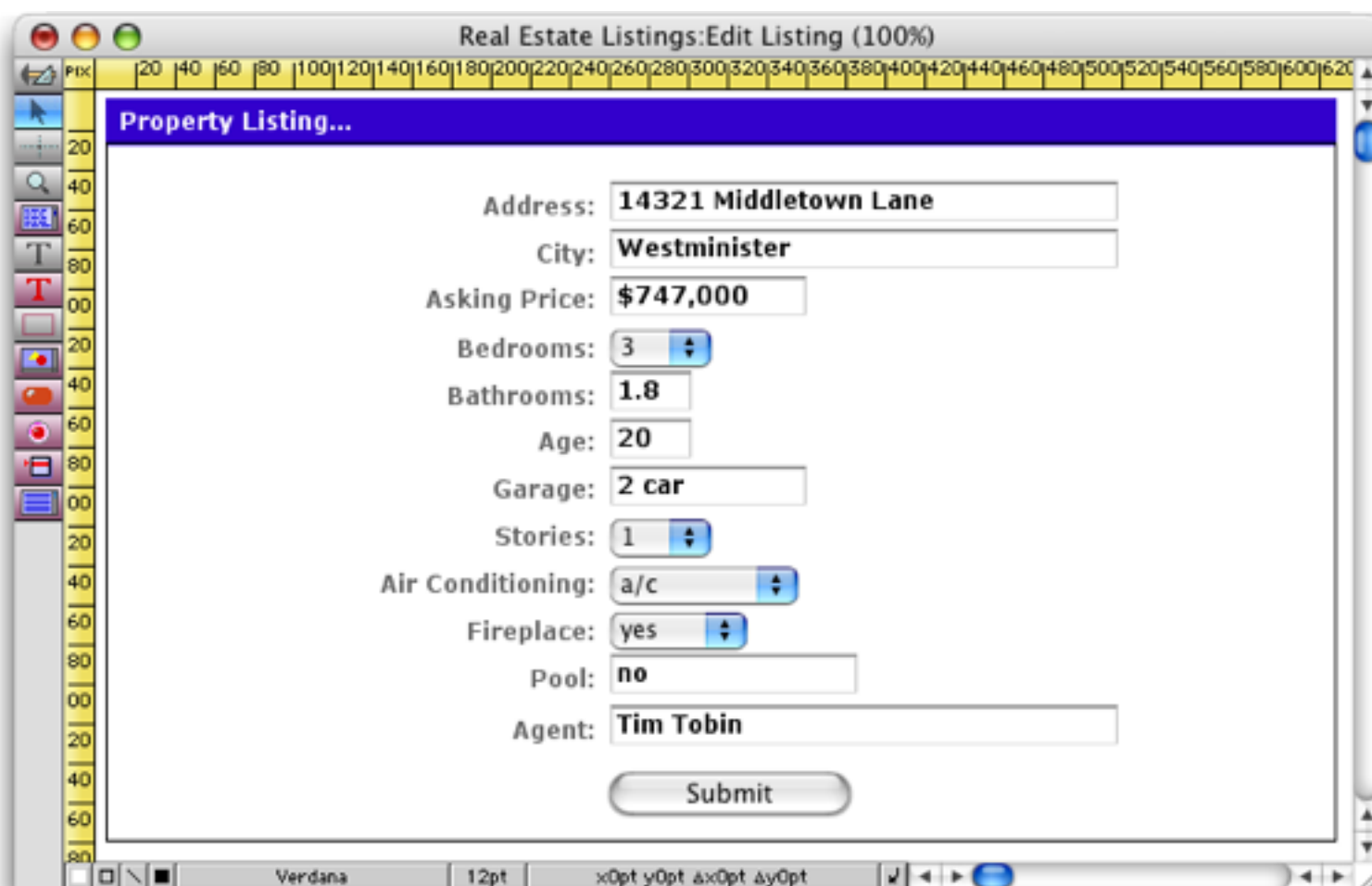
Whenever you add a new variable for display in a web form you can add it to the list and run the procedure. You will also probably want to call your `.InitializeVariables` procedure as a subroutine from the `.Initialize` and `.InitializeServer` procedures. This makes sure that the variables are automatically defined when the database is opened.

Using a Web Form to Submit Data

So far, we've only covered web forms for displaying data. Web forms can also be used to submit data to the server. There are three general applications where data is submitted from a form to the server: 1) Adding new data to a database, 2) Specifying information to be located (search), and 3) Modifying existing data in a database. For the first two of these applications the form will usually start out blank. When you are modifying the database the form will usually be pre-filled in with data from a record in the database (or possibly from variables).

Designing Web Forms for Submitting Data

A web form for submitting data is just like any other web form, except that it contains one or more objects for editing data: Text Editor SuperObjects, Data Cells, pop-up menus, checkboxes, radio buttons, or scrolling lists. The form should also contain a **Submit** button. Here's a typical example.



The screenshot shows a web browser window titled "Real Estate Listings:Edit Listing (100%)". The main content area is titled "Property Listing..." and contains the following form fields:

- Address: 14321 Middletown Lane
- City: Westminister
- Asking Price: \$747,000
- Bedrooms: 3 (dropdown menu)
- Bathrooms: 1.8
- Age: 20
- Garage: 2 car
- Stories: 1 (dropdown menu)
- Air Conditioning: a/c (dropdown menu)
- Fireplace: yes (dropdown menu)
- Pool: no
- Agent: Tim Tobin

A "Submit" button is located at the bottom of the form. The browser's status bar at the bottom shows "Verdana 12pt" and "x0pt y0pt Δx0pt Δy0pt".

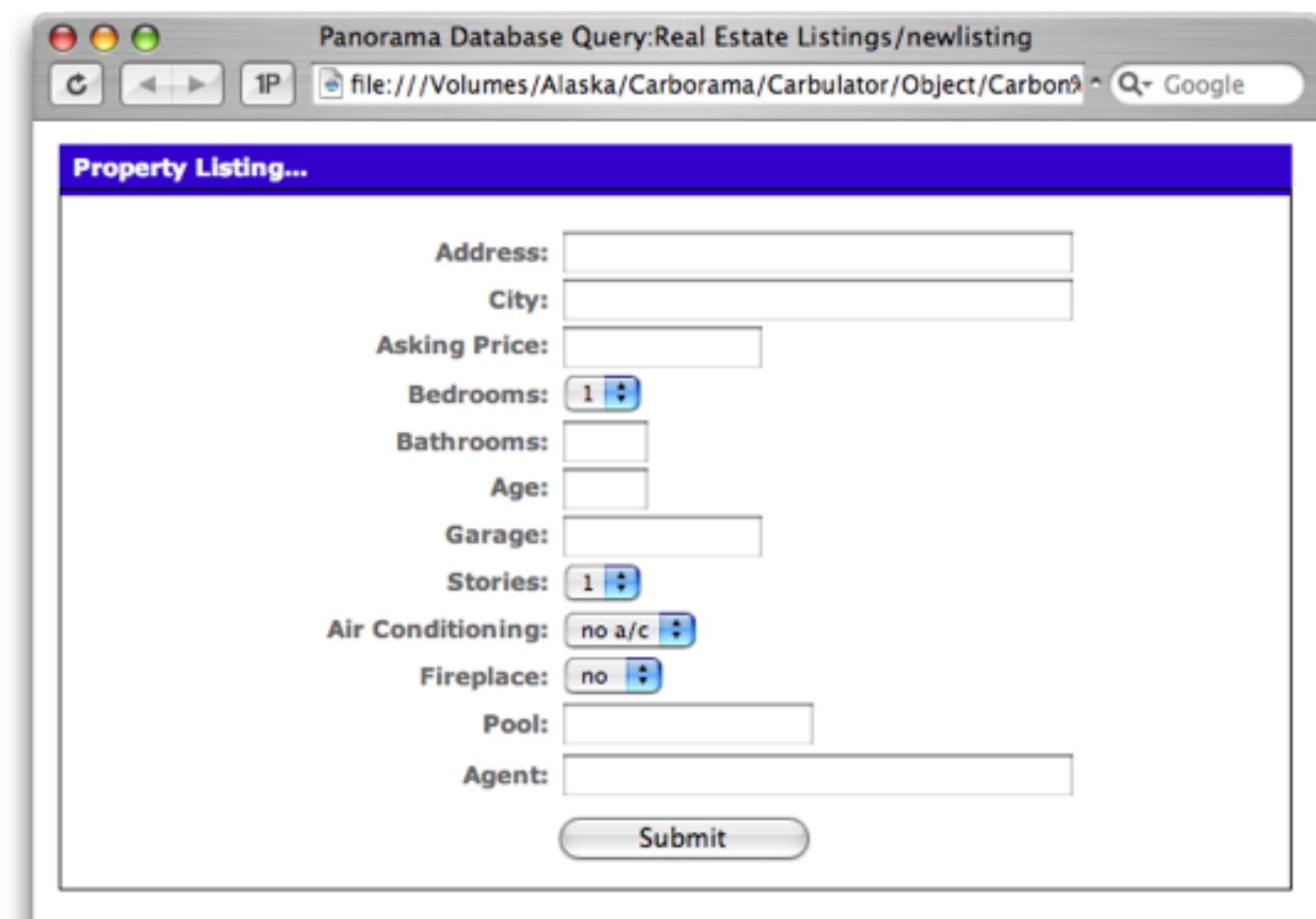
For more information see "[Web Forms](#)" on page 231,

Displaying a Blank Web Form

Displaying a blank web form is easy — just use "" as the second parameter to the `webformitem()` function.



The form will display with all of the editing elements blank (multiple choice items, like pop-up menus, will display with the first choice selected).

A screenshot of a web browser window titled "Panorama Database Query:Real Estate Listings/newlisting". The address bar shows a file path: "file:///Volumes/Alaska/Carborama/Carbulator/Object/Carbon9". The main content area displays a form titled "Property Listing...". The form contains the following fields:

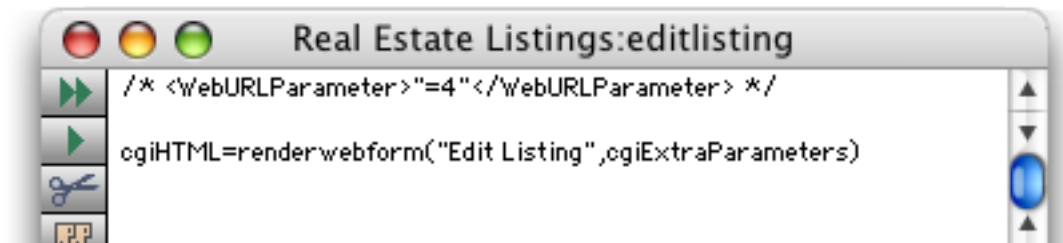
- Address:
- City:
- Asking Price:
- Bedrooms:
- Bathrooms:
- Age:
- Garage:
- Stories:
- Air Conditioning:
- Fireplace:
- Pool:
- Agent:

A "Submit" button is located at the bottom of the form.

For a blank form, that's all there is to it. The next chapter will explain how to process the data submitted from this form.

Pre-Filling Database Fields

If you want the form to display with the database fields filled in you must specify what record to use to fill in the fields. This is done by specifying a second parameter for the `renderwebform()` function. If the database is a shared database usually the best way to do this is with the unique record ID the server keeps for each record. This ID starts with 1 for the first record and increments by one for each record added to the database. This procedure allows you to specify the server record id in the url when running this procedure.



Alternately the `renderwebform()` function accepts the abbreviation "@" instead of `cgiExtraParameters`:



Either version of this procedure will work with URLs like these:

```
http://11.22.33.44/cgi-bin/panoarma.cgi?Real%20Estate%Listings~editlisting~=1
http://11.22.33.44/cgi-bin/panoarma.cgi?Real%20Estate%Listings~editlisting~=2
http://11.22.33.44/cgi-bin/panoarma.cgi?Real%20Estate%Listings~editlisting~=3
```

The first url will edit the top record in the database, the second url will edit the second record, etc. (Note that these may not (and probably will not) correspond to the current order of the records in the data sheet, rather, the numbers correspond to the order in which the records were added to the database.) Here's what the result looks like:

| | |
|-------------------|-------------------|
| Address: | 735 Elliott Place |
| City: | Santa Ana |
| Asking Price: | 989400 |
| Bedrooms: | 2 |
| Bathrooms: | 2.00 |
| Age: | 35 |
| Garage: | 1 car |
| Stories: | 1 |
| Air Conditioning: | no a/c |
| Fireplace: | yes |
| Pool: | spa |
| Agent: | Ann Rachels |

Of course you probably won't want to type in URLs like this manually, and who knows what the server record ID of a particular record is anyway? (You can find out, by the way, with the `info("serverrecordid")` function.) Instead, you'll usually want to create another procedure that lists multiple records, and includes links to edit each record. Earlier in this chapter we examined a procedure that lists selected records from this Real Estate database (see "[Using Variables to Customize a Web Form on the Fly](#)" on page 383), now we can modify this procedure to generate links to the new `editlisting` procedure. The necessary modifications are shown below.

```

/* <WebURLParameter>"600000~650000"</WebURLParameter> */
fileglobal minPrice,maxPrice,avgPrice,listingCount,theListings
local templist,n,city,lastcity,tempprices,at,ac

minPrice=val(array(cgiExtraParameters,1,"~"))
maxPrice=val(array(cgiExtraParameters,2,"~"))
select Asking>=minPrice and Asking<=maxPrice

arrayselectedbuild templist,cr()," ",fixedwidth(City,20)+tab()+Address+tab()+str(Asking)+tab()+str(info("serverrecordid"))
arraysort templist,templist,cr()
arrayfilter templist,tempprices,cr(),array(import(),3,tab())
at=float(arraynumerictotal(tempprices,cr()))
listingCount=float(arraysize(tempprices,cr()))
avgPrice=pattern(at/listingCount,"#,")

n=1 lastcity=""
theListings=""<ul>
loop
city=strip(array(nthline(templist,n),1,tab()))
stoploopif city=""
if city<>lastcity
theListings=theListings+"</ul>"<ul>"<li>"+cgilink("", "editlisting~="+array(nthline(templist,n),4,tab()),array(nthline(templist,n),2,tab()))+
(" "+pattern(val(array(nthline(templist,n),3,tab()),"#,"))+")</li>"</ul>
n=n+1
while forever
theListings=theListings+"</ul>"
theListings=replace(theListings,"<ul></ul>","")

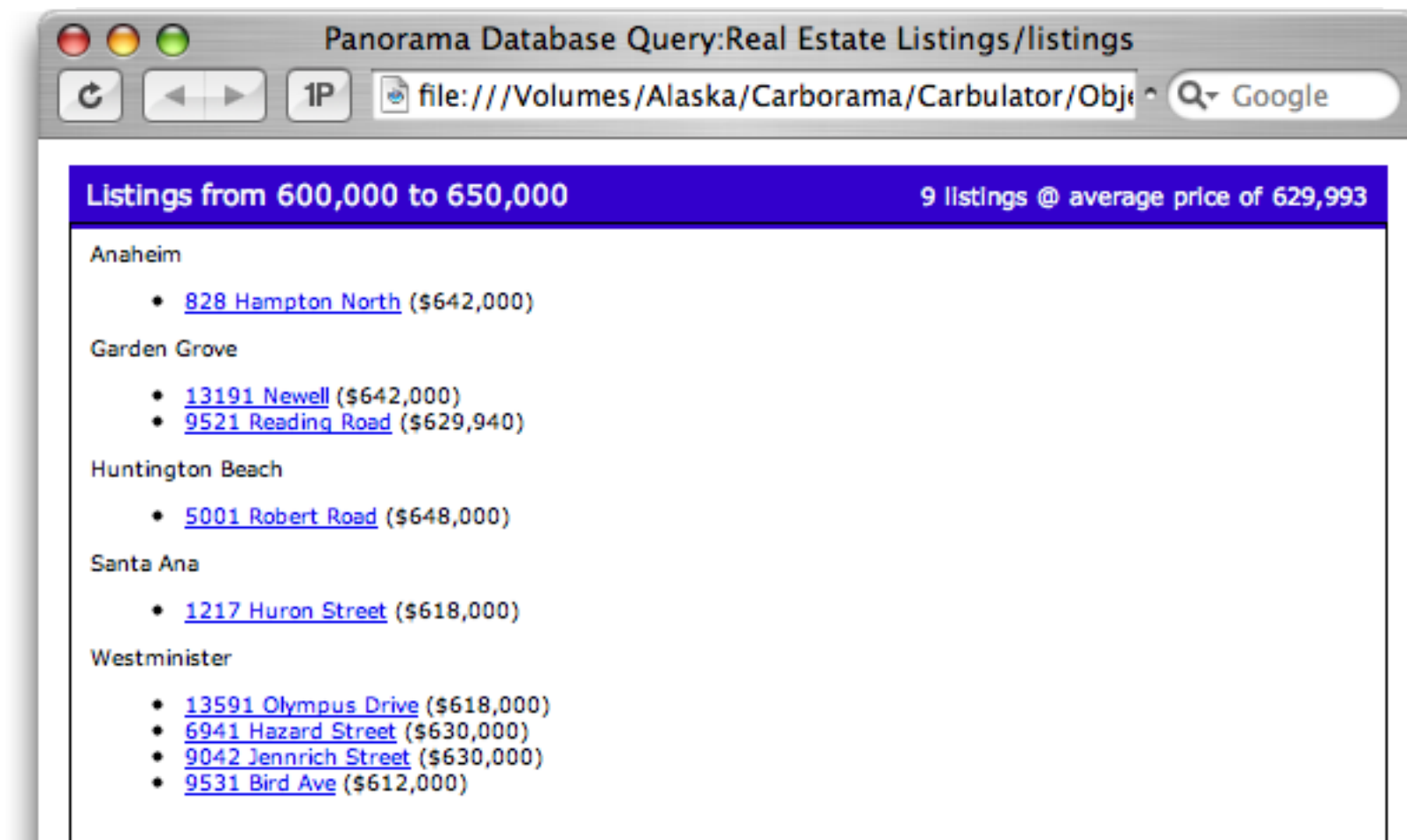
cgiHTML=renderwebform("Listing Summary","")

selectall

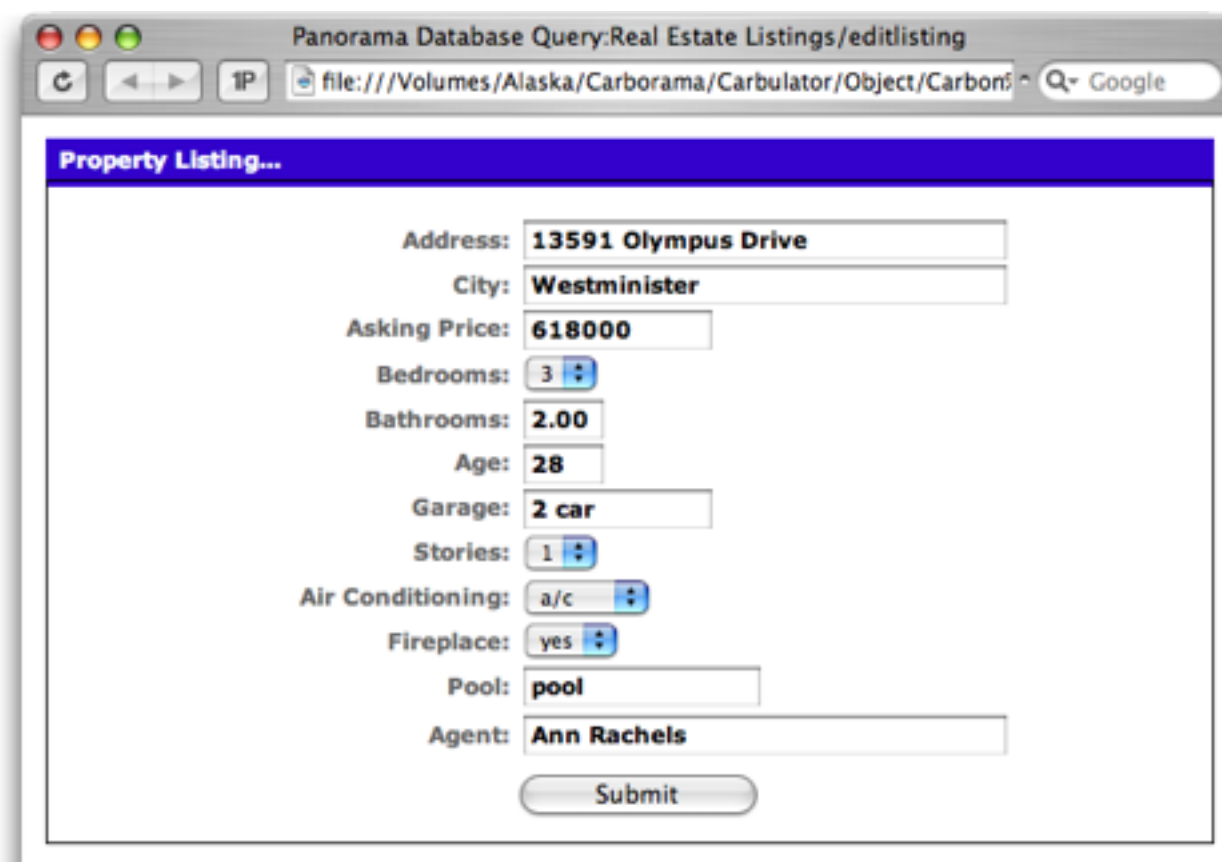
```

Procedure OK.

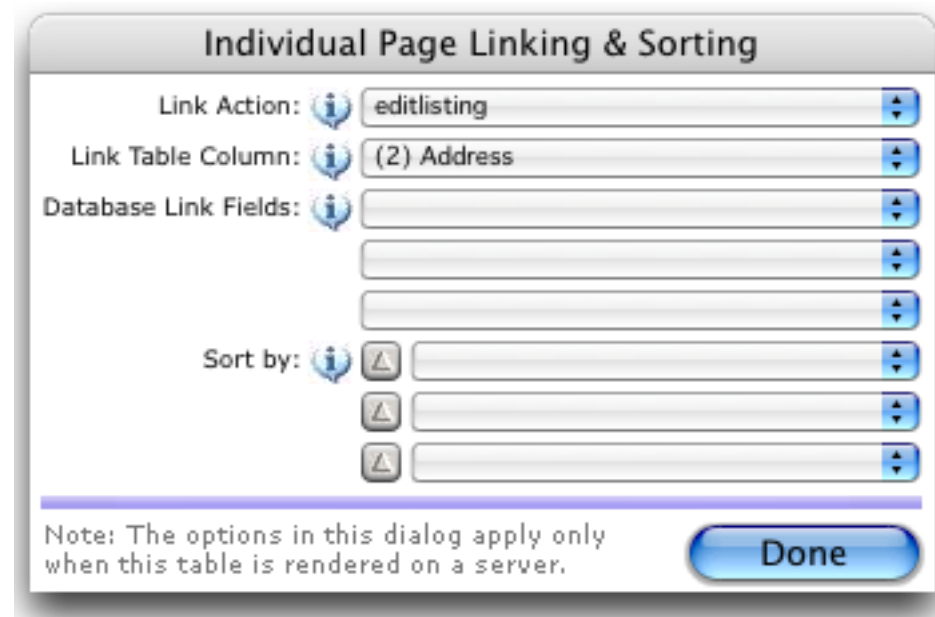
Now this web page contains links to separate editing pages for each property.



Each link contains a URL in the proper format for the `editlisting` procedure. If I click on a particular link, say [13591 Olympus Drive](#), the corresponding record appears.



In this example the code to list the records with links was hand coded, but more commonly you'll use a Panorama web table template to do the heavy lifting for you. A web table allows you to create a nicely formatted list, with links, without doing any programming at all. When designing the table template (see "[Web Tables](#)" on page 277) be sure to link your editing procedure to the table and specify the link table column (see "[Linking Individual Table Rows to a Detail Form](#)" on page 307).



With this one dialog your table will automatically generate the correct format links for the `editlisting` procedure. No further programming is required! Here's an example of a table with links.

| City | Address | Asking | Bedrooms | Bathrooms | Stories |
|------------------|---------------------------------------|---------------|----------|-----------|---------|
| Newport Beach | 6711 Seashore Drive | \$ 2,160,000 | 2 | 1.00 | 1 |
| Huntington Beach | 4031 Diablo Drive | \$ 3,270,000 | 2 | 2.00 | 1 |
| Newport Beach | 323 23rd Street E. | \$ 1,950,000 | 3 | 1.75 | 1 |
| Huntington Beach | 22331 Wallingford Ln | \$ 1,980,000 | 3 | 2.50 | 2 |
| Huntington Beach | 16871 Stiles Circle | \$ 2,103,000 | 3 | 3.50 | 2 |
| Huntington Beach | 16212 Typhoon Way | \$ 2,094,000 | 3 | 2.00 | 1 |
| Newport Beach | 113 G Street | \$ 2,274,000 | 3 | 2.75 | 2 |
| Huntington Beach | 16632 Intrepid Lane | \$ 2,039,400 | 3 | 2.50 | 2 |
| Costa Mesa | 1843 Gisler Ave | \$ 2,310,000 | 3 | 3.00 | 1 |
| Sunset Beach | 16369 Pacific South | \$ 2,550,000 | 3 | 1.75 | 2 |
| Laguna Beach | 727 Emerald Bay | \$ 2,700,000 | 3 | 2.00 | 1 |
| Surfside | 73 Surfside A | \$ 2,850,000 | 3 | 2.00 | 2 |
| Huntington Beach | 3592 Runningtide | \$ 2,394,000 | 3 | 2.50 | 2 |
| Garden Grove | 9261 Royal Palm Drive | \$ 2,370,000 | 3 | 3.00 | 1 |
| Huntington Beach | 16202 Typhoon Lane | \$ 2,399,400 | 3 | 3.00 | 2 |
| Huntington Beach | 16571 Esign Way | \$ 2,910,000 | 3 | 2.00 | 1 |
| Surfside | Surfside A97 | \$ 2,970,000 | 3 | 3.50 | 2 |
| Santa Ana | 12562 Kingsview Rd | \$ 3,150,000 | 3 | 2.50 | 3 |
| Huntington Beach | 16315 Niantic Circle | \$ 8,370,000 | 3 | 3.00 | 2 |
| Huntington Beach | 3332 Venture Drive | \$ 10,500,000 | 3 | 4.50 | 2 |

1 2 3 [NEXT](#)

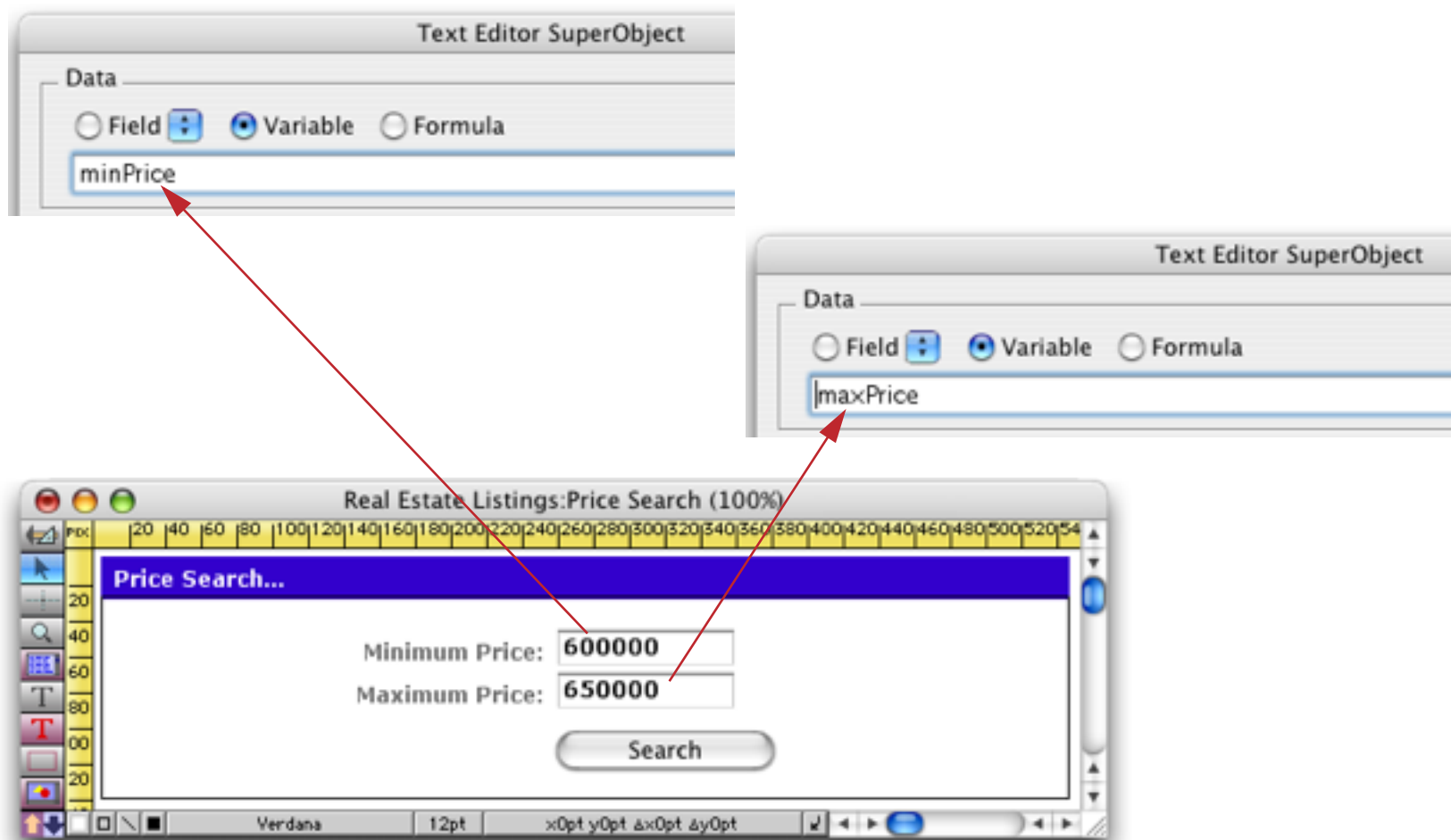
Clicking on any of these links opens the editing form for that record.

During this discussion we've mentioned several times that the techniques used work only with shared databases. This is because the links are made using the unique record id's the server maintains for each record in a shared database. Because of this, we highly recommend using shared databases even if you don't need to share the database. (Keep in mind that you can create and use shared databases even if you haven't licensed the server for sharing, only web publishing. In that case you can only access the database one user at a time, but the database will still have unique record id's for each record.)

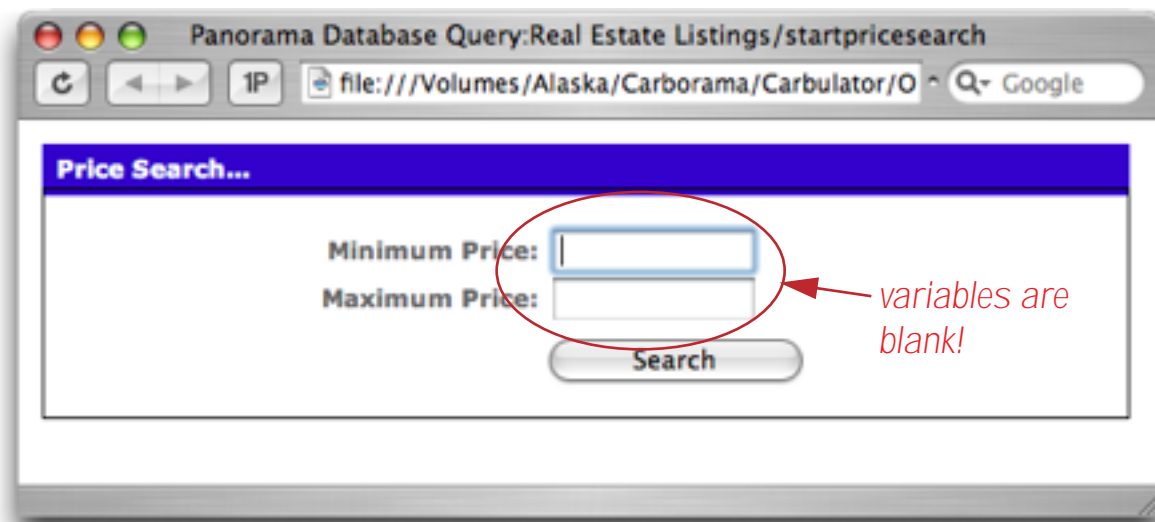
If for some reason you really don't want to use shared databases it is still possible to pre-fill database fields in a form. See "[Using WebURLFind for Navigation in Non-Shared Databases](#)" on page 401.

Pre-Filling Variables

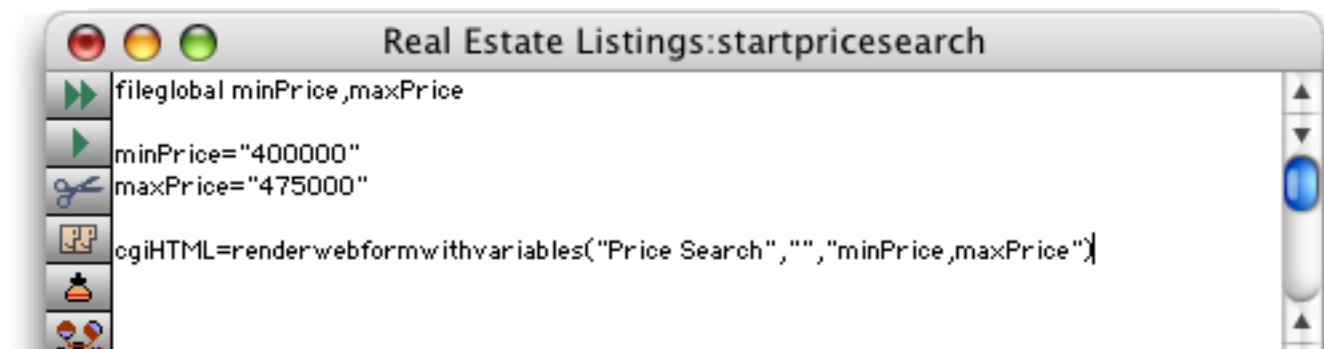
A web form can contain objects for editing variables as well as database fields. For example, this form edits two fileglobal variables, `minPrice` and `maxPrice`.



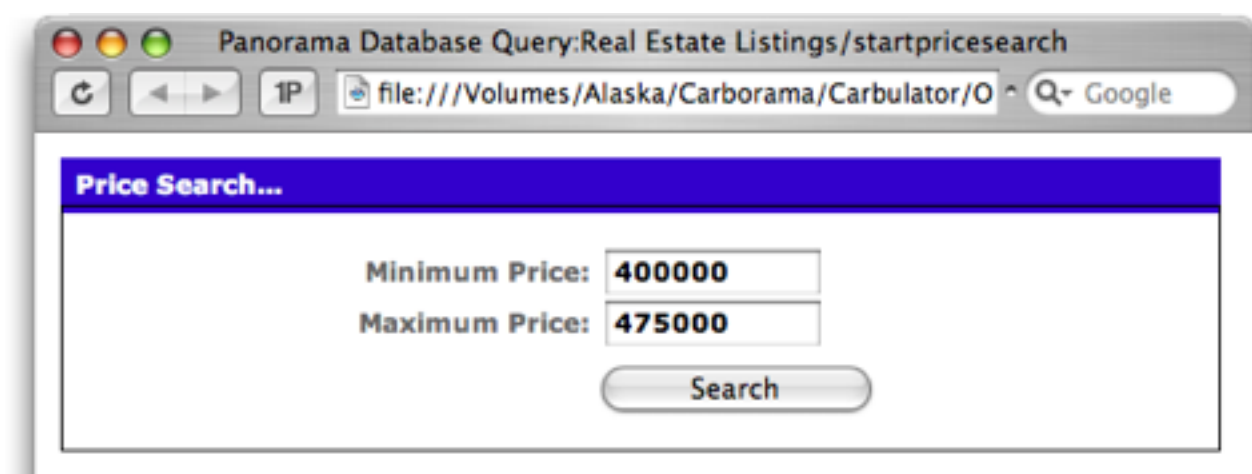
When this form is turned into a web page with `webrenderform()`, however, the form is always blank, even if the variables have values in them.



If you want to pre-fill editable items that are linked to variables you need to use the `renderwebformwithvariables()` function. This function has one additional parameter, a comma separated list of variables. (Note: This list is based on a formula, which usually means that it must be quoted, as shown below).



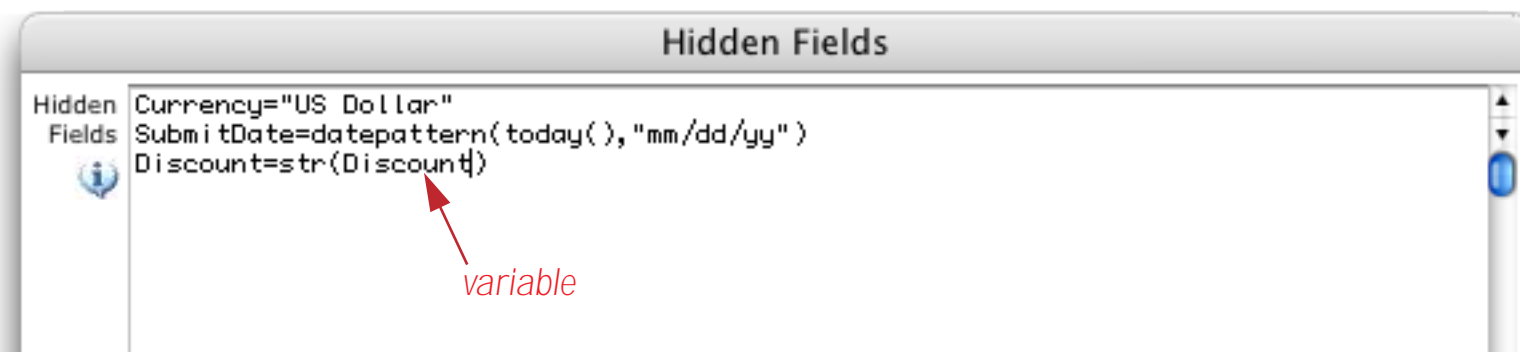
This example sets up two variables, fills them with values (which **must** be text), then generates an HTML page with the values for these variables already filled in.



Note: If your form uses **Buffered Data Entry** (see “[Buffered Data Entry](#)” on page 438) with a prefix for each web item then you don’t need to the `renderwebformwithvariables()` function, you can simply use the `renderwebform()` function. Panorama server will sense that your web form is set up for buffered data entry and fill in the web items with data from the database.

Setting Hidden Form Values

In addition to normal visible data values (text editing cells, checkboxes, radio buttons, etc.) web browsers also support hidden values within a web form. There are two ways that a procedure can set the value of a hidden field. The first method is to set up the hidden field in advance as part of the form setup (see “[Hidden Data](#)” on page 271). When the hidden field definitions are set up simply define one or more hidden fields using a variable, like this.



Then simply make sure you define this variable and assign it a value before rendering the web form.

```
fileglobal Discount
Discount = 7.75
cgiHTML = renderwebform("Invoice", "")
```

If the hidden field hasn't been set up in advance a web procedure can still add it at the last minute when the page is generated. This is done with the `sethiddenwebformitem` statement, which has three parameters.

```
sethiddenwebformitem page,field,value
```

The `page` parameter is the name of the variable that contains the HTML page (often this will be `cgiHTML`). The text in this variable will be modified to add the hidden field information.

The `field` parameter is the name of the hidden field, while the value parameter is the `value` of that field.

Here is a revised version of the previous example that adds a hidden value "after the fact", even though the original form template didn't set up this hidden field.

```
fileglobal Discount
Discount = 7.75
cgiHTML = renderwebform("Invoice", "")
sethiddenwebformitem cgiHTML,"Discount","7.75"
```

Note: You can add as many hidden fields as you like, but you should only add a particular hidden field once. In other words, don't add `Discount` twice. You should also avoid using the `sethiddenwebformitem` statement for a hidden field that has already been set up using the **Hidden Fields** dialog for that form.

What Record Are We Talking About?

So far this chapter has glossed over an important point — when accessing (or modifying) information in database fields, what record is the current record? When using Panorama directly the answer is obvious — whatever record was last clicked on. You can simply look at the data sheet to see (or change) the active record. If you are using a web browser, however, you can't simply use the data sheet — it's not available. Some other method will have to be used to specify what record is going to be displayed or modified.

“Roll Your Own” Web Navigation

As a Panorama programmer you've probably already thought of the obvious idea, simply use an extra URL parameter (see “[URL Extra Parameters](#)” on page 332) to specify a the record to display or modify. For example, consider this price list database.

| Item | Description | Price | Railroad | Category | SubCategory | Manufacturer |
|-----------------------------|--------------------------------|---------|----------|---------------|---------------|--------------|
| Amtrak F7A | One of America's greatest die | \$24.99 | Amtrak | Diesel Engine | Passenger | Atheam |
| Amtrak Genesis | When today's American passe | \$47.99 | Amtrak | Diesel Engine | Passenger | Atheam |
| Amtrak S/V7 | Popular through the 1960s, ti | \$27.49 | Amtrak | Diesel Engine | Switch Engine | Atheam |
| ATSF 40' Single Dome Tank | Single-dome tank cars are ma | \$4.49 | Santa Fe | Freight Car | Tank Car | Atheam |
| ATSF 4427 Covered Hopper | The 4427 cubic foot covered | \$20.79 | Santa Fe | Freight Car | Hopper Car | Life-Like |
| ATSF 50' Box (Chief) | Running on the Atchison Topel | \$4.49 | Santa Fe | Freight Car | Box Car | Atheam |
| ATSF 50' Box (Grand Canyon) | Running on the Atchison Topel | \$4.49 | Santa Fe | Freight Car | Box Car | Atheam |
| ATSF 50' Gondola | Gondola cars haul a variety o | \$5.49 | Santa Fe | Freight Car | Gondola Car | Atheam |
| ATSF Baggage Car | This baggage car is one of sev | \$8.49 | Santa Fe | Passenger Car | Baggage Car | Atheam |
| ATSF Coach | This streamline coach is the | \$8.49 | Santa Fe | Passenger Car | Coach | Atheam |
| ATSF Diner | Dining was never more elegar | \$8.49 | Santa Fe | Passenger Car | Diner | Atheam |
| ATSF Observation | Passengers pay extra to ride | \$8.49 | Santa Fe | Passenger Car | Observation | Atheam |
| ATSF Tank Car | This single-dome tank car bel | \$4.49 | Santa Fe | Freight Car | Tank Car | Atheam |
| ATSF Vista Dome | The Vista Dome streamliner c | \$18.49 | Santa Fe | Passenger Car | Vista Dome | Atheam |

The **Item** field in this database contains a unique name for each product, so that can be used to identify each record. Here is a procedure that finds and displays an item based on the name at the end of the URL. (In this case the spaces and other special characters in the Item are not needed, so the procedure strips them out to make the url simpler.)

```

/* <WebURLParameter>'atsftankcar'</WebURLParameter> */
find stripchar(Item, 'A-Za-z09') match cgiExtraParameters
cgiHTML=renderwebform('Display-Catalog-Item', '')

```

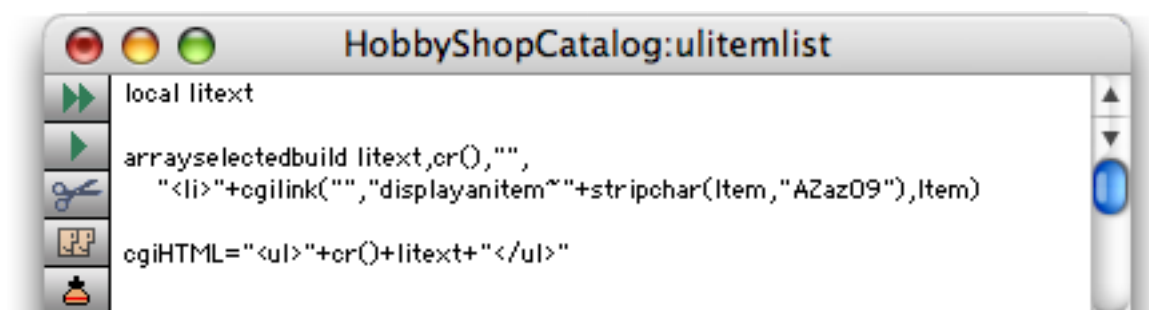
To use this procedure, set up your URL like this, with the item name (minus spaces and special characters) at the end.

```
http://1.2.3.4/cgi-bin/panoarma.cgi?hobbyshopcatalog~displayanitem~atsftankcar
```

Using this URL will display the specified record in your web browser.



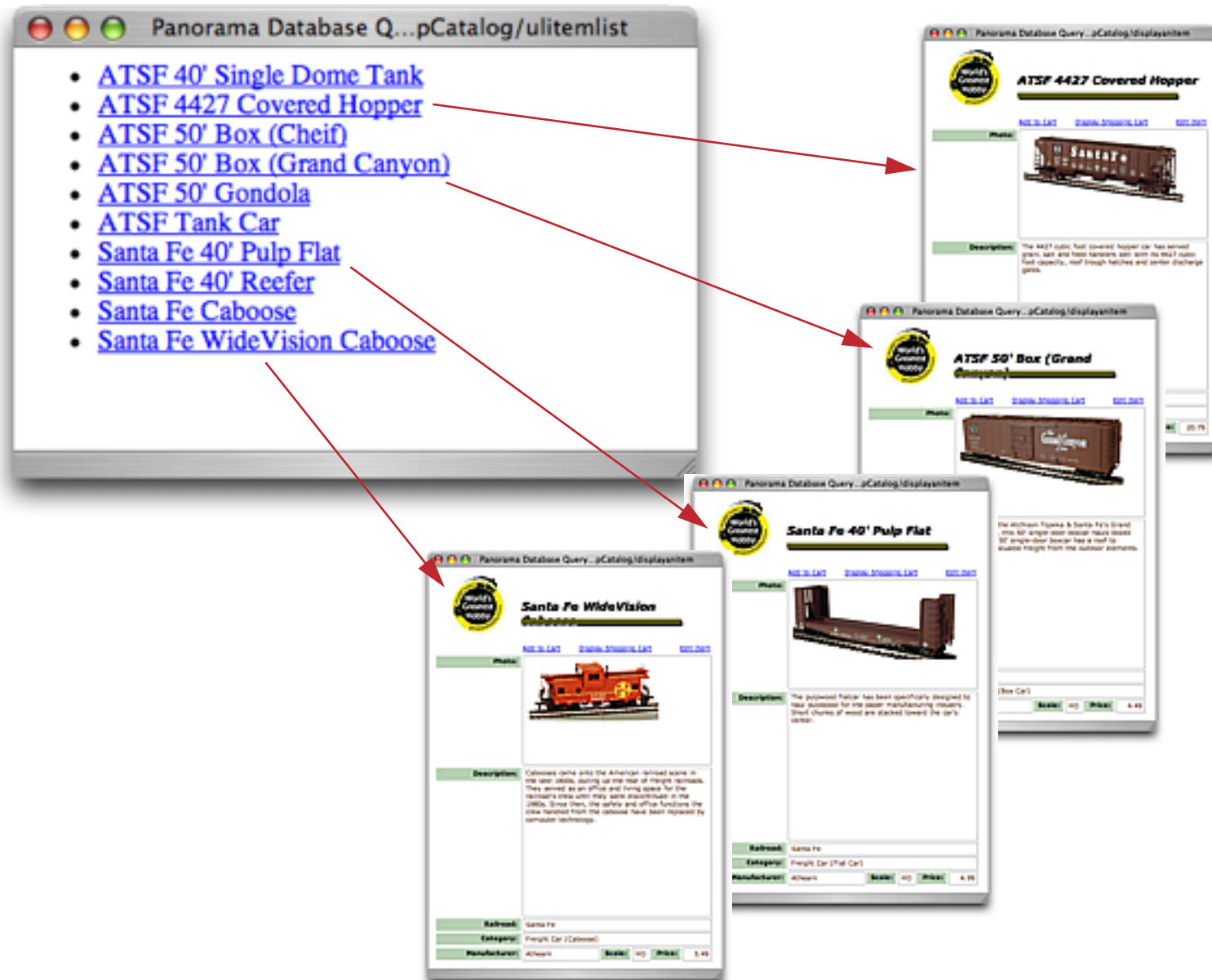
Of course you usually won't want to type in a URL like this — instead, you'll create another procedure that generates a list of links. Then the user can simply click on the item he or she wants. Here is a very simple procedure that does just that.



