



Witango Studio 5.5 User Guide

OS X

August 2003

With Enterprise Pty Ltd
Level 1,
44 Miller Street,
North Sydney, NSW, 2060
Australia

Telephone: +612 9460 0500

Fax: +612 9460 0502

Email: info@witango.com

Web: www.witango.com



Table of Contents

I	Table of Contents	I
Introduction I		
Using Witango Studio 3		
2	Witango Studio Basics	5
	Witango Studio Interface Components	6
	Viewing Interface Components	7
	Contextual Menus	7
	Properties Window	8
	HTML Editing Window	9
	<@INCLUDE> in HTML Editing Windows	16
	Working With Multi-column Lists	19
	The SQL Query Window	22
	Finding and Replacing Text	27
	Keyboard Shortcuts	34
	Windows Menu Shortcuts	34
	Expanding and Collapsing Parent Objects	35
	Witango Actions	36
	Working With Actions	39
	Adding an Action	40
	Naming an Action	41
	Deleting an Action	42
	Editing an Action	42
	Moving an Action	43
	Copying an Action	43
	Contextual Action Menu	45
	Action Properties	45
	Assigning Attributes to Actions	46
	Adding HTML (Results Action)	52
	Presentation Action	52
	Using Witango Application Files	55
	XML Format	55
	Application File Window	56
	Creating an Application File	57
	Saving an Application File	58
	Saving a Witango Application File or Witango Class File as Run-Only	59
	Executing Application Files	60
	Debugging Files	61
	Turning Debug On	61

Viewing Debug	62
3 Using Projects and Source Control	65
Basics of Witango Projects	66
Understanding the Project File	67
Using the Project Workspace	67
Creating a New Project	68
Adding a Folder to a Project	70
Adding Files to a Project	71
Removing Files and Folders From a Project	72
Opening and Closing a Project	72
Editing HTML and Text Files	73
Additional Features of Witango Projects	74
Working With Project Dependencies	74
Working With Application Files	75
Working With Presentation Pages	77
Working With Project Data Sources	78
Working With Project Objects	79
Working With Project FTP Sites	79
Application-Specific Witango (AST) Signatures for Projects	79
Project Root Properties	81
4 Using Data Sources	83
About Data Sources	84
The Data Sources Workspace	85
Using Primary Key Columns	87
Data Source Operations	88
Creating a Data Source	88
Modifying a Data Source	98
Deleting a Data Source	99
Reloading a Data Source	99
Handling Unknown Data Sources	100
Assigning Data Sources to Actions	101
Setting Up Deployment Data Sources	103
Setting Deployment Data Source Properties	103
Meta Tags and Deployment Data Sources	105
Setting Data Sources for Actions	106
Using the Data Source Selection Dialog Box	106
Using the Action Properties Dialog Box	107
Disabling the Use of Meta Tags in Data Sources	109
Working With Data Source Properties	110

Data Source Properties	110
Table Properties	112
Column Properties	112
Connecting to Data Sources	113
Connecting to Large Data Sources	113
Editing and Executing Files on Different Computers	114
5 Using Snippets	117
About Snippets	118
The Snippets Workspace	118
Working With Snippets	120
Inserting Snippets	120
Creating and Editing Snippets	121
Managing Snippets and Snippets Folders	124
Copying, Moving, and Deleting Snippets	126
Column Snippets	128
6 Setting Preferences	129
Using the Preferences Dialog Box	130
Selecting Options	131
General	131
Text	133
Objects	137
Compile	138
Witango Building Blocks 141	
7 Working with Meta Tags	143
About Meta Tags	144
Where You Can Use Meta Tags	145
Combining Meta Tags	146
Quoting Attribute Values	147
Inserting Meta Tags	148
8 Working With Variables	155
Assigning Variables With the Assign Action	156
Editing Variable Assignments	156
Shortcuts to Configuration Variable Assignments: Snippets	160
Witango Builders 163	

9 Building Actions Using Witango Builders	165
Adding a Builder to an Application File	166
Page Format Table Settings	167
Building the Actions	169
10 Configuring the Search Builder	171
About the Search Builder	172
What Users See in Their Web Browser	172
How You Create These Web Pages	174
Main Steps to Use the Search Builder	176
Setting Search Options	178
Search Columns List	178
Column Options	179
Fixed Value	184
Summary: Setting Column Options	185
Formatting the Search Form	186
Customizing Your Search Form and Response Messages	187
Header, Footer, and No Results HTML	187
Changing Button Titles	188
Setting Record List Options	189
Display Columns	189
Order By	190
Column Options	190
Maximum Matches	193
Formatting the Record List Web Page	195
Customizing Your Record List Web Page	196
Header and Footer HTML	196
Setting Record Detail Options	197
Display Columns	197
Column Options	198
Record Maintenance Options	200
Formatting the Record Detail Web Page	201
Customizing Your Record Detail Web Page and Response Messages	202
Header, Footer, Update Response, and Delete Response HTML	202
Button Titles	203
Simplified Steps to Use the Search Builder	204
Defining Joins	205
Actions Built by the Search Builder	206
HTML Snippets	208

I I Configuring the New Record Builder 209

About the New Record Builder	210
Main Steps to Use the New Record Builder	211
Setting New Record Options	212
Summary: Setting Column Options	216
Formatting the New Record Entry Form	217
Customizing Your Form and Response Messages	218
Header, Footer, and New Record Response HTML	218
Changing Button Titles	219
Actions Built by the New Record Builder	220
HTML Snippets	222

Witango Actions 223

I2 Using Actions 225

About Actions	226
Working With Actions	229
Adding an Action	229
Naming an Action	230
Deleting an Action	231
Editing an Action	232
Moving an Action	232
Copying an Action	232
Context-Sensitive Action Menu	234
Action Properties	234
Assigning Attributes to Actions	236
Results HTML	237
No Results HTML	239
Error HTML	240
Push	241
Debug File	241
Adding HTML (Results Action)	242
Presentation Action	243
Uses of the Presentation Action	243
How the Presentation Action Works	243
Setting Up a Presentation Action	244

I3 Grouping Actions 245

About Grouped Actions	246
Working With Action Groups	247
Adding an Action Group	247

Adding an Action to a Group	247
Removing an Action From a Group	247
Ungrouping Actions	248
Deleting an Action Group	248
Effects of Editing an Action Group	248
Branching to an Action Group	248
Executing Grouped Actions	249
I4 Using Basic Database Actions	251
Searching a Database	252
Setting Up a Search Action	252
Executing a Search Action	264
Adding Records to a Database	266
Setting Up an Insert Action	266
Executing an Insert Action	267
Modifying a Database Record	268
Setting Up an Update Action	268
Executing an Update Action	269
Removing a Database Record	270
Setting Up a Delete Action	270
Executing a Delete Action	271
Adding Custom Columns to Database Actions	272
I5 Using Control Actions	273
Jumping to a Designated Action (Branch Action)	274
Branch Action Destination Rules	274
Executing a Branch Action	276
Branch and Return	276
Setting Up a Branch Action	277
Branch Action Destination Navigation	278
Deciding Course of Actions (Conditional Actions)	280
Example: Sports Fan Web Site	280
General Forms of Conditional Actions	280
Nested Conditional Actions	282
Performing Operations on Conditional Actions	282
Setting Up Conditional Actions	283
Repeating a Set of Actions (Loop Actions)	288
Example: Music Store	288
General	
Forms of	
Loop Actions	288

Nested Loop Actions	290
Setting Up Loop Actions	290
Executing Loop Actions	292
Performing Operations on Loop Actions	293
Exiting a Loop (Break Action)	294
Ending File Processing (Return Action)	295
16 Extending Witango Functionality	297
Executing JavaScript	298
Setting Up a Script Action	298
Executing a Script Action	299
Using an External Action	301
Setting Up an External Action	301
Configuring a DLL Call	301
Using a Command Line	302
Configuring a Java Action	304
Assigning Attributes	305
Deleting Parameters	306
Executing an External Action	306
Disabling JavaScript, Java and External Actions	308
17 Sending Electronic Mail From Witango	309
Setting Up a Mail Action	310
General Tab	310
Options Tab	312
Attachments Tab	313
Disabling Mail	315
18 Reading, Writing, and Deleting Files	317
Setting Up a File Action	318
Setting Up Read Options	319
Setting Up Write Options	319
Setting Up Delete Options	321
Handling File Security	323
19 Using Advanced Database Actions	325
Using Database Transactions	326
Setting Up a Transaction Action	326
Executing a Transaction Action	329
Using SQL Directly	330

Setting Up a Direct DBMS Action	330
The Direct DBMS Action Editing Window	331
Executing a Direct DBMS Action	334
Joining Database Tables	336
Working With Joins	337
Creating a Join in a Search Action	338
Inserting a Join	340
Editing a Join	340
Deleting a Join	340
Creating a Join in the Search Builder	341

Witango and Objects 343

20 Understanding Objects in Witango 345

What are Objects?	346
Objects as Black Boxes	346
Object Interface: Methods	347
Method Elements: Parameters	348
Class, Object, and Object Instance	348
Creating Object Instances	349
Using Available Object Instances	350
Calling Methods	350
Example 1: Investment Scenarios	351
Example 2: More Investment Scenarios	352
Benefits of Using Objects in Witango	354
When to Use Objects	354
Object Types Supported in Witango	355
Object Type Independence	355
COM Objects	355
JavaBeans	356
Witango Class Files	356
General Requirements	357
Understanding Data Types	358
Setting up Security for Executing Objects	359

21 Using Objects 361

Preparing to Use Objects in Witango	362
Planning to Use an Object	362
Installing an Object	362
Overview of Using Objects in Witango	364
Adding an Object to the Objects Workspace	365
COM Objects in the Objects Workspace	365

JavaBeans in the Objects Workspace	367
Witango Class Files in the Objects Workspace	369
Removing an Object From the Objects Workspace	371
Viewing Object Information in the Objects Workspace	372
Attributes Folder	373
Object Properties	374
Caching and Refreshing of Object Information	375
Adding a Create Object Instance Action	377
Shortcut to Adding a Create Object Instance Action	378
Completing the Create Object Instance Action	379
Object Name	380
Object Instance Variable	380
Instance	380
Expiry URL	382
Adding a Call Method Action	383
Shortcut to Adding a Call Method Action	384
Completing the Call Method Action	385
Object/Method Name	386
Object Instance Variable	386
Result Variable	386
Parameter List	387
Using the Objects Loop Action	391
Example of Using an Objects Loop	391
Using an Objects Loop	392
22 Witango Class Files	395
What are Witango Class Files?	396
Benefits of Using Witango Class Files	397
When to Develop and Use Witango Class Files	398
Using Witango Class Files	399
Developing Witango Class Files	400
Method List Pane	401
Method Editing Pane	402
Instance Variables List Pane	404
Method Definition Window	405
Creating a Witango Class File	408
Editing a Witango Class File	409
Adding a New Method	410
Renaming a Method	410
Deleting a Method	411

Copying a Method	411
Modifying a Method	412
Setting Return Values and Parameters	412
Method Properties	414
Debugging Methods	415
Setting Search Paths for Witango Class Files	416
Witango Studio	416
Witango Server	416

Witango Compiler 419

23 Compiling Witango Application Files 421

The Compilation Process	422
.....	422
Syntax Checking	423
Creating a Syntax Check Report	423
Filtering the Syntax Check Report	426
Understanding a Syntax Check Report	426
Correcting Issues located in a Syntax Check Report	428
Rechecking the Syntax	429
Compiling you Witango Application	431
Executing a compile for J2EE	431

Glossary of Terms 437

24 Index 453

Introduction

An overview of this User Guide

The *User's Guide* introduces you to Witango and tells you how to perform the tasks necessary to create your applications. It is your main source of information on how to use the Witango Studio. Topics covered include Witango basics, including using application files, data sources, snippets, and action builders. This User Guide should be used in conjunction with the Witango Programmers Guide which provides all detail relating to the Witango Meta Tags, Configuration Variables, Custom Meta Tags, DOMs Error Codes and Expression Operators .

What is Witango

Witango is a powerful yet easy-to-use tool for creating dynamic, intelligent Web sites that integrate with popular database systems. With Witango's actions and builders, you build solutions by using Witango's intuitive point-and-click, drag-and-drop interface. You can create simple applications in minutes—without ever writing any code. You can customize your application files by adding your own HTML, database queries, and control flow, and by accessing external programs. You can send data to and retrieve data from external objects.

Witango Components

Witango consists of two main programs: Witango Studio and Witango Application Server, hereafter known simply as Witango Server.

- Witango Studio is the development environment, featuring a complete graphical user interface in which to develop and compile Witango application files (or simply, application files).
- Witango Server is an application server that executes Witango application files created with Witango Studio. It works in conjunction with an HTTP (Web) server to return HTML to a Web browser.

For definitions of terms used throughout the document, see Appendix A.

SECTION I

Using Witango Studio

How to Use Witango Studio

This section of the *Witango User's Guide* gives details on the basics of Witango Studio, including the Witango Studio environment, working with Witango application files, Witango projects (including source control and deploying and downloading projects), working with data sources, snippets, and setting Witango Studio preferences.

This section contains chapters on the following topics:

- Chapter 2, *Witango Studio Basics* on page 5
- Chapter 3, *Using Projects and Source Control* on page 65
- Chapter 4, *Using Data Sources* on page 83
- Chapter 5, *Using Snippets* on page 117
- Chapter 6, *Setting Preferences* on page 129.

Chapter 2, the applicable parts of Chapter 3, and Chapters 4–5 in this section are strongly recommended for new users of Witango.

Witango Studio Basics

Introducing the basics of the Witango Studio Interface and Witango Application Files

This chapter helps you orient yourself to the Witango Studio interface and some of the common operations available to you, looks at how Witango actions work and describes Witango application file operations.

The topics covered in this chapter include:

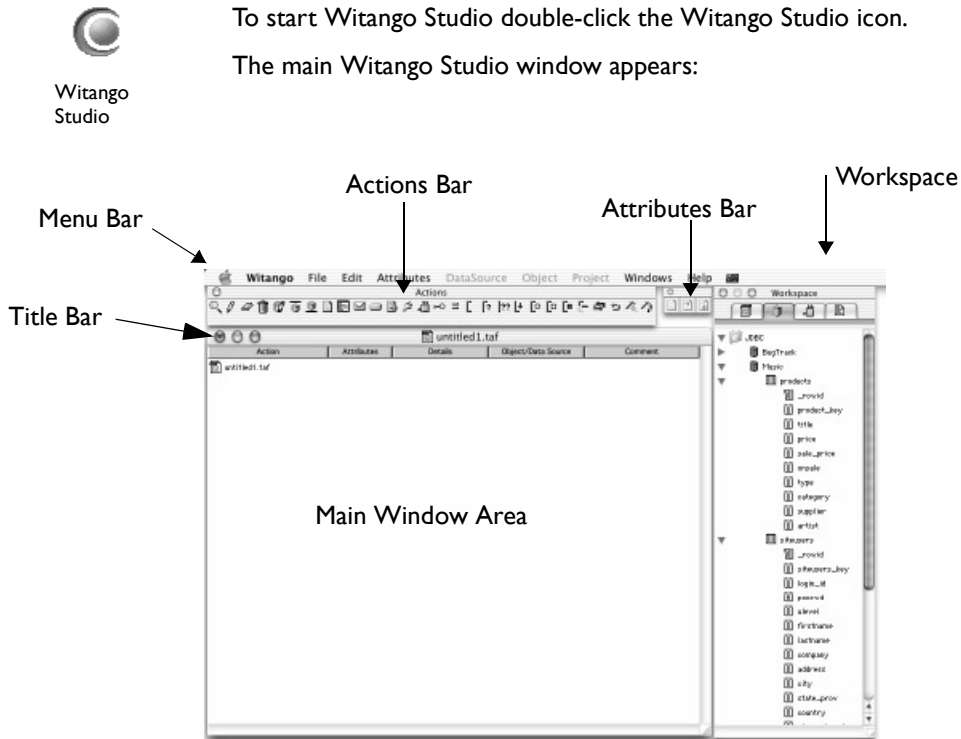
- Witango Studio interface components
 - overview of the Witango Studio interface
 - using contextual menus
 - using HTML editing windows
 - using Word Wrap
 - working with multi-column lists in action editing windows
 - using the SQL Query window
 - finding and replacing text or regular expressions
 - keyboard shortcuts.
- Working with actions:
 - the Actions palette
 - working with actions
 - assigning attributes to actions
 - the Results action
 - the Presentation action.
- Using Witango application files
 - XML file format details
 - the Witango application file window
 - creating and saving Witango application files
 - debugging Witango application files
 - executing Witango application files

Witango Studio Interface Components

When you double-click the Witango Studio icon, the main Witango interface components appear.

To start Witango Studio double-click the Witango Studio icon.

The main Witango Studio window appears:



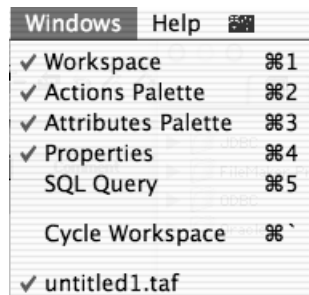
- 1 The **title bar** displays the name of the current (front most) Witango application file or the SQL Query window.
- 2 The **menu bar** contains pull-down menus for Witango Studio commands. Click a menu title to open it, then click a command to select it. Commands appearing in gray are disabled and do not apply to the operation you are trying to perform.
- 3 Click icons on the **Actions bar** and drag them into an open Witango application file to add them to the file.
- 4 Click icons on the **Attributes bar** to assign attributes to selected actions.
- 5 The **Workspace** includes tabs for Data Sources, Objects, Snippets, and Projects, if any exist. You switch among the four sections of the

Workspace by clicking the corresponding tab. The four sections are called Data Sources Workspace, Object Workspace, Snippets Workspace, and Project Workspace, respectively.

- 6 The **Main Window Area** displays one or more Witango application file windows, action editing windows, attribute editing windows, or the SQL Query window.

Viewing Interface Components

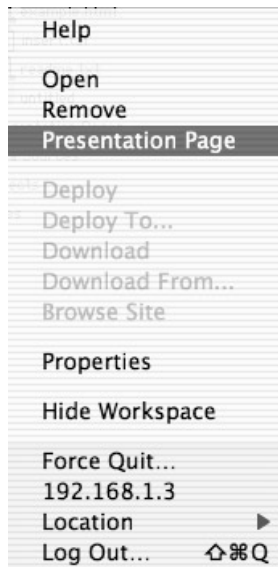
You can choose to show or hide the Workspace window and any of the palettes by enabling the component's name from the Windows menu. A check mark beside the name indicates the component is visible in the interface. Uncheck the name to hide the component.



Contextual Menus

In many Witango Studio interface components, you can position the cursor on a particular area and `control+click` the mouse to display a contextual *menu* of commands. The commands that appear relate to the

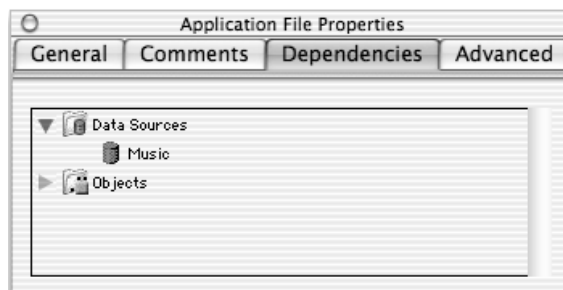
item you click. Grayed-out commands are not applicable to the current item.



Properties Window

The Properties window allows you to view information about and add comments to a selected item. Selectable Witango items include data sources (including tables and columns), application files, and actions. In general, the Properties window changes to show the properties of the currently selected item.

The following is an example of an Application File Properties window for a taf file:



To open any Properties window

- I Select the item you want to view information about.

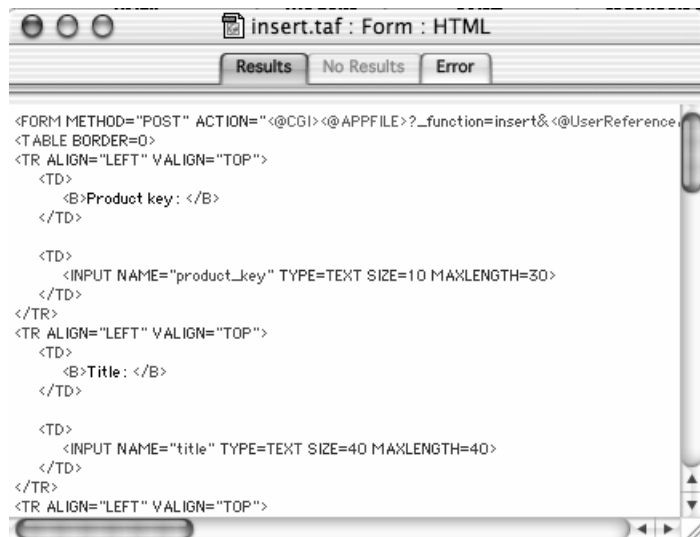
2 Do one of the following:

- From the Windows menu, choose **Properties**.
- `control+click` the item, and choose **Properties** from the contextual menu that appears.

The Properties window can be left open. Clicking an item with properties updates the window to show information about that item.

HTML Editing Window

Most actions in an application file can have HTML associated with them. Whenever you select an action and an attribute option, the HTML editing window appears. This example shows the HTML editing window for the **Results HTML**:



The title of the window follows the form:

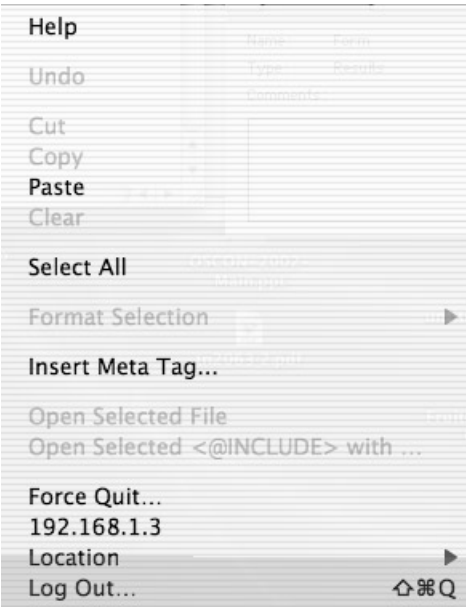
```
<Document> : <Action> : <HTML>
```

Witango Studio supports the standard editing commands. The **Edit** menu displays the following commands:

Edit	Attributes	DataSource
Undo		⌘Z
Redo		⌘Y
Cut		⌘X
Copy		⌘C
Paste		⌘V
Insert		⌘I
Clear		
Select All		⌘A
Find & Replace...		⌘F
Find Again		⌘G
Replace & Find Again...		
Rename Action Item		
Group		
Ungroup		
New Method		
Insert Meta Tag...		⌘M
Insert Custom Column...		
Insert Criteria Separator		
New Snippet...		
New Snippet Folder...		
Word Wrap		
Define Sites...		

Contextual Menu

You can also `control+click` in the HTML editing window to display a similar menu at the cursor position in the window.



The following table lists the commands in the contextual menu:

Command	Function
Help	Accesses HTML help files.
Undo	Undoes the last change made to the text.
Cut	Removes the selected text from the window and places it on the clipboard.
Copy	Copies the selected text to the clipboard.
Paste	Pastes text on the clipboard at the cursor position.
Clear	Clears the selected text.
Select All	Selects all text within the HTML editing window.
Insert Meta Tag	Displays the Insert Meta Tag dialog box. For more information, see “Inserting Meta Tags” on page 148.

For more information, see `<@INCLUDE>` in HTML Editing Windows on page 16.

Command	Function
Open Selected <code><@INCLUDE></code>	Opens the file which the selected <code><@INCLUDE></code> tag points to. The file is opened with either Witango Studio or the application used to create the file, as specified in the Preferences dialog box.
Open Selected <code><@INCLUDE></code> with...	Opens the file which the selected <code><@INCLUDE></code> tag points to. The file is opened with the application specified in the External Editor field in the Preferences dialog box.

Closing the window automatically saves any changes you make. To cancel any changes, you can choose **Undo** from the **Edit** menu or the contextual menu, or close the file without saving it.

Syntax Coloring

To make editing of your files easier and clearer, many of the HTML and text components that appear are color-coded—HTML, Witango meta tags, attributes, default text, and comments. You can change the specified colors.

You can enter any amount of text in an HTML editing window. You can also drag and drop text from elsewhere, for example, from other editing windows.

Word Wrap

Word Wrap

Word wrap is also available in the HTML editing window. From the **Windows** menu, choose **Word Wrap** to enable and disable it. A checkmark indicates that word wrap is enabled as well as many other windows. See Word Wrap page 12 for further details.

Indenting and Selecting Text

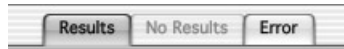
You can position text using tab characters. Tabs are stored as tab characters and are not converted to spaces. Tabs have no effect on the display of HTML in the Web browser; they are used to make the HTML you enter more readable.

For more information on the Preferences dialog box, See “Setting Preferences” on page 129.

You specify the number of space characters that equal one tab character in the **Tab size** field of the Preferences dialog box. You can also specify whether you want Witango to insert tab characters to start a new line at the same indent level as the previous line by enabling or disabling **Auto-indent** in the Preferences dialog box.

HTML Editing Window and Attributes

The HTML editing window allows you to easily access any of the HTML attributes that may be assigned to an action: Results HTML, No Results HTML, and Error HTML. Not all attributes may be assigned to all actions: the applicable attribute tabs for the selected action appear enabled at the top of the window; unavailable tabs are grayed out.



Within the HTML editing window, you can switch between the different HTML attributes by clicking the appropriate tab.

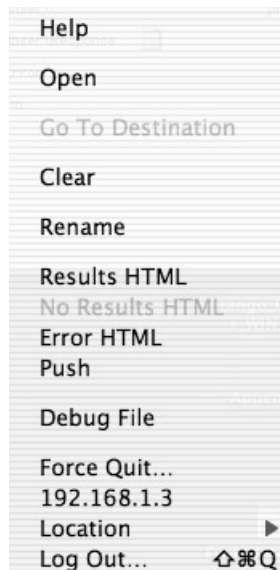
You can also open the attribute HTML associated with an action by doing one of the following:

- Select an action icon/name, and choose the attribute type from the **Attributes** menu.

For information on new keyboard shortcuts, see Keyboard Shortcuts on page 34.



- **control+click** the action icon/name, and choose the attribute type from the contextual menu.



- Double-click the attribute icon beside the action in the application file window.

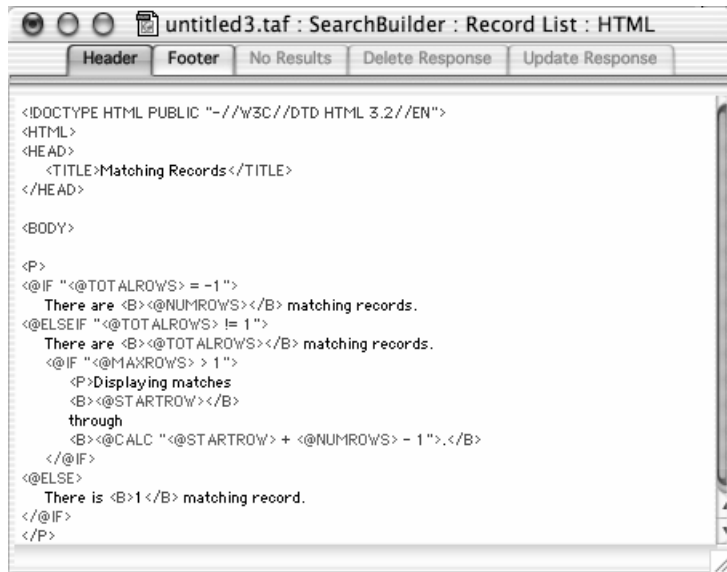


The corresponding HTML editing window opens.

For more information on customizing page and response HTML for the Search Builder, see [Configuring the Search Builder](#) on page 171.

The Search Builder also uses HTML editing windows so you can customize page HTML (Header and Footer) and response HTML (Update Response, Delete Response, and No Results) for the Search, Record List, and Record Detail pages.

For example, the following HTML editing window shows the Header HTML for the Search Builder's Record List page in the `Untitled3.taf` application file.

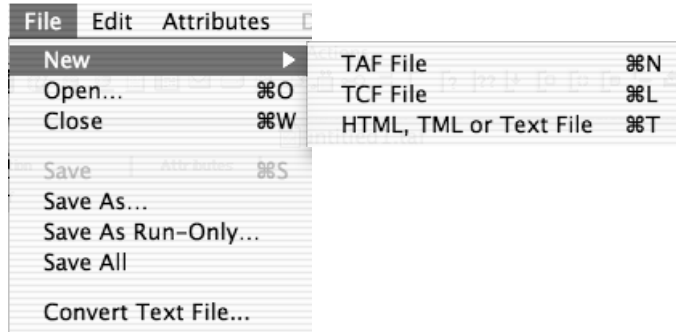


Creating a New HTML or Text File

In addition to editing the HTML associated with an action or a builder, you can use Witango's editing capabilities to create and edit HTML and text files. The editing capabilities and window settings described for HTML action attributes also apply to HTML and text files opened for editing with Witango Studio.

To create a new HTML or text file

- From the **File** menu, choose **New**, then **HTML or Text File**.



A blank editing window opens:



The default window name is “untitled”, until you save it under another name. Subsequent new windows are named “untitled n ”, where n is the next number in the series; that is, the second window opened is “untitled2”, and so on.

To save a new HTML or text file

- From the **File** menu, choose **Save** or **Save As**.

The Save As dialog box appears.

- In the **File name** field, type the name of your file.

When you save a new text file and a project is open, Witango asks if you want to add the saved file to the open project.

To open an HTML or text file

- 1 From the **File** menu, choose **Open**.

The Open dialog box appears.

- 2 Select the file to open.
- 3 Click **Open**.



Tip You can also open a file of a supported type simply by dragging it from the Finder onto the Witango Studio icon, whether or not Witango Studio is already open.

<@INCLUDE> in HTML Editing Windows

HTML editing windows in Witango can contain `<@INCLUDE>` meta tags, which, when executed by Witango Server, return the contents of a specified file. When you edit text and HTML in the HTML editing window, you can open files referenced with `<@INCLUDE>` by selecting one of the commands **Open Selected <@INCLUDE>** or **Open Selected <@INCLUDE> with**.



Note In order for the **Open Selected <@INCLUDE>** and **Open Selected <@INCLUDE> with** commands to function correctly, the application file and the included file you want to access must be in the current project, or they must both be located in the same folder. Otherwise, Witango Studio will be unable to find the file.

Open Selected <@INCLUDE>

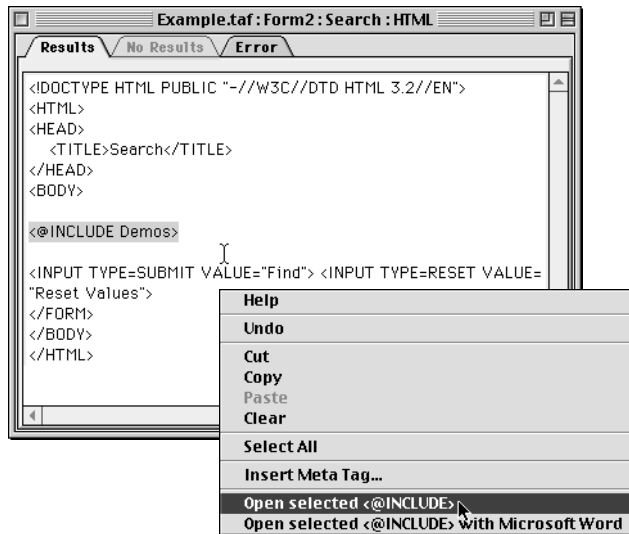
Open Selected <@INCLUDE> causes Witango to open the file referenced by `<@INCLUDE>` according to the preferences you have set in the Preferences dialog box. For example, if you have set the **Open text files using** option to **Witango Studio**, the included file opens in Witango Studio. If you have set this option to **Application used to create the file**, the included file opens in the application with which it is associated. If you have set this option to **Selected external editor**, the included file opens in the application you have specified in the **External editor** field in the Preferences dialog box.

To open an included file using Open Selected <@INCLUDE>

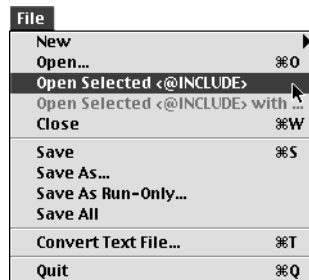
Select the `<@INCLUDE>` tag that you want to open, and do one of the following:

For more information, see
Setting Preferences on
page 129.

- CONTROL+click in the HTML editing window and, from the contextual menu that appears, choose **Open Selected <@INCLUDE>**.



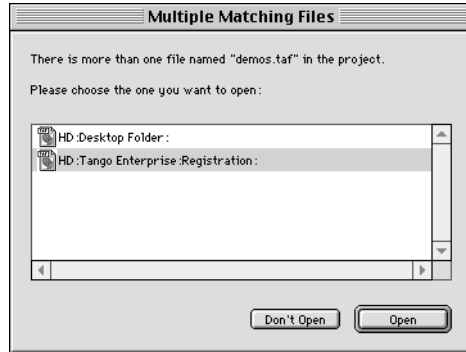
- From the **File** menu, choose **Open Selected <@INCLUDE>**.



For more information, see
Setting Preferences on
page 129.

The included file opens in either Witango Studio or an external editor,
depending on your preference settings.

If there is more than one included file with the same name in the project, the **Multiple Matching Files** dialog box appears:



Select the included file you want to open and click **OK**.



Tip You can open multiple included files simultaneously by selecting all the tags and choosing **Open Selected <@INCLUDE>**.

Open Selected <@INCLUDE> with

Open Selected <@INCLUDE> with causes Witango to automatically open the selected included file in the application you have specified in the **External editor** field of the Preferences dialog box. Even if you have your preferences set to open the file automatically in Witango Studio or the application used to create the file (that is, **Witango Studio** or **Application used to create the file** is selected in the **Open text files using** section of the Preferences dialog box), Witango still uses the application which appears in the **External editor** field to open the included file.

You can open a selected <@INCLUDE> tag with this command in the same way as **Open Selected <@INCLUDE>**; see the previous section.

Working With Multi-column Lists

Many Witango actions include multi-column lists for entering parameters—the criteria list in the Search action, for example. This section describes basic techniques for working with these lists.



To select an entire row

Click the row's Column cell.

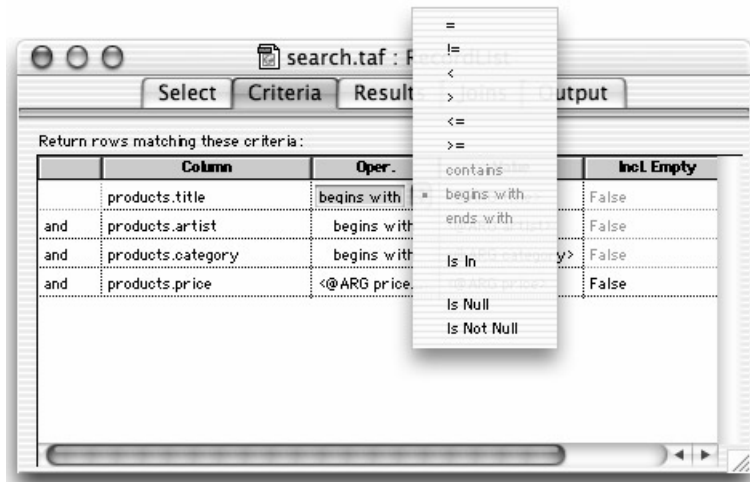


To move a row

Select the row and drag it to the desired location.

**Pop-up Menus**

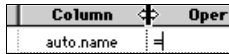
Various columns have pop-up menus in each cell. Place the cursor in the cell and click the mouse. A downward-directional arrow appears. Click the arrow and the pop-up menu appears.



From a cell's pop-up menu, you can select from preset values.

To resize a column

Click the edge of the column in the list's header, and drag.



A double-headed arrow appears when you move your cursor between column. Drag to resize the column.

To resize a column to fit the data in it, double-click its right edge in the header.

To delete a row

- 1 Select the row to delete.
- 2 Do one of the following:
 - From the **Edit** menu, choose Clear.
 - Press `Delete`.

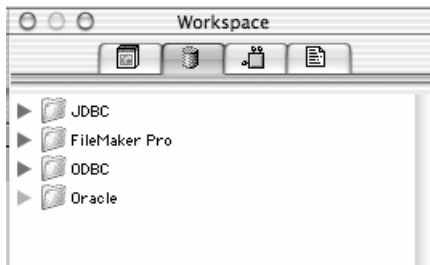
Dragging Columns

When creating or modifying a Witango application file and actions, you must specify which database columns to use in various places. To do this, you drag the columns from the Data Sources Workspace to the appropriate place in the file.

To ...	Do This ...
Select contiguous columns	Click the first column you want to select and <code>Shift</code> +click the last one.
Select discontinuous columns	Click the first column you want to select and <code>⌘</code> +click the additional columns.
Select all columns in a table	Drag the table name into the file.

To see the Data Sources Workspace, click the Data Sources tab. A workspace appears, containing information about data sources, such as

the currently defined data sources and all tables and columns. If no data sources are set up yet, only the data source types appear.

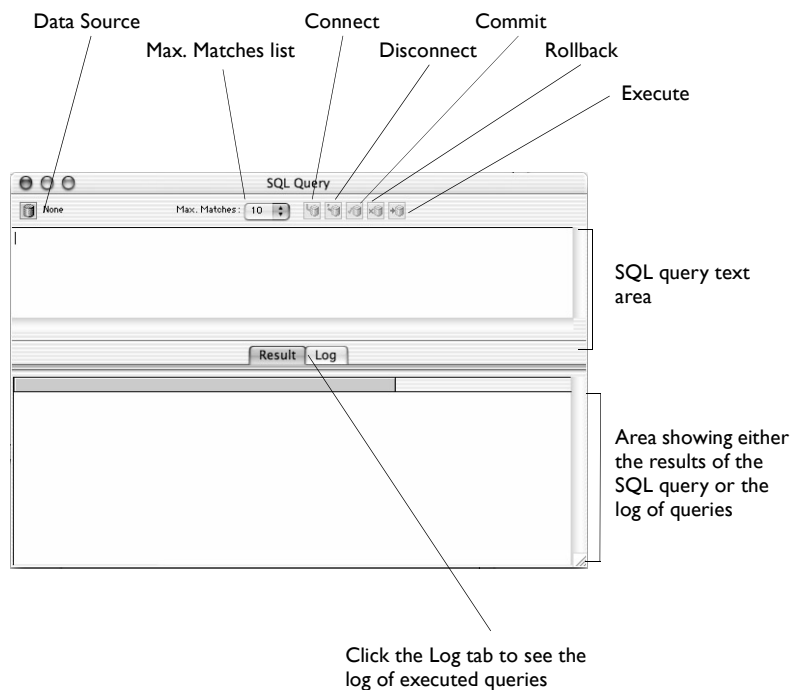


If word wrap is disabled, a horizontal scroll bar is available to view text outside the boundaries of the window.

The SQL Query Window

The SQL Query window gives you a convenient way of performing simple SQL queries within Witango Studio, for example, to test your Direct DBMS actions or to check database values.

The SQL Query window displays the following components:



If you want to resize the query and results areas, place the cursor over the area separator to display the resizing icon. Then click and drag it up or down to change the sizes of each area.

Setting Up a SQL Query

The components and functions of the SQL Query window are as follows:

- **Data Source** button allows you to specify the data source you want to perform query operations against. When you first open the SQL Query window, the data source is set to **None**.

If you change the data source assigned to the window, any existing connection closes.

You cannot perform SQL queries on FileMaker data sources.

- **Max. Matches** displays the maximum number of records you want the SQL query to return. You can select from **1**, **10**, **25**, **50**, or **100**. The default is **10**.

- **Commit** and **Rollback** buttons allow you to perform a SQL COMMIT or ROLLBACK operation on the assigned data source. COMMIT causes any changes made to the data source by the query to be saved. ROLLBACK causes any changes made by the query to be discarded.

These buttons are disabled when you are not connected to a data source.

- **Connect** and **Disconnect** buttons allow you to connect to or disconnect from the current data source.

When you try to connect without first assigning a data source, the Data Source Selection dialog box appears; you must select a data source.

- **Execute** button allows you to execute the SQL query in the query text area. If you are not connected to the data source when **Execute** is selected, the connection is made automatically.

Any data returned by the SQL query appears in the **Results** area of the SQL Query window. If the **Results** area contains data and the current query returns no data, the **Results** area is cleared of any data.

After execution, the connection to the data source remains open.

To cancel an executed query, press `esc` or  `+`. (period). If results are being returned when a cancel request is made, the **Results** area shows all the data returned to that point.

- **Query Text Area** displays the SQL query text to be executed.

For more information on SQL COMMIT and ROLLBACK operations, consult your SQL documentation.

The query text area supports standard cut, copy, and paste operations, including drag and drop. You can drag and drop tables and columns into the SQL Query text area from the Data Sources Workspace.

You can also drag any database action (except Transaction) from an application file to the SQL Query window to see the SQL Witango generates for it.

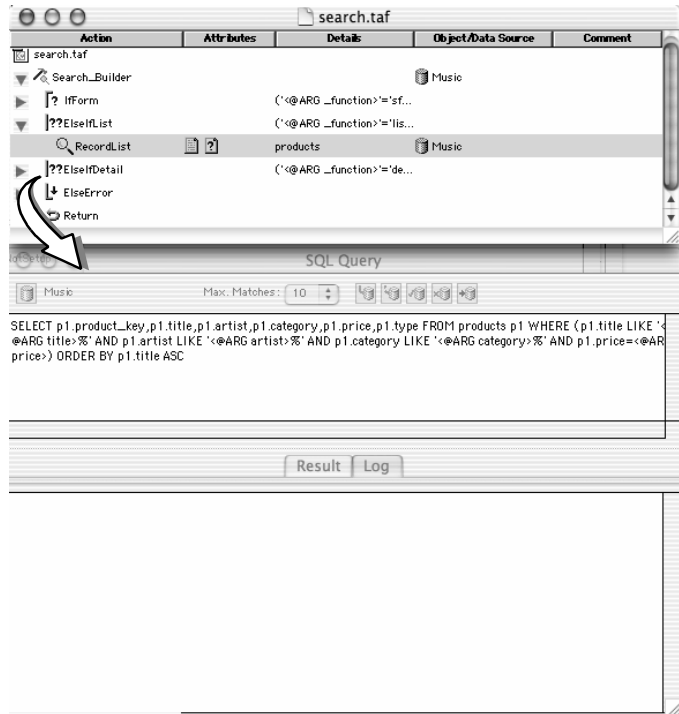
If you select only part of the SQL when executing the query, only that part is sent to query the database.

- **Results** tab displays in columns and rows the results of the SQL query.
- **Log** tab displays the log of executed queries.
- **Status Area** shows the current status of the SQL query. The status messages appear as follows:

Status	Description
Not connected	No connection is established.
Connecting...	Appears during connection to the data source.
Connected	Connection is established.
Executing...	Appears during execution of query.
Rolling back changes...	Appears during rollback operation.
Committing changes...	Appears during commit operation.

Dragging Actions into SQL Query Text

You can drag any database action, except a Transaction action, which does not generate SQL, from an application file into the SQL Query window.



When you do this, some SQL Query window attributes are set based on the contents of the action. The following attributes are automatically set:

- **Max. Matches** (for a search action) is set to the action's maximum matches value; otherwise, it is set to unlimited.
- The data source is set to the action's data source, and closes any existing database connection (if the data source is different from the current data source).
- The SQL text is the data source-specific SQL that Witango Server generates when the action is executed.



Note Any meta tags from the action are placed in the text as-is. The SQL text also does not include any text automatically added to the action's SQL by the server (such as a \$maxrows assignment with DAM data sources).

- The **Results** area is cleared of the currently displayed results.

Performing a SQL Query

To perform a SQL query

- 1 Choose the **SQL Query** command by doing one of the following:
 - From the **Windows** menu, choose **SQL Query**.
 - **control+click** the application file window or an open action window, and choose **SQL Query** from the contextual menu that appears.

An empty SQL Query window appears.

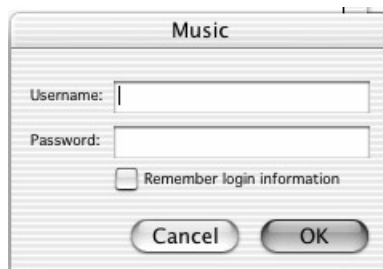
- 2 Click **Data Source**.

When you first open the SQL Query window, the data source is **None**.

The Data Source Selection dialog box appears:



- 3 Select the data source you want to perform SQL Query window operations against, and click **OK** to load the tables and columns of that database. A Log In dialog box appears, allowing you to type your user name and password:



1 Enter your user name and password into the respective fields, and click **OK**.

1 From the **Max. Matches** menu, select the maximum number of records to return from a SQL query: **1**, **10**, **25**, **50**, or **100**.



2 Click **Connect** to connect to the current data source.

3 In the SQL Query text area, enter the SQL query text to be executed.

4 Click the Execute icon.



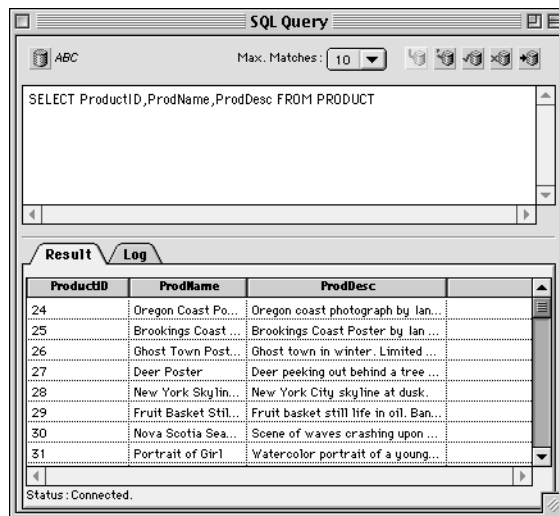
If you select part of the SQL in the SQL Query text area, only that part is executed when you click the button.

5 If you want to perform a **COMMIT** or **ROLLBACK** operation on the assigned data source, click the corresponding **Commit** or **Rollback** button.



The results of the SQL query, if any, appear in the **Results** area.

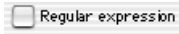
The following is an example of SQL query text and the returned results:



Finding and Replacing Text

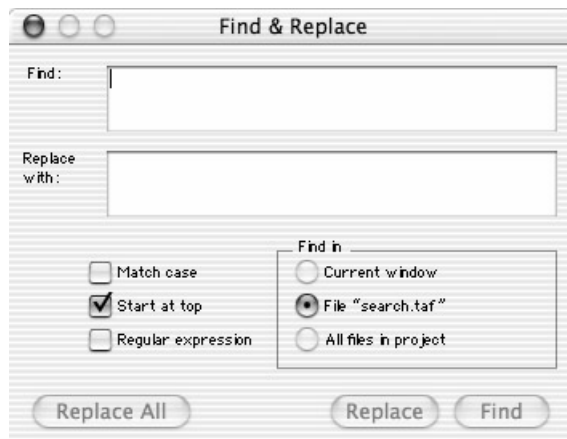
In Witango Studio, you can perform operations to find, or to find-and-replace text in application files. Witango Studio can perform both normal searches and searches using regular expressions.

Performing Find Operations



For the purpose of this discussion, the term *string* refers to both character strings (that is, text) and regular expressions. You specify that the search is to treat the string in the **Find** field as a regular expression by selecting the **Regular expression** option in the Find & Replace dialog box.

If you want to find or replace a certain string, you specify that string in the Find & Replace dialog box



You can find any string that can be entered in any non-modal Witango Studio window. This includes values in criteria lists, action parameters you have entered—such as for the **Limit to** field in a Search action's Results window, custom SQL, If action conditions, External action parameters, custom column definitions, and HTML. Witango Studio cannot find a string you did not explicitly enter, for example, data source names, user names or passwords entered by users, column names in Select lists, and join information.

You can perform find-and-replace operations in open application files, action editing windows, HTML editing windows, and projects. Unless specified otherwise, Witango Studio begins searching at the insertion point indicated by the cursor and continues to the end of the search range specified in the **Find In** section of the dialog box.

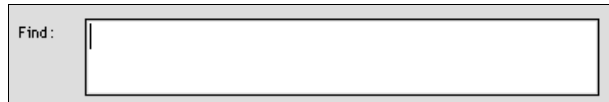
To find or find-and-replace a string

- I Do one of the following:
 - Choose **Find & Replace** from the **Edit** menu.
 - Press ⌘+F.

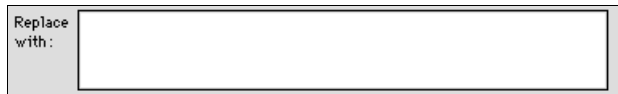
The Find & Replace dialog box appears.

2 Specify your find or replace options as follows:

- **Find.** Enter the string you want to find.

A rectangular input field with a light gray border. To its left, the text "Find:" is displayed in a small font.

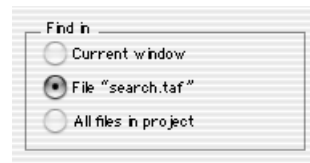
- **Replace with.** Enter the string that you want to replace the string in the **Find** field with.

A rectangular input field with a light gray border. To its left, the text "Replace with:" is displayed in a small font.

☐ Match case

☐ Regular expression

- **Match case.** If you want to perform a case-sensitive search, select the **Match case** option; otherwise, Witango Studio searches for a match irrespective of letter case. For example, a search for “customer” would find all instances of “customer”, “Customer”, and “CUSTOMER”.
- **Regular expression.** If you want to search for the string as a regular expression, you must select the **Regular expression** option. Otherwise, a normal search is performed.
- **Find in.** You specify the search range in this area of the dialog box.

A section titled "Find in" containing three radio button options: "Current window", "File 'search.tif'", and "All files in project". The "File 'search.tif'" option is selected, indicated by a filled circle.

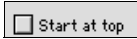
- **Current window.** Select this option to perform the find or replace operation in the window active at the time you choose the **Find** or **Replace** command. If you have a string selected in the active window, it automatically appears in the **Find** field.
- **File filename.** Select this option to perform the find or replace operation in the file specified by *filename*. The name of the currently active file automatically appears as *filename*.
- **All files in project.** If you have a project open, this option is checked. Select this option to perform the find or replace operation in all the files of the active project. If you have another application file open at the same time, which is not

part of the project, Witango Studio excludes it from the find or replace operation.

- **Start at top.** Select this option to start the find or find-and-replace operation at the top of the search range specified in the **Find in** section.



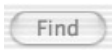
Tip To start your search at the top of your project, check **All files in project** and **Start at top** in the Replace dialog box.



If this option is not selected, Witango Studio performs the search starting from the current cursor position.

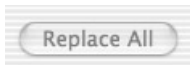


Note If the current cursor position is not within that range, the current cursor position is ignored and the search starts at the top of the specified range.

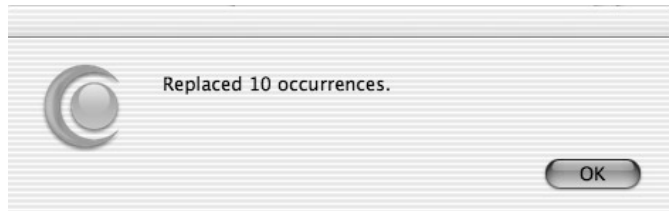


- **Find.** Click to start the search for the string specified in the **Find** field from the specified starting position.
- **Replace.** Click to replace the string specified in the **Find** field with the string specified in the **Replace with** field. Following the replace operation, Witango Studio automatically searches for the next instance of the find string.
- You can undo the last replace performed by choosing **Undo** from the **Edit** menu.
- **Replace All.** Click to replace automatically *all* instances of the string specified in the **Find** field with the string specified in the **Replace with** field.

The following dialog box appears, indicating the number of

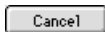


replacements made:



Note You cannot undo the Replace All operation. You can, however, choose to close a file without saving the changes to return it to its former state.

If the search range involves several items, those items in which replacements are made are opened so you can save or discard the changes.



- **Cancel.** Click to end the find or find-and-replace operation and to close the dialog box.

You can also choose **Find Again** from the **Edit** menu or press  +G, which searches again for the string or regular expression previously entered in the Find and Replace dialog box.

Using Regular Expressions

A *regular expression* is formed by one or more special characters that represent a string of text.



Note To find a special character, precede it with a backslash, for example, \ * finds the asterisk (*) character.

To find any single character

A period (.) finds any character except a newline character.

Expression ...	Finds ...
.use	fuse but not house

To repeat expressions

Repeat expressions with an asterisk (*) or a plus sign (+).

A regular expression followed by an asterisk finds zero or more occurrences of the regular expression. If there is any choice, Witango Studio chooses the longest, left-most matching string in a line.

A regular expression followed by a plus sign finds one or more occurrences of the one-character regular expression. If there is any choice, Witango Studio chooses the longest left-most matching string in a line.

Expression ...	Finds ...
a+b	ab and aab but not a or b
a*b	b , ab , and aab but not baa
.*use	use , mouse , and paint the house , but not chair

To group expressions

If an expression is enclosed in parentheses, (), Witango Studio treats it as one expression and applies an asterisk or plus sign to the whole expression.

Expression ...	Finds ...
(ab)*c	abc , ababc , and c , but not aabbcc
(.a)+b	xab , xaxab , but not b

To choose any character from many

A string of characters enclosed in square brackets, [], finds any one character in that string. If the first character in the brackets is a caret (^), it finds any character except those in the string.

Expression ...	Finds ...
[abc]	a , b , or c , but not x , y , or z
[^abc]	x , y , or z , but not a , b , or c

A minus sign (-) within square brackets indicates a range of consecutive ASCII characters. For example, [0-9] is the same as [0123456789]. The minus sign loses its special meaning if it is the first character (after an initial caret, if any) or last character in the string.

If a right square bracket is immediately after a left square bracket, it does not terminate the string; however, it is considered to be one of the

characters to match. If any special character—such as the backslash (\), asterisk (*), or plus sign (+)—is immediately after the left square bracket, it does not have its special meaning and is considered to be one of the characters to match.

Expression ...	Finds ...
[aeiou][0-9]	a9 but not ae
[^bm]ate	date but not bate or mate
END[.]	END. but not END;

To find the beginning or end of a line

- You can specify that a regular expression finds only the beginning or end of the line.
- If a caret (^) is at the beginning of the entire regular expression, it finds the beginning of the line.
- If a dollar sign (\$) is at the end of the entire expression, it finds the end of the line.
- If an entire expression is enclosed by a caret and dollar sign (for example, ^the end\$), it finds an entire line.

Expression...	Finds...
^(the house).+	the house guest but not paint the house
.+(the house)\$	paint the house but not the house guest

To re-use a regular expression in the Replace field

Witango extends the regular expression functionality and allows you to remember and recall a part of a regular expression. Enclose the part to remember with parentheses. To recall it, use \n, where *n* is a digit that specifies which expression in parentheses to recall. Determine *n* by counting occurrences of “(” from the left. You can only use this feature in the **Replace** field of the dialog box.



Tip For more information on constructing POSIX regular expressions, ask your local UNIX guru, consult the FreeBSD regex man page, or try doing an Internet search for the term “POSIX 1003.2”.

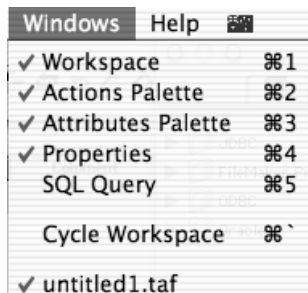
Keyboard Shortcuts

The keyboard shortcuts, as they appear in Witango Editor menus, are as follows:

Menu	Command	Shortcut
File	New Witango Application File	⌘+N
	New Witango Class File	⌘+L
	New HTML or Text File	⌘+T
	Open	⌘+O
	Close	⌘+W
	Save	⌘+S
	Quit	⌘+Q
Edit	Undo	⌘+Z
	Cut	⌘+X
	Copy	⌘+C
	Paste	⌘+V
	Insert	⌘+I
	Select All	⌘+A
	Find & Replace	⌘+F
	Find Again	⌘+G
	Replace & Find Again	⌘+H
	Insert Meta Tag	⌘+M
	Preferences	⌘+;
Attributes	Results HTML	⌘+R
	No Results HTML	⌘+U
	Error HTML	⌘+E
	Debug File	⌘+D
Windows	Workspace	⌘+I
	Actions Palette	⌘+2
	Attributes Palette	⌘+3
	Properties	⌘+\
	Cycle Workspace	⌘+`(single back quote)

Windows Menu Shortcuts

To view the Workspace, Actions palette, or Attributes palette, use the **Windows** menu commands.



For example, to view the Actions palette, either choose **Actions Palette** from the **Windows** menu, or press ⌘+2. If the palette is already visible, choosing the command or pressing the shortcut hides it.

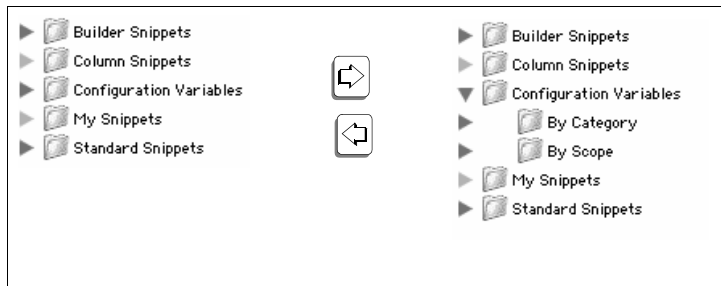
The **Cycle Workspace** command enables you to move consecutively from one Workspace window to the next. For example, if you are currently viewing the Project Workspace, pressing ⌘+` (the single back quote character located to the left of the “1” key on most keyboards), or choosing **Cycle Workspace** from the **Windows** menu shows you the Data Sources Workspace. If you are currently viewing the Snippets Workspace, pressing ⌘+` or choosing **Cycle Workspace** shows you the Project Workspace.

Expanding and Collapsing Parent Objects

When working in the Project, Data Sources, Snippets, or Object Workspaces, or in the application file window, you can expand and collapse any parent object by one level using the left and right keyboard cursor keys.

A *parent* object is any object denoted in the view by the disclosure triangles: ▶, expandable; and, ▼, collapsible.

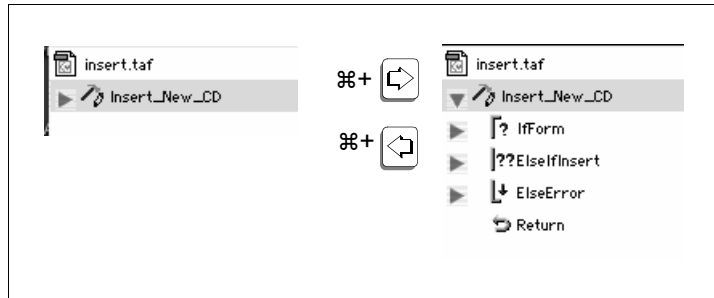
- To expand the selected parent one level, press  (right cursor key).
- To collapse the selected parent one level, press  (left cursor key).



You can also use keyboard shortcut keys in an open application file window to expand and collapse the parent object through all levels at one time.

- To expand the selected parent, press ⌘+ .

- To collapse the selected parent, press **⌘+⇧⬅**.



Witango Actions

Witango Actions are icon based representations of the logic within a Witango application file. Actions exist to deal with all strands of required logic to build a web application. Actions can be categorised into 4 different groups:

- Business Logic
- Database Logic
- Presentation Logic
- External Data Acquisition Logic.

Actions dealing with Business Logic

The Business Logic Actions control how the application flow. They are listed in the table below: The menu bar shows the menus and commands

Icon	Action	Function
	Assign	Makes specified value assignments to a variable.
	Group	Groups related actions together.
	IF, ELSE IF, ELSE	Executes an expression, and, based on the result of that expression affects the control of flow within the file.
	While Loop, For Loop	Repeats a set of contained actions: until an expression evaluates to true or for a specified number of loops.
	Break	Terminates processing within a loop.
	Branch	Causes a jump to another action or action group.
	Return	Ends execution of an application file and returns the accumulated Results HTML to the browser.

applicable to Witango.

Actions dealing with Database Acquisition Logic

The Database Acquisition Actions control the interaction with available databases including SELECT, UPDATE, INSERT and DELETE. Witango actions also exist to allow a developer to carry out ad hoc SQL statements or stored procedure calls with the Direct DBMS action. They

are listed in the table below: The menu bar shows the menus and

Icon	Action	Function
	Search	Retrieves records from a database.
	Insert	Adds records to a database.
	Update	Changes records in a database.
	Delete	Removes records from a database.
	Direct DBMS	Executes SQL statements.
 	Begin Transaction End Transaction	Begins a transaction and ends a transaction with a rollback or commit.

commands applicable to Witango.

Actions dealing with Presentation Logic

The Presentation Actions control how the results appear on the end user's browser. They are listed in the table below: The menu bar shows

Icon	Action	Function
	Results	Performs no special function of its own, this action allows HTML to be appended to the Results HTML.
	Presentation	Allows user to reference presentation pages.

the menus and commands applicable to Witango.

Actions dealing with External Data Acquisition Logic

The External Data Acquisition Actions control the interaction with back office systems, this is typically achieved with actions such as MAIL,

OBJECT, FILE and EXTERNAL. They are listed in the table below:

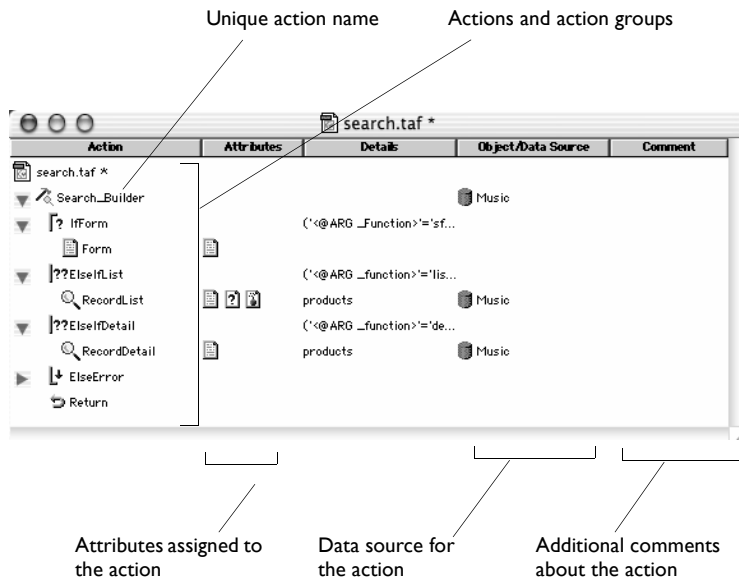
Icon	Action	Function
	Mail	Sends out electronic mail.
	File	Reads, writes and deletes files on the filesystem.
	Script	Used to specify server side JavaScript code to execute such as Shellsript.
	External	Calls an external code module to perform a function and return results.
	Create Object Instance	Creates object instances for COM, Java Beans, and Witango Class File Objects.
	Call Method	Calls methods on the object instances that are created.
	Objects Loop	Loops over collection objects

menu bar shows the menus and commands applicable to Witango.

Working With Actions

The application file window shows the actions that you want Witango Server to execute. Generally speaking, Witango Server executes actions sequentially, from top to bottom, until it encounters a control action. Control actions make decisions and cause execution to jump to another action or action group.

The following is an example of the application file window:



An action icon in the **Action** column indicates the type of action. Each action must have a name that is unique in the application file.

An action can have attributes. Action attribute icons in the **Attributes** column indicate which attributes are associated with the action on that row.

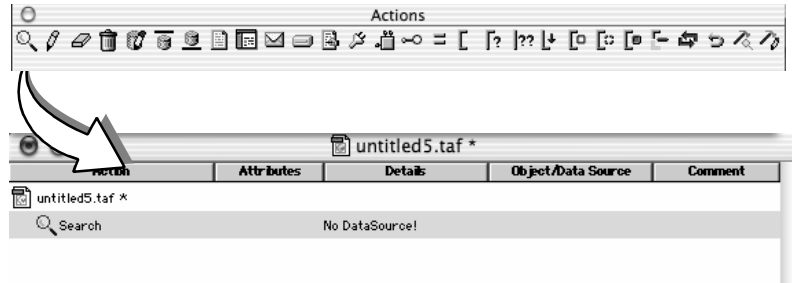
Some actions require database operations. The **Object/Data Source** column indicates which data source an action is associated with.

Adding an Action

To add an action to an application file

Do one of the following:

- Drag an action icon from the Actions palette into the application file window (the cursor changes to a closed palette hand and a box outline), and drop it where you want to add the action.



- Click an action icon, move the cursor into the application file window (the cursor changes to a closed hand), and click where you want to add the action.

In either method, a gray line indicates where the new action is to be placed.

If the action has an editing window, it opens automatically.



Tip To prevent the action's editing window from being opened automatically, hold down the **OPTION** key while dragging the new action into the document window.

Naming an Action

Each action in an application file must have a unique name. Witango Studio gives actions a unique name automatically.

The default name for an action is its action type. When you add an action that already exists in the application file with its default name, Witango appends the default name with a numeric starting at "1"; for example, "Search1".



Tip To make your application files more readable, you should always replace default action names with more meaningful ones.

To rename an action in an application file

- I Click the action's name in the application file window.

You can also `control+click` the name, and choose **Rename** from the contextual menu that appears.

2 Enter the new name.



Note Action names can contain only letters, numbers, and underscores. No spaces, punctuation, or other characters are allowed. Adding spaces automatically adds underscores.

When you rename an action, Witango automatically updates any Branch actions in the same application file referring to the action. If you rename an action that is the destination for branches from other application files, the Branch actions in other application files are *not* updated.

Witango does not automatically update action results references for renamed actions.

Deleting an Action

To delete an action from an application file

- 1 Select the action you want to delete.
- 2 Do one of the following:
 - From the **Edit** menu, choose Clear.
 - Press `Delete`.
 - `control+click` and choose `Clear` from the contextual menu that appears.
- 3 When the dialog box appears, asking you to confirm the deletion, click **OK**.



Tip You can bypass the confirmation dialog box by holding down the `option` key when choosing `Delete` or **Clear**.

Editing an Action

All of the actions—except Return, Group, and Break actions—have associated attributes and parameters. You can set these parameters in the action's editing window.

To edit an action in an application file

- Double-click the action icon in the application file window.
The action's editing window opens.

If the action is associated with a data source, the Data Sources Workspace opens, listing the tables and columns for the data source. If Witango Studio has not loaded the data source yet, it is loaded first.

Moving an Action

Witango executes the actions in an application file sequentially, from top to bottom; however, you can use control actions to modify this sequence.

If you want the actions to be performed in a different order, you can rearrange them. Move them to another location in the application file by dragging them to the position you want.