The key code in this procedure is the second parameter of the `cgilink()` function. This constructs the link in the same format used by our custom `displayanitem` procedure.

```
cgilink("", "displayanitem~"+stripchar(Item, "AZaz09"), Item)
```

When you run this procedure it produces a web page that looks like this:



Clicking on any of the links triggers the `displayanitem` procedure which brings up the corresponding detail page.

As you can see, the “roll-your-own” approach can work, but it has some disadvantages. First of all, this technique has to be rewritten from scratch for each web database you create. If your database doesn’t have a single field with unique values you may have to combine two or more fields into a single “key”. This is possible, but it can make the URLs very unwieldy. If the data contains unusual characters that can also make the URLs clumsy to work with.

Fortunately, if you are using shared databases Panorama has a built in solution that solves all of these problems. Read on to learn more.

Searching the Database in Reverse

Though we don’t recommend roll your own web navigation, here is a variation you may want to use if you do decide to go that route. Instead of using the `find` statement, you may want to use `findbackwards`. In many situations this is faster, sometimes dramatically so, for two reasons. First, `findbackwards` does not have any undo capability, so it saves the time needed to maintain the previous database state. Secondly, new records are always added at the end of the database, and in many applications recently added records are searched for much more frequently than older records. In large databases the `findbackwards` statement will find records near the end of the database much faster than `find` will. Your mileage may vary, but this is definitely a variation worth trying.

Using WebURLFind For Navigation in Shared Databases

When working with a shared database, Panorama Enterprise keeps a unique numeric record id for every record in the database. You can't see the record id, but it is there and for a particular record it is always guaranteed to be unique and unchanging (unless you upload a New Sharing Generation, then all record ID's may and probably will change). If a record is deleted the unique record ID for that will not be re-used (again, unless you upload a New Sharing Generation).

Though you can't see the unique record ID, it can be accessed in a formula with the `info("serverrecordid")` function. Using this function you could "roll your own" web navigation much like the system described in the previous section. However, you don't need to "roll your own" because Panorama Enterprise's form and table rendering function already have this navigation feature built in. As a bonus, this navigation method is faster than "roll your own" because it uses a special stripped down search that is optimized for searching for server record IDs.

URLs that use this built in navigation feature look like this:

```
http://domain/cgi-bin/panorama.cgi?database~procedure~=recordid
```

In other words, the extra parameter is an = followed by the record ID. The record ID is a number from 1 to 2147483648, so typical URLs look like these:

```
http://1.2.3.4/cgi-bin/panoarma.cgi?invoices~display~=3891
http://1.2.3.4/cgi-bin/panoarma.cgi?customers~edit~=82397
http://1.2.3.4/cgi-bin/panoarma.cgi?properties~show~=359
```

If you are generating your HTML by hand you can navigate to the specified record with a single line:

```
weburlfind cgiExtraParameters
```

In fact, you can go even a bit simpler than that...

```
weburlfind "@"
```

Just put this line at the top of your web procedure and this statement will find the specified record for you. You are then ready to generate the HTML for your web page from that record.

If you are using a Panorama web form you don't need a separate `weburlfind` statement — the navigation feature is built into the `renderwebform()` function. This single line of code finds the record specified by the URL and displays it using the `Edit Listing` form.

```
cgiHTML=renderwebform("Edit Listing","@")
```

(Note: You can also use `cgiExtraParameters` instead of "@".)

Panorama Enterprise also has tools to help you generate navigation links based on the server record id. The `cginavlink()` function will automatically build a link to the current record for you.

```
cginavlink(procedure,caption)
```

This function has two parameters: the name of the procedure that will navigate and display or modify the record, and a caption that will appear on the web page.

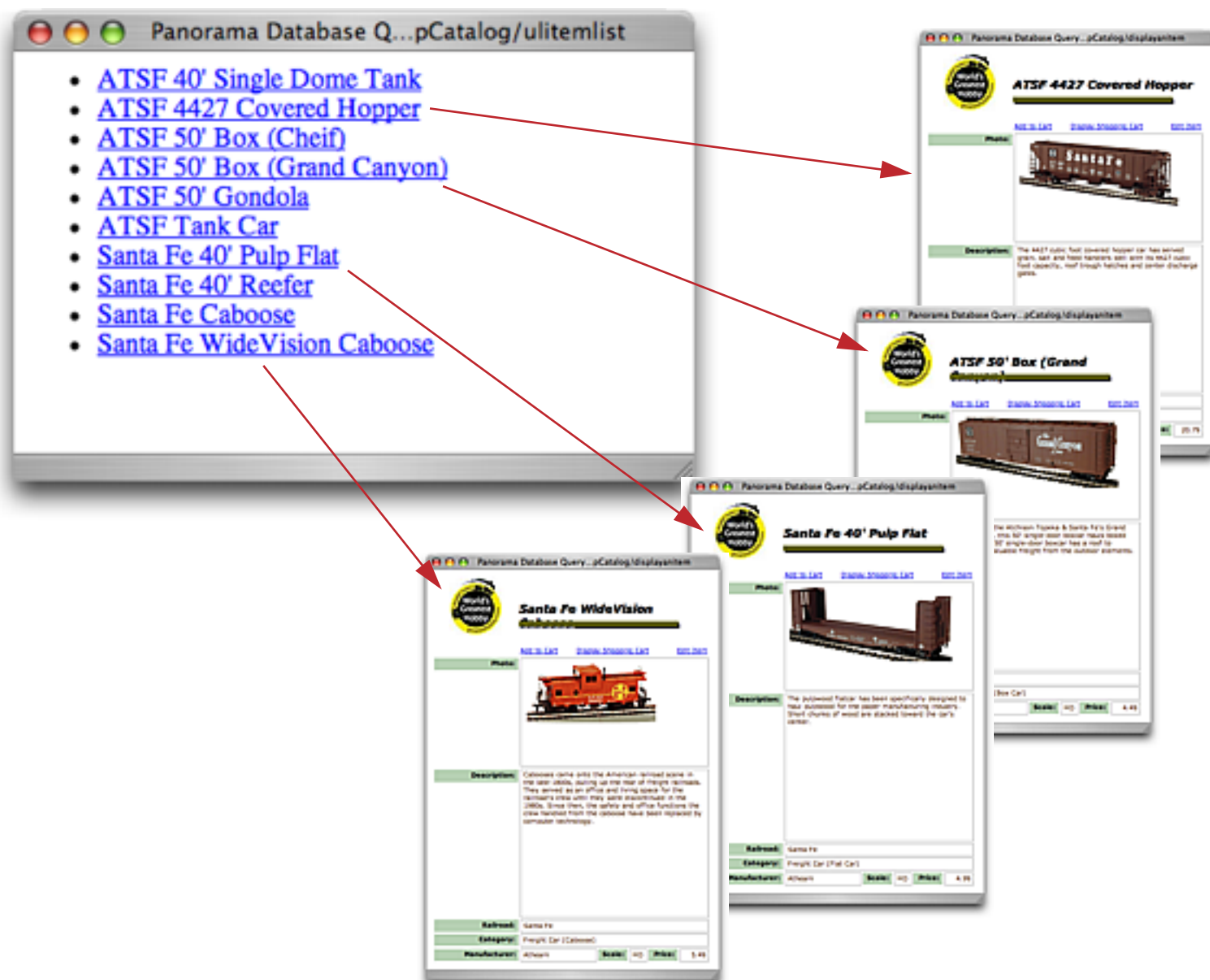
Here is a simple procedure that builds a list of links by using this function inside the `arrayselectedbuild` statement.

```

HobbyShopCatalog:productlist
local litext
arrayselectedbuild litext,cr(),"",
  "<li>"+cginavlink("showproduct",Item)
cgiHTML="<ul>"+cr()+litext+"</ul>"
Procedure OK.

```

When you run this procedure it produces a web page of links:



When you click on a link on this page it will trigger this very simple `showproduct` procedure, which finds and displays the corresponding product record.

```

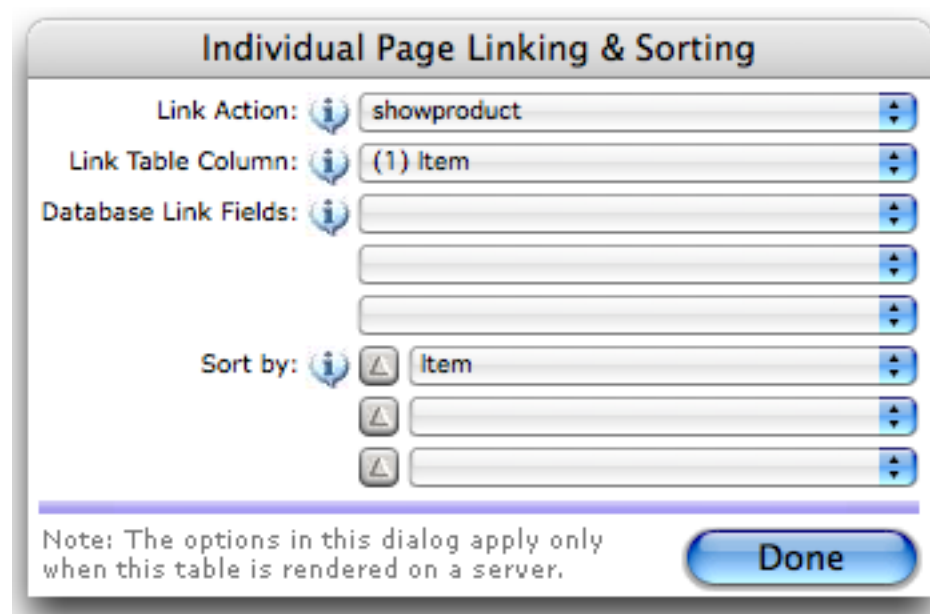
HobbyShopCatalog:showproduct
/* <WebURLParameter>"=2"</WebURLParameter> */
cgiHTML=renderwebform("Display Catalog Item", "@")

```

The `productlist` procedure in this example that generates the list of links is pretty simple (see above), but you can usually eliminate even that code by creating a web table template, which allows you to create a nicely formatted list, with links, without doing any programming at all (see “[Web Tables](#)” on page 277). Here is a table template for this price list:



When designing the table template be sure to link your procedure (in this case `showproduct`) to the table and specify the link table column (see “[Linking Individual Table Rows to a Detail Form](#)” on page 307).



The final table will look something like this. The links to the `showproduct` procedure are automatically generated.

| Item | Railroad | Category | SubCategory | Scale | Price |
|---|----------|---------------|---------------|-------|----------|
| Amtrak F7A | Amtrak | Diesel Engine | Passenger | HO | \$ 24.99 |
| Amtrak Genesis | Amtrak | Diesel Engine | Passenger | HO | \$ 47.99 |
| Amtrak SW7 | Amtrak | Diesel Engine | Switch Engine | HO | \$ 27.49 |
| ATSF 40' Single Dome Tank | Santa Fe | Freight Car | Tank Car | HO | \$ 4.49 |
| ATSF 4427 Covered Hopper | Santa Fe | Freight Car | Hopper Car | HO | \$ 20.79 |
| ATSF 50' Box (Chelf) | Santa Fe | Freight Car | Box Car | HO | \$ 4.49 |
| ATSF 50' Box (Grand Canyon) | Santa Fe | Freight Car | Box Car | HO | \$ 4.49 |
| ATSF 50' Gondola | Santa Fe | Freight Car | Gondola Car | HO | \$ 5.49 |
| ATSF Baggage Car | Santa Fe | Passenger Car | Baggage Car | HO | \$ 8.49 |
| ATSF Coach | Santa Fe | Passenger Car | Coach | HO | \$ 8.49 |

Using WebURLFind for Navigation in Non-Shared Databases

Even if you are not using a shared database you can still use **WebURLFind** for web navigation. However it's quite a bit more complicated and you lose many of the advantages of the method described in the previous section.

In non-shared databases URLs that use this built in navigation feature have one or more field=value pairs at the end of the URL:

```
http://domain/cgi-bin/panorama.cgi?database~procedure~field=value~field=value
```

In other words, the extra parameter is a series of field=value pairs separated by ~ symbols. Typical URLs look like these:

```
http://1.2.3.4/cgi-bin/panoarma.cgi?invoices~display~Order=78341
http://1.2.3.4/cgi-bin/panoarma.cgi?customers~edit~Last=Smith~First=John~Zip=92848
http://1.2.3.4/cgi-bin/panoarma.cgi?properties~show~Address=731%20Peach%20Ave~Zip=49881
```

If you are generating your HTML by hand you can navigate to the specified record with a single line (this is the same as a shared database):

```
weburlfind cgiExtraParameters
```

In fact, you can go even a bit simpler than that...

```
weburlfind "@"
```

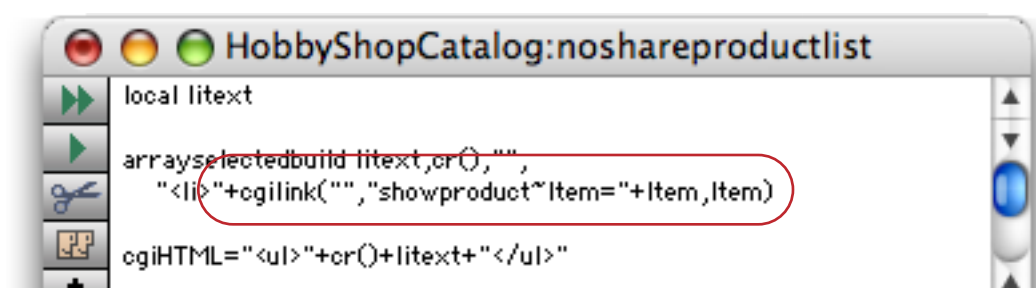
Just put this line at the top of your web procedure and this statement will find the specified record for you. You are then ready to generate the HTML for your web page from that record.

If you are using a Panorama web form you don't need a separate **weburlfind** statement — the navigation feature is built into the **renderwebform()** function. This single line of code finds the record specified by the URL and displays it using the **Edit Listing** form.

```
cgiHTML=renderwebform("Edit Listing","@")
```

(Note: You can also use **cgiExtraParameters** instead of "@".)

When using a non-shared database the **cginavlink()** function for building links won't work. Instead you'll have to build the links manually with the **cgilink()** function. Here is an example that builds a link with one Field=Value pair. If your application requires additional Field=Value pairs to generate a unique link you'll need to code these additional pairs into the formula.



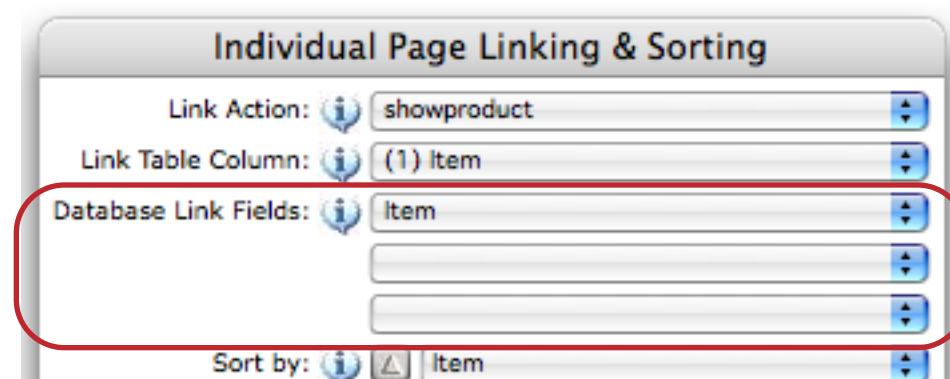
When you run this procedure it produces a web page of links, just like the similar example in the previous section. However instead of using server record id numbers the link contain field/value pairs from the database.



When you click on a link on this page it will trigger the same very simple `showproduct` procedure demonstrated in the previous section.



This linking method can also be used with table templates (see “[Web Tables](#)” on page 277). When designing the table template you must set up one or more **Database Link Fields** in addition to the **Link Action** and **Link Table Column** options (see “[Linking Individual Table Rows to a Detail Form](#)” on page 307).



The links to the `showproduct` procedure are automatically generated.

Generating an HTML Table from a Panorama Array

Using arrays Panorama can store multiple values in a single field or variable. The `htmlarraytable` statement makes it relatively easy to display the contents of an array in a web page. To illustrate this statement this section will use an invoice database uses arrays to store the line items for each order (see below). The `Items` field contains a two dimensional array, with carriage returns as the primary separator character (between rows) and the `~` symbol as the secondary separator (between columns).

| HobbyShopSales | | | | | | |
|----------------|------------------|------|--------------------------|----------------|---|--------------|
| Date | Name | Org. | Address | City | Items | InvoiceStatu |
| 08/30/98 | Lawrence Boitel | | 222 S. Airport Cir | Bozeman | 1~Santa Fe GP60~34.99~34.99 | |
| 08/30/98 | Frank Campbell | | 5 W. Spruce Lane | New York | 1~Bell ACF27 Tank Car~14.39~14. | |
| 08/30/98 | Jane Dockery | | 690 N. Capital Pl | Shrewbury | 1~Union Oil 3 Dome Tank Car~4.49~ | |
| 08/30/98 | Phillip Ballard | | 20 S. Lakeshore Drive | Rhinebeck | 1~Canadian Pacific SW7~27.49~27 | |
| 08/30/98 | Christine Hyde | | 410 S.E. Augusta Trail | Hiawatha | 1~ATSF Vista Dome~8.49~8.49 | |
| 08/30/98 | Gerald Grimes | | 41600 South Mulberry Ave | Raleigh | 1~SP 30' Gondola~13.69~13.69 | |
| 08/31/98 | William Bischoff | | 7823 W Adams Avenue | Sacramento | 1~Pennsy Observation Car~27.29~ | |
| 08/31/98 | Gloria Gamble | | 443 W. Antonio Ter | Jeffersonville | 1~SP 50' SD Box Car~11.19~11.19 | |
| | | | | | 1~WP 2-8-0~94.99~94.99 | |
| | | | | | 1~Amtrak Genesis~47.99~47.99 | |
| | | | | | 1~SP 30' Gondola~13.69~13.69 | |
| | | | | | 1~ATSF 50' Box (Grand Canyon)~4.49~4.49 | |
| | | | | | 1~C & NW SDP-40~29.99~29.99 | |

Normally a Matrix SuperObject is used to display the items, like this:

| | | | |
|--|-----------------------------|----------|--------|
| 1 | SP 50' SD Box Car | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | Amtrak Genesis | 47.99 | 47.99 |
| 1 | SP 30' Gondola | 13.69 | 13.69 |
| 1 | ATSF 50' Box (Grand Canyon) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |
| Invoice # 2256, August 31st, 1998 | | Subtotal | 202.34 |
| <input type="button" value="New Invoice"/> | | Tax | |
| <input type="button" value="Clear Items"/> | | Total | 202.34 |

The Matrix SuperObject can't be used in a web based form, so instead the `htmldatatable` statement must be used. This statement has five parameters.

```
htmldatatable array,rowsepchar,colsepchar,result,options
```

The `array` parameter is the field or variable that contains the array you want to display (in this case `Items`).

The `rowsepchar` is the character used to separate each row in the array (in this case `\n`, carriage return).

The `colsepchar` is the character used to separate each column in the array (in this case `~`).

The `result` parameter is the name of a field or variable that will be filled with the HTML for displaying the table. If the line items are the only thing to be displayed you can use the `cgiHTML` variable, otherwise you'll probably want to set up a fileglobal variable to contain the HTML text. If you are using a form you can display this text in a Text Display SuperObject (see "[Displaying an Array in a Web Form](#)" on page 411) to include the line items as part of a larger form.

The **options** parameter is a list of options that specify what columns should be included in the table and how the table should be formatted. If there are no options supplied the statement will automatically include all columns in the array in a default format. We'll go into much greater detail about all the available options in a moment (see "[Array Rendering Options](#)" on page 404).

Here is a simple procedure that renders the **Items** array into HTML.

```
htmlarraytable Items,¶,"~",cgiHTML,""
```

In the browser the rendered page will look like this:

| | | | |
|---|-----------------------------|-------|-------|
| 1 | SP 50' SD Box Car | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | Amtrak Genesis | 47.99 | 47.99 |
| 1 | SP 30' Gondola | 13.69 | 13.69 |
| 1 | ATSF 50' Box (Grand Canyon) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |

This table isn't very attractive but you can easily customize the appearance (see below).

Array Rendering Options

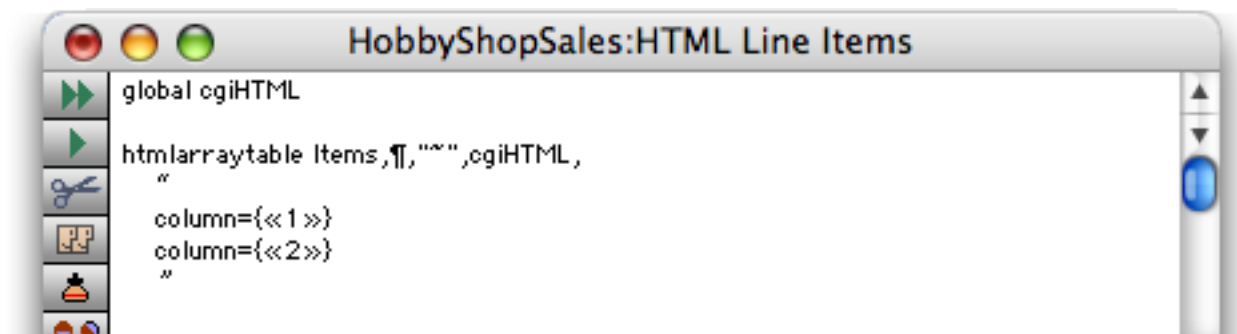
To customize the appearance of the table you'll need to specify some options in the second parameter of the **htmldatatable** statement. The options are specified as assignments, similar to the options in many HTML tags. These options are discussed in the following sections.

Table Column Layout

There are four options that allow you to specify what array columns will be included in the table and how they are titled and arranged.

| Option | Description |
|-------------------------|--|
| column=formula | If you specify any columns at all you must include one column= option for each column in the table. The formula specifies what data will be displayed in the column. To include a cell from the array being rendered use «1», «2», «3», where the number between the « and » represents the column number within the array. |
| title=text | If you specify any titles at all you must include one title= option for each column in the table. |
| width=number | If you specify any widths at all you must include one width= option for each column in the table. The width is specified in pixels, for example width=72 for a column that is 1 inch wide. |
| align=left/center/right | If you specify any alignment at all you must include one align= option for each column in the table. The alignment for each field may be left, center or right (for example align=right). |

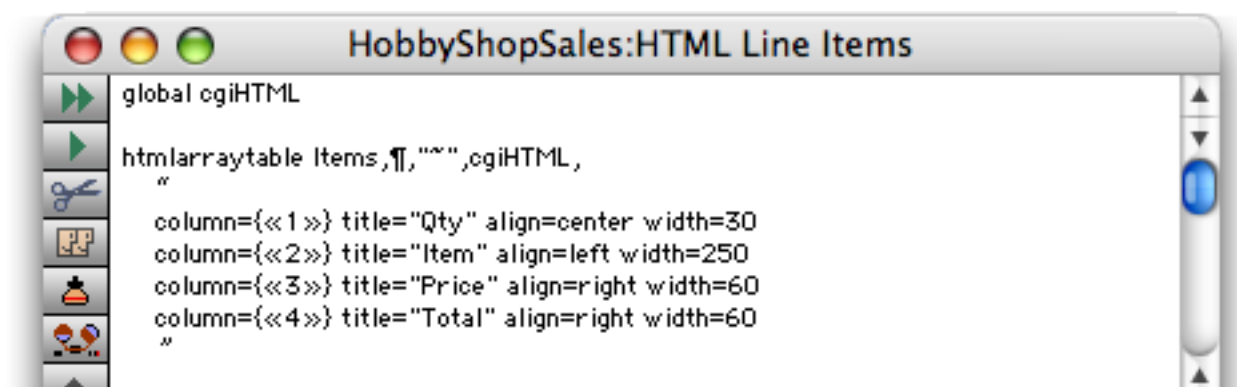
Here's a very simple example that displays only the first two columns from the array. The last two are ignored.



Here's the table rendered by this procedure.

| | |
|---|-----------------------------|
| 1 | SP 50' SD Box Car |
| 1 | WP 2-8-0 |
| 1 | Amtrak Genesis |
| 1 | SP 30' Gondola |
| 1 | ATSF 50' Box (Grand Canyon) |
| 1 | C & NW SDP-40 |

The next example displays all four columns: *Qty*, *Item*, *Price* and *Total*. (Note: In this example the options are shown formatted on separate lines for clarity, but this formatting is not necessary.)



This table includes titles, alignment and specific widths.

| Qty | Item | Price | Total |
|-----|-----------------------------|-------|-------|
| 1 | SP 50' SD Box Car | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | Amtrak Genesis | 47.99 | 47.99 |
| 1 | SP 30' Gondola | 13.69 | 13.69 |
| 1 | ATSF 50' Box (Grand Canyon) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |

The columns don't have to be displayed in the same order that they occur in the array.

```

global cgiHTML
htmlarraytable Items,¶, "", cgiHTML,
"
column={«1»} title="Qty" align=center width=30
column={«3»} title="Price" align=center width=60
column={«2»} title="Item" align=left width=250
column={«4»} title="Total" align=right width=60
"

```

In this example the Price has been moved to the second column of the rendered table, even though it is in the third column of the array.

| Qty | Price | Item | Total |
|-----|-------|-----------------------------|-------|
| 1 | 11.19 | SP 50' SD Box Car | 11.19 |
| 1 | 94.99 | WP 2-8-0 | 94.99 |
| 1 | 47.99 | Amtrak Genesis | 47.99 |
| 1 | 13.69 | SP 30' Gondola | 13.69 |
| 1 | 4.49 | ATSF 50' Box (Grand Canyon) | 4.49 |
| 1 | 29.99 | C & NW SDP-40 | 29.99 |

You can modify the array data with a formula before it is rendered.

```

global cgiHTML
htmlarraytable Items,¶, "", cgiHTML,
"
column={«1»} title="Qty" align=center width=30
column={upper(«2»)} title="Item" align=left width=250
column={«3»} title="Price" align=center width=60
column={«4»} title="Total" align=right width=60
"

```

This formula displays the second column in all upper case (we've also moved the Price column back to the traditional location).

| Qty | Item | Price | Total |
|-----|-----------------------------|-------|-------|
| 1 | SP 50' SD BOX CAR | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | AMTRAK GENESIS | 47.99 | 47.99 |
| 1 | SP 30' GONDOLA | 13.69 | 13.69 |
| 1 | ATSF 50' BOX (GRAND CANYON) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |

You can add text constants to the formula, for example HTML tags. Be careful about quoting, though. In this example you would not want to use " " or { } as quote characters within the formula.

```
global cgiHTML
htmlarraytable Items,[""],cgiHTML,
"
column={«1»} title="Qty" align=center width=30
column={upper(«2»)} title="Item" align=left width=250
column={«3»} title="Price" align=center width=60
column={\"<b>+«4>+\"</b>} title="Total" align=right width=60
"
```

This formula makes the Total column bold.

| Qty | Item | Price | Total |
|-----|-----------------------------|-------|--------------|
| 1 | SP 50' SD BOX CAR | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | AMTRAK GENESIS | 47.99 | 47.99 |
| 1 | SP 30' GONDOLA | 13.69 | 13.69 |
| 1 | ATSF 50' BOX (GRAND CANYON) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |

Remember that numeric values within the array are really text. So if you want to do a calculation with a number (or display it using an output pattern) you must convert it to a number first with the `val()` function.

```
global cgiHTML
htmlarraytable Items,[""],cgiHTML,
"
column={«1»} title="Qty" align=center width=30
column={upper(«2»)} title="Item" align=left width=250
column={«3»} title="Price" align=center width=60
column={\"<b>+pattern(val(«4»),\"$ #,####\")+\"</b>} title="Total" align=right width=70
"
```

This example uses the `pattern()` function to format the totals.

| Qty | Item | Price | Total |
|-----|-----------------------------|-------|-------------------|
| 1 | SP 50' SD BOX CAR | 11.19 | \$ 11.1900 |
| 1 | WP 2-8-0 | 94.99 | \$ 94.9900 |
| 1 | AMTRAK GENESIS | 47.99 | \$ 47.9900 |
| 1 | SP 30' GONDOLA | 13.69 | \$ 13.6900 |
| 1 | ATSF 50' BOX (GRAND CANYON) | 4.49 | \$ 4.4900 |
| 1 | C & NW SDP-40 | 29.99 | \$ 29.9900 |

You can use any operator or function you want in a formula, and you can also include other fields or variables.

Table Font, Font Size and Color

There are four “template” options that allow you to specify how text will be displayed within the table.

| Option | Description |
|-----------------------------|---|
| font=font name | This option specifies the font to use to display the table. (If left blank the browsers default font will be used.) On your own computer you can use any font you want because you have complete control over what fonts are installed and available for use. A web page, however, is the opposite situation since you have zero control over what fonts may be available when the page is viewed. Because of this it is usually best to pick from a restricted subset of fonts that are almost universally available. The fonts that are most commonly available are: <i>Arial</i> , <i>Comic Sans MS</i> , <i>Courier</i> , <i>Georgia</i> , <i>Helvetica</i> , <i>Times</i> and <i>Verdana</i> . If the font name contains spaces or punctuation it must be surrounded with quotes, for example <code>font="Comic Sans"</code> |
| fontsize=1..7/+1..+7/-1..-7 | This option specifies the size of the text to be used for the table. But take note — the size is not specified in pixels. Instead it is specified using a special HTML text size specification. Absolute values (1-7) specify a fixed font size from extremely small (1) to huge (7). Negative values specify sizes smaller than the text size in the rest of the page, for example -2 is two sizes smaller than normal font. Positive values specify sizes larger than the text size in the rest of the page, for example +2 is two sizes larger than normal font. |
| fontcolor=color | By default Panorama web tables use black text, but you can use any color you want. HTML colors are defined using a hexadecimal notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex #00). The highest value is 255 (hex #FF). For example <pre>fontcolor="0000FF"</pre> represents pure blue, <pre>fontcolor="00FF00"</pre> represents pure green, and <pre>fontcolor="FF0000"</pre> represents pure red. Using this notation you can specify millions of different colors. |
| titlecolor=color | The text in the title row is normally the same color as the rest of the text in the table (see fontcolor above) but you can specify a different color, for example <code>titlecolor="660000"</code> for dark red. |

See “[Table Font](#)” on page 289, “[Text Size](#)” on page 290 and “[Text and Background Colors](#)” on page 291 for examples of different font, size and color settings.

Table Borders and Spacing

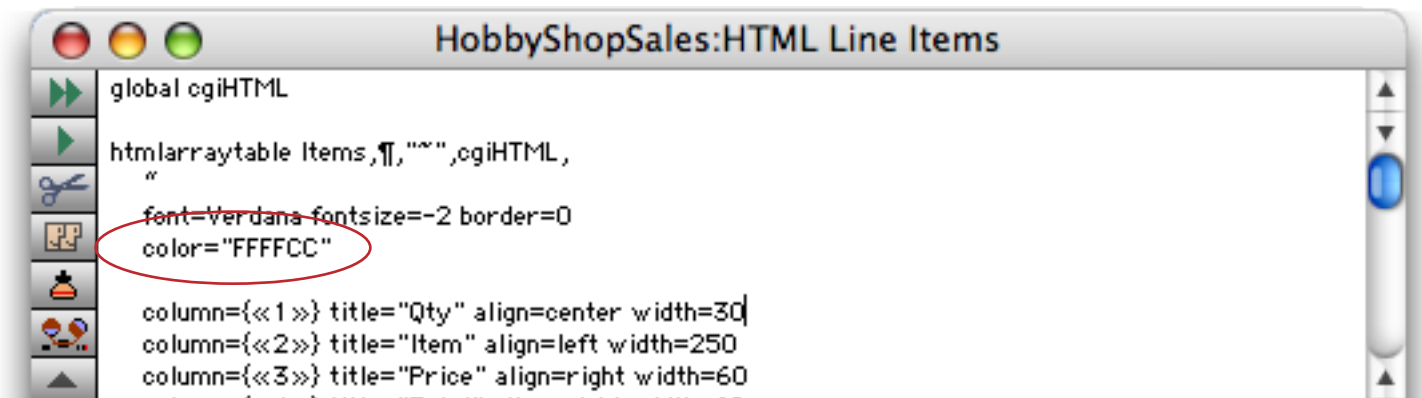
There are four “template” options that allow you to specify how text will be displayed within the table.

| Option | Description |
|--------------------|---|
| border=pixels | This option specifies the thickness (in pixels) of the border around each cell in the table. If the value is set to 0 then there is no border at all. |
| cellspacing=pixels | This option specifies the space between each cell in the table (in pixels). |
| cellpadding=pixels | This option specifies the space between the contents of each cell (text or image) and the edge of each cell (in pixels). |

See “[Table Border](#)” on page 287, “[Cell Spacing](#)” on page 288 and “[Cell Padding](#)” on page 288 for examples of different border and spacing settings.

Table Background Colors.

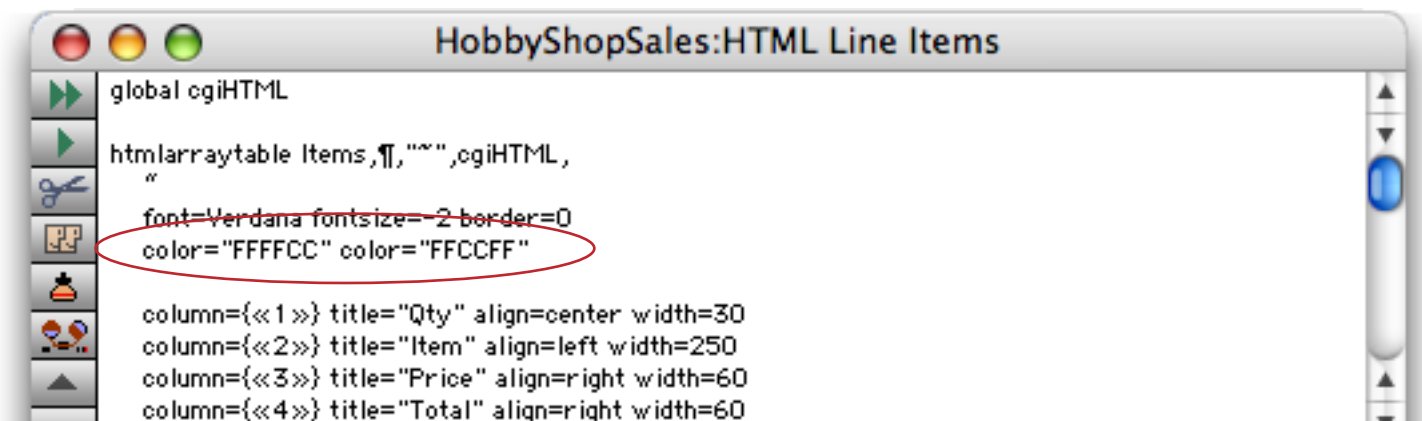
The `color` option allows you to specify a background color for the table.



In this example the background color has been set to pale yellow.

| Qty | Item | Price | Total |
|-----|-----------------------------|-------|-------|
| 1 | SP 50' SD Box Car | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | Amtrak Genesis | 47.99 | 47.99 |
| 1 | SP 30' Gondola | 13.69 | 13.69 |
| 1 | ATSF 50' Box (Grand Canyon) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |

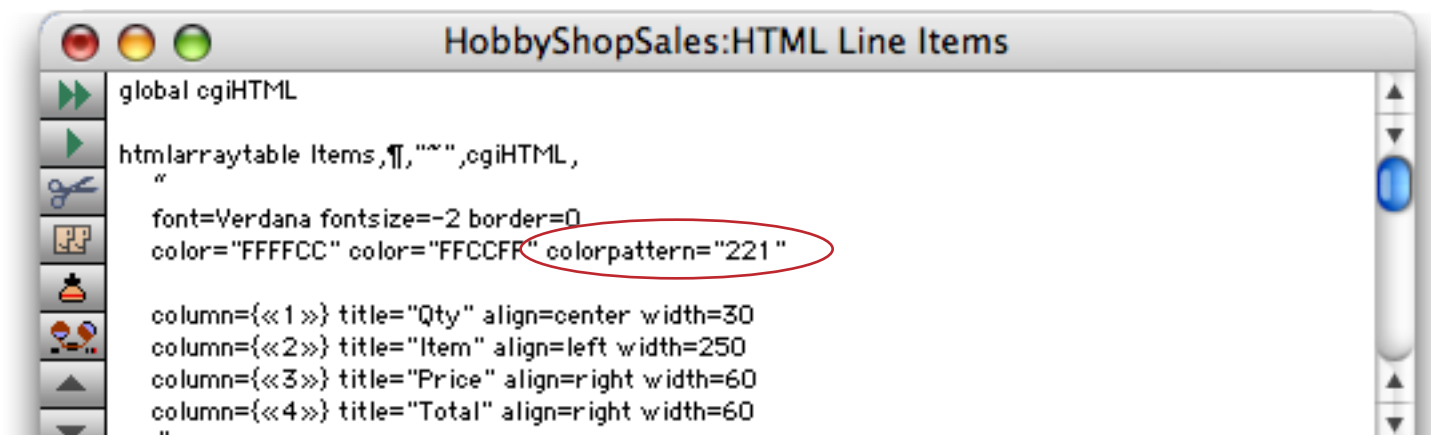
If you include two, three or more colors these colors will alternate as each row is rendered.



In this example the rows alternate between yellow and pink.

| Qty | Item | Price | Total |
|-----|-----------------------------|-------|-------|
| 1 | SP 50' SD Box Car | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | Amtrak Genesis | 47.99 | 47.99 |
| 1 | SP 30' Gondola | 13.69 | 13.69 |
| 1 | ATSF 50' Box (Grand Canyon) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |

The `colorpattern` option customizes the pattern used to alternate between colors. For example the option `colorpattern="1112"` will render two rows in the second color (pink), then one row in the first color (yellow).



```

global cgiHTML
htmlarraytable Items, "", "", cgiHTML,
"
font=Verdana fontsize=-2 border=0
color="FFFFCC" color="FFCCFF" colorpattern="221"

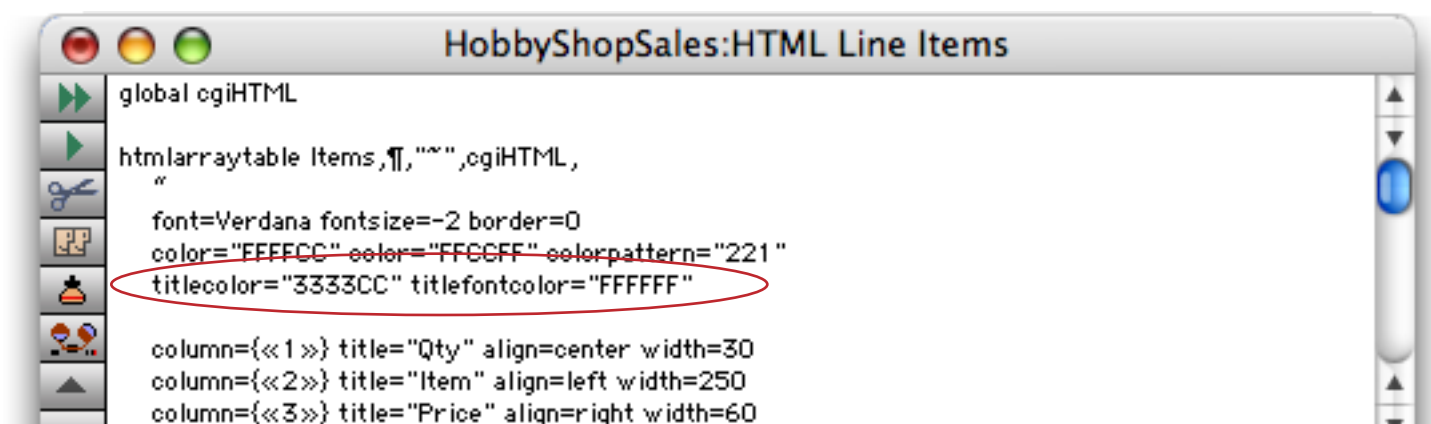
column={<<1>>} title="Qty" align=center width=30
column={<<2>>} title="Item" align=left width=250
column={<<3>>} title="Price" align=right width=60
column={<<4>>} title="Total" align=right width=60
"

```

This pattern produces two rows with a pink background then one with a yellow background.

| Qty | Item | Price | Total |
|-----|-----------------------------|-------|-------|
| 1 | SP 50' SD Box Car | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | Amtrak Genesis | 47.99 | 47.99 |
| 1 | SP 30' Gondola | 13.69 | 13.69 |
| 1 | ATSF 50' Box (Grand Canyon) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |

The `titlecolor` option allows you to specify a separate background color for the title row, while the `titlefontcolor` option specifies the color of the text in the title row.



```

global cgiHTML
htmlarraytable Items, "", "", cgiHTML,
"
font=Verdana fontsize=-2 border=0
color="FFFFCC" color="FFCCFF" colorpattern="221"
titlecolor="3333CC" titlefontcolor="FFFFFF"

column={<<1>>} title="Qty" align=center width=30
column={<<2>>} title="Item" align=left width=250
column={<<3>>} title="Price" align=right width=60
"

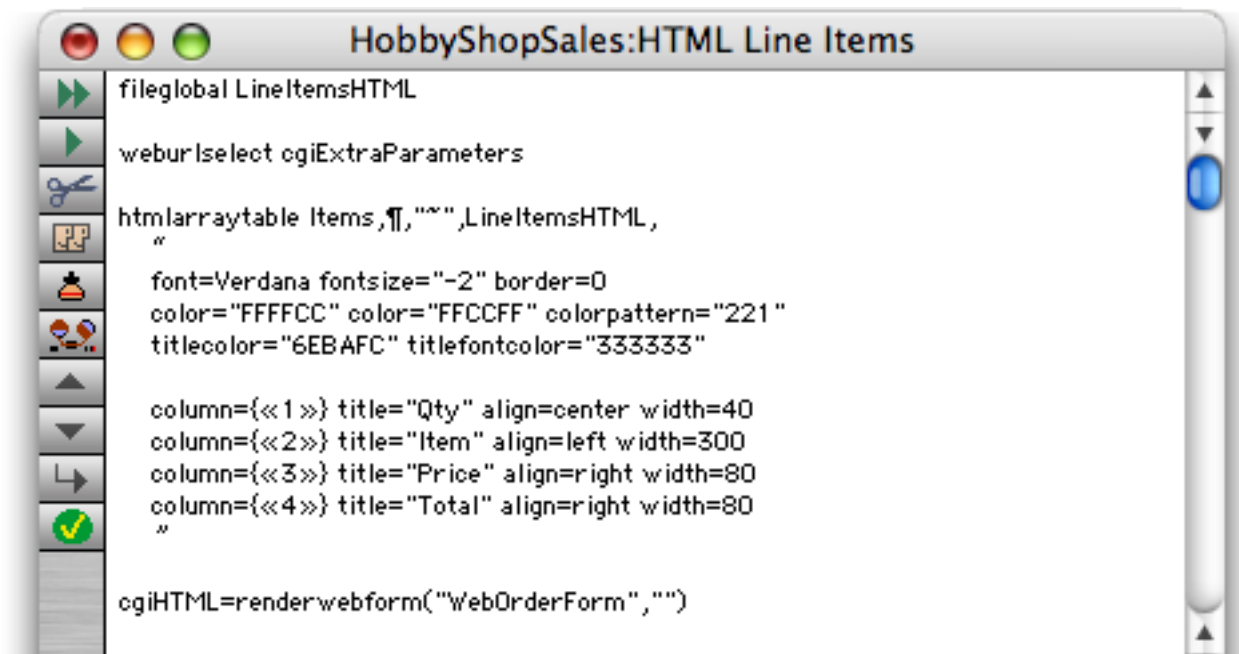
```

This example displays the title with white text on a blue background.

| Qty | Item | Price | Total |
|-----|-----------------------------|-------|-------|
| 1 | SP 50' SD Box Car | 11.19 | 11.19 |
| 1 | WP 2-8-0 | 94.99 | 94.99 |
| 1 | Amtrak Genesis | 47.99 | 47.99 |
| 1 | SP 30' Gondola | 13.69 | 13.69 |
| 1 | ATSF 50' Box (Grand Canyon) | 4.49 | 4.49 |
| 1 | C & NW SDP-40 | 29.99 | 29.99 |

Displaying an Array in a Web Form

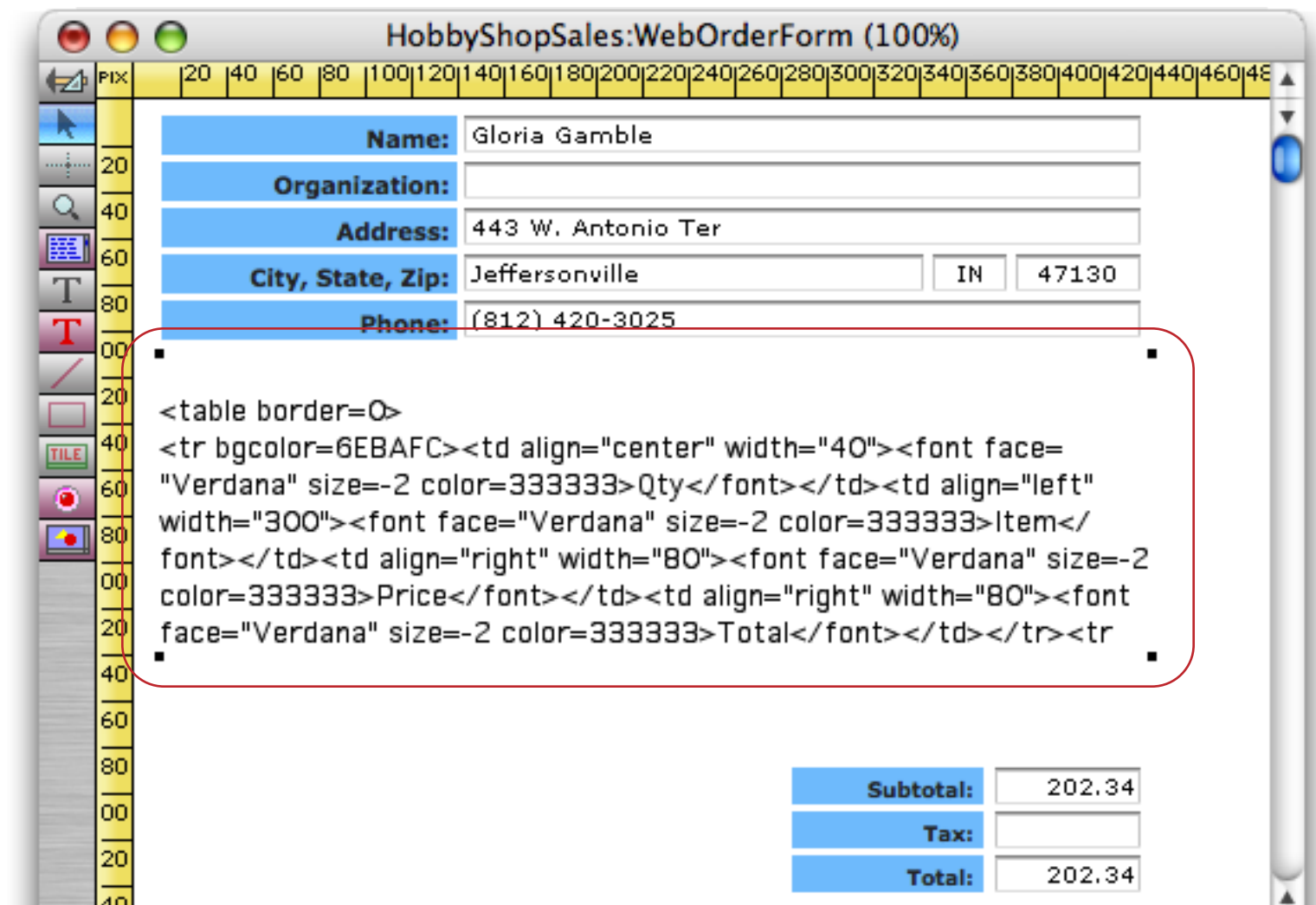
To include an array in a web form you'll need to write a special procedure that converts the array into HTML format and stores it the HTML text in a variable. The procedure then displays the form (you'll need to make sure that the variable is included in the form template, see below).



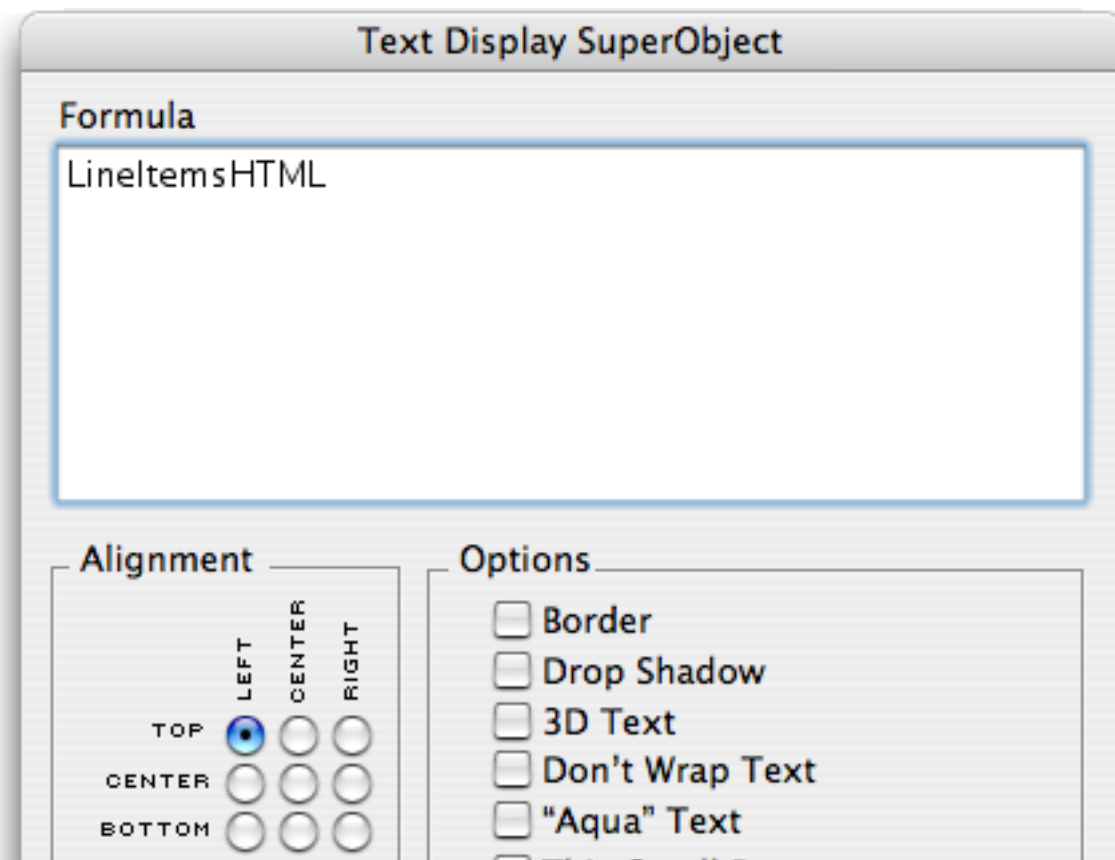
This procedure starts by selecting the record to display using the `weburlselect` statement (see “[What Record Are We Talking About?](#)” on page 395). Depending on your application you might want to use the `webformtodatabase` statement instead (see “[WebFormToDatabase — Modifying an Existing Record](#)” on page 476).

The next step is to generate the line item HTML. The generated text is stored in a fileglobal variable named `LineItemsHTML` (you can of course use any name you want).

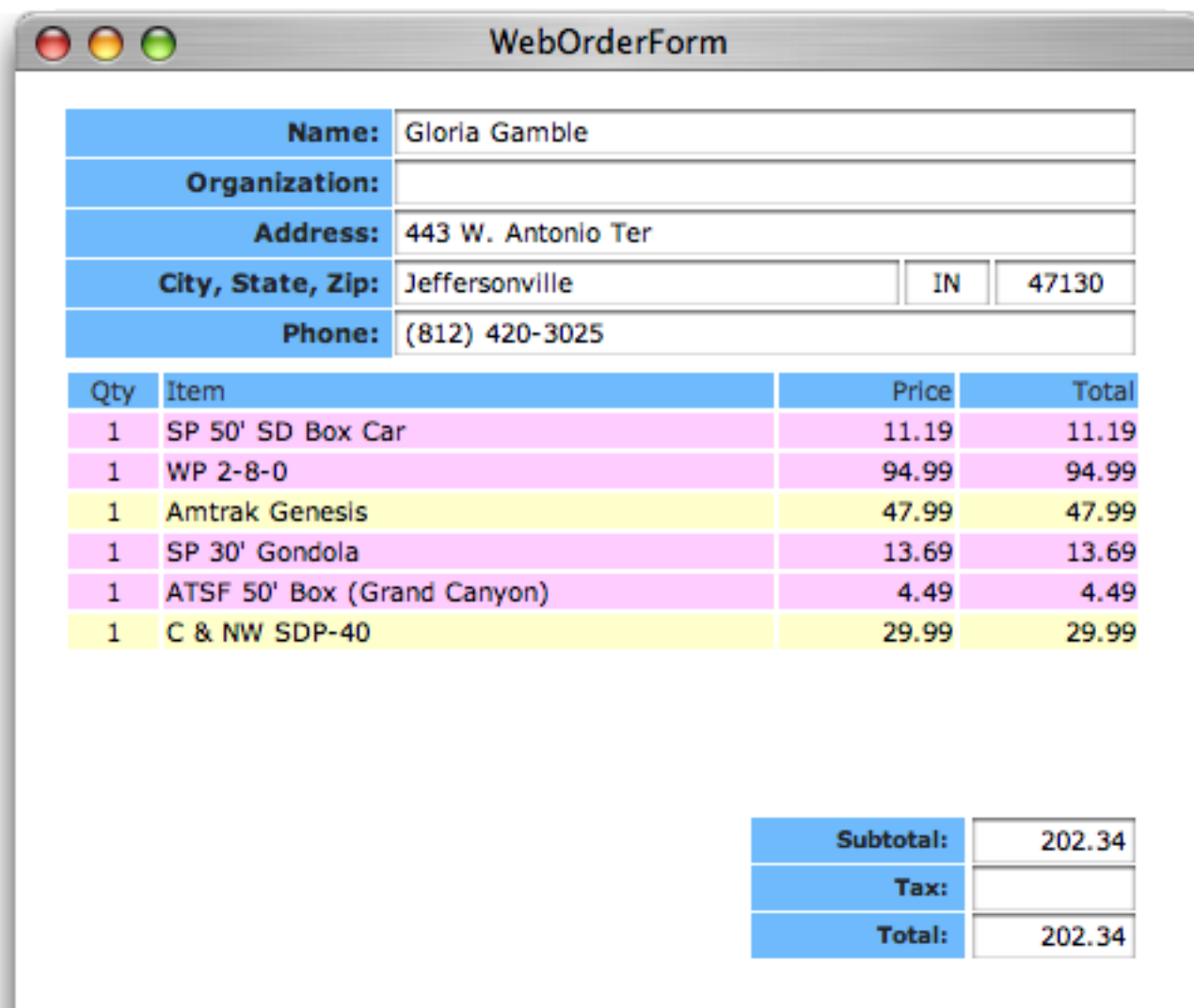
The final step is to render the form (see “[Generating a Page Using a Panorama Form Template](#)” on page 379), which in this case is called `WebOrderForm`. Here's what this form looks like in graphics mode:



The line items are displayed by the Text Display SuperObject that is highlighted. This object has been configured to display the [LineItemsHTML](#) variable, as you can see by double clicking on it:



The name of the variable must match the variable generated by the procedure. When this procedure is run on the web server the form will include the line items:



Since you can't really see what the line items look like in Panorama you'll probably need to use a bit of trial and error to get the line items to look good on the form.

Generating an HTML Table or List from Multiple Records

So far in this chapter we've been primarily been talking about displaying records one at a time — one invoice, one property, one customer, etc. For many applications you'll need to display a set of records as a table or a list. There are several approaches to doing this — you can use a web table template, you can create a table or list “from scratch”, or you can use a combination approach.

What Records are we Talking About? (The WebSelect Statement)

The first step in displaying a list of records is deciding which group of records will be included in the list. Normally in Panorama procedures you use the `select` statement to determine what records are visible. When writing a web procedure, however, you should use the `webselect` statement instead. These statements are very similar, but the `webselect` statement has additional logic that works together with the `htmldatatable` statement and `webdatatable()` function for handling empty selections and to allow selection results to be displayed over multiple pages.

The `webselect` statement is almost identical in use to the `select` statement. It has one parameter, a formula used to specify what records to select. Here's an example:

```
webselect Price >= 250.00 and Price < 1000.00
cgiHTML = renderwebtable("Product Catalog")
```

A side effect of the `webselect` statement is that a special global variable named `cgiSelectedRecordCount` is created. The value of this variable will be set to the number of selected records, or zero if no records match the selection. If your procedure is manually generating HTML you should check this variable to make sure that any records were selected. (If you use the `htmldatatable` statement or the `webdatatable()` function this is taken care of for you.)

Using Variables in a WebSelect Statement

One important restriction of the `webselect` statement is that you cannot directly use variables in the selection formula. If you want to use variables they must be enclosed in special tags. Variables that contain text must be enclosed within `<<` and `>>` tags, for example `<<Color>>` or `<<rCity>>`. Variables that contain numbers (including dates) must be enclosed within `#<<` and `>>#` tags, for example `#<<maxPrice>>#` or `#<<startDate>>#`.

Here's an example of a selection procedure that uses two numeric local variables in the selection formula.

```
local priceOption,lowPrice,highPrice
priceOption=cgiExtraParameters
case priceOption match "low"
  lowPrice=0 highPrice=250
case priceOption match "mid"
  lowPrice=250 highPrice=1000
case priceOption match "high"
  lowPrice=1000 highPrice=999999
defaultcase
  lowPrice=0 highPrice=999999
endcase
webselect Price >= #<<lowPrice>># and Price < #<<highPrice>>#
cgiHTML = renderwebtable("Product Catalog")
```

Here is another example that uses two text variables. (The variables are text because they have been extracted from form input, which is always text. The selection formula converts the values to numbers, which also could have been done in advance.)

```
fileglobal minPrice,maxPrice,priceSearchError

webformitem "minPrice",minPrice
webformitem "maxPrice",maxPrice

webselect Asking >= val("<<minPrice>>") and Asking <= val("<<maxPrice>>")
cgiHTML=renderwebtable("ListingTable")
```

Other Web Selection Statements

In addition to `webselect`, you can also use two specialized statements to perform selections, `webformselection` (see “[Web Form Based Data Selection](#)” on page 479), which selects data based on input from a web form, or `weburlselect` (see “[What Record Are We Talking About?](#)” on page 395), which selects data based on the contents of a URL.

Why Not Use the Select Statement?

In case you are wondering, if you use the `select` statement instead of `webselect`, `webformselection`, or `weburlselect` the first page of the table may display properly, however, any additional pages will not display the proper selection (see “[Splitting a Long Table into Multiple Pages](#)” on page 424). Also if the `select` statement is used the server will not display an empty table if no records are displayed (instead it will display whatever records were previously selected). The bottom line is that you should avoid the standard `select` statement and use one of the web selection statements instead.

Generating HTML Tables Using a Web Table Template

Usually the easiest way to generate an HTML table from multiple database records is to use a pre-defined web table template that you have set up in advance with the **Text Export** wizard. This wizard uses dialogs to define various attributes of the table (“[Web Tables](#)” on page 277). Web tables are normally displayed automatically using the query action (see “[Linking a Table with a Query Form](#)” on page 299) but they can also be used in a web procedure.

To use a Panorama web form use the `renderwebtable()` function in the formula that creates the HTML page. This function has two parameters.

```
renderwebtable(tablename)
```

The function’s one parameter is the name of the table, which must be in the current database, and must already have been created and uploaded to the server with the **Text Export** wizard (see “[Web Tables](#)” on page 277).

Here is an example of how this function can be used to generate a web page with a table:

```
cgiHTML = renderwebtable("Product Catalog")
```

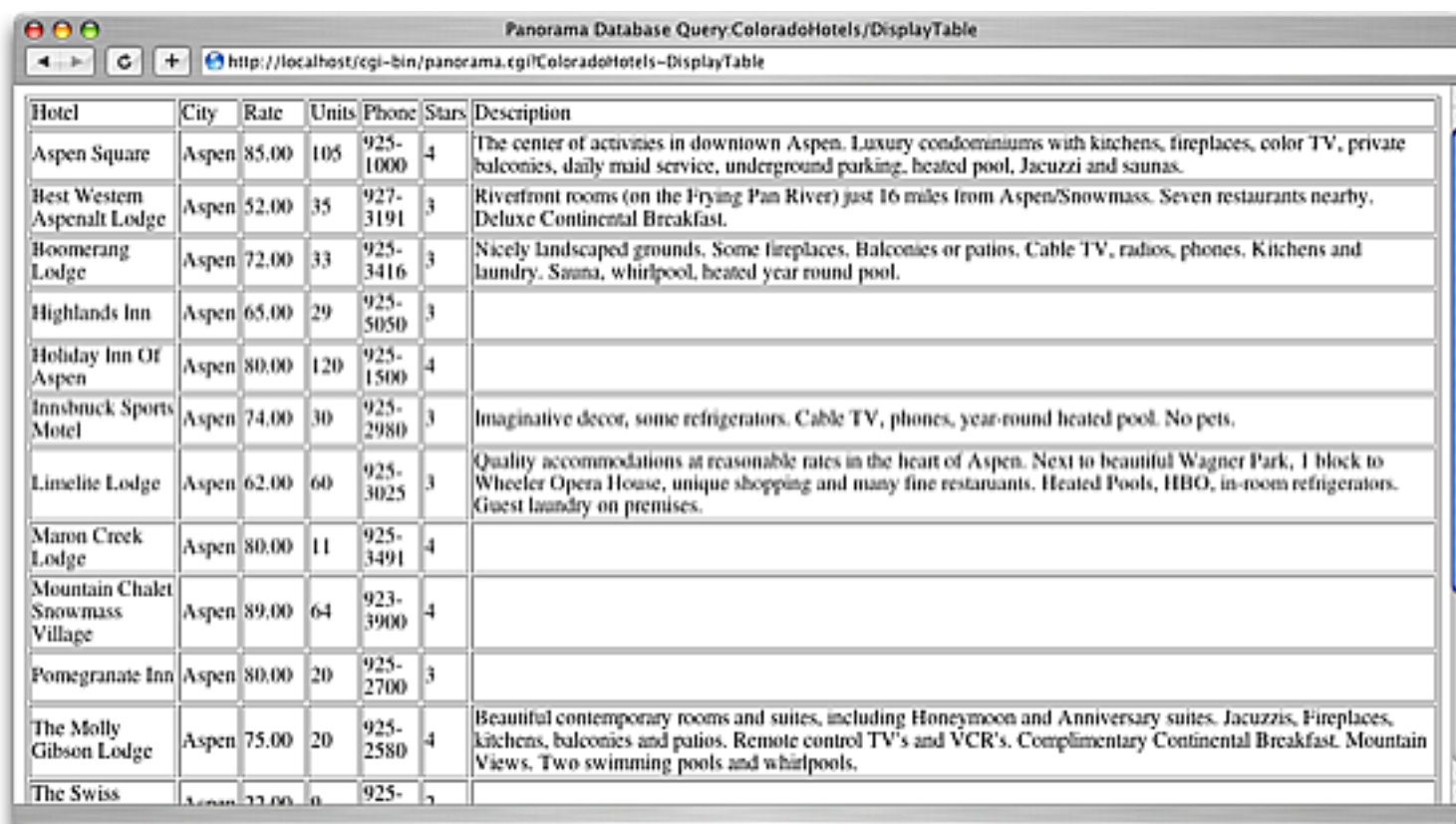
If you want to use a pre-built table template but customize it slightly see “[Modifying a Web Table Template On the Fly](#)” on page 427.

Generating HTML Tables Without a Template (“from scratch”)

If you don’t want to set up a web table template in advance you can use the `htmldatatable` statement to generate an HTML table for you. This statement has all of the same options as a web table template but allows these options to be set up on the fly as the procedure runs. The `htmldatatable` statement will render the currently selected records as an HTML table. This statement has two parameters. The first parameter is the name of the variable that will receive the HTML table. The second parameter is a list of options that specify what fields should be included in the table and how the table should be formatted. If there are no options supplied the statement will automatically include all fields in the database.



The resulting table isn't all that pretty, but it sure didn't take much programming to create!



| Hotel | City | Rate | Units | Phone | Stars | Description |
|----------------------------------|-------|-------|-------|----------|-------|---|
| Aspen Square | Aspen | 85.00 | 105 | 925-1000 | 4 | The center of activities in downtown Aspen. Luxury condominiums with kitchens, fireplaces, color TV, private balconies, daily maid service, underground parking, heated pool, Jacuzzi and saunas. |
| Best Western Aspenalt Lodge | Aspen | 52.00 | 35 | 927-3191 | 3 | Riverfront rooms (on the Frying Pan River) just 16 miles from Aspen/Snowmass. Seven restaurants nearby. Deluxe Continental Breakfast. |
| Boomerang Lodge | Aspen | 72.00 | 33 | 925-3416 | 3 | Nicely landscaped grounds. Some fireplaces. Balconies or patios. Cable TV, radios, phones, Kitchens and laundry. Sauna, whirlpool, heated year round pool. |
| Highlands Inn | Aspen | 65.00 | 29 | 925-5050 | 3 | |
| Holiday Inn Of Aspen | Aspen | 80.00 | 120 | 925-1500 | 4 | |
| Innsbruck Sports Motel | Aspen | 74.00 | 30 | 925-2980 | 3 | Imaginative decor, some refrigerators. Cable TV, phones, year-round heated pool. No pets. |
| Limestone Lodge | Aspen | 62.00 | 60 | 925-3025 | 3 | Quality accommodations at reasonable rates in the heart of Aspen. Next to beautiful Wagner Park, 1 block to Wheeler Opera House, unique shopping and many fine restaurants. Heated Pools, HBO, in-room refrigerators. Guest laundry on premises. |
| Maron Creek Lodge | Aspen | 80.00 | 11 | 925-3491 | 4 | |
| Mountain Chalet Snowmass Village | Aspen | 89.00 | 64 | 923-3900 | 4 | |
| Pomegranate Inn | Aspen | 80.00 | 20 | 925-2700 | 3 | |
| The Molly Gibson Lodge | Aspen | 75.00 | 20 | 925-2580 | 4 | Beautiful contemporary rooms and suites, including Honeymoon and Anniversary suites. Jacuzzis, Fireplaces, kitchens, balconies and patios. Remote control TV's and VCR's. Complimentary Continental Breakfast. Mountain Views. Two swimming pools and whirlpools. |
| The Swiss | Aspen | 77.00 | 0 | 925- | 3 | |

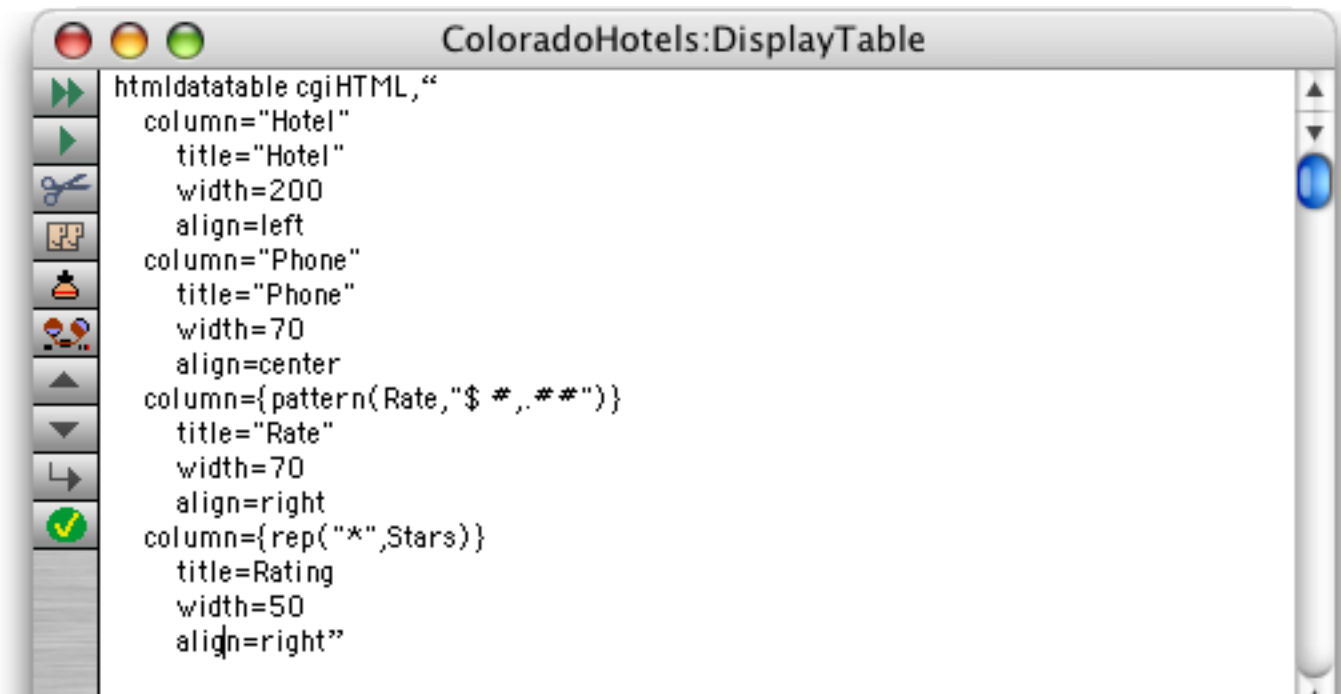
To customize the appearance of the table you'll need to specify some options in the second parameter of the `htmldatatable` statement. The options are specified as assignments, similar to the options in many HTML tags. These options are discussed in the following sections.

Table Field Layout

There are four options that allow you to specify what fields will be included in the table and how they are titled and arranged.

| Option | Description |
|--------------------------------------|---|
| <code>column=formula</code> | You should include one <code>column=</code> option for each column in the table. The formula specifies what data will be displayed in the column. For text fields you can simply include the name of the field. For date fields you must use the <code>datepattern()</code> function, while for numeric fields you must use either the <code>str()</code> or <code>pattern()</code> function. |
| <code>title=text</code> | If you specify any titles at all you must include one <code>title=</code> option for each column in the table. |
| <code>width=number</code> | If you specify any widths at all you must include one <code>width=</code> option for each column in the table. The width is specified in pixels, for example <code>width=72</code> for a column that is 1 inch wide. |
| <code>align=left/center/right</code> | If you specify any alignment at all you must include one <code>align=</code> option for each column in the table. The alignment for each field may be left, center or right (for example <code>align=right</code>). |

This example shows these option in use to create a table with four columns: *Hotel*, *Phone*, *Rate* and *Rating*. (Note: In this example the options are shown formatted on separate lines for clarity, but this formatting is not necessary.)

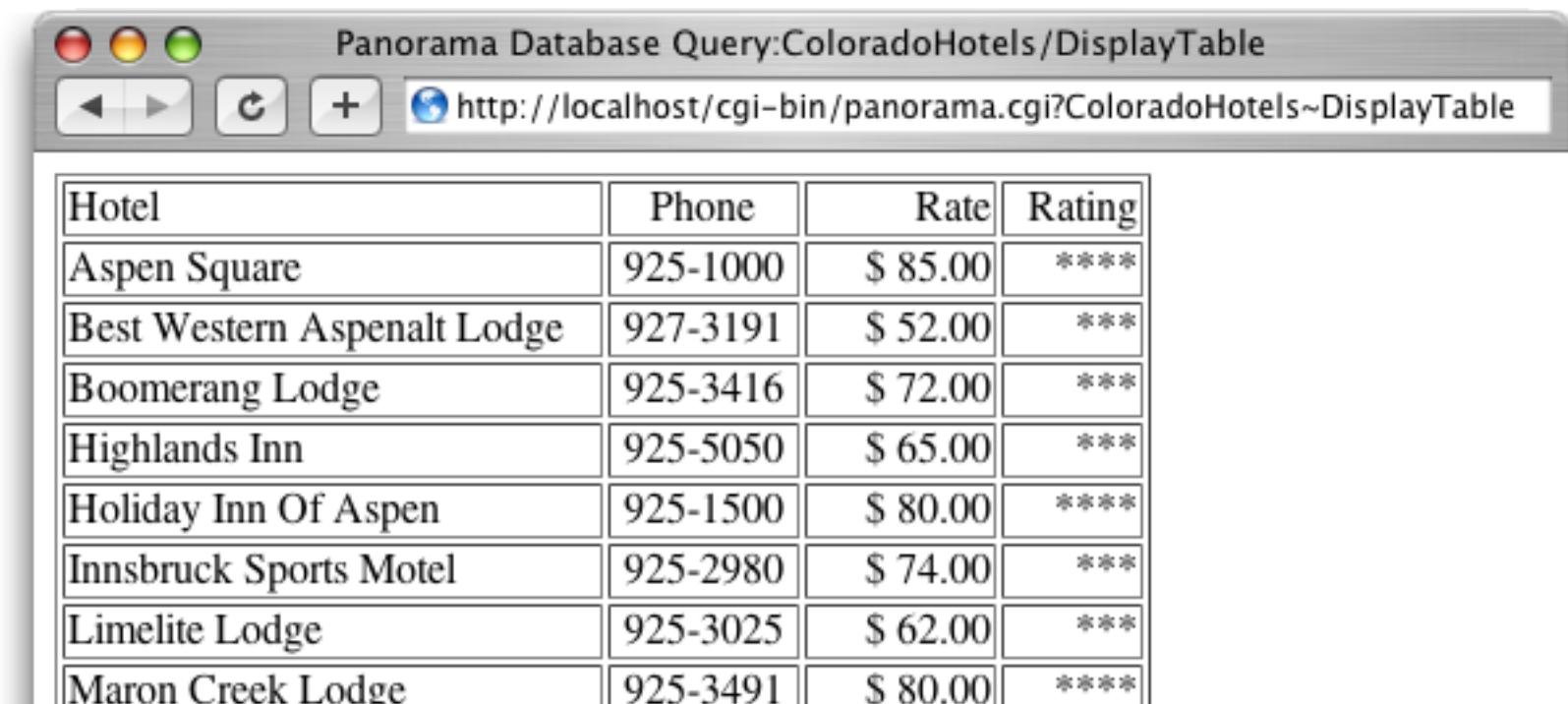


```

htmltable cgiHTML,“
  column="Hotel"
  title="Hotel"
  width=200
  align=left
  column="Phone"
  title="Phone"
  width=70
  align=center
  column={pattern(Rate,"$ #,##")}
  title="Rate"
  width=70
  align=right
  column={rep("*",Stars)}
  title=Rating
  width=50
  align=right”

```

Here's the table rendered by this procedure.



| Hotel | Phone | Rate | Rating |
|-----------------------------|----------|----------|--------|
| Aspen Square | 925-1000 | \$ 85.00 | **** |
| Best Western Aspenalt Lodge | 927-3191 | \$ 52.00 | *** |
| Boomerang Lodge | 925-3416 | \$ 72.00 | *** |
| Highlands Inn | 925-5050 | \$ 65.00 | *** |
| Holiday Inn Of Aspen | 925-1500 | \$ 80.00 | **** |
| Innsbruck Sports Motel | 925-2980 | \$ 74.00 | *** |
| Limelite Lodge | 925-3025 | \$ 62.00 | *** |
| Maron Creek Lodge | 925-3491 | \$ 80.00 | **** |

Table Font, Font Size and Color

There are four “template” options that allow you to specify how text will be displayed within the table.

| Option | Description |
|-----------------------------|---|
| font=font name | This option specifies the font to use to display the table. (If left blank the browsers default font will be used.) On your own computer you can use any font you want because you have complete control over what fonts are installed and available for use. A web page, however, is the opposite situation since you have zero control over what fonts may be available when the page is viewed. Because of this it is usually best to pick from a restricted subset of fonts that are almost universally available. The fonts that are most commonly available are: <i>Arial</i> , <i>Comic Sans MS</i> , <i>Courier</i> , <i>Georgia</i> , <i>Helvetica</i> , <i>Times</i> and <i>Verdana</i> . If the font name contains spaces or punctuation it must be surrounded with quotes, for example <code>font="Comic Sans"</code> |
| fontsize=1..7/+1..+7/-1..-7 | This option specifies the size of the text to be used for the table. But take note — the size is not specified in pixels. Instead it is specified using a special HTML text size specification. Absolute values (1-7) specify a fixed font size from extremely small (1) to huge (7). Negative values specify sizes smaller than the text size in the rest of the page, for example -2 is two sizes smaller than normal font. Positive values specify sizes larger than the text size in the rest of the page, for example +2 is two sizes larger than normal font. |
| fontcolor=color | By default Panorama web tables use black text, but you can use any color you want. HTML colors are defined using a hexadecimal notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex #00). The highest value is 255 (hex #FF). For example <pre>fontcolor="0000FF"</pre> represents pure blue, <pre>fontcolor="00FF00"</pre> represents pure green, and <pre>fontcolor="FF0000"</pre> represents pure red. Using this notation you can specify millions of different colors. |
| titlecolor=color | The text in the title row is normally the same color as the rest of the text in the table (see fontcolor above) but you can specify a different color, for example <code>titlecolor="660000"</code> for dark red. |

See “[Table Font](#)” on page 289, “[Text Size](#)” on page 290 and “[Text and Background Colors](#)” on page 291 for examples of different font, size and color settings.

Table Borders and Spacing

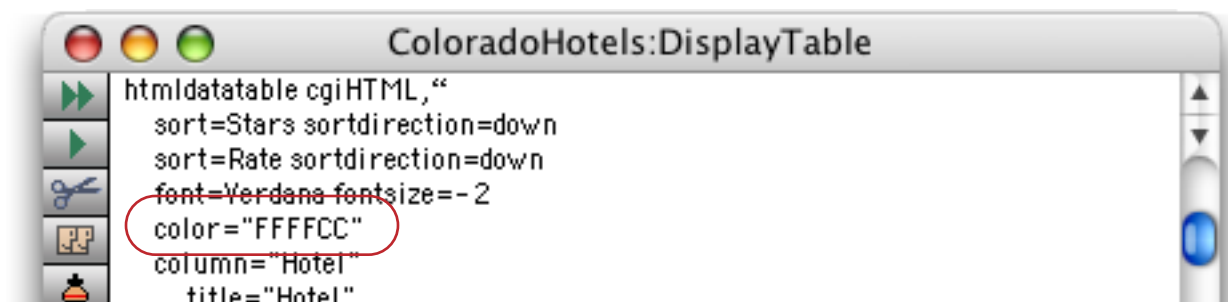
There are four “template” options that allow you to specify how text will be displayed within the table.

| Option | Description |
|--------------------|---|
| border=pixels | This option specifies the thickness (in pixels) of the border around each cell in the table. If the value is set to 0 then there is no border at all. |
| cellspacing=pixels | This option specifies the space between each cell in the table (in pixels). |
| cellpadding=pixels | This option specifies the space between the contents of each cell (text or image) and the edge of each cell (in pixels). |

See “[Table Border](#)” on page 287, “[Cell Spacing](#)” on page 288 and “[Cell Padding](#)” on page 288 for examples of different border and spacing settings.

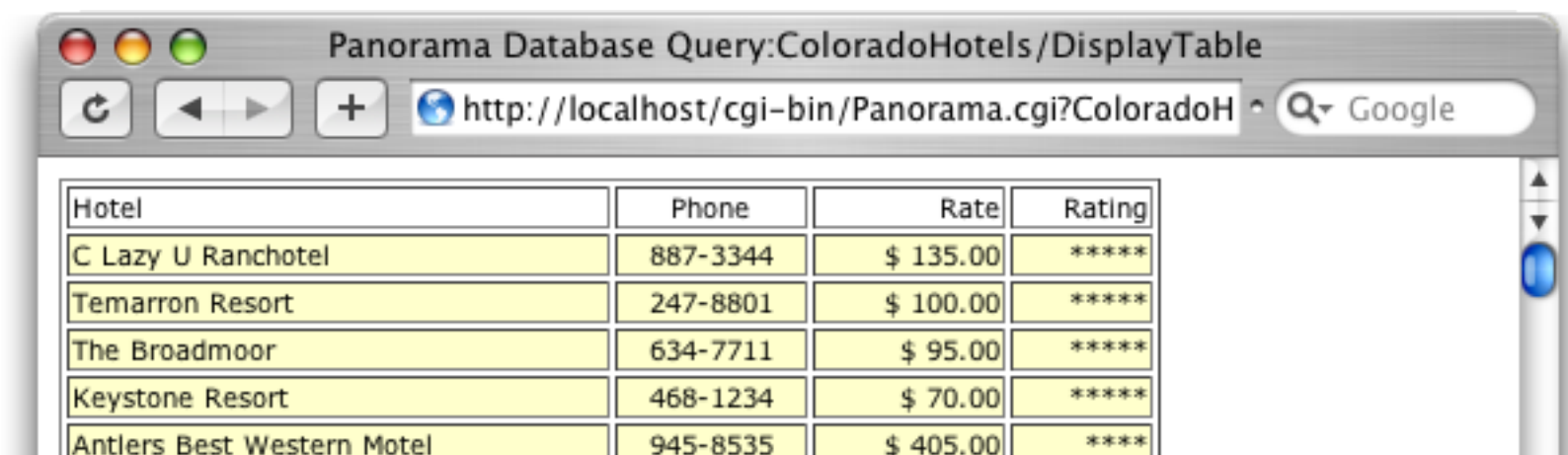
Table Background Colors.

The `color` option allows you to specify a background color for the table.



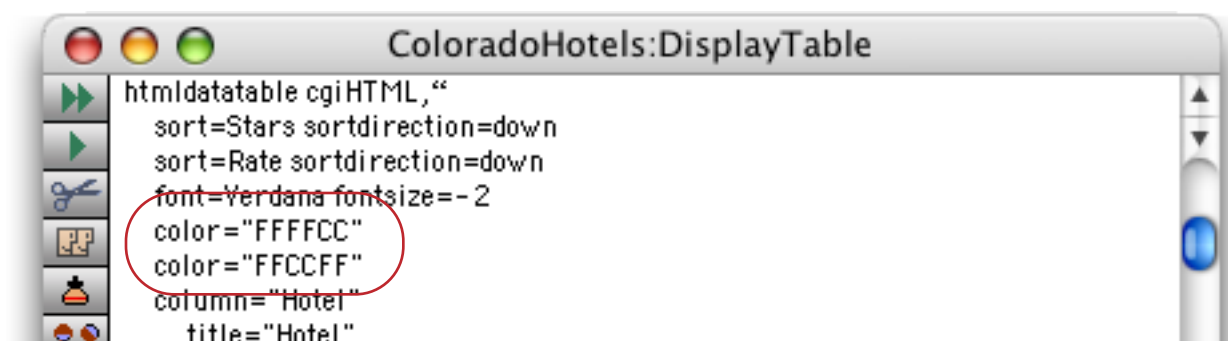
```
htmldatatable cgiHTML,“
  sort=Stars sortdirection=down
  sort=Rate sortdirection=down
  font=Verdana fontsize=- 2
  color="FFFFCC"
  column="Hotel"
  title="Hotel"
```

In this example the background color has been set to pale yellow.



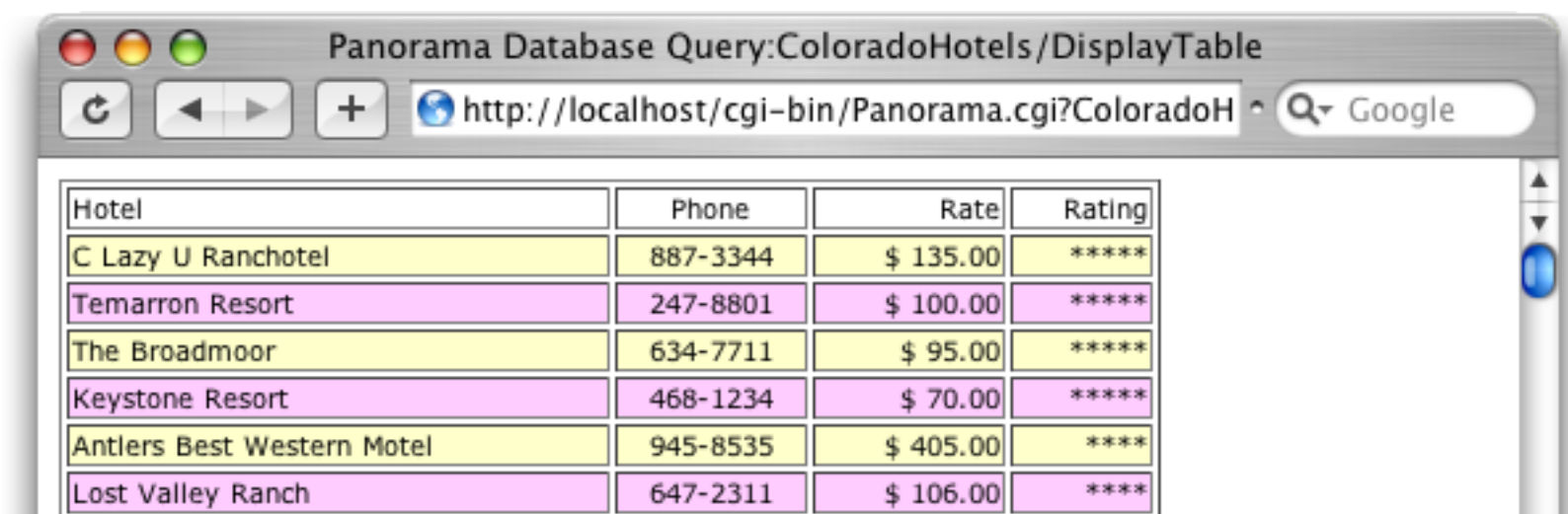
| Hotel | Phone | Rate | Rating |
|----------------------------|----------|-----------|--------|
| C Lazy U Ranchotel | 887-3344 | \$ 135.00 | ***** |
| Temarron Resort | 247-8801 | \$ 100.00 | ***** |
| The Broadmoor | 634-7711 | \$ 95.00 | ***** |
| Keystone Resort | 468-1234 | \$ 70.00 | ***** |
| Antlers Best Western Motel | 945-8535 | \$ 405.00 | **** |

If you include two, three or more colors these colors will alternate as each row is rendered.



```
htmldatatable cgiHTML,“
  sort=Stars sortdirection=down
  sort=Rate sortdirection=down
  font=Verdana fontsize=- 2
  color="FFFFCC"
  color="FFCCFF"
  column="Hotel"
  title="Hotel"
```

In this example the rows alternate between yellow and pink.

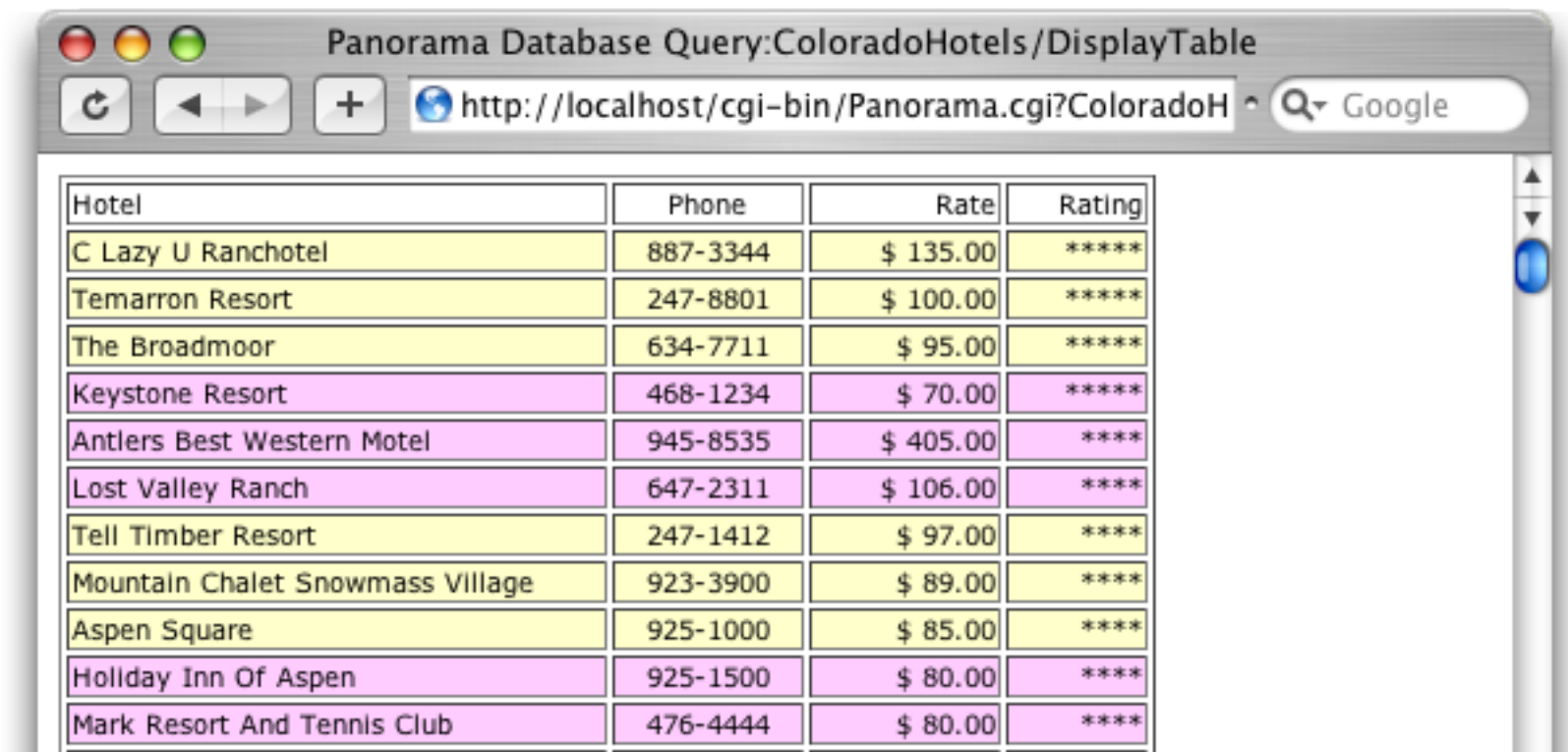


| Hotel | Phone | Rate | Rating |
|----------------------------|----------|-----------|--------|
| C Lazy U Ranchotel | 887-3344 | \$ 135.00 | ***** |
| Temarron Resort | 247-8801 | \$ 100.00 | ***** |
| The Broadmoor | 634-7711 | \$ 95.00 | ***** |
| Keystone Resort | 468-1234 | \$ 70.00 | ***** |
| Antlers Best Western Motel | 945-8535 | \$ 405.00 | **** |
| Lost Valley Ranch | 647-2311 | \$ 106.00 | **** |

The `colorpattern` option customizes the pattern used to alternate between colors. For example the option `colorpattern="1112"` will render three rows in the first color, then one row in the second color.

```
font=Verdana fontsize=-2
color="FFFFCC"
color="FFCCFF"
colorpattern="111222"
column="Hotel"
title="Hotel"
```

This pattern produces three rows with a yellow background then three with a pink background.



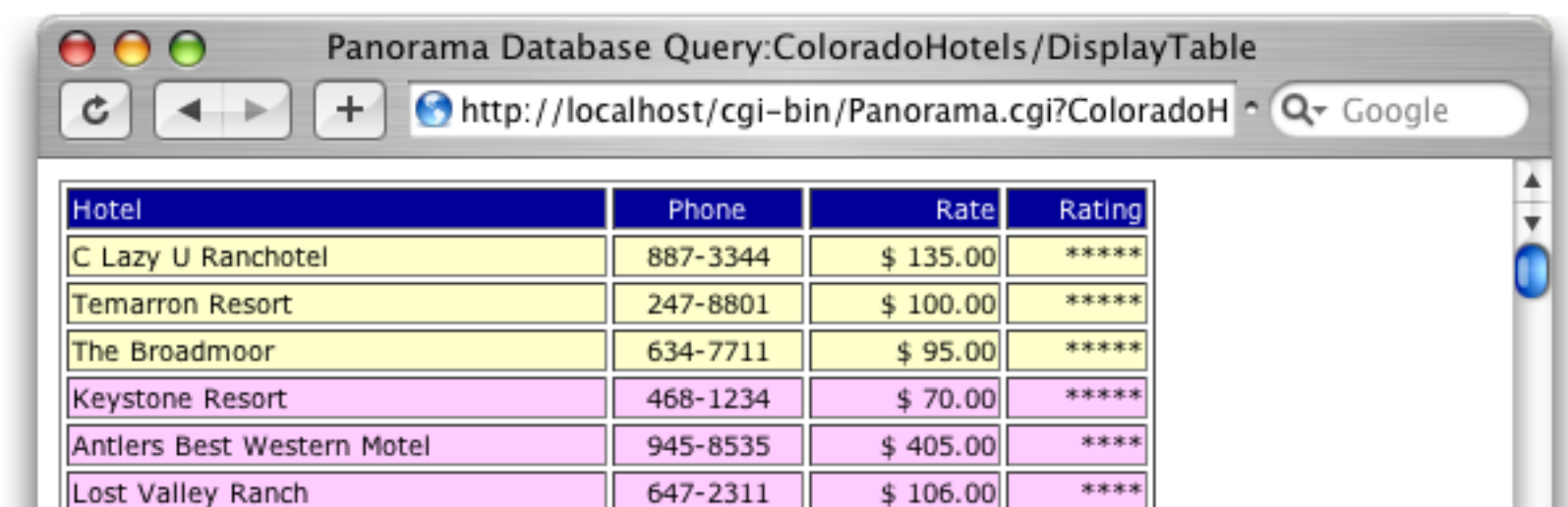
The screenshot shows a web browser window titled "Panorama Database Query:ColoradoHotels/DisplayTable". The address bar shows the URL `http://localhost/cgi-bin/Panorama.cgi?ColoradoH`. The browser displays a table with the following data:

| Hotel | Phone | Rate | Rating |
|----------------------------------|----------|-----------|--------|
| C Lazy U Ranchotel | 887-3344 | \$ 135.00 | ***** |
| Temarron Resort | 247-8801 | \$ 100.00 | ***** |
| The Broadmoor | 634-7711 | \$ 95.00 | ***** |
| Keystone Resort | 468-1234 | \$ 70.00 | ***** |
| Antlers Best Western Motel | 945-8535 | \$ 405.00 | **** |
| Lost Valley Ranch | 647-2311 | \$ 106.00 | **** |
| Tell Timber Resort | 247-1412 | \$ 97.00 | **** |
| Mountain Chalet Snowmass Village | 923-3900 | \$ 89.00 | **** |
| Aspen Square | 925-1000 | \$ 85.00 | **** |
| Holiday Inn Of Aspen | 925-1500 | \$ 80.00 | **** |
| Mark Resort And Tennis Club | 476-4444 | \$ 80.00 | **** |

The `titlecolor` option allows you to specify a separate background color for the title row.

```
color="FFFFCC"
color="FFCCFF"
colorpattern="111222"
titlecolor="000099"
titlefontcolor="FFFFFF"
column="Hotel"
```

This example displays the title with white text on a blue background.