To move an action to a new location

- Select the action you want to move, and drag the action to its new position.

When you move an action, Branch actions referring to it continue to branch to the action, even though its position has changed.

Copying an Action

You may want to create an action that performs a task similar to one performed by an existing action in another application file. Instead of having to recreate the action and specify all its parameters again, Witango Studio allows you to duplicate an action.

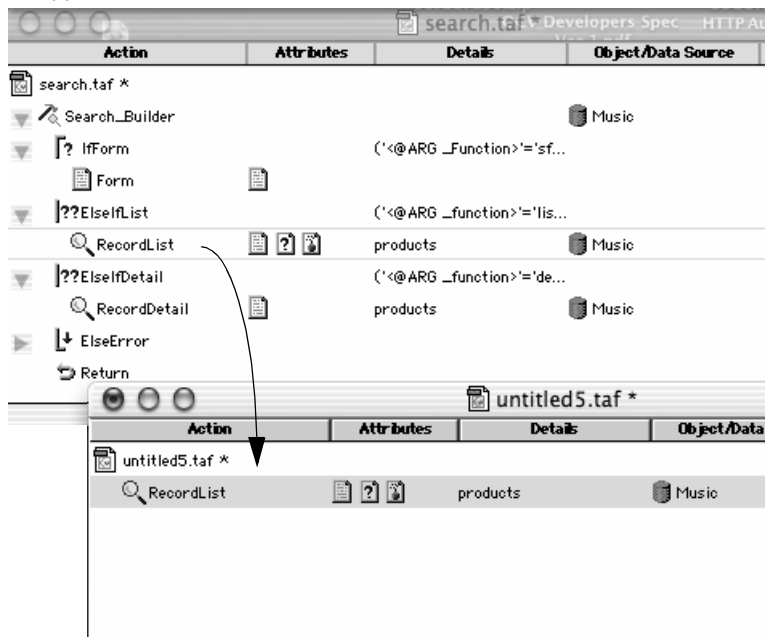
To copy an action in the same application file

- Select the action you want to copy, hold down the `option` key, and drag the action to where you want the new action to appear.

The copied action is given a new, unique name, which you should change to a more descriptive name.

To copy an action into another application file

- Select the action you want to copy, and drag the action into another application file.

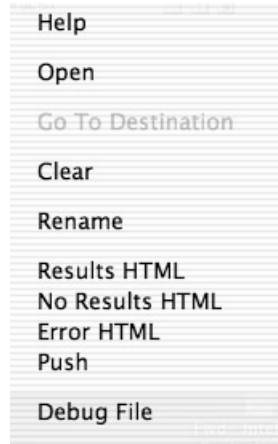


Be careful when copying database actions. For an action to work correctly in the new application file, the data source must be the same as in the original one.

Alternatively, you may assign another data source to the action in the new application file.

Contextual Action Menu

When you `control+click` an action icon in the application file window, or anywhere in the file window with an action selected, a contextual menu of action commands appears:

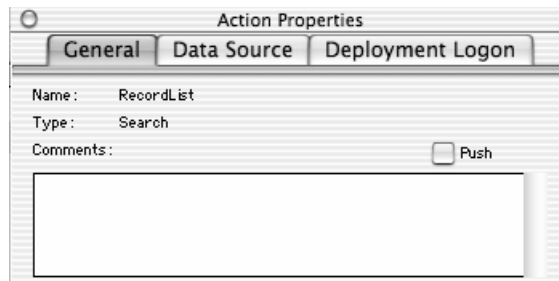


- **Help** accesses the HTML help files.
- **Open** opens the action editing window for the selected action.
- **Clear** performs the standard window editing function.
- **Rename** allows you to edit the current name of the action.
- **Results HTML**, **No Results HTML**, **Error HTML**, and **Push** are attributes you can assign to actions which support them.
- **Debug File** is an attribute of the entire application file or Witango class file.

Action Properties

When you select an action and choose **Properties** from either the Windows menu or the contextual menu, the Action Properties window for that action appears.

This window displays current information about the selected action and the assigned data source.



Using this window, you can change some of the action's properties.



Note The Data Source and Deployment Logon tabs are disabled when the action is not a database action.

Assigning Attributes to Actions

In addition to the parameters specific to each action type, which are edited using the action's editing window, actions can also have the following attributes:

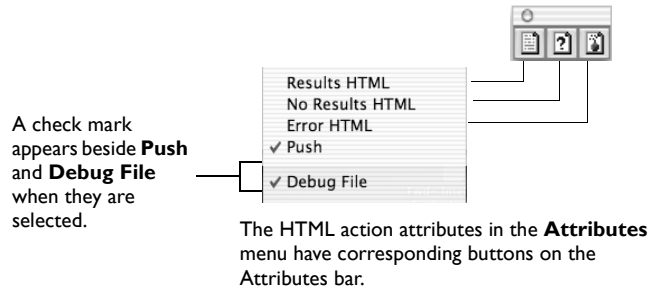
- **Results HTML** applies to all actions, except control actions (other than Branch). After the action is executed, this HTML is added to the results returned.
- **No Results HTML** applies only to Search, Direct DBMS, Script, File, and External actions. When the action does not return data, this HTML is returned instead of the Results HTML.
- **Error HTML** applies to most action types except certain control actions (including Return and Break). In the event of an error in the action's execution, this HTML is returned immediately.
- **Push** causes the Results HTML accumulated so far to be sent back to the Web browser when the action to which it is assigned finishes executing. Execution then continues normally.
- **Debug File** lets you see useful information about your application file or Witango class file execution in your Web browser application. This attribute applies to the entire application file, not a particular action. For more information, see *Debugging Files* on page 61.

To assign Results HTML, No Results HTML, Error HTML, or Push

Do one of the following:

- Select the action in the application file window, then select an attribute from the **Attributes** menu or from the Attributes palette.

- **control+click** the action in the application file window and choose the attribute that applies to the selected action from the contextual menu that appears.



Action attribute icons appear beside the action name in the **Attributes** column of the application file window..

You can switch between the Results HTML, No Results HTML, and Error HTML associated with an action by clicking on the tabs at the top of the HTML editing window.



Results HTML

Many actions in an application file can have HTML associated with them. This HTML is stored in the Results HTML attribute. If Results HTML contains any text, the Results HTML icon appears in the attributes column of the application file window; otherwise, it does not.

As Witango Server executes the actions in a file, the Results HTML associated with each is accumulated. When execution of the file is complete, the HTML is returned.

Results HTML can also contain Witango *meta tags* that Witango Server processes. While all the other text in Results HTML is interpreted by the user's Web browser and returned as is (via the Web server), Witango Server first substitutes meta tags with other values.

The `<@COLUMN>` meta tag causes a database value to be placed in the HTML. There are many others, including tags for referencing form field and search argument values, and conditional tags for displaying HTML only if the result of a given comparison is true.

To create or edit the Results HTML for an action

- 1 Select the action in the application file window.
- 2 Do one of the following:
 - From the **Attributes** menu, choose **Results HTML**.

- Click the **Results HTML** icon on the Attributes palette.
- `control+click` the action and choose **Results HTML** from the contextual menu that appears.

The Results HTML editing window appears:



- 3 Type the Results HTML into the HTML text area. The text can include any valid HTML or Witango meta tags.

You can switch between the Results HTML, No Results HTML, and Error HTML associated with an action by clicking on the tabs at the top of the HTML editing window.

You can add column values (for Search actions only) and any HTML snippets you have defined to the Results HTML editing window from the Snippets Workspace. As well, you can add from the list of standard Witango snippets that allow for easy entry of many of the meta tags.

To include any of these items in your Results HTML, select the snippet and either drag it, or copy and paste it into the desired location in your text.

For HTML snippets that have placeholders for the current selection, select the text and drag the snippet over the selected text. The snippet is wrapped around the selection. For example, "Title" becomes "<H1>Title</H1>".

-
- I Witango does not restrict its content to only HTML format. Using other markup languages such as SGML, VRML, and XML instead of HTML is also possible. If you use other content types, you are responsible for setting the HTTP header appropriately.

You can also easily add many of the common Witango meta tags.

To add a meta tag

- 1 Click the editing area where you want to add a meta tag.
- 2 Do one of the following:
 - From the **Edit** menu, choose **Insert Meta Tag**.
 - `control+click`, and choose **Insert Meta Tag** from the contextual menu that appears.

The Insert Meta Tag dialog box appears. For information on using the Insert Meta Tag dialog box.



No Results HTML

You can associate No Results HTML text with Search, Direct DBMS, Script, and External actions. If the action execution does not return any data, this text is added to the application file's accumulated HTML instead of the Results HTML. This is useful when you want to display a special message to users when their queries do not return data.



Note If both Results HTML and No Results HTML appear as attributes, Witango accumulates one or the other, but never both.

After Witango Server processes the No Results HTML, execution of the application file continues normally to the next action.

No Results HTML can contain any of the Witango meta tags used in Results HTML, except for those related to displaying result data items, such as `<@ROWS>`, `<@COLUMN>`, and `<@COL>`.

To create or edit the No Results HTML for an action

- 1 Select the appropriate action in the application file window (Search, Direct DBMS, Script, and External actions).
- 2 Do one of the following:
 - From the **Attributes** menu, select **No Results HTML**.
 - Click the No Results HTML icon on the Attributes palette.
 - `control+click` the action and choose **No Results HTML** from the contextual menu that appears.

The No Results HTML editing window appears.

- 3 Type the No Results HTML into the HTML text area. The text can include any valid HTML or Witango meta tags.



Error HTML

Error HTML allows you to specify your own error messages in HTML format, instead of having Witango Server produce them. The other alternative is to modify the `Error.htx` file.

You can associate Error HTML with most actions. If an action fails for any reason, execution ends and the Error HTML for the action is returned immediately to the user.

Error HTML can contain all the Witango meta tags used in Results HTML, except for those related to displaying result data items.

There are also special Witango meta tags for displaying error information.

If no Error HTML has been assigned to an action and an error occurs in that action, Witango returns a default error message using the following HTML:

```
<h3>Error</h3>

An error occurred while processing your request:<p>
<@ERRORS>
Position: <b><@ERROR PART=POSITION></b><br>
Class: <b><@ERROR PART=CLASS></b><br>
Main Error Number: <b><@ERROR PART=NUMBER1></b><br>
<@ifequal <@ERROR PART=NUMBER2> 0>
<@else>
    Secondary Error Number: <b><@ERROR
PART=NUMBER2>
</b><br>
</@ifequal><p>
<i>
<@ERROR PART=MESSAGE1><br>
<@ifequal @ERROR PART=MESSAGE2> ">
<@else>
    @ERROR PART=MESSAGE2><br>
</@ifequal><p>
</i>
</@ERRORS>
```

To create or edit the Error HTML for an action

- 1 Select the action in the application file window.
- 2 Do one of the following:
 - From the **Attributes** menu, select **Error HTML**.
 - Click the **Error HTML** icon on the Attributes palette.
 - `control+click` the action and choose **Error HTML** from the contextual menu that appears.

The Error HTML editing window appears.

- 3 Type the Error HTML into the HTML text area. The text can include any valid HTML or Witango meta tags.

To specify your own custom default error message

- 1 Create a text file containing the desired HTML and meta tags.
- 2 Name the file `error.htx`.
- 3 Save or copy it to the `Misc Files` folder under the `root` folder of your Witango server.

If Witango Server finds this file, it processes and returns it instead of the built-in default Error HTML. Error HTML assigned to an action is used if it exists.

The name and location of this file is determined by the `defaultErrorFile` configuration variable, which can be modified using the Administration Application `config.taf`. The values when Witango is first started are given above. If you modify the path or name of the error file, place the file in the directory you specified instead.

Push

The **Push** attribute causes the Results HTML accumulated so far to be sent back to the Web browser, when the action to which the **Push** attribute is assigned finishes executing. Execution then continues.

Normally, Witango waits until all execution is finished before returning the results at one time. If you want the user to see some of the results while Witango continues with the rest of the execution, set the **Push** attribute of the action.



Note Some Web browsers may not display table HTML immediately if you use the Push attribute to return an unclosed table.

Debug File

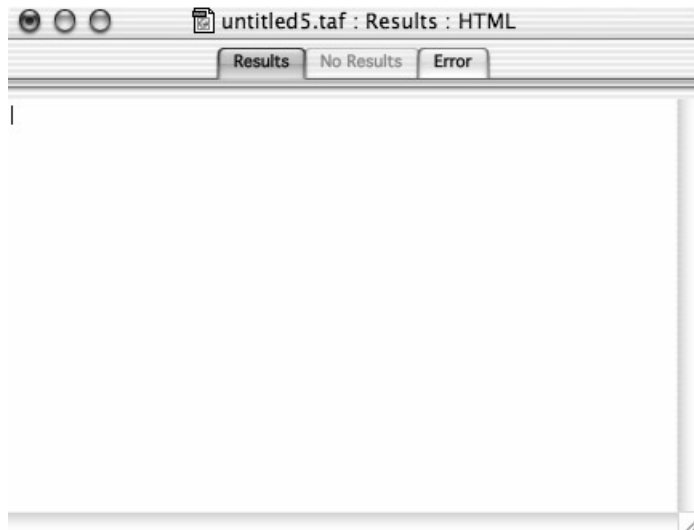
For more information, see [Debugging Files](#) on page 61.

Adding HTML (Results Action)



Results Action

The Results action adds HTML to an application file's results.



Results HTML can contain Witango meta tags that Witango Server processes. While all the other text in Results HTML is returned as is to your Web browser (via the Web server), any meta tags are first substituted with other values by Witango Server. You can also associate Error HTML with the Results action.

Presentation Action

Uses of the Presentation Action

The main benefit of using the Presentation action is to facilitate the separation of the business logic from the presentation logic when you develop your Witango application.

Business logic involves the use of Witango actions and meta tags to access the appropriate Web pages and data sources. *Presentation logic* involves the use of HTML to display the Web pages.

Because developing the business logic and the presentation logic generally require different skill sets, setting up independent teams to work on these two areas can improve the effectiveness and efficiency of the project. Furthermore, changing the business logic—for example, accessing a different data source—often does not affect the presentation logic, or vice versa. Keeping the two areas separate simplifies the maintenance of your project.

A Presentation action in your application file points to an HTML page. It is the link between the business logic and the presentation logic of your project.

The *Document Object Model* (DOM) allows you to create your own complex data structures in XML, and return them into presentation pages.

How the Presentation Action Works

The Presentation action allows you to include individual *presentation pages* in your Witango application file. The presentation page—the file the Presentation action points to—can contain HTML, Witango meta tags, or any other sort of document markup. When Witango Server executes your application file and arrives at a Presentation action, it processes the presentation page associated with the Presentation action.

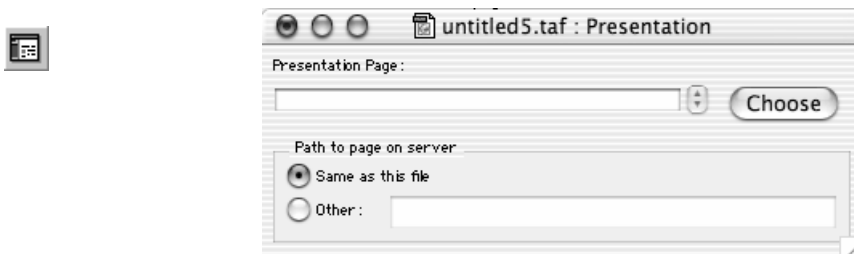
The Presentation action performs an operation similar to that of including an HTML or other file in a Witango application file using the `<@INCLUDE>` meta tag.

The file referenced by the Presentation action is part of the current project, and can be opened and edited by double-clicking on the file icon within the **Presentation Pages** folder in the **Project** section of the Workspace.

You can also designate files in your project as presentation pages, and manage files within the **Presentation Pages** folder.

Setting Up a Presentation Action

When you drag the Presentation action from the Actions bar into an application file, the Presentation dialog box appears:



Do one of the following:

- In the **Presentation Page** field, enter the name of the presentation page, or if you have previously specified a presentation page in the current Project, choose a file name from the pop-up menu.
- Click Choose to navigate to the location of the presentation page.

If the file is not in your current project, you are prompted to add it to the project, where it appears in the **Presentation Pages** folder and in the **Files** folder of the **Project** tab of the Workspace.

In the **Path to page on server** area, select **Same as** this file if the presentation page is located in the same folder as the current application file, or select **Other**.

If you choose **Other**, you specify the path to the presentation page. This value is a slash-separated path from the Web server document root, and may include literal text, meta tags, or both. To insert a meta tag in this field, `control+click` in the text field and choose **Insert Meta Tag...** from the contextual menu that appears.

For example, you could enter the following into the text field:

```
Witango/MyDirectory/
```

This example includes the specified file residing in the `MyDirectory` folder within the `Witango` folder in the Web server document root folder.

```
<@APPFILEPATH>
```

This example includes the specified file residing in the same folder as the currently-executing application file.

Using Witango Application Files

A *Witango application file* (or simply, *application file*) provides a powerful and flexible means for you to construct dynamic applications that run on your Web server and that interact with databases, other applications, and users running Web browsers. They are like programs or scripts in that they determine what operations Witango Server performs. Witango Server provides the brains, but it does nothing without the specific instructions you provide in the form of application files.

You add actions to an application file. When Witango Server runs the application file, it generates the HTML that is used by the Web browser to display the forms required to allow interaction with databases and other applications.

You can use the Search Builder and New Record Builder to have Witango Studio build search and insert record applications for you.

An application file is a file containing a series of Witango actions that, when executed by Witango Server, generates HTML and controls interaction with databases and other applications.

(You can also create *Witango class files*, which are reusable software components that you can incorporate in Witango application files.

XML Format

Witango application files and Witango class files are stored in an *Extensible Markup Language (XML)* format, which means they are structured text based on a specific document type definition. This is a substantial change from the binary formats of files in previous versions of Witango.

What is XML?

XML is a text-based and widely-endorsed standard markup language, similar to HTML, but much more flexible and robust. It is a subset of SGML (Standard Generalized Markup Language), an ISO standard. Its goal is to enable generic SGML (that is, structured documents) to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

For details about the XML file format, see www.w3.org/xml/.

Witango XML file formats give Witango users the following advantages:

- XML files are human-readable.

- Text-processing tools can be used on Witango application files to perform file differences, complex searches involving regular expressions, and so on.
- The Witango XML file format is now public and exactly specified, so other applications can create Witango application files and Witango class files.

SGML and XML specifications require a *document type definition (DTD)*. The DTD defines the structure of the various elements that make up an XML document. It ensures that all applications that read and write the XML document do so consistently way. In effect, it is the schema of the document.

For more information about document type definitions and how to read them, see www.oasis-open.org/cover/sgml-xml.html.

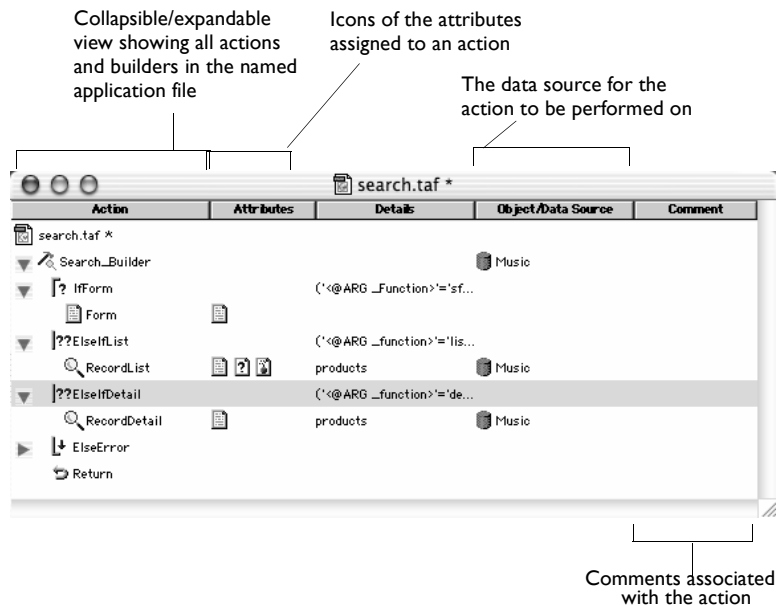
The Witango DTD for Witango application files and Witango class files is specified by the file `Witango.dtd`. This file is located in the `XML` folder inside the folder where Witango is installed.

Application File Window

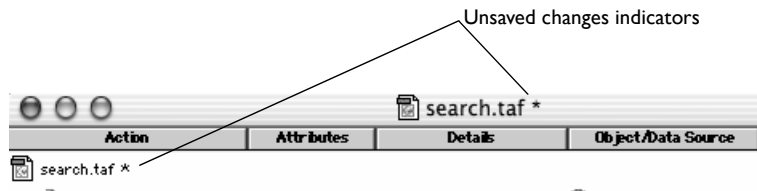
In Witango Studio, whenever you open an application file, the Witango application file window (or simply, application file window) shows you the following information:

- action icons and names, including those for builders, in the order Witango Server executes them (unless a control action redirects the flow of the execution)
- attributes assigned to an action, if any
- data sources for all database actions
- any associated comments.

The application file window also includes icons for attributes, objects, and data sources. The following diagram shows a typical application file window and its components: Unsaved Changes Indicator



Whenever you change a Witango application file or class file, and the file has not been saved, an asterisk appears beside the file name in the title bar of the window and beside the filename in the window. This is called a *dirty* (unsaved changes) indicator.



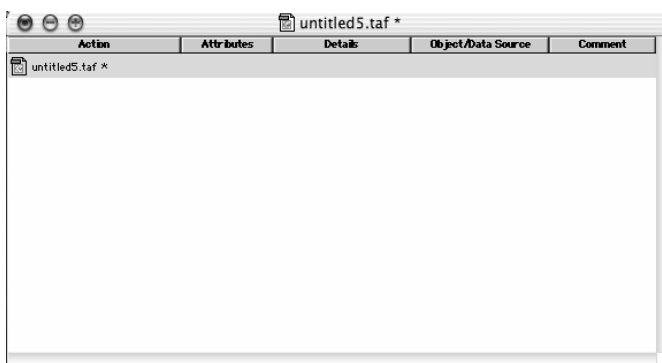
Once you save the application file, these indicators disappear.

Creating an Application File

To create a new application file

- From the **File** menu, choose **New**, then **Witango Application File**.

An untitled application file opens:



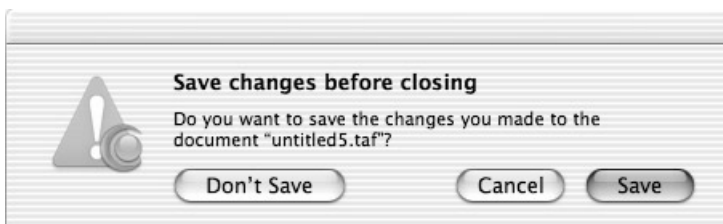
Saving an Application File

To save an application file

- 1 From the **File** menu, choose **Save**.

If the application file has never been saved, the Save As dialog box appears.

If it has been saved previously, Witango Studio saves it using the existing name and location.



- 2 Navigate to the desired location for the application file.
For Witango Server to execute the application file, it must be located in or below the Web server's document root folder.
- 3 Name the Witango application file.
- 4 Witango application file names end in `.taf`. This is the standard suffix used to identify files that Witango Server should execute. Click **Save**.



Tip To save *all* open Witango application and text files with their current name and location, choose **Save all** from the **File** menu, or click the **Save All** icon on the Witango Toolbar. The Save As dialog box appears for new, unnamed files.

Saving a Witango Application File or Witango Class File as Run-Only

Run-only Witango application files and Witango class files can be executed by Witango Server, but they cannot be opened by Witango Studio.

Saving an application file or Witango class file as run-only allows you to create and distribute packaged Witango solutions while preventing users from editing the actual application file.

Run-only application files and Witango class files are executed and referenced by Witango Server in the same way as editable files. Saving an application file or Witango class file as run-only does not make its execution any faster.



Caution You cannot edit a run-only copy of an application file or Witango class file, and there is no way to make a run-only file editable. Make sure you keep an editable copy of any run-only file.

To make an application file or Witango class file run-only

- 1 With an application file open in Witango Studio, choose **Save As Run-Only** from the **File** menu.

The Save As dialog box appears.

You are saving a copy of your Witango application file or Witango class file as run-only. Your original application file or Witango class file is not changed.

- 2 Name the run-only Witango application file or Witango class file.



Tip You may want to give the run-only versions of your files a special name to identify their type, such as `CustomersRO.taf` or `CustomerRO.tcf`, where “RO” represents run-only.

- 3 Click **Save**.

A run-only version of the application file or Witango class file is saved in the location you specified.



Note If you are distributing your Witango solution, your customers need to purchase Witango Server. Alternatively, you can license Witango Server for distribution with your solution. Contact sales@witango.com for more information.

Executing Application Files

Application files are executed in the same way HTML files are viewed—by specifying the name of the file in a URL. For example:

```
http://localhost/shop/additem.taf
```

This example executes an application file called `additem.taf`, located in the `root` directory of your local webserver. If you are using the Witango CGI, you may need to include the Witango CGI in your URL, for example:

```
http://www.example.com/witango.cgi$/additem.taf
```

You can pass parameters to the application file by using *search* arguments. These are name-value pairs appearing after a question mark in the URL. For example:

```
http://www.example.com/shop/additem.taf?item_num=80
```

In this example, the `item_num` search argument has a value of “80”.

There are other ways of passing values to Witango application files. *Form* fields (post arguments) and *cookies* are two examples.

Debugging Files

Setting the debug mode in Witango Studio lets you see useful information about your application file or Witango class file execution in your Web browser application.

Turning Debug On

To set *debug mode*

- 1 Open the application file or Witango class file you want debug information on.
- 2 Do one of the following:
 - From the **Attributes** menu, select **Debug File**.

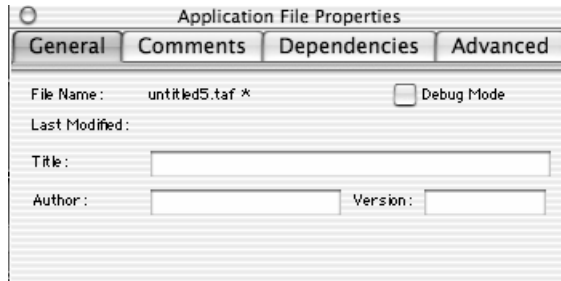


A check mark beside the command indicates the debug mode is on.

- `control+click` the application file window, and select **Debug File** from the contextual menu that appears:

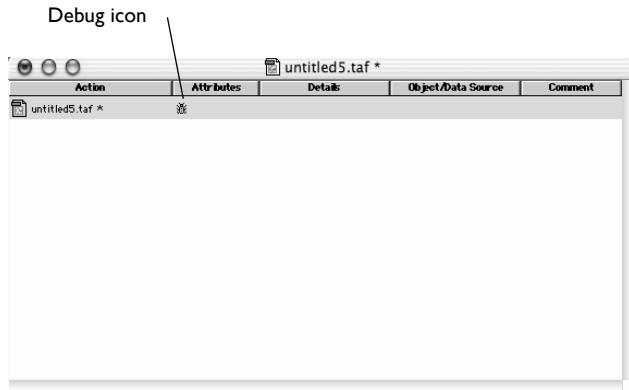


- From the Windows menu, choose **Properties**. Then enable **Debug Mode** in the Application File Properties dialog box that appears:



- Check the Debug Checkbox.

A debug icon appears beside the application file icon when **Debug File** is checked.



Viewing Debug

When you execute the application file, debugging information appears at the bottom of the results returned. The debugging information shows information such as:

- arguments passed in (search and post arguments)
- the actions executed
- values of variables
- SQL generated by database actions
- warnings (such as references to missing arguments).

The debug feature is extremely helpful in tracking the flow through a .taf when the output of the file is not what the programmer is expecting.

Using Projects and Source Control

The Basics of Witango Projects and Managing Files Using Source Control

A *project* is a logical grouping of folders and files. Projects allow you to organize your work in terms of like-sets of files, including application, HTML, and text files—in fact, for any type of file. Projects exist in Witango Studio only and do not interact with Witango Server.

This chapter covers the following topics:

- Working with Witango projects
 - adding and removing project files and folders
 - project dependencies
 - application-specific Witango (AST) signatures for projects

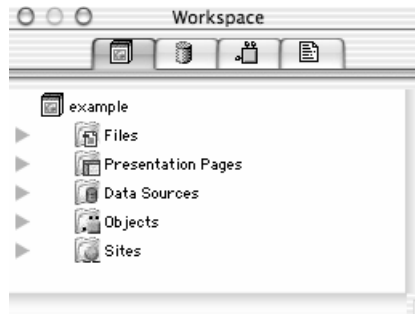
Basics of Witango Projects

When you create a new project or open an existing project, the Project Workspace (Project section of the Workspace) displays the project name and the folders included in the project. The project name is the file name you assigned to the project prefixed to the word “Project”.



Note To see the Project Workspace tab you must first open or create a Project.

The Project Workspace allows you to work with all files, data sources, objects, and resources associated with your Witango project without having to switch tabs in the Workspace.



The following five folders are always displayed at the root level in the Project Workspace and cannot be deleted:

- **Files**
This folder contains all the files referenced in your project. The files in this folder may be organized into a hierarchy of subfolders.
- **Presentation Pages**
This folder contains all the files in your project that you want to designate as presentation pages. Presentation pages are HTML, graphic, or text files available for use with Presentation actions. All the files listed under this folder are also listed under the **Files** folder.
- **Data Sources**
This folder lists the data sources used in your project.
- **Objects**
This folder lists the objects used in your project.

For more information, see “Working With Presentation Pages” on page 77

For more information, see “Working With Project Data Sources” on page 78

For more information, see “Working With Project Objects” on page 79

For more information, see “Working With Project FTP Sites” on page 79

- **Sites**

This folder lists the FTP (file transfer protocol) sites associated with the current project for the deployment of project files.

Understanding the Project File

The project file contains information on your project, including a listing of the project’s folders and files (in the Files folder).

The purpose of the project file is to help you manage your project. It contains pointers to all the folders and files that you include in the project; it does not contain the actual folders and files.

You perform operations on the project file separately from the folders and files it contains; that is, deleting a file from the project removes it from the project file, but does not actually delete the file.



Note Path names of files stored in the project file are stored relative to the project file’s location; that is, if you move the project file, the files within it will not be found.

Using the Project Workspace

For more information on setting Witango Studio preferences, see Setting Preferences on page 129.

For more information on files under source control, see Using Source Control in Witango on page 97 and Modifying a File Under Source Control on page 109.

Opening Files in the Project Workspace

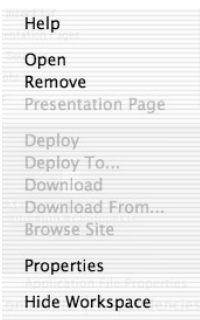
You can open any file appearing in the Project Workspace simply by double-clicking the file name.

The file automatically opens and displays its contents in the application defined by its Windows suffix mapping. Witango application files automatically open in Witango Studio; if you set Witango Studio as the default Studio in the Preferences dialog box, HTML and text files also open in Witango Studio.

If you try to open an application file that is currently under source control and not checked out, Witango Studio prompts you to check it out first.

Using Context-sensitive Menus

You can also conveniently execute certain project commands directly in the Workspace. Right-clicking a project, folder, or file displays a menu of project, folder, and file commands and Workspace window commands.

Project CONTROL+clicking the project name or icon displays a menu of commands	Folder CONTROL+clicking the folder displays a menu of commands	File CONTROL+clicking a file displays a menu of commands
 A screenshot of the context menu for a project. It includes options like Help, Add Files..., Close, Modify Project AST Signature..., Add Site, Deploy, Deploy To..., Download, Download From..., Browse Default Site, Properties, and Hide Workspace.	 A screenshot of the context menu for a folder. It includes options like Help, Add Files to Folder..., New Folder, Properties, and Hide Workspace.	 A screenshot of the context menu for a file. It includes options like Help, Open, Remove, Presentation Page, Deploy, Deploy To..., Download, Download From..., Browse Site, Properties, and Hide Workspace.



Note Source control commands only appear in Witango Studio if you have a supported source control system installed on your machine.

Moving Files and Folders in the Project Workspace

You can move folders and files within the Project Workspace by dragging them to a new location within the **Files** folder.

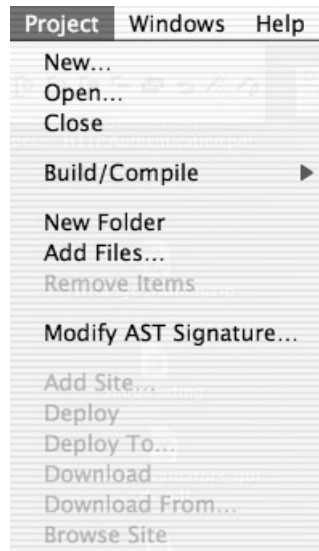
Dragging a file to a folder adds that file to the target folder. Dragging a folder to another folder makes it—and any files in it—a subfolder of the target folder.

Creating a New Project

To create a new project

- I Do one of the following:

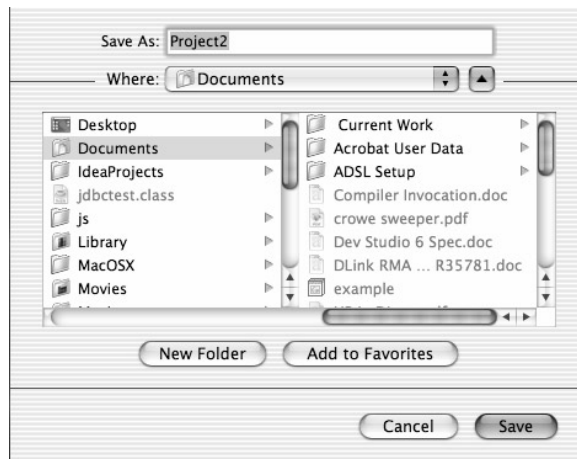
- From the **Project** menu, choose **New**.



New Project

- Click **New Project** on the main toolbar.

The Create a Project File dialog box appears:



- 2 Specify a project file name and location.

Project file names end in `.tep`. This is the standard extension used to identify the file that lists the folders and files forming a project.

- 3 Click **Save**.

The project name appears in the Project Workspace.

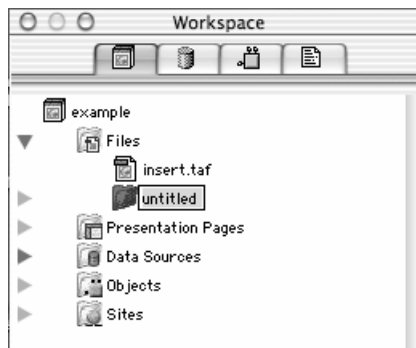
Adding a Folder to a Project

You can add a new folder to the **Files** folder or **Presentation Pages** folder of a project. You can also add an existing folder to the **Files** folder. You cannot add a folder to the other Project folders.

To add a new folder to the Files folder or Presentation Pages folder

- 1 Select the **Files** folder or **Presentation Pages** folder.
- 2 Do one of the following:
 - From the **Project** menu, choose **New Folder**.
 - Right-click the **Files** or **Presentation Pages** folder of the project, and choose **New Folder** from the context-sensitive menu that appears.

When you add a new folder, the name `New Folder` appears under the **Presentation Pages** or **Files** folder. Witango may add a suffix to the default name (for example, `New Folder 2`) to make the name unique. This default name is automatically selected for easy renaming. A folder name must be unique at the level you are adding the folder.



To add an existing folder to the Files folder

- From the Windows Explorer, drag an existing folder into the **Files** folder or any of its subfolders.



Note You cannot drag a folder from the Windows Explorer into the **Presentation Pages** folder or any of its subfolders.

All the subfolders and files within this existing folder are added to the project at the specified location.

A folder name must be unique at the level you are adding the folder; rename a folder if necessary.

To rename a project folder

Do one of the following:

- Click the name of the folder; click the name again.
- Right-click the folder icon or name and choose **Rename** from the context-sensitive menu that appears.

Adding Files to a Project

For more information, see “Working With Presentation Pages” on page 77

You can add files to the **Files** folder from the Windows Explorer. You cannot add a file to the **Files** folder if that filename already exists somewhere within the **Files** folder; rename files if necessary.

You cannot add files to the **Presentation Pages** folder from the Windows Explorer; however, you can designate certain files in the **Files** folder as presentation pages.

Filenames appear alphabetically in the **Files** folder. The order of application files in this folder has no bearing on the order that Witango Server executes them.

For more information on setting source control preferences, see Source Control on page 159.

If you checked the **Prompt to add files when inserted into a project** option in Witango Studio’s source control preferences, you are prompted to add the files to source control. Click **Yes** to add the files, or click **No** to cancel.

To add files to the Files folder or its subfolder

- 1 Select the **Files** folder or one of its subfolders.
- 2 Do one of the following:
 - From the **Project** menu, choose **Add Files...** Go to step 3.
 - Right-click the **File** folder or its subfolder, and choose **Add Files to Folder...** from the context-sensitive menu that appears. Go to step 3.
 - From the Windows Explorer, drag one or more files into the **Files** folder or its subfolder. Go to step 4.
- 3 The Add Files into Project dialog box appears. The types of file selection supported include the following:

Type	File Extension
Witango Application Files	*.taf
Witango Class Files	*.tcf
Witango Application and Query Files	*.taf, *.qry
Text Files	*.txt, *.html, *.xml, *.dtd, *.inc, *.java

Type	File Extension
Graphics Files	*.gif, *.jpg
All Files	*.*

Select the files you want to add to the project and click **Open**.

- The added files appear in the **Files** folder or its subfolder.

Removing Files and Folders From a Project

You can remove files and folders from the **Files** folder. When you remove a folder, you remove it along with all its subfolders and files.

Removing a file from a project does not delete the file. The file remains intact so you can use it again or add it to another project.

To remove files and folders from a project

- Select the files or folder you want to remove.
- Do one of the following:
 - From the **Project** menu, choose **Remove**.
 - Click the Delete icon on the Toolbar.
 - Press `Delete`.
 - Right-click the file or folder, and choose **Remove** from the context-sensitive menu that appears.

A message appears, asking you to confirm that you want to remove the selected item(s).

- Click **Yes**.

Opening and Closing a Project

To open an existing project

- From the **Project** menu, choose **Open**.

Only one project can be open at a time. If another project is already open, Witango closes it and then opens the selected project. Any changes that you made to the project being closed are automatically saved.

When you open a project, the last view state is restored; that is, folders appear expanded or collapsed as they did previously.

To close an open project

- From the **Project** menu, choose **Close**.

Any changes you make to an open project are automatically saved as you make them.

Editing HTML and Text Files

In addition to Witango application files and Witango class files, a project file can include any other type of file. For HTML and text files, Witango has built-in editing capabilities. (See *HTML Editing Window* on page 6.)

When you open any file included in a project that has a Witango extension, Witango's HTML editing window opens (if you check the **Open text files in projects using Witango Studio** option in Witango Studio's source control preferences). Otherwise, Witango launches the **Opens with** application you have specified in the Windows Explorer for that file type.

If a project is open when you save an HTML or text file in Witango, you are automatically asked if you want to add the file to the current project. Click **Yes** to add the file to the project root or **No** to cancel.

Finding and Replacing Text in Projects

For more information, see *Finding and Replacing Text* on page 27.

One of the powerful editing features of Witango is its ability to find and replace character strings in all files—Witango application files, Witango class files, HTML, and text—of a project. The project must be open for the find-and-replace operation to take place in the applicable files of the project; all non-text files are ignored. If Witango finds the specified text string, it automatically opens an editing window showing the corresponding file or HTML attribute for an application file.

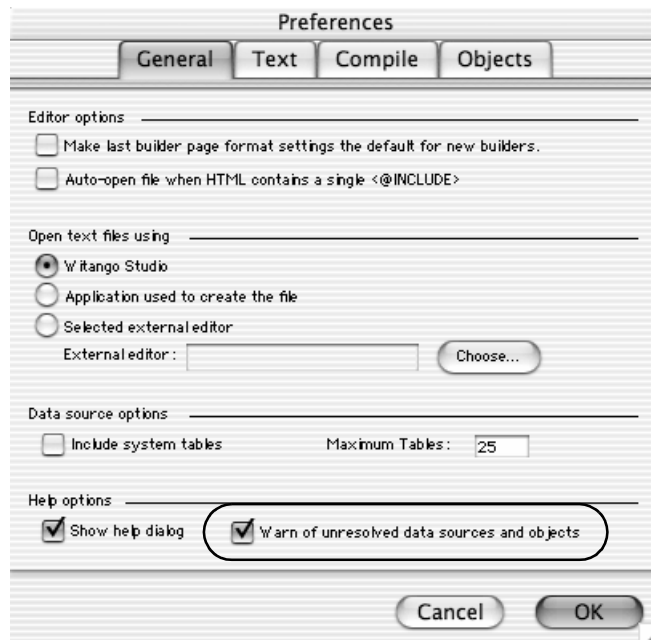
Additional Features of Witango Projects

Working With Project Dependencies

Dependencies are those data sources and objects that are used or referenced by Witango application files and Witango class files in the project. Witango Studio shows the data source and object dependencies of your project, warns you of unresolved dependencies (if enabled), and helps you resolve them.

To enable unresolved dependency notification

- 1 From the **Edit** menu, choose **Preferences**.
- 2 Select the **General** tab and check **Warn me about unresolved data sources and objects**.



When Witango Studio detects an unresolved dependency (for example, when Witango Studio tries to expand an unresolved item or open an action that uses an unresolved item for the first time), the warning message will appear.

An unresolved dependency has a grayed-out icon. Click **OK** to close the Unresolved Dependencies dialog box.

To resolve a dependency

- 1 In the Project Workspace, right-click the unresolved item and choose **Resource Dependency...** from the context-sensitive menu that appears.
- 2 Do one of the following:
 - For a data source, Witango prompts you to resolve the dependency. Clicking **Yes** opens the Create New Data Source dialog box for the type of data source requiring resolution.
 - For a JavaBean or Witango class file, Witango prompts you to locate the object. Clicking **Yes** opens a File Open dialog box. Navigate to the unresolved item and click **Open**.

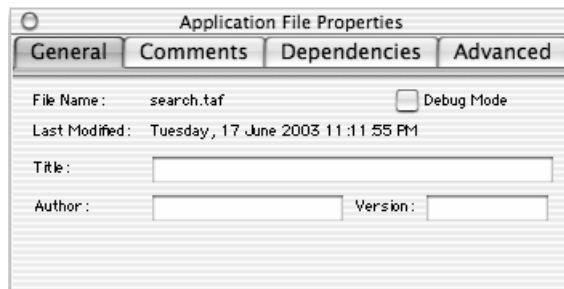
Working With Application Files

The **Files** folder displays all the files used or referenced by your project. You can organize files by creating new folders and moving files to appropriate folders within the **Files** folder.

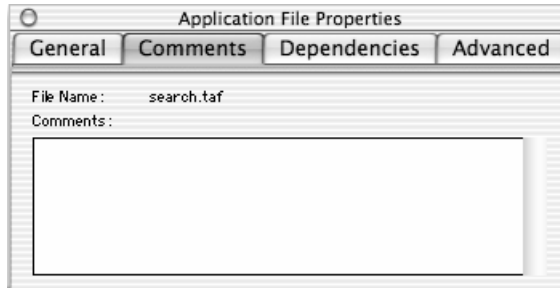
Application File Properties

The Application File Properties dialog box allows you to view information about a selected application file. The Application File Properties dialog box displays four tabs for a Witango application file in a project.

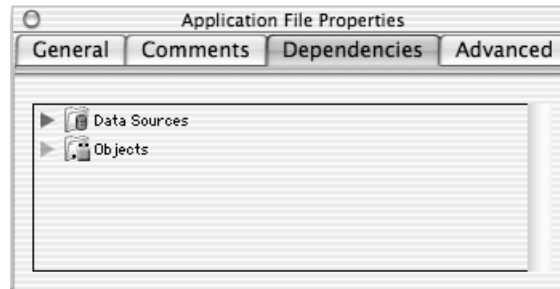
- **General.** This section displays the name of the application file and last modified date. A check box allows you to select or deselect Debug mode. Enter Title, Author, and Version information in the appropriate fields.



- **Comments.** This section allows you to enter comments about the application file in the text field.

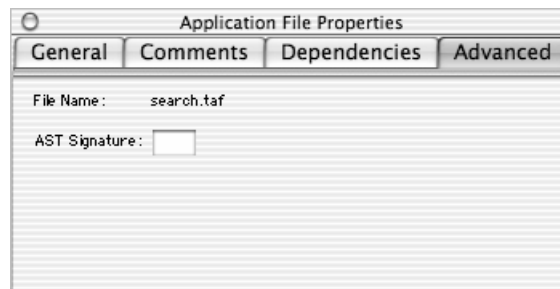


- **Dependencies.** This section displays the data sources and objects referenced by the application file. Unresolved dependencies are identified by grayed-out icons. This section cannot be modified.



For more information, See “Modifying a Project’s AST Signature” on page 79.

- **Advanced.** This section allows you to enter an AST signature for the application file in the **AST Signature** field.



Caution An AST signature assigned to a project application file’s advanced properties will be overwritten by changes to the project’s AST signature, which is assigned through the Advanced section of the Project Root Properties dialog box. For more information, See “Project Root Properties” on page 81.

Working With Presentation Pages

For more information about the Presentation action, see Presentation Action on page 243.

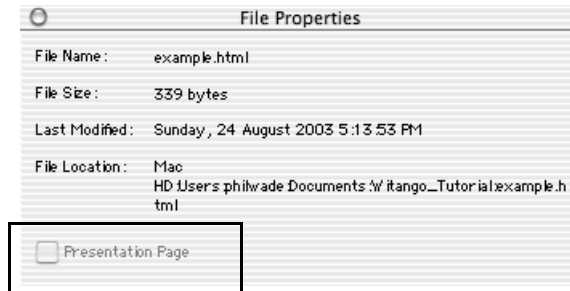
The **Presentation Pages** folder separates presentation pages from Witango application files, Witango class files and other HTML, graphic or text files in the project, allows page-based editing, and makes these files available to the interface of the Presentation action.

When you assign files to this folder, they are designated as presentation pages, but also remain listed in the **Files** folder or its subfolders.

To mark an HTML or text file as a presentation page

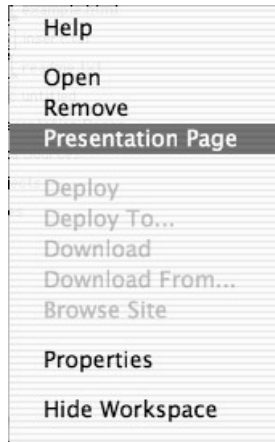
Do one of the following:

- Right-click on an HTML or text file in the **Files** folder of the Project Workspace, and choose **Properties** from the context-sensitive menu that appears. The File Properties dialog box appears.



Check the **Presentation Page** checkbox.

- Right-click on an HTML or text file within the **Files** folder of the Project Workspace, and choose **Presentation Page** from the context-sensitive menu that appears.



A check mark appears next to **Presentation Page**.

Marking a file as a presentation page adds it to the Presentation Pages folder.

- Select an HTML or text file in the **Files** folder of the Project Workspace, and drag it to the **Presentation Pages** folder.

To remove a file from the Presentation Pages folder

Do one of the following:

- Right-click on a file in the **Files** folder of the Project Workspace, and choose **Properties** from the context-sensitive menu that appears.

Uncheck the **Presentation Page** check box in the File Properties dialog box that appears.

- Right-click on a file within the **Files** folder of the Project Workspace, and deselect **Presentation Page** from the context-sensitive menu that appears.
- Drag a file out of the **Presentation Pages** folder.

Working With Project Data Sources

The **Data Sources** folder contains an alphabetically-sorted list of data sources that are used in your project. Unresolved dependencies are identified by grayed-out icons. This folder cannot be modified directly.

Working With Project Objects

The Project **Objects** folder contains an alphabetically-sorted list of objects that are used in your project. Unresolved dependencies are identified by grayed-out icons. This folder cannot be modified directly.

Working With Project FTP Sites

The **Sites** folder lists the FTP (file transfer protocol) sites associated with the current project for the deployment of project files. You associate an FTP site with your project by defining a site in the Define Sites dialog box and adding it to your project.

To view details about a particular site, right-click on a site icon in the Project Workspace, and choose **Properties** from the context-sensitive menu that appears. The details are presented in the Project Site Properties dialog box.

Application-Specific Witango (AST) Signatures for Projects

Application-specific Witango Servers are available if you want to develop a Witango application and distribute it with a Witango Server as an all-in-one solution. This allows your end-user to execute your solution without having to purchase a Witango Server for your single application.

The AST Server works only with the Witango application files in the licensed application with the assigned AST signature. You must add this signature to all Witango application files used in the application in order for them to be executed by the AST Server. Witango application files without an AST signature, or with a different AST signature, do not work with the AST Server.

Contact sales@witango.com sales for information on purchasing an AST license for your application.

Modifying a Project's AST Signature

An application file may belong to more than one project, but it can have only one AST signature. An application file added to a project will lose any signature it previously had as a member of another project.

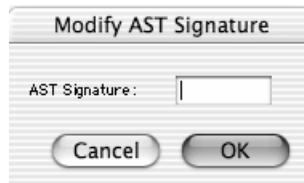
When you modify the AST signature for a project, the project AST signature is assigned to all application files within that project. Application files added to a project are automatically assigned the AST signature of the project. If the project's AST signature has not been assigned, the application file's existing AST signature, if present, is cleared.

To modify a project's AST signature

I Do one of the following:

- From the **Project** menu, choose **Modify Project AST Signature....**

- Right-click the project name in the Project Workspace, and choose **Modify Project AST Signature...** from the context-sensitive that appears.
- Right-click the project name, choose **Properties**, and click **Advanced** tab to display advanced properties. Click **Modify...**
- The **Modify Project AST Signature** dialog box appears:

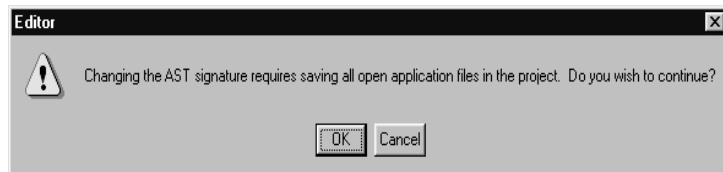


- 2 Enter the project's new AST signature in the **Project AST Signature** field.

Valid AST signatures are three characters long and may contain the characters A to Z (excluding I and O) and the digits 0 to 9.

- 3 Click **OK** to save the signature in the project and in every application file associated with the project.

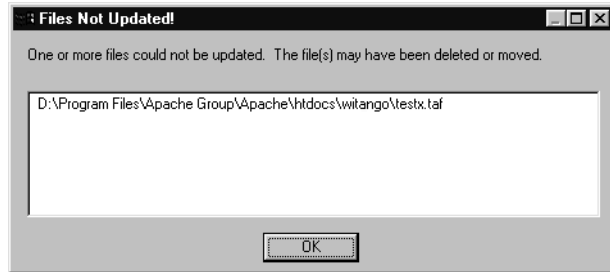
Witango warns you if an application file in the project is open when you try to modify the project's AST signature:



- 4 Click **OK** to update and save all project application files (including open application files).

If one or more application files in a project cannot be opened or saved (for example, files no longer exist, are not checked out from a source

control system, or have “read-only” permission), Witango Studio displays the following dialog box:



If this occurs, correct the problem (for example, check out the necessary files), and repeat the steps, starting from step 1.

Project Root Properties

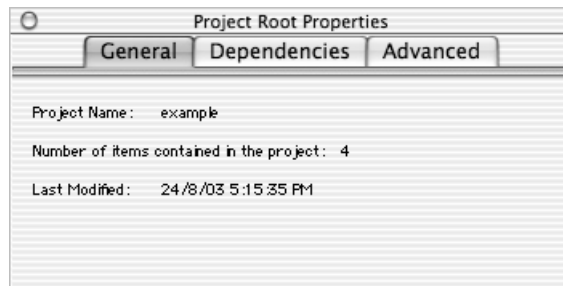
The Project Root Properties dialog box allows you to view information about your project.

To display the Project Root Properties dialog box

- 1 In the Project Workspace, highlight the project name by clicking on the project root.
- 2 Do one of the following:
 - From the View menu, choose **Properties**.
 - Right-click the project root, and choose **Properties** from the context-sensitive menu that appears.

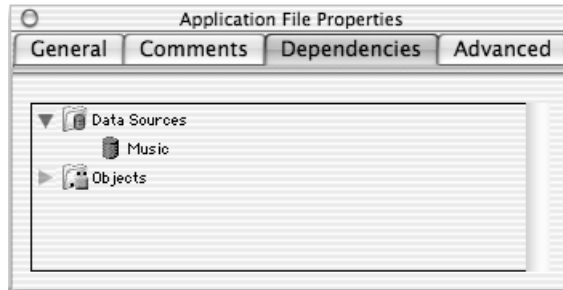
The Project Root Properties dialog box displays three tabs.

- **General.** This section displays the name of the project, the number of items contained in the project, and the last modified date.



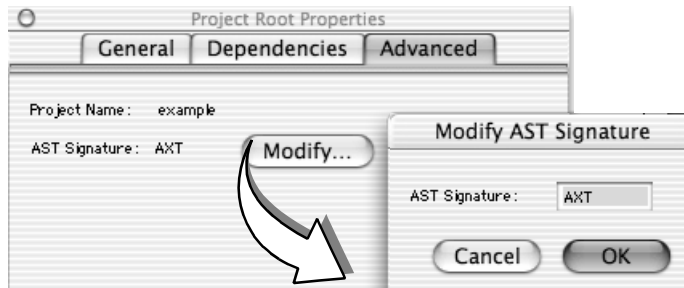
For more information, See “Working With Project Dependencies” on page 74.

- **Dependencies.** This section displays the data sources and objects referenced by the project. Unresolved dependencies are identified by grayed-out icons. This section cannot be modified.



For more information, See “Modifying a Project’s AST Signature” on page 79.

- **Advanced.** This section allows you to enter an AST signature for the project by clicking the Modify button beside AST Signature.



Caution Modifying the project’s AST signature overwrites an application file’s AST signature, which is assigned through the Advanced section of the Application File Properties dialog box. For more information, See “Application File Properties” on page 75.

Using Data Sources

Data Source Basics, Operations, and Properties

A Witango *data source* contains all the information needed to connect to a particular database. You use data sources to tell your Witango applications which databases to connect to. You use Witango Studio to create and manage data sources.

Both Witango Studio and Witango Server need to have access to data sources. Witango Studio uses a data source—via the Data Sources Workspace—to show you the information in the form of tables and their columns. Witango Server requires the same data source, or a data source with the same name on the deployment machine, so it can access the database tables and columns specified within the application file.

The data source *properties* show the information about the data source, including information about its tables and columns.

This chapter covers the following topics:

- understanding Witango data sources
- the Data Sources Workspace
- using the primary column key
- using data sources, including creating, modifying, and deleting them
- setting up deployment data sources
- working with data source properties
- connecting to data sources
- assigning data sources to actions.

About Data Sources

Witango supports the following types of data sources:

- **ODBC** (Open Database Connectivity), in conjunction with third-party drivers, supports connections to a wide variety of database types.
- **JDBC** (Java Database Connectivity), in conjunction with third-party drivers, supports connections to a wide variety of database types.
- **Oracle** supports connections to Oracle databases.

ODBC & JDBC Data Sources

ODBC and JDBC are standards developed to allow applications like Witango to communicate with a wide variety of databases from different vendors. An ODBC/JDBC client application talks to the ODBC/JDBC driver manager that in turn talks to a database driver for a specific type of database.

An ODBC/JDBC driver is a kind of translator. It converts the standard ODBC/JDBC requests made by the application into a format that can be understood by the target database system. ODBC drivers are available for accessing many database management systems (DBMS).

Before creating a data source, you must set up your database server and create or install a database on this server. Depending on the database system, you may also need to install and configure additional software to allow you to connect to the server. Consult your database software and ODBC/JDBC driver documentation for specific instructions.

Oracle Data Sources

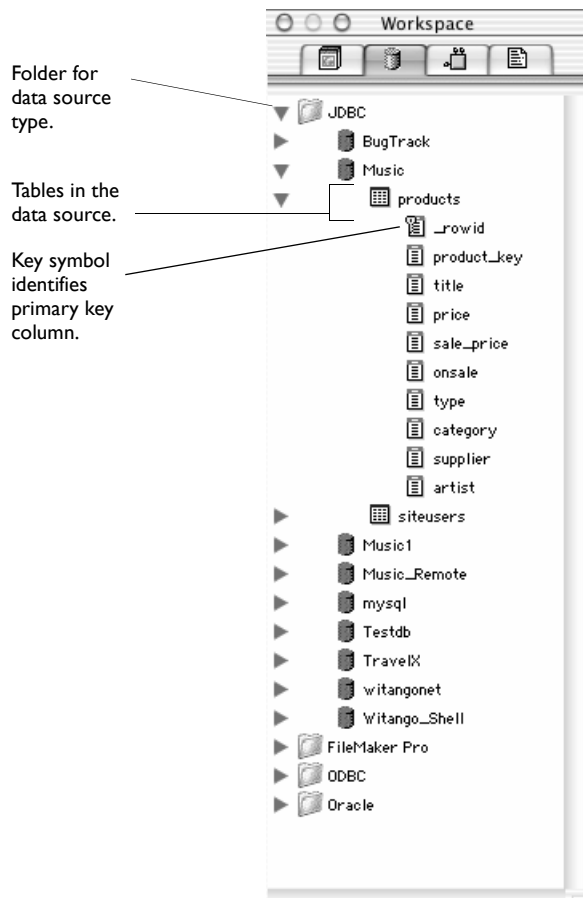
Oracle is a high-performance client/server DBMS. To create and use Oracle data sources, you must have Oracle's SQL*Net installed. Witango supports SQL*Net versions 7.1 and 7.3, and greater. For more information, see the Witango Server Installation Guides for your chosen platform.

The Data Sources Workspace

You perform most data source operations in the Data Sources Workspace (the Data Sources section of the Workspace). To display the Data Sources Workspace, click the Data Sources tab in the Workspace window.

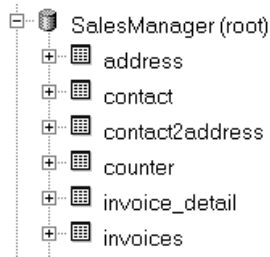
When you open the Data Sources Workspace, you see a folder for each type of data source that Witango supports.

- Expanding a folder shows the defined data sources for that type.
- Expanding a data source shows the tables in that data source. Depending on the settings for the data source, you may need to enter user name and password information before a connection can be made.
- Expanding a table shows the columns in that table.



The bolded data source name in the Workspace indicates the currently active data source—that is, the data source assigned to the front-most open action window. If no database actions are active, no data source names are bolded.

Once a connection is made to a data source, the user name used for the connection appears in parentheses after the data source name. This avoids any confusion when different logins are being used for the same data source.



Using Primary Key Columns



Primary Key

A *primary key* column is a column (or combination of columns) whose value uniquely identifies each record (row) in a table. For example, a customer number might be the primary key in a customers table.

Primary key columns are identified in the Data Sources Workspace by the column/key icon.

Witango builders rely on the primary key column values in various places to identify specific records. When using the builders, it is important to first check that the primary key for each table involved is set correctly. If the specified column or columns do not uniquely identify each record in a table, unexpected results can occur when executing the file. For example, if you mistakenly set the primary key column for a customer table to the “state” column (many customers likely share the same state), using the resulting file to delete a particular customer deletes *A L L* the customers in the same state.

When connecting to a data source, Witango Studio queries the database for information to determine the primary keys. If there is no response, Witango determines the default primary keys by scanning each table for the first column with an appropriate data type (numeric or character).

To change or add a primary key column

Do one of the following:

- In the Data Sources Workspace, right-click the column and choose **Primary Key** from the context-sensitive menu that appears. A check mark in the menu identifies a primary key column.
- Right-click the column in the Data Sources Workspace, choose **Properties** from the context-sensitive menu that appears, and check or uncheck the **Primary Key** check box in the Properties dialog box.
- Select a column in the Data Sources Workspace and choose **Primary Key** from the **Data Source** menu. A check mark in the menu identifies a column as a primary key.

Data Source Operations

Creating a Data ODBC Source

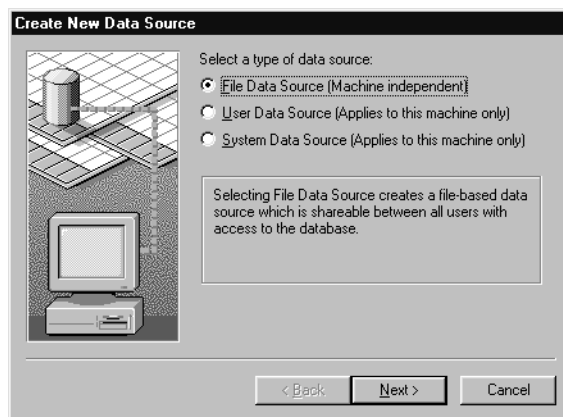
Before you create an ODBC data source, make sure the required ODBC drivers are installed and your database server is running (or, for drivers that access local files, your database files are available).

To create an ODBC data source

1 Do one of the following:

- From the **DataSource** menu, select **New**, and then choose **ODBC**.
- Right-click the Data Sources Workspace, and then select **New** from the context-sensitive menu that appears.

The Create New Data Source dialog box appears:



Note The dialog box appearing may look different. The appearance varies depending on the version of ODBC you have.

2 Select **System Data Source** from the list of data sources.



Note Witango only supports System data sources.

3 Choose **Next**.

See your ODBC driver documentation for detailed configuration instructions.

The Create New Data Source dialog box guides you through the rest of the process. Follow the instructions that appear.

JDBC

To create an JDBC data source

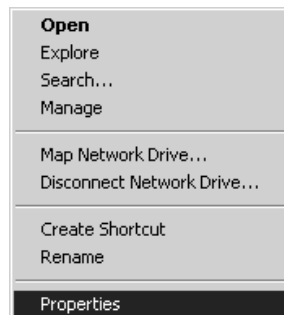


Note Before you will be able to configure a JDBC data source you will need to have the Java 2 Runtime Environment (at least JVM 1.4.1) installed on your machine. This can be downloaded from <http://java.sun.com>.

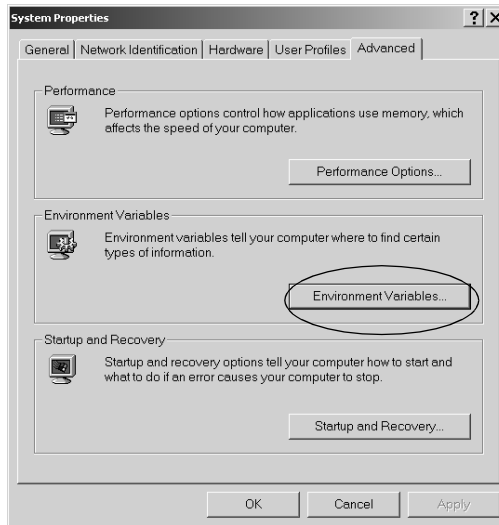


Note Before you will be able to configure a JDBC data source you will need to have the JDBC driver for your chosen database on your machine.

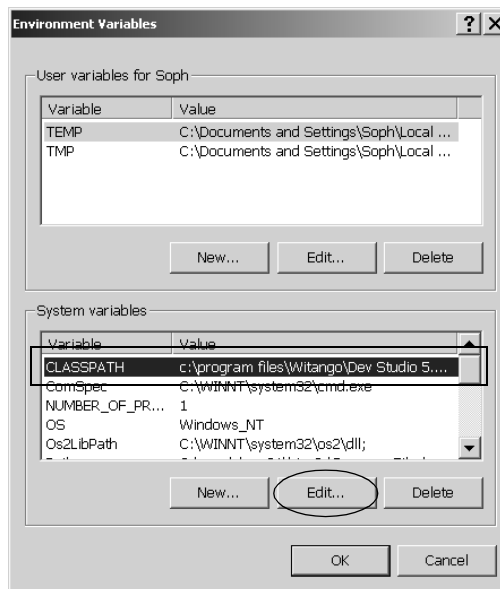
- I Add the JAR file for your database driver to the class path in your environment variables.
 - Right-click on 'My Computer' and select **Properties** from the context sensitive menu that appears.



- The **System Properties** window will appear, move to the **Advanced** tab and click the **Environment Variables** button.



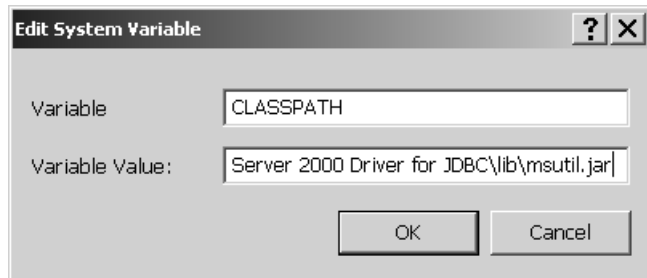
- The **Environment Variables** window will open, and in the **System Variables** pane you should see an entry for **CLASSPATH**. Select the **CLASSPATH** entry and click on the **Edit** button.



- The Edit System Variable window will open, you will need to edit the current variable value to include the path to all of the JDBC drivers (JAR files) you wish to use, each path in the list should be semi-colon separated.

For example:

```
c:\program files\witango\jdbcdrivers\mysql-connector-java-2.0.14-bin.jar
```



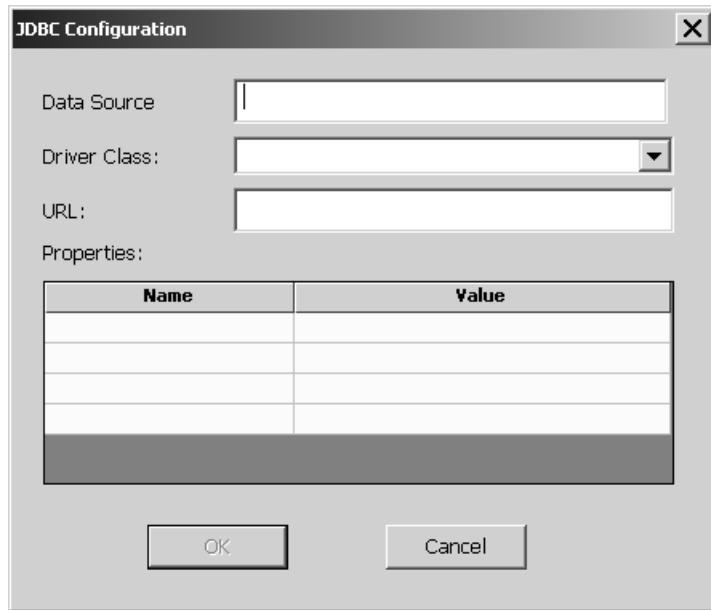
- Click the **OK** button to confirm the changes.