The screenshot shows the same web browser window as before, but with the title row highlighted in blue. The table data is the same as in the previous screenshot:

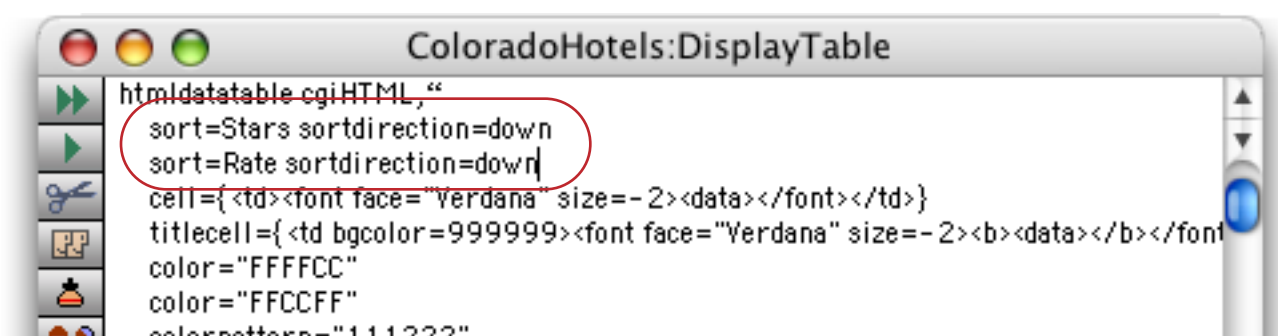
| Hotel | Phone | Rate | Rating |
|----------------------------|----------|-----------|--------|
| C Lazy U Ranchotel | 887-3344 | \$ 135.00 | ***** |
| Temarron Resort | 247-8801 | \$ 100.00 | ***** |
| The Broadmoor | 634-7711 | \$ 95.00 | ***** |
| Keystone Resort | 468-1234 | \$ 70.00 | ***** |
| Antlers Best Western Motel | 945-8535 | \$ 405.00 | **** |
| Lost Valley Ranch | 647-2311 | \$ 106.00 | **** |

Table Sort Order

If you want the table to appear in sorted order you can use one or more sort options. There are four “template” options that allow you to specify HTML templates for each component of the table.

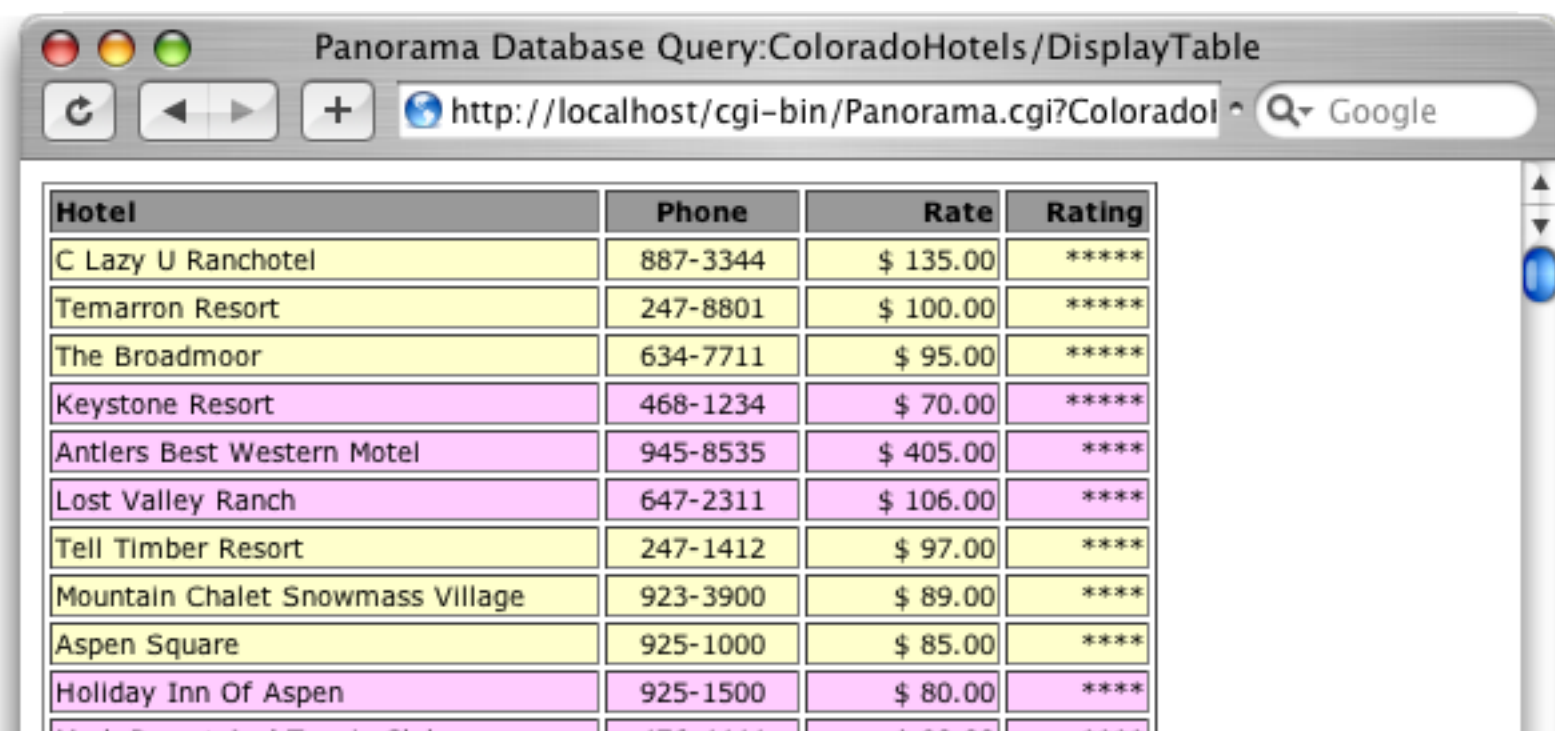
| Option | Description |
|-----------------------|---|
| sort=field | This option tells the server to sort the database before generating the table. If there is more than one sort option, the database will be sorted by each field in turn (for example State, then City, or Last, then First). You can include as many sort options as you like. |
| sortdirection=up/down | This option tell the server whether to sort in ascending (up) or descending (down) order. If more than one sort field is specified then you should also have a sortdirection option for each field. If there is no sortdirection option at all then all sorts are assumed to be ascending (up). |

This modified version of the DisplayTable procedure sorts twice.



```
htmldatatable cgiHTML,"
  sort=Stars sortdirection=down
  sort=Rate sortdirection=down
  cell={ <td><font face="Verdana" size=-2><data></font></td>}
  titlecell={ <td bgcolor=999999><font face="Verdana" size=-2><b><data></b></font>
  color="FFFFCC"
  color="FFCCFF"
  colorpattern="111222"
```

First the database is sorted by the Stars field (rating), from highest rating to lowest. Then within each rating the hotels are sorted by price, again from highest to lowest.



| Hotel | Phone | Rate | Rating |
|----------------------------------|----------|-----------|--------|
| C Lazy U Ranchotel | 887-3344 | \$ 135.00 | ***** |
| Temarron Resort | 247-8801 | \$ 100.00 | ***** |
| The Broadmoor | 634-7711 | \$ 95.00 | ***** |
| Keystone Resort | 468-1234 | \$ 70.00 | ***** |
| Antlers Best Western Motel | 945-8535 | \$ 405.00 | **** |
| Lost Valley Ranch | 647-2311 | \$ 106.00 | **** |
| Tell Timber Resort | 247-1412 | \$ 97.00 | **** |
| Mountain Chalet Snowmass Village | 923-3900 | \$ 89.00 | **** |
| Aspen Square | 925-1000 | \$ 85.00 | **** |
| Holiday Inn Of Aspen | 925-1500 | \$ 80.00 | **** |
| Mark Resort And Tennis Club | 476-4444 | \$ 80.00 | **** |

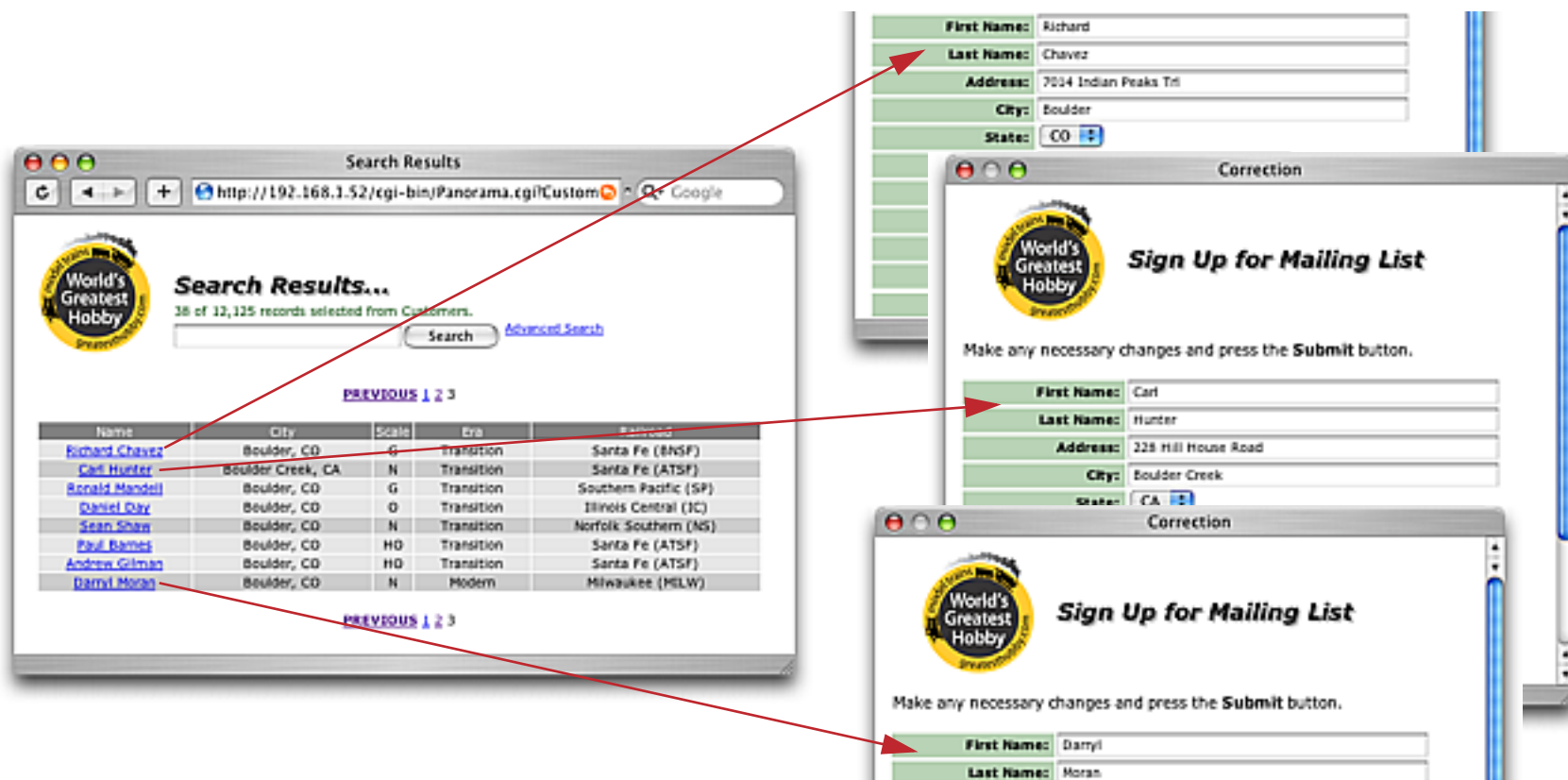
You might think that instead of including the sort options in the table specification you could simply sort the database in the procedure, like this.

```
field Stars sortdown
field Rate sortdownwithin
htmldatatable cgiHTML," ... "
```

In fact this will work if the table is displayed all on one page. However if the table may split into multiple pages (see “[Splitting a Long Table into Multiple Pages](#)” on page 424) then the sort options must be included in the table specification. If they aren’t the additional pages may not be sorted properly.

Linking Individual Table Rows to Detail Pages

In many applications a table is a jumping off point for additional pages. For example a phone list might link to individual address pages, while a product catalog table would probably have links to detail pages for each product listed. This section explains how to set up links from a table to individual pages.



There are four table specification options for setting up the link between the table and the detail pages.

| Option | Description |
|---------------------|---|
| domain=ip address | This option should be set to the ip address of your web server, for example <code>http://www.provue.com</code> or <code>http://192.168.1.3</code> . You can omit this option if you have set up the domain name in the Server Configuration, which we highly recommend (see “ Setting the IP Address/Domain Name ” on page 56). |
| linkProcedure= proc | This option tells the server what detail page to link to. If the table is linking to a Panorama web form (see “ Web Forms ” on page 231) then this option should be set to <code>form~</code> followed by the name of the form, for example: <pre>linkprocedure="form~Invoice"</pre> If the table is linking to a procedure that generates the detail page then simply use the name of the procedure. <pre>linkprocedure="DisplayRecord"</pre> This procedure must be able use the extra parameters in the URL to locate the specific record to be updated (an example of such a URL is shown below - the procedure can call the <code>urlSelect</code> to process this URL format automatically, see “ What Record Are We Talking About? ” on page 395). |
| linkOptions= html | This option can be used to insert additional text into the link tags that are generated. For example, if you want the detail form to open in a new window, you can set <code>linkOptions={target=new}</code> . You could also use the link options to open the detail form in another frame. |

| Option | Description |
|------------------|---|
| linkField= field | <p>This option specifies a field (or multiple fields) to be used to identify the link. This option is only needed for databases that are not shared — if the database is shared then the server will automatically use the shared record ID to identify the link. You may use more than one link field, but the combination of all the link fields you specify should uniquely identify a single record in the database. If our ColoradoHotels database is not shared we can uniquely identify a single record with two fields, City and Hotel, so the options should be set up this way:</p> <pre>linkField="City" linkField="Hotel"</pre> |

In addition to setting up these four options, you also need to add at least one pair of `<link>` and `</link>` tags to either a column definition (`htmldatatable`) or a template (`htmldatamerge`). Here's an example of how these tags could be added to an `htmldatatable` specification.

```
column="<link>+Hotel+</link>"
```

Here's an example of how these tags can be added to the template for `htmldatamerge`. The `<link>` and `</link>` tags must be placed inside curly braces `{` and `}`.

```
"<h2>{<link>+Hotel+</link>}</h2>
{pattern(Rate, "$#, .##")}<br>{Phone}<hr>"
```

Let's go back to the procedure we created in earlier section for rendering a table. By making the highlighted changes shown below, we can add links to detail pages to each line of the table.

```
htmldatatable cgiHTML, "
  color="FFFF66"
  color="FF99FF"
  colorpattern="111222"
  linkProcedure="DisplayRecord"
  linkOptions="target=new"
  linkField="City"
  linkField="Hotel"
  column="<link>+Hotel+</link>"
    title="Hotel"
    width=200
    align=left
  column="Phone"
    title="Phone"
    width=70
    align=center
  column={pattern(Rate, "$ #, .##")}
    title="Rate"
    width=70
    align=right
  column={rep(" *", Stars)}
    title="Rating"
    width=50
    align=right"
```


With this modification each hotel name in the table now becomes a link to a detail page.

| | | | |
|--|----------|-----------|------|
| Holiday Inn Of Aspen | 925-1500 | \$ 80.00 | **** |
| Innsbruck Sports Motel | 925-2980 | \$ 74.00 | *** |
| Limelite Lodge | 925-3025 | \$ 62.00 | *** |
| Maron Creek Lodge | 925-3491 | \$ 80.00 | **** |
| Mountain Chalet Snowmass Village | 923-3900 | \$ 89.00 | **** |
| Pomegranate Inn | 925-2700 | \$ 80.00 | *** |
| The Molly Gibson Lodge | 925-2580 | \$ 75.00 | **** |
| The Swiss Chalet | 925-7146 | \$ 22.00 | ** |
| Ullr Lodge | 769-7146 | \$ 48.00 | *** |
| Wildwood Inn | 923-3550 | \$ 42.00 | *** |
| Hotel Jerome | 920-1000 | \$ 295.00 | *** |
| Snowflake Inn | 925-3221 | \$ 59.00 | **** |

move mouse over link to see URL

Open "http://localhost/cgi-bin/panorama.cgi?Colorad...ayRecord~City=Aspen~Hotel=Ullr%20Lodge" in a new tab

Clicking on a link opens a new window displaying the detailed information for that window. See “[What Record Are We Talking About?](#)” on page 395 to learn how the [DisplayRecord](#) procedure selects and displays the specified record.

Colorado Hotel Information

http://localhost/cgi-bin/panorama.cgi?ColoradoHotels~DisplayRecord~City

Add/Update hotel information

- Update Add
- Hotel**
- City**
- Phone**
- Rate**
- Units**
- Stars**
 1 2 3 4 5

Traditional European style lodging in Aspen. Family owned and operated. Indoor and outdoor whirlpools. Sauna. Walking distance to Music Tent and Aspen Institute.

When a form is linked to from a table the server automatically embeds a record ID into the form so that when the Submit button is pressed Panorama will know what record to update. For more information on this record ID and the update process see “[WebFormToDatabase — Modifying an Existing Record with Embedded Record ID](#)” on page 477.

Splitting a Long Table into Multiple Pages

By default a web table displays all of the selected records in the database. If there are hundreds or thousands of records selected this can make the table very unwieldy (and slow to load). Fortunately the Panorama Enterprise Edition Server can automatically split a large table into manageable pages, with links so you can navigate to different sections of the table. Here's an example of a long table that has been split into multiple pages. A search for cities containing **boulder** in the name has turned up 38 matching records. This table has been configured to display a maximum of 15 records so the table is split into three separate pages.

1 2 3 [NEXT](#)

| Name | City | Scale | Era | Railroad |
|-----------------------------------|------------------|-------|------------|-------------------------------|
| Joseph Irwin | Boulder, CO | N | Modern | Southern Pacific (SP) |
| Harold Donnell | Boulder, CO | HO | Transition | Southern Pacific (SP) |
| Dale McGuire | Boulder, CO | HO | Transition | Santa Fe (ATSF) |
| Craig Ott | Boulder, CO | HO | Transition | New York Central (NYC) |
| Harry Dickson | Boulder, CO | HO | Transition | Conrail |
| Peter Rodriguez | Boulder, CO | HO | Steam | Norfolk Southern (NS) |
| Lawrence Goodrich | Boulder, CO | G | Transition | Santa Fe (ATSF) |
| Marcus Silva | Boulder, CO | HO | Transition | Illinois Central (IC) |
| Michael Allen | Boulder, CO | G | Transition | Union Pacific (UP) |
| Joseph Cohen | Boulder, CO | O | Modern | Chicago & North Western (CNW) |
| Mark Fong | Boulder, CO | HO | Transition | Southern |
| George Derewlanko | Boulder, CO | HO | Transition | Nickel Plate (NKP) |
| Alfred Lamb | Boulder City, NV | N | Transition | Conrail |
| Douglas Dockery | Boulder, CO | HO | Transition | Illinois Central (IC) |
| Steven Hagen | Boulder, CO | HO | Modern | Southern Pacific (SP) |

1 2 3 [NEXT](#)

The Panorama server has automatically generated a “navigation bar” at the top and bottom of the table that allows you skip from page to page.



The options below control the maximum table size and the generation of a navigation bar. In most cases you can simply set the `recordsperpage` option and omit the rest of the options, simply using the default values.

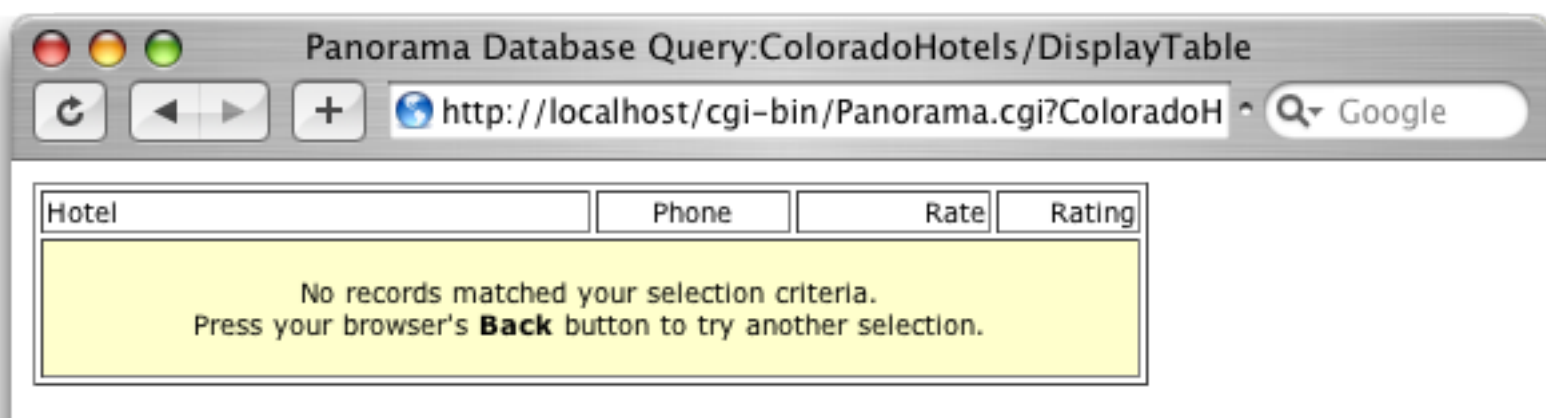
| Option | Default | Description |
|---------------------------------------|--|---|
| <code>recordsperpage=num</code> | | <p>This option specifies the maximum number of records that will be displayed in a single page. You can either type in a value or select from the pop-up menu.</p> <p>If this option is left blank then the Panorama Server will not automatically split the table into multiple pages and the rest of the options below will be ignored (because no page navigation bar will be generated).</p> <p>Note: If you specify a maximum number of records we highly recommend that you also set up a specific sort order for this table (see “Table Sort Order” on page 420). This will make sure that the same order is used for each page when navigating from page to page to page.</p> |
| <code>pagenumber=num</code> | 1 | This option specifies the page of the table to be displayed. Usually you should simply omit this option so that the first page is displayed. |
| <code>previouspagecaption=text</code> | <code>PREVIOUS</code> | The first and last links in the page navigation bar are usually PREVIOUS and NEXT , displayed in bold. Use this option to change the text of the link to the previous page. For example, if your web site is in Italian you might change this caption to <code>PRECEDENTE</code> . You can put any HTML tags you want into the caption. If you want to display an image for this link (for example a back arrow) simply use an <code></code> tag for the caption. |
| <code>nextpagecaption=text</code> | <code>NEXT</code> | The first and last links in the page navigation bar are usually PREVIOUS and NEXT , displayed in bold. Use this option to change the text of the link to the next page. For example, if your web site is in Italian you might change this caption to <code>SUCCESSIVO</code> . You can put any HTML tags you want into the caption. If you want to display an image for this link (for example a back arrow) simply use an <code></code> tag for the caption. |
| <code>pagenavfont=fontname</code> | | This option specifies the font to be used for the page navigation bar. (If left blank the browsers default font will be used.) On your own computer you can use any font you want because you have complete control over what fonts are installed and available for use. A web page, however, is the opposite situation since you have zero control over what fonts may be available when the page is viewed. Because of this it is usually best to pick from a restricted subset of fonts that are almost universally available. The fonts that are most commonly available are: <i>Arial</i> , <i>Comic Sans MS</i> , <i>Courier</i> , <i>Georgia</i> , <i>Helvetica</i> , <i>Times</i> and <i>Verdana</i> . |
| <code>pagenavfontsize=1..7</code> | | This option specifies the size of the text to be used for the page navigation bar. But take note — the size is not specified in pixels. Instead it is specified using a special HTML text size specification. Absolute values (1-7) specify a fixed font size from extremely small (1) to huge (7). Negative values specify sizes smaller than the text size in the rest of the page, for example -2 is two sizes smaller than normal font. Positive values specify sizes larger than the text size in the rest of the page, for example +2 is two sizes larger than normal font. |

| Option | Default | Description |
|------------------------|--|---|
| pagenavfooter=template | <pre><p><center> <pagenavlinks> </center><p></pre> | <p>The page navigation bar is usually centered below the bottom of the table. However you can move the navigation bar to the top of the table, or include the navigation bar at both the top and bottom of the table. You can also customize the HTML tags that are used to separate the page navigation bar from the table and the rest of the page.</p> <p>The header and/or footer can include any HTML tags you want. Whatever you type in must, however, include the special tag <code><pagenavlinks></code> (this must be all in lower case). When the page is displayed the <code><pagenavlinks></code> tag will be replaced with the actual tags and text for page navigation bar. (If the <code><pagenavlinks></code> tag is missing then the navigation bar will be missing also!)</p> <p>If both the header and footer are left blank then a default footer will be used. This default centers the page navigation bar below the table. If either the header or footer is filled in then the default won't be used. So if you want the navigation bar to appear above the table but not below, simply fill in the Page Navigation Header but leave the Page Navigation Footer blank.</p> |
| pagenavheader=template | | See pagenavfooter (above). |

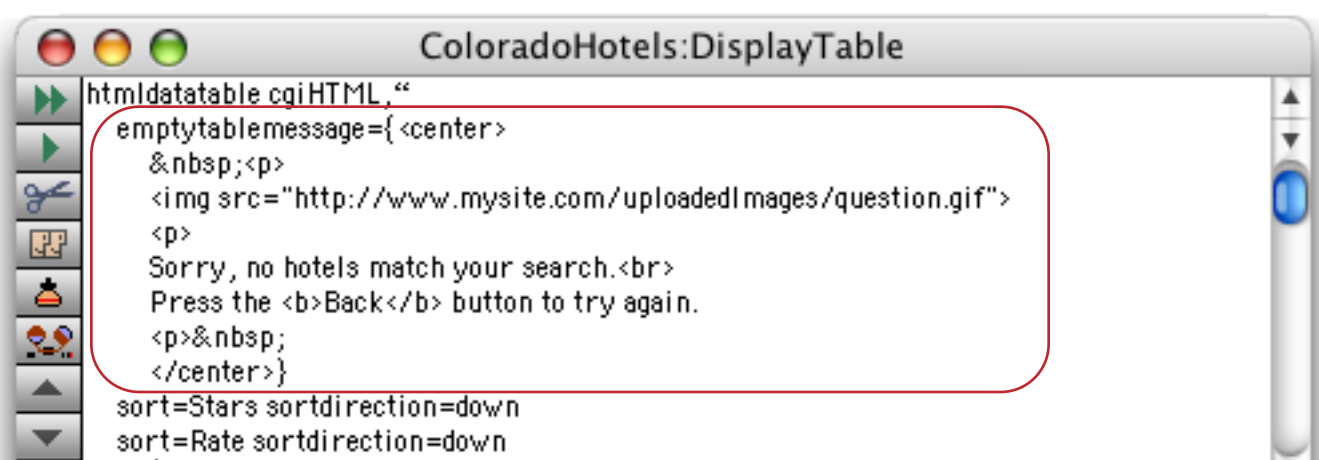
See “[The Multiple Page Table Dialog](#)” on page 304 for examples of different multiple page table settings.

Displaying an Empty Table

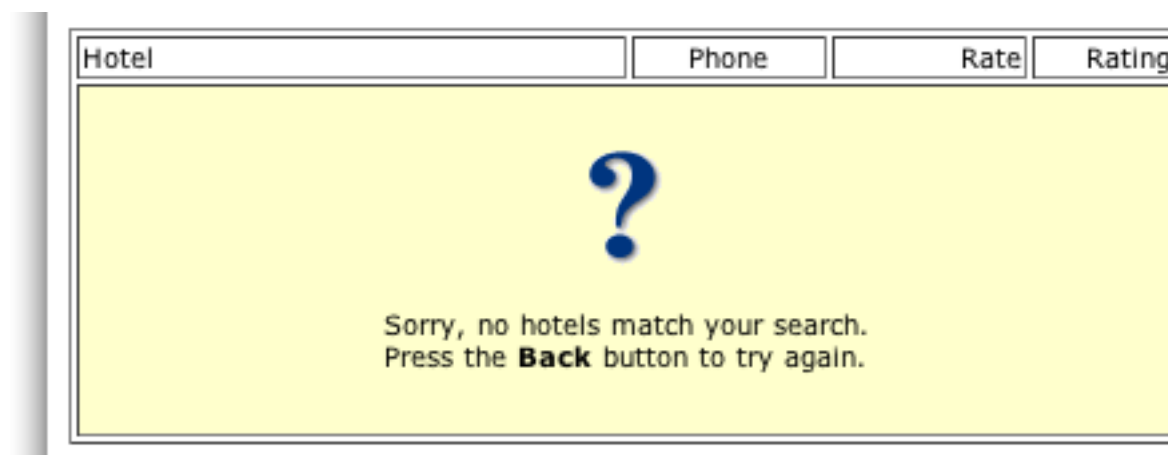
If the database selection is empty (no records match the selected criteria) the server will display an empty table, like this:



The message displayed in the empty table can be customized with the `emptytablemessage` option. You can include any text you like in this option, including HTML tags.



This example includes a GIF image as part of the empty table message.



You could even include a link to a new search form as part of the message.

Table HTML Layout

The font, border, spacing and background color options give you almost complete control over the appearance of the table. If you need even more control there are four “template” options that allow you to specify HTML templates for each component of the table.

| Option | Default Template | Description |
|--------------------|--|--|
| cell=template | <code><td><data></td></code> | This template controls the HTML rendered for each cell. <code><data></code> is replaced by the actual data generated by the formula for each column. For example to render each cell in italic the option would be <code>cell={<td><i><data></i></td>}</code> . |
| row=template | <code><tr><data></tr></code> | This template controls the HTML rendered for each row. <code><data></code> is replaced by the actual data cells for each row. For example to render each row so that the data is vertically centered the option would be <code>row={<tr valign=center><data></tr>}</code> . |
| table=template | <code><table><data></table></code> | This template controls the HTML rendered for the overall table. <code><data></code> is replaced by the actual data rows of the table. For example to render the table so that the total table width is 600 pixels the option would be <code>table={<table width=600><data></table>}</code> . |
| titlecell=template | <code><td><data></td></code> | This template controls the HTML rendered for each cell in the title row. <code><data></code> is replaced by the actual column names. For example to render each title cell using bold italic the option would be <code>cell={<td><i><data></i></td>}</code> . |

Consult an HTML book or web site for more information about the `<table>`, `<tr>`, `<td>` and `<th>` tags.

Modifying a Web Table Template On the Fly

So far this section has discussed building a table specification from scratch. It's also possible to grab a pre-built table template (made by the **Text Export** wizard, see “[Web Tables](#)” on page 277) and modify it before passing it on to the `htmldatatable` statement.

To retrieve a table template use the `getwebtabletemplate()` function, which has two parameters.

```
getwebtabletemplate(database,tablename)
```

The first parameter is the name of the database that contains the database, or `""` for the current database. The second parameter is the name of the template. This example displays the **List** table but using large **Comic Sans MS** font instead of whatever font was defined as part of the table itself.

```
local tbspec
tbspec=getwebtabletemplate("", "List")
htmldatatable cgiHTML, {font="Comic Sans ms" size=+1 }+tbspec
```

A procedure can get a list of the available table templates with the `listwebtemplates()` function. This function has one parameter, the name of the database (or "" for the current database).

Rendering a List (and lists, Formatted Tables, JavaScript)

While the `htmldatatable` statement makes tables, the `htmldatamerge` statement allows you to use a template to generate any kind of HTML you want. Typical applications include lists, formatted tables, and even JavaScript (usually to initialize a JavaScript array).

The `htmldatamerge` statement has three parameters.

```
htmldatamerge result,options,template
```

The first is parameter, *result*, is the name of the variable that will receive the HTML text created by this statement.

The second parameter, *options*, customizes the output in a manner similar to the options parameter of the `htmldatatable` statement. We'll discuss the options parameter in greater detail below, but for many applications this parameter can simply be left blank ("").

The final parameter is the *template*. This template controls how the data is placed into the HTML output by this statement. The template consists of a section of text with HTML tags and Panorama formulas embedded in it. The formulas are embedded in between curly braces { and }. You may embed as many formulas as you want - either one big formula or lots of little ones. However, each formula must produce text as an output, so any numbers must be converted to text with the `str()` or `pattern()` functions.

The `htmldatamerge` statement is best understood with examples. Our first example renders a simple list using `` tags.



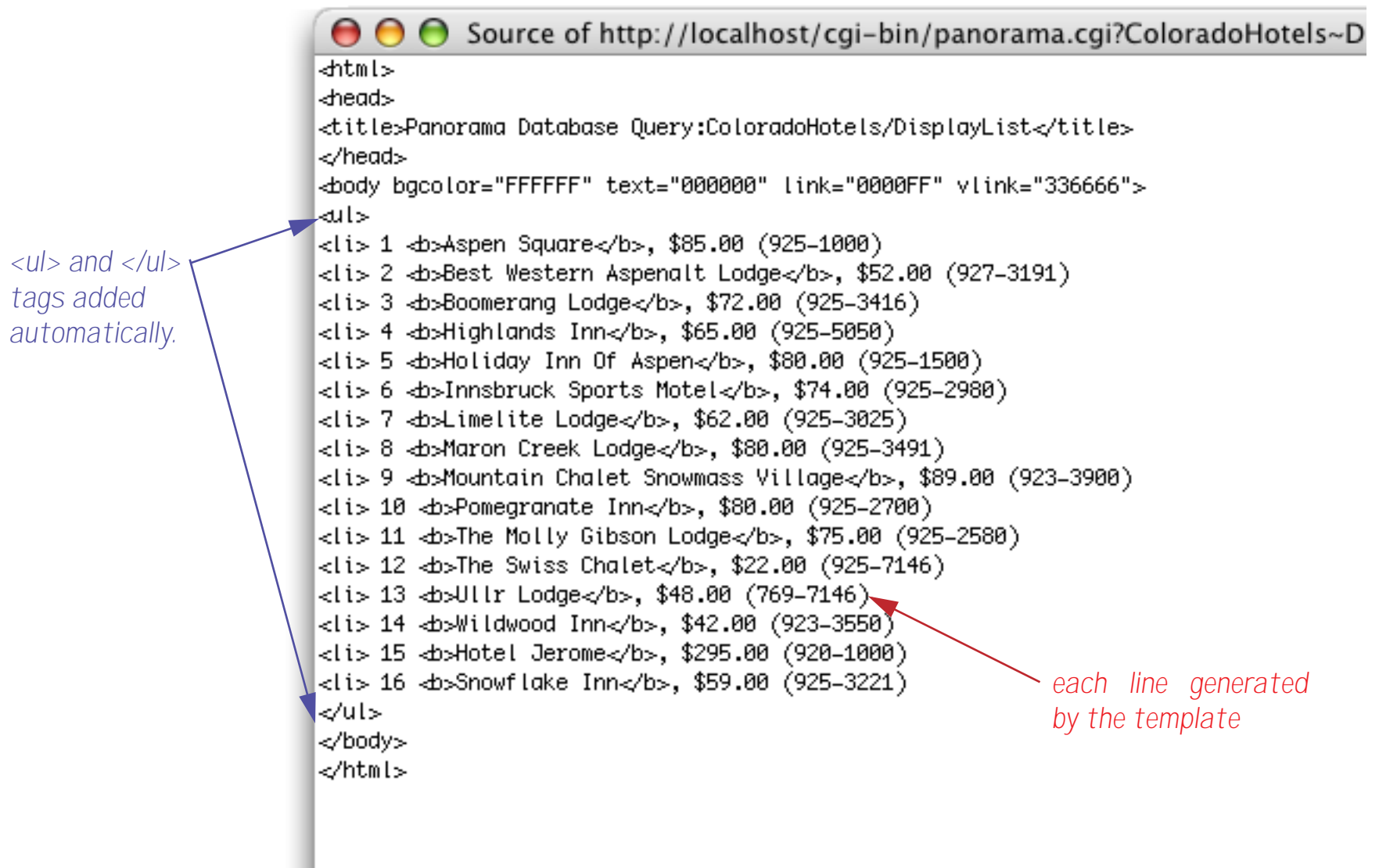
The layout has four formulas embedded in it:

| | |
|--------------------------------|---|
| {str(seq()) } | Merges an automatic sequence number: 1, 2, 3 etc. |
| {Hotel} | Merges the contents of the Hotel field |
| {pattern(Rate,"\$#,#")} } | Merges the Rate field, converted to text |
| {Phone} | Merges the Phone field |

The final result looks like this:



Using the browser's View Source command we can examine the actual HTML generated by the procedure. Notice that the `htmldatamerge` statement was smart enough to add `` and `` tags at the beginning and end of the list. If your layout begins with `` it will automatically add these tags. If your layout begins with `<tr>` it will automatically add `<table>` and `</table>` tags. (You can override these defaults and specify your own header and footer, see the next section.)

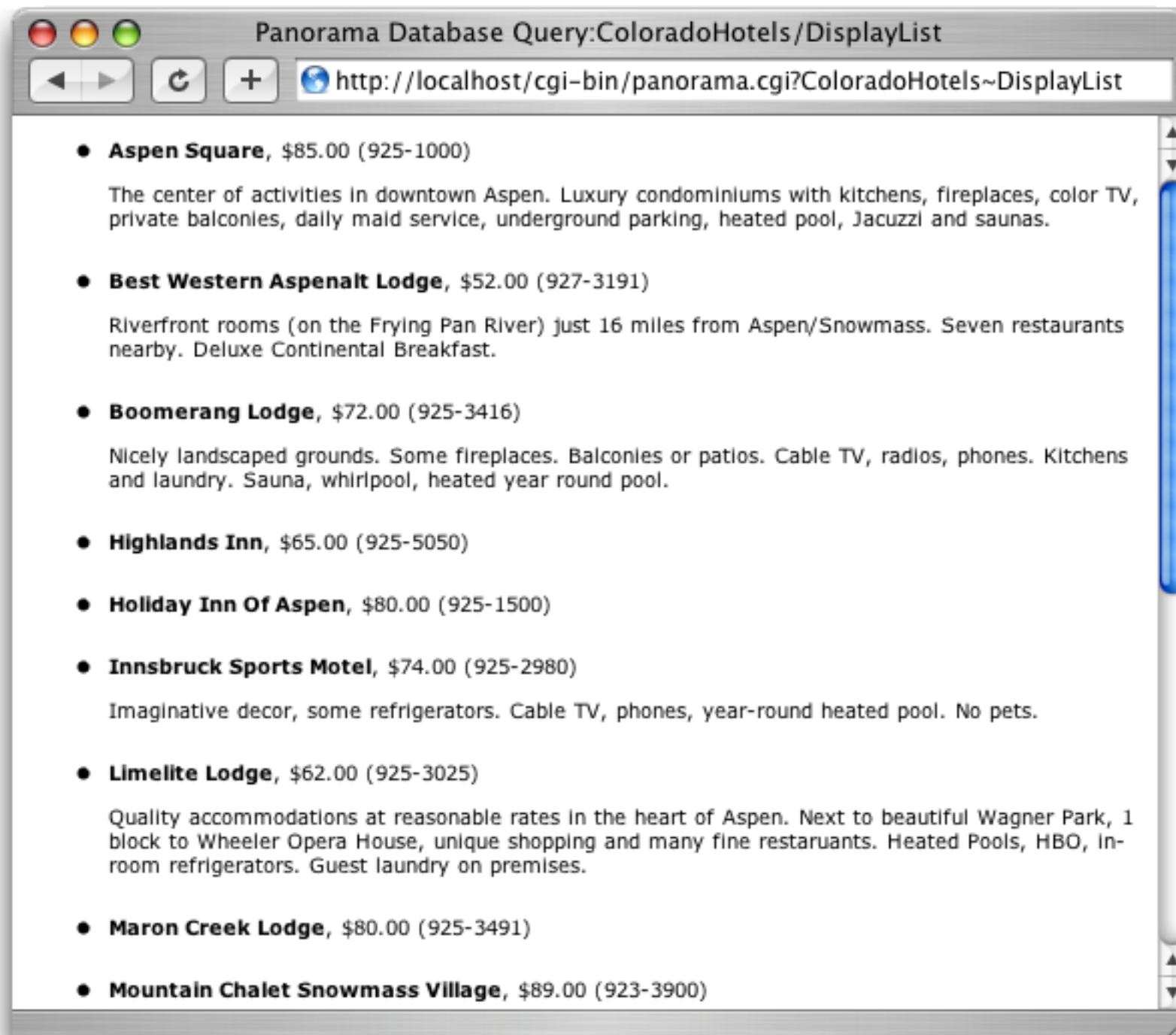


Here is a modified version of the previous procedure.



```
htmldatamerge cgiHTML,{},
  "<li><font face='verdana' size=-2> <b>{Hotel}</b>,<br>
  {pattern(Rate,'$#,# #')} ({Phone})<p>"+
  "{?(Description='',' ',Description+'<p>')}</font>"
```

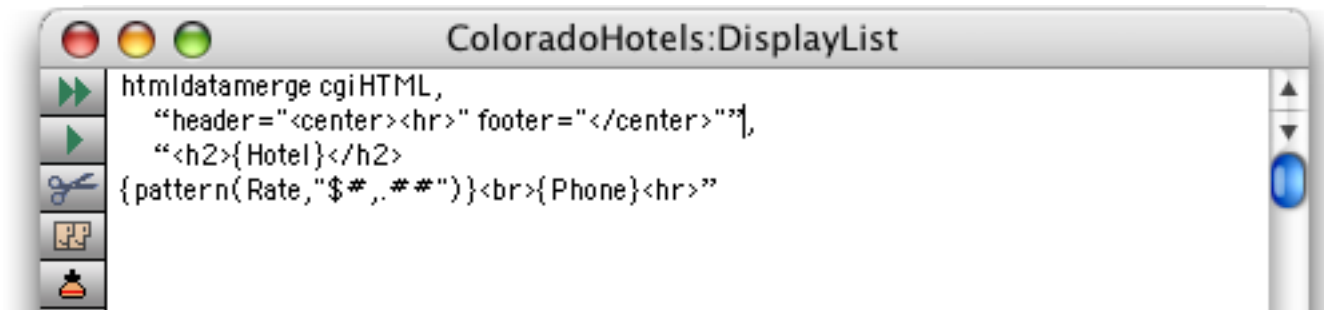
This procedure uses the `?()` function to merge in the description (if any). We've also changed the HTML tags to render the list using extra small Verdana font.



You can customize the HTML produced by the `htmldatamerge` statement with the `options` parameter. The basic options are `header=` and `footer=`. These options allow you to define additional HTML that will automatically be placed at the beginning and end of the generated HTML. (Of course you could also do this with a formula in your procedure.) For example, if you wanted to center the entire page, you could use

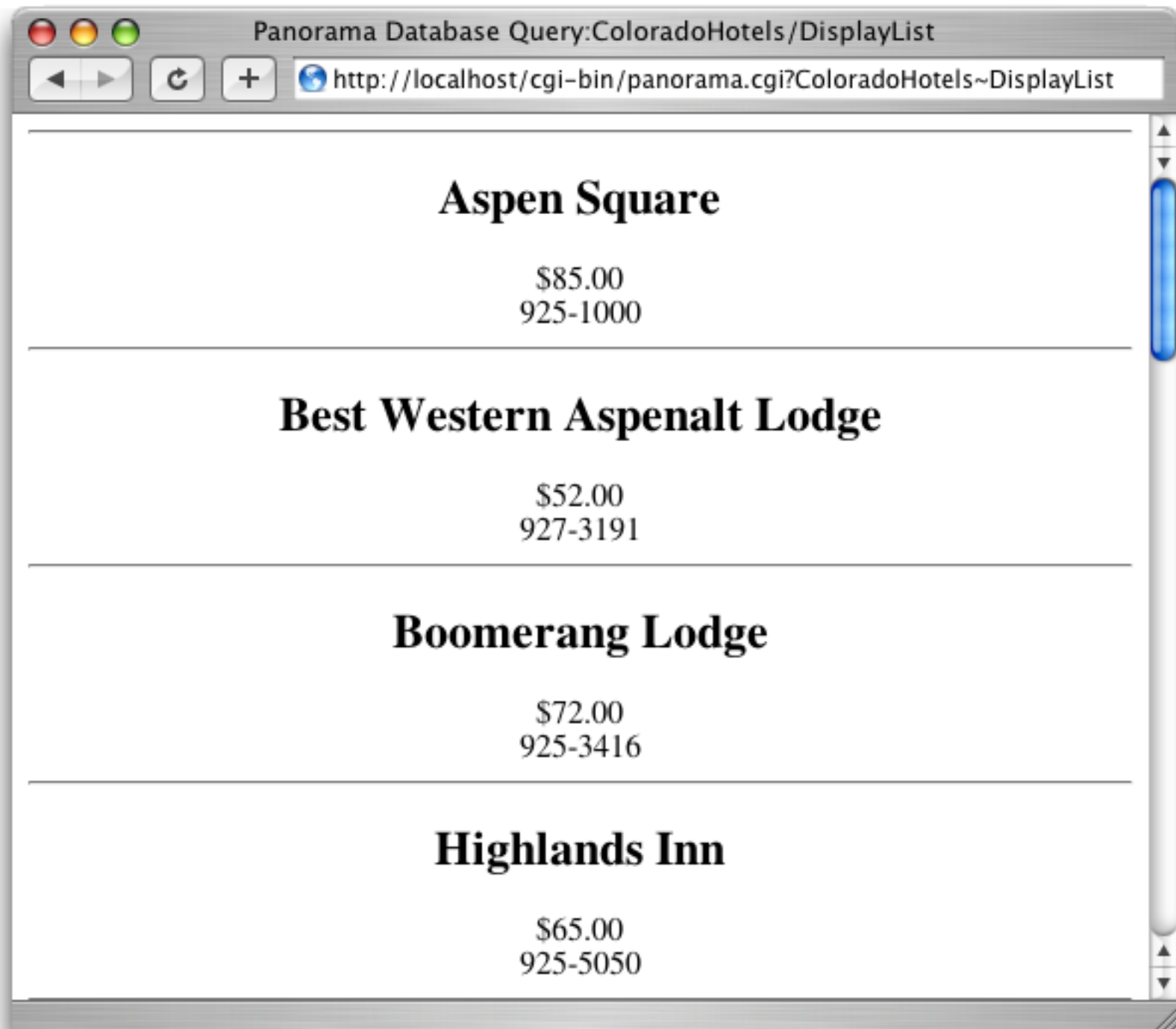
```
header="<center>"
footer="</center>"
```


Here is a modified version of our sample procedure that uses the `header=` and `footer=` options.



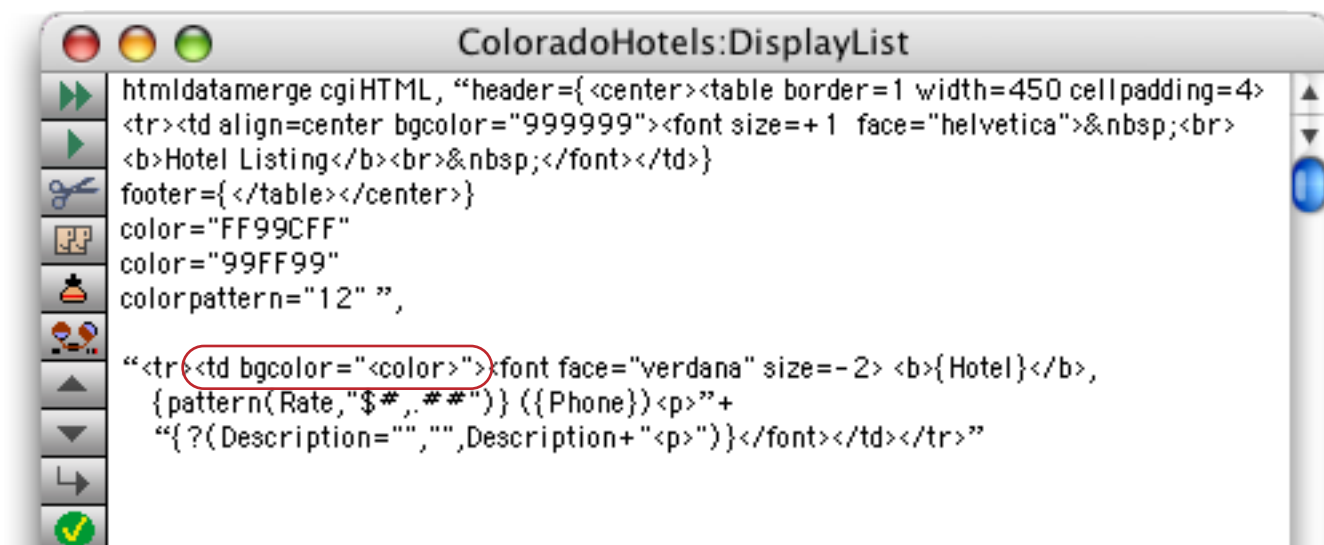
```
htmldatamerge cgiHTML,  
  "header="<center><hr>" footer="</center>"},  
  "<h2>{Hotel}</h2>  
{pattern(Rate,"$#,#")}<br>{Phone}<hr>"
```

And here's the output of this procedure.