2 Do one of the following:

- From the **DataSource** menu in the Witango Studio, select **New**, and then choose **JDBC**.
- Click the JDBC folder in the Workspace of the Witango Studio, and then right-click in the Workspace. Select **New** from the context-sensitive menu that appears.



- The JDNBC Configuration dialog box appears:



The JDBC Configuration dialog box is shown. It has a title bar with 'JDBC Configuration' and a close button. The dialog contains the following fields:

- Data Source: A text input field.
- Driver Class: A dropdown menu.
- URL: A text input field.
- Properties: A table with two columns, 'Name' and 'Value'.

At the bottom of the dialog are 'OK' and 'Cancel' buttons.

Name	Value

3 Complete the details in the JDBC ConfigurationWindow:

Datasource is the name of the datasource being created.

Driver Class is the reference to the JDBC and exact text string will be included in the documentation of your JDBC Driver documentation. Common ones have been included in a drop down menu for your convenience.



Note If you wish to add your common drivers to this drop down list, this can be done by adding the correct entry to the following file:

```
C:\Program Files\Witango\Dev Studio  
5.5\Configuration\JDBCDriverClass.ini
```


JDBC Configuration

Data Source: music

Driver Class:
 com.microsoft.jdbc.sqlserver.SQLServerDriver
 com.mysql.jdbc.Driver
 oracle.jdbc.OracleDriver
 weblogic.jdbc.mssqlserver4.Driver

URL:

Properties:

Name	Value

OK Cancel

- **URL** is the connection string to the database server. This will also be available in your driver documentation.

JDBC Configuration

Data Source: music

Driver Class: com.mysql.jdbc.Driver

URL: jdbc:mysql://127.0.0.1:3306/music

Properties:

Name	Value

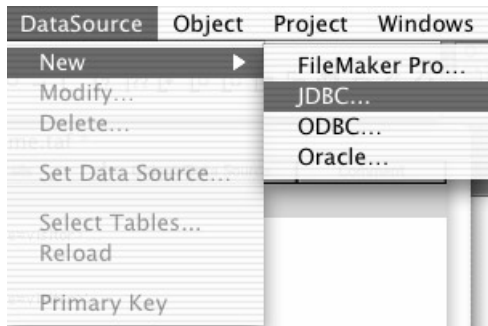
OK Cancel

- Additional Properties required for the Connection can then be added in the Properties table.

- 1 Open Witango Studio.
- 2 Open the **Data Sources** workspace by choosing **Workspace** from the **View** menu or by clicking the **Data Sources** tab in the bottom right corner of the **Workspace** window that appears.



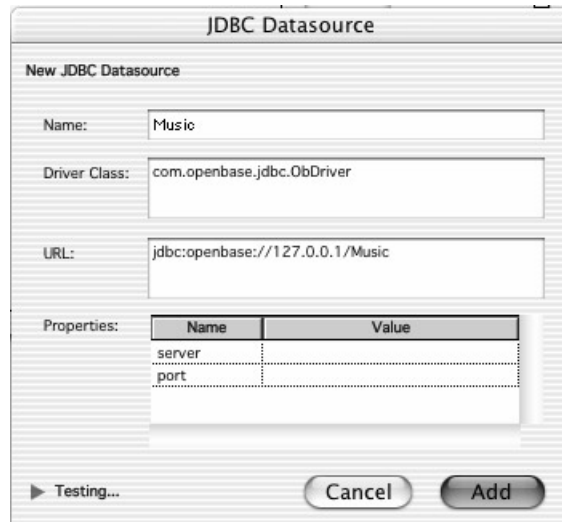
- 3 From the **DataSource** menu, choose **New**, then choose **JDBC...** from the submenu.



The **JDBC Data Source** window appears:

 A screenshot of the 'JDBC Datasource' dialog box. The title bar says 'JDBC Datasource'. Inside, there's a section 'New JDBC Datasource' with three text input fields: 'Name:', 'Driver Class:', and 'URL:'. Below these is a 'Properties:' section containing a table with two columns: 'Name' and 'Value'. The table has two rows: 'server' and 'port'. At the bottom, there are three buttons: 'Testing...' (with a play icon), 'Cancel', and 'Add'.

Name	Value
server	
port	

4 Complete the details in the JDBC Data Source Window:

The image shows a 'JDBC Datasource' configuration window. It contains the following fields and controls:

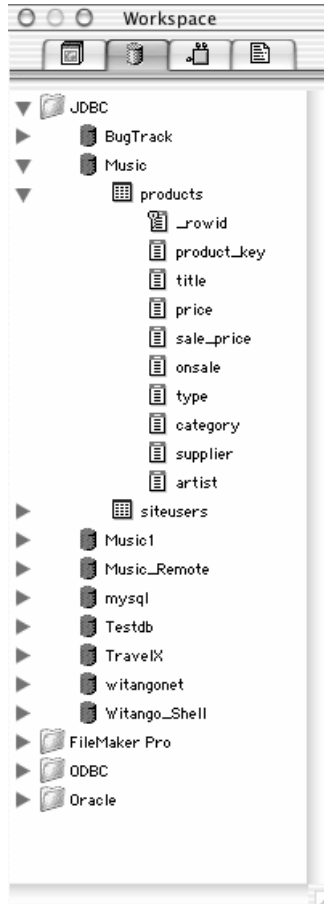
- Name:** Music
- Driver Class:** com.openbase.jdbc.ObDriver
- URL:** jdbc:openbase://127.0.0.1/Music
- Properties:** A table with two columns: Name and Value.

Name	Value
server	
port	

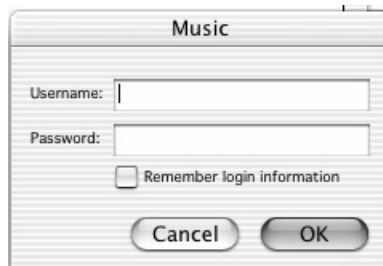
At the bottom of the window, there is a 'Testing...' button with a right-pointing arrow, a 'Cancel' button, and an 'Add' button.

Click the **Add** button:

- 5 Open the **JDBC** folder in the **Workspace** area. You should now see a **Music** folder.



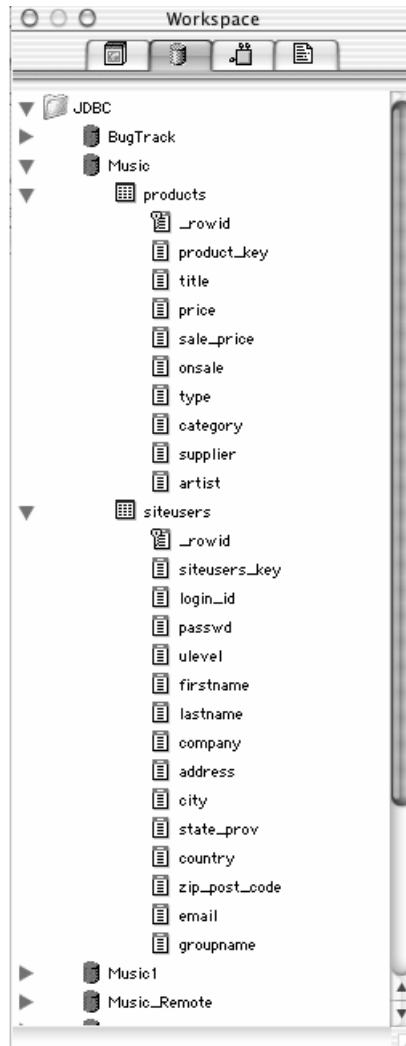
- 6 Open the **Music** folder. You will be prompted to login if you are not already connected to the database..



- 7 Choose **OK**.

See your JDBC driver documentation for detailed configuration instructions.

The database should appear in the JDBC folder in your Workspace.



Oracle

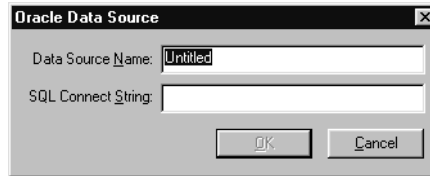
Before creating an Oracle data source, make sure the correct Oracle client software is installed and the database server you want to connect to is available on the network.

To create an Oracle data source

- I Do one of the following:

- From the **Data Source** menu, select **New**, and then choose **Oracle**.
- Right-click the Data Sources Workspace and choose **New** from the context-sensitive menu that appears.

The Oracle Data Source dialog box appears:



- 2 In the **Data Source Name** field, type a name for the data source.
- 3 Type the SQL Connect string in the field provided.

With current versions of SQL*Net, this should be the name of a database alias set up on your computer with the Oracle EasyConfig application.



Note Witango only supports SQL*Net versions 7.1 and 7.3, and greater.

- 4 Click **OK**.

The new data source is added to the Data Sources Workspace.

Modifying a Data Source

To modify a data source

- 1 Select the data source you want to modify.
- 2 Do one of the following:
 - From the **DataSource** menu, choose **Modify....**
 - Right-click the Data Sources Workspace, and then choose **Modify...** from the context-sensitive menu that appears.
 - From the View menu, choose **Properties**, and click **Modify....** in the Data Source Properties dialog box that appears.

- 3 Enter the required information in this dialog box and the subsequent dialog boxes, until you have completed the data source modification process.



Note The data source modification process depends on which driver and which version you have installed on your machine. See your driver documentation for detailed instructions.

If a modified data source is already loaded, the data source is reloaded automatically using the new settings.

For Oracle data sources, modifying a data source does *N O T* affect Witango application files already using that data source, even if they are open when the modification is made. To update a document with the new settings, you must reassign the data source to the document by choosing **Set Data Source** from the **Data Source** menu.

Deleting a Data Source

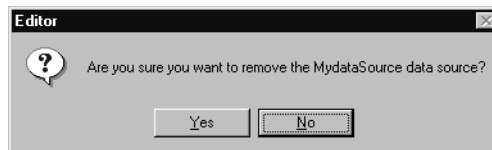
To delete a data source

- 1 Select the data source you want to delete.
- 2 Do one of the following:
 - From the **Data Sources** menu, choose **Delete**.
 - From the **Edit** menu, choose **Delete**.
 - From the main toolbar, click the Delete icon.
 - On the keyboard, press `Delete`.
 - Right-click the Data Sources Workspace, and choose **Delete** from the context-sensitive menu that appears.

A confirmation dialog box appears.



To suppress the confirmation dialog box, hold down `CTRL` when deleting.



- 3 Click **Yes**.

The data source is deleted.

Reloading a Data Source

If the structure of your database changes while Witango Studio is open, you need to reload the data source.

To reload a data source

- 1 Select the data source you want to reload.
- 2 Do one of the following:
 - From the **Data Sources** menu, choose **Reload**.
 - Right-click the Data Sources Workspace, and then choose **Reload** from the context-sensitive menu that appears.

The login information is as specified in the data source's login properties.

Handling Unknown Data Sources

For more information on the default log in settings, see Working With Data Source Properties on page 110.

For Oracle data sources, actions do not rely on anything in Witango Studio's list of data sources to connect to a data source. When connecting, Witango Studio uses only the information stored in the data source. For this reason, Witango Studio could connect to an action's data source while not having a data source defined with matching parameters. Such a data source is called an *unknown* data source.

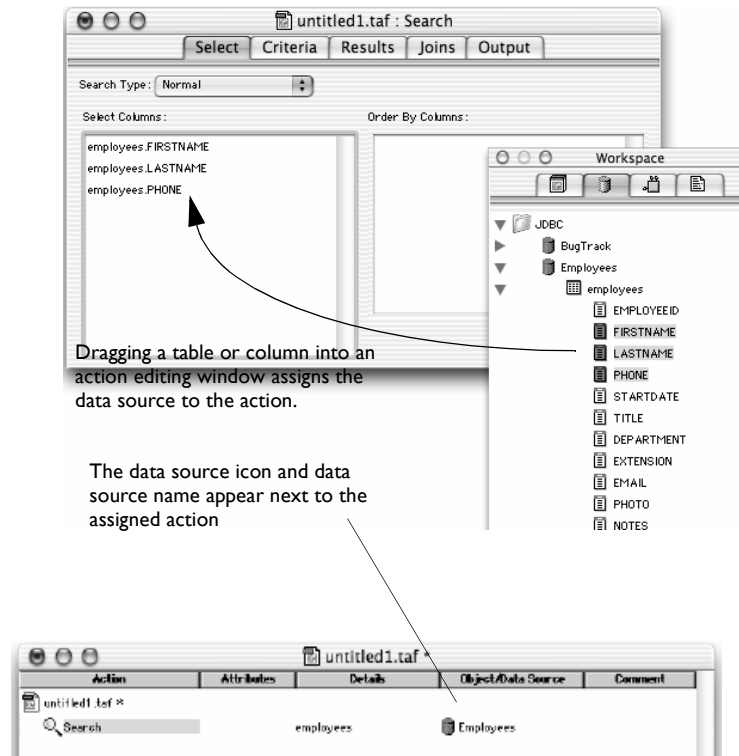
When you open an action having an undefined data source in Witango Studio, yet Witango Studio is able to connect to it, the unknown data source is added automatically to the list in the Data Sources Workspace. The login information for the new data source is set to the default.

This happens even if there is a data source defined with the same parameters, but with a different data source name. That is, all of the pieces of data source information must match in order for an existing one to be used.

Assigning Data Sources to Actions

To assign a data source to an action, do one of the following:

- Drag a table or column from the Data Sources Workspace to a database action editing window or builder window:



From the Data Source menu, choose Set Data Source....

The Data Source Selection dialog box appears. Use this dialog box to set the data source for the action.

- Right-click on an action, and choose **Set Data Source** from the context-sensitive menu that appears.

The Data Source Selection dialog box appears. Use this dialog box to set the data source for the action.

For more information, see "Setting Data Sources for Actions" on page 106.

The data source icon and data source name appear next to the assigned action.



Tip You can also use the **Set Data Source** command to set data sources and data source parameters for one or more actions. For more information, see “Setting Data Sources for Actions” on page 106.

For more information, see “Working With Data Source Properties” on page 110.

If Witango Studio has not yet connected to the data source, a login dialog box may appear. This dialog box only appears if you have the **Ask each time** option checked, which is the default, in the Development section of the Data Source Properties dialog box.



Note Some actions (for example, Transaction actions) do not have columns. If you drag a database action that has no columns, you are prompted to select a data source.

If an action already has a data source assigned to it and you drag a column into it from a different data source, you are asked if you want to cancel the operation or to use the new data source instead.



Note If there are differences in the structures of the databases, changing an action's data source may cause DBMS errors when the action is executed.

If you use a new data source, Witango Studio scans the affected actions and updates the table owner information to match the new data source.

Setting Up Deployment Data Sources

Witango Studio allows you to specify deployment data source parameters that are different from development data source parameters; you can use meta tags in your application files to specify deployment data source parameters. Using deployment data sources, you can:

- execute a Witango application file against multiple data sources
- deploy a Witango application file against a data source (ODBC, OCI, JDBC) other than the one that you developed with.

You can specify deployment data source parameters for each Witango action in an application file on a per-action basis; these can override the default data source settings.



Caution Deployment data sources must point to either the same database as the development data source or one with the same structure and table owner names. Table owner names are stored within the Witango application file and not within the data source, so the development and deployment owner names cannot be different.

The following sections describe how to set the parameters of a data source, and how to set deployment (or development) data sources for actions using the **Set Data Source** command.



Note In order to use meta tags in deployment data sources, the Witango administrator must set the Witango Server's `passThroughSwitch` configuration variable to on.

For more information, see “Disabling the Use of Meta Tags in Data Sources” on page 109.

Setting Deployment Data Source Properties

You can set deployment properties for data sources in the **Deployment** section of the Data Source Properties dialog box. This allows you to specify run-time data source parameters.

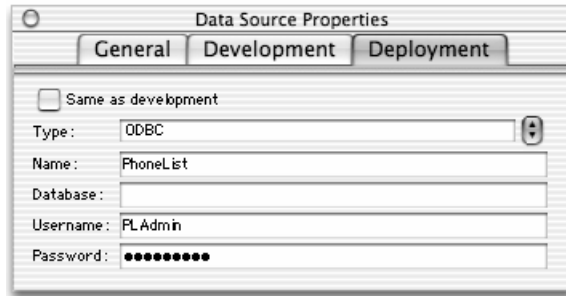
To view the Deployment section of the Data Source Properties dialog box:

- 1 Select the data source in the Data Sources Workspace.
- 2 Do one of the following:
 - From the View menu, choose **Properties**.
 - Type `Alt+Enter`.

- Right-click on the data source and choose **Properties** from the context-sensitive menu that appears.

The Data Source Properties dialog box appears.

3 Click the **Deployment** tab.



If you check **Same as development**, the Type, Name, Database, Username and Password fields are disabled; the default values of these fields are transferred from the development data source to deployment data source.

If you uncheck **Same as development**, specify the deployment data source parameters.

Deployment Data Source Parameters

- **Type.** Specify the type of data source, or enter a meta tag that evaluates to a data source type when the Witango application file is executed (for example, `<@VAR NAME=datasourcetype>`).

The **Type** field must evaluate to one of the type strings shown in the combo box drop-down menu:

- DAM (Macintosh-only)
- FileMaker (Macintosh-only)
- JDBC
- ODBC
- Oracle
- **Name.** This field must evaluate to a valid specifier, dependent upon the data source type:
 - DAM: DAM host name (Macintosh-only)
 - FileMaker: path to the FileMaker Pro application, or the yen symbol “¥” to indicate “Any” (Macintosh-only)
 - JDBC: data source name

- ODBC: data source name
- Oracle: connect string or database alias

The deployment data source name field for Oracle is NOT the name you have given to a data source in Witango Studio.

You can also enter a meta tag that evaluates to a valid specifier when the Witango application file is executed (for example, `<@VAR NAME=datasourcename>`).

- **Database** (Macintosh-only). This field is used only for FileMaker Pro and DAM data source types and must evaluate to a valid database name.

You can also enter a meta tag that evaluates to a valid specifier when the Witango application file is executed (for example, `<@VAR NAME=databasename>`).

- **Username and Password**. These fields may contain meta tags that are substituted when Witango Server executes the application file. Username is not used for FileMaker Pro data sources.

Meta Tags and Deployment Data Sources

For more information, see “Inserting Meta Tags” on page 135.

For more information, see “Disabling the Use of Meta Tags in Data Sources” on page 109.

All fields may contain meta tags, which are substituted when Witango Server executes the application file. When you right-click any text field, a context-sensitive menu appears; it contains standard editing commands and the **Insert Meta Tag...** option.

Choose **Insert Meta Tag...** to open the Insert Meta Tag dialog box. You can insert many of the commonly-used Witango meta tags.

Before connecting to a data source, Witango checks the data source parameters for meta tags. If meta tags are found, and if the `passThroughSwitch` configuration variable is set to `on`, the substitution is performed, and the results are used to establish the connection. If no meta tags are found, the data source parameters are passed as-is.

The following example shows a user name being obtained from the user variable `username`. The user password is taken from the file whose name corresponds to the user name, followed by the `.pwd` extension.

```
Username:<@VAR NAME="username">
```

```
Password: <@INCLUDE FILE="<@VAR username>.pwd">
```

Setting Data Sources for Actions

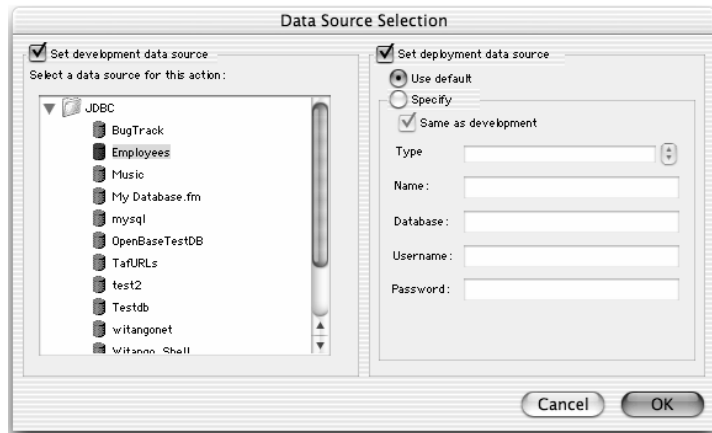
Using the Data Source Selection Dialog Box

You can use the **Set Data Source** command to set development and deployment data source information independently for any selected actions.

To set development and/or deployment data sources for one or more actions

- 1 Select one or more actions by clicking on them. You can select a list of actions by holding down the SHIFT key while selecting, or select discontinuous actions by holding down the CTRL key while selecting.
- 2 Do one of the following:
 - From the **DataSource** menu, choose **Set Data Source**.
 - Right-click and choose **Set Data Source** from the context-sensitive.

The Data Source Selection dialog box appears:



This dialog box opens with the following defaults:

- Both **Set development data source** and **Set deployment data source** are selected, allowing you to set both the development and deployment data sources of the actions.
- The development data source for the first database action in the selection is selected in the list.

- 3 To set a development data source for the selected actions, make sure the **Set development data source** is checked, and select a data source from the list.

To set a deployment data source for the selected actions, make sure **Set deployment data source** is checked, and select one of the following:

- **Use data source default.** This option specifies that the Deployment settings from the selected data source are applied to all the selected database actions.

You specify the deployment settings for a data source in the Deployment section of the properties of the data source. For more information, see *Working With Data Source Properties* on page 110.

- **Specify.** This option specifies that the deployment data source settings in the text fields (**Type**, **Name**, and so on) are applied to all the selected database actions.

If you choose **Specify**, you can check **Same as development**, which causes the development data source to be used for deployment, or you may enter specifications for the deployment data source settings in the fields provided.

For more information, see “Deployment Data Source Parameters” on page 104.

Using the Action Properties Dialog Box

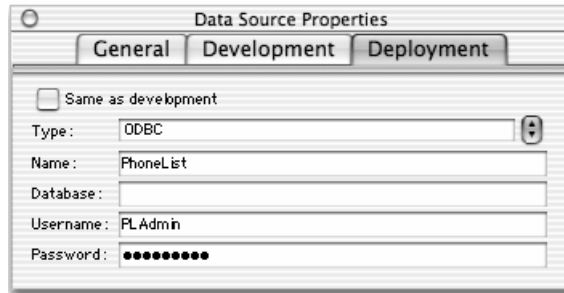
You can also use the **Deployment** tab of the Action Properties dialog box to set data source parameters for actions.

To view the Deployment section of the Action Properties dialog box

- 1 Select the action.
- 2 Do one of the following:
 - From the View menu, choose **Properties**.
 - Type `Alt+Enter`.
 - Right-click on the action and choose **Properties** from the context-sensitive menu that appears.

The Action Properties dialog box appears.

- 3 Click the **Deployment** tab.



For more information, see “Setting Deployment Data Source Properties” on page 103.

- 4 Use the Deployment section of the Action Properties dialog box the same way as the Data Source Properties dialog box.

Disabling the Use of Meta Tags in Data Sources

The `passThroughSwitch` configuration variable allows you to specify whether meta tags are substituted in data source parameters when Witango application files are executed on Witango Server.

Passing through meta tags in deployment data sources is enabled in Witango by default. If you want to disable (or enable) this feature, you can do so by changing the options in the Witango Administrator Application (`config.taf` application file), in the **Feature Switches** screen:

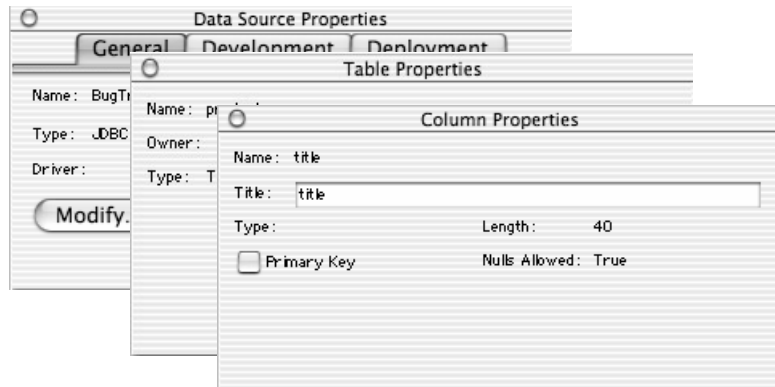
`passThroughSwitch`

Check or uncheck the check box beside the option.

Working With Data Source Properties

The Data Source, Table, and Column Properties dialog boxes allow you to view information about selected data sources, tables, and columns.

In the Data Sources Workspace, right-click on one of these items (a data source, a table under a data source, or a column under a table) and choose **Properties** from the context-sensitive menu that appears.

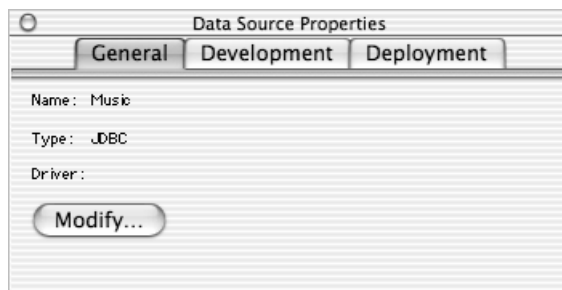


Tip While the **Properties** dialog box is open, you can click on any item in the **Data Sources Workspace** to display its properties. For example, you can switch from a data source to a column, or one table to another.

Data Source Properties

The Data Source Properties dialog box contains three sections: General, Development, and Deployment. Click a tab to display the corresponding section.

- **General.** Clicking the General tab displays basic information about the selected data source. The following example shows the General Properties for an ODBC data source called “Activity Planner”.



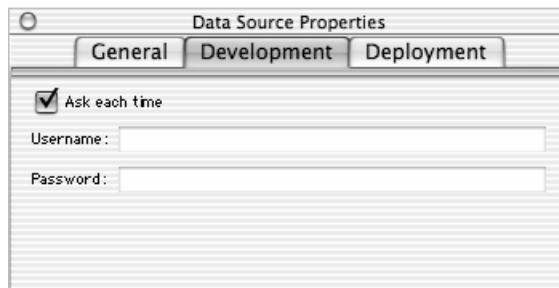
The data source name and type appear for all data sources, but the information in the other fields depends on the type of data source.

Data Source Type	Other Information
JDBC	[None]
ODBC	[None]
Oracle	Connection string

To edit the selected data source, choose **Modify**.

The data source editing dialog box for the data source type appears. When you close the dialog box, new settings appear on the General tab of the Data Source Properties dialog box.

- **Development.** Clicking the Development tab shows the login information required by Witango Studio for connection to the data source:



The Development tab of the Data Source Properties dialog box asks for a user name and password for the data source.

When the **Ask each time** option is checked, Witango asks for connection information whenever the data source is expanded. When you set up a new data source, this option is checked.

- **Deployment.** Clicking the Deployment tab allows you to specify different login information to be used when Witango Server executes the action the data source is assigned to. For more information about this dialog box, see Setting Up Deployment Data Sources on page 103.

Table Properties

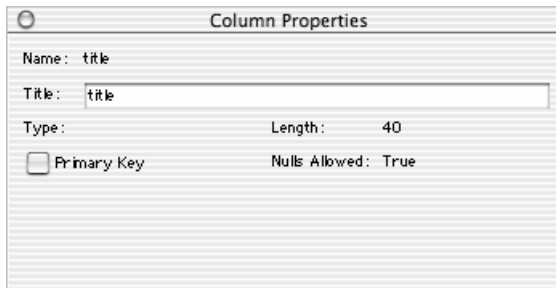
Table Properties shows the name, owner, and type of table.



Column Properties

.For more information, see “Column Options” on page 179.

The Column Properties dialog box displays the name, title, data type, length, whether nulls are allowed or not, and whether the column is a primary key or not.



For more information, see “Using Primary Key Columns” on page 87.

The Column Properties dialog box allows you to edit the **Title** field and select the **Primary Key** option. The title is used by the builders as the default HTML display title for the column.

The Primary Key option is used by the builders to create actions affecting a specific record (record detail display, update and delete).

Connecting to Data Sources

When you expand a data source in the Data Sources Workspace you have not connected to, the login information specified in the Data Source Properties Development window is used for the connection. If you checked the **Ask each time** option, the Log In dialog box appears, allowing you to type your user name and password.

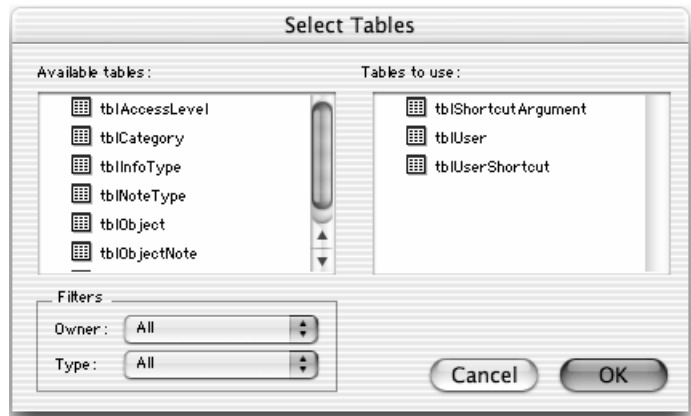
Connecting to Large Data Sources

When Witango Studio connects to a data source containing more than 25 tables, it displays the Select Tables dialog box, allowing you to select which tables you want to work with.

You can also open the Select Tables dialog box by doing one of the following:

- Select a data source; from the **Data Source** menu, choose **Select Tables**.
- Right-click a data source and choose **Select Tables** from the context-sensitive menu that appears.

The following is an example of the Select Tables dialog box:



Selecting Tables

The **Available tables** list in the Select Tables dialog box shows the tables in the data source. Drag the tables you want to work with from this list into the **Tables to use** list.

If you no longer want to use one or more tables, drag them from the **Tables to use** list to the **Available tables** list.

Filtering Tables

You can use the **Owner** and **Type** drop-down menus to filter the tables shown in the **Available tables** list of the Select Tables dialog box.

For example, to show only tables owned by a specific user, select that user from the **Owner** drop-down menu. To show only system tables, select **SYSTEM TABLE** from the **Type** drop-down menu. (The contents of these drop-down menus are determined by the data source; only owners and types existing in the database are listed.)

Editing and Executing Files on Different Computers

When connecting to a data source, Witango relies on configuration information which may not be included in the Witango application file itself. This becomes an issue when Witango Studio and Witango Server reside on different computers, and when editing a file created on a different computer. Witango cannot connect to the data source unless the computer is set up correctly.

The following sections explain which pieces of data source information are stored in the application file, which ones are not, and how to ensure an application file works on a computer other than the one it was created on.

ODBC and JDBC Data Sources

Application files assigned ODBC and JDBC data sources have these pieces of information stored in them:

- ODBC or JDBC data source name
- user name
- password.

For the data source connection to be made on another computer, a data source with the same name pointing to the original database must exist. The user name and password must also be valid for the server pointed to by the data source.

Oracle Data Sources

Application files assigned Oracle data sources have these pieces of information stored in them:

- SQL connect string or database alias name
- user name
- password.

If you specified a SQL connect string (such as `T:199.230.9.8:ORCL`) when defining the data source, your application file works on any computer the string points to, provided that it has access to the Oracle database server. This is because all the connection information is stored right in the string.

Using Snippets

Snippets Basics and Operations

Snippets are named pieces of text, such as Witango meta tags, HTML tags, standard headers and footers, plain text, JavaScript, and SQL. Snippets are a good way of saving text, HTML markup, or other commands that you use frequently. You can insert snippets into most text fields and text windows throughout Witango Studio.

Witango Studio comes with a large defined set of snippets and also lets you create your own snippets.

This chapter discusses how to:

- use snippets
- create new snippets
- edit and organize snippets.
- search snippets.

About Snippets

Snippets allow you to quickly access text, meta tags, and HTML that you use often. You can insert snippets into an editing window by dragging them or double-clicking on a snippet when you have an HTML editing window open.

A special feature of snippets lets you *surround* a selection of text with HTML tags or meta tags, in addition to putting text, HTML tags, or meta tags at a particular place.

Your snippets are saved as XML files within the Witango folder inside your user's Documents and Settings folder.

```
C:\Documents and  
Settings\<your_user>\Witango\MySnippets.xml
```

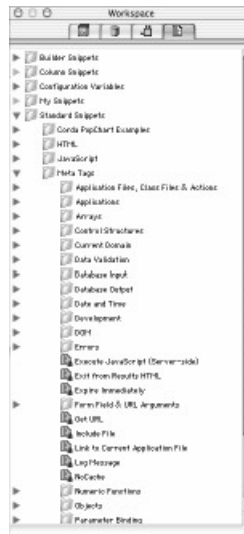
You can edit Witango snippet files with a text editor or HTML editor, or edit them within the Snippets Workspace.

The Snippets Workspace

You manipulate and manage snippets in the Snippets Workspace.

To display the Snippets Workspace

- 1 From the **View** menu, choose **Workspace**.
- 2 Click the **Snippets** tab.



3 To view the contents of a folder, expand it by clicking the “+” sign.

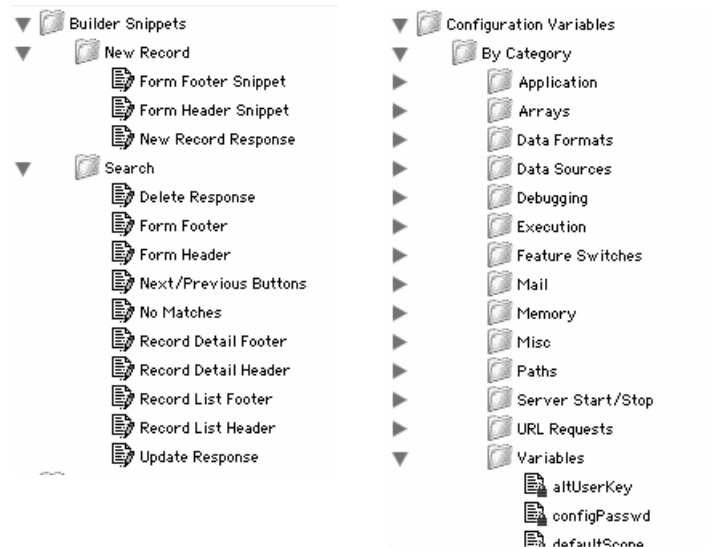
Folders and snippets appear alphabetically.

Snippets are grouped in five folders:

For more information, see
“Column Snippets” on
page 128.

- **Builder Snippets** contains pre-installed snippets used by the Search and New Record builders. These may be edited to change default HTML used in various places in the builders.
- **Column Snippets** contains context-sensitive column snippets, used in the Search action, the Search Builder, and the New Record Builder.
- **Configuration Variables** contains pre-installed snippets of the Witango configuration variables. They are organized by category and by variable scope.
- **My Snippets** contains snippets you create and edit.
- **Standard Snippets** contains pre-installed snippets, which are not editable.

The following are examples of some of the snippet folders and snippets available in the Snippets Workspace:



Working With Snippets

Inserting Snippets

To insert a single snippet into an application file

Do one of the following:

- Drag the snippet to the text field you want (for example, the Results editing HTML window).
- Place your cursor where you want the snippet inserted, then double-click the snippet (HTML editing windows only).
- Right-click on the snippet and choose **Insert** from the context-sensitive menu that appears.
- Right-click on the snippet and choose **Copy** from the context-sensitive menu that appears. Right-click in an editing window, and choose **Paste** from the context-sensitive menu.



Tip The content of a snippet appears as a Windows tool tip if you place your mouse cursor over the snippet.

To insert multiple snippets into an application file

- 1 Select the snippets you want to drag by doing one of the following:
 - Click a snippet, hold down the **SHIFT** key, and click the last item you want to select. All items in order between the first and last snippet you clicked are selected.
 - Click a snippet, hold down the **CTRL** key, and select additional snippets (discontiguous selection).
- 2 Insert the snippets by doing one of the following:
 - Drag the snippets into the text field you want.
 - Place your cursor where you want the snippet inserted, then double-click the snippets to insert them.
 - Right-click on the snippets, and choose **Insert** from the context-sensitive menu that appears.
 - Right-click on the snippets, and choose **Copy** from the context-sensitive menu that appears. Right-click in an editing window and choose **Paste** from the context-sensitive menu that appears.



Note The **Copy** command copies the content of a snippet to the Windows clipboard. The **Insert** command copies the content of the currently selected snippet.

Creating and Editing Snippets

You can create your own snippets in the **My Snippets** folder to automate repeated tasks. You can also edit snippets in the **Builder Snippets** folder, or copy other snippets into the **My Snippets** folder to customize them to your needs.

Editable and Non-Editable Snippets

- Editable snippets can be edited, copied, inserted, deleted and renamed. You can also show their properties. Snippets in the **Builder Snippets** and **My Snippets** folders are editable.
- Non-editable snippets cannot be edited, deleted, nor renamed. You can copy their contents to the clipboard, and show their properties. Snippets in the **Column Snippets**, **Configuration Variables**, and **Standard Snippets** folders are non-editable.

To create a snippet with existing text

- 1 Drag text to a folder in the Snippets Workspace.
- 2 A snippet is created containing the text you dragged. The name of the new snippet defaults to “Untitled” and is editable. Type in a new name.

To create a snippet with new text

- 1 Do one of the following:
 - From the **Edit** menu, choose **Snippet**, then choose **New**.
 - Right-click in the **My Snippets** folder of the Snippets Workspace, and select **New Snippet** from the context-sensitive menu that appears.

The New Snippet window appears:



- 2 Fill in the window as follows.

Name. Assign a name to the new snippet.

Create in this folder. The **My Snippets** folder is shown by default. Select where you want to create the new snippet.

- 3 Click **OK**.

The Snippet contents window appears:



- 4 Type in the content of your snippet or insert text from elsewhere.

Using Placeholders in Snippets

When you right-click the Snippet contents window, the **Insert Placeholder** command is listed in the context-sensitive menu that appears, in addition to the standard Witango editing commands.



Insert Placeholder inserts a special symbol—the yen symbol, ¥—into the Snippet contents window. This placeholder character allows you to create snippets that put HTML tags or meta tags around text you select before inserting the snippet. For example, you can create a snippet of the following text:

```
<H1>¥</H1>
```

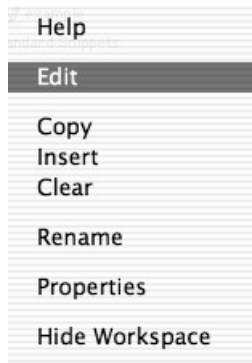
To use this snippet while editing HTML, you select the text you want to apply the `<H1>` format to, and then drag the snippet into the editing window or double-click the snippet. The selected text is surrounded by the `<H1>` and `</H1>` tags.

Editing Snippets

To edit the contents of a snippet

- I Do one of the following:

- Right-click the snippet you want to edit, and choose **Edit** from the context-sensitive menu that appears:



- Hold down the Ctrl key and double-click the snippet.

The Snippet content window appears:



2 Edit the text.

You can edit multiple snippets. Choosing **Edit** from the context-sensitive menu opens the contents window for all the selected snippets.

Managing Snippets and Snippets Folders

You can create new folders to organize your snippets in the **My Snippets** folder. This feature is useful if you want to categorize your snippets by type (for example, meta tags, SQL, HTML, and text snippets). **Snippet Properties** allows you to see the location and size of snippets and snippet folders.

Snippets Folder

To create a new folder in the My Snippets folder

- I Do one of the following:

- Select the My Snippets folder. From the **Edit** menu, select **Snippet**, and then choose **New Folder**.
- Right-click the My Snippets folder, and choose **New Folder** from the context-sensitive menu that appears.

A folder called **Untitled** appears. The name is already selected so you can change it.

2 Type in a different name.

To rename a snippet or snippet folder

- 1 Select a snippet or snippet folder, and do one of the following:
 - From the **Edit** menu, choose **Rename**.
 - Click the snippet or snippet folder again.
 - Right-click the snippet or snippet folder, and choose **Rename** from the context-sensitive menu that appears.
- 2 The name of the snippet or snippet folder becomes editable; type in the new name



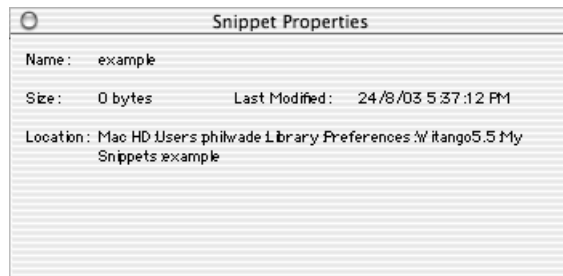
Note Snippet names must be unique inside a folder.

Snippet Properties

To see the properties of a snippet or snippet folder

- 1 Select the snippet or snippet folder.
- 2 Do one of the following:
 - From the View menu, choose **Properties**.
 - Type **Alt+Enter**.
 - Right-click the snippet or snippet folder, and choose **Properties** from the context-sensitive menu that appears.

The Snippet Properties or Snippet Folder Properties dialog box appears:



The Snippets Folder Properties and Snippets Properties dialog boxes displays the name and location of the snippet folder or snippet; the Snippets Properties dialog box also displays the size and last modified date of the snippet.

Once you have the Properties dialog box open, clicking on different snippets or snippet folders displays the properties of that snippet or snippet folder.

Copying, Moving, and Deleting Snippets

Snippets in the **Standard Snippets**, **Configuration Variables** and **Column Snippets** folders are non-editable and cannot be edited within, nor deleted from those folders. However, they can be copied into the **My Snippets** folder to be edited and organized.

To copy a non-editable snippet to the My Snippets folder for editing

- Drag the snippet from the **Standard Snippets**, **Configuration Variables**, or **Column Snippets** folders to the **My Snippets** folder.

To move a snippet within the My Snippets folder

- You can move snippets to a different folder within the **My Snippets** folder by dragging snippets to their new location.

To duplicate a snippet within the My Snippets folder

- Hold down the CTRL key, and drag the snippet.

To delete a snippet or snippet folder

- 1 Select the snippet or snippet folder.

- 2 Do one of the following:

- From the **Edit** menu, choose Delete.
- On the main toolbar, click the Delete icon.
- Press DELETE.
- Right-click the snippet or snippet folder, and choose **Delete** from the context-sensitive menu that appears.

A dialog box appears, asking you to confirm that you want to delete the snippet.



3 Click **OK**.



Note You can delete items from only the **My Snippets** and **Builder Snippets** folders.

If you want to delete multiple snippets, select the snippets and choose **Delete** from the context-sensitive menu.

Column Snippets

The **Column Snippets** area of the Snippets Workspace becomes active when the following are displayed:

- Results HTML of the Search action.

The Search action's **Select columns** appear and the **Columns Snippets** folder expanded, if necessary. (The Search action appears in the actions generated by the New Record Builder and Search Builder, as well as a separate action.)

- New Record Response HTML in the New Record Builder.

Shows all the columns from the table into which the new record is being added.

The content of each snippet is `<@COLUMN name>`, where the column name is the name from the Search action or New Record Builder you are editing.

Setting Preferences

Changing Your Witango Studio Preferences

The default preferences for Witango Studio are automatically set during installation. If you want to change the various settings required by the application, you can do so using the Preferences dialog box.

This chapter describes each of the following preference settings:

- Studio, data source, and online help dialog box options
- HTML and text options
- source control options
- compile options
- objects options.

Using the Preferences Dialog Box

The Preferences dialog box is where you can view and change the many options available in Witango Studio.

To use the Preferences dialog box

- 1 From the **Witango** menu, choose **Preferences**.



The Preferences dialog box appears. See page 131.

- 2 Set Witango preferences using the five tabs in this dialog box: the first for general preferences, the second for options affecting text editing windows, the third for source control the fourth for compile, and the fifth for objects. Switch among preference sections by clicking the appropriate tab to display the options available.

For more information, see “Selecting Options” on page 131

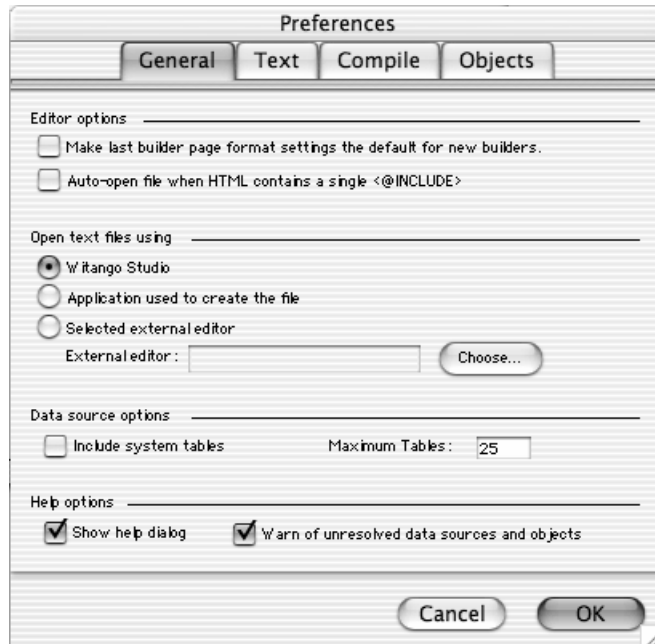
- 3 After setting your preferences, click **OK** to save your changes and close the Preferences dialog box.

Any open editing windows are automatically updated with any new settings.

Selecting Options

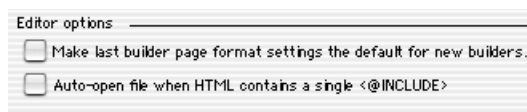
General

To display the General section of the Preferences dialog box, if not already displayed, click the **General** tab.



- **Studio options**

Opening a text or HTML file in the Project Workspace by default opens the file in Witango's built-in editing window. If you want to use the application defined for that file type in the Windows Explorer instead, deselect the **Open text files in projects using Witango Studio** option.



Check the Save last-used page format settings in the Builders and use them as defaults for new Builders option to save and reuse page formatting settings for the Search Builder and New Record Builder. Page formatting settings are saved independently for each type of builder.

- **Data source options**

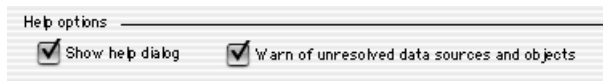
Select the **Include system tables** option to include a data source's system tables in the Data Sources Workspace. This option is disabled by default. System tables contain meta-data; that is, information about the database itself, users, and so on.



Set the maximum number of tables you want to appear. The default is 25. If a data source has more than the specified number of tables, the Select Tables dialog box appears, allowing you to work with a more manageable subset of tables.

- **Help options**

Select the **Show help dialog** option to show the Help Information dialog box, which tells you about associating a Web browser with the HTML help system when you choose an item from the **Help** menu.



Selecting this option is the only way to show the help dialog again if you have previously selected **Don't show this dialog again** in the Help Information dialog box.

Check the **Show TipWitango of the Day** option to show the Tip of the Day dialog box upon entering Witango Studio. The Tip of the Day displays useful information about using Witango Studio more effectively. It can be disabled by unchecking this option here or in the Tip of the Day dialog box.

- **Dependencies**

Check the Warn if unresolved option to show a warning about missing data sources or objects referenced by Witango application files and Witango class files.

- **Site Options**

When you deploy (upload) or download a project file to a remote site via FTP, the File Transfer Status dialog box appears. Check the **Close file transfer progress dialog when operation completes** option to automatically close the File Transfer Status

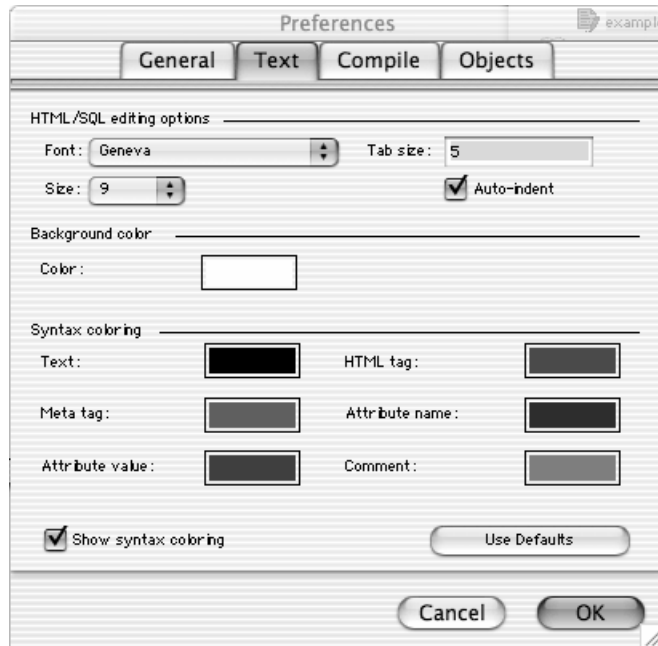
For more information, see "Working With Project Dependencies" on page 74

For more information about FTP, For more information, see "Working With Project FTP Sites" on page 79

dialog box when the transfer is complete. Otherwise, you can close it by clicking **Close** on the File Transfer Status dialog box. This option is unchecked by default.

Text

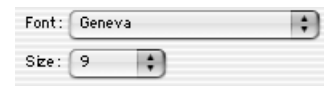
When you click the **Text** tab in the Preferences dialog box, the following text options appear:



Text

Select the basic text attributes for the text that appears in the editing windows:

- **Font.** Select from the monospaced fonts installed on your machine, such as Terminal, Fixdys, Courier, Courier New, and Lucida Console.
- **Size.** Select from the point sizes available for the selected font, such as 9, 10, 11, 12, 14, 16.
- **Tab Size.** Type the number of characters you want to equal one tab character. The default is “3”.



- **Auto indent.** This option, enabled by default, inserts a tab character automatically at the start of a new line at the same indent level as the previous line.
- **Background**

Color refers to the background color of the HTML editing window.



You can select from the colors in the drop-down menu: Black, Maroon, Green, Olive, Navy, Purple, Teal, Gray, Silver, Red, Lime, Yellow, Blue, Fuchsia, Aqua, and White. The default is “White”.

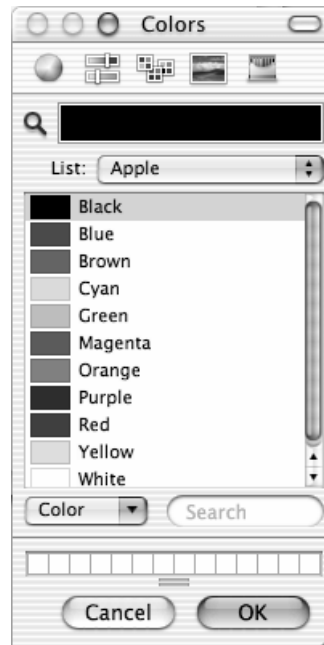
If you select a different color, the background of the **Category** menu changes to show you the selected font, size, and color against that background.

- **Syntax Coloring**

In addition to setting a default font and size for text appearing in the HTML editing window, you can also add color to the selected font for certain categories of text.

Syntax coloring

Text:		HTML tag:	
Meta tag:		Attribute name:	
Attribute value:		Comment:	



Coloring your text can make editing of your text, HTML, and meta tags much faster and easier, and reduce the chances of making syntax errors. Only *valid* HTML and meta tags appear in the specified color.



Note Meta tag attribute names are not currently checked for validity.

The default is to show the editing window text with syntax coloring enabled. If you deselect the **Show syntax coloring** option, all text in an editing window appears black on a white background.

The following table describes each category and the default color for the text in the category:

Category	Text Affected by the Setting	Default Color
Text	Text that is neither a meta tag nor HTML.	Black
HTML Tag	HTML tag names, for example, <code><BODY></code> and <code></BODY></code>	Blue
Meta Tag	Meta tag names without any attributes, for example, <code><@POSTARG></code>	Green
Attribute Name	Meta tag attribute name, for example, <code>NAME=</code> in <code><@POSTARG NAME="Fred"></code>	Purple
Attribute Values	Meta tag attribute value, for example, <code>"Fred"</code> in <code><@POSTARG NAME="Fred"></code>	Red
Comment	Any text enclosed within the <code><@COMMENT></code> <code><@/COMMENT></code> meta tag pair, including the <code><@COMMENT></code> <code><@/COMMENT></code> meta tags. This category also includes the <code><@!></code> meta tag and HTML comments.	Gray

To assign a different color to a category, select the category in the list, then select a color from the **Color** drop-down menu. The colors available are the same colors listed for Background on page 134.

- **Use Defaults**

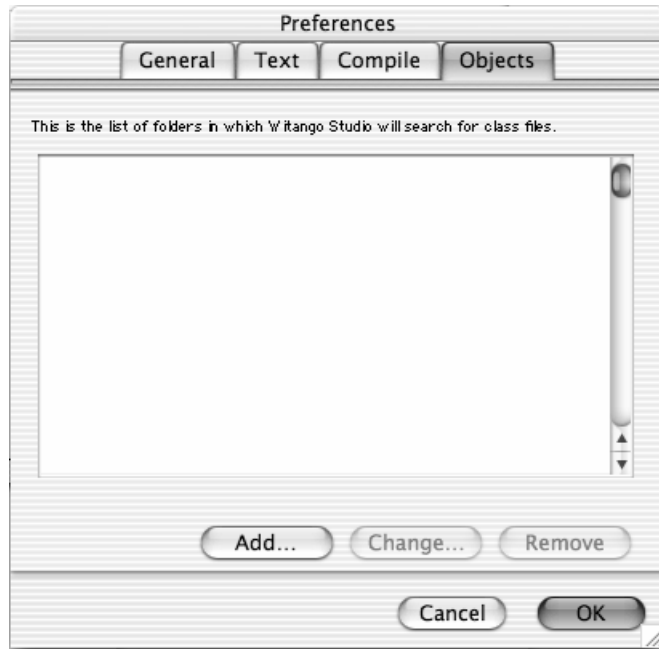
If you change text preferences and want to return to the defaults, click **Use Defaults**.

- **Add new projects to.** To add newly created Witango projects to source control automatically, select the name of your source control system in the drop-down menu. The default is **Don't Add New Projects To Source Control**.
- **Prompt to add files when inserted into project.** If you check this option, when you add new files to a Witango project, Witango Studio asks if you want to add the new files to source control. Click **Yes** to add the files or **No** to cancel.
- **Use dialog for checkout.** Check this option if you want the Check Out File(s) dialog box to always appear when you check out files. Otherwise, the selected files are automatically checked out without displaying the dialog box. A checkmark in the check box (☒) beside the file name in the Project Workspace indicates the file is checked out.

For more information, see Using Source Control in Witango on page 97.

Objects

When you click the **Objects** tab in the Preferences dialog box, the following objects options appear:



The **Objects** section allows you to view and edit search paths for Witango class files. Because of the way Witango locates COM objects and JavaBeans, it is not necessary to set up search paths to search for them. This section applies only to Witango class files.

Directories

For more information, see
“Setting Search Paths for
Witango Class Files” on
page 416

This area displays the list of search paths that Witango uses to find Witango class files whenever a Witango application file refers to them. If you added Witango class files to the Object Workspace, the paths to these Witango class files are automatically placed in this list. You can add or delete paths in this list, using the following buttons:

- **Add.** If the path you want Witango to search is not in the list, click the Add button. A Browse for Folder dialog box appears.
- **Delete.** If there is a path that you no longer need, you can delete it by selecting it and clicking the Delete button.

When Witango searches for a Witango class file, it starts from the first path specified in this list and continues from there. If two Witango class files in different folders have the same name, Witango uses the Witango class file in the first listed path.



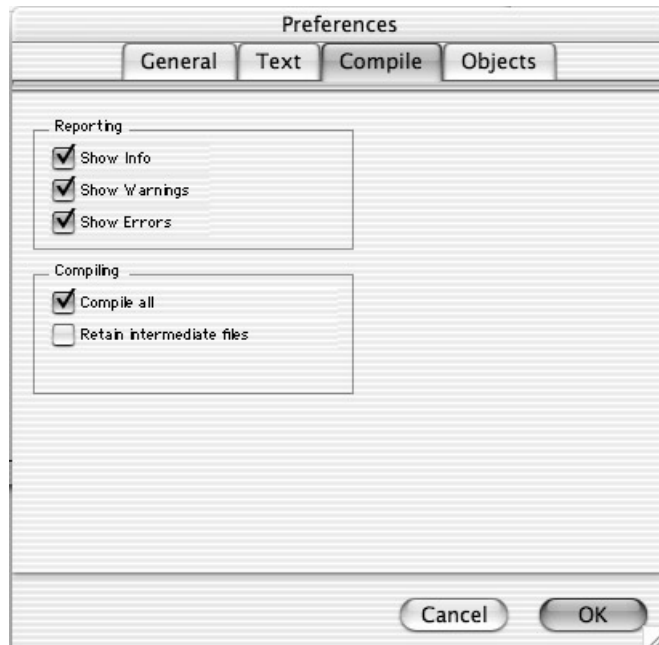
You can change the order in which the directories are searched by changing the relative positions of the paths on this list, using the following buttons:



- **Up and Down.** Move the paths on the list up or down by selecting them and then using the Up or Down buttons, so that they appear in the order you want.

Compile

When you click the **Compile** tab in the Preferences dialog box, the following objects options appear:



The Compile section allows you to preset your defaults when running, syntax checking reports and compiling your applications to J2EE. These settings are only relevant to users of the Witango Studio Professional edition.

The Report Options



- **Show Information**

Check this checkbox if you want the Syntax Report to show information as to the status of the syntax check, ie, which file it is currently working on.



- **Show warnings**

Check this checkbox if you want the Syntax Report to show warnings, that is issues which may cause errors when a compiled application is deployed. These warnings should be reviewed by the user.



- **Show Errors**

Check this checkbox if you want the Syntax Report to show errors. These errors will prevent the application being successfully compiled for J2EE and should therefore be fixed.



Note These checkboxes will not prevent you accessing the information once in the syntax report, they only affect what information is immediately visible to the user.

The Compile Options

The compile information settings allow the user to customise the compile function such that.

- **Compile All**

Check this checkbox if you want the entire directory of source files to be compiled. If the checkbox is not checked, then only those files which have been modified since the last compile was run will be compiled.

- **Retain Intermediate Files**

Once a successful syntax check is run on the source directory, there are two steps which the compile facility undertakes to generate the servlets. The first step is to take the .taf and .tcf files to .java files. The second step is to take the .java files to .class files. If the user wishes to retain the .java files, this checkbox should be checked. The more usual approach would be to run the compile without this option checked.

SECTION II

Witango Building Blocks

How to Use Meta Tags and Variables

This section discusses two of the most important features of Witango application files: Witango *meta tags* and Witango *variables*.

This section contains chapters on the following topics:

- Chapter 7, Working with Meta Tags on page 143
- Chapter 8, Working With Variables on page 155

Chapter 7 and a look through Chapter 8 are recommended for new users of Witango. .

Working with Meta Tags

Understanding how to work with Meta tags

Meta tags are special tags that are entered in Witango Studio and are interpreted by Witango Server when your application files are executed. They can do many different things in an application file, from controlling the flow of information to performing an action on a data source to setting or retrieving variable values.

Meta tags look like HTML tags, with a starting and ending angle bracket (“<” and “>”), but the character after the starting angle bracket is “@”, or in the case of a closing meta tag, “/@”. When you are creating HTML in Witango Studio, you insert meta tags just like HTML tags. The user’s Web browser never sees the meta tags, as they are interpreted by Witango Server before being sent to the Web browser.

You can find a complete and detailed list of Witango meta tags and their attributes in the *Witango Programmers Guide*.

This chapter covers the following topics:

- introduction to meta tags
- how to insert meta tags in Witango application files, Witango class files, and HTML documents created with Witango.

About Meta Tags

Meta tags are special commands to Witango that can do many things, including control execution of Witango application files, return values from a database, create variables, and return the values of variables. One of the places these tags have their effect is in the HTML returned by Witango Server to your Web browser; for example, Witango may return HTML to the browser using meta tags that refer to form field values (`<@POSTARG>`) and values returned from a database (`<@COLUMN>`).

Meta tags are interpreted by Witango Server at the application file execution time and the resulting values, if any, are substituted where the meta tags appear.

Most meta tags return values when interpreted by Witango Server. A few Witango meta tags that control the flow of information or assign values to variables do not return values. These include `<@ROWS>`, `<@IF>`, `<@IFEQUAL>`, `<@IFEMPTY>`, and `<@ASSIGN>` meta tags.

Meta tags begin with the “at” symbol, “@”, to distinguish them from HTML tags. Closing meta tags begin with “/”. This documentation shows meta tags in uppercase, but meta tags are case insensitive; that is, `<@if>`, `<@IF>`, and `<@iF>` are all treated the same.

Meta tags often have attributes, much like HTML tags. These name/value attribute pairs specify required and optional attributes of the meta tag. For example:

```
<@ASSIGN NAME="last_name" VALUE="Flintstone">
```

This example assigns the value “Flintstone” to the variable `last_name`.

You can leave the name of an attribute off if the attribute is required and in its standard position; however, it is recommended that you use attribute names to avoid ambiguity.

Where You Can Use Meta Tags

Most meta tags can be used in all places in Witango application files and Witango class files where text or HTML can be inserted, including these locations:

- attribute HTML that is attached to an action, including:
 - Results HTML
 - Error HTML
 - No Results HTML
- actions in a Witango application file or Witango class file, including:
 - parameters in Search, Update, and Delete actions
 - column values in Update and Insert actions
 - Maximum Matches and Start Match fields in Search actions
 - External action parameters
 - File action parameters
 - Assign actions (both name and value)
 - If action parameters
- custom and column references used in database actions
- SQL entered into the Direct DBMS action window
- HTML files included using the `<@INCLUDE>` meta tag
- most attributes for other meta tags.

Where you can insert meta tags, the context-sensitive menu shows **Insert Meta Tag**.

Combining Meta Tags

When you use meta tags in action fields or in attributes of other meta tags, you can use multiple meta tags and mix literal values with meta tags. For example, in a column value field parameter for an Insert action, you could specify:

```
<@POSTARG NAME=prefix><@POSTARG NAME=suffix>
```

This indicates the concatenation of the `prefix` and `suffix` form fields.

To give a long distance code in a standard format that includes spaces and meta tags, the parameter would look something like the following:

```
+1 <@POSTARG NAME=area_code> <@POSTARG  
NAME=phone_num>
```

Quoting Attribute Values

Only attributes that have spaces in them need to be quoted, but it is never wrong to quote attributes. Either single or double quotes can be used.

For more information on the rules for quoting attributes in meta tags, see the Witango Programmers Guide.

```
<@CALC EXPR=3+4 PRECISION="2">
```

```
<@CALC EXPR="3+4" PRECISION="2">
```

Both examples are correct, as is the single quote

```
<@POSTARG NAME='homer'>.
```



Tip For new users of Witango, the best method to adopt is quoting all attribute values.

Inserting Meta Tags

For more information, see “Working With Snippets” on page 120.

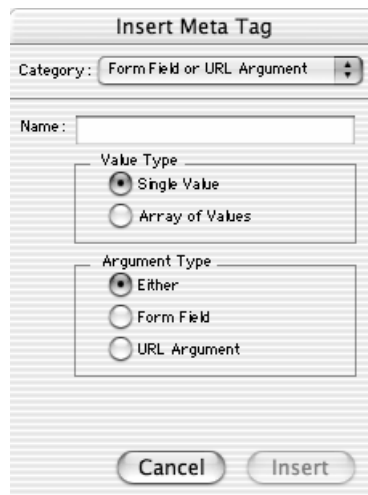
Meta tags can always be entered by typing them or dragging a meta tag snippet into your Witango application file or Witango class file. There is also a shortcut to inserting many common meta tags: the Insert Meta Tag dialog box. This dialog box does not contain all of the meta tags; some must be typed in or dragged in from the Snippets Workspace.

The **Insert Meta Tag** command, available from the **Edit** menu or the context-sensitive menu, inserts a meta tag into a Witango application file or Witango class file. This dialog box shows common meta tags in a category drop-down menu. This provides a quick reference for many common meta tags.

To insert common meta tags into your Witango application file or Witango class file

- 1 From the **Edit** menu, choose **Insert Meta Tag**.

The Insert Meta Tag dialog box appears:



- 2 From the **Category** drop-down menu, select one of the five options:
 - Form Field or URL Argument
 - Variable
 - Current Date or Time
 - Request Parameter

- Action Result Item

Selecting a category changes the dialog box to show the appropriate fields for that category of meta tag.

- 3 Select or enter the attributes necessary for the meta tag you are inserting.

- 4 Click **Insert**.

The **Insert** button is enabled only when you have entered sufficient information to construct the meta tag.

The meta tag and its attributes are placed at the insertion point in your Witango application file or Witango class file when **Insert** is clicked. The various types of meta tags you can insert are described in the following sections.

Form Field or URL Argument

To insert a meta tag that returns the value of a form field or a URL argument, choose **Form Field or URL Argument** from the **Category** drop-down menu. These meta tags are `<@SEARCHARG>`, `<@POSTARG>`, and `<@ARG>`. A name must be specified. (The name of an argument is assigned in the HTML form that is set up by the creator of a Web page.)

The screenshot shows a dialog box titled "Insert Meta Tag". At the top, there is a "Category:" label followed by a dropdown menu currently showing "Form Field or URL Argument". Below this is a "Name:" label followed by an empty text input field. Underneath the name field, there are two sections of radio buttons. The first section, labeled "Value Type", has two options: "Single Value" (which is selected with a filled radio button) and "Array of Values" (which is unselected). The second section, labeled "Argument Type", also has three options: "Either" (selected), "Form Field" (unselected), and "URL Argument" (unselected). At the bottom of the dialog box, there are two buttons: "Cancel" and "Insert".

Once you have specified the name, the radio buttons have the following effects:

- Single value has no effect on the inserted meta tag.

- **Array of values** adds the `TYPE=ARRAY` parameter to the meta tag.
Use this option to get all values for form fields which may contain multiple values (such as lists).
- **Either** inserts `<@ARG>`.
- **Form field** inserts `<@POSTARG>`.
- **URL argument** inserts `<@SEARCHARG>`.

Variables

To insert a meta tag that returns the value of a variable (the `<@VAR>` tag) with the Insert Meta Tag dialog box, choose **Variable** from the **Category** drop-down menu.

The screenshot shows a dialog box titled "Insert Meta Tag". It has several input fields and buttons. The "Category" dropdown menu is set to "Variable". Below it, the "Name" field is empty. The "Scope" dropdown menu is set to "Default". There is an unchecked checkbox labeled "Array Element". Below that, the "Row" field is set to "0" and the "Column" field is also set to "0". At the bottom of the dialog are two buttons: "Cancel" and "Insert".

The following attributes can be assigned for insertion of `<@VAR>` with the Insert Meta Tag dialog box:

- **Name** contains an alphabetized list of variables *assigned to* (via Assign actions) in the current Witango application file or Witango class file. Select one of the variable names or type one in. This attribute is required and inserts the `NAME` attribute.
- The **Scope** drop-down menu contains:
 - Default
 - Request
 - User
 - Cookie
 - Application

- Domain
- System.

If the scope is other than default, the `SCOPE=selectedScope` attribute is added to the meta tag.

- The **Row** and **Column** fields are enabled only when the **Array Element** check box is checked. This option adds `[rownumber, columnnumber]` to the variable name. If you leave either of the **Row** or **Column** fields empty, the value defaults to “*”, which means all rows or columns are returned.

Current Date or Time

To insert the current date or time using a meta tag, choose **Current Date or Time** from the **Category** drop-down menu.

The screenshot shows a dialog box titled "Insert Meta Tag". It has a "Category:" dropdown menu with "Current Date/Time" selected. Below this are three radio buttons: "Current date" (which is selected), "Current time", and "Current timestamp". There is also a "Format:" dropdown menu with "Default" selected. At the bottom of the dialog are two buttons: "Cancel" and "Insert".

For more information, see “<@CURRENTDATE>”, “<@CURRENTTIME>”, “<@CURRENTTIMESTAMP>” in the Witango Programmers Guide.

This action inserts `<@CURRENTDATE>`, `<@CURRENTTIME>`, or `<@CURRENTTIMESTAMP>`. There are various options you can set for **Current Date or Time**.

If you select a format other than **Default**, the `FORMAT` attribute is added to the tag.

The **Format** list contains several common formats of the type denoted by the **Current date**, **Current time**, or **Current timestamp** radio button. When the radio button selection changes, the **Format** selection reverts to **Default**.

Request Parameter

To return a value pertaining to the current user request, choose **Request Parameter** from the **Category** drop-down menu.

The dialog box titled "Insert Meta Tag" shows the "Category" dropdown set to "Request Parameter". Below it, the "Parameter:" label is followed by a list box containing the following items: Client Name, Client Password, Client Domain Name, Client IP Address, Client Browser, Server Address, Server Port, Referrer Page URL, Method, and Full Request. At the bottom are "Cancel" and "Insert" buttons.

The **Parameter** list includes items corresponding to all of the `<@CGIPARAM>` tag parameters.

This action inserts `<@CGIPARAM NAME=paramName>`, where *paramName* is the CGI parameter name corresponding to the selected item in the list.

Action Result Item

To insert a meta tag that returns values from the first row of results for previously executed actions in the current application file execution, choose **Action Result Item** from the **Category** drop-down menu.

The dialog box titled "Insert Meta Tag" shows the "Category" dropdown set to "Action Result Item". Below it, the "Action:" dropdown is set to "Search". The "Item Number:" text box contains the value "0". A note at the bottom states: "Only items from the first row of action results are available with this meta tag. Use the resultSet variable to access items from other rows." At the bottom are "Cancel" and "Insert" buttons.

Action result items specified here are data from the first row of results generated by the action. A Search action, for example, may return 100 rows of data in ten columns. Specifying action result item six from that action (`<@ACTIONRESULT searchActionName 6>`) gives you the value from row one, column six.

Working With Variables

Working with Variables in Witango

Variables are placeholders that you can assign a value to; they are created using the `<@DEFINE>` meta tag assigned values using the Assign action or the `<@ASSIGN>` meta tag.

Every variable belongs to a *scope*, which tells Witango if the variable is to be used only for the particular application file execution, within a Witango application, for a user, or for a particular domain being served with Witango. Variables can also belong to special scopes within Witango class files that apply to a method or an instance of a Witango class file.

Arrays are a special variable type that allow you to create a structured data table with multiple values, as opposed to standard variables which only store one value.

You can also create variables that contain XML data structures (document instance variables) and variables that contain objects.

One important set of variables determines the behavior of certain Witango options. These are called *configuration variables*.

Variables are covered in great detail in the Witango Programmers Guide, and it is recommended that the user familiarise themselves with this material before completing this chapter. This chapter provides a guide as to how the Witango interface allows the user to work with variables.

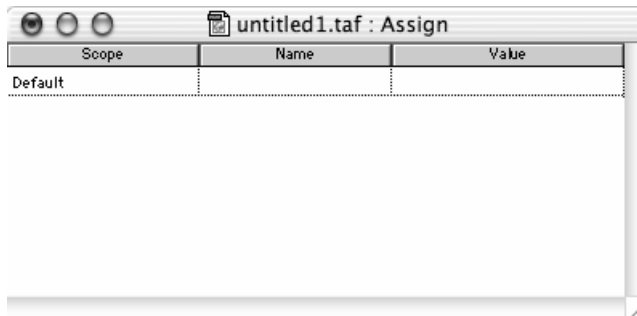
Assigning Variables With the Assign Action

To assign a variable

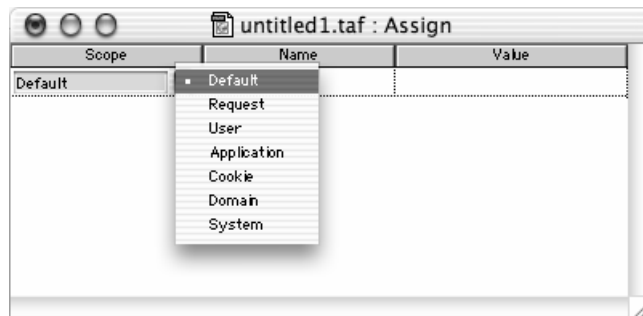


- Drag the Assign action from the Actions bar into the Witango application file or Witango class file window.

The Assign action is inserted into the Witango application file or Witango class file; a new variable assignment appears if it is a new Assign action.



The following is an Assign editing window:



The Assign editing window consists of a three-column list. Each row of the list shows the variable scope, name, and value.

You can add one or more variable assignments to each Assign action.

Editing Variable Assignments

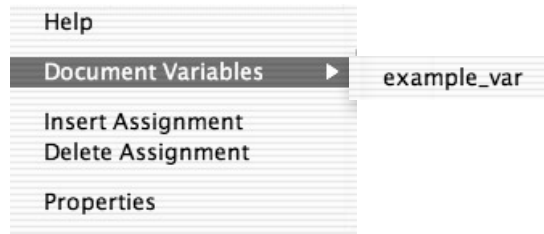
You type text in the **Name** and **Value** fields. The **Scope** field contains a text field/drop-down menu that allows you to select a standard scope or define a custom scope.

The standard scopes are **Request**, **Cookie**, **User**, **Application**, **Domain**, or **System**. You can also choose default scope from the

Scope field. For information on how Witango assigns scope to variables with default specified, see the Witango Programmers Guide.

Context-Sensitive Menu

Using the context-sensitive menu, you can fill in names from previously-assigned variables.



The **Document Variables** context-sensitive submenu contains an alphabetical list of all variables assigned (via Assign actions) in the current Witango application file or Witango class file. Selecting an item from this list sets both the scope and name for the variable assignment.

To add a new variable assignment to this window

Do one of the following:

- From the **Edit** menu, choose **Insert**.
- On the main toolbar, click the Insert icon.
- Right-click the Assign window, and choose **Insert Assignment** from the context-sensitive menu that appears.



To select a variable assignment

- Click the row.

To move a variable assignment

- Click an assignment and drag it to the desired position.

To delete a variable assignment

- 1 Click the assignment you want to delete.
- 2 Do one of the following:
 - From the **Edit** menu, choose Delete.
 - On the main toolbar, click the Delete icon.
 - Press `Delete`.



- Right-click and then choose **Delete Assignment** from the context-sensitive menu that appears.
- 3 When the dialog box appears, asking you to confirm the deletion, click **OK**.



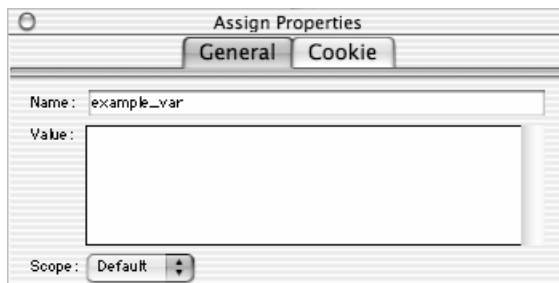
Tip You can bypass the confirmation dialog box by holding down the **Ctrl** key when choosing **Delete**.

To view Assign Properties

Do one of the following:

- From the **View** menu, choose **Properties**.
- Right-click on the row and choose **Properties** from the context-sensitive menu that appears.

The Assign Properties window appears.



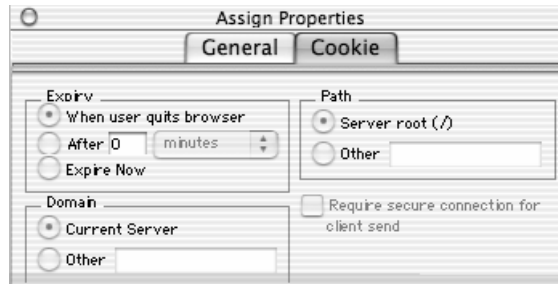
The Assign Properties window consists of two sections:

- **General.** Click the **General** tab to edit the name, value, and scope in the General section.
- **Cookie.** Click the **Cookie** tab to display controls for editing attributes of cookie variables. (Each group of settings corresponds to an element of the Set-Cookie HTTP header line.)



Note The **Cookie** tab to access the Cookie section appears only if the scope for the current assignment is “Cookie”.

The default values for new cookies are as shown:



- **Expiry**

When user quits browser is the default cookie behavior as described in the cookie specifications. When this option is chosen, the `Expires` value is omitted from `Set-Cookie` line in the cookie header.

After __ [time units]. The drop-down menu for time units has minutes, hours, days, and years options. The text entry field holds up to 31 characters; a meta tag can be specified there.

- **Domain**

Current server omits the `Domain` value from the `Set-Cookie` line, causing the cookie to be valid for the current server.

Other allows specification of any domain string up to 31 characters. “.example.com”, for example, would cause the cookie to be sent back to `www.example.com`, `demo.example.com`, `sales.example.com`, and so on.

- **Path**

Server root (/) specifies that the cookie be sent for all paths within the specified domain.

Other allows specification of a path string up to 31 characters. For example, `/Witango/` would cause the cookie to be sent back only for URLs below the `Witango` folder.

- **Require secure connection for client send**

True (enabled) or **False** (disabled). This option sets the `Secure` value of the `Set-Cookie` line. If the value is set to true, then the cookie is sent back by the Web browser only if a secure connection is being made.

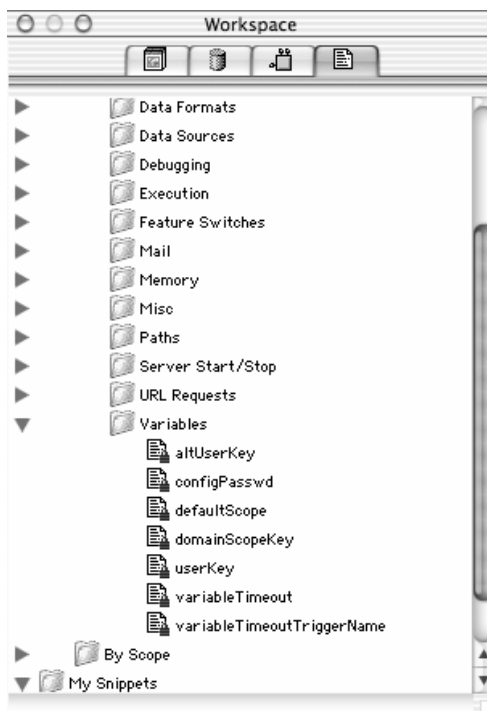
Shortcuts to Configuration Variable Assignments: Snippets

The Workspace contains configuration variable snippets which create assignments for configuration variables. The configuration variable snippets are organized alphabetically by category (Arrays, Data Format, Debugging, and so on) and by Scope (Application, Domain, Local, System, and User). This provides you with a quick way of creating assignments to configuration variables by dragging these snippets into an Assign action window or an HTML editing field.

For more information, see “Using Snippets” on page 117.

To use the configuration variable assignment snippets

- 1 Open the Snippets Workspace by showing the Workspace and clicking the **Snippets** tab.
- 2 Expand the **Configuration Variables** folder.



A tooltip appears if you place your cursor over the snippet, giving the content of the snippet.

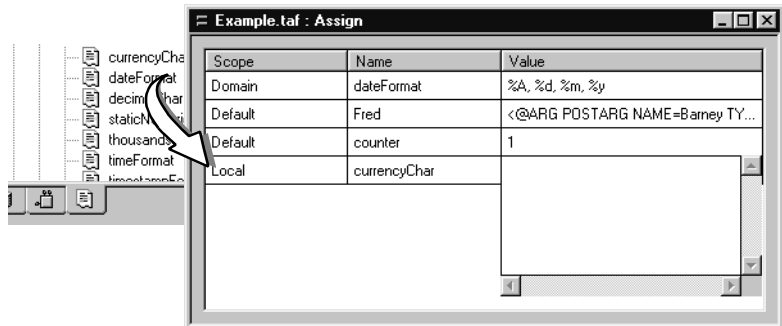
- 3 Choose the configuration variable to which you want an assignment created, and drag it into the HTML or Assign action window.

For more information, see
“Using Snippets” on
page 117.

Dragging a configuration variable snippet into the HTML window creates an assignment using meta tags, for example, `<@ASSIGN NAME="currencyChar" SCOPE="Local" Value="¥">`. Witango puts your cursor at the special ¥ symbol, so you can just start typing the value of the configuration variable.

When you drag a snippet into an Assign action window, the configuration variable assignment is automatically created for you.

The **Value** field becomes active, ready for you to type a value in:



If you choose a snippet without a defined scope (that is, from the category section of the configuration variable snippet), the scope is set to Default.

SECTION III

Witango Builders

How to Use Witango Builders

This section contains a chapter on how to use Builders in general, and two chapters giving details on the Search Builder and New Record Builder.

This section contains chapters on the following topics:

- Chapter 9, Building Actions Using Witango Builders on page 165
- Chapter 10, Configuring the Search Builder on page 171
- Chapter 11, Configuring the New Record Builder on page 209.

This section is recommended for new users of Witango.

Building Actions Using Witango Builders

How Witango Builders Work in Application Files

Witango builders help you create and generate a sequence of actions to perform specific tasks. Builders are added to an application file in exactly the same way you add any other action.

Once you add a builder and configure it, the actions the builder generates are inserted automatically into the application file. Builder information is saved in the application file format, making it cross-platform capable. This means you can use any application file containing a builder across all platforms.

Witango Studio provides two builders.

- The *Search Builder* builds the actions required to perform a search of database records and to update and delete them.

For details about how to configure the Search Builder, See “About the Search Builder” on page 172.

- The *New Record Builder* builds the actions required to add a new record to a database.

For details about how to configure the New Record Builder, See “About the New Record Builder” on page 210.

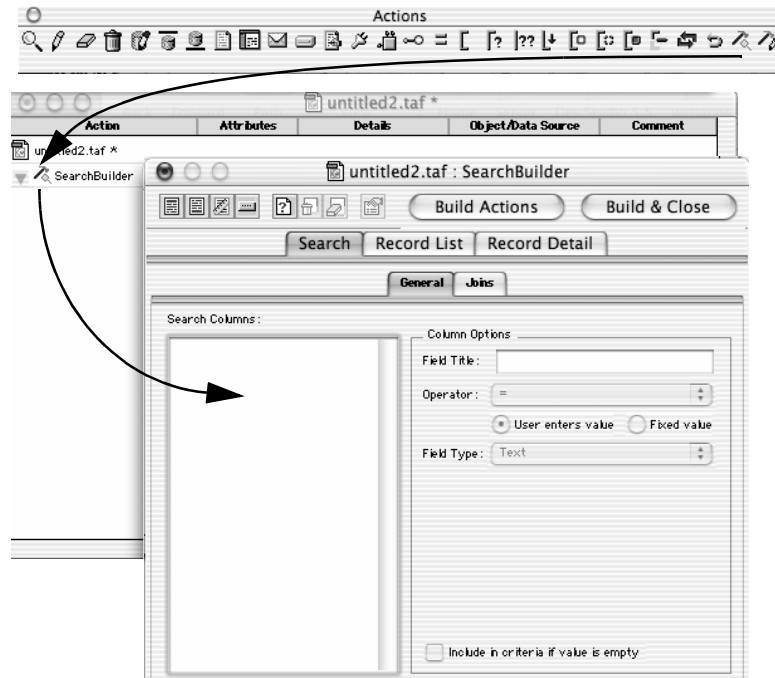
The topics covered in this chapter include:

- adding a builder to an application file
- generating builder actions
- details about the actions generated by the builders.

Adding a Builder to an Application File

With a new or existing application file open, you add a builder in the same way you add an action; that is, you drag the builder icon from the Actions bar into the application file window.

When you drag either of the builder icons into an application file window, the corresponding builder window opens. For example, if you drag the Search Builder icon from the Actions bar into an open application file, the Search Builder window opens so you can immediately set Search Builder options.



Dragging a builder from the Actions bar adds the builder to the application file, and the builder's options

In the application file window, the corresponding builder icon appears with a default name, just like with actions. The default name for the

Search Builder is “Search_Builder” and for the New Record Builder, “Record_Builder”.



Note Witango Studio can properly process one Search Builder and one New Record Builder in the same application file. If you want to use multiple builders of the same type in a single file, you must handle them using If actions and additional request arguments.

For more information, see “General Forms of Conditional Actions” on page 280.

If you add a builder to an application file that already contains the same builder using the default name, Witango adds a number to the default name and increments it by one for each subsequent addition, for example, “Search_Builder1”, “Search_Builder2”, and so on.



Note The title of the builder window is in the same form as an action window, <File name> : <Builder name>.

Within the application file, a builder behaves exactly like an action group. That is, it contains actions that can appear in expanded or collapsed form. In the preceding diagram, the Search Builder icon represents a group, but the group is empty until its actions are built. Witango Server ignores unbuilt builders.

Page Format Table Settings

Witango Studio by default saves your Witango builder page formats and uses these settings for new tables you create in the builders, allowing you to develop a consistent appearance for Web sites and Witango applications quickly.

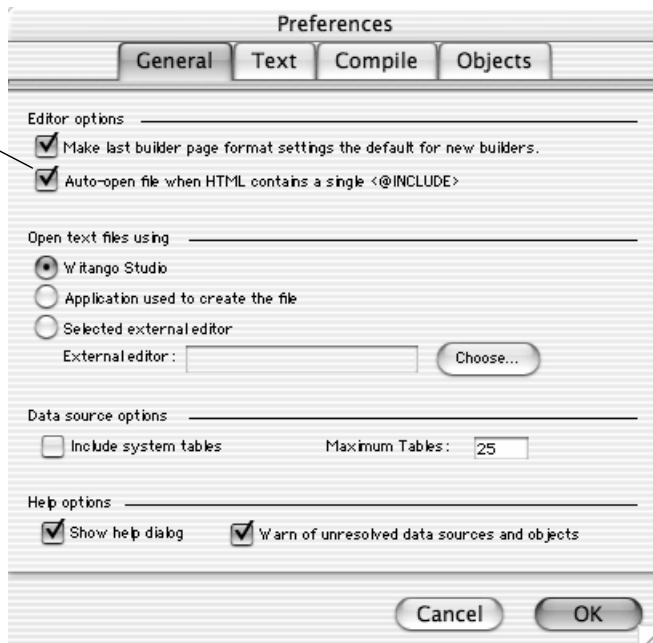
Unique page formats set in the New Record Builder and in each of the Search, Record List, and Record Detail tabs in the Search Builder are saved and used as defaults for new tables.

To change whether page format are saved

- I From the **Edit** menu, choose **Preferences**.

The Preferences dialog box appears:

Save page
format
settings
check box



- 2 Click the **General** tab.
- 3 Under **Studio options**, check or uncheck **Save last-used page format** settings in the Builders and use them as defaults for new Builders.

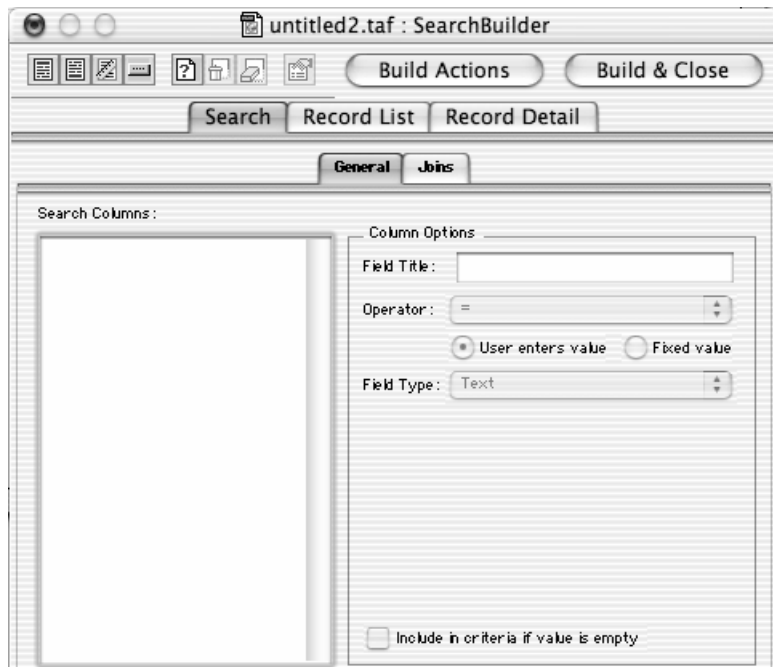
Building the Actions

Once you have entered all the relevant information in the builder window, you can generate the actions.

To build Search or New Record actions in an application file

- 1 Open a builder window (if it is not already open).

The following is an example of a builder window:



- 2 Enter the required information.

- For information on using the Search Builder, see Chapter 11.
- For information on using the New Record Builder, see Chapter 12.

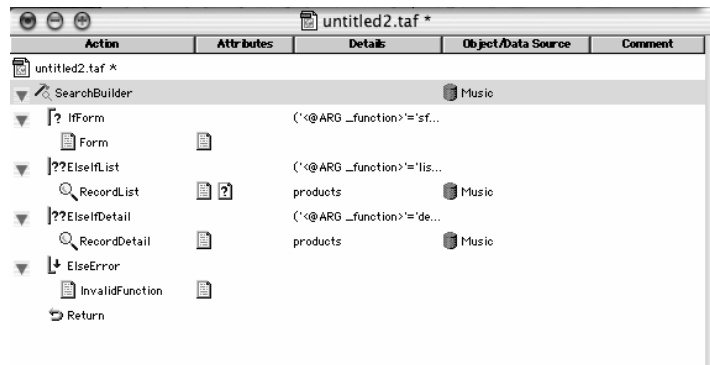
- 3 Click **Build Actions**.

The builder checks to see if you have entered all the information required to build the actions.

- If you have not, an error message appears and the build process stops.
- If you have, it builds the actions required to perform the task.

For information on generating actions in the Search Builder, See “Actions Built by the Search Builder” on page 206. For information on generating actions for the New Record Builder, See “Actions Built by the New Record Builder” on page 220.

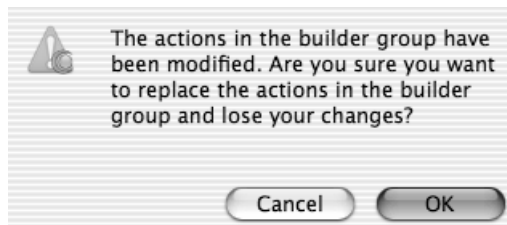
On completion, the actions appear in the builder group. The following figure shows an application file window after successfully building actions using the Search Builder:



You can view the hierarchy of a builder in an application file. Click either the plus sign (+) or minus sign (-) to expand or collapse the builder group accordingly.

When building actions, the builder replaces any existing actions in the group:

- If the builder group is empty, the actions are inserted into the builder group.
- If the builder group only contains the actions that were generated by the builder and those actions have not been modified, the new actions replace the old actions within the builder group.
- If there are actions in the builder group that the builder did not generate, or the actions in the builder group have been modified, a message box appears.



- To replace all actions in the builder group with the new actions, click **Yes**.
- To stop the build action, click **No**.

Configuring the Search Builder

Witango Search Builder Options and Setup

The *Search Builder* builds a series of actions that allows you to create Web pages for users to search a database, view the results of the search, and view, update, and delete individual records.

The topics covered in this chapter include:

- setting search, record list, and record detail options
- formatting the search form, and record list and record detail Web pages
- customizing your Web forms and pages, and creating response messages
- tips on modifying how the Search Builder builds actions
- defining joins.

About the Search Builder

Using the Search Builder, you can quickly and easily build the actions necessary to:

- display a search form allowing users to specify search criteria
- display a list of matching records
- allow viewing of detailed information on a single record
- allow editing and deleting of the records found.

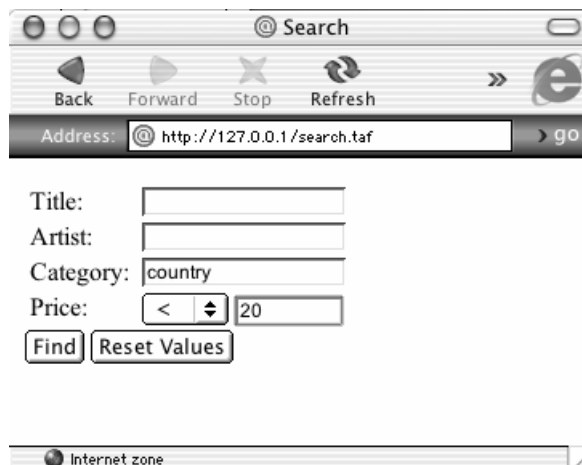
What Users See in Their Web Browser

When searching the database, users generally encounter three types of Web pages: search form, record list Web page, and record detail Web page.

This section describes what users experience when they visit your Web site. The next section, *How You Create These Web Pages* page 174, gives you a brief idea as to how you can easily create this user experience using the Search Builder.

Search Form

Based on the settings in each of the **option** groups and the actions generated, the following is an example of a search form users see in their Web browser. :



The screenshot shows a web browser window titled "Search". The address bar displays "http://127.0.0.1/search.taf". The browser interface includes navigation buttons: Back, Forward, Stop, Refresh, and a search icon. Below the address bar, the search form contains the following fields and controls:

- Title:
- Artist:
- Category:
- Price: (with a spinner control)
- Buttons: Find, Reset Values

At the bottom of the browser window, the status bar shows "Internet zone".

On the search form, the user enters the criteria for the records to return.

By default, a **Begins with** search is performed on text columns and an **=** (equals) search on all others. If you want, you can instead let users select each search operator from a drop-down menu.

To initiate the search, the user clicks **Find**.

Record List Web Page

Witango Server searches the data source for records matching the user's criteria and displays them on the record list Web page.

The following is an example of the record list Web page:



Note Character data from data sources is by default stripped of trailing spaces. You can disable this feature by using the configuration variable `stripChars`: assign the variable the value `false` in local scope for a particular application file, or, if you want to set the variable for all data sources, use the Witango Administration Manager (the `config.taf` application file) to change the variable's value to `false` in system scope.

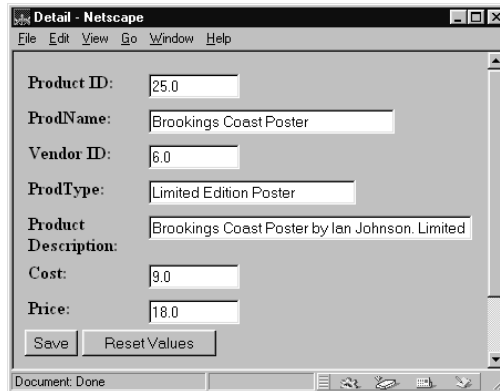
If you want, you can set up the record list Web page to display **Next** and **Previous** buttons, so users can browse through large result sets.

To view detailed information for a record, the user clicks the name of the record in the list, which is hot linked to the record detail.

Record Detail Web Page

The record detail Web page displays more information on the selected record and—if you allow it—lets the user edit or delete it.

The following is an example of the record detail Web page:



The screenshot shows a Netscape browser window titled "Detail - Netscape". The browser's menu bar includes "File", "Edit", "View", "Go", "Window", and "Help". The main content area displays a form with the following fields and values:

Product ID:	25.0
ProdName:	Brookings Coast Poster
Vendor ID:	6.0
ProdType:	Limited Edition Poster
Product Description:	Brookings Coast Poster by Ian Johnson. Limited
Cost:	9.0
Price:	18.0

At the bottom of the form are two buttons: "Save" and "Reset Values". The status bar at the bottom of the browser window displays "Document: Done".

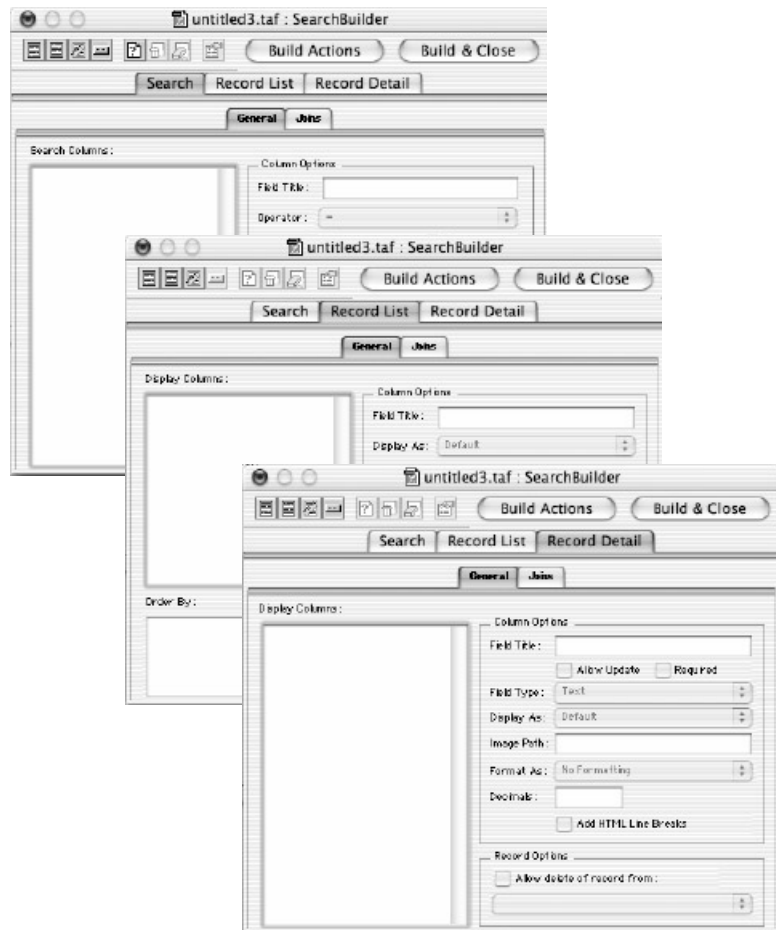
How You Create These Web Pages

When you open the Search Builder window you see three tabs: Search, Record List, and Record Detail. These tabs correspond to the three main groups of options you can specify. Each option group is represented by a page in the Search Builder window:

To switch among these three pages, click the appropriate tab in the Search Builder window.

For more information on joins, see Defining Joins on page 205 and Joining Database Tables on page 336.

Unless you are working with joins, make sure you select the General tab at the bottom of the Search Builder window.



Search Page

The Search page allows you to specify the columns you want users to search. It also defines the format of the search form you want users to see in their Web browsers.

Record List Page

The Record List page allows you to specify the columns you want to display in the record list that is returned to the user after a search. It also defines the format of the record list Web page.

Record Detail Page

The Record Detail page allows you to specify the columns you want to include on the single-record display Web page, which appears after a user clicks a specific record in the record list. It also defines the format of the record detail Web page. In addition, you can set up the record detail Web page to allow users to delete the record and/or edit specified columns.

Determining What is Displayed in the Web Browser

What you specify on the three pages of the Search Builder window determines what the users see and are able to do in their Web browsers. The follow table is a summary of the basic relationship:

Search Builder window	Web browser
Search page	search form
Record List page	record list Web page
Record Detail page	record detail Web page

Main Steps to Use the Search Builder

The standard way to use the Search Builder is to fill out the contents of the three pages of the Search Builder window, in the following sequence:

- Search page
For more information, see “Setting Search Options” on page 178.
- Record List page
For more information, see “Setting Record List Options” on page 189.
- Record Detail page
For more information, see “Setting Record Detail Options” on page 197.

Then, format the pages for display in the Web browser and customize the response messages related to these pages:

- search form
For more information, see “Formatting the Search Form” on page 186 and Customizing Your Search Form and Response Messages on page 187.
- record list Web page

For more information, see “Formatting the Record List Web Page” on page 195 and Customizing Your Record List Web Page on page 196.

- record detail Web page

For more information, see “Formatting the Record Detail Web Page” on page 201 and Customizing Your Record Detail Web Page and Response Messages on page 202.

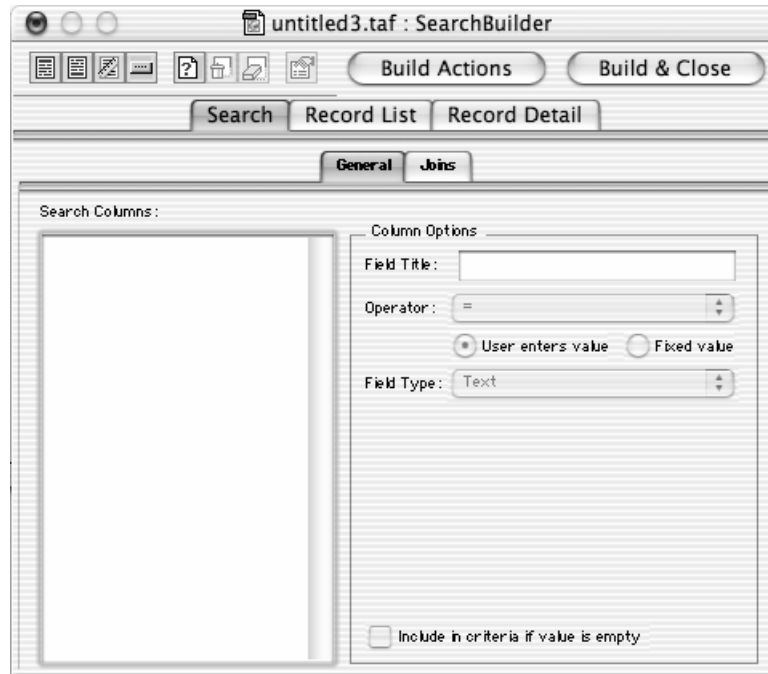
In some cases, you can simplify the process by skipping some steps. For more information, see “Simplified Steps to Use the Search Builder” on page 204.

Setting Search Options



Search Builder

When you drag the Search Builder icon from the Actions bar into an application file, the Search Builder window opens, displaying the Search page:



You use the search options on this page to define how the search form appears to the user, which columns the user can search on, and how the values entered by the user are used to search the database. You can also define fixed criteria in addition to the ones the user enters.



Tip You can save your Witango builder page formats to use for new tables you create in the builders. For more information, see “Page Format Table Settings” on page 167.

Search Columns List

Drag columns from the Data Sources Workspace to this list to use them in defining the search. Columns in the **Search Columns** list appear in the format *table_name.column_name*. The order of the columns in the **Search Columns** list determines the order of the fields on the resulting search form.

The following table describes the operations you can perform on columns:

To ...	Do This ...
Reorder columns	Select the columns and drag them to a different location in the list.
Delete columns	Select the columns. Choose Delete from the Edit menu, press DELETE on the keyboard, select the Delete icon on the main toolbar, or right-click and choose Delete from the context-sensitive menu that appears.
Delete columns without confirmation	Hold down the CTRL key while using the Delete command.

Column Options

Use the **Column Options** portion of the Search page to configure each search column. You can specify how each column's entry field appears on the search form and how the value entered by the user is used to search the database.

How the column value searches the database.

Lets the user enter the search value on the search form.

The type of value entry field to appear on the search form.

Specify the value yourself and prevent this column from appearing on the search page.

This title for this column appears on the

Includes the column, even when no value is specified

Column Options

Field Title:

Operator:

=

☒ User enters value

☐ Fixed value

Field Type:

Text

☐ Include in criteria if value is empty

Field Title

In **Field Title**, set the title of the value entry field for the column as you want it to appear on the search form.

Operator

Use this option to set the search operator for the column. For example, if the operator is set to **Begins with**, Witango searches the database for records that begin with the column's search value. If you select **User Enters**, the search form displays a drop-down menu of the operators available, and the user can select which operator to use.

User Enters Value

Select this option if you want the user to enter a value in an entry field on the search form. The available field types are: **Text**, Drop-down List, List Box, Check Box, and **Radio Buttons**.

Field Type

To select a value entry field type for a column, select the column in the **Search Columns** list and then an item from the **Field Type** drop-down menu.

A Field Properties dialog box appears allowing you to specify values for the field.



Field Properties

You can edit a field's values at any time by clicking the Field Properties button in the Search Builder window or by choosing **Field Properties** from the **Attributes** menu.

The following describes each of the options in the Field Properties dialog box for each field type:

- **Text.** Use the Text field type to provide single- or multi-line entry of values.

**Password
Field check
box**

Default Value is the default value you want to appear on the new record entry form.

Maximum Length is the maximum number of characters the user can enter in the field. This option is not available when **Scrolling Field** is selected because HTML does not support it.

Width is the width of the field in characters.

Height is the number of lines of text displayed in the field without scrolling. This field is available only when you select **Scrolling Field**.

Scrolling Field. The height of a non-scrolling field is always “1”.

Password Field. Selecting this option generate form input elements with the attribute `TYPE=PASSWORD`, which conceals the characters that the user enters into the text field (for example, often asterisks or bullets are shown).

- **Drop-down List, List Box, and Radio Buttons.** Each of these field types lets the user select the search value from a predefined list.

Drop-down List	Color: Red
List Box	Color: Red Green Blue Orange Purple
Radio Buttons	Color: ☒ Red ☐ Green ☐ Blue ☐ Orange ☐ Purple ☐ Pink

For example:

Use the Field Properties dialog box for these field types to specify the values and choose the default. The same dialog box appears for all three field types.

To add an item to the values list

- I From the Field Type drop-down menu, select **Drop-down List**, **List Box**, or **Radio Buttons**; then do one of the following:
 - In the Search Builder window, click the Field Properties icon.
 - From the **Attributes** menu, choose **Field Properties**.



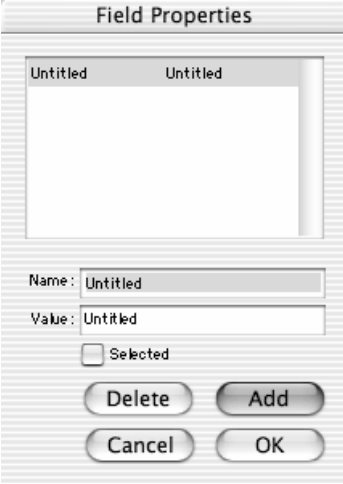
Field Properties

The Field Properties dialog box for the selected column appears:

The image shows a dialog box titled "Field Properties". It has a large empty rectangular area at the top. Below this area are two text input fields: "Name:" and "Value:". Under the "Value:" field is a checkbox labeled "Selected". At the bottom of the dialog are four buttons: "Delete", "Add", "Cancel", and "OK".

2 Choose **New**.

An "Untitled" entry appears:

The image shows the same "Field Properties" dialog box, but now the large rectangular area at the top contains a single entry. The entry has a header bar with the word "Untitled" on the left and "Untitled" on the right. Below the header bar is a large empty rectangular area. Below the main area are the same "Name:" and "Value:" text input fields, both containing the text "Untitled". Under the "Value:" field is a checkbox labeled "Selected". At the bottom are the same four buttons: "Delete", "Add", "Cancel", and "OK".

3 Type a name in the **Name** field.

4 Press TAB to copy the name into the **Value** field, or enter a value for the item if you want it to be different from the name.

5 Click **New** to continue adding items to the values list.

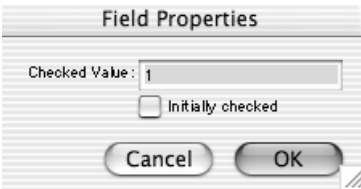
6 When you have added all of your items to the values list, click **OK**.

The **Name** determines what the user sees for this item. The **Value** determines what is used as the search value. If your database column uses abbreviations or codes for values, you can enter a more user-friendly value into the **Name** field and the actual value in the **Value** field. For example, if you are creating a field to search a “state” database column containing abbreviations such as “CA”, “NY”, and “GA”, you can use these values in the list items’ **Value** fields, and the full state names (“California”, “New York”, and “Georgia”) in the **Name** fields.

The following table lists the other operations you can perform on the values list:

To ...	Do This ...
Set which item will be initially selected	Select an item in the list and choose the Selected option. The item then appears in bold in the list. If you do not specify an item as Selected , the user’s Web browser determines which item is initially selected. Most Web browsers choose the first item, but some do not select any in this case.
Delete an item	Select it in the list, and click Delete . To delete without displaying a confirmation dialog box, hold down the CTRL key while deleting.
Create an item that causes the column to be omitted from the search criteria when chosen	Leave the Value field for that item empty and make sure the Include criteria if value is empty option is not selected.

- **Check Box.** The **Check Box** field type lets the user select between an empty search value (unchecked) and a value you specify (checked).



Checked Value is the value to be used for the search if the check box is selected by the user. If the check box is not selected, an empty value is used.

Initially Checked specifies whether the check box should be checked by default.

Fixed Value

For more information, see “Field Type” on page 180.

When this option is selected, the search value is hard-coded and no entry field appears on the search form. The value you specify is used for every search.

Using the **Value** drop-down menu, select one of the following options for a fixed value:

- **Value Entered.** Use the text box provided to enter the search value for the column.
- **SQL Expression.** The value returned by the SQL expression text entered is used as the search value. The text entered is evaluated by the database, and the result is used as the search value.



Note For ODBC data sources, you can enter ODBC scalar functions here.

- **SQL Statement.** The SQL statement entered is executed, the results retrieved, and the first data item of the results is used as the search value. For example, if you enter:


```
SELECT MAX (cust_num) FROM customer
```

 the largest customer number is used as the search value.
- **Current Timestamp.** The current timestamp (date and time combined) on the Witango Server computer is used as the search value.
- **Current Date.** The current date on the Witango Server computer is used as the search value.
- **Current Time.** The current time on the Witango Server computer is used as the search value.
- **CGI Parameters.** The rest of the fixed value options are referred to collectively as *CGI parameters*. They include **Client Name**, **Client Domain**, **Client IP Address**, **Client Browser**, **Server Address**, **Server Port**, **Referer Page URL**, and **Method**. They are passed by either the user's Web browser or the Web server with each request to Witango. When you specify one of these parameters as the search value, the parameter value used is the one passed in when the user clicks **Find**.

Summary: Setting Column Options

The following table describes the settings you can make in the Column Options portion of the Search page:

To ...	Do This ...
Let the user specify the search value for a column	Select the column in the Search Columns list. Select the User enters value radio button. This option defines the column as a user-searchable field, and a value entry field will appear on the search form.
Specify the title to appear for a column's search form value entry field	Select the column in the Search Columns list. The column's name appears in the Field Title field. Replace the name with the desired field title. Witango remembers the entered title and uses it as the default the next time you use the column.
Specify the type of value entry field to be displayed for a column	Select the column in the Search Columns list. Make sure the User enters value option is selected. From the Field Type drop-down menu, select the type of field you want displayed. A Value dialog box appears, allowing you to specify the field attributes.
Include a column in the search even if the user leaves its value entry field empty	Select the column in the Search Columns list. Select the Include criteria if value is empty option. If the user leaves this column's value empty and clicks Find , only records that have an empty value in the column are returned. If the option is not selected (the default), the column is omitted from the search when the user does not enter a value.
Specify the operator Witango uses when comparing the database column values with the search value	Select the column in the Search Columns list. From the Operator drop-down menu, select the operator that specifies how you would like the column searched. For example, Begins with searches for values that begin with the entered value.
Let the user select the search operator for a column	Select the column in the Search Columns list. Select User Enters from the Operator drop-down menu. A drop-down menu of available operators appears beside the column's value entry field on the search form.
Hard-code the search value for a column	Select the column in the Search Columns list. Select the Fixed value option. From the Value drop-down menu, select a search value. You can use one of the preset values, such as current date or time, select Value Entered to enter a value yourself, select one of the SQL options to get a search value from the data source, or select a CGI parameter. Columns specified as Fixed value do not appear on the search form.

Formatting the Search Form

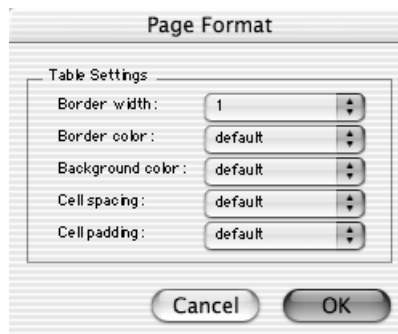
You can set up format options to define how the search fields and their titles appear to users on their Web browsers.

To change the format of the search form

Do one of the following:

- In the Search Builder window, click the **Page Format** icon.
- From the **Attributes** menu, choose **Page Format**.

The Page Format dialog box appears:



Specify the table attributes as follows:

- **Border width.** The width of the table border in pixels. Select **none**, or from numbers **1** to **8**.
- **Border color.** The color of the table border. Select **default** or a color from the list.



Note For **Border color**, **Background color**, **Cell spacing**, and **Cell padding**, selecting **default** instead of a value omits that attribute from the HTML and causes the Web browser's default setting to be used instead.

- **Background color.** The background of the table. Select **default** or a color from the list.
- **Cell spacing.** The amount of space, in pixels, inserted between individual cells in the table. Select **none**, or from numbers **1** to **8**.
- **Cell padding.** The amount of space, in pixels, between the border of a cell and the contents of the cell. Select **none**, or from numbers **1** to **8**.

Customizing Your Search Form and Response Messages

Header, Footer, and No Results HTML

Use Header HTML and Footer HTML to customize the search form by specifying HTML to appear above and below the search form.

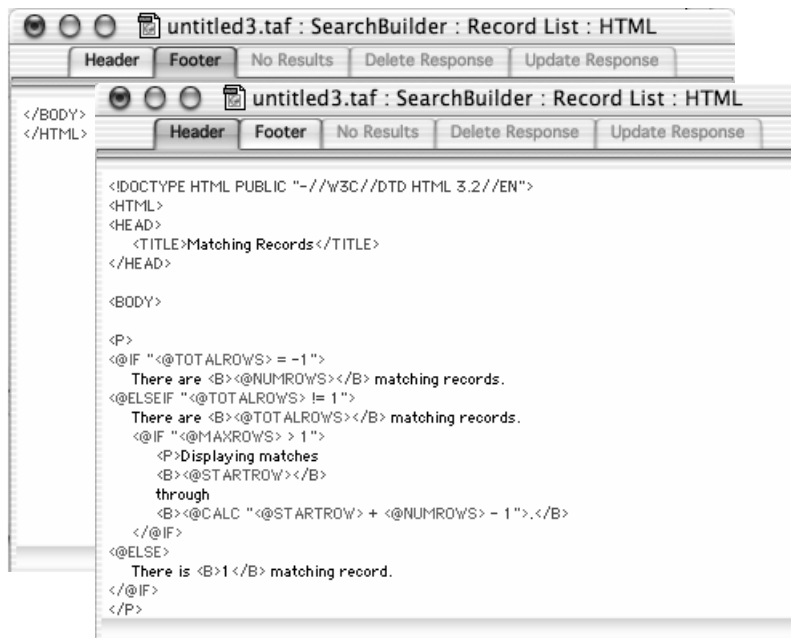
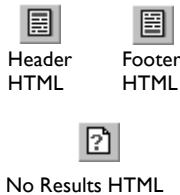
No Results HTML lets you specify the HTML to return when no records match the search criteria specified by the user.

To enter Header HTML, Footer HTML, or No Results HTML

1 Do one of the following:

- In the Search window, click the **Header HTML**, **Footer HTML**, or **No Results HTML** icon.
- From the **Attributes** menu, choose **Header HTML**, **Footer HTML**, or **No Results HTML**.

The corresponding HTML editing window appears:



You can switch between the HTML editing windows by clicking on the **Header**, **Footer**, and **No Results** tabs at the bottom of the HTML window.

- 2 Enter the HTML you want.
- 3 Close the editing window.

Changing Button Titles

The search form contains two buttons below your search fields:

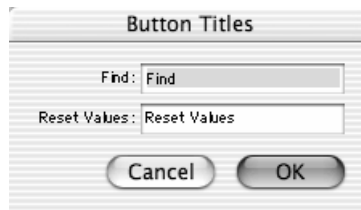
- The **Find** button initiates the search.
- The **Reset Values** button resets the entry fields to their default values.

To change button titles

1 Do one of the following:

- Click the **Button Titles** icon.
- From the **Attributes** menu, choose **Button Titles**.

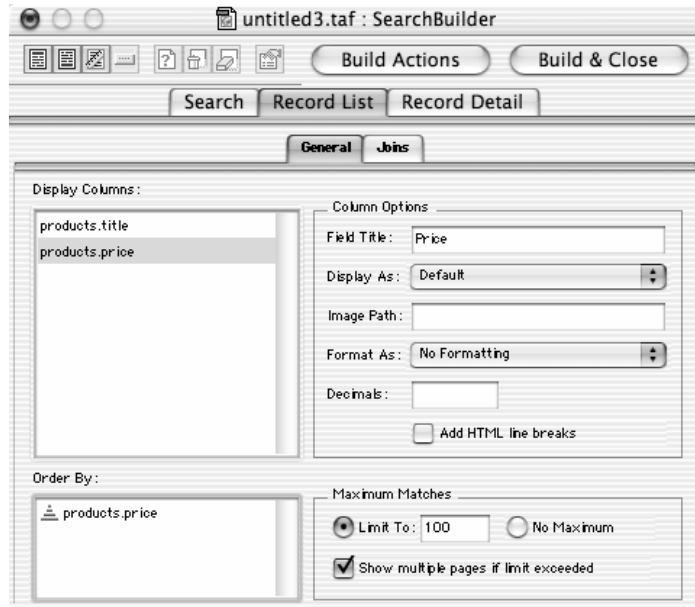
The Button Titles dialog box appears:



- 2 Enter new titles in the corresponding fields.
- 3 Click **OK**.

Setting Record List Options

The record list Web page consists of the results returned to the Web browser after Witango has performed the search. The Record List page of the Search Builder is used to define the appearance and functionality of the record list Web page.



Among other things, you can specify:

- which columns from each matching record are displayed
- the ordering of result records
- the maximum number of records to be returned
- whether you want **Next** and **Previous** buttons to appear, allowing paging through large result sets
- which column or columns appear as links to the record detail Web page.

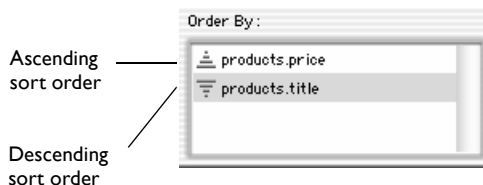
Display Columns

Drag columns from the Data Sources Workspace to this list to have them retrieved from the database and displayed on the record list Web page. The order in which columns appear in the **Display Columns** list determines their order on the Web page.

Order By

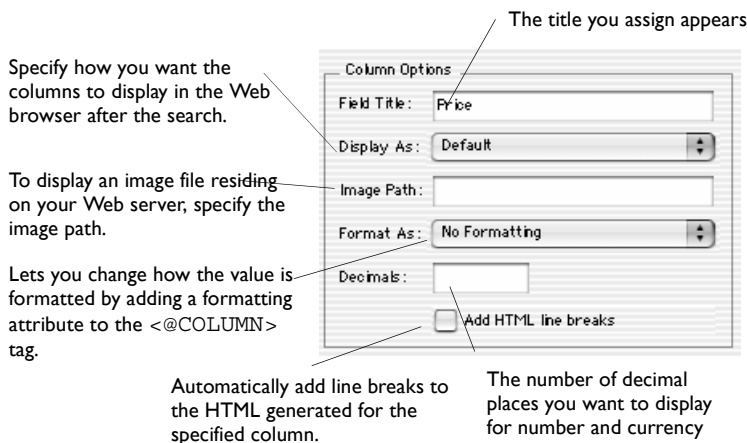
Records from the database are sorted on the record list Web page according to the order specified in the **Order By** list. You can drag any number of columns into this list; however, each of the columns must also appear in the **Display Columns** list.

The records are sorted by the first column listed. Then, records having the same values in that column are ordered by the second column, and so on. The default sort order is ascending, meaning records with lower values in the sort column appear first in the list. You can toggle between ascending and descending by clicking the  and  icons.



Column Options

Use the **Column Options** section to set up options for each column in the **Display Columns** list.



Field Title

In the **Field Title** field, enter the text you want to appear as the column title.

Display As

You can specify how you want the columns to display in the Web browser after the search. From the drop-down menu, select from the following options:

- **Normal Text** adds the following to the HTML generated for the specified column:

`[column tag]`

If the **Add HTML line breaks** option is selected for the specified column, the HTML becomes:

`[column tag] ENCODING=MULTILINE]`

For more information, see "Encoding Attribute" on page 10 of the *Meta Tags and Configuration Variables* manual.

- **Link to Detail.** Select this option to cause the selected column to appear as a hyperlink to the record detail Web page; that is, the user can click a value from this column and the detail for that record is displayed. You can specify more than one column as a link to the record detail.

If you specify no column as a link to the record detail Web page, the first column is automatically chosen for you when actions are built.

- **Link to URL Stored in Column.** If you have a URL stored in your database column, select this option to automatically generate a hot link. This option adds the following to the HTML generated for the specified column:

`<A HREF=" [columnvalue] "> [columnvalue] </A>`

- **Link to E-mail Address Stored in Column.** If you have an e-mail address specified in your database column, select this option to automatically generate a `mailto:` link. This option adds the following to the HTML generated for the specified column:

`<A HREF="mailto: [columnvalue] "> [columnvalue] </A>`

- **Image: File Name Stored in Column.** Select this option to display an image file residing on your Web server.

When you select this option, the **Image path** field is enabled in which you enter the path to the image.

This option adds the following to the HTML generated for the specified column:

`<IMG SRC=" [imagepath] [columnvalue] ENCODING=URL] ">`

- **Image: URL Stored in Column.** Select this option to display an image file residing on the Internet; that is, your database stores a

URL pointing to the image. This option adds the following to the HTML generated for the specified column:

```
<IMG SRC=" [columnvalue] " >
```

For more information, see “Encoding Attribute” on page 10 of the *Meta Tags and Configuration Variables* manual.

- **HTML.** Use this option if your database column contains HTML that you would like to display. This option adds the following to the HTML generated for the specified column:

```
[columnvalue ENCODING=NONE]
```

If the **Add HTML line breaks** option is selected for the specified column, the HTML becomes:

```
[columnvalue ENCODING=MULTILINE]
```

Format As

The **Format As** field is enabled only when you select either the **Normal Text** or **Link to Detail** option from the **Display As** drop-down menu.

Each of the following options in the drop-down menu (except **No Formatting**) adds a `FORMAT="formatstring"` attribute to the `<@COLUMN>` tag in the HTML generated for the column in the Record List action's Results HTML.

The following table lists the options and the corresponding format string:

Option	Format String
No Formatting	None
Date	datetime:@@dateFormat
Time	datetime:@@timeformat
Timestamp	datetime:@@timeStampFormat
Number with Commas	num:3*,'@@thousandsChar','@@decimalChar',,,',',
Number with No Commas	„decimals,'@@decimalChar',,,',',
Currency with Commas	num:3*,'@@thousandsChar',decimals, '@@decimalChar','@@currencyChar',, '@@currencyChar(',')
Currency with No Commas	„decimals,'@@decimalChar', '@@currencyChar',,'@@currencyChar(',')

Decimals

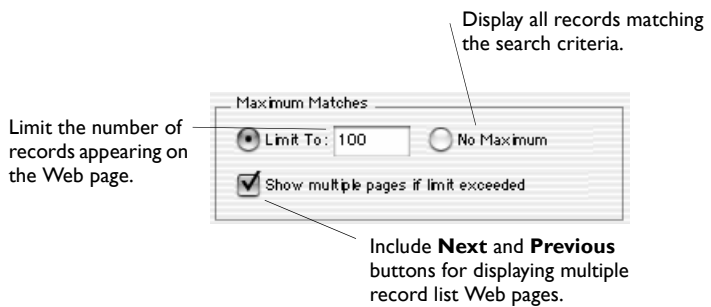
Specify the number of decimal places you want to display for number and currency values. The **Decimals** field is available only when you select one of the number or currency options from the **Format As** drop-down menu. The default is **0** for number options and **2** for currency options. An empty or non-numeric value is evaluated as **0**.

Add HTML line breaks

This option is available only when you select **Normal Text** or **HTML** from the **Display As** drop-down menu and **No Formatting** is selected from the **Format As** drop-down menu. Otherwise, this option is disabled.

Maximum Matches

Use the options in this section to restrict the number of matches displayed on the record list Web page.



Limit To

Select this option to limit the number of records returned by the search to the number specified. For example, to show only the first 10 records matching the search criteria, select this option and enter "10" in the **Limit To** field.

No Maximum

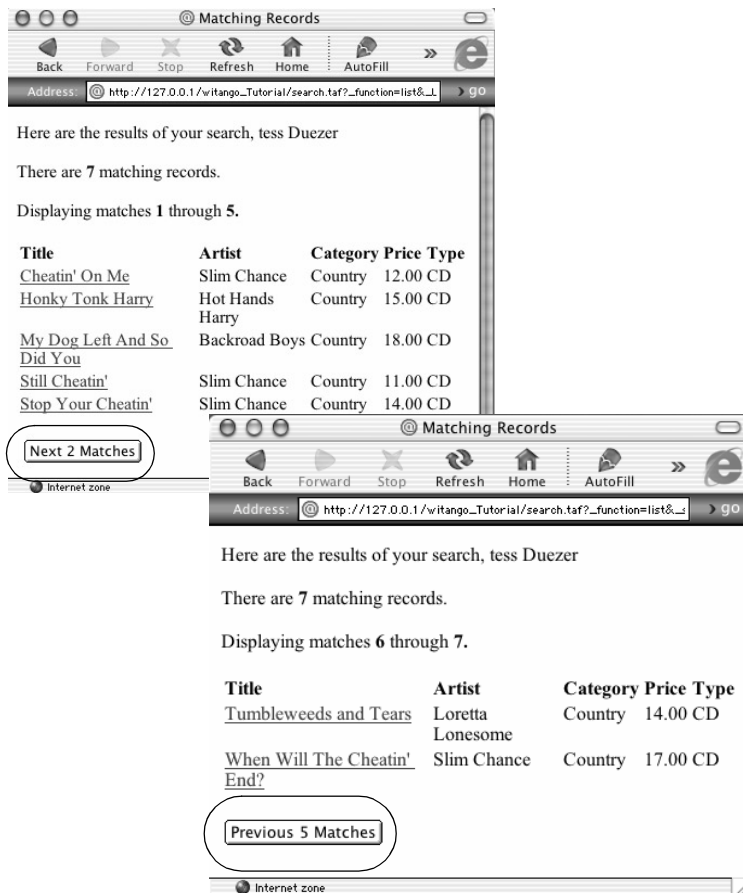
If you select the **No Maximum** option, all records matching the search criteria are retrieved and displayed on the record list Web page.

Show Multiple Pages If Limit Exceeded

If you specify a maximum number of matches in the **Limit To** field, this option is available. If selected, a **Next** button appears on the record list

Web page (if the number of matching records exceeds the limit entered), along with an indication of the total number of records matching and which records are being displayed. When the user clicks the **Next** button, the next group of matching records appears. A **Previous** button appears on record list Web pages beyond the first, which allows the user to go backwards in the list of matching records.

Choosing the **Show Multiple Pages If Limit Exceeded** option causes result set information and a **Next** button to appear on the results list Web page.



Formatting the Record List Web Page

Use the format options to define how the record list is displayed.

Witango displays results records in a table with one row for each record.

First Name	Last Name	Department
Carolynn	Peterson	Sales
Dave	Heartsdale	Accounting
Linda	Stewart	Administration
Aaron	Smith	Accounting
Peter	Barken	Engineering
Linda	Jennings	Engineering
Peter	Jacobson	Sales
Richard	Frankin	Sales
Jenna	Smith	Administration
Erica	Manley	Sales

To change the format of the record list Web page

Do one of the following:

- In the Record List window, click the **Page Format** icon.
- From the **Attributes** menu, choose **Page Format**.



Page Format

The Page Format dialog box appears. This dialog box is identical to the one for the search Web page. See page 186 for details.

Customizing Your Record List Web Page

Header and Footer HTML

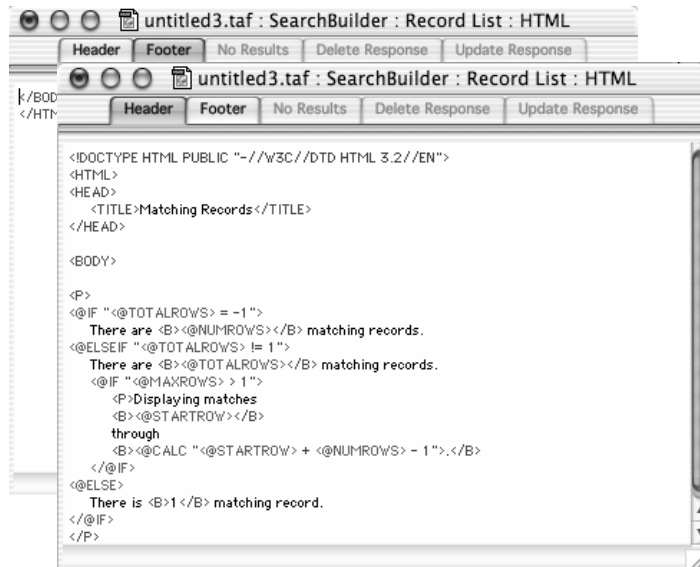
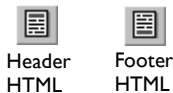
You use Header HTML and Footer HTML to customize the record list Web page by specifying HTML to appear above and below the record list.

To enter Header HTML and Footer HTML

1 Do one of the following:

- In the Record List window, click the **Header HTML** or **Footer HTML** icon.
- From the **Attributes** menu, choose **Header HTML** or **Footer HTML**.

The corresponding HTML editing window appears:



You can switch between the HTML editing windows by clicking on the **Header** and **Footer** tabs at the bottom of the HTML window.

- 2 Enter the HTML you want.
- 3 Close the editing window.

Setting Record Detail Options

Use the options in the Record Detail window of the Search Builder to define the appearance and functionality of the Web page returned when a user clicks on a record on the record list Web page. This Web page displays a single record and supports user editing and deletion, if you choose to allow it.



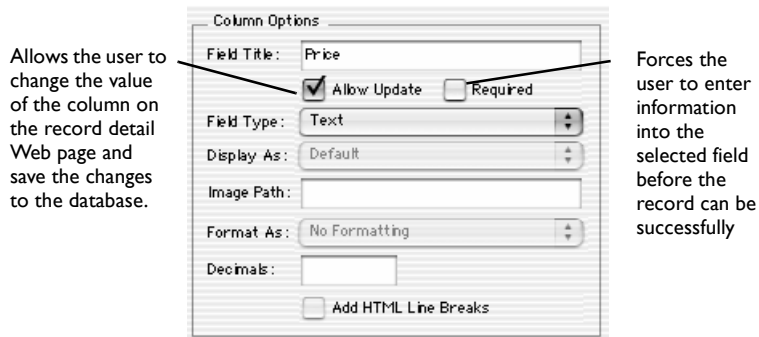
Display Columns

The columns appearing in this list are displayed on the record detail Web page. To add a column to the list, drag it from the Data Sources Workspace. The order in which columns appear in the **Display Columns** list is the order they appear on the record detail Web page.

Column Options

Use the **Column Options** section of the Record Detail window to configure each detail column. This section describes each of the column options.

Other than the **Allow Update** and **Required** options, the Column Options section for the Record Detail window is the same as the Column Options section for the Record List window. See Setting Record List Options on page 189.



Field Title

In the **Field Title** field, enter the title to appear for this column's value on the record detail Web page.

Allow Update

Select this option to allow the user to change the value of the column on the record detail Web page and save the changes to the database.

Required

If you allow users to update a database record by enabling **Allow Update**, you can also select the **Required** check box to force the user to enter information into the selected field before the record can be successfully updated.

If the user tries to update the record without entering a value in a required field, a message appears, telling the user that an entry into the field is required, and the form is displayed again.

Field Type

If you select the **Allow Update** option for a column, the **Field Type** drop-down menu and **Field Properties** icon are enabled, allowing you



Field Properties

to select the type of value editing field you want to appear for the column on the record detail Web page.

As with the search form, you can select from the available field types: **Text**, **Drop-down List**, **List Box**, **Check Box**, and **Radio Buttons**.

For more information, see “Field Type” on page 180.

You specify the field type and its options the same way you do in the Search window of the Search Builder.

The selected column’s Field Properties dialog box for each field type is the same in the Record Detail window as it is in the Search window, except you cannot specify a default value (text field type) or a selected item (drop-down list, list box, check box, and radio buttons). This is because the value of the column in the detail record determines the field’s initial value.

When creating value lists for drop-down list, list box, and radio button field types in the Record Detail window, make sure you enter the item values exactly as they appear in the database, and include all possible values. If Witango cannot find the column’s value in the list when it is constructing the record detail Web page for a record, no item is selected by default. Depending on the user’s Web browser, the first item may be selected or no item may be selected. Either way, if the user saves the record—even if no changes are made to that particular field—a new value (an empty value or the first value in the list) is saved in it.

For similar reasons, make sure check box fields are used only for columns that can contain either an empty value or the value you specify as its checked value.

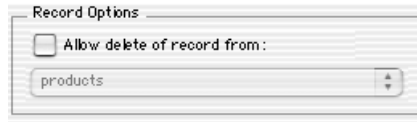
Setting Column Options: Display As, Image Path, Format As, Decimals, and Add HTML line breaks

Setting these options is identical to setting the column options for the record list Web page, except as follows:

- When you select the **Allow Update** option, these options are disabled.
- The **Display As** drop-down menu excludes the **Link to Detail** option.

Record Maintenance Options

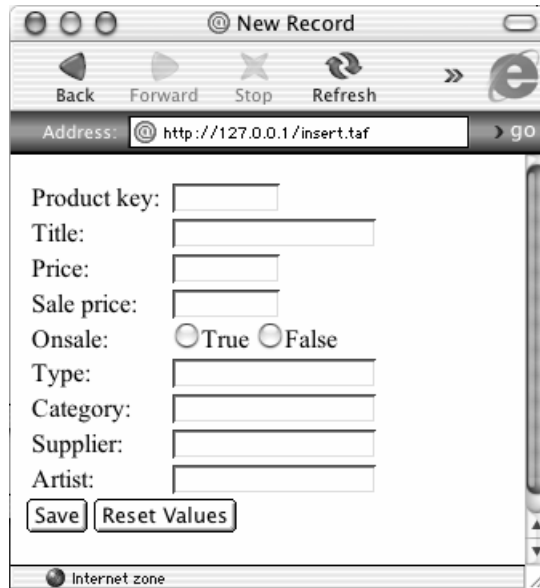
If you select the Allow Delete of Record From option, a Delete button is added to the record detail Web page, giving the user the ability to delete the current detail record.