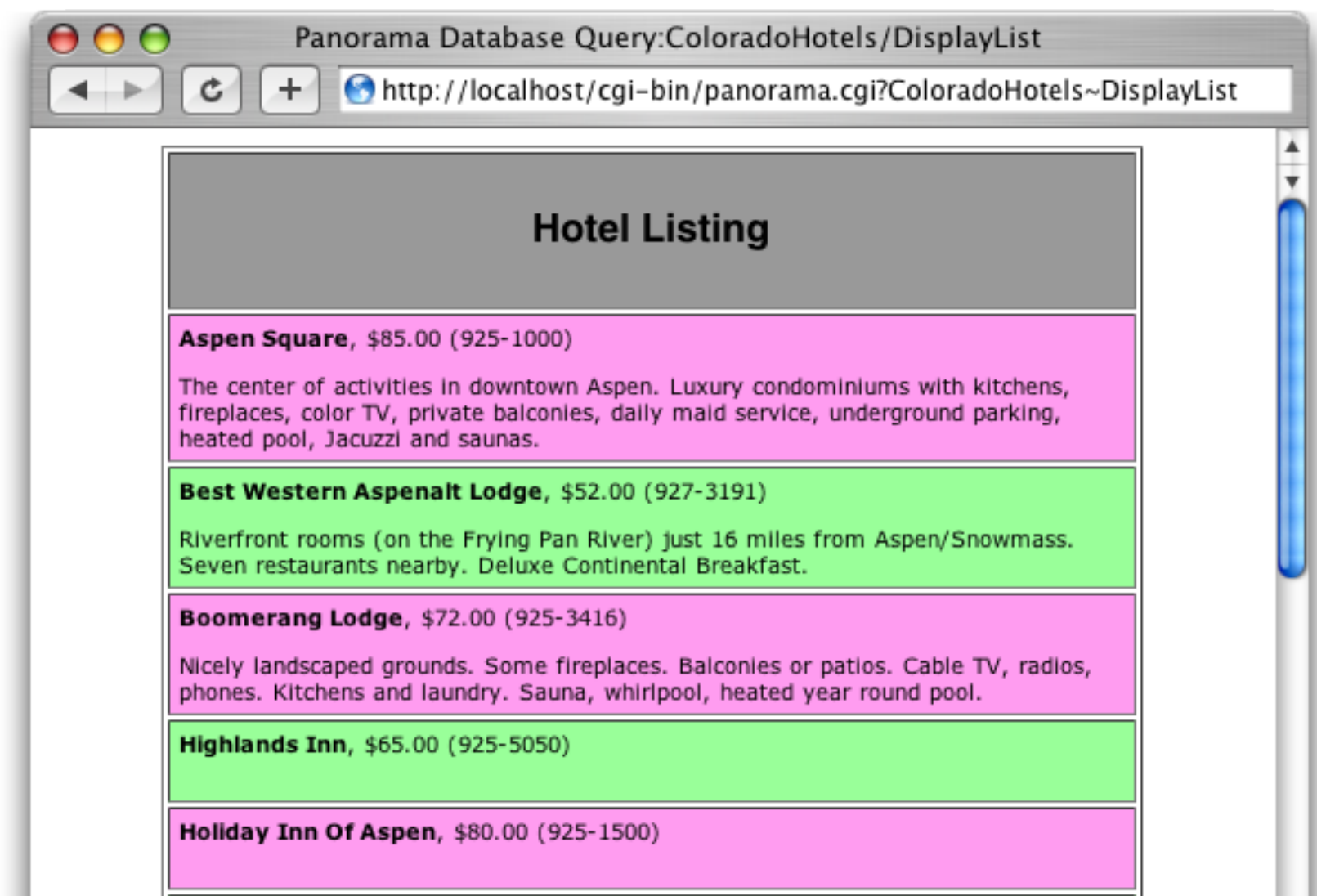
| |
|------------------------------------|
| Aspen Square |
| \$85.00 925-1000 |
| Best Western Aspenalt Lodge |
| \$52.00 927-3191 |
| Boomerang Lodge |
| \$72.00 925-3416 |
| Highlands Inn |
| \$65.00 925-5050 |

You can also specify `color=` and `colorpattern=` options. For a complete description of these options see “[Table Background Colors.](#)” on page 418. To use the color options you must embed the tag `<color>` somewhere in your template. This tag will be replaced by the actual rendered color specified by the options. However, the `<color>` tag must not be in one of the embedded formulas—it should only be in the HTML portion of the template. For example, the `<color>` tag can be embedded in a `<td>` tag to specify the background color of table cells.



```
htmlmerge cgiHTML, "header={<center><table border=1 width=450 cellpadding=4>
<tr><td align=center bgcolor="999999"><font size=+ 1 face="helvetica">&nbsp;<br>
<b>Hotel Listing</b><br>&nbsp;</font></td>
footer={</table></center>}
color="FF99CFF"
color="99FF99"
colorpattern="12" ",
"<tr><td bgcolor="<color>"><font face="verdana" size=- 2> <b>{Hotel}</b>,
{pattern(Rate,"$#,#")}{(Phone)}<p>" +
"{?(Description=","",Description+"<p>)}</font></td></tr>"
```

The result is a nice, formatted table listing each hotel, with a different background color for each hotel.



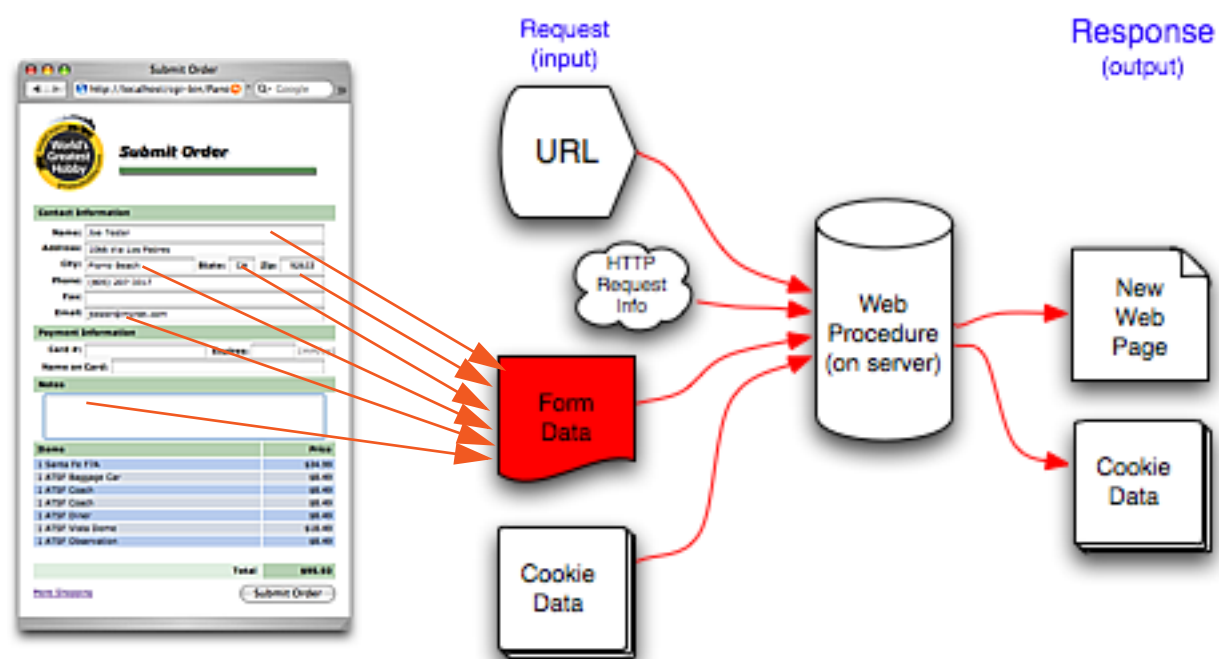
Linking a Table or List with a Detail Form

A list can include links to detail pages, just like a table. For more information on setting up this type of link see “[Linking Individual Table Rows to Detail Pages](#)” on page 421. Be sure to include `<link>` and `</link>` tags in the template for displaying the list.

Chapter 10: Processing Web Forms



The previous chapter dealt with outputting information from Panorama to a web browser. This chapter covers the flip side — processing data that has been input into a form on a web browser.



An HTML web form contains one or more items that the user can fill in. Once the items are filled in the user presses the **Submit** button to send the items to the server.

Each form item consists of two components: the item's name and the value. The name is set up in advance by the designer of the form. The name is invisible and can't be seen by the person filling in the form (the form designer can add a caption to identify each item for the person filling in the form, but the caption doesn't have to match the actual item name. For example the form designer can call an item intended for inputting a phone number `ZrGGuXX` if they wish.)

The item's value is filled in by the user, either by typing it in, choosing from a pop-up menu or list, or checking a box or radio button.

When working with Panorama Enterprise Server the easiest way to set up web forms is to create them with Panorama's graphics editor (see "[Web Forms](#)" on page 231). Panorama will take care of setting up all of the HTML tags for the form, freeing you to concentrate on the design itself. However it is possible to hand code the form in HTML yourself. If you want to do that, see "[Rendering Using an External Text File as a Template](#)" on page 483 for some special tips on the formatting needed for Panorama.

Accessing Web Form Information

The first step in processing the information from a web form is gathering the information into Panorama fields and/or variables. Web form items can be processed individually or in batches.

What Form was Submitted? (The WebFormName() Function)

It's possible to write a single web procedure that is assigned to more than one web form. In that case the procedure will need to know what web form is being submitted. If the web form was created in Panorama (see "[Converting a Panorama Form into a Web Form](#)" on page 231) then the procedure can find out what form was submitted using the `webformname()` function.

```
if webformname() = "New"
    ...
endif
if webformname() = "Renewal"
    ...
endif
```

If you did not use Panorama to create your forms you'll need to come up with some other method to identify which form is being submitted, perhaps with a hidden field (see below).

What Items were Submitted? (The WebFormItems() Function)

The `webformitems()` function returns a list of the names of all of the form items submitted to the server. The list returned is a carriage return separated array. Here is a code fragment that calculates how many form items were submitted.

```
local itemCount,itemNames
itemNames=webformitems()
itemCount=arraySize(itemNames,cr())
```

You probably won't use the `webformitems()` function much because you usually design the forms yourself and already know the names of the form items. However, it can be very useful if you write subroutines that are used to process more than one web form.

Hidden Items

In addition to normal visible data values (text editing cells, checkboxes, radio buttons, etc.) web browsers also support hidden values. These values are generated when the web form is displayed, then stored invisibly within the web form and submitted with the other data values when the **Submit** button is pressed. Essentially these hidden values allow the server to pass values to itself, allowing it to remember a value or calculation from a previous interaction with the server. For example a hidden value could contain the time the form was displayed, allowing the server to calculate how long it took you to fill in the form.

If you are building your web form in Panorama you can use the **Hidden Fields** command in the **Web** sub-menu of the **Setup** menu to create one or more hidden form items (see "[Hidden Data](#)" on page 271). If you are building your web form using some other tool you'll need to use an `<INPUT>` tag for each hidden item. The type of the tag must be set to hidden, and it must include the name and value of the item. Here are a couple of examples.

```
<INPUT TYPE=hidden NAME=recipient VALUE="mjd@help.com">
<INPUT TYPE=hidden NAME=subject VALUE="Feedback on your help system">
```

There are three special functions for finding out what visible and hidden items were submitted in the form.

| | |
|------------------------------------|--|
| <code>webformvisibleitems()</code> | This function returns a carriage return delimited list of the visible items in the form submitted to the server. |
| <code>webformhiddenitems()</code> | This function returns a carriage return delimited list of the hidden items in the form submitted to the server. (However, it does not include several special hidden items that Panorama itself generates, only hidden items that you have created.) |
| <code>webformallitems()</code> | This function returns a carriage return delimited list of the all items in the form submitted to the server. This includes visible items, hidden items you have created, and special hidden items that Panorama itself generates. |

Note: Except for `webformallitems()`, these functions only work when using web forms that have been created with Panorama. They will not work with forms you have hand-coded yourself in HTML.

Accessing Form Item Values in a Formula

The `webformitemvalue()` function gets the value of any form item. It has one parameter, the name of the form item.

```
webformitemvalue(item)
```

This function can be used in any Panorama formula in your web procedure.

```
cgiHTML=webformitemvalue("Name")+ " lives in "+
webformitemvalue("City")+ ", "+webformitemvalue("State")
```

The output of this function is always text. If you need to use the value as a number you must convert it to a number with the `val()` function.

Assigning a Web Form Item Value into a Panorama Field or Variable

The `webformitem` statement gets the value of any form item, then assigns that value to a variable or a database field.

```
webformitem item,fieldorvariable
```

The *name* parameter is the name of the form item you want. The *fieldorvariable* parameter is the name of a field or variable where the value of this item should be stored. For example here is a simple procedure for adding a new record to a web based guestbook.

```
addrecord
webformitem "Name",Name
webformitem "City",City
webformitem "State",State
webformitem "Country",Country
webformitem "Email",Email
webformitem "Name",Name
```

Note: The `webformitem` statement will automatically convert data into the proper format for storage into a text, numeric or date field. If the data submitted in the form is not the proper format (for example if the user types `tree` for a number or date field) `webformitem` will simply leave the field or variable empty.

Assigning Multiple Items into Multiple Fields and/or Variables

The previous example used seven lines to add a new record to a database from the data in a web form. With the `grabwebformitems` statement this can be reduced to only two lines:

```
addrecord
grabwebformitems "assign=FIELDS"
```

This statement will automatically take all of the visible form item fields and assign them to the corresponding database fields, just as if the `webformitem` statement had been used over and over again. (For this to work, the web form item names have to be the same as the database field names.)

The `grabwebformitems` statement has one parameter, a list of options in `option=value` format. There are three possible options: `items`, `assign` and `variables`.

```
grabwebformitems "items=type/list assign=type variables=type"
```

The `items` option specifies what web form items to “grab” and assign to fields and/or variables. If this option not specified then all visible items will be processed. You can specify a list of items (comma separated), or you can specify `"visible"`, `"hidden"` or `"all"`. Here are some examples:

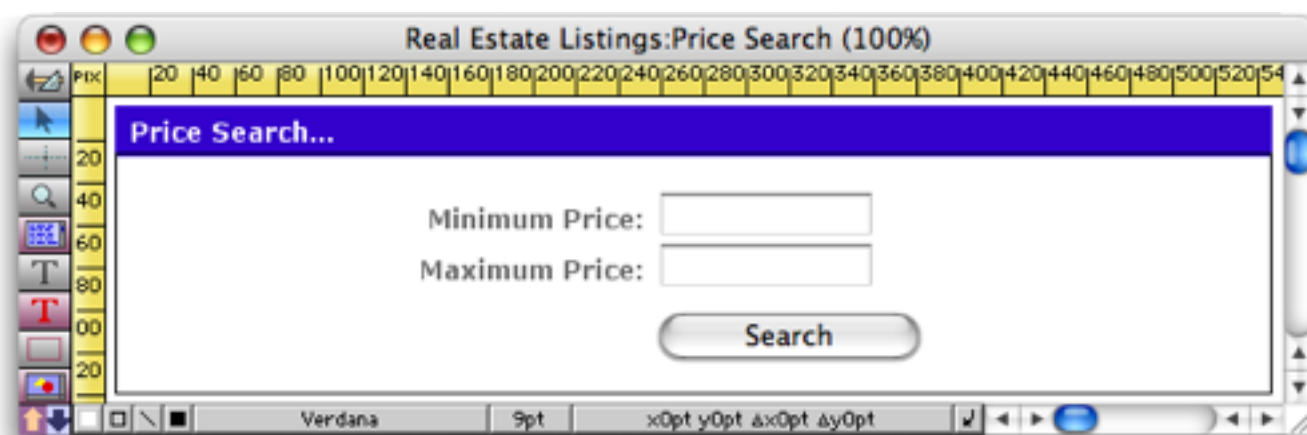
```
grabwebformitems "items="Name, City, State, Country, Email, Name" "  
grabwebformitems "items=Visible"  
grabwebformitems "items=all"
```

The `assign` option specifies whether to grab the values into fields, into variables, or both. There are four possible choices: `fields`, `variables`, `noconflictvariables`, or `any`.

| Option | Description |
|----------------------------------|--|
| <code>fields</code> | If this option is used the web form items will be assigned into corresponding database fields. If there is no corresponding database field an error will occur (which can be trapped with <code>if error</code>). |
| <code>variables</code> | If this option is used the statement will automatically create variables for each of the specified web form items and then assign the item values to the variables. If a web form item has the same name as a database field a variable will still be created, which may make it impossible to access the database field. |
| <code>noconflictvariables</code> | If this option is used the statement will automatically create variables for each of the specified web form items and then assign the item values to the variables. If a web form item has the same name as a database field a variable an error will occur (which can be trapped with <code>if error</code>). This is the default option if no assign option is specified. |
| <code>any</code> | If this option is used any web form items that correspond to database fields will be assigned to the those fields. Any remaining web form items will be assigned to variables, which will be automatically created if necessary. |

The `variables` option specifies what type of variable should be created: `local` (the default), `fileglobal`, `global` (not usually recommended) or `permanent`.

To illustrate the `grabwebformitems` statement we'll start with this web form. The form has two items, `minPrice` and `maxPrice`.



Here is a three line procedure to select the requested property listings. The `grabformwebitems` statement will automatically create fileglobal variables for `minPrice` and `maxPrice`.

```
grabformwebitems "items=visible assign=variables variables=fileglobal"
webselect Asking >= val("<minPrice>") and Asking <= val("<maxPrice>")
cgiHTML=renderwebtable("ListingTable")
```

Validating Data Entry (Error Checking)

In a perfect world users will always submit accurate data the first time. In the real world that often doesn't happen. In many applications you'll want to check to make sure the submitted data is valid before you use it. Panorama Enterprise has some special tools to help you do that.

To check the validity of a particular form item use the `webformitemcheck` statement. This statement has three parameters.

```
webformitemcheck item,message,true-false formula
```

The first parameter, `item`, is the name of the web form item you want to check.

The second parameter, `message`, is text that you write to explain to the user what the problem is. Usually this would be a short phrase or sentence, for example "The card number is invalid" or "Please enter both first and last names". Later you'll learn how this message can be displayed on the web page displayed to the user.

The final parameter is a true-false formula that decides whether the item is valid or not. The formula should be true if there is a problem, false if everything is ok. Within the formula you can use the `import()` function when you need the value of the item. (Of course you can also use the `webformitem()` function.)

Here's a simple example that checks to make sure that a name has been entered:

```
webformitemcheck "Name","Please enter a name.",import()==" "
```

Here's a more complicated example that checks to make sure that the name contains at least two words and doesn't contain any non alphabetic characters (except for space):

```
webformitemcheck "Name",
  "Please enter a valid name.",
  wordcount(import())<2 or rangenotmatch(import(),"AZaz ")
```

How carefully the data is validated is up to you. The procedure can use multiple `webformitemcheck` statement to check multiple web form items. (You should not, however, use more than one `webformitemcheck` statement for a single web form item. If you do the `webformerrors()` function (see the next section) may not list all of the errors.)

Generating a List of Data Entry Errors

A particular form can have multiple data entry problems. To generate a list of all the problems, use the `webformerrors()` function. Here's an example.

```
webformitemcheck "Name",
  "Please enter a valid name.",
  wordcount(import())<2 or rangenotmatch(import(),"AZaz ")
webformitemcheck "Card",
  "Please enter a valid credit card number.",
  cardvalidate(import())=false()
webformitemcheck "CardYear",
  "Your credit card is expired or has an invalid expiration date.",
  cardexpirevalidate(webformitemvalue("CardMonth"),import())=false()
cgiHTML=webformerrors("<ul>")
if cgiHTML<>" rtn endif
...
... continue with data entry procedure
...
```

If there are problems with the credit card this procedure will display a list of errors that looks something like this:

- Please enter a valid credit card number.
- Your credit card is expired or has an invalid expiration date.

The `webformerrors()` function has one parameter, a pattern that specifies how the list of errors should be formatted.

```
webformerrors(pattern)
```

If the pattern is empty, the list will be formatted as a carriage return separated array (one line per error).

If the pattern is "``", the list will be formatted as an HTML unnumbered list (each item preceded with a `` tag). If you want a numbered list use "``".

If the pattern is "`<table>`", the list will be formatted as an HTML table. You can include options inside the table tag, for example "`<table border=0 cellpadding=4>`".

For even more control you can specify separate `header=`, `footer=`, `prefix=` and `suffix=` tags. The `prefix` and `suffix` will be added to each individual error message. The `header` and `footer` tags will be added at the top and bottom of the entire list. Here's an example for creating a custom html table:

```
webformerrors(|||header="<table border=0>" footer="</table>" prefix="<tr><td><font  
face="Verdana" size=-1>" suffix="</font></td></tr>"|||)
```

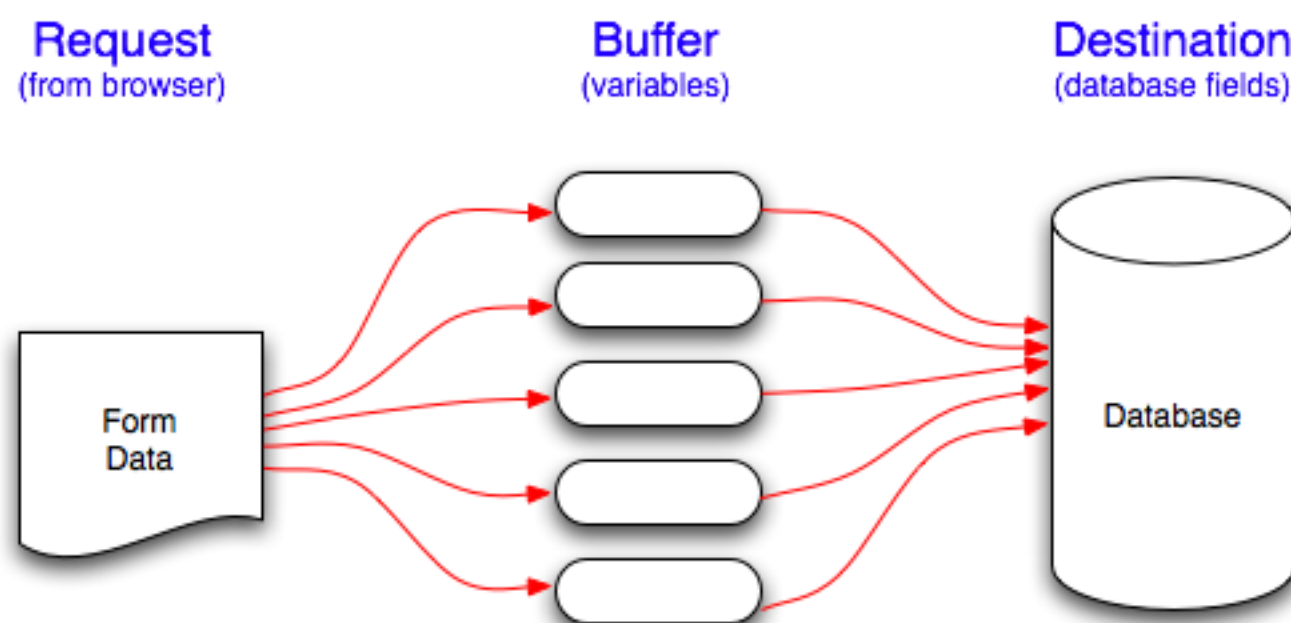
You'll probably want to give the user a bit more information about what to do next. Here's one way to do that.

```
cgiHTML=sandwich("Your form entries are incomplete or invalid:",  
webformerrors("<ul>"),"Press the browser's BACK button to correct these problems.")
```

A better method is to re-display the original form with the error(s) displayed directly on the form. A technique for doing this is covered in a few pages (see "[Identifying Items with Data Validation Problems](#)" on page 442).

Buffered Data Entry

So far in this chapter the techniques demonstrated have transferred data from the input form directly into database fields. Another technique is to transfer the data first to a set of variables, then to the fields.



This buffered technique makes it easier to validate and pre-process the data before permanently storing it in the database, and also enables a superior method for handling data entry errors and problems.

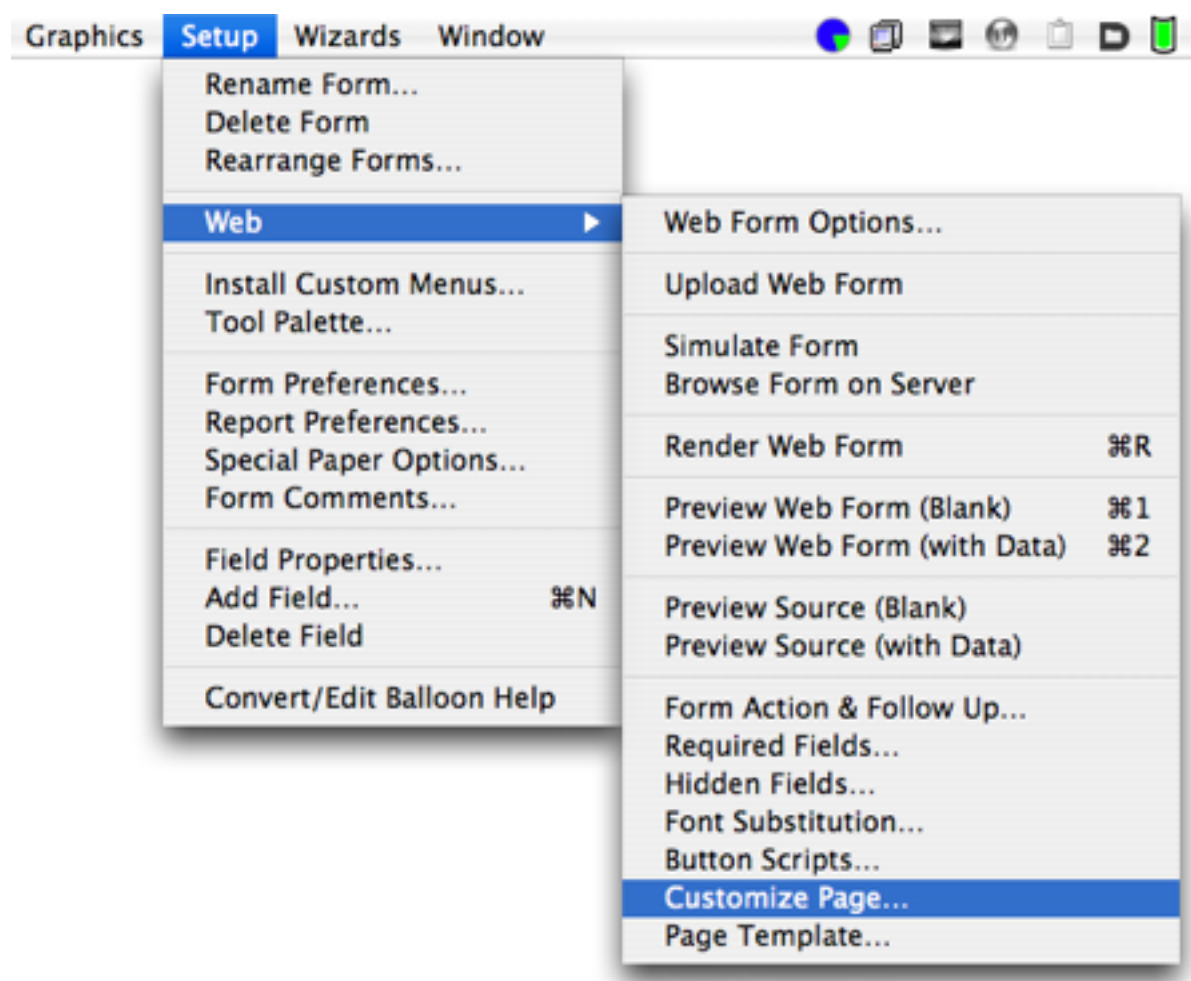
You can implement the buffered technique manually by defining the variables yourself (using `fileglobal`, `local`, etc.) and then using individual assignments with the `webformitemvalue()` function to copy the form data into the variables. I don't know about you, but that sounds like way too much work! Fortunately Panorama has some special statements that do most of the work for you. The first of these is the `grabwebformitems` statement. This statement automatically grabs a bunch of form items, creates variables for them, and copies the item values into the variables. To automatically copy all visible form items into a buffer of variables all that is needed is this one simple line of code:

```
grabwebformitems ""
```

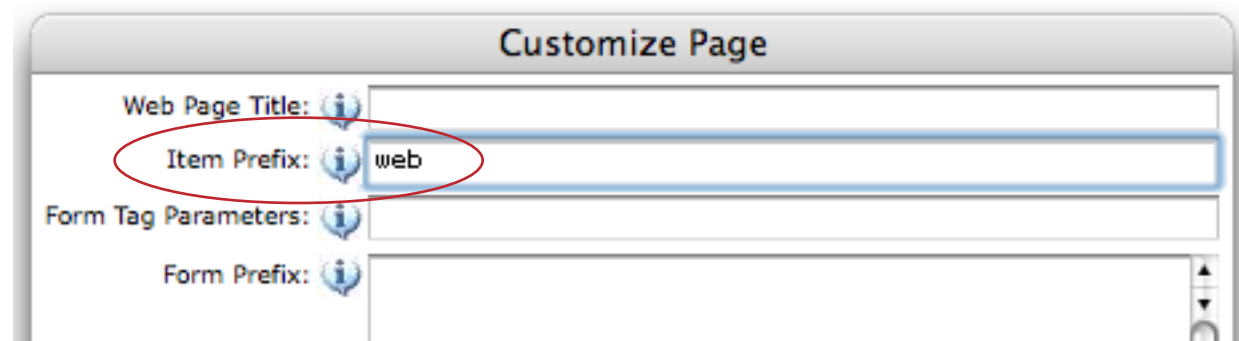
The variables created by this statement will have the same names as the input form items. For example, if your input form has five items: Name, Address, City, State and Zip, the statement `grabwebformitems ""` will essentially be the same as this short program:

```
fileglobal Name,Address,City,State,Zip
Name=webformitem("Name")
Address=webformitem("Address")
City=webformitem("City")
State=webformitem("State")
Zip=webformitem("Zip")
```

If your database already has fields named `Name`, `Address`, `City`, `State` and `Zip`, then these variables will conflict with the database fields. To eliminate this conflict you'll need to add a prefix to each of the item names in your form. Fortunately you don't have to add this prefix manually to each item — Panorama can do this automatically when the form is rendered. To do this choose `Customize Page` from the `Web` submenu of the `Setup` menu.



The second item in this dialog is the prefix that will be added to each visible item in the HTML version of the form. If the prefix is set to `web` as shown below then `Name` will become `webName` in the HTML version, `City` will become `webCity`, etc.



Be sure to re-render the form after changing the prefix.

The `grabwebformitems` statement has an optional parameter that allows you to customize the action of this statement. For most applications you can simply leave this blank, as shown in the examples in this section. See the Programming Reference wizard to learn more about these options and how they can be used.

Copying from the Variable Buffer into the Database

The next step is to validate the data and do any pre-processing required. For now we'll skip over that step and proceed directly to the last step, copying the data from the variables into the database itself. Again, this could be done manually with a series of assignment statements, but Panorama has a special statement that will do this for you: `stashwebformitems`. Here is a complete program that copies the form input into variables, then into a new record in the database.

```
grabwebformitems ""
addrecord
stashwebformitems
```

Of course we've already seen an even simpler program that does the same thing without the variable buffer (see "[Assigning Multiple Items into Multiple Fields and/or Variables](#)" on page 435):

```
addrecord
grabwebformitems "assign=FIELDS"
```

So why use a variable buffer if it's more complicated? Using a buffer makes data pre-processing and error checking much simpler, as you'll see in the following sections.

Data Pre-Processing in the Variable Buffer

Once the form input data is in the variable buffer you can easily work with it. Any changes you make in the variables will ultimately be reflected in the final data placed in the database itself. Consider the following form:

The screenshot shows a web browser window titled 'Real Estate Listings: Property Listing (100%)'. The browser's address bar shows a URL starting with 'http://'. The page content is a form titled 'Property Listing...'. The form contains the following fields and values:

- Address:
- City:
- Asking Price:
- Bedrooms:
- Bathrooms:
- Age:
- Garage:
- Stories:
- Air Conditioning:
- Fireplace:
- Pool:
- Agent:

A 'Submit' button is located at the bottom of the form. The browser's status bar at the bottom shows 'Helvetica 18pt xOpt yOpt axOpt ayOpt'.

This form has a dozen items. The item prefix has been set to `web`, so these items are `webAddress`, `webCity`, `webAsking`, `webBedrooms`, `webBathrooms`, `webAge`, `webGarage`, `webStories`, `webAir`, `webFireplace`, `webPool` and `webAgent`. The procedure below grabs the web input into variables, then makes sure that the address, city and agent name are properly capitalized. It also removes any non-numeric characters the user entered into the # of bathrooms and age items. The pre-processed data is then entered into the database.

```
grabwebformitems ""
webAddress=upperword(webAddress)
webCity=upperword(webCity)
webBathrooms=striptonum(webBathrooms)
webAge=striptonum(webAge)
webAgent=upperword(webAgent)
addrecord
stashwebformitems
```

You can put any kind of pre-processing you want in between the `grabwebformitems` and `stashwebformitems` statements.

Validating Data in the Variable Buffer

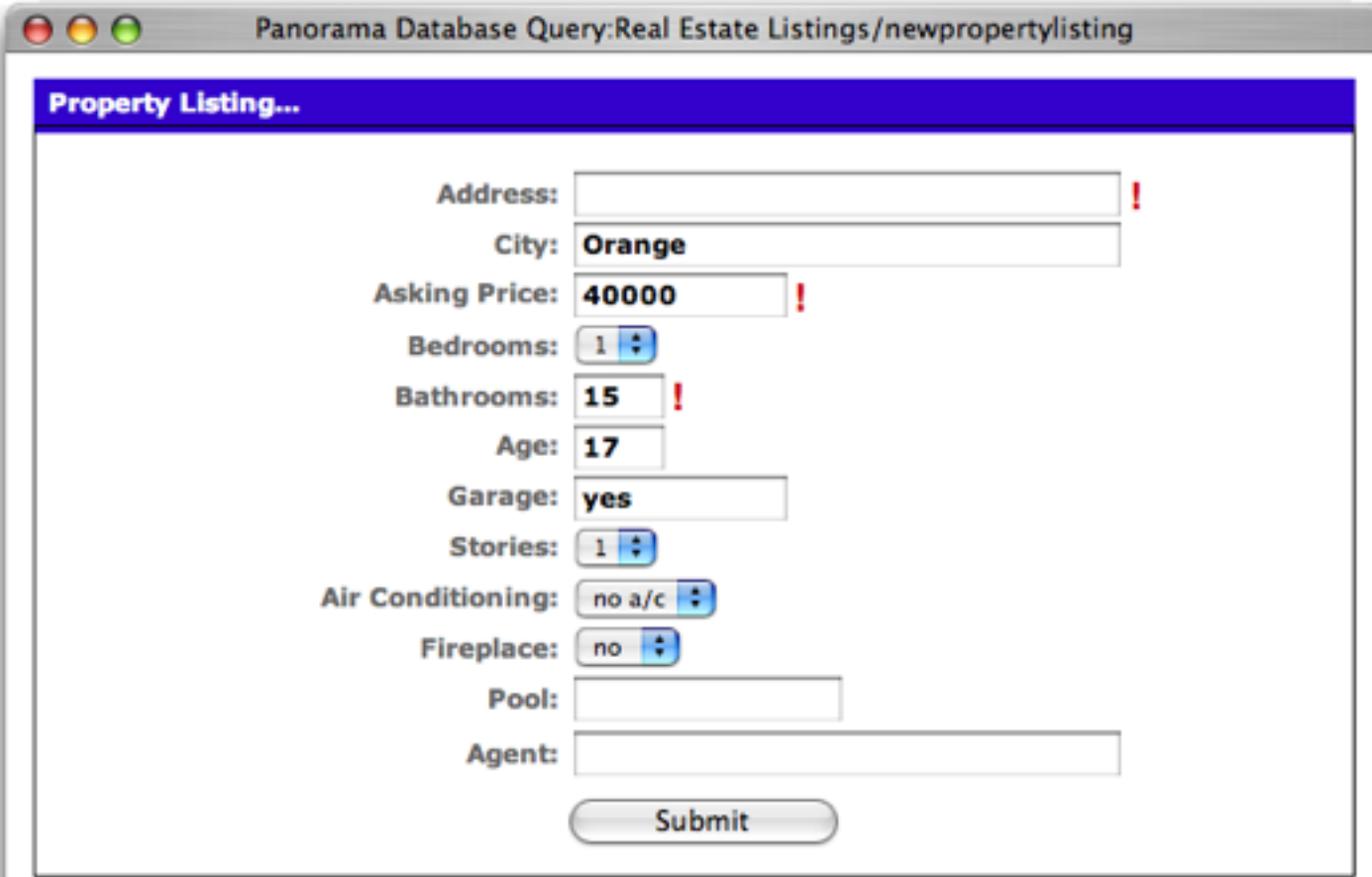
Data in the variable buffer can be validated with the `webformitemcheck` statement (see “[Validating Data Entry \(Error Checking\)](#)” on page 437). You can use the `import()` function in the formula, or just use the variable name. (Be sure to include the prefix in the item name.) Here’s an example.

```
webformitemcheck "webName",
  "Please enter a valid name.",
  wordcount(import())<2 or rangenotmatch(import(),"AZaz ")
webformitemcheck "webCard",
  "Please enter a valid credit card number.",
  cardvalidate(import())=false()
webformitemcheck "webCardYear",
  "Your credit card is expired or has an invalid expiration date.",
  cardexpirevalidate(webCardMonth),import())=false()
```

If there is an error the procedure can re-display the original form with the data the user entered. The user can then modify the data so that it will be acceptable and re-submit it. Here is the code that re-displays the form — this goes after the `webformitemcheck` statements and before the `stashwebformitems` statement.

```
if webformerrors("<")<>" "  
    retrywebform  
    rtn  
endif
```

Identifying Items with Data Validation Problems. Of course if you simply re-display the original form the user won't have any idea what the problem is or why he or she is being asked to re-enter the same data again. By slightly modifying the form we can alert the user to exactly what the problem is. Here's an example of what this will look like:



Property Listing...

Address: !

City:

Asking Price: !

Bedrooms:

Bathrooms: !

Age:

Garage:

Stories:

Air Conditioning:

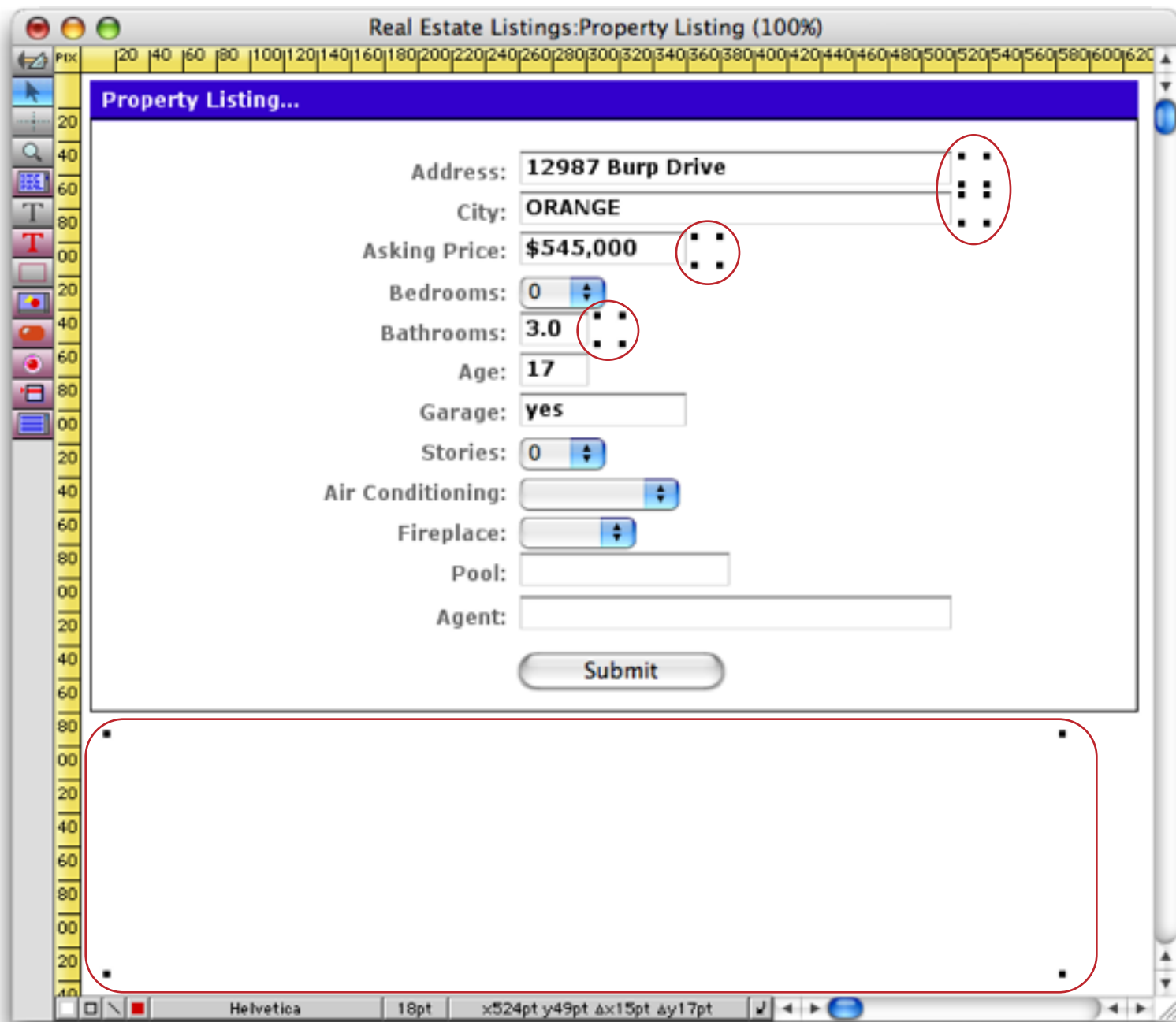
Fireplace:

Pool:

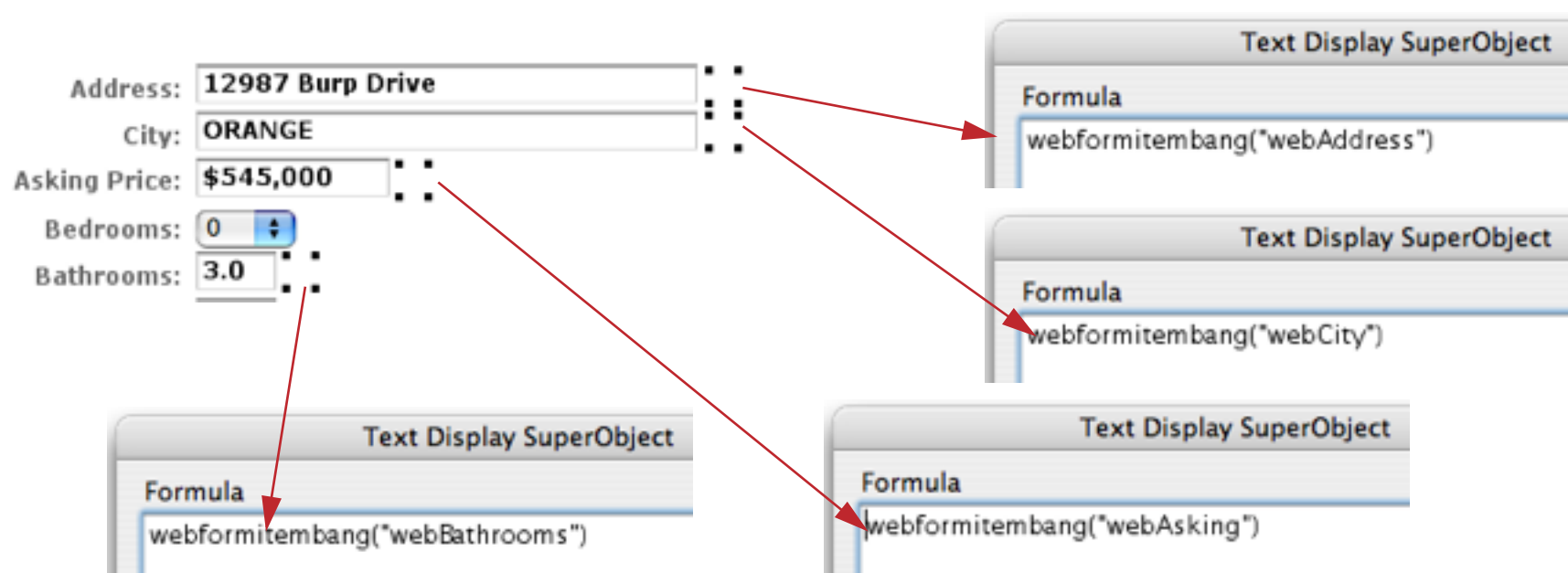
Agent:

- You must enter an address.
- The number of bathrooms must be from 1 to 9.
- You must enter a valid asking price for this property.

Here is the original Panorama form. As you can see, five objects were added to the form to display the error information. All five of these objects are Text Display SuperObjects.



The small objects to the right of the fields display a ! symbol if there is a problem with that field (in a moment we'll show you how to display a graphic instead of the ! symbol if you wish). To do this, each object contains a special function, `webformitembang()`, with the name of the corresponding item. (As you can see, the item name must contain the prefix, in this case `web`.)

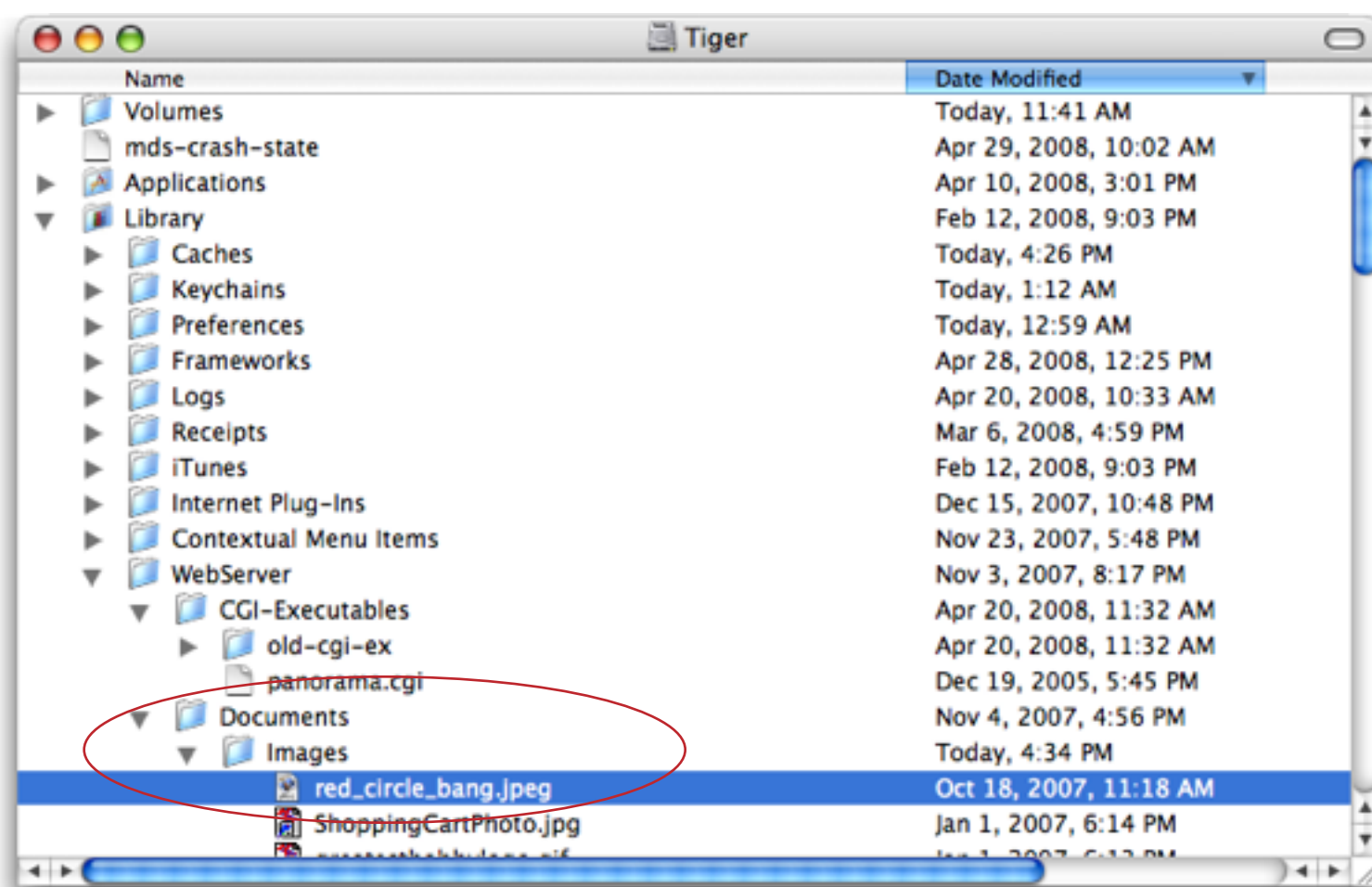


The `webformitembang()` normally doesn't generate anything, leaving the object invisible. But if the `webformitemcheck` statement has identified a problem with the corresponding item, the function will generate a **!** symbol (also called an exclamation point or bang symbol). You can use Panorama's font and color tools to customize the appearance of the **!** symbol. In this example we have set the color to red and the font to Helvetica 18.