A screenshot of a 'Record Options' dialog box. It contains a checkbox labeled 'Allow delete of record from:' which is currently unchecked. Below the checkbox is a text input field containing the word 'products'. To the right of the input field is a small drop-down arrow icon.

Deleting records from multiple tables simultaneously is not supported by the Search Builder, so if you have included columns from more than one table in the **Display Columns** list, use the drop-down menu to select the table whose record you want to delete.

Formatting the Record Detail Web Page

Use the format options to define how the detail column values and their titles are displayed.



To change the format of the record detail Web page

Do one of the following:

- In the Record Detail window, click the **Page Format** icon.
- From the **Attributes** menu, choose **Page Format**.



The Page Format dialog box appears. This dialog box is identical to the one for the search Web page. See page 186 for details.

Customizing Your Record Detail Web Page and Response Messages

Header, Footer, Update Response, and Delete Response HTML

You use Header HTML, Footer HTML, Update Response HTML, and Delete Response HTML to customize the record detail Web page.

Using Header HTML and Footer HTML, you can edit the HTML that you want to appear above and below the record data.

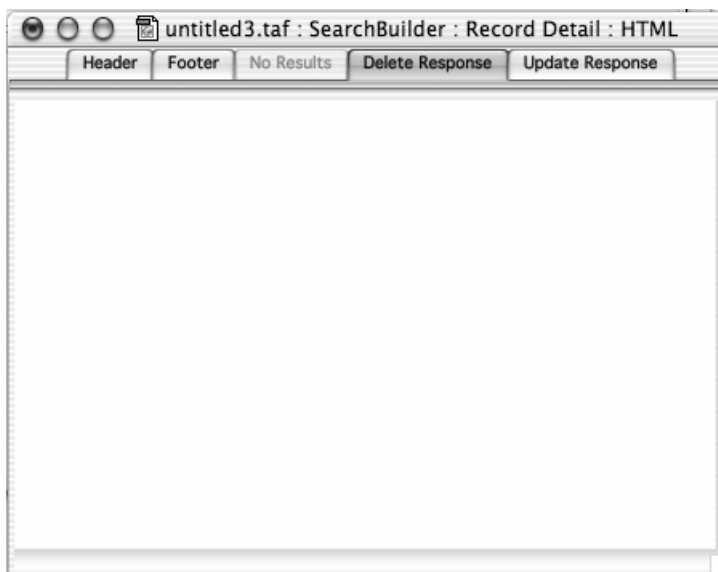
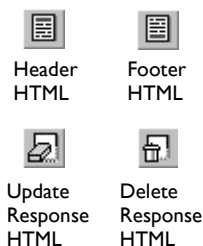
Using Update Response HTML and Delete Response HTML, you create messages in response to record updates and deletions.

To enter Header HTML, Footer HTML, Update Response HTML, or Delete Response HTML

I Do one of the following:

- In the Record Detail window, click the **Header HTML**, **Footer HTML**, **Update Response HTML**, or **Delete Response HTML** icon.
- From the **Attributes** menu, choose **Header HTML**, **Footer HTML**, **Update Response HTML**, or **Delete Response HTML**.

The corresponding HTML editing window appears.



You can switch between the HTML editing windows by clicking on the **Header**, **Footer**, **Delete Response**, and **Update Response** tabs at the bottom of the HTML window.

- 2 Enter the HTML you want.
- 3 Close the editing window.

Button Titles

When you make a field updatable, or when you allow users to delete records from the record detail Web page, buttons for these actions are added to the record detail Web page.

The record detail Web page contains three buttons below your record detail fields: **Save**, **Reset Values**, and **Delete**.

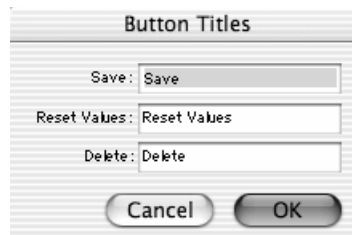
To change button titles

- I Do one of the following:
 - In the Record Detail window, click the **Button Titles** icon.
 - From the **Attributes** menu, choose **Button Titles**.



Button Titles

The Button Titles dialog box appears:



- 2 Enter new titles in the corresponding fields.
- 3 Click **OK**.

Simplified Steps to Use the Search Builder

In general, you use the Search Builder by following the standard sequence described in *Main Steps to Use the Search Builder* on page 176. In some cases, you can simplify the process by skipping some steps. The following information may be useful to you:

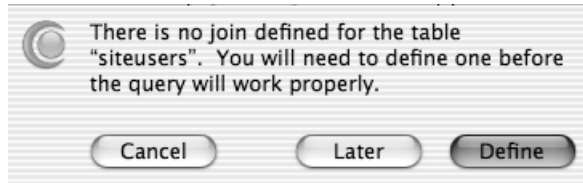
- **Using only fixed values in the Search options.** If you configure all of the Search columns to search for fixed values, the built actions on execution take the user directly to the record list and display the records matching the criteria you specify.
- **Specifying no Search columns.** If you do not specify Search columns, the built actions on execution take the user directly to the record list and display all the records in the database table.
- **Specifying no Record Detail columns.** If you do not specify Record Detail columns, the built actions do not contain detail functionality and no links appear in the record list.
- **Specifying no Record List columns.** If you do not specify Record List columns, the built actions on execution take the user straight to the record detail Web page for the first record matching the Search criteria.

Defining Joins

For complete details on what joins are and how to define them using Witango Studio, see [Joining Database Tables](#) on page 336.

You can include columns from more than one table in a search, if you define joins for the tables.

If you select columns from more than one table in a search, a dialog box appears telling you to define a join.



Either choose **Define** to go directly to the **Joins** section or **Later** if you want to define the join at a later time.

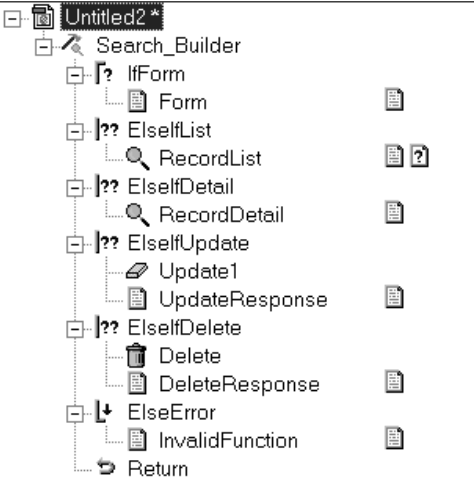
When you define the join, it adds the columns to a search. You must, however, define the join before you build the actions for the search or you save the application file.



Note You must define separate joins for the initial search (the one that displays the record list) and for the detail search.

Actions Built by the Search Builder

The actions built by the Search Builder appear in the application file as follows:



For more information, see “Building the Actions” on page 169.

The following table describes the actions resulting from the Search Builder process and the conditions under which actions are built:

IfForm

<@ARG _function> = sform or <@ARG _function> is empty

This section appears only if one or more user-enterable search columns are specified.

Form

ElselfList

<@ARG _function> = list

This section appears only if record list columns are specified. If no Form is present, this is an If action named IfList.

RecordList

If you selected **SQL Statement** for any column value, a Direct DBMS action (one for each) appears immediately before the RecordList action.

?? ElselfDetail

```
<@ARG _function> = detail
```

This section appears only if detail columns are specified. If no RecordList or Form section exists, this is an If action named IfDetail.

 RecordDetail

?? ElselfUpdate

```
<@ARG _function> = update
```

This section appears only if updatable detail columns are specified.

 Update

 UpdateResponse

For information about the update response, See “Header, Footer, Update Response, and Delete Response HTML” on page 202.

?? ElselfDelete

```
<@ARG _function> = delete
```

This section appears only if the **Delete** option is specified for the record detail Web page.

 Delete

 DeleteResponse

For information about the delete response, See “Header, Footer, Update Response, and Delete Response HTML” on page 202.

↓ ElseError

 Invalid Function

The HTML for this action displays the following message:

```
Error: Invalid Function
```

```
An unknown function was specified.
```

 Return

This action ends execution of the application file and returns the accumulated Results HTML to the Web browser.

HTML Snippets The Snippets Workspace contains a snippets folder named **Builder Snippets**, and a subfolder named **Search**. The **Search** folder contains snippets for the Form Header, Form Footer, Record List Header, Next/Previous Buttons, Record List Footer, No Matches, Record Detail Header, Record Detail Footer, Update Response, and Delete Response. The Search Builder uses these snippets in the designated places as default values for the named attributes. To change the default values, you can edit these snippets.

Configuring the New Record Builder



Witango New Record Builder Options and Setup

The *New Record Builder* builds a series of actions that allows users to add a record to a database, on the new record entry form in their Web browser. For the new record, you specify the database columns the user can add data to and the response message to return after the record is added. Witango does the rest.

The topics covered in this chapter include:

- setting new record column options
- formatting the new record entry form
- customizing your form and creating response messages
- a summary of how to set column options.

About the New Record Builder

You use the New Record Builder to build actions that, when Witango Server executes them, display a form allowing users to enter data for a new record and return a response message after the record is added to the database.

The following is an example of a new record entry form, which appears in the users Web browser:



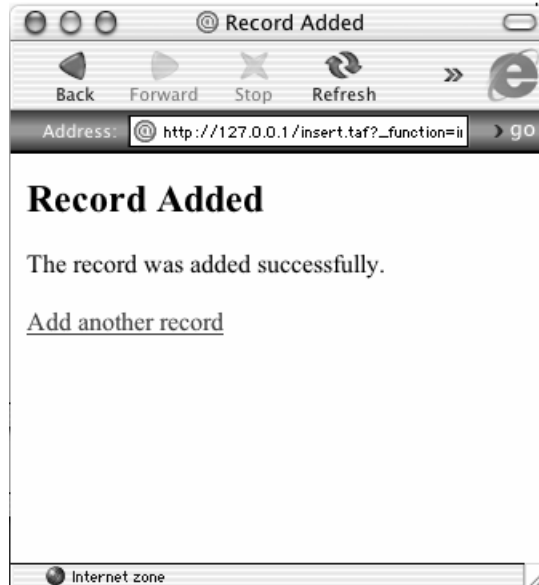
The screenshot shows a web browser window with the title "New Record". The address bar displays "http://127.0.0.1/insert.taf". The browser's navigation bar includes buttons for Back, Forward, Stop, Refresh, and a "go" button. The main content area contains a form with the following fields and controls:

- Product key:
- Title:
- Price:
- Sale price:
- Onsale: ☐ True ☒ False
- Type:
- Category:
- Supplier:
- Artist:
- Buttons:

The status bar at the bottom indicates the "Internet zone".

The user enters the values for the columns in the new record, and clicks **Save** to save the record. Witango Server saves the record to the data

source and returns the HTML response you specified in the New Record Response HTML.



Main Steps to Use the New Record Builder

To use the New Record Builder, fill out the contents of the New Record Builder window. For more information, see “Setting New Record Options” on page 212.

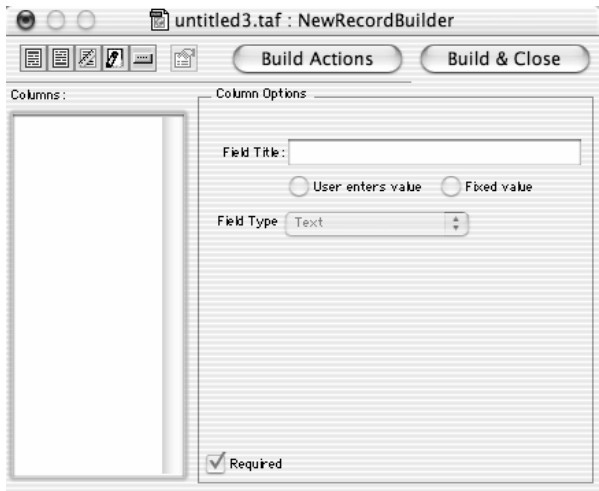
Then, format the new record entry form for display in the Web browser and customize the response messages related to this form. For more information, see “Formatting the New Record Entry Form” on page 217 and Customizing Your Form and Response Messages on page 218.

Setting New Record Options



New Record Builder

When you drag the New Record Builder icon from the Actions bar into an application file, the New Record Builder window appears:



All the options necessary for configuring the New Record Builder appear in its options window.



Tip You can save your Witango builder page formats to use for new tables you create in the builders. For more information, see “Page Format Table Settings” on page 167.

Columns

The columns you include in this list are the columns the user assigns values to in the new record. To add columns to the list, drag them from the Data Sources Workspace. Columns appear in the format `table_name.column_name`. You can only add columns from one table. The order in which the columns appear in the **Columns** list determines their order on the resulting new record entry form.

The following table describes the operations you can perform on columns:

To...	Do This...
Reorder columns	Select the columns and drag them to a different location in the list.

To...	Do This...
Delete columns	Select the columns. Choose Delete from the Edit menu, press the DELETE key on the keyboard, select the Delete icon on the main toolbar, or right-click and choose Delete from the context-sensitive menu that appears.
Delete columns without confirmation	Hold down the CTRL key while using the Delete command.

Columns Options

Use the **Column Options** area to configure each column appearing in the **Columns** list. You can specify how each column's entry field appears on the new record entry form and whether a value is required for it or not.

Lets the user enter the value for the new record.

The type of value entry field to appear on the new record entry form.

Prevents the user from adding the new record without specifying a value for this column.

This title appears for this column on the new record entry Web form.

Specify the value yourself and prevent this column from appearing on the new record entry form.

Column Options

Field Title:

☐ User enters value ☐ Fixed value

Field Type:

☒ Required

Field Title

In this field, set the title of the value entry field for the column as you want it to appear on the new record entry form.

User Enters Value

Select this option if you want the user to enter a value in an entry field on the new record entry form.

Field Type

To select a value entry field type for a column, select the column in the **Columns** list and select an item from the **Field Type** drop-down menu (**Text**, Drop-down List, List Box, Check Box, or **Radio Buttons**).

A Field Properties dialog box appears, allowing you to specify properties for the field.



Field Properties

For details on the different types of field that can appear here (text, drop-down list, list box, check box, or radio buttons), see **Field Type** on page 180.

You can edit a field's properties at any time by clicking the **Field Properties** icon in the New Record Builder window or by choosing **Field Properties** from the **Attributes** menu.

The Field Properties dialog box for the specified type of field appears. For example, in the case of a text field:

Fixed Value

If you select this option, no entry field appears on the new record entry form. The value you specify is used for every new record.

Using the **Value** drop-down menu, select one of the following options for a fixed value:

- **Value Entered.** Use the text box provided to enter the value for the **Field Title**.
- **SQL Expression.** The value returned by the SQL expression text entered is used as the value. The text entered is evaluated by the database, and the result is used as the column value in the new record.



Note For ODBC data sources, you can enter ODBC scalar functions here.

- **SQL Statement.** The SQL statement entered is executed, the results are retrieved, and the first data item of the results is used as the column value in the new record. For example, if you enter:

```
SELECT (MAX (cust_num)+1) FROM customer
```

the largest customer number plus one is used as the value for the column in the new record.

- **Current Timestamp.** The current timestamp (date and time combined) on the Witango Server computer is used as the value.
- **Current Date.** The current date on the Witango Server computer is used as the value.
- **Current Time.** The current time on the Witango Server computer is used as the value.
- **CGI Parameters.** The rest of the fixed value options are referred to collectively as CGI parameters. They include **Client Name**, **Client Domain**, **Client IP Address**, **Client Browser**, **Server Address**, **Server Port**, **Referer Page URL**, and Method. When you specify one of these parameters as the column value, the parameter value passed in when the user clicks the **Save** button is used.

Required

Select this column option to prevent a record from being added, unless the user enters a value. If the user tries to leave the value field empty, an error message like the following is returned.



If you select **Fixed Value** for the column, the **Required** option is not available. It is also not available for columns configured to use the **Check Box** field type. This is because a check box can have only two values: empty and the one you specify in the Field Properties dialog box. Making it required would mean the record could not be added unless the user checks the checkbox, in which case you could use the **Fixed Value** option for the column and have it not appear on the new record entry form.

Summary: Setting Column Options

The following table summarizes how to set column options for the New Record Builder:

To ...	Do This ...
Let the user specify the value for a column	Select the column in the Columns list. Select the User Enters Value option. This option defines the column as a user-enterable field, and a value entry field appears on the new record entry form.
Specify the title to appear for a column's new record form value entry field	Select the column in the Columns list. The column's name appears in the Field Title field. Replace the text with the desired field title. Witango remembers the entered title and uses it as the default the next time you choose it.
Specify the type of value entry field you want to display for a column	Select the column in the Columns list. Make sure the User Enters Value option is selected. From the Field Type drop-down menu, select the type of field you want to display. A dialog box appears, allowing you to specify the field properties.
Prevent the user from omitting a column value	Select the column in the Columns list. Make sure the User Enters Value is selected. Select the Required option. If the user leaves the field empty and tries to save the record, an error message appears, explaining the problem.
Hard-code the value for a column	Select the column in the Columns list. Select the Fixed Value option. From the Value drop-down menu, select a value to use for the new record. You can use one of the preset values, such as Current Date or Current Time , or select Value Entered to enter a value yourself. Select one of the SQL options to get a value from the data source. Columns specified as Fixed Value do not appear on the new record entry form.

Formatting the New Record Entry Form

Use the format options to determine the layout of the entry fields and their titles in the Web browser.

To change the page format options of the new record entry form

Do one of the following:

- In the New Record Builder window, click the **Page Format** icon.
- From the **Attributes** menu, choose **Page Format**.

The Page Format dialog box appears. This dialog box is identical to the one for the Search page of the Search Builder. See page 186 for details.



Page Format

Customizing Your Form and Response Messages

The Snippets Workspace includes the default builder HTML snippets described in this section. They include snippets for Form Header, Form Footer, and New Record Response.

Header, Footer, and New Record Response HTML

You use Header HTML and Footer HTML to customize the new record entry form by specifying HTML that is placed above and below the entry form.

The New Record Response HTML is returned after the user saves the new record.

The Column Snippets folder in the Snippets Workspace contains the names of all the columns in the table being inserted into. Dragging a column name to the HTML editing field causes an `<@COLUMN>` tag to be added to the HTML. When the application file is executed, the column's value in the new record is included at that location in the response.

To display values from the new record, Witango must be able to get those values from one of three places: the new record form submitted by the user, a fixed value you have specified, or from the database. If you include a column in the result message that does not appear in the **Columns** list of the New Record Builder window, Witango must do a search of the database to retrieve the new record after it is added.

To do so, Witango must have the value of the record's primary key column(s). This means the primary key column(s) must appear in the **Columns** list of the builder. Without this information, Witango Studio does not permit you to build the New Record actions.

To enter Header HTML, Footer HTML, or New Record Response HTML

I Do one of the following:

- In the New Record Builder window, click the **Header HTML**, **Footer HTML** or **New Record Response HTML** icon.
- From the **Attributes** menu, choose **Header HTML**, **Footer HTML**, or **New Record Response HTML**.



Header
HTML



Footer
HTML



New Record
Response HTML

The corresponding HTML editing window appears:



- 2 If you want to include HTML other than the default snippets, enter it.
- 3 Close the editing window.

Changing Button Titles

The new record entry form contains two buttons at the bottom of your form:

- The **Save** button saves the record.
- The **Reset Values** button resets the entry fields to their default values.

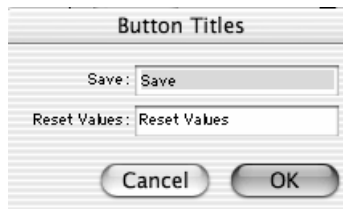
To change button titles

- I Do one of the following:
 - In the New Record Builder window, click the **Button Titles** icon.
 - From the **Attributes** menu, choose **Button Titles**.



Button Titles

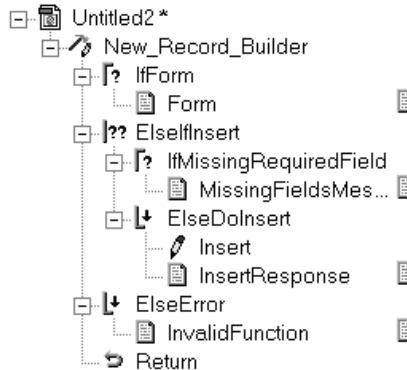
The Button Titles dialog box appears:



- 2 Enter new titles in the corresponding fields.
- 3 Click **OK**.

Actions Built by the New Record Builder

The actions built by the New Record Builder appear in the application file window as follows:



For more information, see “Building the Actions” on page 169

The following shows the actions resulting from the New Record Builder process and the conditions under which the actions are built:

IfForm

```
<@ARG _function> = nrform
```

This section appears only if one or more user-enterable fields exist.

Form

ElseIfInsert

```
<@ARG _function> = insert
```

If no form is present, this is an If action named IfInsert.

IfMissingRequiredFields

This action contains one criterion for each required field, to check if `<@ARG fieldName>` is empty. All the criteria are connected with OR operators.

If there are no required fields, this If/Else condition does not exist.



MissingFieldsMessage

If the user leaves out values for required fields, the HTML for this action displays the following message:

```
Error: Missing Required Fields
```

```
The record could not be added because the
following required fields were left empty:
```

```
...
```

```
Please go back and enter values for these
fields.
```



ElseDoInsert



Insert

If you selected **SQL Statement** for any column value, a Direct DBMS action (one for each) appears immediately before the Insert action.



InsertResponse

For information about the new record entry response, See “Header, Footer, and New Record Response HTML” on page 218.



ElseError



Invalid Function

The HTML for this action displays the following message:

```
Error: Invalid Function
```

```
An unknown function was specified.
```



Return

This action ends execution of the application file and returns the accumulated Results HTML to the Web browser.

HTML Snippets The Snippets Workspace contains a snippets folder named **Builder Snippets**, and a subfolder named **New Record**. The New Record folder contains snippets for the Form Header, Form Footer, and New Record Response.

The New Record Builder uses these snippets in the designated places as default values for the named attributes.

SECTION IV

Witango Actions

How to Use Witango Actions

This section gives details on many of the actions that can be used in Witango. Actions not discussed in this chapter are discussed in the Section V, “Witango Objects.”

This section contains chapters on the following topics:

- Chapter 12, Working With Actions on page 229
- Chapter 13, Grouping Actions on page 245
- Chapter 14, Using Basic Database Actions on page 251
- Chapter 15, Using Control Actions on page 273
- Chapter 16, Extending Witango Functionality on page 297
- Chapter 17, Sending Electronic Mail From Witango on page 309
- Chapter 18, Reading, Writing, and Deleting Files on page 317
- Chapter 19, Using Advanced Database Actions on page 325.

Chapter 12, which gives a general overview of Witango actions, is recommended for new users of Witango. Whenever you need to know about another action type, you can read the relevant chapter at that time.

Previous users of Witango may wish to read about the Presentation action, in Chapter 12, and about revisions to the Mail functionality in Chapter 17.

Using Actions

The Basics of Using Witango Actions

A Witango application file is made up of a series of one or more *actions*. Each action performs a specific type of function and can have results, usually in the form of HTML, associated with it. The applications you create may be used to input data to information systems, compose and display information from data sources, and many more interactions.

When an application file is called, the actions in it are executed by Witango Server. When execution is complete, the HTML results are returned to the user's Web browser. These results can be from the user or from interaction with other servers, normally DBMSs.

Several actions allow you to search, add, update, and delete database records. There are also actions for executing manually-entered database statements and controlling the flow of execution within an application file. You can also automatically create a sequence of actions using the builders.

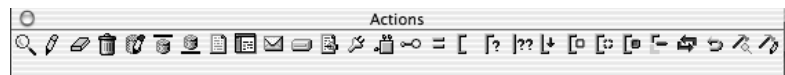
This chapter covers the following topics:

- the Actions bar
- working with actions
- assigning attributes to actions
- the Results action
- the Presentation action.

I Witango does not restrict its content to only HTML format. Using other markup languages such as SGML, VRML, and XML instead of HTML is also possible.

About Actions

The Actions bar shows all the available action types. It appears whenever an application file is active, or you choose **Actions Bar** from the View menu.

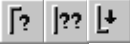


Tip You can drag the Actions bar to anywhere on your desktop and resize it.

You add all Witango actions to an application file from the Actions bar.

The following table lists each action, its function, and where in this *User's Guide* you can find more information:

Icon	Action	Function	User's Guide Reference
	Select	Selects actions in the application file window.	This chapter.
	Search	Retrieves records from a database.	Searching a Database on page 252
	Insert	Adds records to a database.	Adding Records to a Database on page 266
	Update	Changes records in a database.	Modifying a Database Record on page 268
	Delete	Removes records from a database.	Removing a Database Record on page 270
	Direct DBMS	Executes SQL statements.	Using SQL Directly on page 330
	Begin Transaction, End Transaction	Begins a transaction and ends a transaction with a rollback or commit.	Using Database Transactions on page 326

Icon	Action	Function	User's Guide Reference
	Results	Performs no special functions of its own, but it lets you append HTML to the results.	Results HTML on page 237
	Presentation	Allows you to reference presentation pages (external files) in your Witango application file.	PresentationAction on page 243
	Mail	Sends out electronic mail.	Sending Electronic Mail From Witango on page 309
	File	Reads, writes, and deletes files on the Witango Server machine.	Reading, Writing, and Deleting Files on page 317
	Script	Allows you to specify server-side JavaScript code to execute.	ExecutingJavaScript on page 298
	External	Calls an external code module to perform a function and return results.	Using an External Action on page 301
	Create Object Instance	Creates object instances from COM, JavaBean, and Witango class file objects.	Adding a Create Object Instance Action on page 377
	Call Method	Calls methods on the object instances that are created.	Adding a Call Method Action on page 383
	Assign	Makes specified value assignments.	Assigning Variables With the Assign Action on page 156
	Group	Groups related actions.	Grouping Actions on page 245
	If, Else If, Else	Executes an expression and, based on the result of the expression, affects the control flow in the application file.	General Forms of ConditionalActions on page 280
	While Loop, For Loop	Repeats a set of contained actions until an expression evaluates to true or for a specified number of times.	Repeating a Set of Actions (Loop Actions) on page 288
	Objects Loop	Loops over collection objects.	Using the Objects Loop Action on page 391

Icon	Action	Function	User's Guide Reference
	Break	Terminates processing in a loop.	Exiting a Loop (Break Action) on page 294
	Branch	Causes a jump to another action or action group.	Jumping to a Designated Action (Branch Action) on page 274
	Return	Ends execution of the application file and returns the accumulated Results HTML to the Web browser.	Ending File Processing (Return Action) on page 295

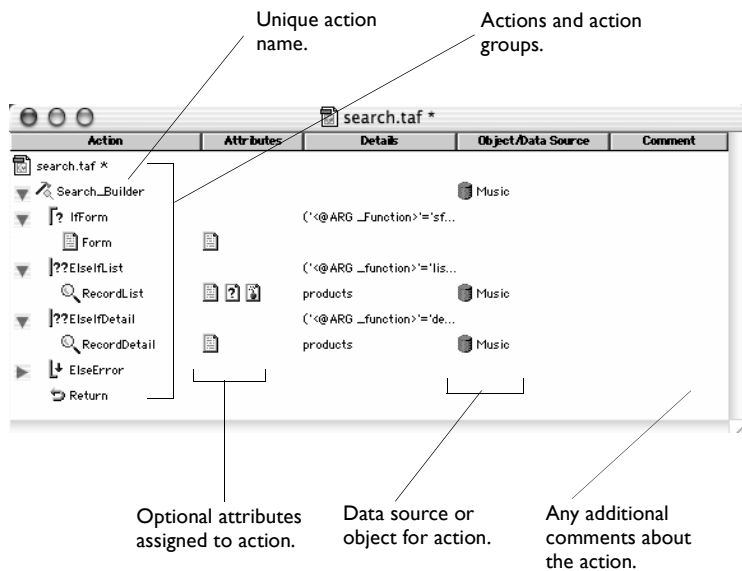
As well as actions, the Actions bar includes icons for the Search Builder and the New Record Builder. You add the builders to an application file in exactly the same way you add actions.

Icon	Builder	Function	User's Guide Reference
	Search Builder	Builds the actions required to perform a search.	Configuring the Search Builder on page 171
	New Record Builder	Builds the actions required to add a new record.	Configuring the New Record Builder on page 209

Working With Actions

The application file window shows the actions that you want Witango Server to execute. Generally speaking, Witango Server executes actions sequentially, from top to bottom, until it encounters a control action. Control actions make decisions and cause execution to jump to another action or action group.

The following is an example of the application file window:



An action icon in the **Action** column indicates the type of action. Each action must have a name that is unique in the application file.

An action can have attributes. Action attribute icons in the **Attributes** column indicate which attributes are associated with the action on that row.

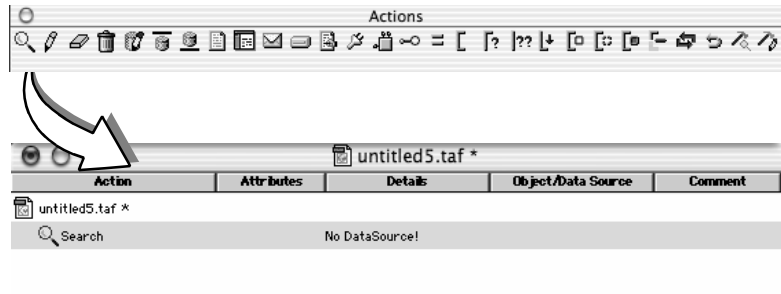
Some actions require database operations. The **Object/Data Source** column indicates which data source an action is associated with.

Adding an Action

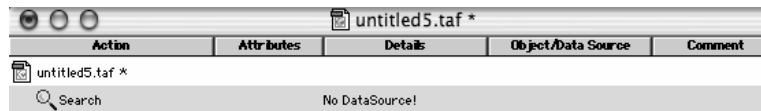
To add an action to an application file

Do one of the following:

- Drag an action icon from the Actions bar into the application file window (the cursor changes to include crosshairs and the action icon you are adding), and drop it where you want to add the action.



- Click an action icon, move the cursor into the application file window (the cursor changes to crosshairs), and click where you want to add the action.



In either method, a gray line indicates where the new action is to be placed.

If the action has an editing window, it opens automatically.



Tip To prevent the action's editing window from being opened automatically, hold down the CTRL key while dragging the new action into the document window.

Naming an Action

Each action in an application file must have a unique name. Witango Studio gives actions a unique name automatically.

The default name for an action is its action type. When you add an action that already exists in the application file with its default name, Witango appends the default name with a numeric starting at "1"; for example, "Search 1".



Tip To make your application files more readable, you should always replace default action names with more meaningful ones.

To rename an action in an application file

- 1 Select the action you want to rename.
- 2 Do one of the following:
 - Click the name of the action.
 - From the **Edit** menu, choose **Rename**.
 - Right-click the selected action and choose **Rename** from the context-sensitive menu that appears.
- 3 Type the new name.



Note Action names can contain only letters, numbers, and underscores. No spaces, punctuation, or other characters are allowed. Adding spaces automatically adds underscores.

When you rename an action, Witango automatically updates any Branch actions in the same application file referring to the action. If you rename an action that is the destination for branches from other application files, the Branch actions in other application files are *not* updated.

Witango does *NOT* automatically update action results references for renamed actions.

Deleting an Action

**To delete an action from an application file**

- 1 Select the action you want to delete.
- 2 Do one of the following:
 - From the **Edit** menu, choose Delete.
 - On the main toolbar, click the Delete icon.
 - Press DELETE.
 - Right-click and choose Delete from the context-sensitive menu that appears.
- 3 When the dialog box appears, asking you to confirm the deletion, click **OK**.



Tip You can bypass the confirmation dialog box by holding down the **Ctrl** key when choosing **Delete**.

Editing an Action

All of the actions—except Return, Group, and Break actions—have associated attributes and parameters. You can set these parameters in the action's editing window.

To edit an action in an application file

- Double-click the action icon in the application file window.

The action's editing window opens.

If the action is associated with a data source, the Data Sources Workspace opens, listing the tables and columns for the data source. If Witango Studio has not loaded the data source yet, it is loaded first.

Moving an Action

Witango executes the actions in an application file sequentially, from top to bottom; however, you can use control actions to modify this sequence.

If you want the actions to be performed in a different order, you can rearrange them. Move them to another location in the application file by dragging them to the position you want.

To move an action to a new location

Do one of the following:

- Select the action you want to move, and drag the action to its new position.
- Select the action, and cut and paste it using the edit commands.

Actions are pasted after the currently selected action, or at the end of the file if no action is selected.

Edit commands are available from the Witango Studio **Edit** menu, from the main toolbar, and from the context-sensitive menu.

When you move an action, Branch actions referring to it continue to branch to the action, even though its position has changed.

Copying an Action

You may want to create an action that performs a task similar to one performed by an existing action in another application file. Instead of having to recreate the action and specify all its parameters again, Witango Studio allows you to duplicate an action.

To copy an action in the same application file

Do one of the following:

- Select the action you want to copy, hold down the **Ctrl** key, and drag the action to where you want the new action to appear.

- Select the action, and copy and paste it using the edit commands.

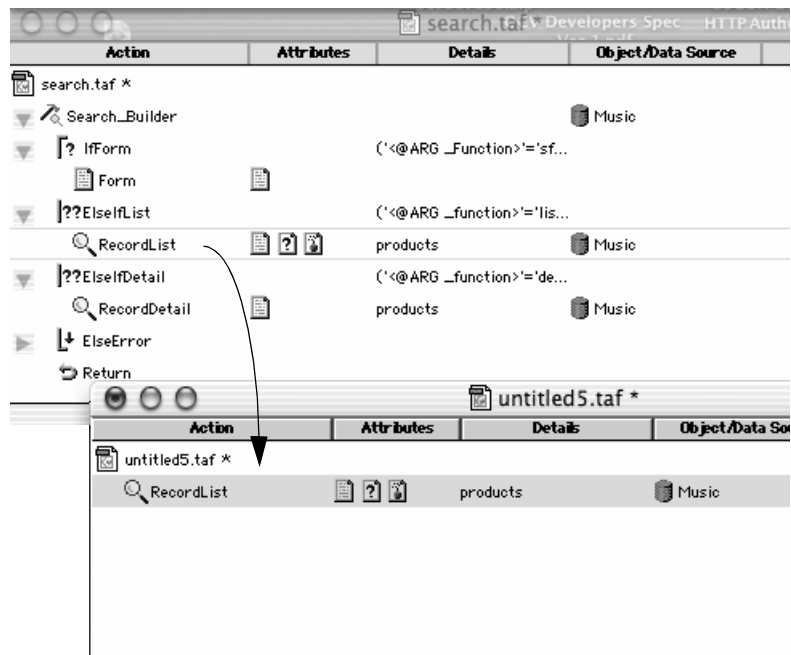
Edit commands are available from the Witango Studio **Edit** menu, from the main toolbar, and from the context-sensitive menu.

The copied action is given a new, unique name, which you should change to a more descriptive name.

To copy an action into another application file

Do one of the following:

- Select the action you want to copy, and drag the action into another application file.



- Select the action, and copy and paste it using the edit commands.

Edit commands are available from the Witango Studio **Edit** menu, from the main toolbar, and from the context-sensitive menu.

Be careful when copying database actions. For an action to work correctly in the new application file, the data source must be the same as in the original one.

Alternatively, you may assign another data source to the action in the new application file.

Context-Sensitive Action Menu

When you right-click an action icon in the application file window, or anywhere in the file window with an action selected, a context-sensitive menu of action commands appears:



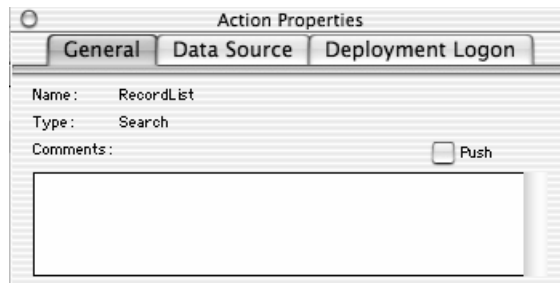
For more information on using these commands, see [Setting Data Sources for Actions](#) on page 106, [Assigning Attributes to Actions](#) on page 236, [Debugging Files](#) on page 61, [The SQL Query Window](#) on page 22, [Grouping Actions](#) on page 245, and [Action Properties](#) on page 234.

- **Open** opens the action editing window for the selected action.
- **Cut, Copy, Paste** and **Delete** perform the standard window editing functions.
- **Rename** allows you to edit the current name of the action.
- **Set Data Source** allows you to set the data source for one or more actions.
- **Results HTML, No Results HTML, Error HTML, and Push** are attributes you can assign to actions which support them.
- **Debug File** is an attribute of the entire application file or Witango class file.
- **SQL Query** opens the SQL Query window so you can perform SQL queries from within Witango.
- **Group** and **Ungroup** allows you to group related actions and also to ungroup them.
- **Properties** displays the action properties window.

Action Properties

When you select an action and choose **Properties** from either the View menu or the context-sensitive menu, the Action Properties window for that action appears.

This window displays current information about the selected action and the assigned data source.



For more information, see
"Properties Window" on
page 8.

Using this window, you can change some of the action's properties.

Assigning Attributes to Actions

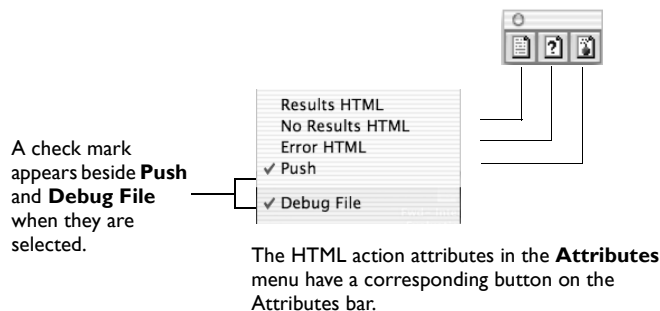
In addition to the parameters specific to each action type, which are edited using the action's editing window, actions can also have the following attributes:

- **Results HTML** applies to all actions, except control actions (other than Branch). After the action is executed, this HTML is added to the results returned.
- **No Results HTML** applies only to Search, Direct DBMS, Script, File, and External actions. When the action does not return data, this HTML is returned instead of the Results HTML.
- **Error HTML** applies to most action types except certain control actions (including Return and Break). In the event of an error in the action's execution, this HTML is returned immediately.
- **Push** causes the Results HTML accumulated so far to be sent back to the Web browser when the action to which it is assigned finishes executing. Execution then continues normally.
- **Debug File** lets you see useful information about your application file or Witango class file execution in your Web browser application. This attribute applies to the entire application file, not a particular action. For more information, see *Debugging Files* on page 61.

To assign Results HTML, No Results HTML, Error HTML, or Push

Do one of the following:

- Select the action in the application file window, then select an attribute from the **Attributes** menu or from the Attributes bar.
- Right-click the action in the application file window and choose the attribute that applies to the selected action from the context-sensitive menu that appears.



For more information, see “HTML Editing Window” on page 6.

Action attribute icons appear beside the action name in the **Attributes** column of the application file window. See the example on page 229.

You can switch between the Results HTML, No Results HTML, and Error HTML associated with an action by clicking on the tabs at the bottom of the HTML editing window.

Results HTML



Many actions in an application file can have HTML associated with them. This HTML is stored in the Results HTML attribute. If Results HTML contains any text, the Results HTML icon appears in the attributes column of the application file window; otherwise, it does not.

As Witango Server executes the actions in a file, the Results HTML associated with each is accumulated. When execution of the file is complete, the HTML is returned.

Results HTML can also contain Witango *meta tags* that Witango Server processes. While all the other text in Results HTML is interpreted by the user’s Web browser and returned as is (via the Web server), Witango Server first substitutes meta tags with other values.

For more information, see “Working with Meta Tags” on page 143.

The `<@COLUMN>` meta tag causes a database value to be placed in the HTML. There are many others, including tags for referencing form field and search argument values, and conditional tags for displaying HTML only if the result of a given comparison is true.

To create or edit the Results HTML for an action

- 1 Select the action in the application file window.
- 2 Do one of the following:
 - From the **Attributes** menu, choose **Results HTML**.
 - Click the **Results HTML** icon on the Attributes bar.
 - Right-click the action and choose **Results HTML** from the context-sensitive menu that appears.

The Results HTML editing window appears:



- 3 Type the Results HTML into the HTML text area. The text can include any valid HTML or Witango meta tags.

You can switch between the Results HTML, No Results HTML, and Error HTML associated with an action by clicking on the tabs at the bottom of the HTML editing window.

For more information, see “Working With Snippets” on page 120.

You can add column values (for Search actions only) and any HTML snippets you have defined to the Results HTML editing window from the Snippets Workspace. As well, you can add from the list of standard Witango snippets that allow for easy entry of many of the meta tags.

To include any of these items in your Results HTML, select the snippet and either drag it, or copy and paste it into the desired location in your text.

For HTML snippets that have placeholders for the current selection, select the text and drag the snippet over the selected text. The snippet is wrapped around the selection. For example, “Title” becomes “<H1>Title</H1>”.

You can also easily add many of the common Witango meta tags.

-
- I Witango does not restrict its content to only HTML format. Using other markup languages such as SGML, VRML, and XML instead of HTML is also possible. If you use other content types, you are responsible for setting the HTTP header appropriately.

To add a meta tag

- 1 Click the editing area where you want to add a meta tag.
- 2 Do one of the following:
 - From the **Edit** menu, choose **Insert Meta Tag**.
 - Right-click, and choose **Insert Meta Tag** from the context-sensitive menu that appears.

The Insert Meta Tag dialog box appears.

No Results HTML



You can associate No Results HTML text with Search, Direct DBMS, Script, and External actions. If the action execution does not return any data, this text is added to the application file's accumulated HTML instead of the Results HTML. This is useful when you want to display a special message to users when their queries do not return data.



Note If both Results HTML and No Results HTML appear as attributes, Witango accumulates one or the other, but never both.

After Witango Server processes the No Results HTML, execution of the application file continues normally to the next action.

No Results HTML can contain any of the Witango meta tags used in Results HTML, except for those related to displaying result data items, such as `<@ROWS>`, `<@COLUMN>`, and `<@COL>`.

To create or edit the No Results HTML for an action

- 1 Select the appropriate action in the application file window (Search, Direct DBMS, Script, and External actions).
- 2 Do one of the following:
 - From the **Attributes** menu, select **No Results HTML**.
 - Click the **No Results HTML** icon on the Attributes bar.
 - Right-click the action and choose **No Results HTML** from the context-sensitive menu that appears.

The No Results HTML editing window appears.

- 3 Type the No Results HTML into the HTML text area. The text can include any valid HTML or Witango meta tags.

Error HTML



Error HTML allows you to specify your own error messages in HTML format, instead of having Witango Server produce them. The other alternative is to modify the `Error.htx` file; see [To specify your own custom default error message page 241](#) on this page.

You can associate Error HTML with most actions. If an action fails for any reason, execution ends and the Error HTML for the action is returned immediately to the user.

Error HTML can contain all the Witango meta tags used in Results HTML, except for those related to displaying result data items.

There are also special Witango meta tags for displaying error information.

If no Error HTML has been assigned to an action and an error occurs in that action, Witango returns a default error message using the following HTML:

```
<h3>Error</h3>

An error occurred while processing your request:<p>
<@ERRORS>
Position: <b><@ERROR PART=POSITION></b><br>
Class: <b><@ERROR PART=CLASS></b><br>
Main Error Number: <b><@ERROR PART=NUMBER1></b><br>
<@ifequal @ERROR PART=NUMBER2> 0>
<@else>
    Secondary Error Number: <b><@ERROR
PART=NUMBER2>
</b><br>
</@ifequal><p>
<i>
<@ERROR PART=MESSAGE1><br>
<@ifequal @ERROR PART=MESSAGE2> ">
<@else>
    @ERROR PART=MESSAGE2><br>
</@ifequal><p>
</i>
</@ERRORS>
```

To create or edit the Error HTML for an action

- 1 Select the action in the application file window.
- 2 Do one of the following:
 - From the **Attributes** menu, select **Error HTML**.
 - Click the **Error HTML** icon on the Attributes bar.
 - Right-click the action and choose **Error HTML** from the context-sensitive menu that appears.

The Error HTML editing window appears.

- 3 Type the Error HTML into the HTML text area. The text can include any valid HTML or Witango meta tags.

To specify your own custom default error message

- 1 Create a text file containing the desired HTML and meta tags.
- 2 Name the file `error.htx`.
- 3 Save or copy it to the following directory
`WITANGO_PATH\MiscFiles`.

If Witango Server finds this file, it processes and returns it instead of the built-in default Error HTML. Error HTML assigned to an action is used if it exists.

The name and location of this file is determined by the `defaultErrorFile` configuration variable, which can be modified using the Witango Administration Application. The values when Witango is first started are given above. If you modify the path or name of the error file, place the file in the directory you specified instead.

Push

The **Push** attribute causes the Results HTML accumulated so far to be sent back to the Web browser, when the action to which the **Push** attribute is assigned finishes executing. Execution then continues.

Normally, Witango waits until all execution is finished before returning the results at one time. If you want the user to see some of the results while Witango continues with the rest of the execution, set the **Push** attribute of the action.



Note Some Web browsers may not display table HTML immediately if you use the Push attribute to return an unclosed table.

Debug File

For more information, see [Debugging Files](#) on page 61.

Adding HTML (Results Action)



Results Action

The Results action adds HTML to an application file's results.

When you drag the Results action icon from the Actions bar into an application file, a blank HTML editing window appears.



Results HTML can contain Witango meta tags that Witango Server processes. While all the other text in Results HTML is returned as is to your Web browser (via the Web server), any meta tags are first substituted with other values by Witango Server. You can also associate Error HTML with the Results action.

Presentation Action

Uses of the Presentation Action

The main benefit of using the Presentation action is to facilitate the separation of the business logic from the presentation logic when you develop your Witango application.

Business logic involves the use of Witango actions and meta tags to access the appropriate Web pages and data sources. *Presentation logic* involves the use of HTML to display the Web pages.

Because developing the business logic and the presentation logic generally require different skill sets, setting up independent teams to work on these two areas can improve the effectiveness and efficiency of the project. Furthermore, changing the business logic—for example, accessing a different data source—often does not affect the presentation logic, or vice versa. Keeping the two areas separate simplifies the maintenance of your project.

A Presentation action in your application file points to an HTML page. It is the link between the business logic and the presentation logic of your project.

How the Presentation Action Works

The Presentation action allows you to include individual *presentation pages* in your Witango application file. The presentation page—the file the Presentation action points to—can contain HTML, Witango meta tags, or any other sort of document markup. When Witango Server executes your application file and arrives at a Presentation action, it processes the presentation page associated with the Presentation action.

The Presentation action performs an operation similar to that of including an HTML or other file in a Witango application file using the `<@INCLUDE>` meta tag.

The file referenced by the Presentation action is part of the current project, and can be opened and edited by double-clicking on the file icon within the **Presentation Pages** folder in the **Project** section of the Workspace.

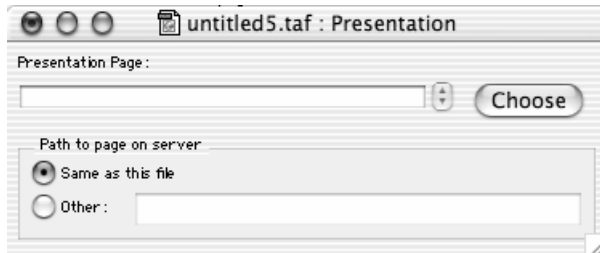
You can also designate files in your project as presentation pages, and manage files within the **Presentation Pages** folder.

For more information, see “Working With Presentation Pages” on page 77.

Setting Up a Presentation Action



When you drag the Presentation action from the Actions bar into an application file, the Presentation dialog box appears:



Do one of the following:

- In the **Presentation Page** field, enter the name of the presentation page, or if you have previously specified a presentation page in the current Project, choose a file name from the drop-down menu.
- Click Browse to navigate to the location of the presentation page.

If the file is not in your current project, you are prompted to add it to the project, where it appears in the **Presentation Pages** folder and in the **Files** folder of the **Project** tab of the Workspace.

In the **Path to target page on server** area, select **Same as** source page if the presentation page is located in the same folder as the current application file, or select **Other**.

If you choose **Other**, you specify the path to the presentation page. This value is a slash-separated path from the Web server document root, and may include literal text, meta tags, or both. To insert a meta tag in this field, right-click in the text field and choose **Insert Meta Tag...** from the context-sensitive menu that appears.

For example, you could enter the following into the text field:

```
Witango/MyDirectory/
```

This example includes the specified file residing in the `MyDirectory` folder within the `Witango` folder in the Web server document root folder.

```
<@APPFILEPATH>
```

This example includes the specified file residing in the same folder as the currently-executing application file.

Grouping Actions

Organizing Related Actions

The *Group* action allows you to organize actions within the file by grouping related actions and naming the group.

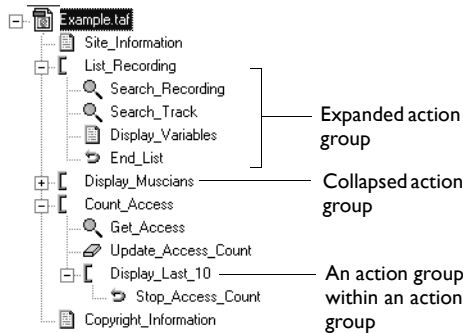
The topics in this chapter cover:

- working with action groups
- executing grouped actions.

About Grouped Actions

In large Witango application files, it is common to place a number of related actions together; however, when viewing an application file, it can be difficult to locate the related actions. To allow you to better organize actions within the file, you can group related actions and name the group. They also provide a destination for branching.

The following shows a typical application file containing action groups:



For more information, see “Application File Window” on page 56.

You can view grouped actions in a collapsed or expanded state. Groups added to an application file are in the expanded state by default. The collapsed or expanded state of an action group is saved in the application file. When the file is next opened, the last state is restored.

You can also include an action group within another action group.

You cannot associate action attributes with an action group. When you select a group, Results HTML, No Results HTML, Error HTML, and Push attributes are disabled.

Working With Action Groups

Adding an Action Group



Group

For more information, see “Naming an Action” on page 230.

To add an action group to an application file

- From the Actions bar, drag the Group icon to the location you want in the application file.

Follow the instructions in “Adding an Action to a Group” on this page to add actions.

To automatically group selected actions

- Select the actions you want to group together, and choose **Group** from the **Edit** menu or the context-sensitive menu.

A new action group containing all selected actions is created and positioned where the top-most selected action was.

Once added, the action group appears with a default name, just like other actions. You rename action groups in the same way you rename other actions. Just like action names, the action group name must be unique within the application file.

Adding an Action to a Group

To add an action to a group

Do one of the following:

- Drag an action between two actions that are already in the group.
The action is added to the group at that location.
- Drag an action onto the group icon.
The action is added to the bottom of the group.
- Use the **Copy** and **Paste** commands from the **Edit** menu, main toolbar, or context-sensitive menu to copy and paste an action into a group.



Note You can select discontinuous actions in the application file and drag, or copy and paste them into a group; the actions do not need to be together already.

Removing an Action From a Group

To remove an action from a group

Do one of the following:

- Drag the action you want to remove outside the group.

If you drag the action above or below the group, the action appears immediately before or after the group, respectively.

- Use the **Copy** and **Paste** commands from the **Edit** menu, main toolbar, or context-sensitive menu to copy and paste an action outside of a group.



Note Removing actions from a group does not delete the group, even if *all* actions are removed from the group.

Ungrouping Actions

To ungroup all actions with a group

- 1 Select the Group action.
- 2 Do one of the following:
 - From the **Edit** menu, choose **Ungroup**.
 - Right-click the Group action and choose **Ungroup** from the context-sensitive menu that appears.

This deletes the group action but keeps the actions that were within the group.

You can also drag actions out of the action group, but that does not delete the Group action itself.

Deleting an Action Group

You delete an action group and all actions within it the same way you delete any action. Deleting a group also deletes all the actions within it.

For more information, see [Deleting an Action](#) on page 231.

Effects of Editing an Action Group

Editing an action group affects the actions within it:

- Moving an action group automatically moves all actions within the group.
- Copying an action group copies all actions within the group as well.
- Deleting an action group deletes the action group and all actions within the group.

Branching to an Action Group

You can specify an action group as the destination of a Branch action.

For more information, see “Jumping to a Designated Action (Branch Action)” on page 274.

Executing Grouped Actions

When Witango Server encounters an action group during file execution, no operation is performed on the action group itself, only on the actions within the group.

For more information, see “Exiting a Loop (Break Action)” on page 294.

Even though an action group has no effect on the execution of an application file, Witango Server supports the ability to branch to an action group and the ability to break out of an action group. If a Break action is encountered within a group, the next statement to be executed is the first statement outside of the group.

Using Basic Database Actions

Search, Insert, Update, and Delete Actions

Witango includes several fundamental database actions that allow you to search (*Search* action), add (*Insert* action), modify (*Update* action), and delete (*Delete* action) database records. You do not need any knowledge of SQL to use any of these actions.

The topics covered in this chapter include setting up and executing Search, Insert, Update, and Delete actions.

Searching a Database

Search actions retrieve database records matching a given criteria.

You use the Search action editing window to define what columns are selected, the order of the data retrieved, and the criteria that determine which rows (records) are found.



Tip The SQL Query window gives you a convenient way to look at your database values. Choose **SQL Query** from the **Windows** menu or from the context-sensitive menu that appears when you right-click the Search action editing window. For more information, see “The SQL Query Window” on page 22.

You use the action’s Results HTML to format the results of the search.

Setting Up a Search Action



Search Action

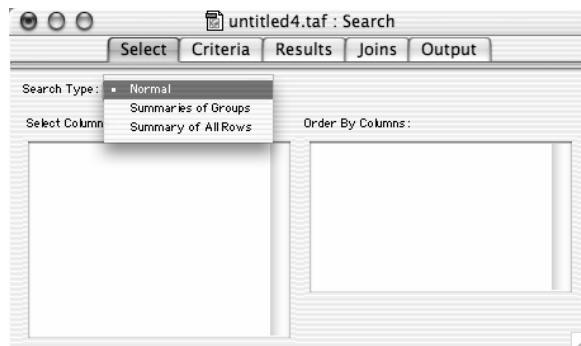
When you drag the Search action icon from the Actions bar into an application file, the Search action editing window appears. The window consists of four sections, which you can access by clicking the respective tabs. The Select, Criteria, and Results sections are covered in this chapter. The Joins section is covered in *Working With Joins* page 337 on page 337.

Select Section

You use the Select section to select the type of search to perform, the columns to retrieve, and the ordering of the records returned.

You can perform three types of search with a Search action: **Normal**, **Summaries of Groups**, and **Summary of All Rows**.

Select which type of search you want to perform from the **Select Type** drop-down menu.

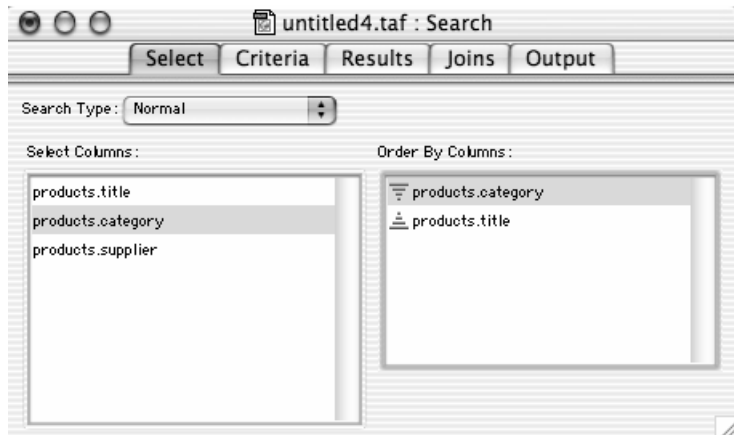


- **Normal** returns rows matching specified criteria.
- **Summaries of Groups** returns summaries of rows whose values in given columns (the grouping columns) are the same.
- **Summary of All Rows** returns a single row summarizing all rows matching your criteria. This kind of search lets you get information such as the maximum or average value of a particular column in a database table.

Normal Search

The **Normal** type of search returns rows matching specified criteria. This is the most common type of search.

When you select **Normal** from the **Select Type** drop-down menu, the Select section appears as follows:



Specify values for the parameters in the Select section:

- **Select Columns.** Drag into this list from the Data Sources Workspace the columns whose data is to be retrieved from the database.
You can include columns from multiple tables; if you do, you must define joins to describe how the tables are related.
- **Order By Columns.** Drag into this list the columns that are used to sort the results returned to the user. Ordering by columns is optional.

For more information, see “Joining Database Tables” on page 336.

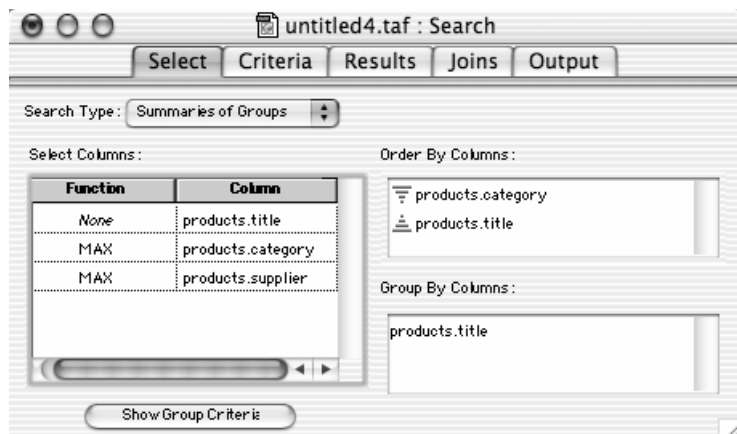
The order of the columns in the list determines the ordering hierarchy. For example, if the first order column is “state or province” and the second “customer name”, the results are first ordered by state or province; customers in the same state or province are then ordered by name.

The triangle to the left of the column name determines whether the ordering is ascending ($\triangle$) or descending (∇). To change the order direction for a column, click the triangle.

Summaries of Groups

The **Summaries of Groups** search type returns summaries for groups of rows with the same values in specified columns. For example, it allows you to find out the total sales for each sales region in an invoices table by selecting the sum of invoice amounts and grouping by sales region.

When you select **Summaries of Groups**, the Select section appears as follows:



- **Select Columns.** Drag into this list the columns you want to select. Select columns for this select type have an associated function. This function is performed on the column for all the rows in a particular group, as determined by the **Group By Columns** list. For example, if you selected the MAX function for a “price” column and group by the “classification” column, you would receive one row for each unique classification. Each row would contain the maximum value of the “price” column for the classification being summarized.

The following table lists the available functions:

Function	Description
MAX	The maximum value of column in the group.
MIN	The minimum value of column in the group.
AVG	The average value of column in the group. Valid only for numeric columns.
SUM	The sum of all column values in the group. Valid only for numeric columns
COUNT	The number of non-null values in the column for the group.
None	Perform no function; return the value of the column for each group. Columns with this option must appear in the Group By Columns list, because only group columns are sure to have the same value within a group.

To choose the function for a column, click the **Function** column and select the function from the drop-down menu.

- **Order By Columns.** As with the normal select type, you specify in this list the ordering of results. You can drag columns from the Data Sources Workspace or from the **Select Columns** list. You can order only by columns appearing in the **Select Columns** list.
- **Group By Columns.** The columns in this list determine how rows are grouped before being summarized. Groups consist of all the rows that have the same values in the columns specified. For example, if you group by the “cust_state” and “cust_rep” columns in a customer table, you get one summary row for each group of rows with the same values in the “cust_state” and “cust_rep” columns.
- **Show Group Criteria.** Normally, all the summary rows are returned for records matching the user’s criteria. You can eliminate summary rows by specifying group criteria. The group criteria have a different function from the regular criteria in that the regular criteria specify which rows are eligible for grouping, while the group criteria specify which summary rows are returned.

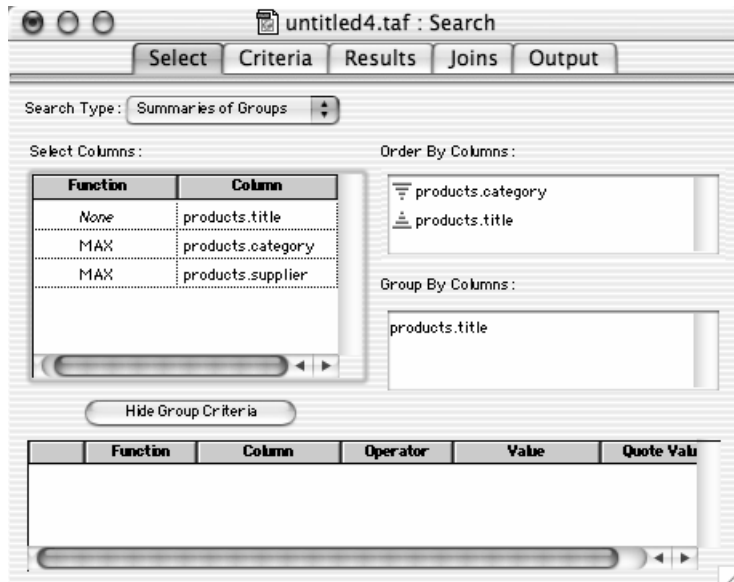


Note The group criteria section is equivalent to the HAVING clause in a SQL SELECT statement.

For example, if you are grouping by classification and selecting the maximum order amount, you can use group criteria to limit the returned rows to those customers whose maximum order amount is greater than \$5,000.

To specify group criteria, click **Show Group Criteria**.

The Select section expands to show the area for entering group criteria.



Drag columns from either the Data Sources Workspace or the **Select Columns** list.



Note You can specify **only** columns that appear in the **Select Columns** list.

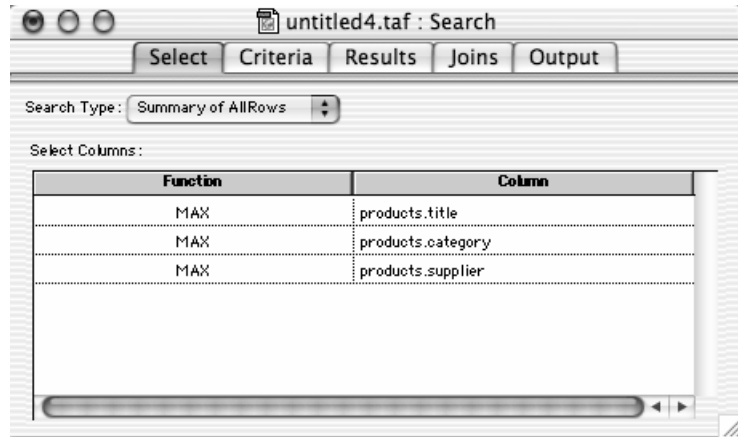
For more information, see "Criteria Section" on page 257.

Except for the function option, you specify group criteria just like normal criteria.

Summary of All Rows

To get a summary of all rows matching a specified criteria, use the **Summary of All Rows** search type. Only one summary row is

returned. For example, you could use this search type to find the average amount of all orders in an orders table.



As with the **Summaries of Groups** search type, each select column has an associated function that determines how that column is summarized. All the column values in the rows matching the criteria are aggregated using the specified function.

Criteria Section

The Search action criteria determine which rows from the database are returned by the action. If no criteria are specified, all rows are returned; otherwise, each row in the database is compared to your criteria and only those meeting them are returned.



To specify the criteria, drag columns from the Data Sources Workspace to the Criteria list. For each column, you must specify:

- Logical Operator
- Column
- Operator
- Value
- Include Empty
- Quote Value.

Column

In the **Column** field, specify the column whose value you want to compare. Drag the column from the Data Sources Workspace.

Logical Operator

The first field in the criteria list is the logical operator, which is either **and** or **or**. To specify the operator, do one of the following:

- Click the row, then click the field to display a drop-down menu and choose an operator.
- Right-click the field, and choose **Edit** from the context-sensitive menu. Then choose an operator from the drop-down menu.



Note You must specify at least two columns before the logical operators are available.

The logical operator determines whether the current and previous criteria must be true for a record to be included in the result or whether a match on either the current or previous criterion causes a record to be included in the result. For example, if your criteria are:

```

           cust_num      =          5100
and    cust_name      Begins with  A

```

only records matching both criteria are returned. If the logical operator is changed to **or**, records matching either one of the criteria are returned.

There is an implied order of operation for logical operators. Criteria joined with the **and** logical operator are evaluated before those joined with the **or** logical operator.

For example, in the following criteria:

```

           cust_num      =          5100
or    cust_name      Begins with  A
and    cust_state      =          NY

```

For more information about inserting meta tags in entry fields, see Inserting Meta Tags on page 148.

For more information, see “Criteria Separators” on page 262.

a match is made if both the second and third criteria are true or the first criterion is true.

You can also use the **Insert Meta Tag** command to enter in the Criteria window entry fields many of the commonly used meta tags.

To insert a meta tag, either click the field and choose **Insert Meta Tag** from the **Edit** menu, or right-click the field and choose **Insert Meta Tag** from the context-sensitive menu that appears.

You can use criteria separators to control the order of criteria evaluation, regardless of this default logical operator hierarchy.

Operator

In the operator field (**Oper.**), specify the operator to use when comparing the field by doing one of the following:

- Click the row, then click the field to display a drop-down menu to choose an operator from.
- Right-click the field, and choose **Edit** from the context-sensitive menu. Then choose an operator from the drop-down menu.

Possible operators include:

Operator	Meaning
=	is equal to
!=	is not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
Contains	field contains these character(s)
Begins with	field begins with these character(s)
Ends with	field ends with these character(s)
Is In	matches one of a list of values (see page 261)
Is Null	matches an empty field
Is Not Null	matches a non-empty field

Text columns permit the use of any operator; for other columns, the **Contains**, **Begins with**, and **Ends with** operators are disabled.

You can either specify a static operator or insert a meta tag to get the value at execution time. Using a variable operator allows you, for example, to put a drop-down menu on your Web page to let users choose the comparison operator.

When using a variable to specify the criterion operator, Witango requires you to use special values to represent each of the operators. The following table lists these special values:

To Specify This Operator	Use This Value
=	iseq
!=	isnt
>	gthn
<	lthn
>=	gteq
<=	lteq
Contains	cont
Begins with	swth
Ends with	ewth
Is In	isin
Is Null	inul
Is Not Null	nnul

For example, to create an operator drop-down menu in an HTML form whose value you want to use as the operator in a search criterion, you could use HTML similar to the following:

```
<SELECT NAME="cust_name_op" SIZE=1>
<OPTION VALUE = "iseq" SELECTED>=
<OPTION VALUE = "isnt">!=
<OPTION VALUE = "gthn">&gt;
<OPTION VALUE = "lthn">&lt;
<OPTION VALUE = "gteq">&gt;=
<OPTION VALUE = "lteq">&lt;=
<OPTION VALUE = "swth">Begins With
<OPTION VALUE = "ewth">Ends With
<OPTION VALUE = "cont">Contains
</SELECT>
```

and set the operator in the Search action to

```
<@ARG cust_name_op>
```


The **Is In** operator needs some additional explanation. It matches records where a column value is in a list of values.

For example, the following criteria:

cust_num	Is in	200, 300, 400

matches records in which the `cust_num` field has a value of 200, 300, or 400. The **Is in** operator can be thought of as a shortcut for a series of **OR** equals criteria:

	cust_num	=	200
or	cust_num	=	300
or	cust_num	=	400

The value specified can be a single-column or single-row array (as would be returned by the `<@ARG>` tag with a `type` attribute of `ARRAY`, for example) or a comma-separated list of values.

Value

In the **Value** field, enter the value to use in the comparison.

The value can also contain any value-returning Witango meta tags, which are substituted when the application file is executed. Use the **Insert Meta Tag** command to enter many of the commonly used meta tags.

For more information about inserting meta tags in entry fields, see [Inserting Meta Tags](#) on page 148.

Include Empty

In the **Incl. Empty** field, specify whether the criterion is included, even if the comparison value is empty, by doing one of the following:

- Click the row, then click the field to display a drop-down menu to choose a value from.
- Right-click the field, and choose **Edit** from the context-sensitive menu. Then choose an operator from the drop-down menu.

The values appear as **false** and **true**.

false omits the criterion if the value (after meta tag substitution) is empty; **true** includes the criterion regardless of the value's contents.

This option is used mainly for columns whose search value is taken from a search form on a Web page. For example, you may have a search form that allows you to enter search values for several columns, but you want the search done only on the columns you enter values for. To do this, set the **Incl. Empty** option for each of the corresponding Search action criteria to **false**.

There are cases where you do want a criterion included, even if the value is empty. For example, suppose you have a Web page that asks for a user name and password, and a corresponding Search action that finds the

For more information about inserting meta tags in entry fields, see Inserting Meta Tags on page 148.

user in a users' database. In the Search action, you probably want to set the **Incl. Empty** option for each of the values to **true**. If you do not, and the user leaves both fields empty, the Search action omits both criteria and returns all user records.

You can right-click the **Incl. Empty** field, and choose **Insert Meta Tag** from the context-sensitive menu that appears to enter many of the commonly used meta tags.

Quote Value

In the **Quote Value** field, specify whether Witango puts quotation mark characters around the value in the SQL it generates for this criterion by doing one of the following:

- Click the row, then click the field to display a drop-down menu to choose a value from.
- Right-click the field, and choose **Edit** from the context-sensitive menu. Then choose an operator from the drop-down menu.

The values appear as **false** and **true**.

For text, date, time, and timestamp columns, you should set this option to **true**. For date, time, and timestamp columns, this option has special meaning. **true** converts the specified value from the default Witango format to the format required by the database server; **false** passes the value specified as is to the database server.

If you want to use an expression that the database server evaluates (instead of a literal Witango-supplied value), set the **Quote Value** option to **false** and enter the expression in the **Value** field.

For numeric and Boolean types, you should set the **Quote Value** option to **false**.

For more information about inserting meta tags in entry fields, see Inserting Meta Tags on page 148.

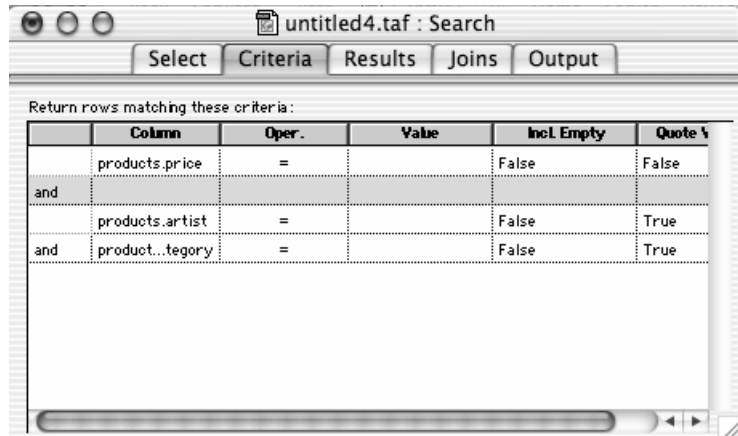
You can right-click the **Quote Value** field, and choose **Insert Meta Tag** from the context-sensitive menu that appears to enter many of the commonly used meta tags.

Criteria Separators

To group criteria, select the position between the criteria you want to group and do one of the following:

- From the **Edit** menu, choose **Insert Criteria Separator**.
- Right-click and choose **Insert Criteria Separator** from the context-sensitive menu that appears.

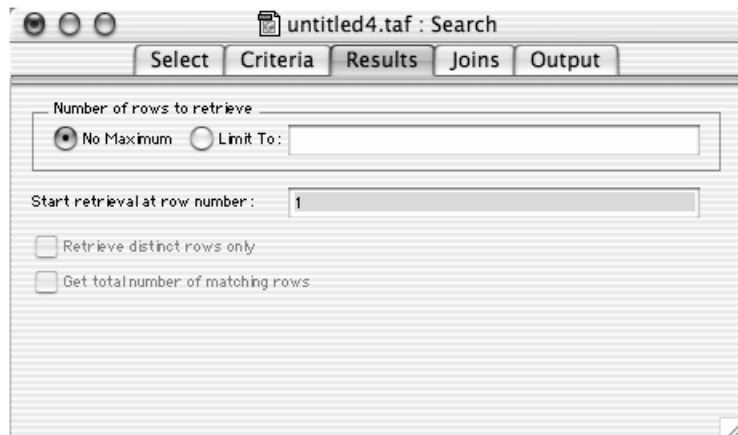
Only the logical operator cell can be edited for separator items.



Upon execution, the criteria before the separator are combined with the criteria after the separator using the logical operator specified in the separator line in the criteria list.

Results Section

In the Results section, you specify the maximum number of records to retrieve from the data source, at which result record number retrieval begins, and whether Witango gets the count of matching records.



Number of rows to retrieve

To return all matching records, select **No Maximum**.

To limit how many records you want the search to return, select **Limit To** and enter the maximum number of records to retrieve.

The following options are only available for the **Normal** search type.

Start retrieval at row number

Select this option if you want to skip some of the matching records. Specify the row number you want the Search action to start retrieval at. The default is “1”. When the value is other than “1”, the Search action returns records starting at that number, skipping any records before it.

This option is most useful when you use a variable (such as `<@SEARCHARG start>`) for the starting record number.

For more information, see “Show Multiple Pages If Limit Exceeded” on page 193.

For an example of how to use this option to provide results paging for large result sets, look at the Search action in a Search Builder-generated file created with the **Show Multiple Pages If Limit Exceeded** option selected.

Retrieve distinct rows only

If you select this option, Witango Server adds the `DISTINCT` keyword after the `SELECT` keyword in the generated SQL. The `DISTINCT` keyword specifies whether duplicate rows are to be eliminated from the result set. For example,

```
SELECT c1.cust_state, c1.cust_zip
FROM customer c1;
```

becomes

```
SELECT DISTINCT c1.cust_state, c1.cust_zip
FROM customer c1;
```

Get total number of matching rows

You use this option to retrieve the number of records matching the search criteria, irrespective of how many records are actually retrieved.

Using this option, you can access the value in the Search action’s Results HTML by using the `<@TOTALROWS>` meta tag.



Note Selecting this option involves an extra database operation, so unless you require the information it provides, do not select it.

Executing a Search Action

When Witango Server executes a Search action, the search is performed against the associated data source. The result rowset is automatically

stored as an array in the local variable `resultSet`. The Results HTML for the Search action is then processed.

The HTML in the `<@ROWS></ROWS>` block, if any, is processed once for each record in the results. Use `<@COLUMN>` or `<@COL>` meta tags to include field values.

If the Search action generates no results, and you have specified No Results HTML for the action, that HTML is processed instead of the Results HTML.

Adding Records to a Database

The Insert action adds a record (row) to a table in a database.

Setting Up an Insert Action



Insert Action

When you drag the Insert action icon from the Actions bar into an application file, the Insert action editing window appears:

Column	Value	Quote Value

To set up an Insert action

- 1 From the Data Sources Workspace, drag into the **Column** list the columns whose values you want to set in the new record.



Note You can select columns from only one table.

If you do not add all of the table's columns to the Insert action, the omitted columns are given the default values defined when the database was created.

- 2 In the **Value** field for each column, enter the value for that column in the new record. The value can contain any of the value-returning Witango meta tags, which are substituted upon execution of the application file.

For more information about inserting meta tags in entry fields, see Inserting Meta Tags on page 148.

To insert a meta tag, either click the field and choose **Insert Meta Tag** from the **Edit** menu, or right-click the field and choose **Insert Meta Tag** from the context-sensitive menu that appears.

For more information, see "Quote Value" on page 262.

The **Quote Value** option operates the same as it does in search criteria.

Executing an Insert Action

When Witango Server executes an Insert action, a record is added to the database with the column values specified. The Insert action returns no results.

Modifying a Database Record

The Update action modifies database records matching specified criteria.

Setting Up an Update Action



Update Action

When you drag the Update action icon from the Actions bar into an application file, the Update action editing window appears.

Specify the criteria for the Update

untitled4.taf : Update

Update the table:

Select rows matching these criteria:

Column	Oper.	Value	Incl. Empty	Quote

Update them with these values:

Column	Value	Quote Value

Records in the database matching the criteria are updated with the values specified in this list.

To set up an Update action

- I In the criteria list at the top of the action's editing window, specify which records you want to update.

You edit the criteria list the same way you edit the Search action's criteria list.

For more information, see "Criteria Section" on page 257.



CautionFor an Update action, be *E X T R E M E L Y* careful when setting the **Incl. Empty** option to **false**. You may end up affecting more rows than you intend, possibly even updating all the records from your database table. Just like leaving **Incl. Empty** set to false in a Search action returns all the records, leaving it set to false in an Update action updates all records.

- 2 From the Data Sources Workspace, drag the columns whose values you want to update into the update columns list at the bottom of the action's editing window.



Note You can specify columns from only one table. If you want to update multiple tables, use an Update action for each table. In this case, consider using a Transaction action to make sure all or none of the updates are processed. Only the values in the columns you specify are modified when the action is executed.

- 3 Under **Value** for each column, enter the new value for that column.

The value can contain any of the value-returning Witango meta tags, which are substituted upon execution of the application file.

To insert a meta tag, either click the field and choose **Insert Meta Tag** from the **Edit** menu, or right-click the field and choose **Insert Meta Tag** from the context-sensitive menu that appears.

If you always want to update a column with a fixed value, simply enter that value.

The **Quote Value** option operates in the same way it does in search criteria.

For more information about inserting meta tags in entry fields, see [Inserting Meta Tags](#) on page 148.

For more information, see ["Quote Value"](#) on page 262.

Executing an Update Action

When Witango Server executes an Update action, Witango searches for records matching the specified criteria and updates them with the specified column values.

The Update action returns no results.

Removing a Database Record

For more information, see “Criteria Section” on page 257.

The Delete action removes database records that match the specified criteria. You edit the criteria list the same way you edit the Search action’s criteria list.

Setting Up a Delete Action

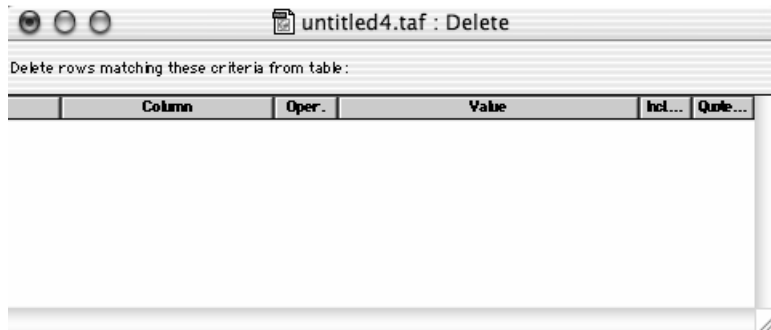


Delete Action

To set up a Delete action

- 1 Drag the Delete action icon from the Actions bar into an application file.

The Delete action editing window appears:



- 2 In the criteria list of the Delete action’s editing window, specify which records you want to delete.



Note You can specify columns from only one table. If you want to delete multiple tables, use a Delete action for each table. In this case, consider using a Transaction action to make sure all or none of the deletes are processed.

You must specify at least one criterion for the Delete action to be valid.



Caution For a Delete action, be *E X T R E M E L Y* careful when setting the **Incl. Empty** option to **false**. You may end up affecting more rows than you intend, possibly even deleting all the records from your database table. Just like leaving **Incl. Empty** set to false in a Search action returns all the records, leaving it set to false in a Delete action deletes all records.

Executing a Delete Action

When Witango Server executes a Delete action, records matching the specified criteria are deleted.

The Delete action returns no results.

Adding Custom Columns to Database Actions

A custom column entry lets you enter any text as the column reference. You can use custom columns wherever Witango accepts columns dragged from the Data Sources Workspace.

Make sure the text entered makes sense in the database action. For example, in a Search action, you could enter the following calculation as a Select column:

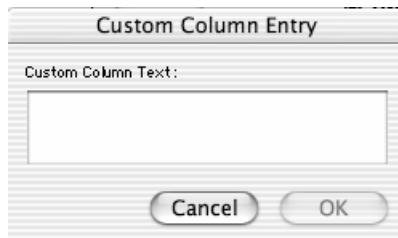
```
orders.order_amt + 20
```

To add a custom column to a database action

1 Do one of the following:

- From the **Edit** menu, choose **Insert Custom Column**.
- Right-click any database action editing window where you can add columns, and choose **Insert Custom Column** from the context-sensitive menu that appears.

The Custom Column Entry dialog box appears:



2 Enter the text to use as the column reference.

You can insert meta tags here.

3 Click **OK**.

Custom columns can be edited later by double-clicking the column reference in the list.

Using Control Actions

Branch, Conditional, Loop, Break, and Return Actions

Normally, actions in a Witango application file are executed sequentially, from top to bottom. However, you can use control actions in an application file to redirect the flow or repeat a sequence, depending on various conditions. The control actions in Witango include Branch action, conditional actions, loop actions, Break action, and Return action.

This chapter covers the setup and operation of the following actions:

- The *Branch* action causes a jump to a designated action or group.
- *Conditional* actions evaluate an expression, and based on the result of that expression, affects the control flow of the file.
- *Loop* actions repeat a set of contained actions a given number of times or while an expression evaluates to true.
- The *Break* action terminates processing in the loop.
- The *Return* action ends execution of the application file and returns any accumulated Results HTML to the Web browser. It can also return to another application file.

Jumping to a Designated Action (Branch Action)

For more information, see “Branching to Other Application Files” on page 348.

The Branch action causes a jump to a designated action or to an action group.

You can set the Branch action to jump to an action in the same application file, or to an action in another application file. If the Branch action jumps to an action in another application file, you can choose whether to return to the previous application file when a Return action is encountered.

Branch Action Destination Rules

There are rules governing where a Branch action can jump to. If the rules are violated, Witango Server returns an error. The rules are dependent on what kind of file you use and whether you are branching to a different file.

Branching within the Same Application File

When branching within the same application file, a Branch action *can* branch to:

- any action at the outermost level.
- any action at the same level in the same block of Witango actions.
- any action that is an ancestor (for example, a parent or a parent’s parent).
- any action that is a first-level child of an ancestor.

A Branch action *cannot* branch to:

- Else If or Else actions.
- any action that is its descendent (for example, a child or a child’s child).
- any action that is a descendent of an action at the same level as this Branch action.

Branching to a Different Application File

When branching to a different application file, a Branch action can only branch to an action at the outermost level.

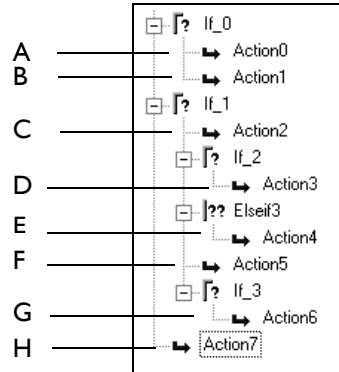
Witango Class File

When using a Witango class file, the rules for Branch actions are similar to those for branching within the same application file. There is an

additional rule: Branch actions in a Witango class file cannot branch outside the current method.

Examples

Consider the following partial application file:



Examples of valid branches:

- BranchActionC can branch to BranchActionF.
Reason: The actions are in the same action block and at the same level.
- BranchActionC can branch to BranchActionH.
Reason: BranchActionH is at the outermost level.
- BranchActionD can branch to If20 and If21.
Reason: If20 and If21 are both ancestors of BranchActionD.
- BranchActionD can branch to BranchActionC.
Reason: BranchActionC is the first-level child of an ancestor (If20) of BranchActionD.

Examples of invalid branches:

- BranchActionB cannot branch to BranchActionE.
Reason: Incorrect relationship. BranchActionE is a descendent of Elsef22, which is at the same level as BranchActionB.
- BranchActionC cannot branch to BranchActionG.
Reason: Incorrect relationship. BranchActionG is a descendent of If23, which is at the same level as BranchActionC.
- BranchActionC cannot branch to Elsef22.

Reason: Even though Elself22 is the first-level child of an ancestor (If20) of BranchActionC, this branch is not allowed because branching to an Else If action is invalid.

Executing a Branch Action

For more information, see “Setting Up a Branch Action” on page 277.

When Witango Server executes a Branch action, it jumps to the designated action. Later, when Witango Server encounters a Return action, one of the following happens:

- If you did not select the **Return to next action after branch** option when setting up the Branch action, Witango Server ends execution.
- If you selected the **Return to next action after branch** option when setting up the Branch action, Witango returns to the action



Note This may not happen if the Branch action branches to a different application file.

For more information, see “Branching to a Different Application File” on page 276.

Branch and Return

You can branch to the same application file or a different application file. In both cases, you can select the **Return to next action after branch** option when setting up a Branch action.

Branching within the Same Application File

When a Branch action in an application file branches to the same application file with the **Return to next action after branch** option selected, Witango Server returns to the action following the Branch action when it encounters a Return action.

Branching to a Different Application File

When a Branch action in one application file branches to another application file with the **Return to next action after branch** option selected, Witango Server returns to the action following the Branch action (in the previous application file) when it encounters a Return action. However, there are certain circumstances under which the Branch action never returns, even when the Return option is selected.

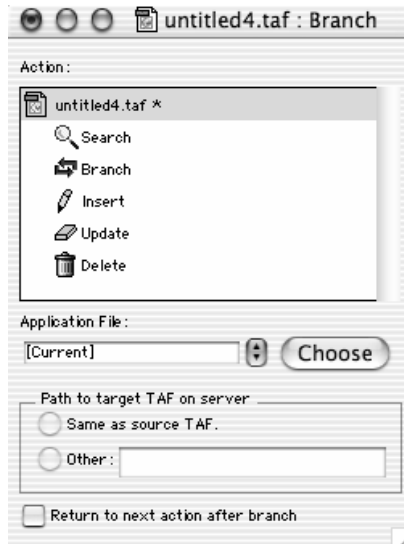
If a Branch action with the Return option selected branches to another application file, and then encounters another Branch action with the Return option *not* selected, the first Branch action never returns to the first application file, even though Witango Server encounters a Return action. That is, the lack of a Return option in the second Branch action takes precedence.

Setting Up a Branch Action



Branch Action

When you drag the Branch action icon from the Actions bar into an application file, the Branch action editing window appears:



To set up a Branch action



Note If you are using the Branch action in a Witango class file method, skip to step 4.

Only branches within the method are allowed; the Application File field and the “Path to target application file on Server” section are disabled.

- 1 From the **Application File** drop-down menu, select the file you want the Branch action to jump to, by doing one of the following:
 - Accept the default (that is, select the current file); go to step 3.
 - If the current file is part of a project, the drop-down menu also shows the other files in the project; select the file you want.
 - If you want to select an application file elsewhere from your hard drive, click **Browse** from the drop-down menu.

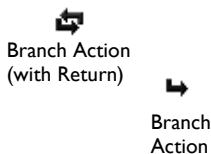
A standard file selection dialog box appears; select an application file.

- 2 If you select an application file that is not the current file, the Action list changes to show the actions in the selected file. The **Path to target TAF on Server** section is also enabled to allow you to specify the path to the application file.

Do one of the following:

- Select the **Same as source TAF** option (the default) to cause Witango Server to always look in the current file's folder.
- Specify in the **Other** field the path to the folder you want, which causes Witango Server to look for the application file in that location. This path is specified relative to the Web Server's document root directory.

For more information, see "Executing a Branch Action" on page 276.



3 For the **Return to next action after branch** option, do one of the following:

- Select the option if you want execution to continue with the action following the Branch action when a Return action is encountered in the destination.
- Do not select this option if you want execution to end when a Return action is encountered in the destination.

The Branch icon in the Action list of the Witango application file changes to reflect the change in the action's behavior.

4 In the **Action** list, select the action you want the Branch action to jump to.

5 Close the Branch action editing window.

The destination action for a Branch must be valid according to the rules on page 274; otherwise, Witango Server returns an error.

Branch Action Destination Navigation

You can navigate from a Branch action to its destination action with the **Go To Destination** context-sensitive menu command.

When you have selected a destination for a Branch action, right-clicking the Branch action allows you to select the **Go To Destination** command from the context-sensitive menu that appears. The name of the target action appears beside **Go To Destination**; the path to the file also appears if the action is in a different application file than the Branch action.

To navigate to a Branch action destination

I Do one of the following:

- Right-click anywhere in the Branch action editing window.
- Right-click the Branch action icon in the application file window.

The **Go To Destination** command appears in the context-sensitive menu, along with the name of the destination action:

- 2 Choose **Go To Destination** from the context-sensitive menu that appears.

Witango Studio automatically selects the designated action if it is in the same application file. If the designated action is in a different application file, Witango opens the application file and selects the target action.

The **Go To Destination** context-sensitive menu item is available only when you right-click either the Branch action or the Branch action editing window. A destination action must be designated, or **Go To Destination** is disabled.

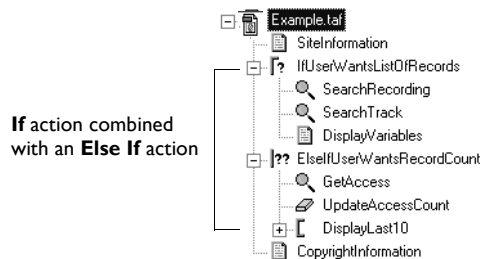
Deciding Course of Actions (Conditional Actions)

Conditional actions consist of three closely related actions: If action, Else If action, and Else action.

The If action is associated with an expression. During execution, Witango Server evaluates whether the conditions stated in the expression are met. If the conditions are met (true), Witango Server proceeds with a sequence of actions in the application file; if the conditions are not met (false), Witango Server proceeds with a different sequence of actions in the application file.

Example: Sports Fan Web Site

Consider a Witango application file executing on a Web site for sports fans. If the user chooses to display information on hockey, the variable `sport` is set with the value `hockey`. Then, an If action evaluates the `sport` variable, and, if the variable has the correct value (in this case, `hockey`), Witango Server searches for and displays hockey information. If the user chooses to display information on football (the variable `sport` is set with the value `football`), an Elself action evaluates the `sport` variable, and, if the variable has the correct value (in this case, `football`), Witango Server searches for and displays football statistics. Witango Server also displays general sports news when this application is executed. The following is taken from an application file designed for our sports fan Web site:



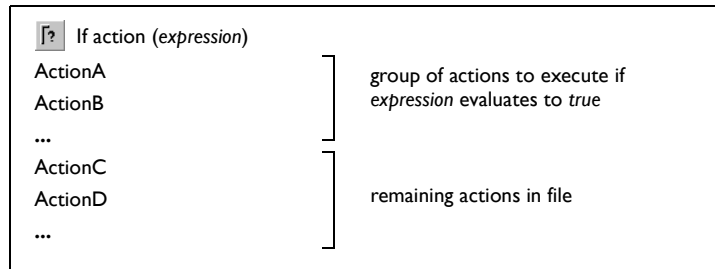
We see that with the use of If and Elself actions (conditional actions), different sets of actions can be executed during the execution of an application file.

General Forms of Conditional Actions

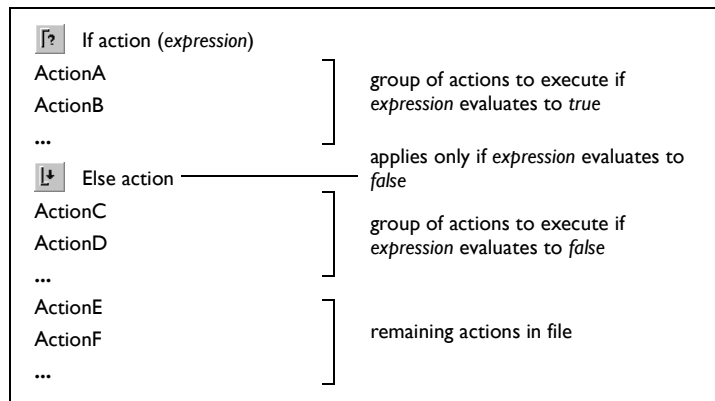
The If action has two related actions: Else If and Else. These conditional actions are often used together: an If action followed by one or more Else If actions and an Else action. However, an If action can exist without Else If actions; it can also exist without an Else action.

The general forms of conditional actions are as follows:

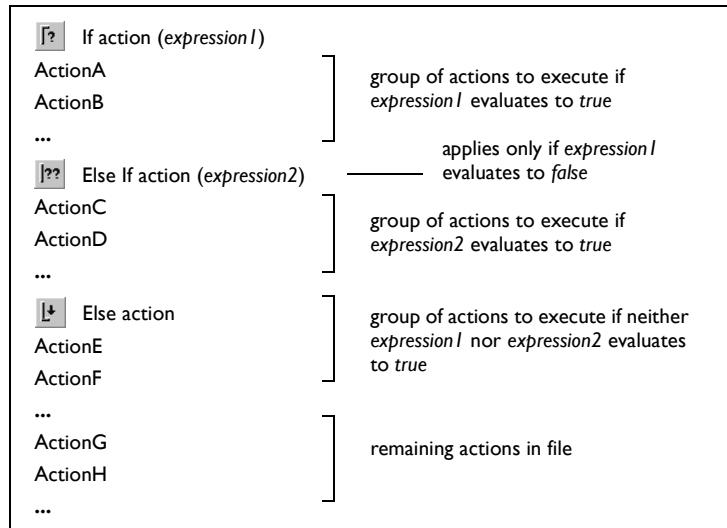
- If Action



- If and Else Actions



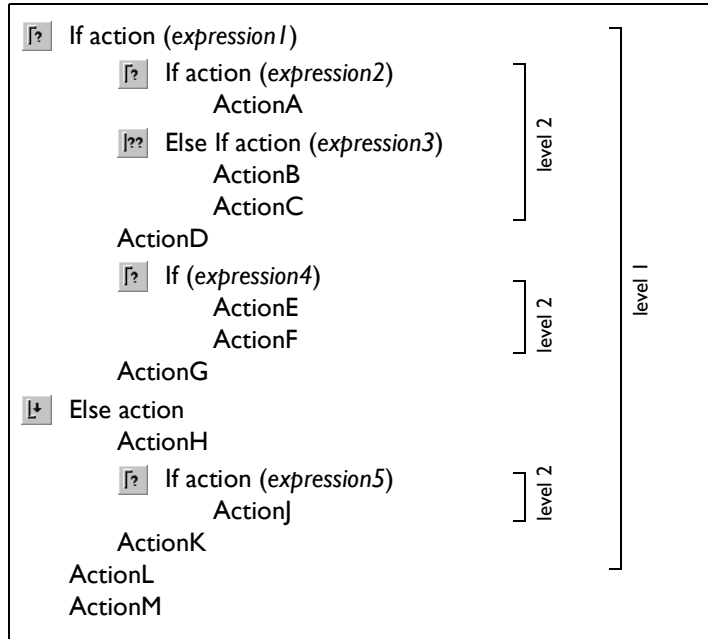
- If, Else If, and Else Actions



Nested Conditional Actions

Conditional actions may be nested; that is, the indented actions under an If, Else If, or Else action may contain other If, Else If or Else actions. You can have multiple levels of nesting.

The following is an example of nested conditional actions:



Performing Operations on Conditional Actions

Working with conditional actions is similar to working with grouped actions. Each If, Else If, or Else action—together with the indented actions under it—acts like a group.

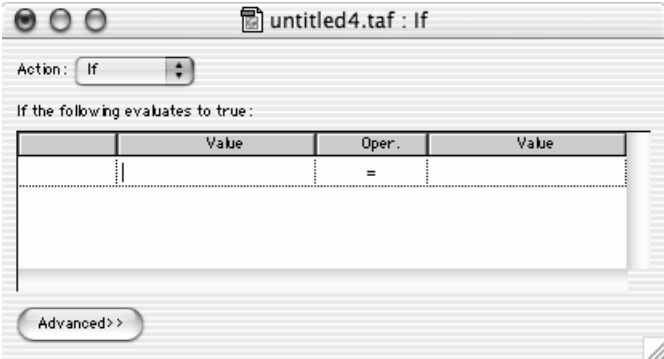
For information on the operations you can perform on groups, see [Working With Action Groups](#) on page 247.

Setting Up Conditional Actions



If, Else If, and Else Actions

When you drag the If or Else If action icon from the Actions bar into an application file, the If action editing window appears:



By default, the If action editing window appears in basic view, which allows you to create expressions quickly. An empty parameter row appears in the dialog box, ready for you to edit.



Note Dragging the Else action icon into an application file does not open any action editing window. However, if you double-click an Else action in an application file, the action editing window opens so you can change the Else action to an If or Else If action.



Else Action

For more information, see “Advanced View” on page 285.

An advanced view is also available that gives you more flexibility than the basic view when specifying evaluation expressions.

You change the type of conditional action by selecting **If**, **Else If**, or **Else** from the **Action** drop-down menu.

The If and Else If action editing windows are basically the same, and you enter evaluation expressions the same way for both of them. When you select **Else**, however, only the **Action** drop-down menu is active. This allows you to change to another type of conditional action.



Note If you change back to an If or Else If action type from an Else action type before closing the action editing window, any If or Else If expressions you specified previously are retained.

Basic View

The basic view consists of a parameter list, which works as follows:

- Each row in this list contains a parameter; it allows two values to be compared, using an operator.
- All the parameters in this list are connected together, using logical operators.
- All the parameters together constitute a single expression that Witango Server evaluates.
- If the expression evaluates to “1” or “true”, it is considered true; otherwise, it is considered false.

To specify values for the basic view parameters

Specify values as follows:

- **Logical Operator.** The first field in the parameter list is the logical operator. There are two logical operators: **and** and **or**.

Click the parameter row, then click the field to display a drop-down menu to choose a logical operator. The logical operator is used when the expression includes more than one row; it specifies the relationship between the two rows.

- **Value.** Enter the values to use in the parameter. Do not add quotation marks around the values.
 - If you are using the =, !=, >, <, >=, or <= operator (see the description of **Oper.** on page 284), enter the two values you want to compare in the two **Value** columns respectively.
 - If you are using the `Is Empty` or `Is Not Empty` operator (see the description of **Oper.** on page 284), enter the single value that you want to compare in the left **Value** column.

The values can contain any value-returning Witango meta tags, which are substituted when Witango Server executes the action.

You can also use the **Insert Meta Tag** command to enter many of the commonly-used meta tags.

To insert a meta tag, either click the field and choose **Insert Meta Tag** from the **Edit** menu, or right-click the field and choose **Insert Meta Tag** from the context-sensitive menu that appears.

- **Oper.** Specify the operator to use to compare the two values specified on the same row.

Click the parameter row, then click the field to display a drop-down menu to choose an operator.

For more information, see “Logical Operator” on page 258.

For more information about inserting meta tags in entry fields, see Inserting Meta Tags on page 148.

For more information, see “Operator” on page 259.

Possible operators include:

Operator	Meaning
=	is equal to
!=	is not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
Is Empty	matches an empty field
Is Not Empty	matches a non-empty field

To add a new parameter row

- 1 Open the If action editing window (if it is not already open).
- 2 Do one of the following:
 - From the **Edit** menu, choose **Insert**.
 - On the main toolbar, click the Insert icon.
 - Right-click the list area, and choose **Insert** from the context-sensitive menu that appears.
 - Press **Insert**.



Insert icon

To delete a parameter row

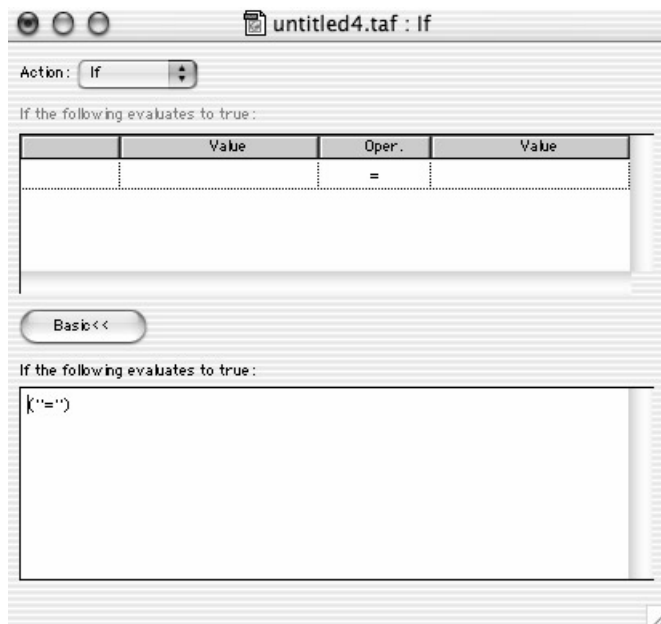
- 1 Open the If action editing window, if it is not open already.
- 2 Do one of the following:
 - Select the row you want to delete; from the **Edit** menu, choose **Clear**.
 - Right-click the row you want to delete and choose **Clear Criterion** from the context-sensitive menu that appears.
 - Select the row you want to delete; press **Delete**.

Advanced View

When you click **Advanced** in the basic view, the following happens:

- The window expands to show a text area where you can enter text-based expressions.
- The **Advanced** button changes to **Basic**.

- The parameter list in the basic view is disabled (appears grayed).
- The parameters in the basic list are automatically converted to an equivalent text expression in the advanced text area.



There are some important differences between basic view and advanced view:

- For simple situations, basic view is easier to use.
- Advanced view presents a free-form text area to give you more flexibility than the basic view when specifying evaluation expressions. For example, if you want to use parentheses to control the evaluation order, you can enter the expression in this area.

The expression entered here takes the same form as expressions specified for the `<@CALC>` meta tag.

- If the expression in the advanced text area evaluates to “1” or “true”, the expression is considered true; otherwise, it is considered false.
- Any editing you do in the advanced text area supersedes the parameters in the basic view list area.
- If you return to the basic view from the advanced view, any editing you do in the advanced text area is lost.

Witango can take the parameters appearing in the basic view and regenerate the equivalent text-based expression in the advanced view.

To regenerate the parameters from the basic view

1 Do one of the following:

- To insert the text-based expression in the advanced text area, right-click where you want the expression.
- To replace selected text with the text-based expression in the advanced text area, select the text and right-click.

2 From the context-sensitive menu that appears, choose **Insert Expression As Above**.

The text-based expression appears in the advanced text area.



Tip You can also drag snippets from the Snippets Workspace to this text area.

To return to the basic view

- Click **Basic**.



Caution If you changed the expression in the advanced view, your changes are lost when you return to the basic view. An alert box asks if you want to continue.

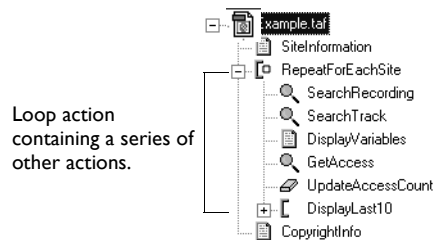
Repeating a Set of Actions (Loop Actions)

A loop action repeats the execution of a set of actions for a given number of times or while an expression evaluates to true. In an application file, the actions to be repeated in the loop are indented under the loop action.

Example: Music Store

Consider an online music store. It allows customers to search for their favorite recordings and artists. As an additional service, this store also searches other sites for recordings and artists that it does not have in stock. A loop action can be set up such that Witango Server goes through the sites that this store has an agreement with. For each of these sites, Witango Server searches for recordings and artists, and then updates the results and displays them to the customer. This process continues until Witango Server comes to the end of the sites. The loop ends at this point, and Witango Server proceeds to the next action outside the loop, which is to present the order information.

The following is taken from an application file designed for our music store:



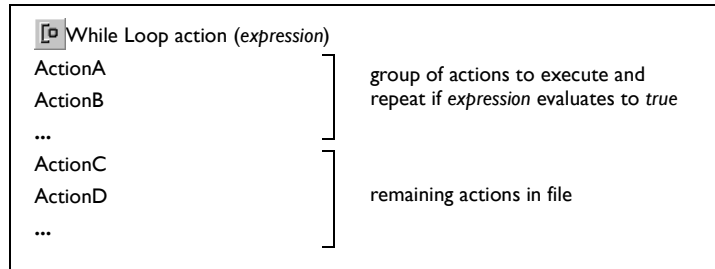
General Forms of Loop Actions

There are three kinds of loop actions:

- **While Loops.** See the following sections for details.
- **For Loops.** See the following sections for details.
- **Objects Loops.** See Using the Objects Loop Action on page 391 for details.

While Loop

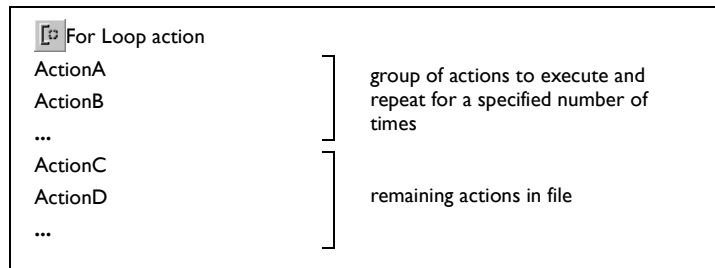
The While Loop takes the following general form:



A While Loop is associated with an expression. If this expression evaluates to true, Witango Server executes the indented actions listed under the While Loop action. (In this example, these indented actions include ActionA and ActionB.) When all the indented actions are executed, the expression is evaluated again. Witango Server repeats the indented actions as long as the expression evaluates to true. When the expression evaluates to false, Witango Server proceeds to the next action at the same level as the While Loop action. (In this example, it is ActionC.)

For Loop

A For Loop action takes the following general form:



A For Loop specifies that a group of indented actions listed under the For Loop is to be executed and repeated a number of times. (In this example, these indented actions include ActionA and ActionB.) After repeating so many times, Witango Server proceeds to the next action at the same level as the For Loop action. (In this example, it is ActionC.)

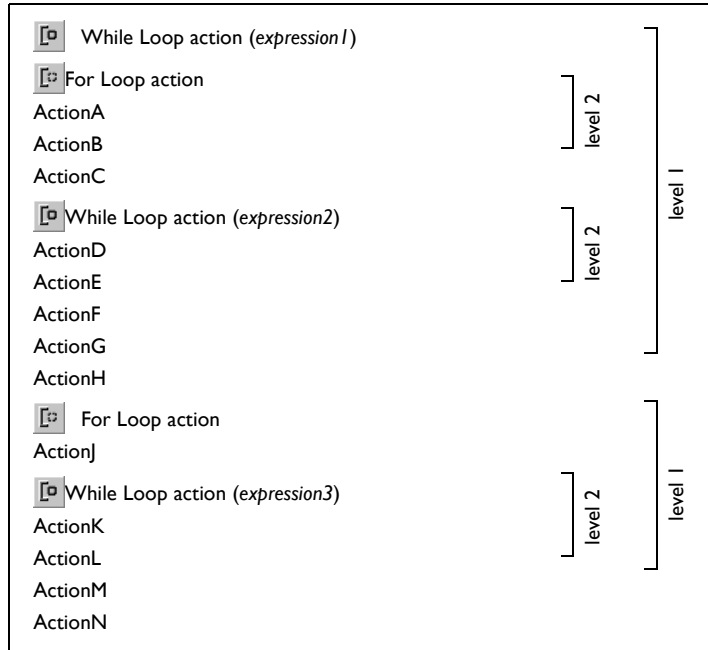
For more information, see "Exiting a Loop (Break Action)" on page 294.

Witango also includes a Break action you can use to exit a loop action before the loop conditions for termination are met.

Nested Loop Actions

Loop actions may be nested; that is, the indented actions under a While Loop or For Loop action may contain other While Loop or For Loop actions. You can have multiple levels of nesting.

The following is an example of nested loop actions:



Setting Up Loop Actions

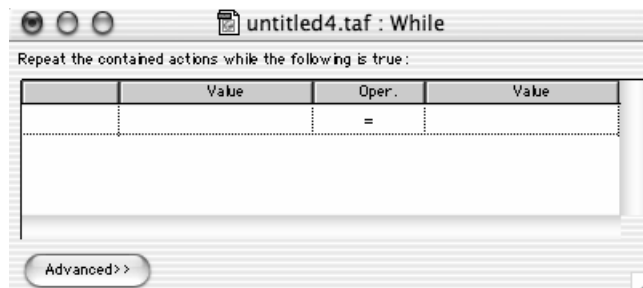


While Loop Action

While Loop

A While Loop action executes and repeats the actions in the loop (shown as indented actions) *while* an expression evaluates to true.

When you drag the While Loop action icon from the Actions bar into an application file, the While Loop action editing window appears in its basic view, allowing you to create evaluation expressions quickly.



In order for the While Loop to work properly, you must avoid the following pitfalls:

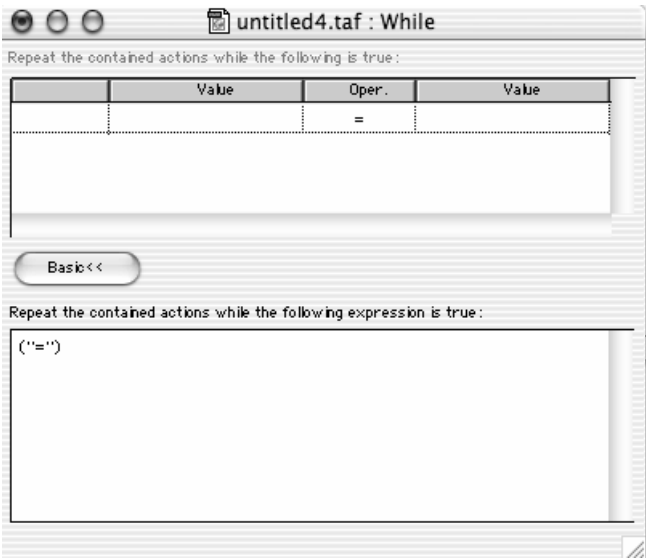
- **Never entering the loop.** If the expression you specify does not evaluate to true when Witango first executes the While Loop action, the indented actions under it are never executed.
- **Infinite looping.** Make sure that at least one value being compared in the expression is being changed inside the loop. If this is not the case, a “true” evaluation causes Witango to execute the enclosed actions forever.

Be careful when constructing a While Loop expression. You want to ensure that it eventually evaluates to false.

For more information, see “Basic View” on page 283.

The basic view for a While Loop action is similar to the basic view for If and Else If actions.

The While Loop action editing window also has an advanced view for more flexibility when constructing evaluation expressions. For example, you can use parentheses to control the evaluation order.



For more information, see “Advanced View” on page 285.

This view is similar to the advanced view for If and Else If actions.

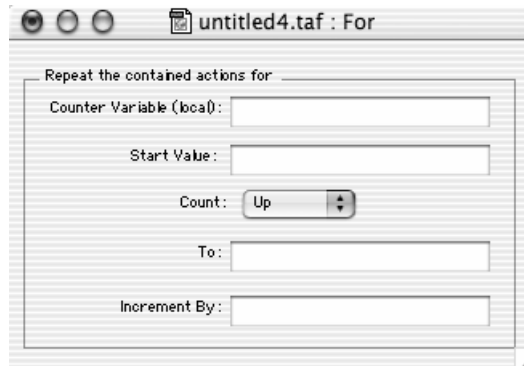
For Loop

A For Loop action executes and repeats the actions in the loop (shown as indented actions) for a given number of times.



For Loop Action

When you drag the For Loop action icon from the Actions bar into an application file, the For Loop action editing window appears:



Set the parameters in a For Loop action as follows:

- **Counter Variable (local).** The name of a local variable to use to access the current value of the counter.



Note This parameter is optional. It is not required to use the action.

- **Start Value.** The starting value for the loop counter. The default is 1.
- **Count.** The direction of the counting from the starting value to the ending value. You must specify this parameter so Witango can increment or decrement the counter properly. Choose **Up** or **Down** from the drop-down menu to set the counter to increment or decrement, respectively. The default is **Up**. When **Down** is selected, the **Increment By** field name becomes **Decrement By**.
- **To.** The ending value for the loop counter.
- **Increment/Decrement By.** The value the counter increments or decrements by on each loop.



Tip All For Loop action fields, except for the **Count** field, support Witango meta tags.

Executing Loop Actions

The [General Forms of Loop Actions](#) page 288 section on page 288 explains the basics of how Witango Server executes the While Loop and the For Loop. This section provides some additional information.

While Loop

- If Witango Server finds the expression invalid, it returns a runtime error.
- Witango Server evaluates any meta tags in the expression on each pass through the loop.

For Loop

- If **Start Value** is a meta tag, Witango Server evaluates it prior to the first pass through the loop.
- If the **To** and **Increment/Decrement By** fields contain meta tags, Witango Server evaluates them on each pass through the loop.

Performing Operations on Loop Actions

Working with loop actions is similar to working with grouped actions.

For information on the operations you can perform on grouped actions, see [Working With Action Groups](#) on page 247.

Exiting a Loop (Break Action)

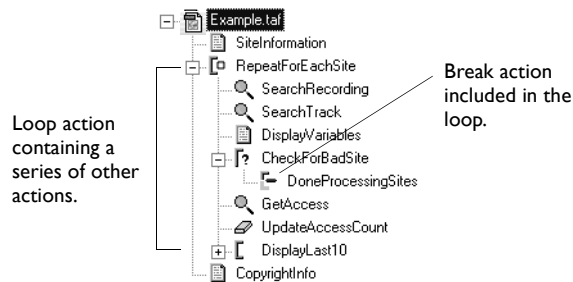
The Break action prematurely terminates processing in a loop or group action. On termination, processing continues at the next action after the loop or group.



Break Action

Drag the Break action icon from the Actions bar into a loop or Group action at the point you want the loop or group to terminate.

The following is an example of an application file showing how loop and Break actions appear:



On execution, the Break action terminates the loop, and processing continues at the next action after the loop.



Note If you include a Break action outside a loop or group, Witango Server generates a runtime error on execution.

Ending File Processing (Return Action)



Return Action

The Return action ends application file processing and returns any accumulated Results HTML to the Web browser.

For more information, see “Jumping to a Designated Action (Branch Action)” on page 274.

The exception to this is if the current execution flow is the result of a Branch that had its **Return to next action after branch** option set. In this case, the execution returns to the action following the Branch when a Return action is encountered.

Extending Witango Functionality

Script and External Actions

The *Script* action provides an interface within Witango for executing JavaScript code at the server. The script executed can return to Witango a value you can access using Results HTML.

The *External* action calls an executable invoked using a command line, a Dynamic Link Library (DLL), or a Java class file to perform processing and, if desired, return results.

Witango may also be extended through COM, JavaBean, and Witango class file objects. For more information, see *Witango and Objects* on page 343.

This chapter covers the following topics:

- setting up and executing a Script action
- configuring a DLL call or a Java action
- using a command line
- assigning variables to action parameters
- assigning action attributes
- deleting action parameters
- executing an External action.

Executing JavaScript

The Script action provides an interface for executing core JavaScript code at the server. The script executed can return a value to Witango, which is accessible using Results HTML.

Witango is JavaScript 1.4 compatible, meaning it includes the official JavaScript Reference implementation from Netscape, and conforms to version 1.4 of the language.



Note Witango supports the general purpose core of the language. The objects representing the Web browser and its contents are *not* supported, because the scripts are executed at the server where these objects do not exist. For your convenience, a JavaScript HTML reference is provided with the Witango HTML help.

Setting Up a Script Action



Script Action

When you drag the Script action icon from the Actions bar into an application file, the Script action editing window appears:



The **JavaScript object and variable scope** parameter defines the lifetime of the objects and functions declared in the script. This is a concept similar to the scope of variables in Witango.

The drop-down menu/text box has several choices, or you can enter a custom scope:

- Default specifies Witango's default scope, which is set by the `defaultScope` configuration variable. .
- **Immediate** specifies that the objects and functions go away immediately after the action is executed.
- **Method** specifies that anything defined in the script can be referenced within the current method of a Witango class file.
- **Instance** specifies that anything defined in the script can be referenced within the current instance of a Witango class file.



Note Method and Instance appear only if the Script action is within a Witango class file.

- **Request** specifies that anything defined in the script can be referenced in another script in the same application file execution.
- **User** specifies that anything defined in the script can be referenced in another script run by the same user.
- **Application** specifies that anything defined in the script can be referenced in another script in an application file in the current Witango application.
- **Domain** specifies that anything defined in the script can be referenced in another script in an application file in the Witango domain.

For information on using the script text area, see HTML Editing Window on page 9, and The SQL Query Window on page 22.

You enter the text script to be executed in the **JavaScript script** text area. The script may contain meta tags. All meta tags in the script are substituted before the script is executed.



Tip You could use an `<@INCLUDE>` meta tag to reference an external JavaScript file.



Tip The script text area functions the same as HTML text editing windows.

When you right-click the script text area, the same context-sensitive menu for the SQL text area of a Direct DBMS action appears.

Executing a Script Action

Witango Server executes the Script action in a similar way it executes the `<@SCRIPT>` meta tag.

Any value returned by a script is accessible in the Results HTML by using `<@COL 1>` inside a `<@ROWS>` block. The action's result set (a 1 by 1 array) is also stored automatically in a local variable, `resultSet`.

- 2 In the **DLL** field, type the fully qualified path to the DLL you want to call, for example:

C:\Program Files\Witango\externals\Test.dll.



Note The path specified here is appended to the value of the `absolutePathPrefix` configuration variable. If this configuration variable has a value (in either application or system scope), this field should contain a path *relative* to that location.

This path may contain meta tags.



Note The DLL called by the External action must be written to conform to the API described in Appendix B.

- 3 Insert a new parameter row by doing one of the following:

- From the **Edit** menu, choose **Insert**.
- On the main toolbar, click the Insert icon.
- Right-click then the **Parameters** area, and choose **Insert** from the context-sensitive menu that appears.

A new parameter row appears. The parameters are numbered for easy identification.



Tip You may re-order parameters by dragging them within the list.

- 4 In the **Value** field, type a parameter value.

All parameters are passed to the corresponding APIs by pointers of type `(char *)`. Parameter values may include any value-returning Witango meta tags, which are substituted when the action is executed.

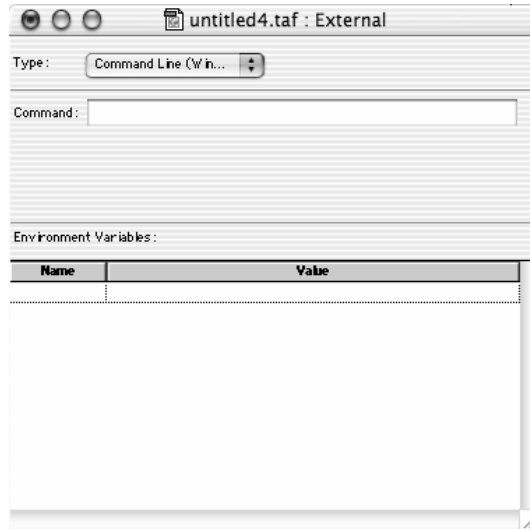
Using a Command Line

External actions allow you to run any executable file (for example, batch file, shell or Perl script, C application) and, if desired, retrieve results. Values are passed from Witango to the executable by means of environment variables. Results are retrieved by Witango from text that the External action has written to the standard output stream (stdout). Rows and columns are delimited with tabs and returns, respectively.

To configure a command line External action

- 1 From the **Type** drop-down menu, select **Command Line**.

The External action editing window for the Command Line type appears:



- 2 In the **Command** field, specify the executable file name.

The value of this field (after meta tag substitutions) must be a valid file path (for example, `c:\temp\dir.bat`). Command line parameters are not allowed here.



Note The path specified in the **Path** field is appended to the value of the `absolutePathPrefix` configuration variable. If this configuration variable has a value (in either application or system scope), this field should contain a path *relative* to that location

You may include command line switches here (Linux only).



Note A command line containing a DOS-like command (for example, `dir`) does not work because 'dir' is not a file. Commands calling other command processors (shell) also do not work. If you want to execute an operating system command, you should create a batch file or shell script.

On Unix, shell scripts must have read/execute permission for the user running the Witango daemon. In order to be properly executed under the appropriate shell, the shell script must have a shell execution directive such as `#!/bin/sh` as its first line.

- 3 Insert a new environment variable row by doing one of the following:



- From the **Edit** menu, choose **Insert**.
- On the main toolbar, click the Insert icon.
- Right-click the environment variables area, and choose **Insert** from the context-sensitive menu that appears.

A new environment variable row appears.

- 4 In the **Name** field, enter the name of an environment variable to create for the process. This value is passed on the command line to the External action.



Note The name of an environment variable is case sensitive.

- 5 In the **Value** field, enter the value to assign to the named environment variable.

Environment variables may include any value-returning Witango meta tags, which are substituted when the action is executed.



Tip You may re-order variable rows by dragging them within the list.

For more information, see “Assigning Attributes” on page 305.

Configuring a Java Action

Witango ships with its own Java server, which is on the same machine as Witango Server. The Java type External action allows you to connect to the Java server.

To configure a Java external action

- 1 From the **Type** drop-down menu, choose **Java**.

The External action editing window for the Java type appears:

Name	Value
1	

- 2 In the **Java Server** field, type the IP address or *hostname.domain* of the machine running the Java server.
The default server name is **localhost** (127.0.0.1), meaning that the Java server is on the same computer as Witango.
- 3 In the **Port** field, type the port number the Java server is listening on. The default port is 4000.
- 4 Select either **Java Class** or **Java Bean** to specify the type of Java file.
- 5 In the **Path** field, type the fully qualified path to the Java file. This path can contain meta tags.



Note The path specified here is appended to the value of the `absolutePathPrefix` configuration variable. If this configuration variable has a value (in either application or system scope), this field should contain a path *relative* to that location.

- 6 Insert a new argument row by doing one of the following:

- From the **Edit** menu, choose **Insert**.
- On the main toolbar, click the Insert icon.
- Right-click then the arguments area, and choose **Insert** from the context-sensitive menu that appears.

A new argument row appears. The arguments are numbered for easy identification.



Tip You may re-order arguments by dragging them within the list.



Note Arguments may include any value-returning Witango meta tags, which are substituted when the action is executed.

Assigning Attributes

For a description of each attribute, see *Assigning Attributes to Actions* on page 236.

You can also assign the following attributes to an External action:

- Results HTML
- No Results HTML
- Error HTML.

You assign these attributes using the **Attributes** menu.

You can also right-click the External action icon or name in the application file, or click the open External action editing window to display a context-

sensitive menu of available attributes. The **Properties** command is also available in this menu.

If you right-click an action parameter, the **Edit** command is active allowing you to edit the parameter value; otherwise, it is disabled (grayed).

Deleting Parameters

To delete an External action parameter

- 1 Open the External action editing window.
- 2 Select the parameter row you want to delete, and do one of the following:
 - From the **Edit** menu, choose Delete.
 - Right-click the parameter, and choose **Delete** from the context-sensitive menu that appears.
 - Click the Delete icon on the main toolbar.
 - Press **Delete**.
- 3 When you are asked to confirm deletion of the selected row, click **OK**.



Executing an External Action

When an External action is executed, the DLL, command line, or Java specified is called and the parameters are passed to it.

Any results returned are accessible in Results HTML in the same way as in the Search action—by using a `<@ROWS></@ROWS>` block. The `<@COLUMN>` meta tag, however, does not work in the External action. You must use the `<@COL>` meta tag along with an item number to refer to data items, as the items do not have names.

A single item result is treated as a one-column row, a list of items is treated as a row of columns, and a list of lists is treated as a rowset.

For example, if an External action returns a list of lists of three data items each, all the results can be viewed with the following Results HTML:

```
<@ROWS>
Row <@CURRETROW>, Item 1: <@COL 1><BR>
Row <@CURRETROW>, Item 2: <@COL 2><BR>
Row <@CURRETROW>, Item 3: <@COL 3><BR>
<HR>
</@ROWS>
```

For the command line and Java options, the value returned is treated as tab and return delimited: a tab separates columns and a return separates rows.

Only textual data can be returned from an External action.

The entire result rowset from an External action is automatically assigned to a local array variable, `resultSet`.

If the External action generates no data, and you have specified No Results HTML for the action, that HTML is processed instead of the Results HTML.

Disabling JavaScript, Java and External Actions

You can specify that External actions are executed only in a specified directory of Witango Server using the `absolutePathPrefix` configuration variable. Using this configuration variable to set the path ensures that users cannot access directories other than the specified ones when using the External action.

JavaScript, Java, and External actions are by default enabled in Witango. If you want to disable (or enable) these features, you can do so by changing the following options in the Witango Administration Manager (the `config.taf` application file), in the **Feature Switches** screen:

```
javascriptSwitch
javaSwitch
externalSwitch
```

Check or uncheck the check boxes beside the options.

Sending Electronic Mail From Witango

Mail Action

The *Mail* action sends out electronic mail (e-mail) using the Simple Mail Transfer Protocol (SMTP). SMTP is the main protocol used to send mail on the Internet.

The Mail action lets you send e-mail messages from Witango application files and Witango class files. For example, you might send an e-mail message to a list of recipients notifying them of a change to a database or that a particular file has been executed. Many types of information gathering are possible. For example, you can use e-mail for inventory management, shipping and receiving, data compilation, generating sales leads, or any function that can use data derived from activity in a database or an object.

You can also attach files to an e-mail message generated from Witango, add custom headers, and specify the character set and encoding used for the e-mail message.

This chapter covers the following topics:

- setting up a Mail action
- disabling the Mail action in Witango.

Setting Up a Mail Action



Mail Action

When you drag the Mail action into your application file, the Mail action editing window appears:

Specify the attributes of your message under the three tabs in the top panel. All of the Mail action fields support the use of Witango meta tags, which are evaluated at the time the e-mail message is sent.

General Tab

- **From.** Enter the e-mail address of the sender of your message. Normally, this is also the address that replies and error messages go to.

If this field is left empty, Witango uses the system configuration variable `mailDefaultFrom` to determine the default value.

The value of this configuration variable can be changed in the Witango Admin Application, `config.taf`: enter the address of the person sending the e-mail message, for example, `Witango@example.com`. This configuration variable is stored in the Witango Server configuration file (`witango.ini`).



Note If both the configuration variable and the **From** field are empty, then the e-mail message cannot be sent. An e-mail message must always be *from* somebody.

- **To.** Enter the e-mail address or a comma-delimited list of addresses.
- **Cc** (Carbon Copy). Enter the e-mail address of the person you want to send a copy of the message to. This field also allows you to enter a comma-delimited list of addresses.
- **Bcc** (Blind Carbon Copy). This field is the same as **Cc** except the recipients are not listed in the message; that is, the **To**, **Cc** and other **Bcc** recipients do not know that the message was also sent to those addresses.
- **Subject.** Enter the subject of the e-mail message.

You compose your e-mail message in the bottom pane of the Mail action window.

Proper E-Mail Address Syntax

You must use a valid e-mail address format in the **From**, **To**, **Cc**, and **Bcc** fields of the Mail action. The following formats are supported by Witango Server:

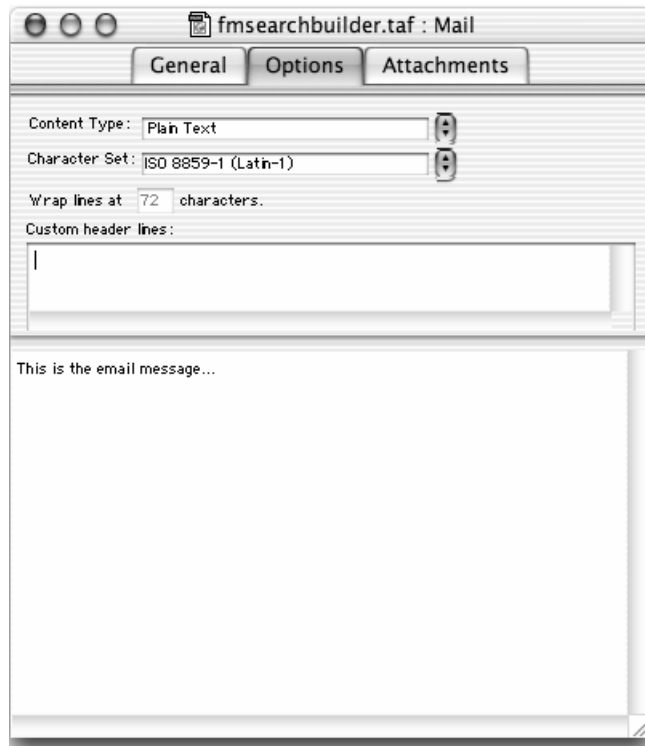
- `johnndoe@example.com`
e-mail address.
- `<johnndoe@example.com>`
e-mail address in angle brackets.
- `"John Doe" <johnndoe@example.com>`
name in quotes, e-mail address in angle brackets.
- `johnndoe@example.com (John Doe)`
e-mail address, name in parentheses.



Note Some e-mail clients may not show the name when displaying an e-mail header if you use the format shown in the final example.

Options Tab

Click the **Options** tab and specify how you want your message to appear to the recipient, as shown in the following fields:



- **Content Type.** Specify the output format of your e-mail message by selecting Plain Text or HTML from the drop-down menu, or enter a content type in the text field provided.
- **Character Set.** Select an option from the drop-down menu to specify the character format for the e-mail message.



Note When the message is sent, Witango Server adds the following MIME header lines, displaying the message to the recipient in the chosen content type and character set:

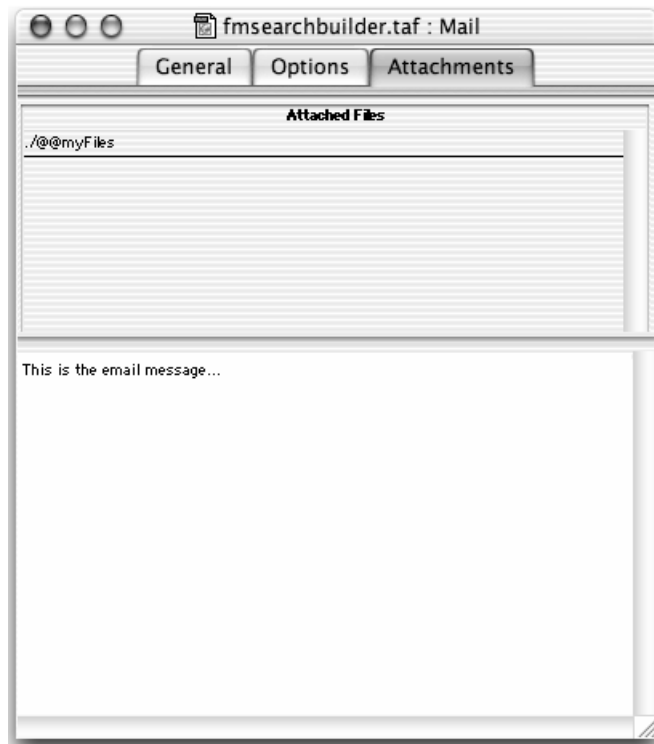
```
MIME-Version 1.0
Content-type: type; charset="charset"
Content-transfer-encoding: encoding
X-Mailer: Witango <@VERSION>
```

The type, charset, and encoding options are replaced by the content type (for example, text/plain or text/html), character set (for example, ISO-8859-1), and encoding (for example, quoted-printable) selected by the user.

- **Wrap Lines At.** Enter the line length of your message body. The value of this field must be between 30 and 132 characters. The default value is 72 characters. This field is available when the **Character Set** option is set to `ASCII` or a user-entered value. With non-ASCII character sets, no wrapping occurs.
- **Custom Header Lines.** Enter text or meta tags that are to be displayed as custom headers at the end of the message headers. Data in this field should not exceed 32K.

Attachments Tab

- Click the **Attachments** tab to attach a file to your message.



To attach a file to a message

- I Do one of the following:
 - From the **Edit** menu, choose Insert.
 - On the main toolbar, click the Insert icon.
 - Right-click in the Filenames window and choose **Insert...** from the context-sensitive menu that appears.



- 2 Specify the full path and name of the file to be attached; for example, `C:\Outbox\Attachments\myattachment.txt`.



Note The paths specified here are appended to the value of the `absolutePathPrefix` configuration variable. If this configuration variable has a value (in either application or system scope), this field should contain a path *relative* to that location.

For each path field, meta tags can be inserted to a maximum length of 1024 characters. Witango evaluates meta tags in the file path for each attachment in the list. After meta tag substitution, Witango determines whether the value is text or an array. If the value is an array, Witango processes every cell in the array as a separate file path.

For example, you could enter the following in a field in the Filenames section of the Mail action dialog box:

```
@@myFiles
```

When the application file is executed, the file or files specified by the variable `myFiles` (single value or array) are attached to the generated e-mail message.

When the Mail action is executed, Witango Server connects to the SMTP server. Witango then sends the message to all the specified recipients.

The SMTP server is defined by the configuration variables `mailServer` and `mailPort`. These variables can be changed using the Witango Configuration Manager (`config.taf` application file). They are stored in the Witango Server configuration file (`witango.ini`).

The result of the action is a one-column array of the messages sent to and received from the SMTP server. Use `<@COL 1>` inside a `<@ROWS>` block in the Mail action's Results HTML to display these results. This information can be useful for debugging a Mail action.

The `resultSet` for a Mail action shows both the mail server's and the client's side of the SMTP conversation, and the commands Witango sends to the mail server.

Disabling Mail

You can specify that mail attachments to come only from a specified directory of Witango Server using the `absolutePathPrefix` configuration variable. Using this configuration variable to set the path for mail attachments ensures that users cannot access directories other than the specified ones when using the Mail action. This configuration variable also affects all other actions which have absolute paths as parameters.

Mail actions are enabled in Witango by default. If you want to disable (or enable) this feature, you can do so by changing the following option in the `config.taf` application file **Feature Switches** screen:

```
mailSwitch
```

Check or uncheck the check box beside the option.