Displaying an Image Instead of an Exclamation Point. Instead of an exclamation point you may want to display an image like this:



The first step is to obtain the image file in either .jpg or .gif format and copy it onto your web server. It needs to go somewhere within the `WebServer Documents` folder (which you'll find inside the `Library` folder on your systems primary hard drive). In this case an image called `red_circle_bang.jpeg` has been placed inside a subfolder (`Images`) of the `Documents` folder.



To use this image you need to add one line of code to your database.

```
webformitembang "/Images/red_circle_bang.jpeg"
```

This line can be placed just before the `retrywebform` statement, or it can be placed in the `.ServerInitialize` statement. Once used, it will apply to all of the forms in this database unless you use another `webformitembang` statement. You don't have to change the forms at all (unless you need to adjust the layout of the form objects to handle the size of the image). Here's the finished result.

The screenshot shows a web browser window titled "Panorama Database Query:Real Estate Listings/newpropertylisting". Inside the browser, there is a form titled "Property Listing...". The form contains the following fields and controls:

- Address: (with a red exclamation mark icon to its right)
- City: (with a red exclamation mark icon to its right)
- Asking Price: (with a red exclamation mark icon to its right)
- Bedrooms: (with a blue up/down arrow icon to its right)
- Bathrooms: (with a red exclamation mark icon to its right)
- Age:
- Garage:
- Stories: (with a blue up/down arrow icon to its right)
- Air Conditioning: (with a blue up/down arrow icon to its right)
- Fireplace: (with a blue up/down arrow icon to its right)
- Pool:
- Agent:
- Submit:

In this case the image was on the same web server as the Panorama Enterprise server. If the image is on a different server you can use the complete URL.

```
webformitembang "http://www.somedomain.com/Images/red_circle_bang.jpeg"
```

If you want this image displayed for errors in every web form in every database on your server, add the word `global` before the URL.

```
webformitembang "global /Images/red_circle_bang.jpeg"
```

If you include this in the `.ServerInitialize` procedure of a database that gets loaded automatically when the server starts up you'll only need to include this line once to have it take effect globally across your entire server.

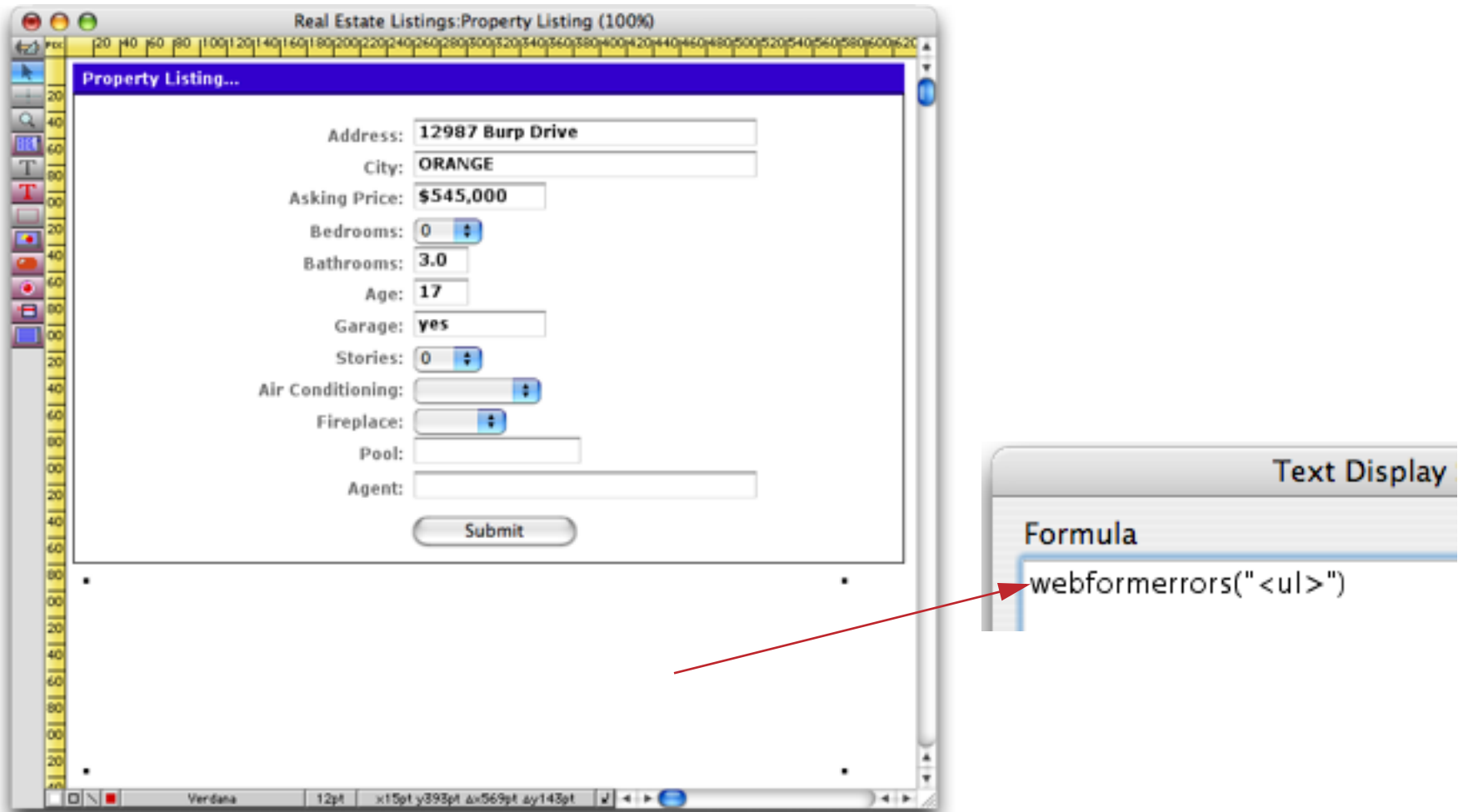
Displaying Error Explanations. The `webformitembang()` function tells the user which form items have problems, but doesn't explain what the problems are. To display explanations you'll need to use a separate text display object with the `webformerrors()` function. This function will have one parameter, a pattern that specifies how the list of errors should be formatted.

```
webformerrors(pattern)
```

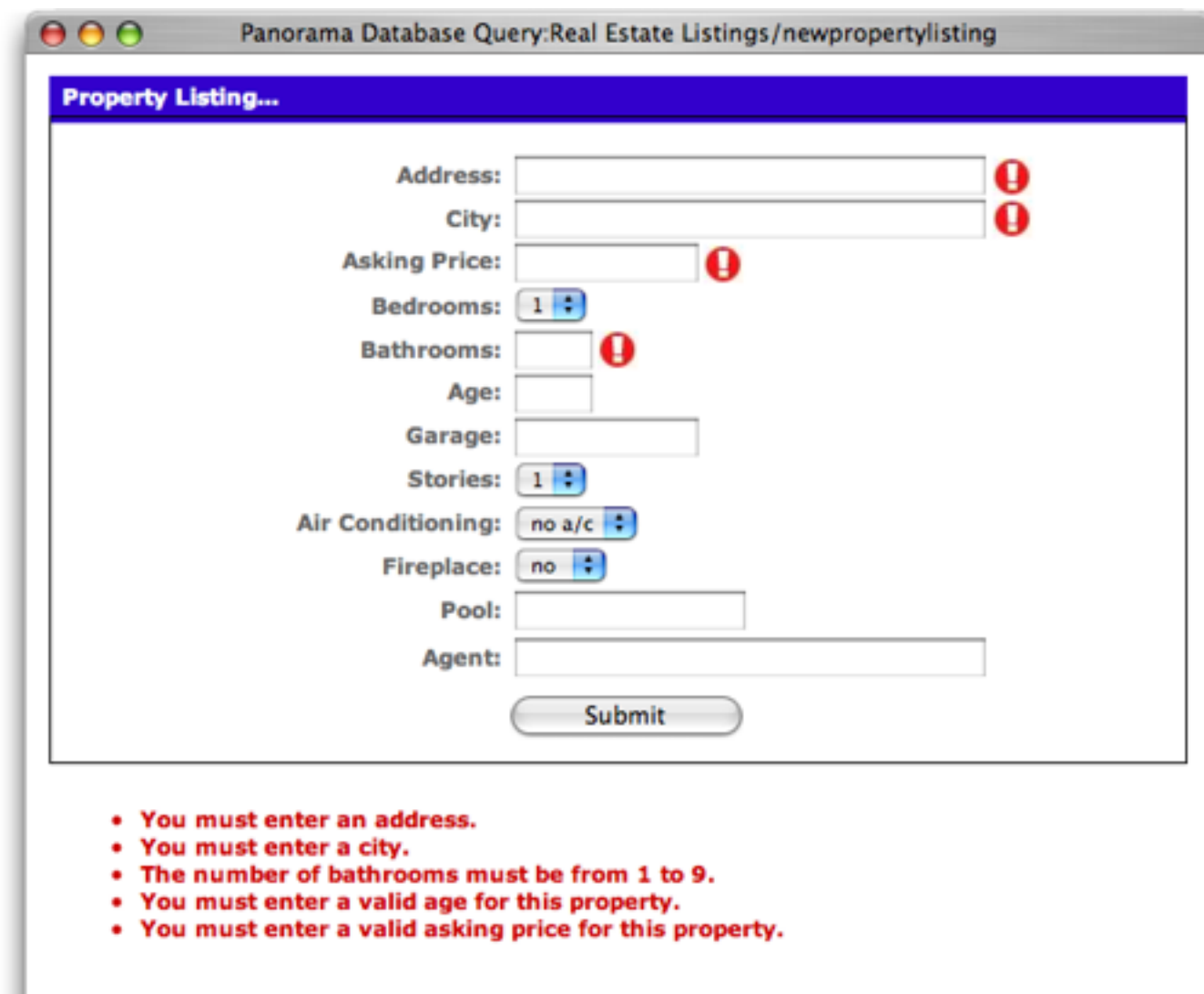
If the pattern is empty, the list will be formatted as a carriage return separated array (one line per error).

If the pattern is "``", the list will be formatted as an HTML unnumbered list (each item preceded with a `` tag). If you want a numbered list use "``". For other pattern options see "[Generating a List of Data Entry Errors](#)" on page 437.

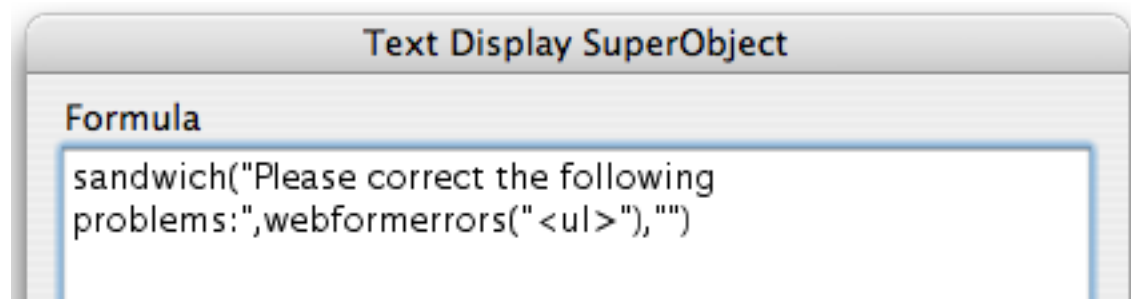
To use this function simply include it in a Text Display SuperObject on the form.



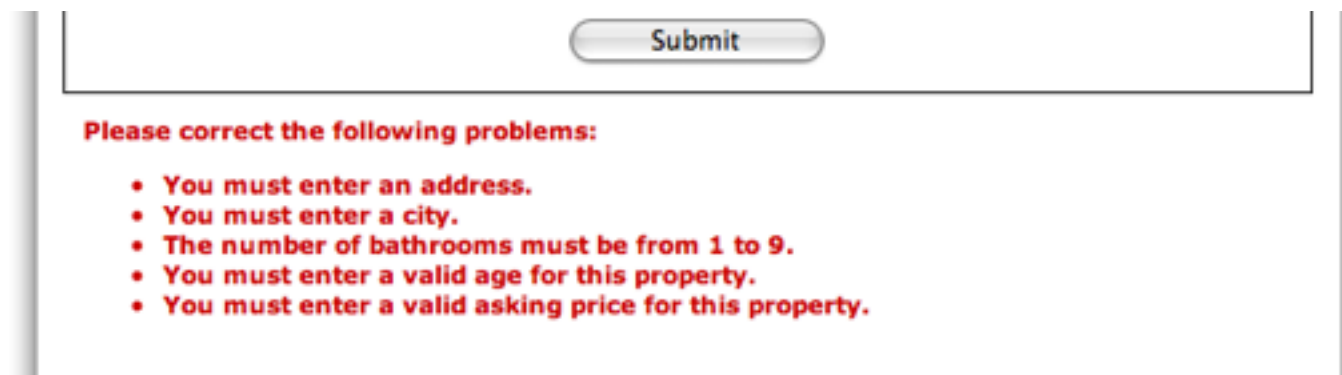
If there are no errors this function will return an empty string, and this object will be invisible. But if there are one or more errors they will be displayed according to the pattern you have specified, like this.



If you want to add additional text you can do so with the `sandwich()` function.



The additional text will appear if there is an error, but will not appear if there is no error.



Like any other Text Display SuperObject you can specify the font, size and color with the **Graphic Control Strip** or the **Text** and **Graphics** menus. In this example the object has been set to Verdana 12 red.

Pre-Filling Database Fields when Displaying a Form that uses Buffered Data Entry

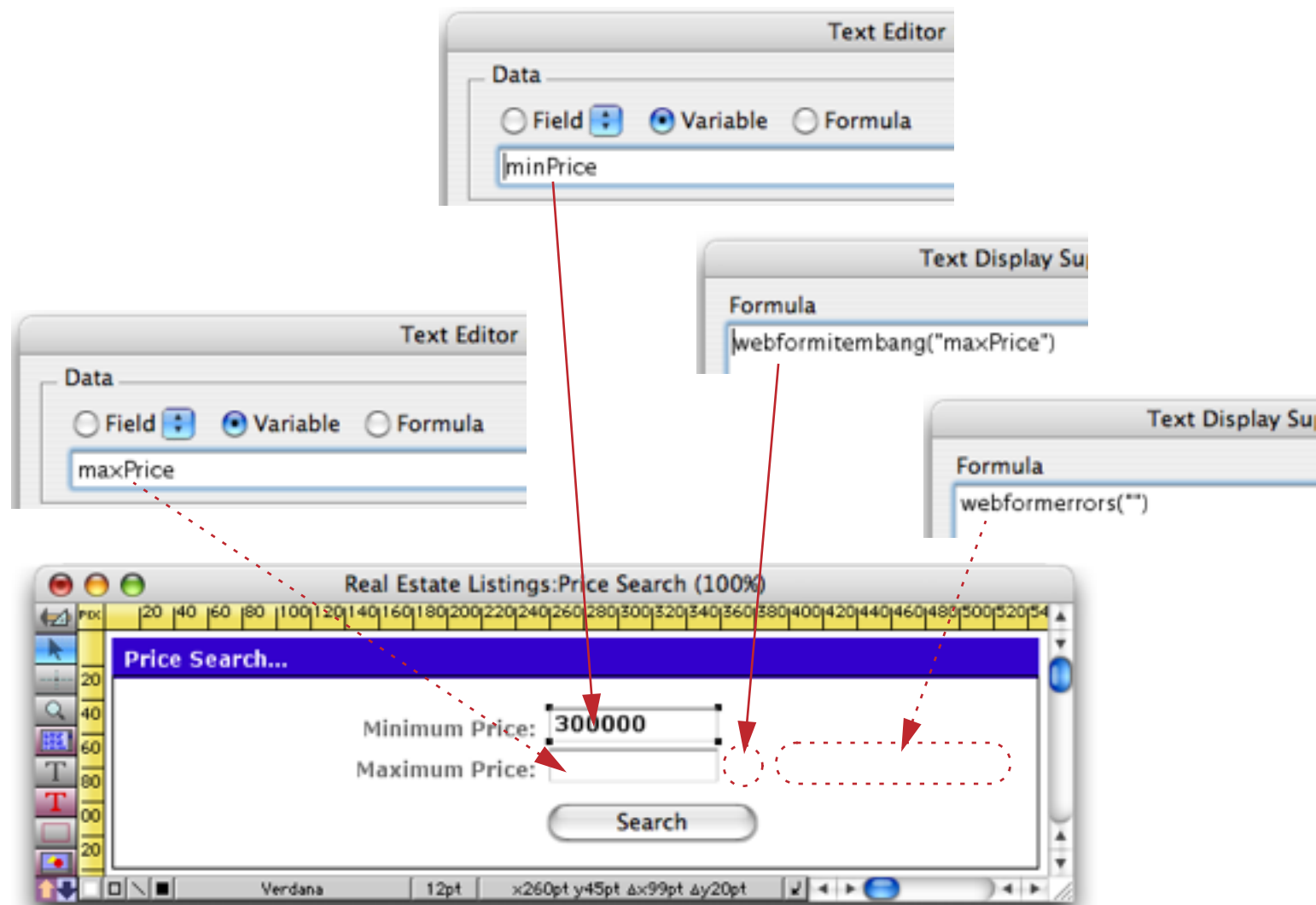
In the previous chapter you learned how to pre-fill database fields when displaying a normal web form (see "[Pre-Filling Database Fields](#)" on page 388). The exact same technique, using the `renderwebform()` function, works when using a web form that uses buffered data entry.



Note: Behind the scenes, the `renderwebform()` function is actually creating fileglobal variables with the item prefix for the form, filling these variables with data from the database fields, then using those variables to pre-fill the form. This is all handled automatically for you, however, so you don't need to worry about it at all.

Non Data Entry Forms (Searching, Navigation, etc.)

The techniques described for buffered data entry can be adapted for use with forms that are not linked to data entry into database fields. The illustration below shows a form designed to allow a user to view a range of properties by price. There are four objects of interest on this form: two Text Editor SuperObjects for editing the `minPrice` and `maxPrice` variables, and two Text Display SuperObjects for displaying any error that occurs (see “[Displaying an Image Instead of an Exclamation Point](#)” on page 444 and “[Displaying Error Explanations](#)” on page 445).



Here's the procedure that handles this form when the **Search** button is clicked.

```

Real Estate Listings:pricesearch
grabwebformitems ""
webformitemcheck "maxPrice","Maximum price must be greater than the minimum price!",val(maxPrice)<val(minPrice)
if webformerrors("")<>""
  maxPrice=str(val(minPrice)+10000)
  retrywebform
  rtn
endif
webselect Asking >= val("<<minPrice>>") and Asking <= val("<<maxPrice>>")
cgiHTML=renderwebtable("ListingTable")

```

The procedure starts by getting the values the user types into Panorama variables:

```
grabwebformitems ""
```

This line will create two fileglobal variables, `minPrice` and `maxPrice`, and fill them with the input data from the form. By the way, these variables now contain text, not numbers. The `val()` function must be used to convert these into numbers, as you'll see later in this procedure.

The next line uses checks to make sure that the maximum price is larger than the minimum price.

```
webformitemcheck "maxPrice",
  "Maximum price must be greater than the minimum price!",
  val(maxPrice)<val(minPrice)
```

The next four lines re-display the form (see “[Validating Data in the Variable Buffer](#)” on page 441) if the maximum price is not larger than the minimum price. The second line (highlighted in orange) tries to help the user by pre-filling the new maximum value with whatever the user entered for the minimum plus 10000.

```
if webformerrors("<>")<>" "
  maxPrice=str(val(minPrice)+10000)
  retrywebform
  rtn
endif
```

If the minimum is greater than the maximum the form will be re-displayed like this:



Otherwise the final two lines of the procedure will come into play.

```
webselect Asking >= val("<minPrice>") and Asking <= val("<maxPrice>")
cgiHTML=renderwebtable("ListingTable")
```

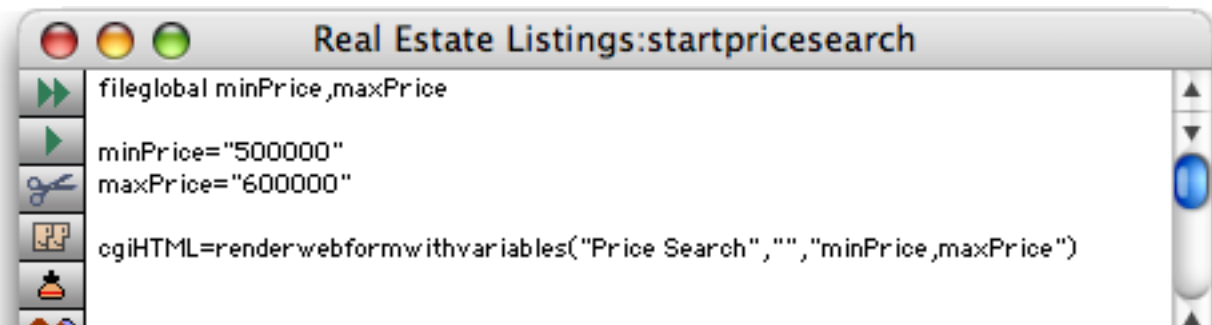
These lines select the matching properties and display them in a table.

| City | Address | Asking | Bedrooms | Bathrooms | Stories |
|------------------|-----------------------|------------|----------|-----------|---------|
| Midway City | 14802 Adams Avenue | \$ 714,000 | 2 | 1.00 | 1 |
| Westminister | 15392 Nantucket St. | \$ 707,400 | 3 | 1.75 | 1 |
| Huntington Beach | 18303 Lisa Lane | \$ 707,400 | 3 | 1.75 | 1 |
| Santa Ana | 2221 Monica Avenue | \$ 701,400 | 3 | 2.50 | 1 |
| Westminister | 15711 Azalea Street | \$ 705,000 | 3 | 1.75 | 1 |
| Huntington Beach | 18371 Delaware Street | \$ 707,400 | 3 | 1.50 | 1 |
| Huntington Beach | 8232 Michael Circle | \$ 711,000 | 3 | 1.75 | 1 |
| Huntington Beach | 2415 Florida Road | \$ 717,000 | 3 | 1.50 | 1 |
| Fountain Valley | 11801 Violet Circle | \$ 717,000 | 3 | 2.00 | 1 |
| Huntington Beach | 15911 Carmie Lane | \$ 717,000 | 3 | 1.75 | 1 |
| Huntington Beach | 6001 Calvin Avenue | \$ 717,000 | 3 | 1.50 | 1 |
| Westminister | 8990 Neptune Circle | \$ 719,400 | 3 | 1.50 | 2 |
| Westminister | 9421 Jennich Avenue | \$ 719,400 | 3 | 1.75 | 2 |
| Santa Ana | 4425 Edinger | \$ 719,400 | 3 | 2.00 | 1 |
| Garden Grove | 11281 Gary Street | \$ 719,400 | 3 | 1.75 | 2 |
| Huntington Beach | 1922 Delaware Street | \$ 720,000 | 3 | 1.75 | 1 |
| Irvine | 4891 Barkwood | \$ 726,000 | 3 | 1.75 | 1 |
| Santa Ana | 1102 Ross Avenue | \$ 731,400 | 3 | 1.50 | 1 |
| Westminister | 8191 22nd Street | \$ 731,400 | 3 | 1.75 | 1 |
| Huntington Beach | 217 Portland Avenue | \$ 737,700 | 3 | 1.75 | 1 |

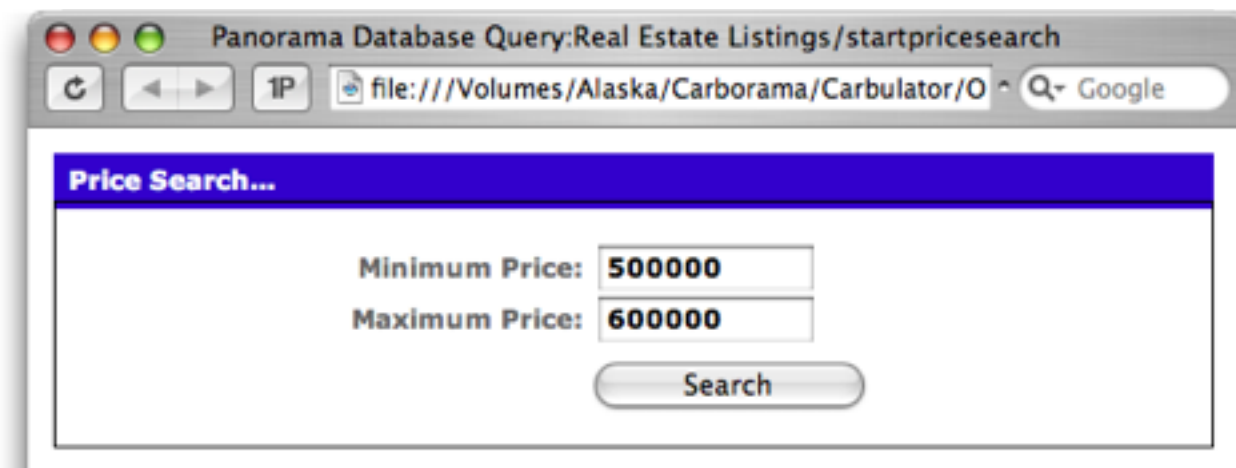
See “[What Records are we Talking About? \(The WebSelect Statement\)](#)” on page 413 and “[Generating HTML Tables Using a Web Table Template](#)” on page 414.

Pre-Filling Non Data Entry Forms

A web form that uses variables instead of fields normally comes up blank. You can, however, write a procedure that pre-fills these variables. Here's what such a procedure would look like for the search form introduced in the previous section. The procedure uses `renderwebformwithvariables()` function instead of the `renderwebform()` function that is normally used. This function has one additional parameter, a comma separated list of variables. (Note: This list is based on a formula, which usually means that it must be quoted, as shown below).



When this procedure is run the form appears on the user's browser with the minimum and maximum values already filled in.



The user can then modify the values as necessary and press the **Search** button to perform the search.

Using Cookies to Remember Form Values

With a small amount of extra effort you can use cookies to remember the values of one or more variables from the last time a form was used. No changes to the form are required. In this example, one line of code is added to the end of the search procedure to save the minimum and maximum values for the next time the form is used. The minimum and maximum could be stored in separate cookies, but this procedure combines them into one cookie with a comma between the values.

```

grabwebformitems ""
webformitemcheck "maxPrice", "Maximum price must be greater than the minimum price!", val(maxPrice)<val(minPrice)
if webformerrors("")<>""
    maxPrice=str(val(minPrice)+10000)
    retrywebform
    rtn
endif

webselect Asking >= val("<<minPrice>>") and Asking <= val("<<maxPrice>>")
cgiHTML=renderwebtable("ListingTable")
setcookie "PriceSearchBounds",minPrice+", "+maxPrice

```

Procedure OK.

The procedure that originally displays the form requires six new lines. This code tries to load the cookie value into a local variable, and if successful, initializes the minimum and maximum variables.

```

fileglobal minPrice,maxPrice
minPrice="500000"
maxPrice="600000"
local psCookie
psCookie=getcookie("PriceSearchBounds")
if psCookie<>""
    minPrice=array(psCookie,1,",")
    maxPrice=array(psCookie,2,",")
endif
cgiHTML=renderwebformwithvariables("Price Search", "", "minPrice,maxPrice")

```

Procedure OK.

Once these changes are made the form will “remember” the minimum and maximum values if the form is used from the same browser on the same computer. Each time a new search is started the default min and max will be the same values used the last time you searched. To learn more details about cookies see “[Working with Cookies](#)” on page 453.

Chapter 11: Advanced Topics



This chapter covers advanced web programming topics that don't fit in any of the previous chapters, including cookies, custom HTML headers, non-HTML content, and simultaneously sharing and web publishing.

Working with Cookies

An **HTTP cookie**, or a **Web cookie** (also called simply a **cookie**), is a parcel of text sent by a server to a web browser and then sent back unchanged by the browser each time it accesses that server. HTTP cookies are used for authenticating, tracking, and maintaining specific information about users, such as site preferences and the contents of their electronic shopping carts. Without cookies, each retrieval of a web page or component of a web page is an isolated event, mostly unrelated to all other views of the pages of the same site. By returning a cookie to a web server, the browser provides the server a means of connecting the current page view with prior page views. (This paragraph was adapted from material found on Wikipedia. You can find a lot more at http://en.wikipedia.org/wiki/HTTP_cookie).

Cookie Name and Value

A cookie has a name and a value. The name is used to identify the cookie, the value is the actual contents. This is similar to a field or variable, which also have a name and a value.

Creating a Cookie

To create a cookie use the `setcookie` statement, which has four parameters:

```
setcookie name,value,expiration,options
```

The last two parameters are optional, but the first two are required. The first parameter is the name of the cookie. This must contain only letters and numbers. The second parameter is the value of the cookie. Here are some examples of how to create cookies:

```
setcookie "SalesTax","8.25"  
setcookie "TableFontPref","Verdana"  
setcookie "MyAreaCode","858"
```

Cookie Expiration Date

Cookies don't last forever. In fact, if you don't specify an expiration date for a cookie then it will disappear when the user quits the browser. If you want to keep a cookie around for a while set an expiration date for it, like this:

```
setcookie "SalesTax","8.25","12 hours"  
setcookie "TableFontPref","Verdana","12 weeks"  
setcookie "MyAreaCode","858","12 years"
```

In addition to the units listed above you can also specify the expiration time in seconds, minutes, and days (but not months).

Cookie Options

This optional parameter allows you to add more options to the cookie. The best source of information we've found for additional cookie options is at <http://www.ietf.org/rfc/rfc2965.txt>, however, please note that the Panorama Server uses the original cookie format, not version 2 (it appears that the version of Apache shipped with MacOS does not support version 2 cookies). In general you probably don't need to worry about these extra options.

Retrieving a Cookie Value

To get the value of a previously stored cookie use the `getcookie()` function. This function has one parameter, the name of the cookie. Here are some examples.

```
local salestax,tablefont,phoneareacode
salestax=val(getcookie("SalesTax"))
tablefont=getcookie("TableFontPref")
phoneareacode=getcookie("MyAreaCode")
```

Listing Cookies

The `getcookielist()` function gets a list of all cookies that have been stored by this server. It has no parameters.

```
local myCookies
myCookies=getcookielist()
```

Building a Very Simple Shopping Cart

One of the most common uses for cookies is to build a shopping cart for ordering items from a catalog. In the following sections we'll examine the basic principles of creating a shopping cart using cookies.

The contents of this example shopping cart will be stored in a cookie named `HobbyShopCart`. Within this cookie each line item is stored as a separate line. Within each line there are three columns separated by tabs: Quantity, Item and Price. A typical `HobbyShopCart` cookie would look something like this:

```
1      Southern Pacific 50' Box Car      11.19
1      Amtrak Genesis                    47.99
1      Southern Pacific 30' Gondola      13.69
1      ATSF 50' Box (Grand Canyon)      4.49
```

The following sections will describe three procedures for displaying and manipulating the shopping cart cookie.

Displaying the Shopping Cart

The routine `CartDisplay` displays the current contents of the shopping cart. The top section of the procedure handles the cookie.

```

fileglobal ShoppingCartItems,ShoppingCartPrices,ShoppingCartTotal
local theShoppingCart,webformtext

/* get and format the shopping cart */
theShoppingCart=getcookie("HobbyShopCart")
htmlarraytable theShoppingCart,1,1,ShoppingCartItems,
"
font=Verdana fontsize="-2" border=0

column={«1»+" "+«2»} align=left width=290
column={«3»} align=right width=90
"

/* calculate the total */
arrayfilter theShoppingCart,ShoppingCartPrices,1,array(import(),3,1)
if ShoppingCartPrices=""
ShoppingCartTotal=0
else
execute "ShoppingCartTotal="+replace(replace(ShoppingCartPrices,"$",""),1,"+")
endif

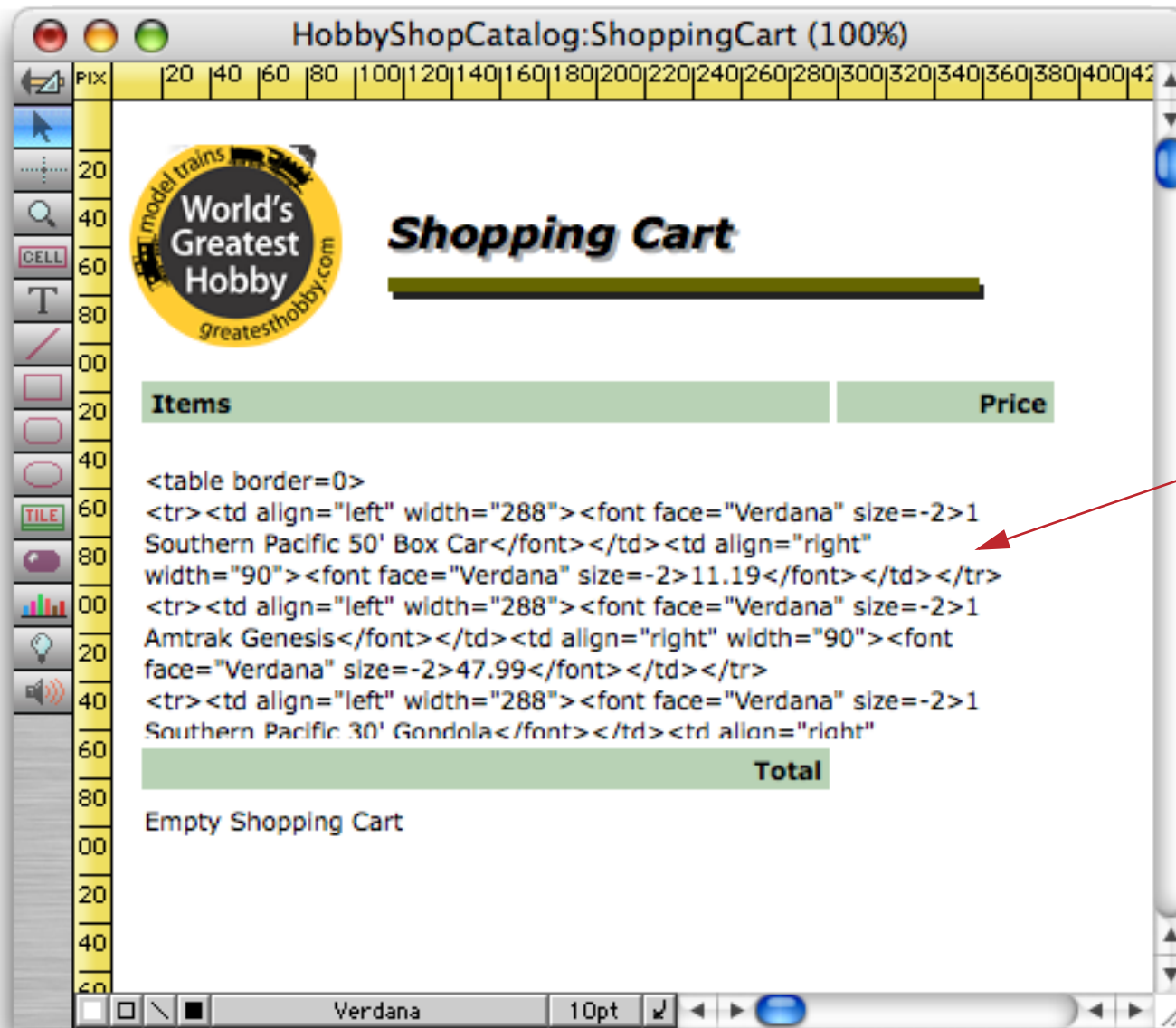
/* render the page */
webformtext=getwebtemplatetext("ShoppingCart")
webmerge "",webformtext,webformtext
cgiHTML=webformtext

```

Procedure OK.

The procedure starts by using the `getcookie()` function to get the current contents of the shopping cart and store it in the variable `theShoppingCart`. The cookie contains an array, so the procedure uses the `htmlarraytable` statement to format the data as a table (see “[Displaying an Array in a Web Form](#)” on page 411). The rest of the procedure uses ordinary techniques to calculate the total and render the page using a form.

The form contains a Text Display SuperObject that displays the line items:



Text Display Super-Object displays the ShoppingCartItems variable.

When this procedure is used the shopping cart is displayed on the browser.

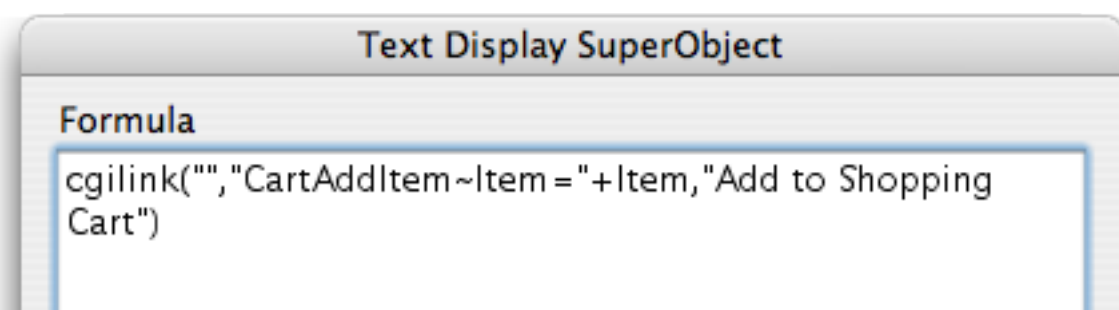


Adding a New Item to the Cart

To add a new item to the shopping cart the first step is to select the item from the database. Usually this is done by some sort of search, which is covered elsewhere in this handbook. Once the item is located the web page displaying the item will have a link or button for adding the item to the shopping cart.



In this example the link has been created with a Text Display SuperObject (see "[Linking to Panorama Procedures \(from a Panorama Form\)](#)" on page 261). Here's the formula used in this object.



This link triggers the [CartAddItem](#) procedure, which contains the code which actually adds the item to the shopping cart.

```

fileglobal ShoppingCartItems,ShoppingCartPrices,ShoppingCartTotal
local item,price,newCartLine,theShoppingCart,webformtext

/* locate the database record for the item to add */
weburlselect cgiExtraParameters

/* build the new line */
item=Item
price=Price
newCartLine="1 "+item+"pattern(price,"$.##")

/* add the new line to the cart "cookie" */
theShoppingCart=getcookie("HobbyShopCart")
theShoppingCart=sandwich("",theShoppingCart,¶)+newCartLine
setcookie "HobbyShopCart",theShoppingCart,""

/* format the shopping cart for display */
htmlarraytable theShoppingCart,¶,ShoppingCartItems,
"
font=Verdana fontsize="-2" border=0
column={«1»+" "+«2»} align=left width=290
column={«3»} align=right width=90
"

/* calculate the new total */
arrayfilter theShoppingCart,ShoppingCartPrices,¶,array(import(),3,-)
if ShoppingCartPrices=""
ShoppingCartTotal=0
else
execute "ShoppingCartTotal="+replace(replace(ShoppingCartPrices,"$",""),¶,"+")
endif

/* render the page */
webformtext=getwebtemplatetext("ShoppingCart")
webmerge "",webformtext,webformtext
cgiHTML=webformtext

```

Procedure OK.

The procedure starts by selecting the database record (see [“What Record Are We Talking About?”](#) on page 395).

```
weburlselect cgiExtraParameters
```

Next it builds the new line that it will add to the cookie, using the format described in [“Building a Very Simple Shopping Cart”](#) on page 454.

```

item=Item
price=Price
newCartLine="1 "+item+"pattern(price,"$.##")

```

The next step actually updates the cookie. First it reads the cookie into a variable (see [“Retrieving a Cookie Value”](#) on page 454), then it adds the new line, and finally it writes out the new cookie (see [“Creating a Cookie”](#) on page 453).

```

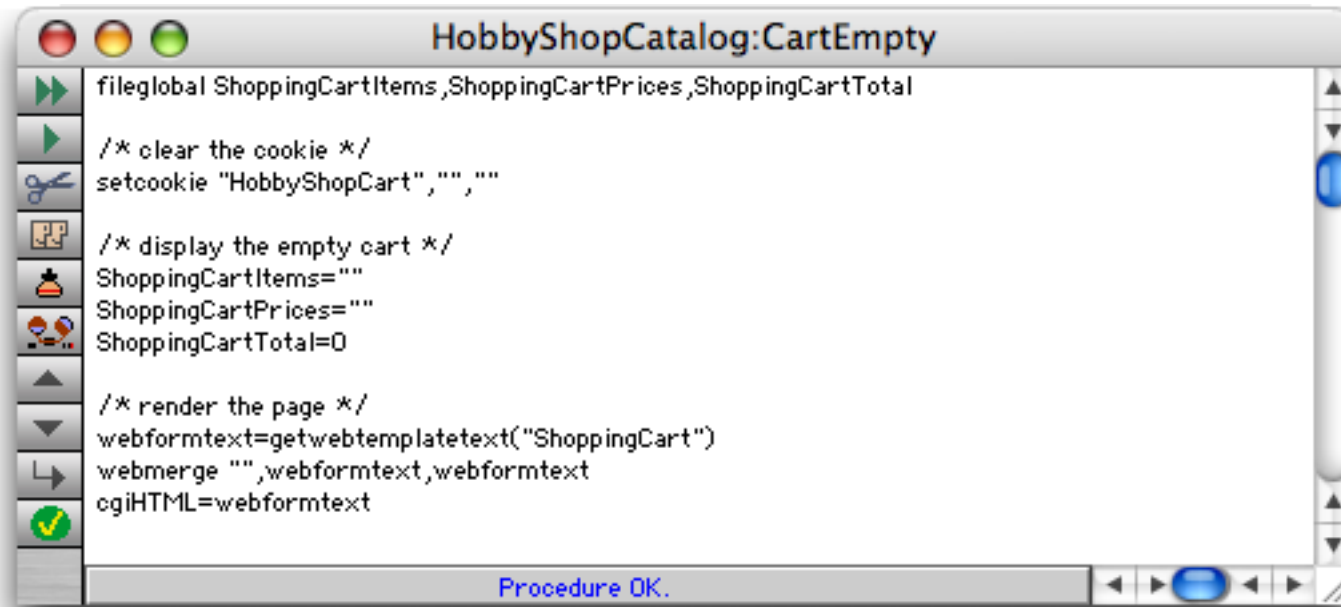
theShoppingCart=getcookie("HobbyShopCart")
theShoppingCart=sandwich("",theShoppingCart,¶)+newCartLine
setcookie "HobbyShopCart",theShoppingCart,""

```

The rest of procedure displays the updated shopping cart, and is almost identical to the [CartDisplay](#) procedure described in the previous section.

Clearing the Shopping Cart

This procedure is very simple — it simply sets the cookie to "" and displays the empty cart. Make sure there's a link or button that triggers this procedure!



Accessing Additional Web Server Information

The `webserverinfo` statement allows you to access additional information about the web server and the current browser request. This statement has two parameters

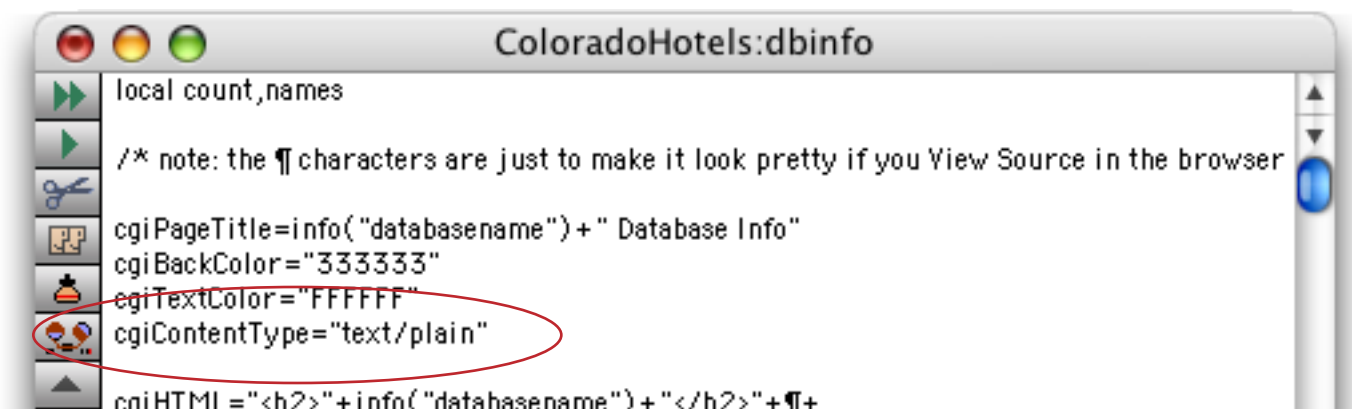
```
webserverinfo option,result
```

The second parameter, *result*, is simply the name of the variable to receive the information. The first parameter, *option*, specifies what additional information you want to access. Choose from the options in the table below. (Note: Some of these options may not be available on all servers, or when running the server simulator described in the next chapter.)