Reading, Writing, and Deleting Files

File Action

The *File* action allows you to read, write, and delete files on the Witango Server machine. Some of the functions you might want to perform using this action are as follows:

- Store data permanently to disk for later retrieval (as opposed to variables which are in memory).
- Keep a log file, appending to it when a certain function gets called in your application file.
- Write HTML files using database-generated data to your Web server document folder.
- Store data to a file for export to an external system, providing data navigation.
- Import data from a file from an external system, for example, daily reports from a mainframe, newswire feeds, and periodic updates to a third-party supplier.
- Use the action instead of giving FTP access to deploy (upload) one or two files from a Web site.
- Use the action as an administrative tool to deploy graphics or as an intranet tool to deploy word processor documents or other types of documents to the server.

The topics covered in this chapter include:

- setting up a File action
- handling file security.

Setting Up a File Action

You can set up the three types of file operations using the File action.

- **Read** (the default) allows you to specify the path and file name of the file you want to read. You also specify if you want to read the entire file or some part of it, and store the data in a local variable.
- **Write** allows you to specify the path and file name of the file to write data to. You also specify what that data is and, if the file exists, whether the data should be appended to or overwrite the existing data. You can also store the file name in a local variable.
- **Delete** allows you to specify the path and file name of the file you want to delete.



File Action

When you drag the File action icon from the Actions bar, the File action editing window appears:



Note For any of the editing fields, you can include value-returning Witango meta tags or drag snippets from the Snippets Workspace. You can also right-click an editing field to display a context-sensitive menu of editing commands, including the **Insert Meta Tag** command.

Setting Up Read Options

From the **File Operation** drop-down menu, select **Read**, if it is not already selected.

The action editing window changes to show the options for the **Read** type operation, which you specify as follows:

- **File.** The path and file name of the file you want to display to the user. You must specify the full, absolute path, not the path relative to the Web server root.



Note The path specified here is appended to the value of the `absolutePathPrefix` configuration variable. If this configuration variable has a value (in either application or system scope), this field should contain a path *relative* to that location.

- **Read.** Sets which part of the file to read. You can choose **Entire file**, or select one of the following options:
 - First.** Type the number of bytes to read from the start of the file.
 - Last.** Type the number of bytes to read at the end of the file.
 - Bytes.** Type the starting and ending bytes. If the ending byte you are specifying is the end of the file, you can select **EOF** instead.
- **Store data in local variable.** The name of a local variable to store the read data in.



Note The read data is also available as `<@COL 1>` in the action's Results HTML and is stored in the local `resultSet` variable.

If you do not specify a file, the action does not do anything. In other words, the action behaves like a file exists, but the file is empty. Specifically, the specified variable is empty.

Setting Up Write Options

From the **File Operation** drop-down menu, select **Write**.

The action editing window changes to show the options for the **Write** type operation.

Specify the **Write** options as follows:

- **File.** The full, absolute path and file name of the file you want to write data to, for example:

```
c:\inetpub\wwwroot\client\uploads\mydoc.doc
```



Note The path specified here is appended to the value of the `absolutePathPrefix` configuration variable. If this configuration variable has a value (in either application or system scope), this field should contain a path *relative* to that location. This file information can also come from a variable or an argument.

You can also tell Witango to generate a temporary file by selecting **Temporary file** from the drop-down menu. If you select this option, the server creates a temporary file using standard routines for the operating system.

- **Data to write.** The data to write to the specified file. For example, you could enter the named post argument for a form field where the user enters the data to be saved in the specified file, or enters a file name to deploy.

- **If file exists.** Specify what you want to do if the file already exists. Select one of the following options:
Append to it appends to the existing file the data you are writing.
Overwrite it replaces the existing data in the file with the data you are writing.
Generate an error generates an error message on execution.
- **File Attributes** (used by Mac OS servers only). These Mac OS only **Type** and **Creator** codes are used when creating a file. `TEXT` and `R*ch` are the default values for new actions. They are also the values used by the server if either field evaluates to empty. Witango Server uses the first four characters of each field (after substitution). If you specify fewer than four characters, the value is space padded to the end.
- **Store file name in local variable.** The name of a local variable in which to store the path and file name of the file written to. You would use this option when you write data to a temporary file.

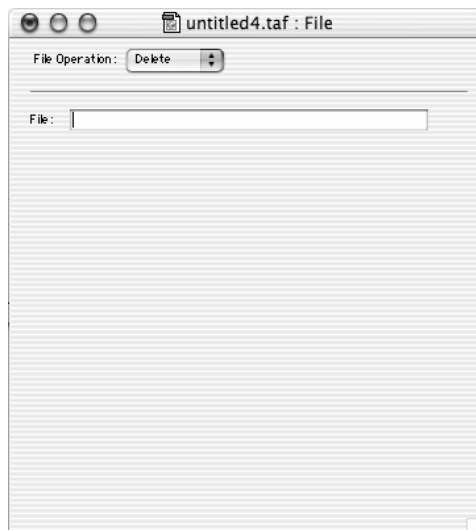


Note The read data is also available as `<@COL 1>` in the action's Results HTML and is stored in the local `resultSet` variable.

Setting Up Delete Options

From the **File Operation** drop-down menu, select **Delete**.

The action editing window changes to show the options for the **Delete** type operation.



In the **File** field, specify the full, absolute path, and file name of the file to delete.



Note The path specified here is appended to the value of the `absolutePathPrefix` configuration variable. If this configuration variable has a value (in either application or system scope), this field should contain a path *relative* to that location.

Handling File Security

You can specify that file reads, writes, and deletes only take place in a specified directory of Witango Server using the `absolutePathPrefix` configuration variable. Using this configuration variable to set the path for file reads, writes, and deletes ensures that users cannot access directories other than the specified ones when using the File action.

File reads, writes, and deletes are enabled in Witango by default. If you want to disable (or enable) these features, you can do so by changing the following options in the Witango Configuration Manager (the `config.taf` application file), in the **Feature Switches** screen:

```
fileReadSwitch
fileWriteSwitch
fileDeleteSwitch
```

Check or uncheck the check box beside the option.



Note The `fileReadSwitch` configuration variable also applies to the `<@INCLUDE>` meta tag.

Using Advanced Database Actions

Transaction and Direct DBMS Actions, and Joining of Database Tables

You can put several database actions together to create a single *transaction* that manages the work being performed. Using *Begin Transaction* and *End Transaction* actions you can specify where to begin, commit, and rollback database changes.

You can use the *Direct DBMS* action to execute specified SQL statements and return any results generated.

Relational databases let you specify joins to permit searches involving more than one table. A *join* tells the database how the tables are related. A *standard join* preserves only those rows from a search in which a match exists with the joined table. An *outer join* preserves all the rows in one of the tables, even if there is no match with the other table.

The topics covered in this chapter include:

- setting up and executing transaction actions
- setting up and executing the Direct DBMS action
- understanding joins
- creating and editing joins.

Using Database Transactions

Witango supports special database actions that allow you to specify where to begin, commit, and rollback database changes. Using the Begin Transaction and End Transaction actions, you can create a well-defined single transaction.

Normally, actions executed by Witango Server that change the content of databases (Insert, Update, Delete, and Direct DBMS actions) cause an immediate change to the database. This is because Witango automatically sends a `COMMIT` command as the final step in its execution of these actions.

However, transaction actions let you control when database changes are made permanent and also let you undo (or `ROLLBACK`) the effects of actions that have been executed.

To perform a transaction action, Witango maintains a database connection longer than it would for other actions. You should consider the impact this may have on your server and database resources before deciding to use transactions in your application file.



Note Witango transaction actions have no effect on databases that do not support transactions.

Setting Up a Transaction Action



Begin Transaction

Begin Transaction

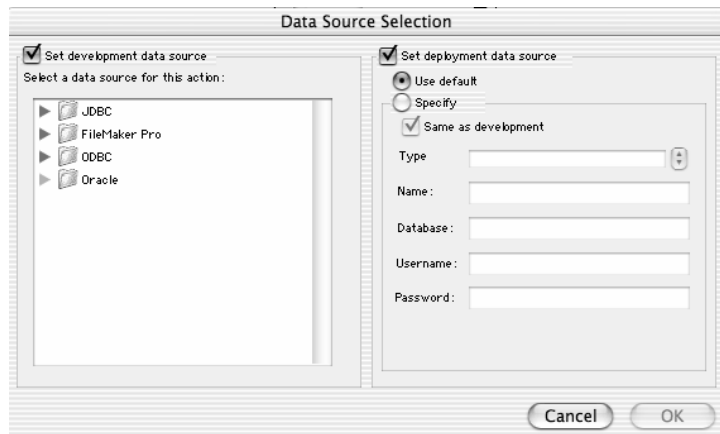
The Begin Transaction action indicates the beginning of a transaction on a particular data source.

To set up a Begin Transaction action

- I Drag the Begin Transaction action icon from the Actions bar into an application file.

For more information, See
“Setting Data Sources for
Actions” on page 106.

The Data Source Selection dialog box appears:



2 Select a data source and click **OK**.

If username and password are required for this data source, a dialog box appears, where you can enter your username and password:



3 Enter your Username and Password into the respective fields, and click **OK**.

For more information, see
“Connecting to Large Data
Sources” on page 113.

If there are more than the maximum number of tables in the data source, and this is the first time you have accessed this data source in this Witango Studio session, the Select Tables dialog box appears, allowing you to choose which tables you want visible in the data source. Select the tables and click **OK**.

The Begin Transaction action dialog box becomes active:



- 4 Click a radio button to select the isolation level you want to assign to the Begin Transaction action.
 - **Read/Write exclusive.** Locks rows that are read as part of the transaction until a `COMMIT` or `ROLLBACK` command is issued to the database server.
 - **Read uncommitted.** Reads rows that have been changed by other database users in a transaction, but for which the transaction has not been committed or rolled back.
- 5 Click anywhere outside the Begin Transaction dialog box to close it.

End Transaction

The End Transaction action marks the end of the transaction and either commits it (saves all the changes) or rolls it back (discards all the changes).

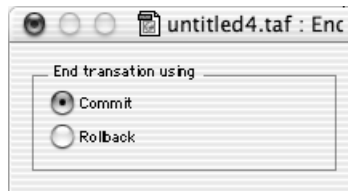
To set up an End Transaction action



End Transaction

- 1 Drag the End Transaction action icon from the Actions bar into an application file.

The End Transaction action dialog box appears:



- 2 Click a radio button to select the option you want to the End Transaction action:



Commit
End
Transaction



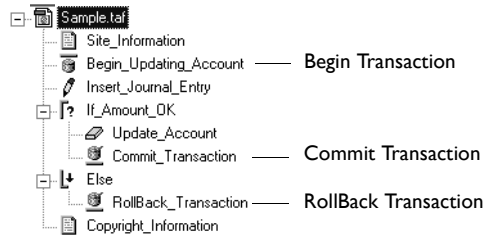
RollBack
End
Transaction

- **Commit.** Commits any modifications made to the database during the current transaction.
- **Rollback.** Undoes any modifications made to the database during the transaction.

In the application file, the End Transaction action icon changes to reflect the associated attribute.

- 3 Click anywhere outside the End Transaction dialog box to close it.

The following is an example of valid transaction actions appearing in an application file.



Executing a Transaction Action

If Witango Server detects that an End Transaction action is being executed without first executing a Begin Transaction action, it reports a runtime error. It is also an error to begin another transaction before an existing transaction is committed or rolled back.

Database actions on data sources that are not the transaction data source are automatically committed.

If the application file ends without executing an associated End Transaction action or a Return action, then a Rollback End Transaction action executes automatically.



Tip When executing a transaction, your application could slow down; additional RAM may be required for Witango Server.

Using SQL Directly

The Direct DBMS action executes specified SQL statements and returns any results generated.

Setting Up a Direct DBMS Action



Direct DBMS Action

To set up a Direct DBMS action

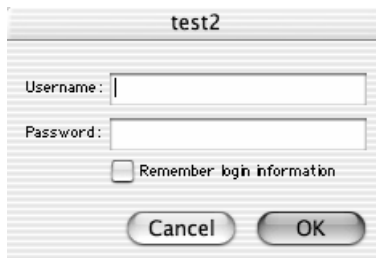
- 1 Drag the Direct DBMS action icon from the Actions bar into an application file.

The Data Source Selection dialog box appears:



For more information, See "Setting Data Sources for Actions" on page 106.

- 2 Select a data source.
- 3 A dialog box appears where you can enter a username and password, if necessary.



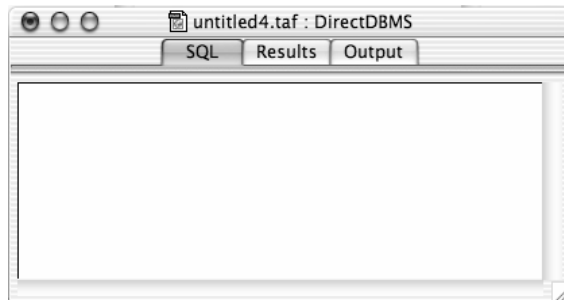
Enter the Username and Password and click **OK**.

For more information, see “Connecting to Large Data Sources” on page 113.

If there are more than the maximum number of tables in the data source, and this is the first time you have accessed this data source in this Witango Studio session, the Select Tables dialog box appears, allowing you to select which tables you want visible in the data source.

4 Click **OK**.

An empty Direct DBMS action editing window appears, displaying SQL and Results tabs.



Fill in the tabbed windows as described next.

The Direct DBMS Action Editing Window

For information on constructing SQL statements, consult your database or ODBC driver documentation.

The Direct DBMS action editing window consists of two sections: SQL and Results.

SQL Section

Click the SQL tab to display the SQL section. You can enter SQL statements in the text area for execution.

All statements are executed against the database specified in the data source associated with the Direct DBMS action.

You can easily enter column or table names by dragging them from the Data Sources Workspace. If you drag multiple columns, they are separated with commas.

You can also perform standard editing operations in the Direct DBMS action editing window by one of the following methods:

- From the **Edit** menu, choose the editing command you want.
- Click the appropriate editing icon on the main toolbar.
- Right-click to display a context-sensitive menu of commands.

You can reference any value-returning Witango meta tags in your SQL.

To insert a meta tag in the Direct DBMS window

- 1 Click the SQL text area of the editing window where you want to enter a meta tag.
- 2 Do one of the following:
 - From the **Edit** menu, choose **Insert Meta Tag**.
 - Right-click the action editing window, and choose **Insert Meta Tag** from the context-sensitive menu that appears.

The Insert Meta Tag dialog box appears, allowing you to specify a meta tag, and inserts it at the insertion point in the SQL text area.

For information on filling in the Insert Meta Tag dialog box, see Inserting Meta Tags on page 148.

You can use the `<@IF>`, `<@IFEQUAL>`, and `<@IFEMPTY>` meta tags in the SQL text to include or exclude SQL based on the result of a comparison at execution time. For example, you could use this capability to execute different SQL based on the type of data source in use.

Direct DBMS SQL Auto-Encoding

Witango Server automatically performs SQL encoding on meta tag values substituted in Direct DBMS SQL. For example, if a variable called `myName` contains "O'Brien":

```
SELECT * FROM customer WHERE cust_name = '<@VAR
NAME=myName>'
```

This results in:

```
SELECT * FROM customer WHERE cust_name = 'O''Brien'
```

If Witango did not do this, the result would be:

```
SELECT * FROM customer WHERE cust_name = 'O'Brien'
```

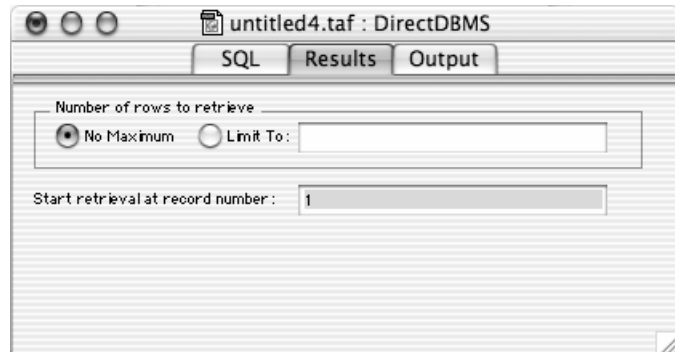
and a DBMS error would result due to the unescaped quote.

If your Direct DBMS SQL contains meta tags that evaluate to an entire (or partial) SQL statement constructed elsewhere, the quote-doubling may cause DBMS errors. This is because all single quotes are doubled, even those meant to delimit a string. In this case, the solution is to modify the meta tag(s) returning your SQL by adding the `ENCODING=none` attribute. For example:

```
<@VAR NAME=mySQL ENCODING=none>
```


Results Section

Click the Results tab to display the results options you can set for the Direct DBMS action.



You can specify options for the maximum number of records to retrieve from the data source and at which result record number retrieval begins.

Number of rows to retrieve

- To return all matching rows, select **No Maximum**.
- To limit how many records are returned by the action, select **Limit To** and enter the maximum number of rows to retrieve.

Start retrieval at match number

Use this option to skip some of the matching records. Enter "1" (the default) to start retrieval with the first matching record. When a value other than "1" is entered into this field, the Direct DBMS action returns records starting at that number, skipping any records before it.

Both of these fields can contain meta tags which return values.

Executing a Direct DBMS Action

When Witango Server executes a Direct DBMS action, the specified SQL is sent to the data source for execution.



The result rows are returned to Witango and may be accessed in the action's Results HTML. As with the Search action, a `<@ROWS>` `</ROWS>` block is used to iterate through the records returned. You must specify column references differently, however.

You can use `<@COLUMN>` to refer to your columns by name for ODBC data sources, but for non-ODBC data sources, you must refer to your columns in Results HTML by number, using the `<@COL>` meta tag.

For example, if your Direct DBMS action executed the following statements with a non-ODBC data source:

```
SELECT maintable.price, maintable.classification,
       maintable.manufacturer
FROM maintable
```

the following Results HTML would print the database results:

```
<@ROWS>
maintable.price: <@COL 1><BR>
maintable.classification: <@COL 2><BR>
maintable.manufacturer: <@COL 3><BR><HR>
</ROWS>
```

When you perform an SQL query where you are selecting an aggregate function or calculated column, the column name depends on the database:

Database	Return Value	Comments
Access	Expr100x...	Returns Expr1000, Expr1001, and so on, for each column with an expression.
SQL Server	blank	No column name.

Database	Return Value	Comments
Oracle (OCI)	blank	No column name.
Oracle (ODBC)	Expression	Returns the expression as the column name; for example, if the column took the maximum of a list of prices using MAX(price), the column name is MAX (price).

If the Direct DBMS action generates no database results, and you have specified No Results HTML for the action, that HTML is processed instead of the Results HTML.

You can use bound values in SQL by using the <@BIND> meta tag. When calling a stored SQL server procedure from a Direct DBMS action with an ODBC data source, you should use the following syntax:

```
{ call procedureName (param1 , param2 , paramX) }
```

Joining Database Tables

For more information on joins, consult your DBMS documentation, such as,

**T H E
P R A C T I C A
L S Q L**

H A N D B O O

K (J.S. Bowman, *E T* .

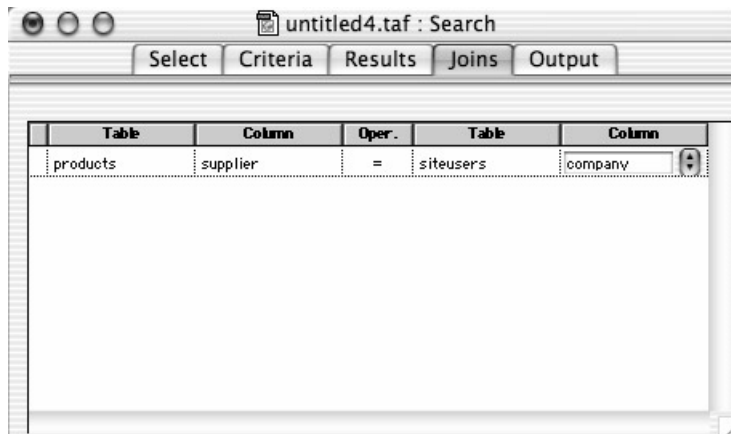
A L ., ISBN: 0-201-62623-3), or any other good SQL reference guide.

For more information, see “Creating a Join in a Search Action” on page 338.

To understand how joins work, consider a database with vendor and associated account manager information in two different tables. You want to create a search to find the account manager for any given vendor, and display in your browser the vendor information with the corresponding account manager’s name. Because the vendor table contains only the account manager’s identifier, you have to *join* the two tables to get the account manager’s full name.

The vendor table (VENDOR) has a record for each vendor including a vendor identifier, name, contact information, payment terms, and an account manager identifier. The account manager table (ACCTMGR) has a record for each account manager including the account manager’s identifier, name, and telephone number. The vendor table is related to the account manager table by an identifier in the AcctMgr column; the account manager table has a MgrID column that contains the identifier corresponding to the AcctMgr column in the vendor table.

Using a Witango search, you select the columns you want to relate and define the type of join in the Joins section of the Search action or Search Builder.



In addition to the standard join, you can define an *outer join*, which can be left or right. A *left outer join* means all rows in the left-specified table are returned, including those with no match in the right-specified table. A *right outer join* means all rows in the right-specified table are returned, including those with no match in the left-specified table.

For this example, you would select the MgrID column in the left table, ACCTMGR, and the AcctMgr column in the right table, VENDOR. Then, from the drop-down menu you select the type of join you want to use.

- If you select a *standard join* (=), the search returns *only* rows of vendor information where a valid account manager's identifier is found. If none is found, the corresponding row is not returned. For instance, if a vendor has not been assigned an account manager and thus the MgrID column is blank for that record, that vendor is not returned.

Vendor ID	Name	Account Manager	City	State
1	ACME Accessories	Erin Shanahan	Groton	MA
2	Norris Corporation	Andreas Franck	Groton	MA
4	Smith & Waterman Assoc.	Wendell Ainsworth	Foster City	CA
5	West End Corp.	Paula Jason	Cheylen	WV
6	Worldwide Posters Unlimited	Andreas Franck	Eugene	OR

Only rows with matching account managers are returned.

- In contrast, if you define a *right outer join* (=*), the search returns *all* rows of vendor information, regardless of whether an account manager is found or not.

Vendor ID	Name	Account Manager	City	State
1	ACME Accessories	Erin Shanahan	Groton	MA
2	Norris Corporation	Andreas Franck	Groton	MA
3	PrintCo		Carson City	NV
4	Smith & Waterman Assoc.	Wendell Ainsworth	Foster City	CA
5	West End Corp.	Paula Jason	Cheylen	WV
6	Worldwide Posters Unlimited	Andreas Franck	Eugene	OR

In this case, the row is returned even though no matching account manager is found.

- A *left outer join* (*=) returns rows of vendor information based on the account managers found, including any account managers without vendors assigned.

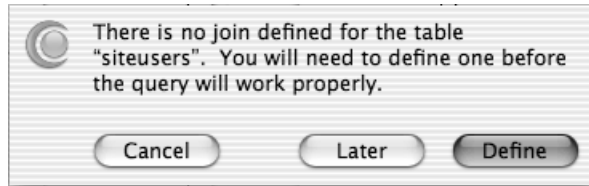
Vendor ID	Name	Account Manager	City	State
0		Jamie Frame		
1	ACME Accessories	Erin Shanahan	Groton	MA
2	Norris Corporation	Andreas Franck	Groton	MA
4	Smith & Waterman Assoc.	Wendell Ainsworth	Foster City	CA
5	West End Corp.	Paula Jason	Cheylen	WV
6	Worldwide Posters Unlimited	Andreas Franck	Eugene	OR

In this case, the row is returned even though the account manager found has no matching vendor.

Working With Joins

To work with joins, you must first have your Search action or Search Builder action editing window open.

You can include columns from more than one table in a search, if you define joins for the tables. If you select columns from more than one table in a search, a message appears telling you to define a join.



Choose either **Define** to create the join definition now or **Later** if you want to define the join at a later time. Choosing **Define** opens the Joins section of the current dialog box.

You create, modify, and delete joins using the Joins section of the Search editing window or Search Builder.

When you define the join, it adds the columns to the search. In the Search Builder, you must define the join before you build the actions for the search or save the application file.



Note In earlier versions of Witango, you could get join information by using the **Attribute** menu's **Joins** command or the Joins icon in the Attributes bar. Join information is viewed in the Search action or Search Builder editing window.

Creating a Join in a Search Action

To create a join, drag the columns you want to include in the search from the Data Sources Workspace into the Joins section.

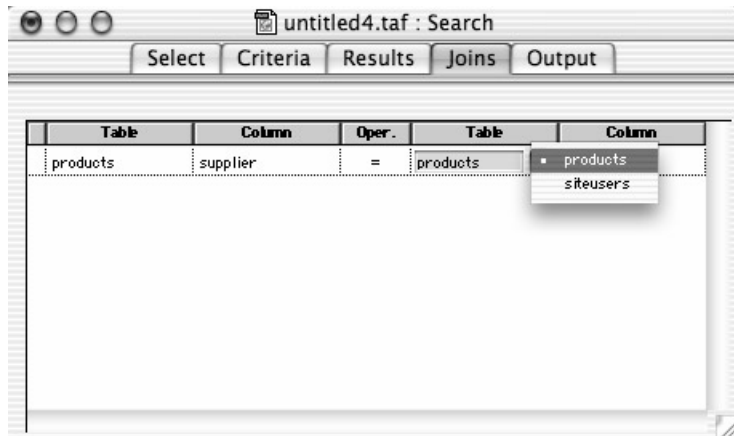


Note You cannot create a join from two different data sources.

To create a new join in a Search action

- I In the Search editing window, click the Joins tab.

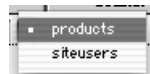
The Joins section appears:



If you added columns from different tables to the **Select Columns** list (under the Select tab), a join definition already appears, showing you the tables selected and the first column added from each table. The default operator is “=”.

If you have not added columns, do so now by dragging the columns into the joins section.

- 2 From the **Table** drop-down menus, select the left and right tables for the join. The following is an example of the drop-down menu:



- 3 From the **Column** lists, select the columns you want to join in each table. A table's first column appears as the default in the list.
- 4 From the **Oper.** drop-down menu, select a join operator.

Join Operator	Description
=	Standard join (the default). Only records matching the join criterion are returned.
*=	Left outer join. All left-table rows are returned, including those with no match in the right table.
=*	Right outer join. All right-table rows are returned, including those with no match in the left table.

Inserting a Join *To insert a join definition*

Do one of the following:

- Click in the list area of the Joins section. From the **Edit** menu, choose **Insert**.
- Right-click the Joins section and choose **Insert** from the context-sensitive menu that appears.

Editing a Join *To edit a join definition*

- 1 Click the Joins tab in the Search action editing window.

The Joins section appears, showing you the current join definition(s) including table names, joined columns, and join operator.

- 2 Do one of the following to edit a definition field:

- Click the field twice.
- Right-click and choose **Edit** from the context-sensitive menu that appears.

The field changes to a drop-down menu so you can choose a different entry.

Deleting a Join *To delete a join definition*

- 1 Click the Joins tab in the Search action editing window.

The Joins section appears, showing you the current join definition(s) including table names, joined columns, and join operator.

- 2 Select the join definition you want to delete, and do one of the following:

- From the **Edit** menu, choose Delete.
- On the main toolbar, click the Delete icon.
- Press DELETE on your keyboard.
- Right-click the selected join definition(s) and choose **Delete** from the context-sensitive menu that appears.



A message appears asking you to confirm that you want to delete the selected row(s).



- 3 Click **Yes** to delete the selected rows or **No** to cancel.

Tip You can bypass the confirmation dialog box by holding down the

You can SHIFT-click (contiguous rows) or CTRL-click (discontiguous rows) to select multiple join definitions.

Ctrl key when choosing **Delete**.



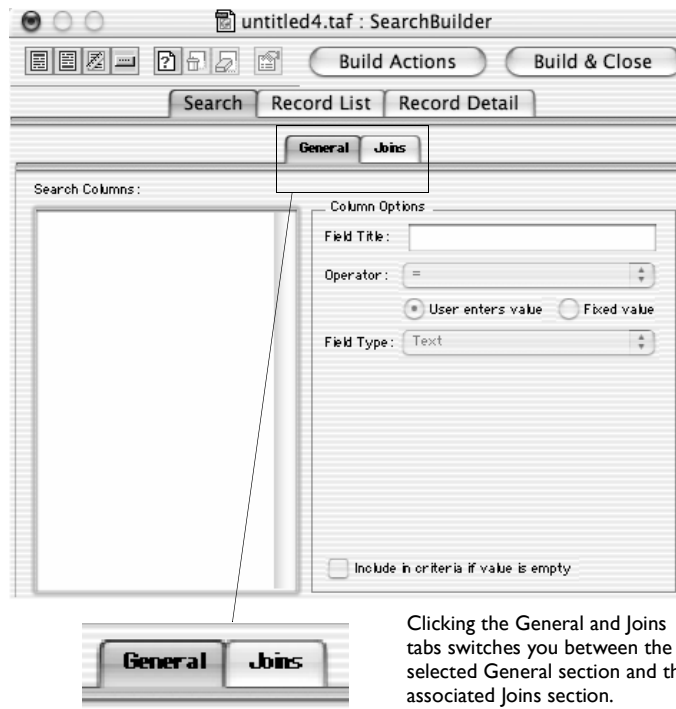
Note If your Search action refers to columns from the deleted joined table, you need to remove these columns and references from the action or builder window manually.

Creating a Join in the Search Builder

You create, edit, and delete joins in the Joins section the same way you do for a Search action. See *Creating a Join in a Search Action* on page 338.

The Search and Record List pages of the Search Builder share the same join information because they both apply to the same generated Search action. You can specify separate join information on the Record Detail page.

You switch from the Search, Record List, or Record Detail section to the associated Joins section by clicking the General and Joins tabs, respectively, at the bottom of the Search Builder window.



Note You cannot create a join between two different data sources.

SECTION V

Witango and Objects

How to Use Objects and Create Objects (Witango Class Files)

This section gives details on using objects in Witango.

This section contains chapters on the following topics:

- Chapter 20, Understanding Objects in Witango on page 345
- Chapter 21, Using Objects on page 361
- Chapter 22, Witango Class Files on page 395.

This section is recommended to all users of Witango who are going to add object functionality to their application files or create their own objects as Witango class files.

Understanding Objects in Witango

How Objects Work in Witango

Objects are reusable software components. Witango supports the use of objects in Witango application files. The use of objects can simplify the development process and reduce development time.

Witango supports different object types:

- COM objects
- JavaBeans
- Witango class files.

The topics covered in this chapter include:

- an introduction to objects
- the benefits of using objects in Witango
- an overview of the types of objects supported
- the data types used in Witango.

This chapter covers only the basic concepts of objects in Witango. See the next chapter for the details of using objects in Witango.

What are Objects?

Objects are reusable software components. Each object is generally designed to deal with a specific issue within a programming project. For example, if you have a project that prepares financial statements for customers, you may use an object that calculates the compound interest of an account, and another object that organizes information into a report.

When you develop a Witango application file that requires calculation of compound interest and presentation of reports, you do not necessarily have to write your own code to do the calculation or the presentation; you could simply select and use the appropriate objects that perform those tasks. Using objects can simplify your application development and reduce your development time.

Objects as Black Boxes

Objects are treated as “black boxes”; that is, as a user of objects, you do not need to know the source code inside objects or the programming languages used for writing this code. In fact, unless you are the developer of an object, you have no access to the inside of that black box.

Instead of editing the source code within an object, you interact with the object through an interface.

Example: Interest Calculator

Let us assume there is an object called Interest Calculator, which calculates compound interest. When you use this object in your Witango application, the object calculates compound interest as one of the tasks in the Witango application file. You do not know how Interest Calculator performs this task, nor do you care. You only want to be able to input your principle, interest rate, and period, and get the result.

With the help of the Interest Calculator example, let us introduce the basic elements of an object used in Witango:

- Interest Calculator is an object; it is treated as a black box—you have no access to how the object performs its computations.
- An *object instance* is a specific application of the Interest Calculator object. In general, a separate object instance has to be created from Interest Calculator to handle each customer account. You create an object instance in Witango using a Create Object Instance action or the `<@CREATEOBJECT>` meta tag.

- Interest Calculator has an interface that allows you to input data (principle, interest rate, period) into the object and to get results (interest) from the object. A *method* is a specific way to interface with an object instance. You call a method in Witango using a Call Method action or the `<@CALLMETHOD>` meta tag.
- Each method of Interest Calculator specifies precisely the requirements of the input (for example, what type of data is principle) and output data. These requirements are the *parameters*.

For more information, see
Method Elements:
Parameters on page 348.

Object Interface: Methods

The interface of an object consists of one or more methods. A *method* allows you to tell the object to input data, get data, or carry out any other action.

Each object includes at least one method; otherwise, you cannot use the object. An object may include several methods. For example, the Interest Calculator object (See “Objects as Black Boxes” on page 346.) includes a method which gives you the interest; it may include another method which gives you the total payment amount.

When you use an object, you need to choose a method of the object you want to use.

While an object itself is a black box, the methods of an object are visible to users. Through a process called introspection, Witango discovers the methods and properties of the objects you want to use; it then presents this information to you. For each object, you can view a list of its methods and their associated parameters (see Method Elements: Parameters on page 348) in the Objects Workspace. You can view additional object information in the Object Properties window.

Viewing object information that Witango generates through introspection merely displays a list of what methods are available for your use; it does not give you all the detailed information about the objects. Objects may come with documentation from the vendors of these objects. You need to find out from this documentation whether a particular object suits your purpose and how you might use it.

For more information, see
“Adding a Call Method
Action” on page 383.

You use objects in Witango by incorporating Create Object Instance and Call Method actions into Witango application files. Create Object Instance action and Call Method action are types of Witango actions: when Witango Server executes a Witango application file that contains a Call Method action, it performs the task specified by the method.

You can also create object instances and call methods in Witango by using the `<@CREATEOBJECT>` and `<@CALLMETHOD>` meta tags. However, these meta tags do not show you the methods and parameters of your objects, and are therefore recommended for advanced users of Witango.

Method Elements: Parameters

Parameters are the basic data elements of a method. A parameter defines what the object accepts as an input, output, or both. Each method consists of one or more parameters.

For example, the Interest Calculator object (See “Objects as Black Boxes” on page 346.) includes a method that contains a parameter called *Principle*. You have to input data into *Principle* according to the requirements of this parameter. Another parameter of this method is called *Interest*. The object outputs the result of interest calculation according to the requirements of the *Interest* parameter.

Class, Object, and Object Instance

For more information, see “Object Types Supported in Witango” on page 355.

In theory, a *class* is a category of objects. A class defines all the common properties of the different objects that belong to it. In the computer industry, however, the term “object” is often used loosely. What is called an object may in fact be a category of objects.

Many of the “objects” available on the market—such as COM objects and JavaBeans—are, strictly speaking, classes. Witango class files, which you can create using Witango Studio, are also classes.

Witango Server does not execute a class directly; instead, it executes an object instance of a class. An *object instance* is an application of an “object” in the real world. You can create multiple object instances based on the same “object.”

The example below illustrates the relationship between a class (“object”) and an object instance:

Interest Calculator (See “Objects as Black Boxes” on page 346.) can be seen as a class. It is a generic interest calculator, in the sense that it does not calculate interest for any particular customer; it can potentially perform similar tasks for many customers. When your Witango application file asks Interest Calculator to calculate interest for John Smith, Witango Server creates an object instance of Interest Calculator for the John Smith account. Similarly, Witango Server creates an object instance of Interest Calculator for the Mary Brown account, when it is needed.

The object instance created for the John Smith account has nothing to do with the object instance created for the Mary Brown account; yet both object instances are based on the common properties of Interest Calculator.

In the computer industry, an object instance is often also called an “object”.



Note While Witango attempts to use an unambiguous terminology whenever possible, it is not always practical to avoid terms that are already widely used in the computer industry. The following is a brief description of what “object” may mean in Witango, depending on context:

- a class (a category of objects with the same behavior)
- an object instance (a particular use of an object in the real world)
- a data type used with objects.

Creating Object Instances

For more information, see “Using Available Object Instances” on page 350.

For more information, see “Working With Variables” on page 155.

Before you can use an object in Witango, an object instance has to be available for use. In general, this means you have to create the object instance first.

This section explains how you can create an object instance in the same application file that you use it. There are circumstances when you do not have to create an object instance in the same application file before using it.

In your Witango application file, use a Create Object Instance action to create an object instance from an object that Witango supports.

Object instances are stored in Witango variables. The scope of the variable determines where the object instance is available. For example, `local` scope applies to the current execution, and `user` scope applies to all the Witango application files executed by the current Witango user.

An object instance is destroyed when the variable in which it is stored expires or is purged, using the `<@PURGE>` meta tag. .

You can also use the `<@CREATEOBJECT>` meta tag to create an object instance.

In some cases, you may want to create more than one object instance from the same object. The following example shows such a case:

You use the Interest Calculator object to calculate interest for various customer accounts. In your application file, you already have an object instance of Interest Calculator that calculates interest for John Smith, and now you need to calculate interest for Mary Brown. Because the two accounts are not related, you do not want to use the existing instance (with John Smith’s data) for Mary Brown.

In this scenario, you create a new object instance of Interest Calculator for the Mary Brown account.

Using Available Object Instances

For more information, see “Creating Object Instances” on page 349.

There are several ways in which object instances can be available for use in your application file:

- An object instance has been created for this application file, using the Create Object Instance action.
- An object instance has been created for a different application file and, during execution of the current application file, the variable in which that object instance is stored has not yet expired.

For example, application files A and B are running simultaneously under the same username. File A created an object instance stored in the `user` scope. File B can access the object instance that has already been created by file A.

For more information, see Calling Methods page 350 on this page.

- An object instance has been created as a result or an output of a previous Call Method action, provided that the variable in which that object instance is stored has not yet expired.

Calling Methods

After creating an object instance or finding one that is available for use in your application file, you can select a method from this object instance and call the method using a Call Method action.

You can also use the `<@CALLMETHOD>` meta tag to call methods on objects.

An object typically consists of a number of methods in its interface. You can call more than one method from the same object instance, and you can call the same method several times in the same application file. Every time you call a method, you need to use a Call Method action. The following example shows how you might use several methods from the same object instance:

You use the Interest Calculator object to calculate interest for John Smith. The first Call Method action is `SetPrinciple`, and you set the principle to \$100. The second Call Method action is `SetInterestRate`, and you set the interest rate to 5%. Your second Call Method action must use the previous object instance of the Interest Calculator object; otherwise, Witango does not know that the \$100 and the 5% refer to the same account. The third Call Method action is `GetBalance`. This action must also use the previous object instance of the Interest Calculator object; otherwise, Witango does not get the balance from the John Smith account.

In this scenario, every time you use a Call Method action, you refer to the same object instance, that is, the one set up for the John Smith account.

The following example shows how you might use the same methods in the same application file, but with different object instances:

Let us say you now want to perform interest calculation for both the John Smith account and the Mary Brown account in the same application file. Because you do not want to mix up the two accounts, you have to create two separate object instances for the two accounts before you use the Call Method actions.

A Call Method action may return a result or an output. In some cases, the result or output is another object instance, which you may then use with another Call Method action.

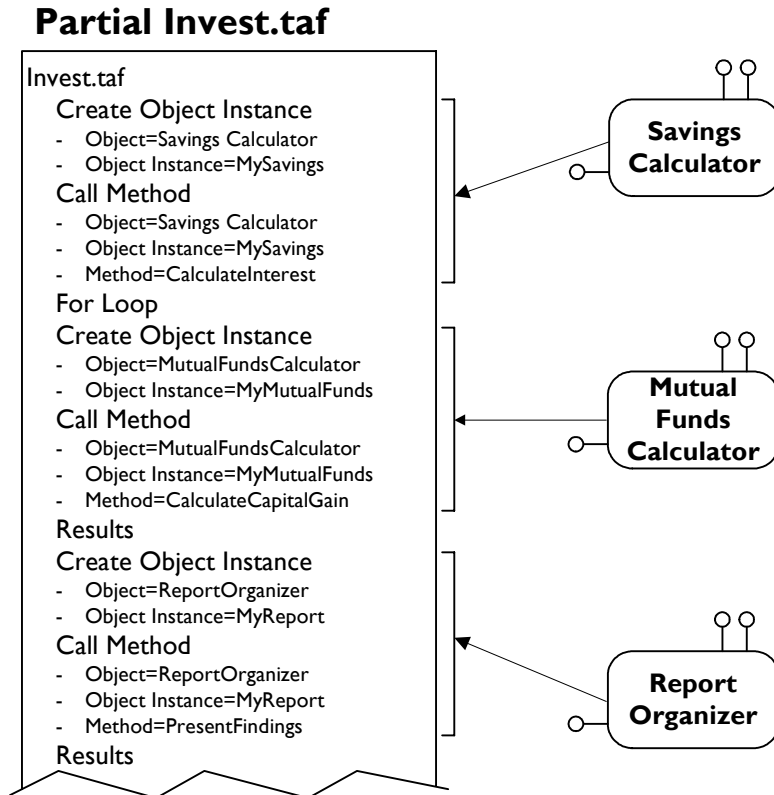
Example I: Investment Scenarios

Let us assume you are developing a Witango application file (`Invest.taf`) for a financial institution that advises customers on retirement investment strategies. You want to present scenarios to customers based on different mixes of savings and mutual funds.

You can use an object that calculates accumulated savings (Savings Calculator), another object that projects the future values of mutual funds (Mutual Funds Calculator), and another object that works out different investment scenarios and organizes the results into a report (Report Organizer).

The parameters of Savings Calculator may include current age, retirement age, monthly contribution, and total savings; the parameters of Mutual Funds Calculator may include monthly contribution, mutual fund category, risk factor, and total value; the parameters of Report Organizer may include savings, mutual funds, and total investments.

The following diagram shows how you might incorporate several objects into your Witango application file (The other actions shown in the application file are entirely arbitrary.):



Example 2: More Investment Scenarios

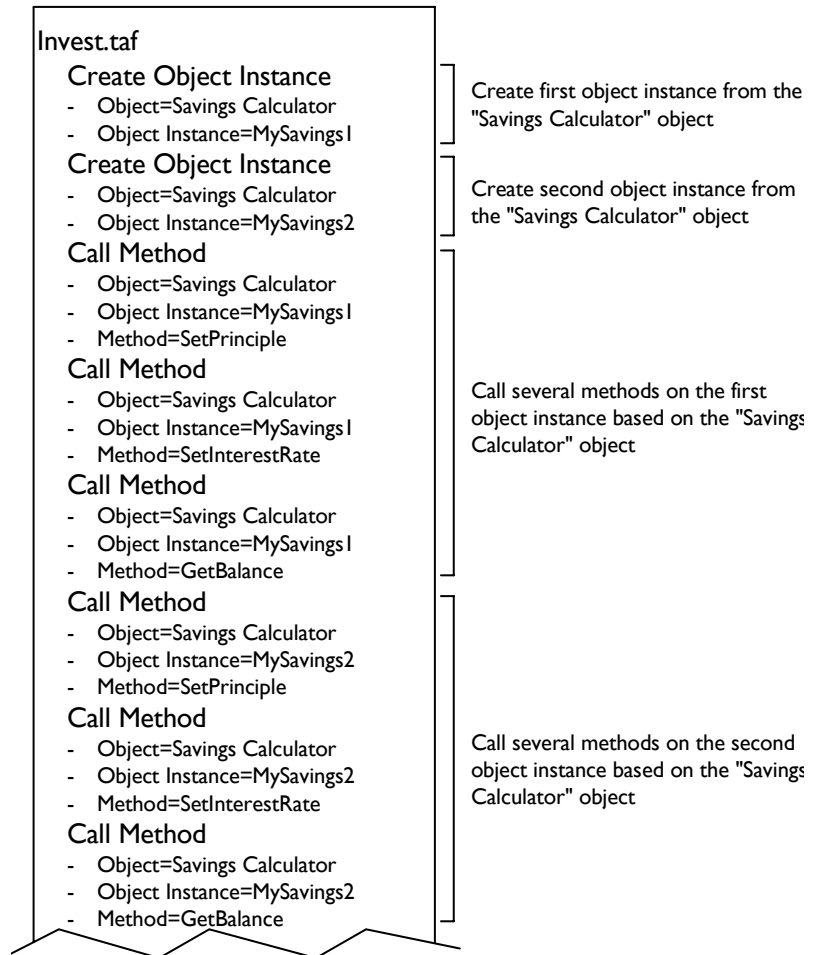
Let us assume that you now want **Invest.taf** the Witango

application file shown in Example 1) to keep track of two separate savings accounts and you want to use three different methods—**SetPrinciple**, **SetInterestRate**, and **GetBalance**—on each of these accounts.

Because both savings accounts behave in the same way, but with different data, you create two separate object instances—**MySavings1** and **MySavings2**—based on the “Savings Calculator” object. When doing your interest calculation for each of these accounts, you call the three different methods on each of the two object instances.

The following diagram describes the logical flow in a part of a Witango application file, which shows how you might incorporate two or more object instances, each with several Call Method actions, into your Witango application file:

Partial Invest.taf



Benefits of Using Objects in Witango

When you use objects in your Witango application files, you obtain a number of benefits:

- Many objects are available for free or for purchase from a variety of vendors. They perform various programming tasks, some of which may be useful to your Witango application files. When you incorporate objects in your Witango application files, you save on development and testing time.
- Many programming tasks are most efficiently performed by writing in particular programming languages, such as C++, Visual Basic, or Java. You do not have to know any of these languages to use objects written in these languages.
- Developers of objects are often experts in their respective fields. When you use objects to develop Witango application files, you draw on their expertise.
- You can develop your own objects in Witango (Witango class files). You can then reuse these objects in your current and future Witango application files.
- You can mix and match different object types supported by Witango: objects available from many vendors and objects you develop yourself can be used in the same Witango application file.
- Because of the modular structure of objects, using objects in your Witango application files helps you organize and maintain these files.
- Developers of objects may modify the source code inside their objects to improve their design. As long as they do not change the interface—which is typically the case—you can benefit from the improved design, without having to alter the code in your Witango application files.
- When you use objects to isolate certain areas of functionality in your application file, it is relatively easy to replace those objects later with other objects, without requiring a great deal of rewriting of the main application file.

When to Use Objects

You can incorporate objects in any Witango application file. In general, the more complex your Witango application file is, the more benefits you can derive from using objects.

Object Types Supported in Witango

Witango supports different types of objects. You can use any of the following types or a combination of them within the same Witango application file:

- COM/DCOM objects
- JavaBeans
- Witango class files.

For more information, see “Class, Object, and Object Instance” on page 348.

Strictly speaking, all “objects” that Witango supports are classes; however, because they are typically called “objects” in the computer industry, Witango Studio and Witango documentation generally conform to the popular usage.



Note Because the method calls are made on the server side by Witango Server, COM objects and JavaBeans that have a user interface are generally not appropriate for use with Witango. A server-side object with user interface elements works with Witango as long as the object works without user interaction.

Object Type Independence

In Witango Studio, you do not have to write code to identify whether the object you are using is a COM object, JavaBean, or Witango class file. Witango takes care of all the implementation details for you. This Witango design feature simplifies the use of objects.

For more information, see “Adding a Call Method Action” on page 383.

When you incorporate objects into your Witango application, you follow a very similar procedure, no matter which object type you are using.

COM Objects

COM (Component Object Model) objects are objects that conform to the COM objects specifications developed by Microsoft. COM objects currently run on the Windows platform only.

Automation Servers

Witango supports the kind of COM objects known as *Automation Servers* (also called Automation Objects). Many Automation Servers, when installed, inform Witango that they are “programmable”. These objects generally offer sufficient information on their methods and parameters to make them useful in Witango application files.

For more information, see “Adding an Object to the Objects Workspace” on page 365.

To help you identify COM objects useful in Witango, by default, Witango gives you a list of available “programmable” Automation Servers when you want to add a COM object. It is recommended that you select COM objects from this list.

Witango can also give you a complete list of available COM objects, if you select the “Show All Objects” option. However, COM objects which are not “programmable” Automation Servers may not respond correctly, if at all.

DCOM Environment

Witango supports the DCOM (Distributed COM) environment. In the DCOM environment, you can deploy COM objects on machines other than the one running Witango Server.

For more information, see “Installing an Object” on page 362.

You set up the DCOM environment by indicating which particular object implementation is available on which machine. Once you have done that, your Witango applications work transparently; you need not be concerned with object locations. You simply build your Witango application files as though all objects are executed on Witango Server.

Licensed COM Objects

Some COM objects require licensing. When a COM object is installed, the license is generally installed with it. This works fine with Witango Studio on the development machine. During deployment, however, Witango Server may or may not be able to validate the license on the deployment machine.

In many cases, the vendor of a licensed COM object generates a license key, which allows the license to be transferred from the development machine to the deployment machine. If the deployment machine has a license already or receives a license key, Witango Server executes the COM object transparently; otherwise, it returns an error message.

For more information, see “Details” on page 375.

When you use Witango Studio, you can find out whether the COM object you plan to use requires a license, has a license, or has a license key. This information is displayed in the Detail section of the Object Properties window.

JavaBeans

JavaBeans are objects that conform to the JavaBeans specifications developed by Sun Microsystems. JavaBeans can run on any platform.

Witango Class Files

Witango class files are objects designed specifically for use in Witango application files.

For more information, see “Witango Class Files” on page 395.

You can create Witango class files in Witango Studio and incorporate them in Witango application files, just like any other objects that Witango supports. Witango class files can run on any platform.

General Requirements

Regardless of object types, only objects that satisfy certain requirements are appropriate for use with Witango:

- objects intended for use in a server environment
- objects that do not present a user interface.

When you call methods in Witango, the objects you use are run on Witango Server, not in the end-user’s browser. Therefore, objects which present user interface elements (for example, windows, buttons, grid controls, and charting applets) are not appropriate.

Of course, you may include objects that present user interface elements in your Witango application files; but to do so, you need to include the appropriate markup in the HTML returned to the user (for example, `<EMBED>`, `<APPLET>`, or `<OBJECT>`). Refer to an HTML reference book for detailed instructions.

Understanding Data Types

Different object types have different specifications for their data types, which are often incompatible with one another. For example, although both COM objects and other object types have a data type called “float”, the specifications are not necessarily the same.

Witango facilitates the use of COM objects, JavaBeans, and Witango class files within the same application file by converting the various data types to the Witango data types, whenever possible. When you use a combination of data types from different object types, you do not have to worry about their compatibility with one another. Witango handles the conversion transparently.

Witango Studio displays native data type names. For example, if the object is a COM object, you see the names of the COM object data types in the Objects Workspace (see page 372) and Call Method Action window. Refer to the respective object vendors for the details of these native data types.

Setting up Security for Executing Objects

For security reasons, you may want to control which objects can be executed by Witango Server. The file that contains the control settings is the object configuration file. The default name of this file is `objects.ini`; this name is user-definable.

During execution of a Witango application file, if Witango Server encounters a Create Object Instance action involving an object that it is not allowed to run, it returns an error.

The control of object execution can take place at the system level (system scope) or at the application level (application scope).

Using Objects

Incorporating Objects in Application Files

You use objects in Witango by adding Create Object Instance and Call Method actions to application files.

You can also use the `<@CREATEOBJECT>` and `<@CALLMETHOD>` meta tags to create and use object instances in Witango. This alternative is recommended for advanced users. For more information, see the *Witango Programmers Guide*.

The basic concepts of objects in Witango are covered in the previous chapter. This chapter focuses on the procedure and details of using objects.

The topics covered in this chapter include:

- preparing to use objects
- adding objects to the Objects Workspace
- viewing object information in the Objects Workspace
- creating object instances
- using the Create Object Instance action window
- calling methods
- using the Call Method action window
- using the Objects Loop action.

Preparing to Use Objects in Witango

Planning to Use an Object

Here are some issues to be aware of before using objects with Witango application files:

- Before you use an object in Witango, you have to know what the object can do for you, what the requirements of using the object are, and whether the object suits your Witango application file. You may have to consider several alternatives to select the object that is the best for your situation.
- Read the documentation on the object, provided by the object vendor. Find out about the methods you can use with the object and the information on parameters. Understanding the parameters helps you set up your Witango application file in order to use the object effectively.
- Make sure the object you plan to use belongs to one of the types supported by Witango : COM object, JavaBean, or Witango class file.

Installing an Object

The following are some guidelines for installing objects before developing and deploying Witango application files:

- It is best to install the objects you plan to use on your development machine. At least, these objects must be visible on the network from your machine.
- Some objects already exist on your machine. If the object you need is not on the machine or visible on the network, you must install it first. Follow the instructions provided by the object vendor.

Local and Remote Machines

The different object types that Witango supports are COM objects, JavaBeans, and Witango class files.

In most cases, you use objects installed on the local machine (same machine as Witango Server). If your deployment machine is not the same as your development machine, you need to install the objects on both machines.

Depending on the object type, you may also be able to use objects installed on remote machines. When deciding which machine to use, consider load balancing and fault tolerance issues.

- COM object

You can access and execute COM objects on both local and remote machines. Use the DCOM environment to deploy COM objects on remote machines.

- **JavaBeans**

Witango uses the CLASSPATH environment variable and the `beanpaths.ini` file, maintained by Witango, to locate JavaBeans.

When using Witango Studio, adding a JavaBean to the Objects Workspace prompts you to add the path to the JavaBean to the `beanpaths.ini` file, if it is not already in the CLASSPATH environment variable or the `beanpaths.ini` file.

When using Witango Server, you must manually add the path to the `.jar` file to the `beanpaths.ini` file, if it is not already in the CLASSPATH environment variable or in the `beanpaths.ini` file on the Witango Server machine.

The `beanpaths.ini` file is located in the root folder of Witango (by default, this is `C:\PVSW\Witango\`).

- **Witango class files**

While you can access Witango class files on both local and remote machines, you are able to execute Witango class files only on the local machine.

When using Witango Studio, use the **Objects** section of the Preferences dialog box to list the search paths on various machines.

When using Witango Server, use the `TCFSearchPath` configuration variable.

For more information, see “Setting Search Paths for Witango Class Files” on page 416 and Objects on page 137.

Overview of Using Objects in Witango

You use an object in Witango by adding Create Object Instance and Call Method actions associated with an object to your Witango application file or Witango class file. The procedure can be broken down into three steps.

The following steps allow you to use the graphical interface of Witango Studio to develop Witango application files with objects. Advanced Witango users may want to use Witango meta tags to create and use object instances. See the *Witango Programmers Guide* for details.

1 Add Objects to the Objects Workspace

For more information, see “Adding an Object to the Objects Workspace” on page 365.

You can only use objects that are visible in the Objects Workspace. If the objects you want to use are not there, you have to add them first.



For more information, see “Viewing Object Information in the Objects Workspace” on page 372.

Note Witango allows you to view object information in the Objects Workspace, if the objects have already been added to the Objects Workspace.

2 Add Create Object Instance Actions

Witango does not use the objects you see in the Objects Workspace directly; it uses instances created from these objects. The instances created from these objects are called “object instances”, or simply “objects”.

For more information, see “Using Available Object Instances” on page 350.

Before you can call a method involving an object in a Witango application file or Witango class file, an object instance must be available.

For more information, see “Adding a Create Object Instance Action” on page 377.

Depending on the circumstances, you may have to create an object instance with a Create Object Instance action.

3 Add Call Method Actions

For more information, see “Adding a Call Method Action” on page 383.

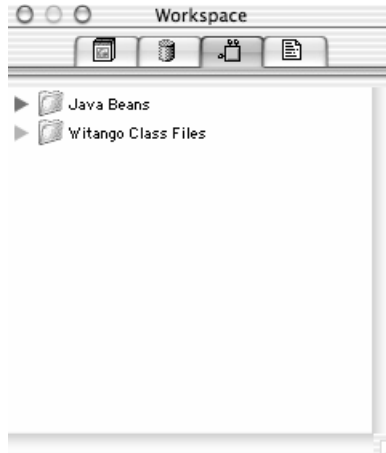
Once an object instance is available, you can incorporate its methods into your Witango application file or Witango class file by adding Call Method actions associated with this object.

Adding an Object to the Objects Workspace

If the Workspace is not visible on your screen, choose **Workspace** from the Windows menu.



To view the Objects Workspace, click the **Objects** tab in the Workspace. The following is an example of the Objects Workspace:



COM Objects in the Objects Workspace

To add a COM object to the Objects Workspace

- I Do one of the following:
 - From the **Object** menu, choose **Add Objects**, then choose **COM/DCOM Objects**.
 - From the Objects Workspace, right-click the COM/DCOM

Objects icon. Choose **Add ...** from the context-sensitive menu that appears.

The Select Object dialog box appears:



You can choose which objects are displayed in this dialog box:

- Showing all COM objects

By default, the dialog box shows a list of the installed Automation Servers. If you want to see a list of *all* the installed COM objects—including Automation Servers—check the **Show All Objects** check box.

- Filtering objects

The **Filter** text box allows you to enter the name or partial name of a COM object. The scroll-down list of the dialog box then shows only the COM objects that match what you have entered.

- Seeing additional information

The **Library** area shows the library to which this object belongs. Objects in a library often work together; they make references to one another. When added, objects in a library are placed in the same folder, generally bearing the name of the library.

The **Details** area gives you additional information about the object you select. For example, it may state the program

For more information, see "Automation Servers" on page 355.

For more information, see Referenced Objects page 367 on this page.

identifier and indicate whether the object is programmable.



Caution If the **Details** area indicates that the object is not programmable, it means the object may not work properly with Witango. It is strongly recommended that you do not use objects described as **not** programmable.

- 1 From the scroll-down list in the dialog box, select the COM object you want to use. (You can select more than one object simultaneously by pressing **Ctrl** or **Shift**.)

2 Click **OK.**

In the Object Workspace, the name of the COM object you have

added appears under the COM/DCOM Objects object type.

Referenced Objects

In some cases, a COM object contains references to one or more related COM objects. You probably need all these objects to work together as a group, called a library. When you add one of these objects, the Add Referenced Objects dialog box appears.

The dialog box asks whether you also want to add the related objects. It is recommended that you click **Yes**, to add the entire group of COM objects. When you add the group, these objects appear together in a folder in the Objects Workspace.

JavaBeans in the Objects Workspace

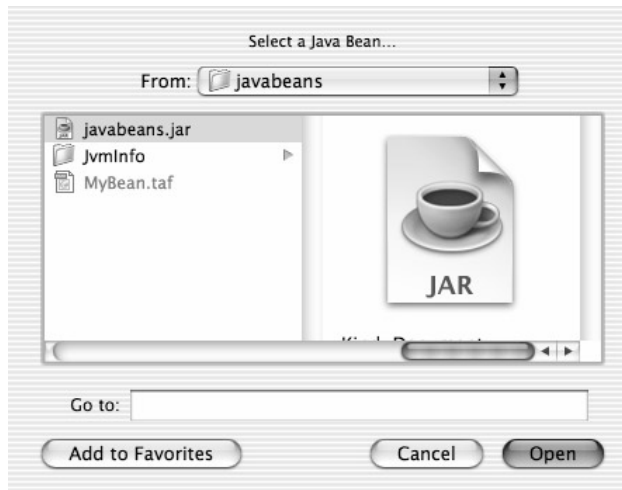
To add a JavaBean to the Objects Workspace

- 1 Do one of the following:
 - From the **Object** menu, choose **Add**, then choose **JavaBeans**.
 - From the Objects Workspace, right-click the

icon. Choose

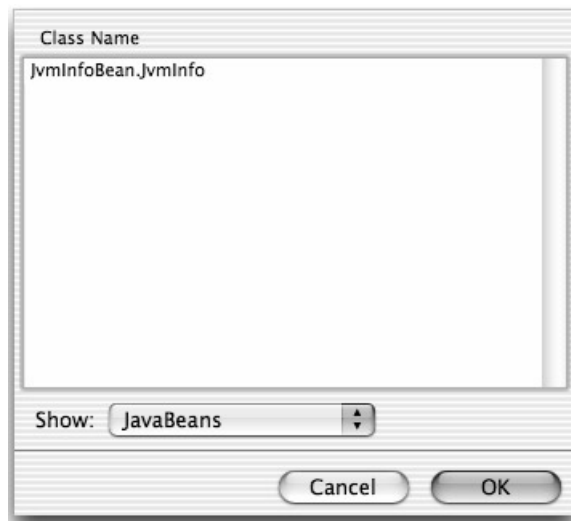
Add... from the context-sensitive menu that appears.

The Select JavaBean(s) dialog box appears:



- 2 Locate the Java archive files (.jar) and select the file you want to use. Click **Open**.

The Select Bean(s) dialog box appears:



The dialog box displays all the JavaBeans found in the Java archive file. You can add one or more JavaBeans from this archive file.

- 3 Select one or more JavaBeans from the list. (To select more than one object simultaneously, press **Ctrl** or **Shift**. You can also select or deselect all the JavaBeans on the list by clicking **Select All** or **Deselect All**, respectively.) Click **Add**.

In the Objects Workspace, the name(s) of the JavaBean(s) you have added appears under the `JavaBeans` object type. The JavaBeans that belong to the same Java archive file (`.jar`) are contained in a folder bearing the name of that Java archive file.



Note Witango Studio uses the `CLASSPATH` environment variable and the `beanpaths.ini` file maintained by Witango for locating JavaBeans. If the path to that JavaBean does not already exist in the `CLASSPATH`, Witango Studio prompts you to add the path to `beanpaths.ini`. Click **Yes** to add the path.

Witango Class Files in the Objects Workspace

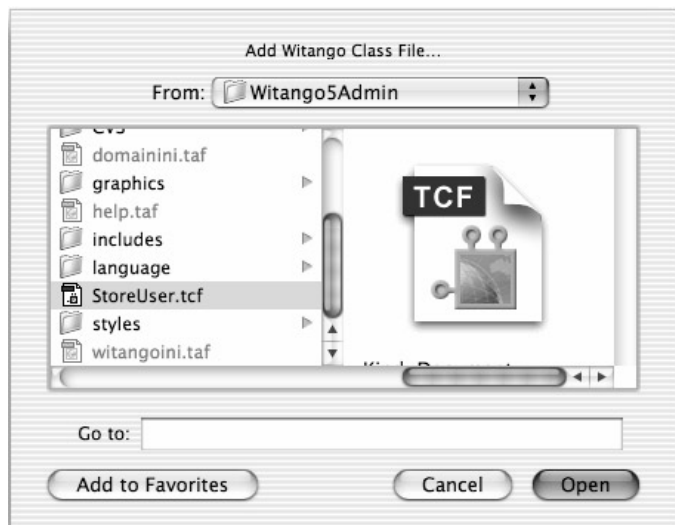
To add a Witango class file to the Objects Workspace

I Do one of the following:

- From the **Object** menu, choose **Add Objects**, then choose **Witango Class Files**.
- From the Objects Workspace, right-click the `Witango Class`

`Files` icon and choose **Add...** from the context-sensitive menu that appears.

The Select Witango Class File(s) dialog box appears:



- 2 Locate the Witango class files (.tcf) in the dialog box and select one or more Witango class files from the list. (To select more than one object simultaneously, press `Ctrl` or `Shift`.) Click **Open**.

In the Objects Workspace, the name(s) of the Witango class file(s) you have added appears under the `Witango Class Files` object type.

Removing an Object From the Objects Workspace

If there are objects in the Objects Workspace that you do not need, you can remove them.

When you remove an object from the Objects Workspace, you are not deleting the object from your machine, just from the Objects Workspace. You can easily add the object to the Objects Workspace again, when you need it.

If you have already incorporated an object in a Witango application file or a Witango class file, removing the object from the Objects Workspace does not affect the application file or the Witango class file.

To remove an object from the Objects Workspace

Do one of the following:

- Select the object you want to remove. From the **Object** menu, choose **Remove**.
- Right-click the object you want to remove, and choose **Remove** from the context-sensitive menu that appears.

Viewing Object Information in the Objects Workspace

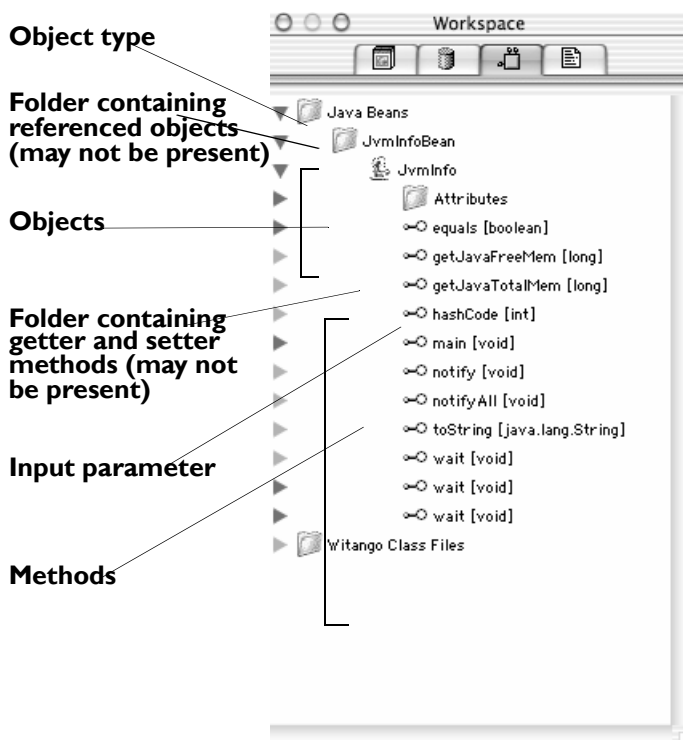
In the Objects Workspace, you can view information on objects by successively expanding items at each level. The information you get from the Objects Workspace includes the following:

- the objects available for each object type
- the methods available for each object
- the parameters for each method
- whether the method returns a result.

For more information, see “Object Properties” on page 374.

Additional object information is available to you in the Object Properties window. For further information about objects, consult the documentation supplied by the respective object vendors.

The following is an example of the Objects Workspace with expanded items:



To view information on an object

- 1 In the Objects Workspace, click the plus sign (+) to the left of an

object type to expand one of the three object types: COM/DCOM Objects, JavaBeans, or Witango class files.

A list of all the objects available for the object type appears.



Note Different object vendors tend to use different icons to represent their objects. Witango generally displays these icons next to the respective object names.

Related objects may be grouped together in a folder. An example of such a folder is a library of COM objects or a Java archive file of JavaBeans.

2 Click the plus sign (+) to the left of an object to expand the object.

A list of all the methods available for the object appears.



Note If the `Attributes` folder exists under this object, click the plus sign (+) to see additional methods in this folder.

3 Click the plus sign (+) to the left of a method to expand the method.

- A list of all the parameters for the method appears. Input, output, and input/output parameters are shown with the following icons:

-  input parameter
-  output parameter
-  input/output parameter

The data type of the parameter is indicated in brackets after the parameter name, for example, `[double]`.

- If the method returns a result, the data type is indicated in brackets after the method name, for example, `[text]`; otherwise, it is indicated as `[none]` or `[void]`.

For more information, see "Parameter List" on page 387.

For more information, see "Result Variable" on page 386.