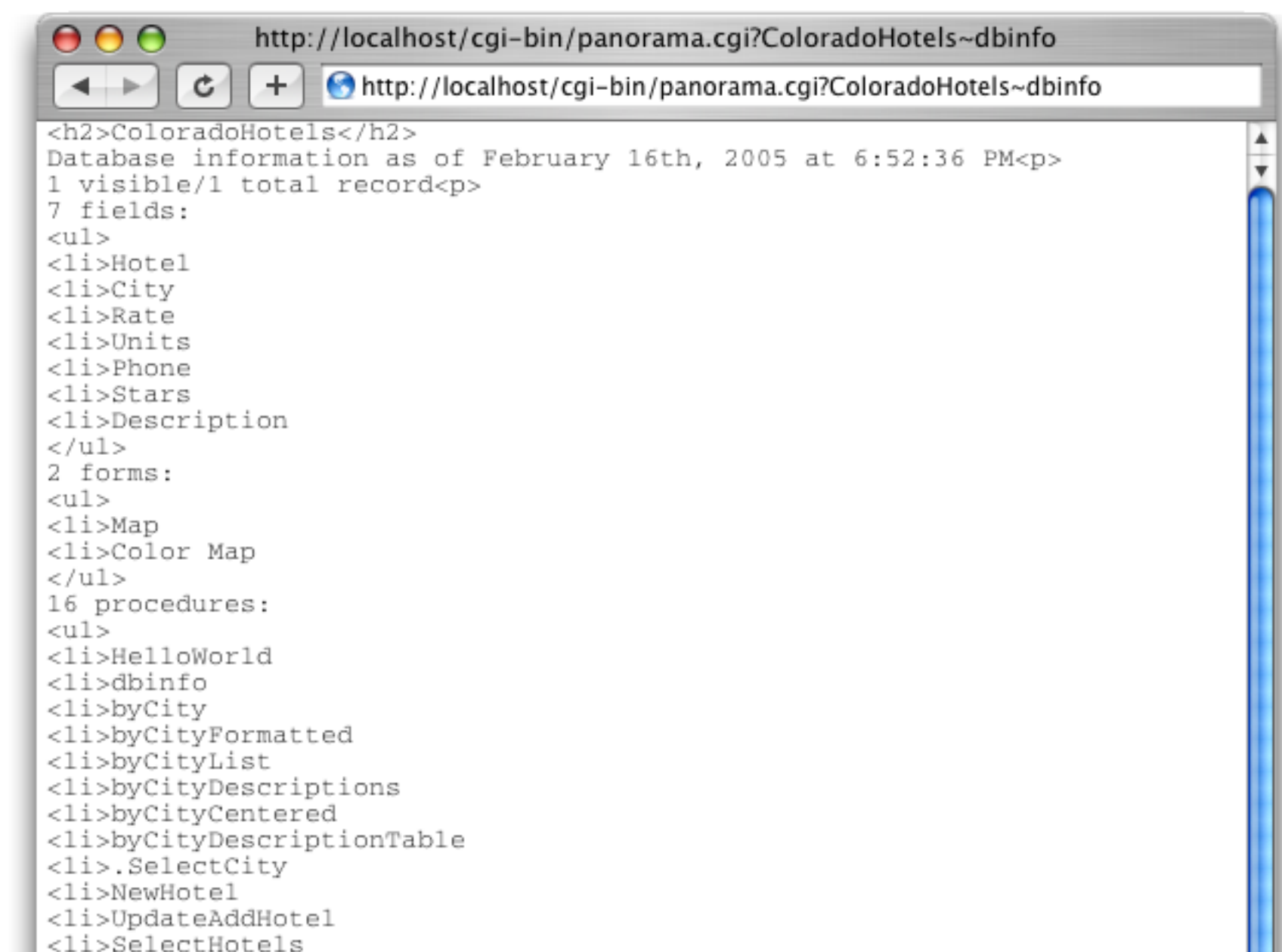
| Option | Description | Typical Value |
|------------------------|---|--|
| "QUERY_STRING" | The information (if any) following the "?" in the URL for this request. | ColoradoHotels~byCity~Aspen |
| "SERVER_SOFTWARE" | The name and version of the information server answering the query. | Apache/1.3.29 (Darwin) |
| "SERVER_ADDR" | The IP address of the server for this URL. | 207.35.76.24 |
| "SERVER_ADMIN" | The administrators e-mail address for this server (if it has been set up). | webmaster@myserver.com |
| "SERVER_NAME" | The servers host name, DNS alias or IP address. (For Apache the name appearing on the relevant ServerName directive (may be in the general section or a <virtualhost> section). | www.myserver.com |
| "SERVER_PORT" | The port number on this server to which this request was directed. | 80 |
| "SERVER_SIGNATURE" | The HTML string that may be embedded in the page to identify this host. | Apache/1.3.29 Server at jim-reas-computer.local Port 80 |
| "REMOTE_ADDR" | The IP address of the host making this request. | 207.35.76.27 |
| "REMOTE_PORT" | The port number used by the remote host when making this request. | 4325 |
| "REQUEST_METHOD" | The method used for this request for HTTP "GET", "HEAD" or "POST" | GET |
| "HTTP_ACCEPT" | The MIME types the requestor will accept as defined in the HTTP header. | */* |
| "HTTP_ACCEPT_ENCODING" | The compression types the requestor will accept as defined in the HTTP header. | gzip, deflate;q=1.0, identity;q=0.5, *,q=0 |
| "HTTP_ACCEPT_LANGUAGE" | The LANGUAGE types the server is requested to accept as defined in the HTTP header and typically used for content negotiation. | en-us |
| "HTTP_CONNECTION" | The type of connection as defined in the HTTP header. | Keep-alive |
| "HTTP_HOST" | The base URL of the host. | www.myserver.com |
| "HTTP_REFERER" | The URL of the page that made this request. If linked from e-mail or manually entered this value is NULL. (Depending on your server settings this may be disabled, in which case the value will always be NULL.) | http://www.samplesite.com/stuff.html |
| "HTTP_USER_AGENT" | The browser id or user-agent string identifying the browser (nominally defined by RFC 1945 and RFC 2068). For a list of current browsers see: http://www.zytrax.com/tech/web/browser_ids.htm | Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en-us) AppleWebKit/125.2 (KHTML, like Gecko) Safari/125.8 |

Non HTML Content Types (text/plain)

The response generated by a Panorama procedure is normally an HTML page. However, it is also possible to render a plain text page. To do this change the `cgiContentType` variable to "text/plain".



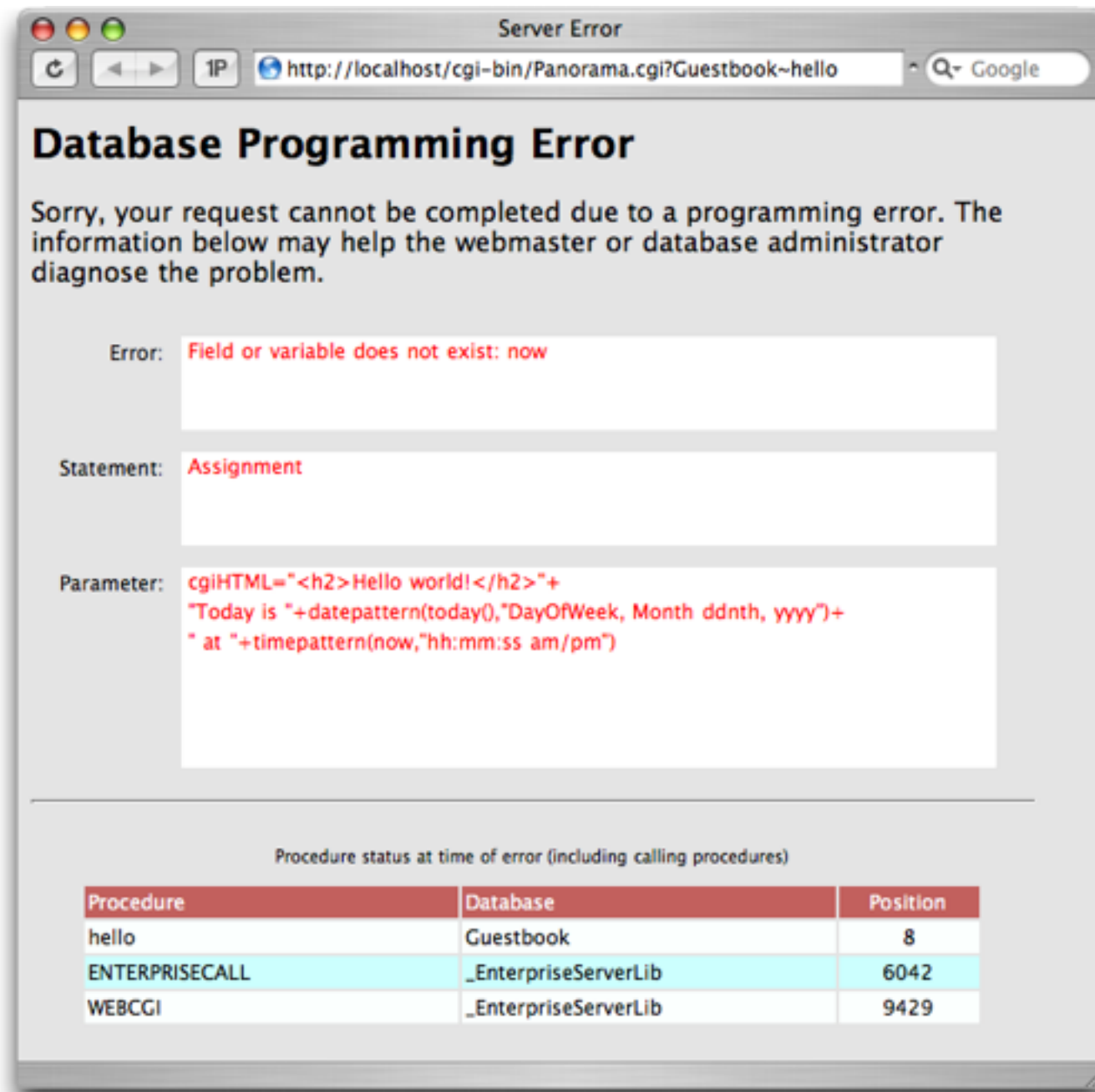
When the browser receives this page it will display the text "as-is" using a monospaced font (the exact appearance depends on your browser). Any HTML tags in the text will simply be displayed rather than be interpreted to control the appearance of the page.



Advanced note: Theoretically it would also be possible to return other mime types, for example `image/jpeg` or `image/gif`. However, we have not tested and don't support any other mime types other than `text/html` or `text/plain`. Unless you come up with some way to generate custom images on the fly you'll be better off using standard image files displayed via the HTML `` tag.

Custom Handling of Programming Errors

There are three possible types of errors that can occur when running a Panorama web procedure. 1) An error will occur if the database specified by the URL doesn't exist on the server, 2) An error will occur if the procedure specified by the URL doesn't exist on the server, or 3) The procedure itself contains a programming error (for example it tries to access a variable that doesn't exist). Panorama Enterprise's standard method for handling these situations is to display a web page describing the error (see "[Web Procedure Errors](#)" on page 342).



This standard error display page is helpful for tracking down problems, but you may not want to display that much information on a production web site. You can customize the way Panorama Server responds when an error occurs to display as much or as little information as you want. When a programming error occurs Panorama looks in three places for code that will handle the error:

If the current database contains a procedure named `cgiErrorHandler`, that procedure will be executed. (This will work for programming errors, but will not work if a URL specifies a database that does not exist. In that case the standard error handling will be used.)

If a global variable exists named `cgiGlobalErrorHandler`, the text in that variable will be treated as a procedure and executed.

If there is no `cgiErrorHandler` procedure and no `cgiGlobalErrorHandler` variable the server will display the standard error page shown above. You can customize the HTML and CSS templates used in this situation by modifying the `WebError.html` and `WebError.css` files in the `Templates` folder inside the `Panorama:Extensions:Enterprise` folder. (If you modify these files, be sure to keep a copy of your modifications outside of this folder so that you can restore your changes the next time you update Panorama.)

Writing a Procedure to Handle Programming Errors

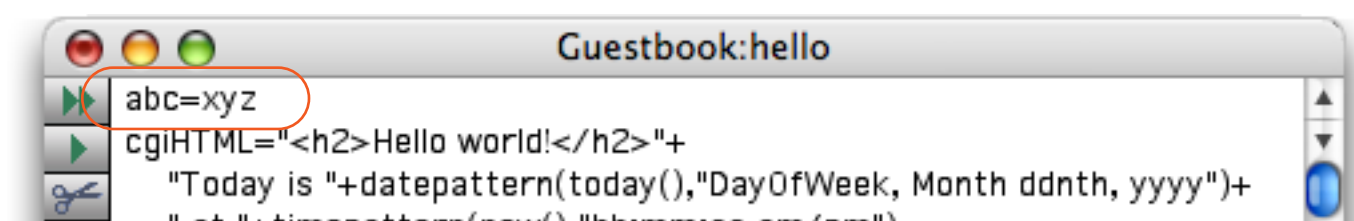
If you write a `cgiErrorHandler` procedure or use the `cgiGlobalErrorHandler` variable, the code should generate a page in the global variable `cgiHTML` that will display the error in some form. Before starting the procedure, Panorama sets up five global variables with information about the error that occurred.

| Variable | Description |
|------------------------------------|--|
| <code>cgiResponseError</code> | This variable contains the error message. |
| <code>cgiErrorStatement</code> | This variable contains the statement that caused the error. |
| <code>cgiErrorParameter</code> | This variable contains the parameter that caused the error. |
| <code>cgiErrorStack</code> | This variable contains an array that contains a line for each procedure that is currently active (the current procedure and any procedures that called it). Each line contains four entries separated by tabs: the procedure name, the database name, the trigger for the procedure, and the location of the error within the procedure (character count). |
| <code>cgiErrorExecuteSource</code> | If the error was in an <code>execute</code> statement, this variable contains the source of the executed code. |

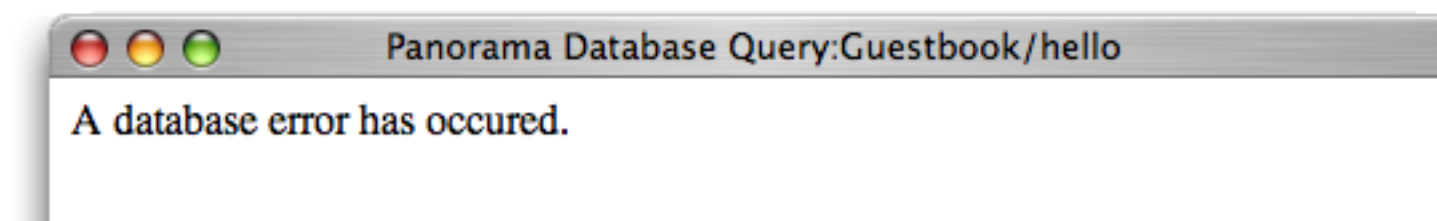
Here is an example of a very simple error handling procedure.



Now suppose a procedure is run that has an error, like this procedure which tries to use two undefined variables (`abc` and `xyz`).



Instead of displaying the standard error page, the server will display this simplified error page.



Of course you'll probably want to create a page a bit fancier than this! The point, however, is that you have full control over the page that is displayed.

If you want to completely override the error page for every database on the server, include code like this in the `.InitializeServer` procedure of a database that gets automatically opened when the server starts up.

```
global cgiGlobalErrorHandler
cgiGlobalErrorHandler=|||cgiHTML="A database error has occurred."|||
```

The error handling procedure allows you to control the page that is displayed when an error occurs, but it does not disable the server's normal error logging routine. The error will still be logged and can be viewed in the **Server Administration** wizard, and if the option has been enabled an e-mail will be sent to the system administrator.

Opening and Closing Databases on the Server

Panorama normally takes care of opening and closing files on the server for you — you don't have to worry about it. When a web URL references a database that database is automatically opened (if it isn't opened already).

However, occasionally you'll need to explicitly open a database. The most common reason to do this is to use the `lookup()` function. Panorama does not automatically open databases that are referenced by a `lookup()` function, you'll need to do that yourself with the `openserverfile` statement. When your done you can close the database with the `closeserverfile` statement.

The `openserverfile` statement has one parameter, the name of the database to open.

```
openserverfile filename
```

Unlike the `openfile` statement, the filename must be only a filename — you cannot specify an alternate folder. The `openserverfile` statement will make sure that the database is opened in a manner consistent with the Panorama Server's normal practices for opening databases. It's ok if the database is already open, the server will handle that properly. (For debugging, the `openserverfile` statement can also be used on a normal copy of Panorama. In that case the target database must be in the same folder as the current database.) The database will not open with it's normal windows, but only with the data sheet in the upper left hand corner of the display. When a database is opened with `openserverfile` the `.Initialize` procedure is not called. However, if the database has a `.InitializeServer` procedure, it will be called.

When your done with a database you can use the `closeserverfile` statement to close it. If a web procedure needs to closes a shared or web published database (any database that has been uploaded to the server with the Database Sharing Options wizard) it should use this statement instead of the `closefile` statement. The `closeserverfile` statement makes sure that the database is closed in a manner consistent with the Panorama Server's normal practices for closing databases (in fact, if the database is shared and other users are currently accessing the database, it will not close). For debugging, the `closeserverfile` statement can also be used on a normal copy of Panorama. In that case it will simply close the database.

Combining Web Publishing with Database Sharing

If a database has both the sharing and web publishing options enabled that database can be simultaneously accessed by Panorama users and over the web. Any changes made with the web interface will automatically appear in Panorama clients the next time they are synchronized, while changes made using the Panorama interface will immediately appear when the database is accessed from a web browser (no synchronization is needed in this case since the web interface accesses the server directly). In most cases you don't have to worry about any of this, simply write your web procedures as described in this chapter and everything will work fine.

Web Publishing vs. Record Locking

Panorama Enterprise's web publishing interface does not automatically do record locking. A web user can modify a record even if it is currently locked by a Panorama user, and a Panorama user can lock a record that is currently being edited by a web user. (In addition, two web users can both edit the same record simultaneously). In each of these cases the "winner" will be whoever changes the record last.

Manually Locking and Unlocking Records

A web procedure can use the `webshare` statement to manually lock or unlock a record. For example, if a procedure displays a form that allows a user to edit a record it can also lock the record.

```
weburlselect cgiExtraParameters
webshare "lock"
if error
  /* record was already locked, so we can't edit it now */
  cgiHTML = renderwebform("Record Lock","")
  rtn
endif
cgiHTML = renderwebform("Edit Record","")
```

The procedure that processes the data when the **Submit** button is pressed must unlock the record.

```
webformtodatabase "whatrecord=record_id"
webshare "unlock"
```

However, there's potentially a serious problem with this technique — what if the user never presses the **Submit** button? If they walk away from the computer (or simply click away to another web page) the record will never be unlocked, and no one else can ever edit it (though this can be cleared using the Server Administration wizard, see "[Forcing a Session to Close](#)" on page 79). This is the reason why the web publishing interface does not automatically lock records.

Checking Record Lock Status

The `webshare` statement can also be used to find out the status of locked records. Use the `"locked"` option to find out if the current record is locked.

```
local rlock
webshare "locked",rlock
if rlock=true()
  /* uh-oh, this record is locked */
  cgiHTML = renderwebform("Record Lock","")
  rtn
endif
/* continue with program */
```

Use the `"locklist"` option to get a list of all currently locked records. This procedure selects all locked records.

```
local busyrecords
webshare "locked",busyrecords
select arraycontains(busyrecords,str("serverrecordid"),¶)
```

This list will contain the record ID numbers of all currently locked records, one per line. (You can use the `info("serverrecordid")` function to determine the record ID of the current record, as shown above.)

Running a Program Every Minute on the Server

Do you need to run code every minute on the Panorama Server, for example to run a daily report or perform some similar task? We recommend that you avoid this if possible, but if you do need to do this you must prefix your code with `serverbackupcheck` statement, like this:

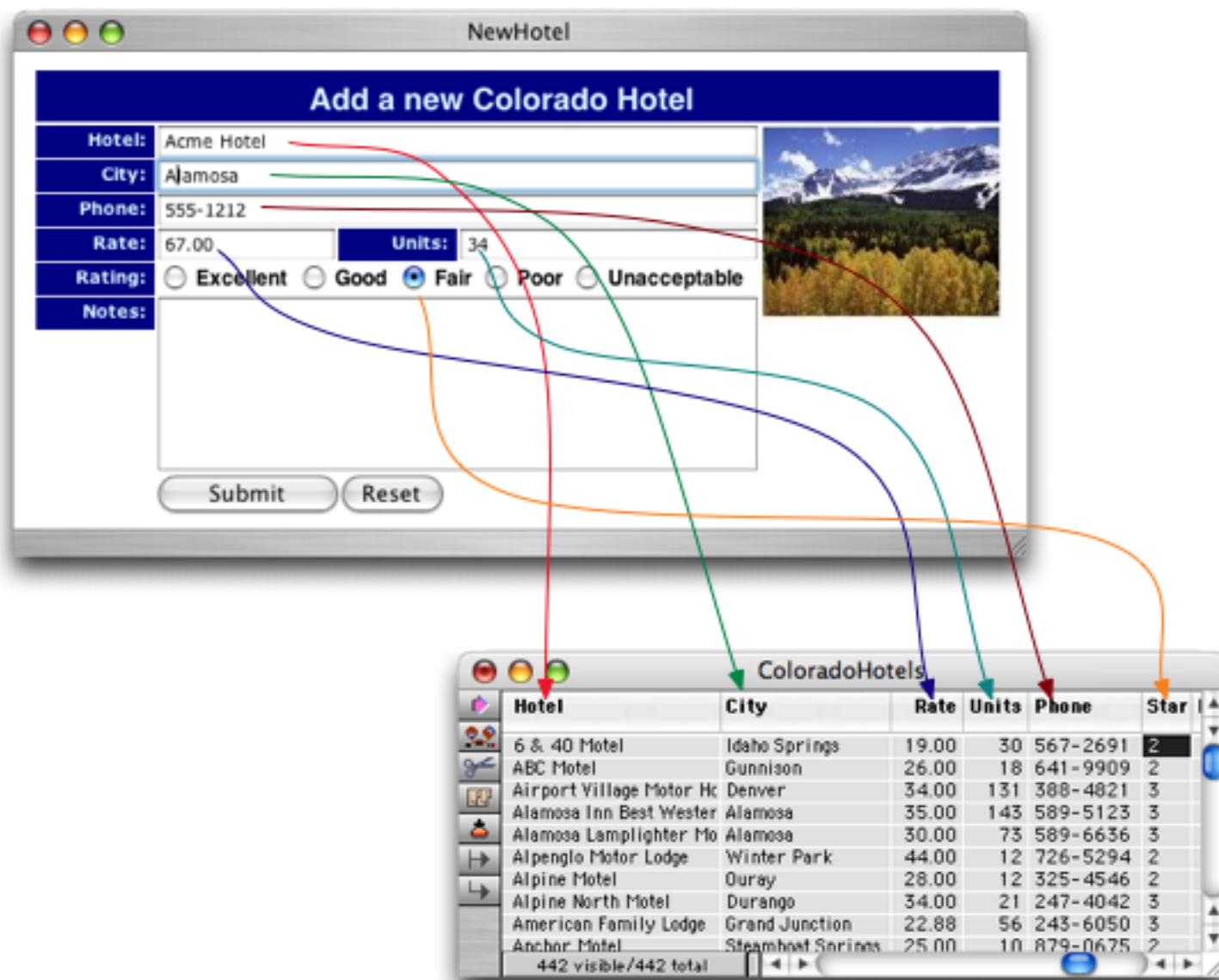
```
ExecuteEveryMinute = {serverbackupcheck your ... code}
```

This insures that periodic events get logged, that you have error trapping for your periodic code, and that server backups work. (To enable logging of periodic events you must put "periodic" or "all" in the `< SHARE-LOGACTIVITIES >` list.) If your periodic code contains an error or displays an alert this will be trapped and logged and an .xlg file will be created. All periodic activity is also logged in the debug point logs of recent activity (.dbg files).

Web Based Data Entry with the WebFormToDatabase Statement

The most basic application for web based forms is data entry - adding new entries to a database. Chapter 10 covered several techniques for getting data from a web form into Panorama, including our preferred technique, data entry buffered through variables (see “[Buffered Data Entry](#)” on page 438). The following sections describe another technique for performing web based data entry using the `webformtodatabase` statement. This statement was originally developed for a predecessor to Panorama Enterprise (called PanSTAR). The advantage of this statement is that it is quite easy to user — if the item names in the submitted form match the database names then `webformtodatabase` will do most of the work for you with a single line of code (the item names will automatically match if you use a Panorama web form see “[Web Forms](#)” on page 231). The disadvantage is that this statement is not as flexible as the techniques described in Chapter 10, especially when it comes to reporting data entry errors (input validation). For new applications we recommend that you use the techniques described in Chapter 10. However, the `webformtodatabase` statement is still included in Panorama Enterprise Server for compatibility with older databases.

The operation of the `WebFormToDatabase` statement is very simple. The statement scans the items submitted to it from the browser, checking items that match a database field. For each match it copies the submitted data into the database. That’s why it’s so important that the field names in your form match the field names in the database. If they don’t match, the `WebFormToDatabase` statement won’t update the database with those values.



The `WebFormToDatabase` statement has one parameter, a list of options that customize the operation of the statement. For example the options can be used to specify whether the submitted data should be entered into a new record or into an existing record. Add one or more of the options listed below to customize data entry.

| Option | Description |
|--------------------------------|---|
| <code>whatRecord=option</code> | <p>This option controls what record in the database is modified. There are five options:</p> <p><code>add</code> - Add a new record (this is the default if no option is specified, see “WebFormToDatabase — Adding a New Record” on page 472).</p> <p><code>record_id</code> - Update the current record based on an embedded record id (see “WebFormToDatabase — Modifying an Existing Record with Embedded Record ID” on page 477).</p> <p><code>active</code> - Update the current record, whatever that happens to be (be careful!!). (See “WebFormToDatabase — Modifying an Existing Record” on page 476.)</p> <p><code>find</code> - Find the record to update based on the key field(s). (See “WebFormToDatabase — Modifying an Existing Record” on page 476.)</p> <p><code>select</code> - Select the record to update based on the key field(s). (See “WebFormToDatabase — Modifying an Existing Record” on page 476.)</p> <p><code>findadd</code> - Find record based on the key field(s). If it is not found, add a new record (See “WebFormToDatabase — Modifying an Existing Record” on page 476.)</p> <p><code>selectadd</code> - Select record based on the key field(s). If not found, add a new record (See “WebFormToDatabase — Modifying an Existing Record” on page 476.)</p> <p><code>formdump</code> - This option doesn't actually modify the database, just displays the submitted items. This is useful for debugging. (See “WebFormToDatabase — Displaying Form Items” on page 471.)</p> |
| <code>keyfield=field</code> | <p>This option defines a key field to be used with the <code>find</code>, <code>select</code> or <code>findadd</code> options. You may define more than one key field. The keyfield is discussed in more detail later in this section (see “WebFormToDatabase — Modifying an Existing Record” on page 476).</p> |
| <code>upper=field</code> | <p>This option specifies that one or more fields must be automatically capitalized to all upper case (see “Forcing Input to Upper or Lower Case” on page 473). For example, the word <code>computer</code> would be converted to <code>COMPUTER</code>. You should have one <code>upper=</code> option for each field you want capitalized, for example:</p> <pre>upper="State" upper="Country"</pre> |
| <code>upperword=field</code> | <p>This option specifies that one or more fields must be automatically capitalized to Initial Caps (see “Forcing Input to Upper or Lower Case” on page 473). For example, the word <code>computer</code> would be converted to <code>Computer</code>. You should have one <code>upperword=</code> option for each field you want capitalized this way, for example:</p> <pre>upperword="Name" upperword="Company" upperword="Address"</pre> |
| <code>lower=field</code> | <p>This option specifies that one or more fields must be automatically converted to all lower case (see “Forcing Input to Upper or Lower Case” on page 473). For example, the word <code>COMPUTER</code> would be converted to <code>computer</code>. You should have one <code>lower=</code> option for each field you want capitalized this way.</p> |

| Option | Description |
|---------------------------------------|--|
| required=field | <p>This option specifies that a field is required. If a required field is not entered (left blank) the procedure will not update the database. Instead, it will display an error message in the browser. The default message is shown below, but you can create your own custom message using the missingFieldError tag (see below).</p> <div data-bbox="832 469 1710 752" style="border: 1px solid black; padding: 10px; margin: 10px auto; width: fit-content;"> <p>The following required fields are missing:</p> <ul style="list-style-type: none"> ● City ● Phone <p>Please press the Back button on your browser to re-edit the form and input the missing fields.</p> </div> <p>You may specify as many required fields as you like. For more information on this option, see “Checking for Missing Fields” on page 473</p> |
| missingFieldError=msg | <p>This option controls the error message to display if a required field was not entered. You can insert a list (formatted) of the missing fields with the special <missing> tag. For example:</p> <pre data-bbox="716 1057 1755 1134">missingFieldError= {Please go back and enter these fields!<p><missing>}</pre> |
| autosave=yes/no | <p>This option controls whether the webformtodatabase statement saves the database to disk after it has been updated. If the database is a shared database this option should be left off (and will default to no). If the database is not shared the default is yes. See “Saving the Modified Database” on page 476 for more information.</p> |
| responsePage=html | <p>The WebFormToDatabase statement automatically generates a response page to display the result of the update to the user. This response is placed in the cgiHTML variable. The responsePage option allows you to control the content and appearance of this response page. If you don't want an automatic response page generated, set responsePage={-}. For a simple canned text message, just type in the message:</p> <pre data-bbox="716 1586 1738 1662">responsePage= {<center>Thank you for your submission.</center>}</pre> <p>The responsePage can include information about the update that was just performed - how many fields were updated, what fields were updated with what values, etc. Use these special tags to insert this information:</p> <p><fieldcount> - number of fields modified (example 6 fields) <fields> - list of fields/values, in ... list format <database> - name of the database that was updated <extras> - information about any extra form fields that were ignored (see the extraPages option below)</p> <p>Here is a simple example of how these special tags are used.</p> <pre data-bbox="716 2180 1852 2285">responsePage= {Thanks for your contribution to the <database> database! <p>Here's the data you entered:<fields>}</pre> |

| Option | Description |
|---------------------------------------|---|
| <p><code>extraPage=html</code></p> | <p>The <code>WebFormToDatabase</code> statement automatically scans the fields passed to it from the form and updates the corresponding database fields. There may, however, be additional fields in the web form that do not correspond to any database field. If this occurs, the response page can include this information to warn the user that some of the input was ignored. This extra response is placed in the <code><extras></code> tag of the responsePage (see above). If you don't want any error message to be displayed at all, set the <code>extraPage</code> option to a single dash</p> <pre>extraPage="-"</pre> <p>The <code>extraPage</code> option allows you to control the content and appearance of this portion of the response page. For a simple canned text message, just type in the message:</p> <pre>extraPage={<p>Warning: One or more form fields ignored!}</pre> <p>The <code>extraPage</code> can include information about the extra fields that were ignored. Use these special tags to insert this information:</p> <p><code><fieldcount></code> - number of fields that were ignored (example 6 fields) <code><fields></code> - list of ignored fields/values, in <code>...</code> list format</p> <p>Here is a simple example of how these special tags are used.</p> <pre>extraPage={<p>Warning: <fieldcount> ignored!}</pre> <p>Of course if your form and database are designed correctly you should never need to use this option. However, it can be useful for debugging your procedures.</p> |
| <p><code>notFoundError=msg</code></p> | <p>This option controls the error message to display if the find or select options don't find the requested record. For example:</p> <pre>notFoundError={The record you tried to update for doesn't exist!}</pre> |
| <p><code>toManyError=msg</code></p> | <p>This option controls the error message to display if the select options finds more than one record that matches the key field. For example:</p> <pre>notFoundError={Can't decide which record to update!}</pre> |

WebFormToDatabase — Displaying Form Items

For debugging you may want to simply display the names and values of the form items that are submitted. This can be easily done with a one line procedure.

```
webformtodatabase "whatrecord=formdump"
```

For example, suppose you have created a Panorama form like this:

The screenshot shows a web browser window titled "ColoradoHotels:NewHotel (100%)". The form has a blue header with the text "Add a new Colorado Hotel". The form fields are as follows:

- Hotel: 6 & 40 Motel
- City: Idaho Springs
- Phone: 567-2691
- Rate: 19.00
- Units: 30
- Rating: Excellent Good Fair Poor Unacceptable
- Notes: (empty text area)

At the bottom of the form are two buttons: "Submit" and "Reset". To the right of the form is a small image of a mountain landscape. The browser's status bar shows "Verdana 10pt x0pt y0pt &x0pt &y0pt".

Make sure that this form triggers the procedure ("[Custom Form Actions](#)" on page 254) and is uploaded to the web server. When the **Submit** button is pressed the server will respond with a list of the fields on this form along with their values.

The screenshot shows a web browser window titled "Panorama Database Query:ColoradoHotels/NewHotel". The content of the window is:

7 fields submitted to the *ColoradoHotels* database:

- Hotel=Wanderlust Inn
- City=Alamosa
- Phone=555-1212
- Rate=3
- Units=34
- Stars=3
- Description=Near center of town.

When the `formdump` option is used the database isn't actually modified — this option is strictly for debugging.

WebFormToDatabase — Adding a New Record

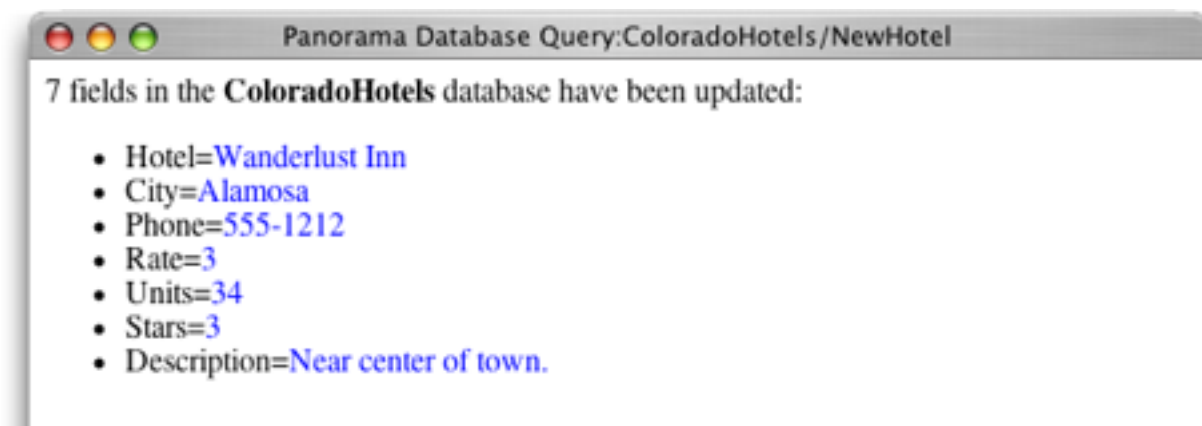
It's very easy to write a procedure to use a form to add a new record to the web database:

```
webformtodatabase "whatrecord=add"
```

In fact, this procedure can be even shorter!

```
webformtodatabase ""
```

When the **Submit** button is pressed the server will add a new record, then display a list of the fields that have been updated.



If the database is shared you can see the new record by synchronizing the database (see "[Synchronization](#)" on page 144).

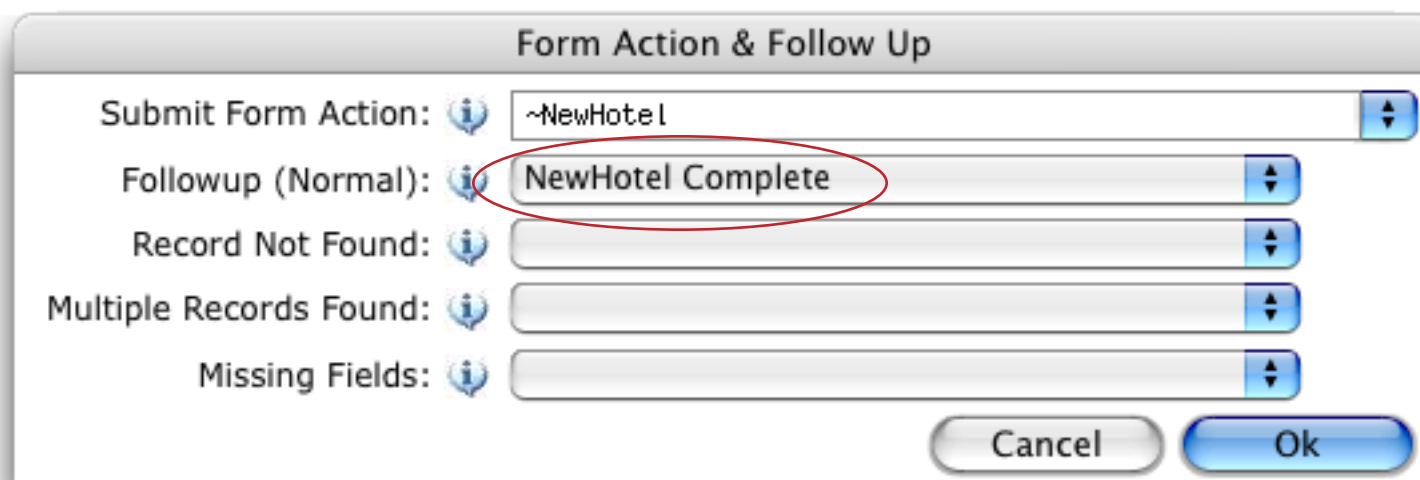
| Hotel | City | Rate | Units | Phone | St | Description |
|---------------------|------------------|--------|-------|----------|----|--------------------|
| Wildwood Inn | Aspen | 42.00 | 142 | 923-3550 | 3 | |
| Wind River Ranch | Estes Park | 67.00 | 20 | 586-4212 | 2 | |
| Workshire Lodge | Estes Park | 22.00 | 11 | 586-2198 | 2 | |
| Writers Manor Hotel | Denver | 50.00 | 325 | 756-8877 | 4 | |
| Yucca Lane Lodge | Colorado Springs | 30.00 | 31 | 636-3941 | 2 | |
| Hotel Jerome | Aspen | 295.00 | 94 | 920-1000 | 3 | Restored 1889 h |
| International Hotel | Denver | 129.50 | 34 | 555-9876 | 2 | A great hotel in c |
| Snowflake Inn | Aspen | 59.00 | 38 | 925-3221 | 4 | In Room: Cable T |
| Wanderlust Inn | Alamosa | 3.00 | 34 | 555-1212 | 3 | Near center of to |

443 visible/443 total

Setting up a Custom Response Page

As shown in the previous section the `webformtodatabase` statement will automatically generate a response confirming that the new data has been entered. However, this response page isn't very attractive. There are several methods for generating a custom response page.

If the original data entry form was a Panorama generated web form you can specify that another web form be used as the response page using the **Form Action & Follow Up** dialog (see "[Setting Up a Custom Response Page](#)" on page 243).



If a follow-up form has been specified then the `webformtodatabase` statement will automatically use it instead of the standard generic response.

If a followup form hasn't been set up, or if you want to use a different form or generate custom HTML simply program the response like any other web page (see "[Generating HTML](#)" on page 369). Here's an example that displays the web form **Thanks!** when data is submitted (see "[Generating a Page Using a Panorama Form Template](#)" on page 379).

```
webformtodatabase ""
cgiHTML=renderwebform("Thanks!","")
```

Another way to generate a custom page is to use the `responsepage` option. Here's a simple example.

```
webformtodatabase "responsepage={<center>Thanks!</center>}"
```

For more information about this option see the table above.

Forcing Input to Upper or Lower Case

Sometimes a particular field must be all upper case, or have the first letter of each word capitalized. For example, US state abbreviations are always capitalized (AK, AZ, ... WA) while city names always have the first letter capitalized (Anchorage, Bakersfield, Costa Mesa, etc.). The `webformtodatabase` statement can automatically capitalize data before it gets entered into the database.

```
webformtodatabase "whatrecord=add upperword="Hotel" upperword="City" upper="State"
```

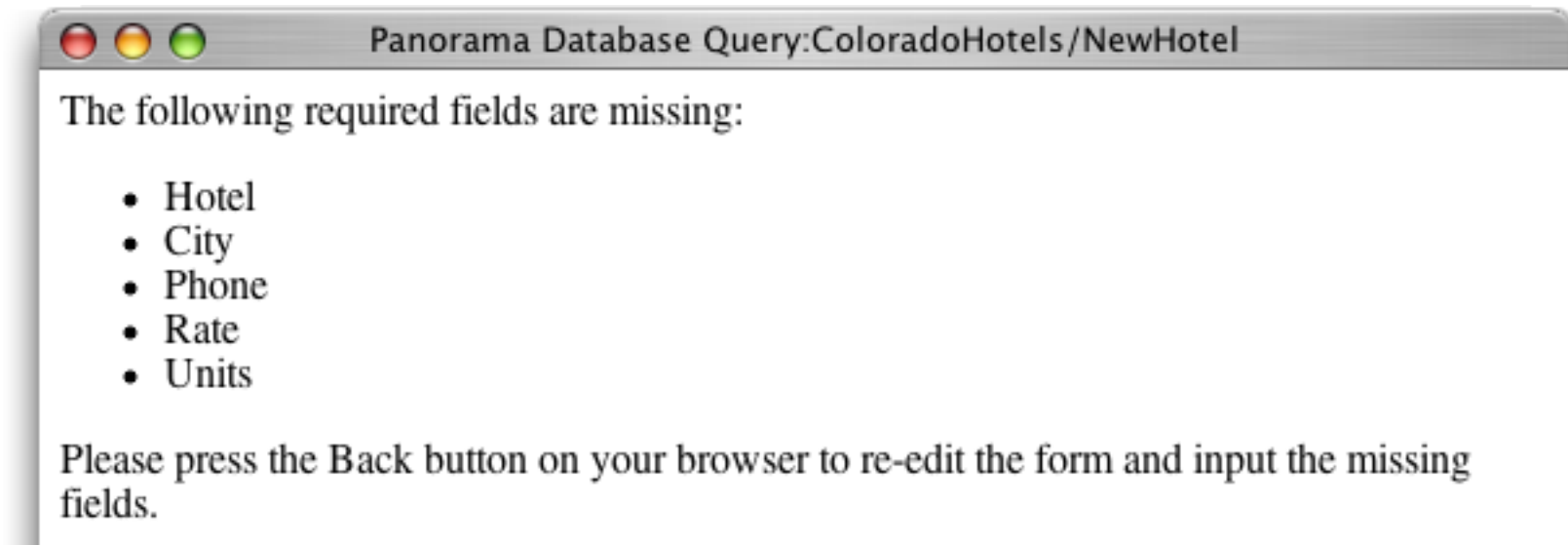
If data must be forced to all lower case use the `lower` option.

Checking for Missing Fields

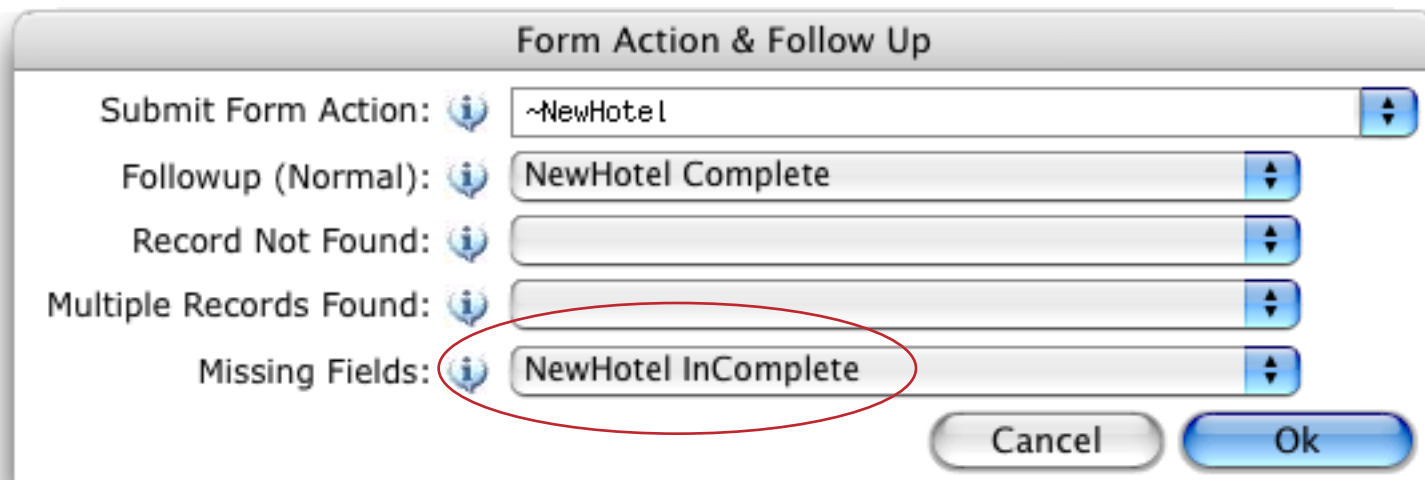
The `webformtodatabase` statement can check for missing fields — in other words if the user fails to enter something important (like their name) it can refuse to update the database and display a page requesting that they enter the missing information. To do this add one or more `required` tags with the field name, like this.

```
webformtodatabase "whatrecord=add required="Hotel" required="City"
required="Phone" required="Rate" required="Units" "
```

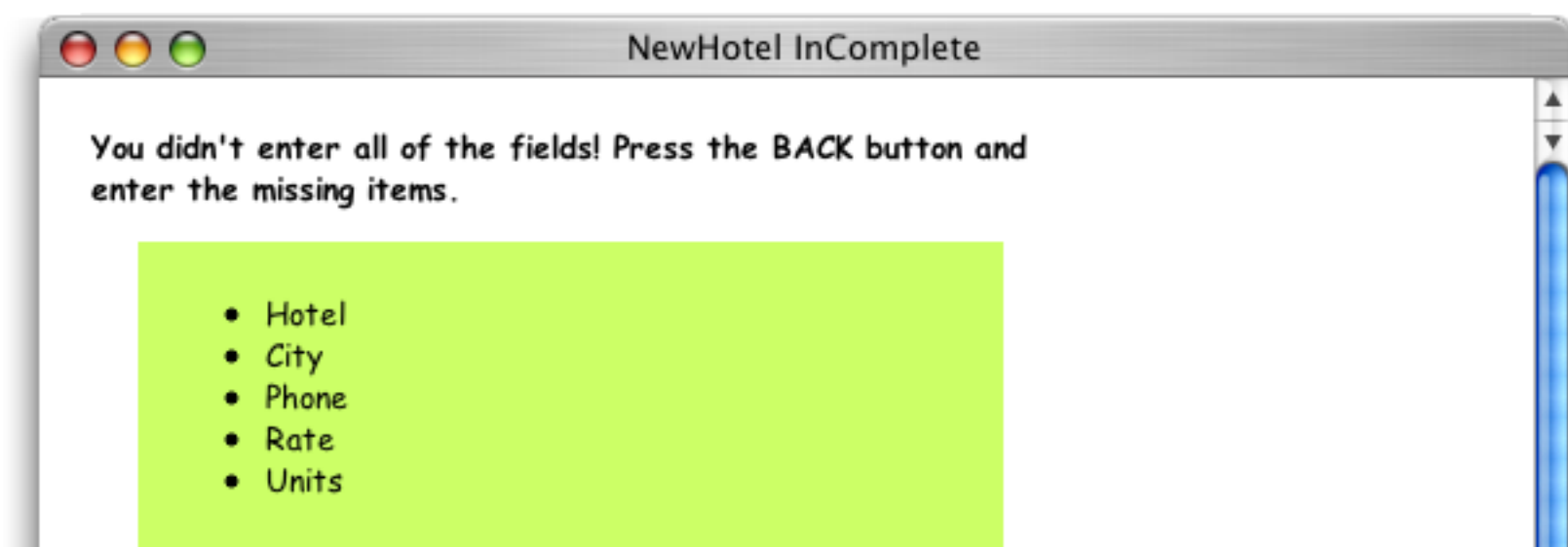
If you try to submit an empty form the server will complain:



The response shown above is the generic response from the server. You can also create a custom response page. If you are using a Panorama web form for data entry then one method for doing this is to create another Panorama web form for missing fields and then specify that form with the **Form Action & Follow Up** dialog (see "[Setting Up a Custom Response Page](#)" on page 243).




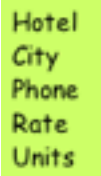


Now if there are any missing fields the designated form will appear (as shown below). Hopefully you can come up with a more graphically appealing form than this one!



This form uses `webformmissingfields()` function to display a list of the fields that are missing. This function has one parameter, a template that the function uses to format the list. The template should contain the tag `<field>`, and any other HTML tags you want to use to format the list. When the form is rendered the template will be repeated once for each missing field, with the name of the missing field substituted for the `<field>` tag. (Or you can use the template "", which produces a carriage return delimited array of the missing field names. If displayed as HTML this will appear as names separated by spaces.)

This table shows examples of different formulas using the `webformmissingfields()` function and the resulting display on the missing item web page (the green background is from a rectangle object and has nothing to do with these formulas).

| Formula | Result |
|--|---|
| <code>""+webformmissingfields("<field>")+"</code> |  |
| <code>replace(webformmissingfields("<field>, ")[1,-3],cr(),"")</code> |  |
| <code>"<table border=1>"+ webformmissingfields("<tr><td><field></td></tr>")+ "</table>"</code> |  |
| <code>webformmissingfields("")</code> |  |

If using the **Missing Fields** option in the **Form Action & Follow Up** dialog isn't appropriate (for example if you're not using a Panorama web form you can include logic in your procedure using the `webformmissingfields()` function to check for missing fields, and then take whatever action you like.

```
webformtodatabase "whatrecord=add required="Hotel" required="City"
  required="Phone" required="Rate" required="Units" "
if webformmissingfields("")=""
  cgiHTML=renderwebform("Thanks!","")
else
  cgiHTML=renderwebform("Error_MissingFields","")
endif
```

This example uses the `renderwebform()` function to display the response page, but you can use any of the techniques described earlier in this chapter to render the page (see "[Generating HTML](#)" on page 369).

Data Entry for Number and Dates

The `webformtodatabase` statement uses the `val()` function to convert numeric fields. There is no error message if an invalid number is entered, however, any characters after the first non-numeric field will be ignored.

The `date()` function is used to convert date fields. This means that you can use Smart Dates™ in your web forms! Very cool.

Saving the Modified Database

The `webformtodatabase` statement modifies the RAM copy of the database. To make the change permanent the database must be saved to the disk. There are several ways to do this. (Note: The following comments apply to all changes made to a web published database, whether the changes are made using the `webformtodatabase` statement or some other technique.

If the database is a shared database, you don't need to do anything at all! The change is immediately recorded on disk in the journal file, where it is safe even if server operation is interrupted for some reason (power failure, etc., see "[Handling Interruptions in Server Operation \(Crash Recovery\)](#)" on page 153). When the journal file reaches a certain size the server will automatically save the database itself. This cuts way down on disk activity since the entire database doesn't have to be saved each time any change is made.

If the database is not a shared database then it should be saved every time a change is made. You can simply include a `save` statement in the procedure itself, like this:

```
webformtodatabase ""
save
```

You can use an autosave option as part of the `webformtodatabase` statement, like this.

```
webformtodatabase "autosave=yes"
```

Finally, you can configure the database so that it is saved every time it is accessed from the web server (see "[Automatically save database after each access](#)" on page 196).

WebFormToDatabase — Modifying an Existing Record

To write a procedure that modifies an existing record the procedure must somehow specify which record to modify. One method to do that is to pre-locate the record with the `find` or `select` statements. This example will locate the record containing Apple Computer and update it with the information submitted from the form.

```
find Company="Apple Computer"
webformtodatabase "whatrecord=active"
```

Of course you probably don't want to create a specific form and procedure just for updating the Apple Computer record in your database! If the form contains a `Company` field the procedure could use the submitted company name to determine what record to update, like this:

```
local whatCompany
whatCompany=webformitemvalue("Company")
/* see "Accessing Form Item Values in a Formula" on page 435 */
find Company=whatCompany /* or you could use select here */
webformtodatabase "whatrecord=active"
```

There's a simpler way to do this. Instead of the 4 lines above, you can let the `webformtodatabase` statement locate the record for you. This example will do exactly the same operation as the example above.

```
webformtodatabase "whatrecord=find keyfield=Company"
/* or you could use whatrecord=select */
```

If necessary you can specify more than one key field. For example you might use the last name, company name, and zip code as key fields in a contact or mailing list database.

Handling Missing or Ambiguous Records

Using the `whatrecord=find` and `whatrecord=select` options generally relies on the user accurately typing in the key fields. For example, if someone types in `Apple` or `Apple Inc` instead of `Apple Computer`, the `find` will fail. Later we'll show a foolproof method to avoid this problem completely (see "[WebFormToDatabase — Modifying an Existing Record with Embedded Record ID](#)" on page 477), but for now let's look at methods to handle this problem when using the key field technique for updating records.

One way to handle the situation is to simply decide that the user knows best and to automatically add a new record if the submitted data doesn't already exist in the database. To do this use either the `whatrecord=find-add` or `whatrecord=selectadd` options. Here's how one the `findadd` option would be used.

```
webformtodatabase "whatrecord=findadd keyfield=Company"
```

The problem with this idea is that you'll quickly wind up with lots of extra records, so a more common approach is to not modify the database at all and display a page with an error message. In fact, that's what the `webformtodatabase` statement will normally do when the `whatrecord=find` and `whatrecord=select` options are used. You can customize the error page with the `notfounderror` option, like this:

```
webformtodatabase "whatrecord=findadd keyfield=Company
  notfounderror={<h1>Sorry, that company isn't in the database!</h1>}"
```

If the `whatrecord=select` option is used then another possible error is that more than one record matches the key fields (for example if the database contains two or more records for `Apple Computer`). In that case a different error message is displayed, which can be customized with the `toomanyerror` option.

```
webformtodatabase "whatrecord=findadd keyfield=Company
  notfounderror={<h1>Sorry, that company isn't in the database!</h1>}
  toomanyerror={<h1>Hey - there is more than one record for this company</h1>}"
```

For more control you can program a response to missing or ambiguous records. The `cgiSelectedRecordCount` global variable will contain 1 if the update was successful, 0 if the record could not be found, or 2 (or greater) if there were multiple records found. This example will display a Panorama web form if one of these errors occurs.

```
webformtodatabase "whatrecord=select keyfield=Company"
if cgiSelectedRecordCount=0
  cgiHTML=renderwebform("CompanyNotFound","")
endif
if cgiSelectedRecordCount>1
  cgiHTML=renderwebform("CompanyDuplicates","")
endif
```

WebFormToDatabase — Modifying an Existing Record with Embedded Record ID

The technique for modifying an existing record in the previous section relies on the user typing in one or more correct key fields to identify the record. A more reliable method is to embed record ID information in the data entry form as it is being displayed. This is done automatically if the form is generated by the `htmldatatable` statement (see "[Linking Individual Table Rows to Detail Pages](#)" on page 421). If the form is generated some other way you'll need to use the `webformrecordid` statement to embed the record ID into the form being generated. If this database is shared, the only parameter needed by this statement is the name of the field containing the HTML for the page. Here's an example that displays a record in a form. The record that will be displayed is determined by the extra parameters in the URL.

```
weburlselect cgiExtraParameters /* see "What Record Are We Talking About?" on page 395 */
cgiHTML = renderwebform("Invoice","")
webformrecordid cgiHTML /* add record id to form about to be displayed */
```

The user can now see the current information for the selected record in their web browser, then edit that information and press the **Submit** button. Here's the procedure that processes the submitted input and updates the record:

```
webformtodatabase "whatrecord=record_id"
```

That's it! The server will locate and update the record (unless it has been deleted, which can be handled by checking the `cgiSelectedRecordCount` variable as described in the previous section).

If the database is not shared you'll need to add additional parameters to the `webformrecordid` statement to identify one or more "key" fields. (Do you get the idea that it's really a good plan to use shared databases for web publishing, avoiding lots of extra hassle?) The combination of all the key fields you specify should uniquely identify a single record in the database. For example in an invoice database the `InvoiceNumber` field probably uniquely identifies a record.

```
weburlselect cgiExtraParameters /* see "What Record Are We Talking About?" on page 395 */
cgiHTML = renderwebform("Invoice","")
webformrecordid cgiHTML,"InvoiceNumber" /* add record id to form about to be displayed */
```

If a single field won't reliably identify a single record you can add more, as many as are necessary.

```
weburlselect cgiExtraParameters /* see "What Record Are We Talking About?" on page 395 */
cgiHTML = renderwebform("AddressUpdateForm","")
webformrecordid cgiHTML,"LastName","FirstName","LastName","City","State"
```

No matter how many key fields there are the procedure for processing the submitted form data doesn't change, in fact it is the same as the procedure for a shared database.

```
webformtodatabase "whatrecord=record_id"
```

Again, this procedure uses the embedded record ID to locate and update the correct record.

Manually Creating Web Forms for use with the WebFormToDatabase Statement

If you let Panorama create your web forms for you it will automatically set up all the fields in the correct format to be processed by the `WebFormToDatabase` statement. If you create your web forms externally (for example using Dreamweaver or BBEdit) you'll need to make sure that the form is created using the exact format required by the Panorama server. See "[Rendering Using an External Text File containing a Form as a Template](#)" on page 484 for the exact details on how to do it. It's important to get all the details exactly right or the `WebFormToDatabase` statement may not be able to process the submitted data.

Web Form Based Data Selection

Like the `WebFormToDatabase` statement, the `WebFormSelection` statement processes a form submitted over the web from a browser. Instead of using the submitted information to update the database, the `WebFormSelection` statement finds or selects a subset of the database. Often the `WebFormSelection` statement is used prior to the `htmldatatable` statement to display a subset of the database in a table (see “[What Records are we Talking About? \(The WebSelect Statement\)](#)” on page 413).

To use the `WebFormSelection` statement you must set up a form for searching. This form usually contains several fields with the same names as database fields. Here’s a typical example.

The screenshot shows a web browser window titled "Advanced Search" with the URL `http://192.168.1.59/cgi-bin/Panorama.cgi?MailingL`. The page features the "World's Greatest Hobby" logo and a search form. The form is titled "Advanced Search" and includes the instruction "Enter the data you want to search for:". The fields are as follows:

- First Name:
- Last Name:
- Address:
- City:
- State:
- Zip Code:
- Phone:
- Email:
- Scale:
- Era:
- Railroad:
- Notes:

At the bottom of the form are two buttons: "Search" and "Reset".

An alternate technique is to set up a form with a single field named `SearchAllFields`.

The screenshot shows a web browser window titled "Mailing List" with the URL `http://localhost/cgi-bin/Panorama.cgi?MailingList~forn`. The page features the "World's Greatest Hobby" logo and a search form. The form is titled "Mailing List" and includes the text "12,147 records.". The search input field contains the text "charles". To the right of the input field is a "Search" button and a link to "Advanced Search". Below the search field is a "Sign Up" link and the text "Please add me to your mailing list.".

Either way the form must be set up with a submit action that will trigger a procedure that contains a `webformselection` statement (see “[Custom Actions](#)” on page 222). This statement processes the fields submitted from the form and scans the database to find records that match the submitted data. In it’s simplest form the `webformselection` statement can be used with no options at all, like this:

```
webformselection {}
htmldatatable myTableOptions
```

This example displays the selected records in a list. For example, if the user had typed **Boulder** in the City field a table something like this would be displayed.

| Name | City | Scale | Era | Railroad |
|-----------------------------------|------------------|-------|------------|-------------------------------|
| Joseph Irwin | Boulder, CO | N | Modern | Southern Pacific (SP) |
| Harold Donnell | Boulder, CO | HO | Transition | Southern Pacific (SP) |
| Dale McGuire | Boulder, CO | HO | Transition | Santa Fe (ATSF) |
| Craig Ott | Boulder, CO | HO | Transition | New York Central (NYC) |
| Harry Dickson | Boulder, CO | HO | Transition | Conrail |
| Peter Rodriguez | Boulder, CO | HO | Steam | Norfolk Southern (NS) |
| Lawrence Goodrich | Boulder, CO | G | Transition | Santa Fe (ATSF) |
| Marcus Silva | Boulder, CO | HO | Transition | Illinois Central (IC) |
| Michael Allen | Boulder, CO | G | Transition | Union Pacific (UP) |
| Joseph Cohen | Boulder, CO | O | Modern | Chicago & North Western (CNW) |
| Mark Fong | Boulder, CO | HO | Transition | Southern |
| George Derewianko | Boulder, CO | HO | Transition | Nickel Plate (NKP) |
| Alfred Lamb | Boulder City, NV | N | Transition | Conrail |
| Douglas Dockery | Boulder, CO | HO | Transition | Illinois Central (IC) |
| Steven Hagen | Boulder, CO | HO | Modern | Southern Pacific (SP) |

By default the `webformselection` statement takes each entered item and checks for records that contain that text (in the example above records where the City contains **Boulder**. If more than one item is entered then only records that contain both will be selected, for example only records where the City contains **Boulder** and the Railroad contains **Santa Fe**. If a field is numeric then = is used instead of contains, for example if the price specified is 9 then only prices that are exactly 9.00 will match, not 99, 19 or 0.9. By adding options to the `webformselection` statement you can change the way the statement looks for matching records. The table

The `WebFormSelection` statement has one parameter, a list of options that customize the way the statement looks for matching records. In the simple example above there were no options specified at all — the example simply used the default behavior of the `WebFormSelection` statement. Add one or more of the options listed below to customize the selection.

| Option | Description |
|-------------------------------|--|
| <code>do=statement</code> | <p>This option controls the statement used to find or select data. There are five choices.</p> <p><code>select</code> <code>selectwithin</code> <code>selectadditional</code> <code>find</code> <code>findall</code></p> <p>The default is <code>select</code>. Use <code>selectwithin</code> or <code>selectadditional</code> if you want to combine the new selection with a previous selection in the same procedure. The <code>findall</code> option does a <code>selectall</code> first, then does the <code>find</code>.</p> |
| <code>multiple=and/or</code> | <p>This option controls how multiple fields will be combined in the selection formula. The default is <code>and</code>.</p> |
| <code>compare=operator</code> | <p>This option specifies the comparison operator that will be used in the selection formula. Any Panorama comparison operator can be used, including <code>=</code>, <code><</code>, <code>></code>, <code>≤</code>, <code><=</code>, <code>≥</code>, <code>=></code>, <code>≠</code>, <code><></code>, <code>contains</code>, <code>match</code>, <code>beginswith</code>, <code>endswith</code>, and <code>soundlike</code>. The default is <code>contains</code>.</p> <p>For numeric or date fields, only the <code>=</code>, <code><</code>, <code>></code>, <code>≤</code>, <code><=</code>, <code>≥</code>, <code>=></code>, <code>≠</code> and <code><></code> operators are legal. If you specify one of the other operators, <code>=</code> will be used for any numeric or date fields. By the way, Panorama's Smart Date™ feature will work with any date fields.</p> |

To illustrate these options, suppose a user submitted a search form with these values:

With no options specified the `WebFormSelection` statement will select all records where the last name contains `wilson` and the state contains `CA`.

If a compare option is added like this:

```
webformselection {compare="="}
```

the statement will select all records where the last name equals `wilson` and the state equals `CA`. Since probably there are no `wilson`'s in the database, only `Wilson`'s, this search will probably turn up nothing. A better search would be

```
webformselection {compare="match"}
```

The multiple option controls how multiple items interact. Here's a modified procedure that uses `or` instead of `and`.

```
webformselection {multiple="or"}
```

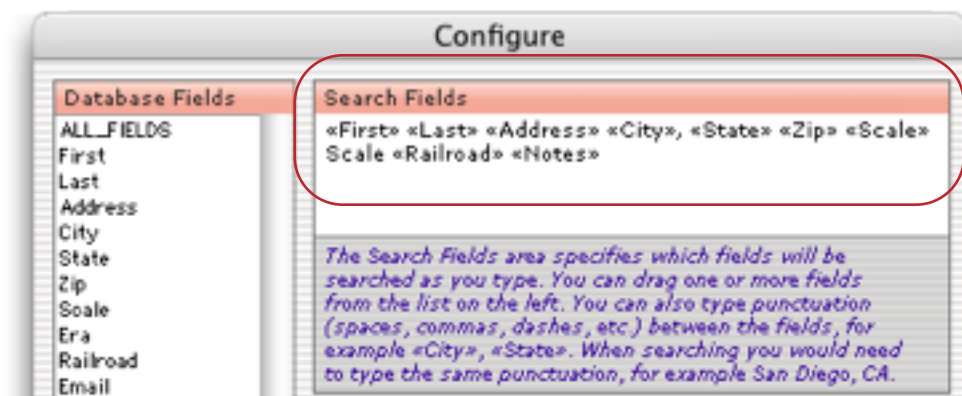
This statement will select all records where the last name contains `wilson` *or* the state contains `CA`., in other words, everyone named Wilson and everyone in California.

Searching All Fields

If the `WebFormSelection` statement detects a field named `SearchAllFields` in the submitted data it works differently. In that case it ignores any other fields and only processes the `SearchAllFields` field. It searches the database looking for any records that contain the contents of the `SearchAllFields` field in any database field. Here's an example of such a form.



If you've used the Live Clairvoyance wizard to set up custom search fields for the database (shown below) the `WebFormSelection` statement will use this template to search the database instead of searching every field.



See the documentation for the Live Clairvoyance wizard for more information.

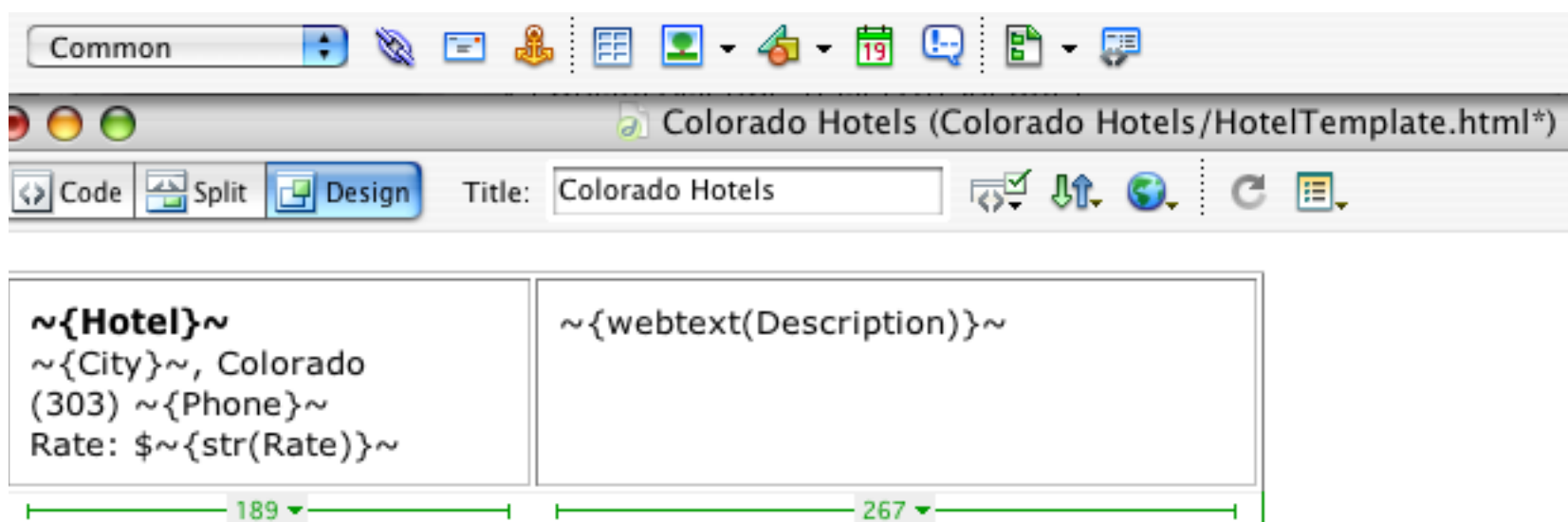
Rendering Using an External Text File as a Template

The rest of this manual assumes that you will create web forms with Panorama's built-in form editor ("[Converting a Panorama Form into a Web Form](#)" on page 231). When you take this approach Panorama will take care of many of the details for you. However, it is possible to use a separate program for designing web forms — Dreamweaver, GoLive, BBEdit, etc. The `webmerge` and `webformmerge` statements allow you to use your favorite editor to build templates for pages generated by the Panorama server. To do this you'll need to upload the text files created by your editor onto the server. If the web page is only used for displaying data you can use the `webmerge` statement. If the web page contains an HTML form with data that can be edited and sent back to the server you'll need to use the `webformmerge` statement.

The `webmerge` statement has three parameters: *folder*, *template*, and *result*.

| Parameter | Description |
|-----------|---|
| Folder | This is the location of the template file, or "" if the template is in the same folder as the database (the Public Database folder). If the template file is in a subfolder of the Public Database folder use the <code>dbsubfolder(</code> function, for example <code>dbsubfolder("Dreamweaver")</code> . If the template file is in a subfolder of Apache's Document folder (the folder containing the web site's home page) use the <code>webhomesubfolder(</code> function, for example <code>webhomesubfolder("Dreamweaver")</code> . |
| Template | This is the name of the template file. (You can instead put the actual template text itself in this parameter. If the folder is "" and this parameter contains the text <code></</code> Panorama will assume that this text is the actual template, rather than the name of the template file.) |
| Result | This is the name of the variable to receive the merged text. If this parameter is omitted then the text will be placed in <code>cgiHTML</code> . |

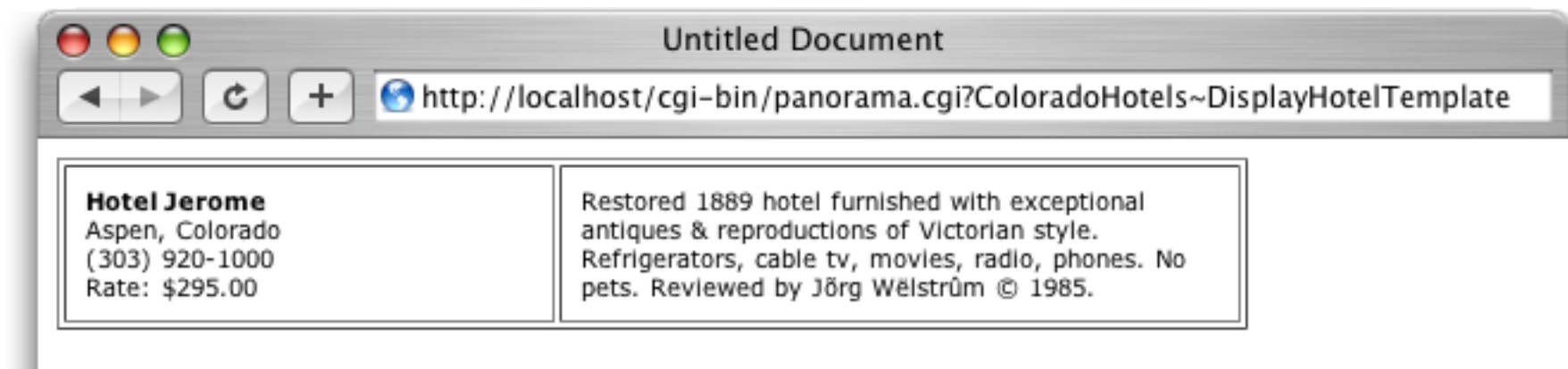
Within the template file you can place Panorama formulas by enclosing them with `~{` and `}~`. The example below, created with Dreamweaver, shows a web template with five formulas embedded in the page.



The template file can be stored anywhere on the server, but it is usually most convenient to upload it into a subfolder of the web server's home folder (along with all of the other HTML files that make up your web site). This simple one line procedure will merge the fields into the template.

```
webmerge webhomesubfolder("Hotels"), "HotelTemplate.html"
```

Here's the final result displayed in the browser.



When you use a template created in an external editor you cannot customize the HTML header using the variables described in “[Customizing the HTML Page Header](#)” on page 373 (cgiPageTitle, etc.). These options must be set up in the external editor (Dreamweaver, etc.)

Rendering Using an External Text File containing a Form as a Template

The previous sections described how to render a fixed, non-interactive web page. In this section we'll show how to render an externally created form which can not only display fields and variables, but can also allow this data to be edited and sent back to the server. This is done with the `webformmerge` statement, which like the `webmerge` statement merges fields and variables into a template. However, in addition to looking for embedded formulas, the `webformmerge` statement works by locating all the form element tags (<input>, <hidden>, <textarea>, <select>, etc.) in the template and adjusting them to display the fields and/or variables.

When you create the template you must make sure that each form element name is the same as the name of a field or variable in the database (including upper or lower case). You also need to format your form element tags according to the rules listed below. If these tags are not set up “just so” the `webformmerge` won't work. It's not difficult, but you do have to follow the rules. (Keep in mind that all of the rules below are for forms that you create by hand using an external editor. If you use Panorama to create your web forms all of this is done automatically for you.)

Text input fields must be set up like this:

```
<input type="text" name="Field" value="" size=20>
```

In particular, the highlighted (red) portion of the tag must be exactly in this format. The `name=` parameter must appear just before the `value=` parameter, with one space in between. Both `name=` and `value=` must be in all lower case. You cannot omit `value=""` parameter. There must be quotes around the field name, and the value must be `""`. The arrangement of the other options doesn't matter.

Checkboxes and radio buttons must be set up like this:

```
<input type="checkbox/radio" name="Field" value="value"> value
```

Again, the highlighted (red) portion of the tag must be exactly in this format. The `name=` parameter must appear just before the `value=` parameter, with one space in between. Both `name=` and `value=` must be in all lower case. There must be quotes around the field name and the value. Do not include a `checked` option in the tag - the `webformmerge` statement will do that for you. The arrangement of the other options doesn't matter.

Multiline input fields must be set up like this:

```
<textarea cols=40 rows=6 name="Field"></textarea>
```

Once again the highlighted (red) portion of the tag must be exactly in this format. The `name=` parameter must appear at the end of the tag, right before the `>`, and it must be quoted, and `name=` must be in lower case. There must be no blanks between the `>` and `</textarea>`. The arrangement of the other options doesn't matter.

Pop-up menus and scrolling lists require a slightly non-standard format. The `<select>` tag is the same as normal, but the `<option>` tags must be formatted like this:

```
<name="Field" option>value
```

The `name=` must appear just before `option`, with one space in between. Both `name=` and `option` must be in all lower case. There must be quotes around the field name. There must be a carriage return after the value. Here is an example of a pop-up menu that will work with the `webformmerge` statement.

```
<select size=1 name="Color">
  <name="Color" option>Red
  <name="Color" option>Blue
  <name="Color" option>Green
  <name="Color" option>Orange
  <name="Color" option>Purple
  <name="Color" option>Yellow
  <name="Color" option>Gold
  <name="Color" option>Silver
</select>
```

Here is an example of a complete form template. This template was created in BBEdit, and includes all seven **ColoradoHotels** fields in the form. In the illustration below we've highlighted the form element tag for each of these seven fields with a green box.

```

<html>
<head>
<title>Colorado Hotel Information</title>
</head>
<body>

<font size=+2><b>Add/Update hotel information</b></font></br>
<hr width=400 align=left>

<form method="post" action="/cgi-bin/panorama.cgi?ColoradoHotels~UpdateAddHotel">

<ul>
<li><input type="radio" name="UpdateAdd" value="Update"> Update
<input type="radio" name="UpdateAdd" checked value="Add"> Add
<hr align=left width=100>
<p>

<li><b>Hotel</b><br>
<input type="text" name="Hotel" value="" size=50><p>

<li><b>City</b><br>
<input type="text" name="City" value="" size=30><p>

<li><b>Phone</b><br>
<input type="text" name="Phone" value="" size=30><p>

<li><b>Rate</b><br>
<input type="text" name="Rate" value="" size=12><p>

<li><b>Units</b><br>
<input type="text" name="Units" value="" size=12><p>

<li><b>Stars</b><br>
<input type="radio" name="Stars" value="1"> 1
<input type="radio" name="Stars" value="2"> 2
<input type="radio" name="Stars" value="3"> 3
<input type="radio" name="Stars" value="4"> 4
<input type="radio" name="Stars" value="5"> 5
<p>

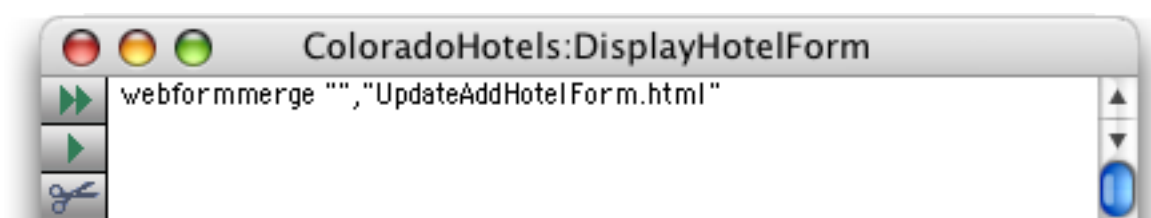
<textarea rows=5 cols=50 wrap=virtual name="Description"></textarea>
<p>

<input type="submit" value="Submit New Hotel"> <input type="reset">
</ul>
<p>
<hr width=400 align=left>
</form>

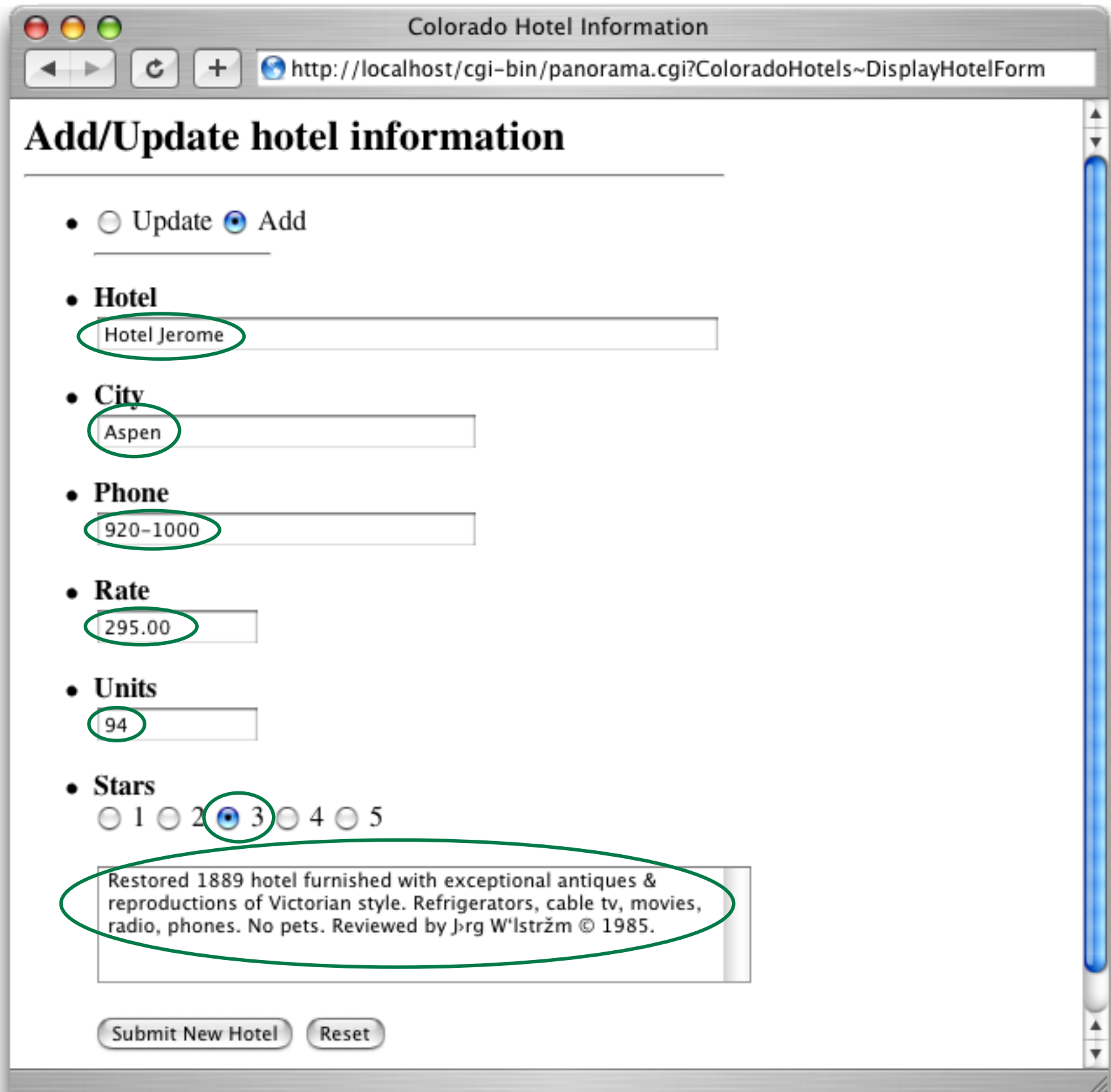
</body>

```

This simple procedure will fill in the form with the data for the current hotel. In this case the HTML template file is in the Public Database folder, the same folder as the database itself.



Here's what this page looks like when it is rendered. The database values have been highlighted.



Colorado Hotel Information

http://localhost/cgi-bin/panorama.cgi?ColoradoHotels~DisplayHotelForm

Add/Update hotel information

- Update Add
- Hotel**
- City**
- Phone**
- Rate**
- Units**
- Stars**
 1 2 3 4 5

Restored 1889 hotel furnished with exceptional antiques & reproductions of Victorian style. Refrigerators, cable tv, movies, radio, phones. No pets. Reviewed by Jrg W'istržm © 1985.

If you use the browsers **View Source** command you can see how the form element tags have been modified to include the actual values from the database fields.

```

Source of http://localhost/cgi-bin/panorama.cgi?ColoradoHotels~DisplayHotelForm

<form method="post" action="/cgi-bin/panorama.cgi?ColoradoHotels~UpdateAddHotel">

<ul>
<li><input type="radio" name="UpdateAdd" value="Update"> Update
<input type="radio" name="UpdateAdd" checked value="Add"> Add
<hr align="left" width="100">
<φ>

<li><b>Hotel</b><br>
<input type="text" name="Hotel" checked value="Hotel Jerome" size="50"><φ>

<li><b>City</b><br>
<input type="text" name="City" checked value="Aspen" size="30"><φ>

<li><b>Phone</b><br>
<input type="text" name="Phone" checked value="920-1000" size="30"><φ>

<li><b>Rate</b><br>
<input type="text" name="Rate" checked value="295.00" size="12"><φ>

<li><b>Units</b><br>
<input type="text" name="Units" checked value="94" size="12"><φ>

<li><b>Stars</b><br>
<input type="radio" name="Stars" value="1"> 1
<input type="radio" name="Stars" value="2"> 2
<input type="radio" name="Stars" checked value="3"> 3
<input type="radio" name="Stars" value="4"> 4
<input type="radio" name="Stars" value="5"> 5
<φ>

<textarea rows="5" cols="50" wrap="virtual" name="Description">Restored 1889 hotel furnished with
exceptional antiques & reproductions of Victorian style. Refrigerators, cable tv, movies, radio,
phones. No pets. Reviewed by J'rg W'lstr'zm @ 1985.</textarea>
<φ>

<input type="submit" value="Submit New Hotel"> <input type="reset">
</ul>
<φ>
<hr width="400" align="left">
</form>

```

Remember, if you use Panorama's graphic editor to create your web forms you won't have to worry about any of these details. Panorama will automatically build the tags for you in the proper format.

Chapter 12: Enterprise Sharing vs. Butler



This chapter is written for Panorama users transitioning from Butler to the new Panorama Enterprise Server system. It describes the similarities and differences between the two systems, and explains how to convert databases shared with Butler into the new system.

System Architecture

The basic architecture and operation of the Panorama Enterprise system the same as the original Panorama/Butler system first introduced in 1996. Both systems use a distributed approach with multiple copies of the database on the server and each client. Both systems perform data browsing, searching, sorting, calculating and printing on the local RAM based copy of the database, while using the server for record locking and synchronization. See “[Database Sharing Concepts and Operation](#)” on page 18 for a detailed description of this system.

The difference between Panorama/Butler and Panorama Enterprise is in the implementation of this architecture. The Butler system uses a disk based SQL database on the server, while Enterprise uses a RAM based server on both the client and the server, with a journalling system to prevent data loss if there is any kind of service interruption (power failure, crash, etc.). Switching to a RAM based server has three significant advantages: speed, easier database setup, and more flexible data storage.

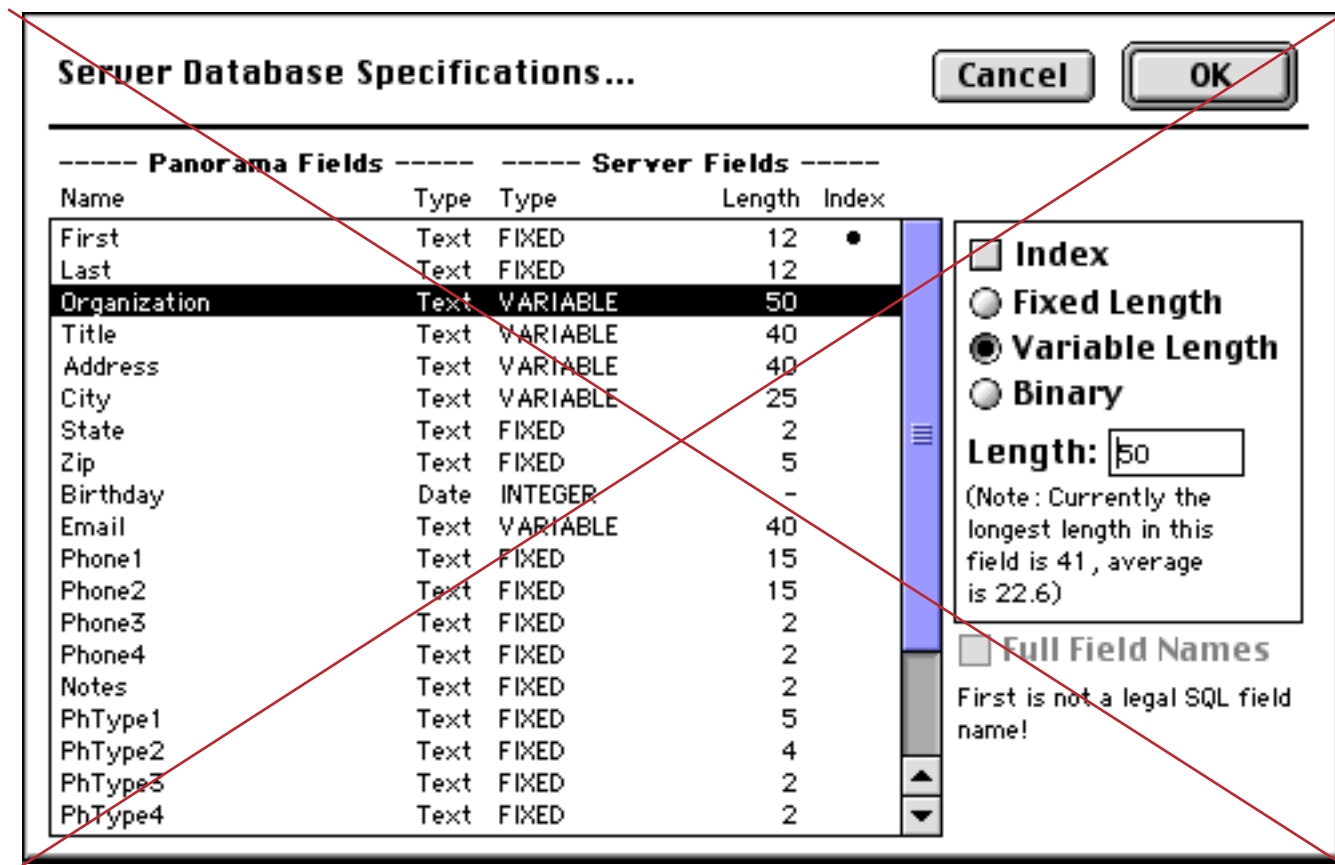
Speed

Because it is RAM based, the Panorama Enterprise Server is much faster than Butler, especially for mass operations like creating new shared databases, synchronizing, importing and appending records. For these operations the Enterprise Server gains speed not only because it is RAM based but because it can handle large numbers of records as a single unit. For example, when creating a new shared database Butler would have to handle each record separately, sometimes resulting in conversion times measured in hours. The Panorama Enterprise Server processes the entire database as a single unit, eliminating the per record overhead.

In a single user Panorama database the Formula Fill operation is blazing fast, but in Butler this operation slows to a crawl as each record is individually processed. The Panorama Enterprise Server still requires each record to be individually processed, but is faster because it is RAM based. However, with a slight rewrite most procedures that do formula fills can be dramatically accelerated using the `serverformula` fill statement (see “[ServerFormulaFill — A Much Faster Option for Select/Formula Fill Operation](#)” on page 163). While not quite as fast as a single user formula fill it is in the same ballpark.

Database Configuration

SQL disk based databases require more advance setup than Panorama's RAM based databases. You need to decide which fields should be indexed, and pre-define maximum field lengths. Panorama 3 and 4 have a complicated dialog box for configuring these options, which you are undoubtedly familiar with:



Good news! This dialog is completely eliminated in the Panorama Enterprise system. Since the same RAM based database structure is used on both the client and the server there is no advance configuration required at all. Converting from a single user to shared database takes only seconds (plus the time needed to transfer the data to the server, which, as described above, is much less than when using Butler).

Data Storage Flexibility

Panorama is very flexible in what it allows in database fields. Any text field can contain a virtually unlimited amount of text (up to 4 megabytes) and can also contain binary (non ASCII) data, including nulls and other non-printable data. Numeric fields can differentiate between zero and an empty cell.

Unfortunately, many SQL databases aren't nearly as flexible. The Butler server required that you specify a maximum field length in advance, and could not store binary data in a field. It also could not store an empty numeric cell (only zero). These restrictions caused problems for many existing single user Panorama databases.

Since the Enterprise Server uses a Panorama RAM based database on the server as well as the client, all of these restrictions are lifted. The Panorama Enterprise system is just as flexible about what you put in database fields as the single user version of Panorama. It allows unlimited length text, binary values in a text field, and both zero and empty numeric values.

Client Subsets

The Panorama/Butler system allowed you to create "subsets" where some or all clients would only contain a subset of the entire database, with only the server containing the entire database. The Panorama Enterprise Edition Server does not support subsets -- all clients use the entire database.

Converting from Butler to Enterprise Edition Server

To convert a database from Butler to Panorama Enterprise the first step is to convert the database into a single user database. To do this, open the database with Panorama 3 or 4. Open the Design Sheet and then use the **Permanently Detach Server** command (in the **Server** menu) to convert the database back into a single user database. (You can do this even if you no longer have a Butler server available.) Once this is done save the database and shut down (quit) Panorama 3 or 4. Then open Panorama 5.5 (or later) and convert the database to shared using the **Database Sharing Options** wizard (see “[Creating a Shared Database](#)” on page 103).

Reprogramming your Application

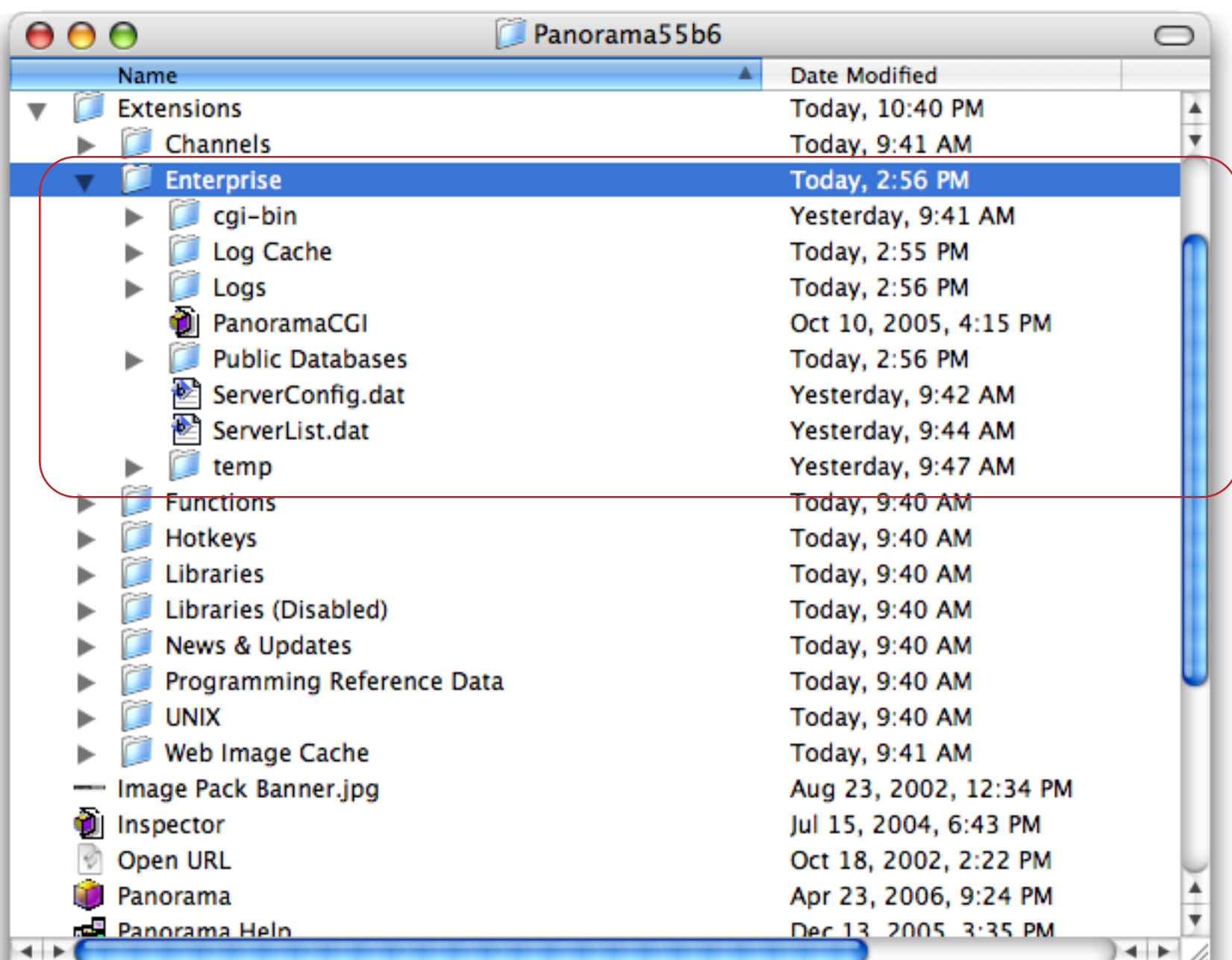
Because the most common Butler programming statements also work in Enterprise, 99.99% of Butler shared databases will work with Panorama Enterprise server with no changes. The **LockRecord**, **LockOrStop**, **UnlockRecord**, **ServerUpdate**, **ServerLookup**, and **ServerTimeout** statements all work exactly the same as they did under Butler, as do the `info("servertimeout")` and `info("serverstatus")` functions.

The Panorama Enterprise server also includes a number of new programming capabilities, including the ability to set up server variables, the ability to find out what other users are currently using a database, the ability to do fast formula fills, the ability to find out what records are currently locked and more. See “[Saving the Database](#)” on page 152 for more information.

Chapter 13: Secrets of the Enterprise Folder



Panorama Enterprise has a special folder it uses for storing configuration information, databases, logs, and other data. This folder is inside the Extensions folder which is inside the Panorama application folder.



In general, you shouldn't need to mess with this folder or even look at it. However system administrators may benefit from some knowledge of this folder, especially if there is some kind of problem.

cgi-bin

This folder is part of the Panorama distribution, don't modify or remove the contents of this folder. If the contents of this folder are disturbed you won't be able to enable web publishing or internet sharing without re-installing the server.

Log Cache

On client machines this folder contains copies of logs downloaded from server (see "[Monitoring Server Logs](#)" on page 82). It's ok to delete old log files at any time.

Logs

This folder only appears on the server computer. It contains the original log files (see "[Monitoring Server Logs](#)" on page 82). It's ok to delete old log files at any time.

PanoramaCGI

This file is part of the Panorama distribution. It provides compatibility with databases created with the old OS 9 WebSTAR/PanSTAR CGI system.

Public Databases

This folder only appears on the server computer. It contains all of the shared and web published databases, along with configuration information. This folder contains three types of files.

| Type | Extension | Description |
|-----------------|--------------|--|
| Config | .cfd | Contains database configuration information. Every public database (shared or web published) must have a corresponding .cfd file. You cannot edit these files. |
| Shared Database | .ees | Contains a Panorama database. In an emergency, this database (or a copy) can be opened by a regular copy of Panorama by change .ees to .pan. You should remove the database (or the copy) from the Public Databases folder before you do this. |
| Web Database | .pan or none | Contains a normal Panorama database. |

Note: Shared databases can also be used for web publishing.

Databases are normally deleted from the server using the **Server Administrator** wizard (see "[The Pop-up Database Context Menu](#)" on page 72). You can also delete a database from the server manually by simply dragging it into the trash (or any other location). If you do this, be sure to remove both the database itself and the corresponding .cfd file. If you do delete a database from the server this way we recommend that you do so when the server is not running or when you are otherwise sure that the database is not currently in use.

ServerConfig.dat

This text file contains server configuration information. It is normally modified with **Server Administrator** wizard. If you modify this file with a text editor we recommend that you keep a backup copy in case your changes cause a problem. You will need to shut down and restart the server to cause your changes to take effect.

ServerList.dat

This file contains information used to connect to internet based servers. The information in this file is managed by the **Available Servers** wizard (see "[Opening a Shared Database on the Internet](#)" on page 116) and should not be edited or modified.

temp

This folder contains temp files used by server. You can erase the contents of this folder if neither client or server is currently running on this machine (normally it should be empty anyway in that situation).