Attributes Folder

In some cases, the `Attributes` folder appears under an object. This folder contains the *getter* and *setter* methods of the object.

The *getter* and *setter* methods let you get and set attributes (also known as properties or data members). Unlike most methods of an object, *getter* and *setter* methods are very simple methods:

- Getter method**

A *getter* method returns the value of an attribute from the object. An example of this is `GetBalance`.

- Setter method

A setter method lets you set the value of an object attribute. An example of this is `SetBalance`.



Note If the object vendor does not classify these methods as attributes, Witango does not place them in the `Attributes` folder.

Object Properties

The Object Properties window displays a summary of the information about the object you select. You cannot change the information in this window.



Note Much information about objects is available in the Objects Workspace. See Viewing Object Information in the Objects Workspace on page 372.

To view object properties

- 1 Right-click an object in the Objects Workspace.
- 2 Choose **Properties** from the context-sensitive menu that appears.
- 3 From the Object Properties window, select the **General** tab or the **Details** tab.

The General section or the Details section appears.

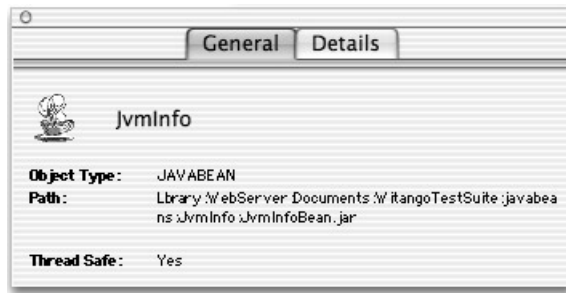
General

The General section of the Object Properties window displays the following information:

Item	Description
Object type icon	This icon identifies the type of object as JavaBean, or Witango class file. The icon may be provided by the object vendor.
Object name	This field, located to the right of the icon, identifies the name of the object.
Object type	This field identifies the type of object as JavaBean, or Witango class file.
Path	This field shows the location of the object.

Item	Description
Thread safe	This field indicates whether the object is thread safe. Witango Server can execute several Witango application files simultaneously. Depending on its design, an object may or may not be thread safe. An object instance that is not thread safe may interfere with the execution of another Witango application file using the same instance. If this field indicates Yes (thread safe), Witango Server executes multiple calls to the instance simultaneously. If the field indicates No (not thread safe), Witango Server waits for one execution of the instance to complete before it starts another, to avoid any interference between the two. From the user's perspective, the difference is in performance only.

The following is an example of the General section (in this case, the object is a JavaBean object):



Details

The Details section of the Object Properties window displays attributes and their associated values. The attributes displayed in this section depend on the object type and the object. For a detailed description of the attributes, refer to the documentation from the object vendor.

Caching and Refreshing of Object Information

When you add an object to the Objects Workspace, Witango stores the introspection information about the object—including methods and parameters—in the `\Witango\Cache\ Object Cache` file in the `Support Files` folder within the Witango folder.

The cache offers you some advantages:

- Because Witango stores the object and introspection information in the cache, it can access this information faster.
- The cache allows you to work away from your network or in a location where objects are unavailable.

Developers of objects sometimes improve the design of their objects. If you want to update the object information in the cache, you can refresh the objects as follows:

- 1 From the Objects Workspace, select the object or objects you want to update.
- 2 Do one of the following:
 - From the Object menu, choose **Refresh**.
 - Right-click the object(s), and choose **Refresh Object(s)** or **Refresh All** from the context-sensitive menu that appears.



Note Refreshing does not update any actions that are already using the object(s).

Adding a Create Object Instance Action

For more information, see “Adding an Object to the Objects Workspace” on page 365.

Before you add a Create Object Instance action to your Witango application file or Witango class file, make sure the object from which you want to create an object instance is available in the Objects Workspace.

A Create Object Instance action is a Witango action. The action icon is available on the Actions bar.

The procedure for adding a Create Object Instance action is the same whether you are working with a COM object, JavaBean, or Witango class file.

To add a Create Object Instance action



- 1 Create or open the Witango application file or Witango class file in which you want to create an object instance.
- 2 Drag the Create Object Instance action icon from the Actions bar to the location you want in your application file or Witango class file.

The Create Object Instance action window appears. The field next to the Create Object Instance action icon displays a message: “Drag an object from the Workspace.”

- 3 Click the **Objects** tab in the Workspace to view the Objects Workspace.
- 4 Expand one of the object types—COM/DCOM Objects, JavaBeans, or Witango class files—that you want to view, by clicking the plus sign (+) to the left of this object type.

A list of all objects of this type appears.

- 5 Select the object from which you want to create an object instance and drag it to the Create Object Instance action window.

The Create Object Instance action window displays the name of the object. An example of the Create Object Instance action window is shown on page 379.

- 6 Complete the information in the Create Object Instance action window.



Note You can skip this step for the time being. At a later time, open this action item and complete the information.

- 7 Close the Create Object Instance action window.

For more information, see “Completing the Create Object Instance Action” on page 379

A Create Object Instance action appears in your application file or Witango class file. The default name for the action is `Create_Object_Instance`.

For more information, see “Working With Actions” on page 229.

- 8 If you want, you can change the default name to whatever is appropriate in your case.

Shortcut to Adding a Create Object Instance Action

An alternative approach to adding a Create Object Instance action from the Actions bar is to do it directly from the Objects Workspace. This procedure is a slight modification of the one described on page 365.

Skip the step involving the Create Object Instance action icon. Just drag the object you want to use from the Objects Workspace to the location you want in the application file or Witango class file.

Completing the Create Object Instance Action

The Create Object Instance action window contains important information about a Create Object Instance action. It allows you to view and edit this information.

You can open the Create Object Instance action window by doing one of the following:

For more information, see “Adding a Create Object Instance Action” on page 377.

For more information, see “Shortcut to Adding a Create Object Instance Action” on page 378.

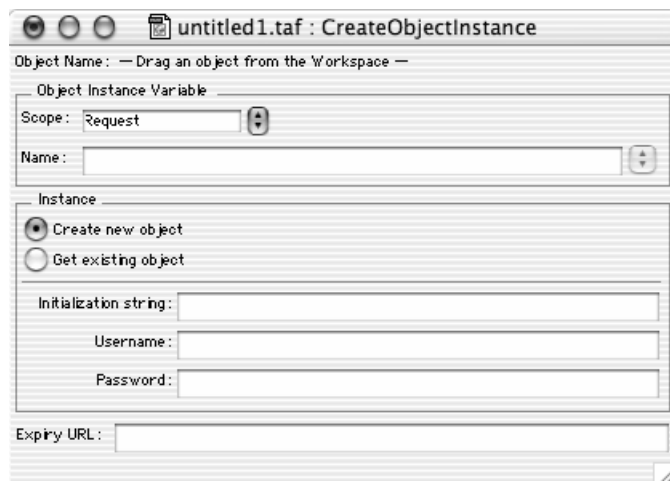
- dragging the Create Object Instance action icon from the Actions bar to an application file (when you create a new Create Object Instance action)
- dragging an object from the Objects Workspace to an application file (when you create a new Create Object Instance action)
- double-clicking the name of a Create Object Instance action in your application file (when you edit an existing Create Object Instance action).

The title of the Create Object Instance action window consists of the name of your Witango application file, followed by the name of the action, separated by a colon.

For more information, see “Working With Actions” on page 229.

The default name of the Create Object Instance action is `Create_Object_Instance`. You can change it in the application file window.

The following is an example of the Create Object Instance action window:



There are several sections to the Create Object Instance action window: object name, object instance variable, instance, and expiry URL.

Object Name

This is the field next to the Create Object Instance action icon. The object name is the name of the object from which you create the object instance. If no object has been assigned to this action, the field displays the message *Drag an object from the Workspace*.

Object Instance Variable

This section sets the name and the scope of the variable that refers to the object instance once it is created.

Scope

You can select a scope from the drop-down menu, or enter a custom scope. The default is `Request`.

When you select a scope, consider how you plan to use the variable. For example, you are using the object instance to calculate interest based on principle and interest rate, and you want to get the balance later, in another application file. In this case, you should select the `User` scope instead of the `Request` scope, because the `User` scope makes the variable available beyond the execution of the current application file.

Name

Enter the variable name in this dialog box.

The name that you enter in this box becomes available in the drop-down menu in the Object Instance Variable section of Call Method action windows in the same Witango application file or Witango class file.

Instance



Create New Object

Create New Object is the default option. Witango Server creates a new instance of the object when the application file is executed. The new instance exists until the expiry time is reached—which is determined by the variable scope that you specify in the **Object Instance Variable** section—or until it is purged with a `<@PURGE>` meta tag.

This is the only selection, unless you are using a COM object. Even if you are using a COM object, in most cases, this is the appropriate selection.

Get Existing Object

In some special cases, you may want Witango to connect to an object instance that is already existing, instantiated by the operating system of

your machine, independent of your Witango application file. If you are familiar with the way this process works and you want to use the object instance in this manner, you can make this selection.

Currently, Witango supports this feature only in COM objects. If you are using a JavaBean or a Witango class file, this selection is disabled.

If the object you specify is not registered properly as a global COM resource when Witango Server executes the Create Object Instance action, WitangoServer generates an error.

Initialization String

This feature allows you to bring up COM objects in some initialized state, as implemented by the referenced moniker. (A *referenced moniker* is a term used with COM objects; it is essentially a string of characters in a special format for creating initialized COM objects. See your COM object documentation for details.) Specifying an initialized state can combine several actions into one, and thus save development time.

An example of using a referenced moniker is to specify an action based on a spreadsheet file. When Witango Server executes this initialized object instance, it opens the spreadsheet program and the spreadsheet file, lets the spreadsheet program perform its operations and returns the result to Witango.

Currently Witango supports the use of initialization string only in COM objects. If the instance you are creating is from a JavaBean or a Witango class file, the field is disabled.

You can do one of the following with the **Initialization String** text box:

- Leave the text box empty.

Witango creates an uninitialized instance of the COM object (the object indicated in the **Object Name** field).

- Enter the referenced moniker in the text box. (Refer to the documentation from your COM object vendor for the use of referenced monikers.)

Witango reads the initialization string to create an initialized instance; it ignores whatever COM object that you may have selected when you started the Create Object Instance action (that is, the object indicated in the **Object Name** field). The information you enter in the **Object Instance Variable** section applies to this initialized instance.

Username and Password

By default, a COM object uses the username and password of your Witango Service login.



Tip You can view your Witango Server login information as follows. From the Control Panel, choose **Services**, then select the Witango Server you want to view, then click **Startup**. In the Service dialog box that appears, you see the login information under **Log on as**. The default is `System Account`.

However, you have the option to specify a different username and password for this COM object.

The **Username** and **Password** text boxes allow you to enter optional security information for this COM object. Both text boxes accept meta tags and are encrypted in the Witango application file or Witango class file. Username and password fields are limited to 128 characters.

Currently, Witango supports the use of username and password only in COM objects.

Expiry URL

When the object instance expires, Witango destroys the instance.

You can direct Witango Server to perform application-specific cleanup operations prior to the destruction of the instance. Enter either a valid URL or one or more meta tags that evaluate to a valid URL. The URL must be a complete HTTP URL that Witango Server can access. For example:

```
http://127.0.0.1mycleanup.taf?object=myobject
```

Adding a Call Method Action

For more information, see “Adding an Object to the Objects Workspace” on page 365, Using Available Object Instances on page 350, and Completing the Call Method Action on page 385.

Before you add a Call Method action to your Witango application file or Witango class file, make sure the following conditions are met:

- the object associated with that method is available in the Objects Workspace
- the object instance to which this Call Method action refers has already been created earlier in the execution of your Witango application file
- the information for the object instance is complete; in particular, the object instance has been assigned to a Witango variable.

A Call Method action is a Witango action. The action icon is available on the Actions bar.

The procedure for adding a Call Method action is the same whether you are working with aCOM object, JavaBean, or Witango class file.

To add a Call Method action

- 1 Open the Witango application file or Witango class file in which you want to call a method.
- 2 Drag the Call Method action icon from the Actions bar to the location you want in your application file or Witango class file. (It must be in a logical sequence after the Create Object Instance action that this Call Method action refers to.)



The Call Method action window appears. The field next to the Call Method action icon displays a message: “Drag a method from the Workspace.”

- 3 Click the **Objects** tab in the Workspace to view the Objects Workspace.
- 4 Locate the object and method you want to use, as follows:
 - Expand one of the object types—COM/DCOM Objects, JavaBeans, or Witango class files—that you want to view, by clicking the plus sign (+) to the left of this object type.



Note The object you select for your Call Method action must be the same as the object in the Create Object Instance action that you want this Call Method action to refer to. Otherwise, Witango Server returns an error during execution of the application file.

A list of all objects of this type appears. (A group of related

objects may be contained in a folder.)

- Expand the object that you want to use by clicking the plus sign (+) to the left of this object.

A list of available methods appears.



Note Some methods are listed under the `Attributes` folder, if this folder exists. Expand the `Attributes` folder to see those methods. For more information, see “Attributes Folder” on page 373.



For more information, see “Completing the Call Method Action” on page 385.

- 5 Select the method you want to use and drag it to the Call Method action window.

The Call Method action window displays the names of the object and the method, as well as the list of parameters for this method. An example of the Call Method action window is shown on page 385.

- 6 Complete the information in the Call Method action window.



Note You can skip this step for the time being. At a later time, open this action item and complete the information.

- 7 Close the Call Method action window.

A Call Method action appears in your application file or Witango class file. The default name for the action is the name of the `method`.

For more information, see “Working With Actions” on page 229.

- 8 If you want, you can change the name of the action to whatever is appropriate in your case.

Shortcut to Adding a Call Method Action

An alternative approach to adding a Call Method action from the Actions bar is to do it directly from the Objects Workspace. This procedure is a slight modification of the one described on page 383.

Skip the step involving the Call Method action icon. Just drag the method you want to use from the Objects Workspace to the location you want in the application file or Witango class file.

Completing the Call Method Action

The Call Method action window contains important information about a Call Method action. It allows you to view and edit this information.

You can open the Call Method action window by doing one of the following:

For more information, see “Adding a Call Method Action” on page 383.

For more information, see “Shortcut to Adding a Call Method Action” on page 384.

- dragging the Call Method action icon from the Actions bar to an application file (when you create a new Call Method action)
- dragging a method from the Objects Workspace to an application file (when you create a new Call Method action)
- double-clicking the name of a Call Method action in your application file (when you edit an existing Call Method action).

The title of the Call Method action window is the name of your Witango application file, followed by the name of the Call Method action, separated by a colon.

For more information, see “Working With Actions” on page 229.

You can change the name of the Call Method action in the application file window.

The following is an example of the Call Method action window:

Method Name: authentication.Authentication

Object Instance Variable

Scope: Request

Name:

Put result into variable (Text)

Scope: Request

Name:

Parameters:

Name	Type	Format	Value	Incl. Empty
login_id	Text	Value		true
passwd	Text	Value		true
firstname	Text	Variable	Request	true
lastname	Text	Variable	Request	true
ulevel	Text	Variable	Request	true
validlogin	Text	Variable	Request	true

There are several sections to the Call Method action window: object/method name, object instance variable, result variable, and parameter list.

Object/Method Name

This is a non-editable field at the top of the window, next to the Call Method action icon. The object/method name consists of the name of the object, followed by the name of the method, separated by a dot. If no method has been assigned to this action, the field displays the message *Drag a method from the Workspace*.

Object Instance Variable

This section allows you to specify the object instance you want to use in your Call Method action. You must have created the object instance in a Create Object Instance action and given it a variable name.

Before you complete the information in this section, you need to know the following:

- the variable name given to the object instance you want to use
- the scope of this variable.

You can view this information by double-clicking the appropriate Create Object Instance action.

From the **Scope** and **Name** drop-down menus, select the appropriate items or enter them.



Caution The information you enter in the **Object Instance Variable** section of the Call Method action window must be identical to the corresponding section of the Create Object Instance action window. (This includes the scope and the name of the variable.) Otherwise, Witango Server returns an error when it executes the application file.

Example

The Create Object Instance action creates an object instance, `MyObjectInstance`, from the object, `MyObject`; the variable to which this instance is assigned is `MyVariable`, in `Local` scope. To specify a Call Method action using this object instance, drag `MyObject` in from the Objects Workspace, and specify `MyVariable`, in `Local` scope, in the **Object Instance Variable** section of the Call Method action window.

Result Variable

The Call Method action may generate a result. The data type of the return result is indicated in parentheses after the **Put Result into Variable** title. An example is `(bool)`.

For more information, see “Viewing Object Information in the Objects Workspace” on page 372.

This information is also shown in the Objects Workspace.



Note If the Call Method action does not generate a result, it is indicated as `(none)` after the **Put Result into Variable** title. The Scope and Name drop-down menus are disabled.

If you plan to use the result later in your application file, put the return result into a variable. From the **Scope** and **Name** drop-down menus, select the appropriate items or enter them.

Example

You are using the Interest Calculator object to calculate the interest on a customer account. When you get the result of the calculation, you can store it in a variable called `Interest`. When you need to use this result later to present a statement on a Web page, refer to the `Interest` variable.

Parameter List

A parameter allows a Witango application file to exchange data with an object. The exchange can be an input (passing data from the application file to the object), output (passing data from the object to the application file), or input/output (passing data from the application file to the object and putting the new value from the object in the original variable).

Name

This column shows the parameter names, and whether a parameter is an input, output, or input/output parameter. The following icons are used:

-  input parameter
-  output parameter
-  input/output parameter

For more information, see “Object Properties” on page 374.

Parameter names are also listed in the Objects Workspace. If you want to change a parameter name, do it in the Objects Workspace. After the change, Witango updates the Call Method action window. (You have to close and reopen the Call Method action window to see the new name.)

Type

This column shows the data type of each parameter. There are two categories of data types:

- Fixed

In most cases, the data type for a parameter is fixed. It does not vary from one Call Method action to another. This is the data type specified by the object vendor. For details, see the documentation supplied by your object vendor.

- **Variant**

When using COM objects, you may encounter variant parameters. A *variant parameter* is a parameter which does not have a fixed data type assigned to it; the data type may vary from one Call Method action to another. A variant parameter can be an input parameter or an input/output parameter.

If you click the **Type** column of a variant parameter, you see a drop-down menu of data types used in COM objects. Refer to the documentation from your COM object vendor and select the appropriate item from the menu. The default is `Text`.



Tip Instead of selecting from the drop-down menu, you can also type in a meta tag which evaluates to a data type when Witango Server executes the Witango application file.

The data type you select for the variant parameter applies to this particular Call Method action only.

For more information, see “Viewing Object Information in the Objects Workspace” on page 372.

Parameter data types are also listed in the Objects Workspace, in brackets, after the respective parameter names. The exception is the variant data type, which depends on the particular Call Method action.

Format

This column shows the format of a parameter. A parameter has one of two formats: `Variable` and `Value`.

Parameter Type	Possible Formats
Input	Variable or Value
Output or Input/Output	Variable

When you click the **Format** column of a parameter, the format for this parameter becomes enclosed in a drop-down menu. If the parameter is an input parameter, you can select `Variable` or `Value`; otherwise, it can only be `Variable`.

Value

This column shows the value or variable of a parameter. You can change the values and variables in this column.

- **Format is Value**

If the format of a parameter is **Value**, you input a literal value or a meta tag that evaluates to a literal value.

Examples of literal value are 237 (integer), John Smith (text), and 3.14159265 (floating point).

An example of a meta tag that evaluates to a literal value is

```
<@SUBSTRING STR="alpha" START="3" NUMCHARS="2">
```

(This meta tag evaluates to the literal value, ph.)

- **Format is Variable**

If the format of a parameter is **Variable**, this column shows the scope and the variable name of the variable.

When you click the **Value** column of a parameter, two cells appear. The left-hand cell, which contains the scope name, becomes a drop-down menu. You can make changes by selecting another scope or entering a custom scope. The right-hand cell, which contains the variable name, becomes a text box. Enter a variable name or change the existing variable name.

When the Call Method action is executed, the value for this parameter is taken from the variable specified (Input/Output or Input type) and any output value is placed in the variable (Input/Output or Output type).

For more information, see
“Working with Meta Tags”
on page 143.

For more information, see
“Working With Variables”
on page 155.

Incl. Empty

This column is enabled for optional parameters. With an *optional parameter*, you can specify whether or not to include this parameter in a Call Method action. Only COM objects can have optional parameters.

It specifies whether a value is passed or not if the Value field evaluates to empty at execution time.

- If the parameter is an optional parameter, you can select either **True** or **False** as the value.

- **True**

You ask Witango to always use this parameter. Complete the information in the **Type**, **Format**, and **Value** columns.

- **False**

You ask Witango to use this parameter only when the value specified is non-empty.

- If the parameter is **not** an optional parameter (that is, you are required to use this parameter) the value in this column is indicated as `True`. This value is hard coded; you cannot change it. Complete the information in the **Type**, **Format**, and **Value** columns.

Using the Objects Loop Action

In some cases, a method call returns several items collected into an object, called a *collection object*. The purpose of a collection object is to allow you to work with all the items or selected items in the collection, one by one, in a loop action.

For more information, see “Repeating a Set of Actions (Loop Actions)” on page 288.

The Objects Loop action is a loop action which works similarly to the For Loop action. You can have nested Objects Loop actions (an Objects Loop action within an Objects Loop action).

The Objects Loop action can work with appropriate objects retrieved from COM objects and JavaBeans.

Example of Using an Objects Loop

Suppose there is a group of related objects that allow you to look up and present all the product names in a product category. This involves the following objects:

- `ProductCollection`

This is a collection object containing product objects.

- `ProductCategory`

This is an object which allows you to access the collection. This object contains a method, `getProduct`, which puts the result into a variable, `x`, assigned to `ProductCollection`.

- `Product`

This is an object which contains all the attributes of an individual product. This object contains a method, `getName`.

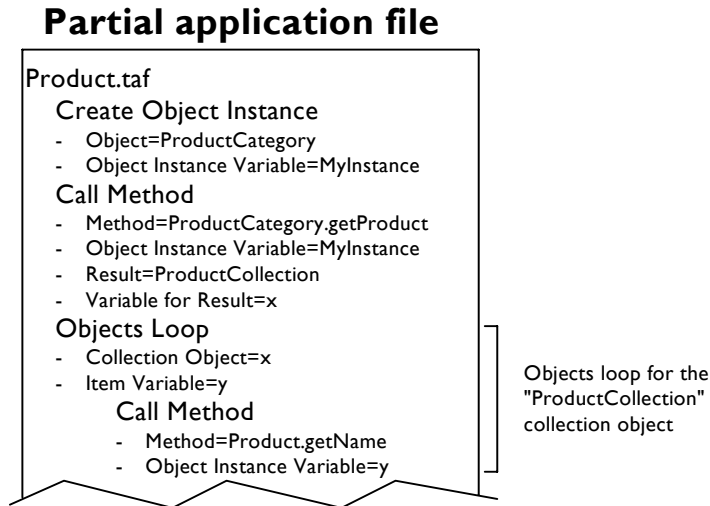
You first create an object instance of the `ProductCategory` object, using a **Create Object Instance** action. Then use a **Call Method** action to call `getProduct` and to put the result of this method—the `ProductCollection` collection object—into variable `x`.

Now you start an **Objects Loop** action to work with the items in the collection object, which is the result of the aforementioned **Call Method** action. This **Objects Loop** action uses the variable `x` to refer to the `ProductCollection` collection object and the variable `y` to refer to each individual item in the collection.

Place a **Call Method** action inside the **Objects Loop** action, to call `getName`, so that it looks up and presents the items in `ProductCollection`. Use the variable `y` as the object instance variable to refer to the current item in the collection.

See an example of the Objects Loop action dialog box on page 393.

The following diagram shows how you might incorporate a sequence of actions in your Witango application file to use an objects loop:



Using an Objects Loop

For more information, see "Using Available Object Instances" on page 350

For more information, see "Adding a Create Object Instance Action" on page 377 and .

Using an objects loop requires a sequence of actions, generally including at least a Create Object Instance action, a Call Method action, and an Objects Loop action. The precise requirements depend on the collection object you use and what you want to do with it. Refer to the documentation from your object vendor for details.

The heart of the sequence is the Objects Loop action, which works on a collection object. Typically you get a collection object as the result or an output from a Call Method action, which in turn needs to use an available object instance.

The use of Create Object Instance action and Call Method action is covered in other sections. This section focuses on using the Objects Loop action.

To specify an objects loop for a collection object

- I Open the Witango application file or Witango class file that you want to use an objects loop.



- 2 Drag the Objects Loop action icon from the Actions bar to the location you want in your application file or Witango class file.



Note The Objects Loop action must be placed in a logical sequence after a collection object has become available. Typically, you get a collection object as the result or an output from a Call Method action. For an illustration of how this works, see Example of Using an Objects Loop on page 391.

The Objects Loop action dialog box appears:

- 3 Complete the information in the Objects Loop action dialog box:

- Collection Object

The name and scope must be identical to those of the variable assigned to the result or an output (a collection object) from the Call Method action (or the `<@CALLMETHOD>` meta tag) to which this Objects Loop action refer.

- Item Variable

This is the variable to which the current item in the collection is assigned on each iteration of the loop. When the counter advances, the same variable name is used for the next item in the collection.

- Limits

Enter the numbers—or meta tags that evaluate to numbers—which represent the items that you want the loop action to start and stop with.

If the value is something other than a number, Witango ignores it and uses the default value (1 for start and the last item in the collection for stop). If you leave the boxes empty, the loop action

covers all the items in the collection.

Witango Class Files

Creating Your Own Witango Modular Code

Witango class files are reusable software components that you can incorporate in Witango application files. You can create and edit Witango class files using Witango Studio.

Witango supports the use of several types of reusable software components—called classes or “objects”—in Witango application files. Witango allows you to use the Witango class files that you create yourself, along with other objects that are available from third-party vendors.

For more information, see “Understanding Objects in Witango” on page 345.

This chapter assumes you are familiar with the basic concepts of using objects in Witango. The topics covered in this chapter include:

- an introduction to Witango class files
- the benefits of using Witango class files
- creating and editing Witango class files
- using editing windows for Witango class files
- setting search paths for Witango class files.

For more information, see “Using Objects” on page 361.

Once you have created your Witango class files, you can incorporate them in your Witango application files.

What are Witango Class Files?

For more information, see “Understanding Objects in Witango” on page 345.

Witango class files are reusable software components that you can create and edit using Witango Studio.

Witango supports several types of reusable software components available on the market, such as COM objects and JavaBeans. These software components are, strictly speaking, called classes. A class is a category of objects; it defines all the common properties of the different objects that belong to it. In industry, however, the term “object” is often used loosely. What is called an object, such as a COM object, may in fact be a class or a category of objects.

A Witango class file is a class or “object” you can use in Witango. You can use a Witango class file in Witango in the same way that you use a COM object or a JavaBean. The main difference is that, with Witango class files, you can create and edit your own reusable software components using Witango Studio.

Because Witango class files are treated like COM objects and JavaBeans in Witango, they are generally called “objects” in the *User’s Guide*.

Example

To see how you might use a Witango class file, refer to Example 1: Investment Scenarios on page 351. You are looking for an object to organize your report in the `invest.taf` application file. Suppose you cannot find a suitable object from a third-party vendor—you may want to create your own Witango class file.

After you have created the Report Organizer as a Witango class file, you can incorporate it in `invest.taf`.

Benefits of Using Witango Class Files

When you use Witango class files in your Witango application files, you obtain a number of benefits:

- Because Witango class files work like other objects—such as COM objects and JavaBeans—in Witango application files, the benefits you get from using these objects also apply when you use Witango class files.
- You do not have to rely exclusively on third-party object vendors. If the objects available on the market do not suit your purpose, you can create your own.
- Any Witango class file you develop is potentially reusable in future Witango application files. You can create a library of Witango class files that you and others can use.
- In addition to the Witango class files that you develop, you can draw on the Witango class files that others have developed.
- You do not have to know any of the programming languages—such as C++, Visual Basic, or Java—commonly used for writing objects. Developing a Witango class file is similar to developing a Witango application file, a process you are already familiar with.
- You can mix and match different object types supported by Witango: objects available from many vendors and objects you develop yourself (that is, Witango class files) can be used in the same Witango application file.
- You may modify the source code inside Witango class files to improve their design. As long as you do not change the interface—which is normal practice—you can benefit from the improved design, without having to alter the code in your Witango application files.
- You can use Witango class files as “wrappers” for COM objects and JavaBeans to simplify calling them from your application files. One example is to replace several COM object or JavaBean method calls with a single Witango class file method call. Another example of using a Witango class file as a “wrapper” is to simplify the parameter list of a COM object or JavaBean method by exposing in a Witango class file only those parameters that you normally change and hard-coding the others.

When to Develop and Use Witango Class Files

If the objects available from third-party vendors suit your purpose, it is probably easiest to use them.

If you plan to reuse sections of your Witango application file in future application files, it may be more efficient to develop Witango class files and then use them in your current and future application files.

In general, Witango code that you often call by using a Branch action with the return option set is a good candidate for encapsulation into a Witango class file method.

You can incorporate Witango class files in any Witango application file, in the same way you incorporate other objects that Witango supports.

Using Witango Class Files

There are two main steps in using Witango class files:

1 Creating and editing Witango class files

- If the Witango class file you plan to use is existing—that is, it has been created by you or by others—go directly to step 2.
- If the Witango class file you plan to use does not exist, you have to create it first. Then go to step 2.
- If an existing Witango class file does not meet your precise needs, you can modify it. Then go to step 2.

For more information, see “To create a Witango class file” on page 408.

For more information, see “Editing a Witango Class File” on page 409.

2 Incorporating Witango class files in Witango application files

Once created, Witango class files can be incorporated in Witango application files, just like other objects. Witango class files are treated as “black boxes”; that is, as a user of Witango class files, you do not need to know the source code inside the Witango class files. You simply interact with a Witango class file through its interface, that is, its methods.

For more information, see “Overview of Using Objects in Witango” on page 364.

Using Witango class files in Witango is similar to using other objects in Witango.

Developing Witango Class Files

Two windows in Witango Studio are specially designed for developing Witango class files:

- The *Witango class file window* is the main environment in which you develop Witango class files. It allows you to set up the methods for each Witango class file and specify actions for each method.
- The *Method Definition window* allows you to set up the return value and parameters for each method.

This section focuses on the features available in the Witango class file window and Method Definition window.

The other sections in this chapter give you the procedures for creating and editing Witango class files.

For more information, see “Creating a Witango Class File” on page 408 and “Editing a Witango Class File” on page 409.

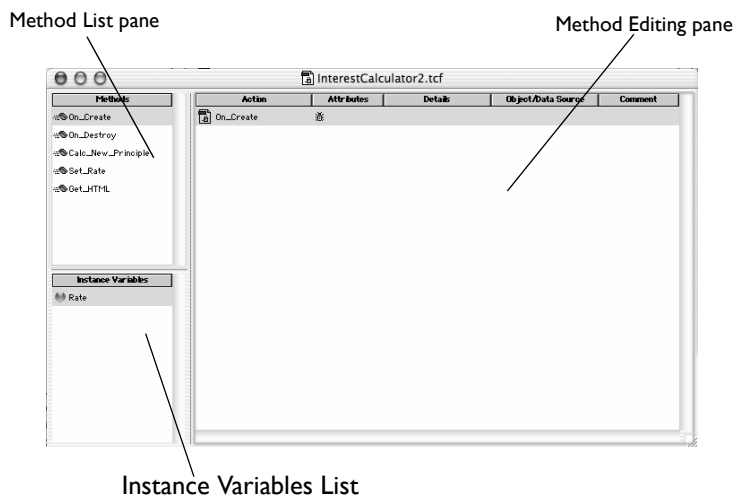
To open the Witango class file window

Do one of the following:

- New Witango class file: From the **File** menu, choose **New**, then choose **Witango Class File**.
- Existing Witango class file: From the **File** menu, choose **Open**, then locate and open the Witango class file you want.

The Witango class file window consists of three panes: Method List pane, Instance Variables List pane, and Method Editing pane.

The following is an example of the Witango class file window:



Method List Pane

The Method List shows the names of all the methods in the Witango class file. Two types of methods are shown in the list: user-created methods and default methods.

User-Created Methods

These methods constitute the Witango class file's interface when you incorporate the Witango class file into your Witango application file. When you use a Call Method action, you are specifying one of the user-created methods on this list.

For more information, see "Editing a Witango Class File" on page 409.

You can create as many methods for a Witango class file as you want. You can delete the user-created methods that you no longer need. You can also change the order in which the methods appear in the list.

Default Methods

The two default methods in the list are `On_Create` and `On_Destroy`.

- `On_Create`

When Witango Server creates an instance of the Witango class file, it automatically calls the `On_Create` method before any further processing.

The `On_Create` method is initially empty. When you select this method and specify actions in the Method Editing pane, you direct Witango Server to execute these actions prior to the creation of the instance. An example of using the `On_Create` method is to initialize certain instance variables.

The use of the `On_Create` method is optional. If you do not want to use it, just leave the Method Editing pane empty.

- `On_Destroy`

When Witango Server destroys an instance, it automatically calls the `On_Destroy` method just before it proceeds with the destruction.

The `On_Destroy` method is initially empty. When you select this method and specify actions in the Method Editing pane, you direct Witango Server to execute these actions prior to the destruction of the instance. An example of using the `On_Destroy` method is to perform certain clean up activities.

The use of the `On_Destroy` method is optional. If you do not want to use it, just leave the Method Editing pane empty.

The default methods differ from user-created methods as follows:

- you cannot delete them

- they have no parameters or return values
- you cannot edit the parameter list
- they are invisible when you view Witango class file information in the Objects Workspace
- you cannot call them using Call Method actions or the `<@CALLMETHOD>` meta tag.

Method Editing Pane

When you select a method on the Method List, the Method Editing pane shows the actions included in this method. (The root item in the Method Editing pane is the method currently selected on the Method List.)

Because a method of a Witango class file is simply a reusable software component that you insert into a Witango application file, the content of a method looks like and behaves like part of the content of a Witango application file.

For more information, see “Witango Studio Basics” on page 5.

Indeed, the Method Editing pane resembles the Witango application file window. In most cases, you use the Method Editing pane similarly to the way you use the Witango application file window: you drag actions from the Action bar into the pane and edit them.

You can copy an action or a series of actions within the same Witango class file and between Witango class files. You can also copy actions to and from Witango application files. Other than a few exceptions, copying actions involving Witango class files is similar to copying actions in Witango application files.

Differences from Witango application files

Because Witango class files and Witango application files do not perform identical functions, there are some differences between Witango class files and Witango application files that you should be aware of:

- **Executing a Witango class file.** You cannot execute or branch to a Witango class file directly. All access to Witango class file methods is via Call Method actions or `<@CALLMETHOD>` meta tags and executed in a Witango application file.
- **Branch action.** The Branch action, when used in a method, is limited to branching within the current method. If you copy from a Witango application file to a Witango class file, Witango Studio does not allow you to copy a Branch action that branches to a location outside the current method.
- **Return value/Results HTML.** Each method call has its own private Results HTML, which is empty at the start of its execution.

For more information, see “Return Value” on page 406.

Like Witango application files, the Results HTML associated with each action is accumulated as execution progresses. Unlike Witango application files, however, the Results HTML of a method is not automatically appended to the Results HTML of the Witango application file. You may return the Results HTML as the method's return value or in an output parameter (using `<@RESULTS>`). To append the method's Results HTML to the calling Witango application files, you need to include the variable you stored it in the Witango application files's Results HTML, for example, `<@VAR NAME="myresults" ENCODING="none">`.

- **Push attribute.** The Push attribute is disabled for actions inside a method. If you copy from a Witango application file to Witango class file, Witango Studio turns off the Push attribute automatically.
- **Assign action.** The Assign action in a Witango class file allows you to set variables in the instance and method scopes, in addition to those scopes available to Witango application files.

When copying Assign actions from a Witango class file to a Witango application file, all the variables are copied; however, the scopes of instance and method variables are changed from `instance` and `method` to `default`.

- **Meta tags to set and get parameters.** Within a Witango class file method, there are two meta tags available (`<@GETPARAM>` and `<@SETPARAM>`) which set and return the value of a named parameter. The tags get and set the named method variables, with the added benefit of error checking; that is, if the variable specified is not a parameter, an error occurs.
- **Recursion.** You can make recursive calls to a method. The `returnDepth` configuration variable tracks recursive calls to methods as well as Branches, and the `maxActions` configuration variable includes actions in methods in its count of the total executed.
- **Error handling.** When an error occurs in a Witango class file method action:
 - If the action has Error HTML, Witango processes it and stops method processing.
 - If the method's return value is the Results HTML, Witango returns the Error HTML.
 - If the action has no Error HTML, Witango passes the error up the calling chain to the original Witango application file.

Nested Method Calls

When you specify actions for a method, you can include Call Method actions involving other methods. The calling of one method from another is a nested method call.

Assuming method A calls method B, the following issues affect the nested method call:

- The instance and method variables in method A become unavailable until Witango Server has completed the execution of method B and returns to method A.

The exception is that, if method A and method B are part of the same instance, the instance variables are shared.

- Assignments to configuration variables in the instance and method scopes of method A have no effect on method B.

If an assignment is made to `method$dateFormat` in method A, the `method$dateFormat` in method B remains empty until set explicitly. Further, the scope of `method$dateFormat` in method B is determined by the default scoping rules, not the scope set in method A.

Self-Referencing

When you specify actions for a method, you can include a Call Method action that refers to the current Witango class file. In this case, the object instance variable for this self-referencing Call Method action is set automatically: the scope is `Method` and the name is `this`.

Instance Variables List Pane

The Instance Variables List shows the names of all the unique instance variables for the Witango class file. All instance variables in the list are available to all the methods in the Witango class file.

Instance variables (variables in the instance scope) are available only in Witango class files; they are not available in Witango application files.

The Instance Variables List in a new Witango class file is empty. It is populated automatically if you assign instance variables in the Witango class file by doing one of the following:

- using an Assign action
- using the **Put result into variable** section of a Call Method action window
- using the Out or In/Out parameters of the Parameter List of a Call Method action window.

An item that appears in this list may be dragged into an Assign editing window or Results HTML window, automatically assigning the instance variable in that Assign action or creating a snippet with the `<@ASSIGN>` meta tag in that Results HTML.

When you delete all references to an instance variable in all the Assign actions of a Witango class file, this instance variable automatically disappears from the Instance Variables List.

To use an instance variable in an assignment

- 1 Open an Assign editing window or a Results HTML window in the Method Editing pane of the Witango class file window.
- 2 Drag an item from the Instance Variables List into the Assign editing window or Results HTML window.



Tip When you click in the Witango class files window, the Witango class files window is brought to the front, which may hide the Assign editing window or Results HTML window. Rearrange the windows so that both are visible at the same time.

A new assignment is added to that window.

Method Definition Window

For more information, see “To open the Witango class file window” on page 400.

The Method Definition window contains important information about a Witango class file method. It allows you to view and edit this information.

To open the Method Definition window

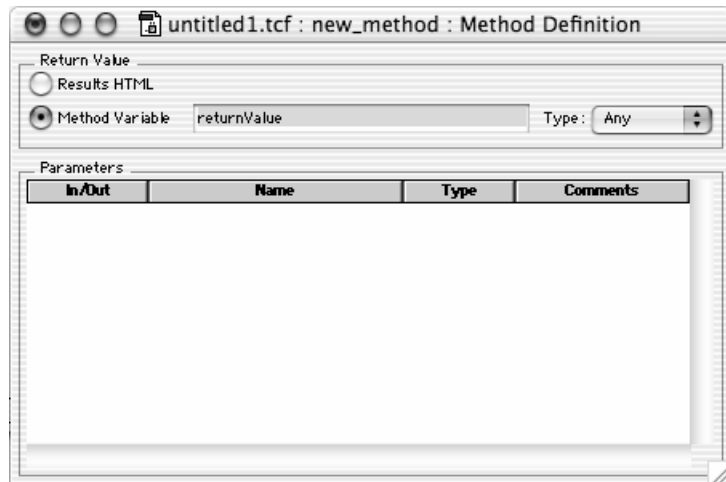
- 1 Open the Witango class file window.
- 2 Do one of the following:
 - In the Method List pane, double-click a user-created method.
 - In the Method List pane, select a user-created method, right-click it, and choose **Open** from the context-sensitive menu that appears.
 - From the Method List pane, select a user-created method. Then, double-click the root item in the Method Editing pane.
 - From the Method List pane, select a user-created method. Then right-click the root item in the Method Editing pane, and choose **Open** from the context-sensitive menu that appears.



Note Default methods have no return values or parameters.

The Method Definition window consists of the return value section and the parameter list for a method.

The following is an example of the Method Definition window:



Return Value

You return either the Results HTML or the contents of a method variable, by choosing one of the options.

If you select **Method Variable**:

- The default variable name is `returnValue`.
- The default data type for this variable is **Any**. You can select the data type you want from the **Type** drop-down menu. When returning results, Witango generates an error if the value is not of the type specified here.

Parameter List

The Parameter List is where you define the interface to a method of a Witango class file.

A parameter allows a Witango application file to exchange data with a Witango class file. The exchange can be an input (passing data from the application file to the Witango class file), output (passing data from the object to the application file), or input/output (passing data from the application file to the object and putting the new value from the object in the original variable in the application file).

For more information, see
“Parameter List” on
page 387.

The Parameter List for a method of a Witango class file resembles the Parameter List in the Call Method action window. When you perform a Call Method action on a Witango class file, the information from the Parameter List in the Method Definition window is transferred to the Parameter List in the Call Method action window.

The Parameter List for a Witango class file consists of four columns: In/Out, Name, Type, and Comments.

- In/Out

This column shows whether a parameter is an input (**In**), output (**Out**), or input/output (**In/Out**) parameter.

The following icons are used:

-  input parameter
-  output parameter
-  input/output parameter.

When you click the **In/Out** column of a parameter, you see a drop-down menu consisting of the three possible values. Select the one you want to use.

- Name

This column shows the parameter names.

Assign a unique name to each parameter. It is best to use friendly and informative names. The rules for assigning parameter names are the same as those for naming variables.

- Type

This column shows the data type for each parameter. You can accept the default, which is `Text`, or select one of the following from the drop-down menu: `Any`, `Array`, `Object`, and `DOM`.

If `Any` is selected, any type of value is allowed; otherwise, Witango Server checks the input value for the parameter when it starts to execute a method. If the value does not match the specified type, Witango Server generates an error.

- Comments

This column allows you to enter your optional comments for each parameter.

Creating a Witango Class File

To create a Witango class file



- 1 From the **File** menu, choose **New**, then choose **Witango Application File**, or, on the main toolbar, click the **New Witango Class File** icon.

The Witango class file window opens. This window consists of three panes: Method List pane, Instance Variables List pane, and Method Editing pane.

An example of the Witango class file window is shown on page 400.

For more information, see “Default Methods” on page 401.

- 2 If you want to use the `On_Create` method, select `On_Create` from the Method List. In the Method Editing pane, specify the actions you want to include in this method.

- 3 If you want to use the `On_Destroy` method, select `On_Destroy` from the Method List. In the Method Editing pane, specify the actions you want to include in this method.

- 4 Add a method to the Method List.

The new method is automatically selected and becomes the root item in the Method Editing pane.

- 5 In the Method Editing pane, specify the actions you want to include in this method.

- 6 Double-click the root item (that is, the selected method) in the Method Editing pane to open the Method Definition window.

- 7 Complete the information in the Method Definition window.

- 8 Repeat steps 4 to 7 until you have added all the methods you want.

- 9 Save the Witango class file.

For more information, see “Adding a New Method” on page 410.

For more information, see “Method Editing Pane” on page 402.

For more information, see “Method Definition Window” on page 405.

For more information, see “Saving a Witango Application File or Witango Class File as Run-Only” on page 59.



Tip If you want, you can save the Witango class file as run-only. This feature works in a way similar to that of the Witango application file.

Editing a Witango Class File

There is a major difference between editing a Witango class file *before* and *after* it is incorporated into a Witango application file, through a Call Method action. Before you use a Witango class file in a Witango application file, you can edit it any way you want to suit your purpose. After you include a Witango class file in a Witango application file, you are restricted in what you can do with the interface of the Witango class file.

You can think of the interface of a Witango class file as a contract between the Witango class file and the Witango application file that uses the Witango class file. Once the contract is in effect, you can no longer change the terms of the contract.

When you incorporate a Witango class file into a Witango application file, the latter treats the former as a “black box” and interacts with it only through its interface. Both the Witango class file and the application file honor the interface like a contract. Thus, you have to be cautious when altering the interface of a Witango class file once a Call Method action is used on that Witango class file.



Caution Do not alter the parameters or variables of a method, or the names of the Witango class file and the method, after using a Call Method action on that method. Witango Server cannot execute the Call Method action when these elements are altered; it returns an error.

You can generally add new methods, parameters and variables without affecting existing Call Method actions. In many cases, you can modify the actions within a method, as long as you do not change the existing parameters and variables.

The following windows and dialog box are used for editing Witango class files:

- **Witango class file window.** This is where you do most of the editing, with the exception of return values and parameters.
- **Method Definition window.** This is where you edit the return value and parameters for each method.
- **Method Properties dialog box.** This is where you edit comments of a method.

All the procedures in this section require you to start with an open Witango class file window. To open a Witango class file window, choose

Open from the File menu, then locate and open the Witango class file you want to edit.

Adding a New Method

To add a new method

1 Do one of the following:

- From the **Edit** menu, select **New Method**.
- Right-click anywhere in the Method List pane and choose **New Method** from the context-sensitive menu that appears.

The new method appears at the bottom of the Method List. The default name is `new_method`.



Note Witango resolves name conflicts automatically. If `new_method` already exists, the name becomes `new_methodX`, where X is an integer.

2 If you want, change the name of the method to one that is more meaningful.



Tip You can also drag the method to anywhere you want in the list. The only restriction is that `On_Create` and `On_Destroy` are always at the top of the list.

Renaming a Method

The default name of a method is `new_method` or `new_methodX`, where X is an integer. It is recommended that you change the method name to one that is more meaningful.

To rename a method

1 From the Method List, select the method you want to rename.



Note Although the name of a method also appears as the root item in the Method Editing pane, you cannot change the name there.

2 Click the name of the method, or from the **Edit** menu, choose **Rename**.

3 Type the new name.

The rules for assigning method names are the same as those for naming variables.

Deleting a Method

You can delete a method from a Witango class file if you no longer need it, provided that no Witango application file is including this method in its Call Method actions.



Caution If a Witango application file calls a method that no longer exists, Witango Server returns an error when attempting to execute that Call Method action.

To delete a method

- 1 From the Method List, select the method you want to delete.
- 2 Do one of the following:
 - From the **Edit** menu, choose Delete.
 - On the main toolbar, click the Delete icon.
 - Press `Delete`.



Note You cannot delete the two default methods, `On_Create` and `On_Destroy`. To prevent Witango from using either of these default methods, delete all its actions.

- 3 When a dialog box appears, asking you to confirm the deletion, choose **OK**.

Copying a Method

You may want to create a method that performs a task similar to one performed by an existing method in the current Witango class file or another Witango class file. Instead of having to re-create the method—along with all its actions and parameters—you can copy an existing method to a new location, and then modify the newly created method.

To copy a method

- 1 From the Method List, select the method you want to copy.
- 2 Do one of the following:
 - To copy to the same Witango class file, press `Ctrl` and drag the method to the location you want on the same Method List.
 - To copy to a different Witango class file, drag the method to the location you want in the Method List of that Witango class file.



Tip Alternatively, you can copy and paste the method using the Edit commands. Edit commands are available from the Witango Studio Edit

For more information, see “Renaming a Method” on page 410.

For more information, see [Modifying a Method](#) page 412 and [Setting Return Values and Parameters](#) page 412 on this page.

Modifying a Method

For more information, see “Method Editing Pane” on page 402.

Setting Return Values and Parameters

For more information, see “Method Definition Window” on page 405.

For more information, see “To edit a parameter” on page 413.

menu and from the context-sensitive menu.

Witango resolves name conflicts automatically. If you want, change the name of the new method to one that is more meaningful to you.

- 3 Where appropriate, edit the actions in the new method (in the Method Editing pane of the Witango class file window) and edit the return value and parameters (in the Method Definition window) for the new method.

To modify the actions in a method

- 1 From the Method List, select the method you want to modify.
- 2 Edit the actions in the Method Editing pane, similar to the way you edit a Witango application file.

For each selected method, you can specify where you want Witango Server to put the return value; you can also insert, delete, or edit its parameters.

In all cases, open the Method Definition window first.

To assign a return value

- 1 Select either **Results HTML** or **Method Variable**, using the appropriate radio button.
- 2 If you select **Method Variable**, enter a name for this variable (the default is `returnValue`), and select the data type for this variable from the **Type** drop-down menu (the default is `Any`).

To add a parameter

- 1 Right-click anywhere on the Parameter List and choose **Insert** from the context-sensitive menu that appears.
- 2 Enter the required information in the new parameter.

To delete a parameter

- 1 Right-click the parameter you want to delete. (You can select more than one parameter for deletion by pressing $\chi\tau\rho\lambda$ or $\Sigma\eta\iota\Phi\tau$, and then right-click.)
- 2 Choose **Delete** from the context-sensitive menu that appears.

To edit a parameter

- 1 Click in the In/Out column of the parameter you want to edit. Select In, Out, or In/out from the drop-down menu that appears.
- 2 Click in the Name column of the same parameter and edit the parameter name.
- 3 Click in the Type column of the same parameter. Select the Witango data type you want to use, from the drop-down menu that appears. If you want the parameter to accept all Witango data types, select Any.
- 4 If you want, edit the comments in the Comments column.
- 5 Repeat steps 1 to 4, for each parameter you want to edit.

Getting and Setting Parameters Within a Witango Class File Method

Inside a method, you must get the values of parameters and set the values of parameters, if your method uses them.

To get parameter values within a Witango class file method

Do one of the following:

- Use `<@GETPARAM NAME=myparam>` to return the value.
`<@GETPARAM>` retrieves the value of a parameter within a Witango class file. This tag is similar to `<@VAR>`, but performs error checking to ensure that only parameters in the current method can be retrieved.
- Use `<@VAR NAME=myparam SCOPE=method>` to return the value.
Parameters are always method variables.

To set parameter values within a Witango class file method

Do one of the following:

- Use `<@SETPARAM NAME=myparam VALUE=myvalue>`.
`<@SETPARAM>` sets the value of a parameter within a Witango class file. This tag is similar to `<@ASSIGN>`, but performs error checking to ensure that only Out and In/Out parameters in the current method can be set.
- Use `<@ASSIGN NAME=myparam SCOPE=method VALUE=myvalue>` to set the value.

For more information, see “Assigning Variables With the Assign Action” on page 156.

- Use the Assign action to set a parameter (method scope).
Parameters are always method variables.

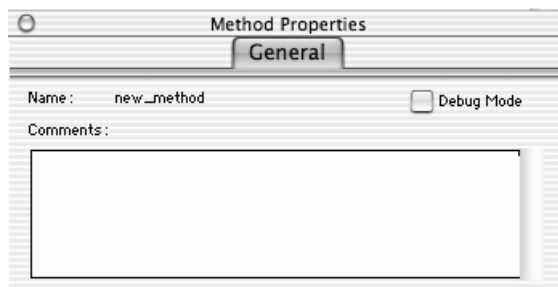
Method Properties

The Method Properties dialog box displays the name of the method you selected. It allows you to enter or edit comments on this method.

To view method properties

- 1 Select a method (either the name on the Method List or the root item in the Method Editing pane).
- 2 Do one of the following:
 - From the **Windows** menu, choose **Properties**.
 - Right-click the name of the method and choose **Properties** from the context-sensitive menu that appears.

The following is an example of the Method Properties dialog box:



Debugging Methods

For more information, see “Debugging Files” on page 61.

This feature allows you to see useful information about the execution of a method. The debug mode applies to an entire Witango class file, not a particular method or a particular action within a method.

In general, debugging a method works in a way similar to debugging a Witango application file. The following characteristics are specific to debugging methods:

- When you enable or disable the debug mode for any method in a Witango class file, it applies to all the methods in the Witango class file.
- The debug mode for a Witango class file operates independently from the debug mode for an application file. If an application file includes a method from a Witango class file and you want to debug both the application file and the method, you have to enable the debug mode in both files.

To set the debug mode in a Witango class file

Do one of the following:

- From the **Attributes** menu, choose **Debug File**.
- Right-click the Method Editing pane and choose **Debug File** from the context-sensitive menu that appears.
- Type CTRL-D.



When the debug mode is enabled, the debug icon appears in the Attributes column of the Method Editing pane. You can repeat this procedure to alternate between enabling and disabling the mode.

Setting Search Paths for Witango Class Files

Setting search paths for Witango Studio and Witango Server are two separate tasks. You need to do both.

Witango Studio The list of search paths for Witango class files is stored on your machine and displayed in the Objects section of the Preferences dialog box. Witango uses this list to locate Witango class files when a Witango application file refers to them.

Adding Paths to the List

For more information, see “Objects” on page 137.

When you add a Witango class file to the Objects Workspace, Witango Studio automatically adds the absolute path to that object to the list of search paths. You can view this list in the Objects section of the Preferences dialog box.

In a team development environment, there may be Witango class file libraries on other machines that you want to use; you can manually add the paths to these libraries to your list.

Setting the Search Order

When searching for a Witango class file, Witango Studio starts from the path at the top of the list and moves down the list. Because Witango identifies Witango class files by names, the search stops as soon as Witango finds the first Witango class file with that name.

If you have more than one version of a Witango class file in your system, you may want to order the paths so that Witango finds the one you want to use for a particular purpose. Re-ordering of the paths is important if you want to use one version for development and another for deployment.

Witango Server When locating Witango class files in Witango Server, a configuration variable called `TCFSearchPath` defines the search path for Witango application files. This variable is valid in all scopes: local, user, application, cookie, domain, and system.

The `TCFSearchPath` configuration variable contains a semi-colon separated list of web root relative paths in which to look for the Witango class files. Witango Server’s treatment of this configuration variable is the same as all others; that is, it uses the narrowest scope in which the variable is first defined to do its search.

Because Witango class files are stored on the Web server, the paths in the `TCFSearchPath` configuration variable are always relative to the Web root, as mentioned. For example, a valid `TCFSearchPath` is:

```
TCFSEARCHPATH=MyApp/TCFs/Logon/;MyApp/TCFs/  
GuestBook/;  
FoneList/Objects/;DougApp/OtherStuff/MyObjects/
```

If the object is not found in the Witango class file search path, Witango Server tries to find the object in the Web server document root folder as a last resort.

SECTION VI

Witango Compiler

How to Configure Witango Server

This section contains details on the operation of the Witango compiler operations which are available in Witango Studio Professional Edition. This functionality has been built to allow Witango applications to be deployed to J2EE environments. This section contains a single chapter:

- Chapter 23, Compiling Witango Application Files on page 421.

Compiling Witango Application Files

Compiling taf files for deployment on J2EE

About Witango Compiler For Java

Witango Compiler For Java compiles Witango application files into java servlets which can be executed on J2EE compliant application servers.

Witango Compiler For Java is written in java language and consists of two parts:

- A compiler which is contained by Witango Studio 5.5 Professional. It can compile Witango application files (.taf files and .tcf files) into java servlet files (.class files).
- A runtime library which resides in the J2EE compliant application server you choose. It can execute compiled java servlet files.

This chapter takes the user through the process of syntax checking the Witango application file (taf and tcf files), and compiling a Witango web application for deployment on J2EE servers.

Before you start

Before you will be able to compile your Witango Application Files you will need to have the Java 2 Runtime Environment (at least JVM 1.4.1) installed on your machine. This can be downloaded from <http://java.sun.com>. For more information, see “To create an JDBC data source” on page 89.

The Compilation Process

The compilation process works around 2 main steps being conducted on a source directory which contains a Witango web application. The steps are:

- 1 **Syntax Checking** - which checks all the `taf` and `tcf` files which are located in the specified source directory (and sub-directories contained therein) to ensure that all Witango meta tags exhibit correct syntax.
- 2 **Compiling for J2EE** - which takes all the `taf` and `tcf` files that are located in the specified source directory (and sub-directories contained therein) and compiles them into java class files and javabeans. The files that are created are located in a destination directory as specified by the user.



Note Step 2 includes step 1, however; it is recommended that as a matter of good practice, step 1 is performed prior to step 2.

Syntax Checking

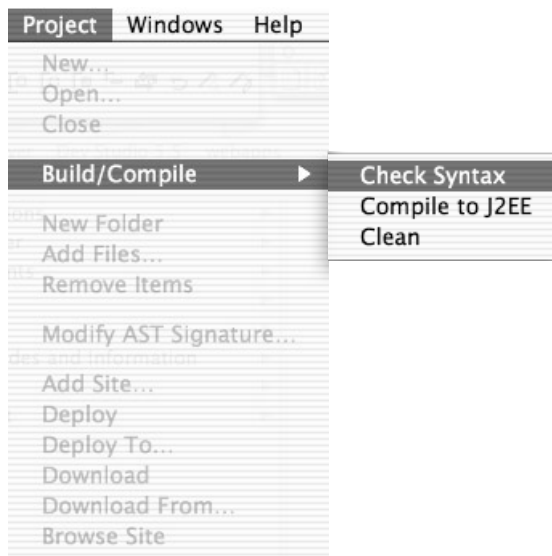
Witango Studio Professional has syntax checking functionality. This function checks all the `taf` and `tcf` files which are located in the specified source directory (and sub-directories contained therein) to ensure that all Witango meta tags exhibit correct syntax. A report is generated for the user.



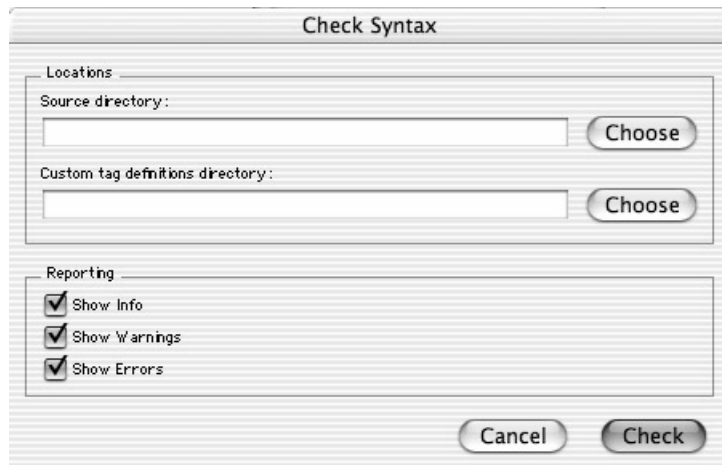
Note Witango Syntax checking only reviews `taf` and `tcf` files in the source directory and sub-directory, `.html`, `.tml` and `.inc` files are NOT checked by this function.

Creating a Syntax Check Report

- 1 Select **Build/Compile** from the **Project** menu, a sub-menu of compiling options as shown below will appear.



- 2 Select the **Syntax Check** option and the Syntax Check window will appear:



The image shows a 'Check Syntax' dialog box. It has a title bar 'Check Syntax'. Inside, there are two sections: 'Locations' and 'Reporting'. The 'Locations' section has two text input fields: 'Source directory:' and 'Custom tag definitions directory:'. Each field has a 'Choose' button to its right. The 'Reporting' section has three checked checkboxes: 'Show Info', 'Show Warnings', and 'Show Errors'. At the bottom right, there are 'Cancel' and 'Check' buttons.

- 3 Complete the following information in the Syntax Check Window and select the **CHECK** button.

Location Information

- **Source Directory**

Use the **BROWSE** button to locate the source directory for your syntax check. That is, the directory which contains the `taf` and `tcf` files you wish to check.



Note You cannot specify just one `taf` file, all `taf` and `tcf` files in the directory and sub-directories will be syntax checked.



Note If the `taf` or `tcf` files reference include files using `<@INCLUDE>` or Presentation action or File action, these include files will not be syntax checked.

- **Custom Tags Directory**

Use the **BROWSE** button to locate the directory which contains custom meta tag definition files (`.xml` files). All `.xml` files under this directory and sub-directories will be validated against Witango `ctags.dtd`.

Report Information



- **Show Information**

Check this checkbox if you want the Syntax Report to show information such as the status of the syntax check, ie, which file it is currently working on.



- **Show warnings**

Check this checkbox if you want the Syntax Report to show warnings.



- **Show Errors**

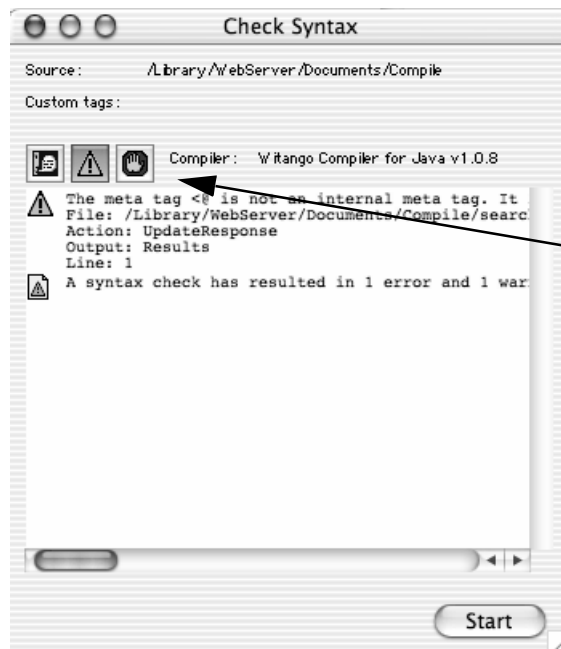
Check this checkbox if you want the Syntax Report to show errors.



Note These checkboxes will not prevent you accessing the information once in the syntax report, they only affect which part of the report is immediately visible to the user.

The default values for the settings can be customised in user preferences. For more information, see "Compile" on page 138.

4 The Syntax Check window as pictured below will appear.



Report
Option
Buttons

The above report shows a Syntax Report which has been run with the error and warning reporting option checkboxes checked. The Report can also be filtered to assist the processing

of the results by selecting the appropriate report option button, see the above picture.

Filtering the Syntax Check Report

The Syntax Check Report can be filtered to show any combination of the three reporting options available.

To filter the Syntax Check Report, the user simply presses the Report Option Buttons which toggle on/off the information.



The image below shows the report being filtered to show only errors:



The Syntax Report can be similarly filtered on errors and warnings, or, any combination of the three reporting options.

Understanding a Syntax Check Report

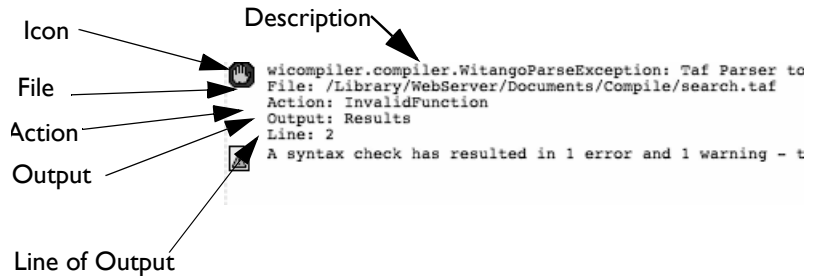
The Syntax Report lists a series of entries detected in the Syntax Check followed by an overall report summary.

Report Entries

Report Entries have the following information contained in them:

- An **icon** which graphically represents to the user the type of entry this is. It can be either an information entry, a warning entry or an error entry.
- A **description** which outlines the issue detected by the Syntax Checker;
- Where appropriate, a full path of the `taf` or `tcf` file to which this entry relates.
- Where appropriate, the **action** where this entry issue is located, the **output option** (Results HTML, No ResultsHTML or Error HTML) and the **line number** in that output HTML which causes the issue.

For entry in the example pictured below:

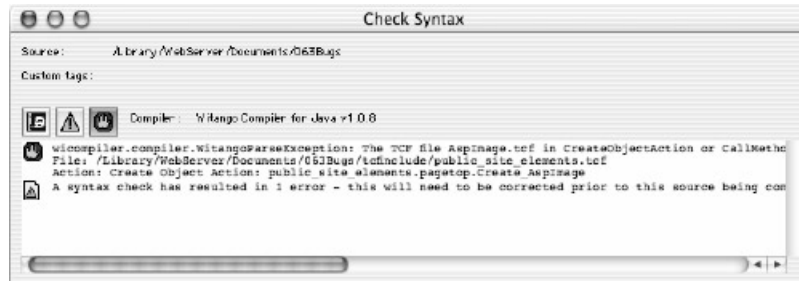


The error would be located on line 2 of the Results HTML of the action Invalid Function in search.taf.



Note Some entries will not be able to be tied to a specific action, when this happens, no information relating to line number or action will appear.

The entry in the example pictured below demonstrates that some errors may not be linked directly to an action and will therefore not include specific location information.



Overall Summary Report

The report is finished with an overall summary at the end, this summary briefly states the outcome of the Syntax Check. This summary identifies:

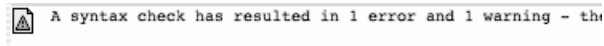
Status of the Syntax Check

- **Successful:** it is considered that the Syntax Check was successful and no errors or warnings were encountered.
- **Successful with warnings:** it is considered that the Syntax Check was successful and no errors were encountered. However issues were encountered which may cause execution time errors.
- **Unsuccessful:** it is considered that the Syntax Check was unsuccessful and issues were encountered which will definitely cause errors upon execution.

Summary of the Syntax Check

- count on the number of errors and the number of warnings detected.

For example, in the entry pictured below:

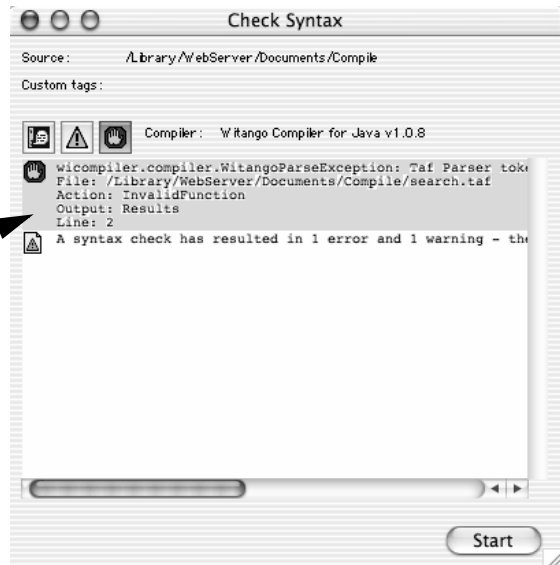


Correcting Issues located in a Syntax Check Report

Where a syntax issue is located in the output HTML of an action, the syntax report will identify the location of the issue. For more information, see “Understanding a Syntax Check Report” on page 426. In these cases the Syntax Check Report allows the user to double-click on an issue to open the corresponding `tcf` or `taf` file at the location of the issue.

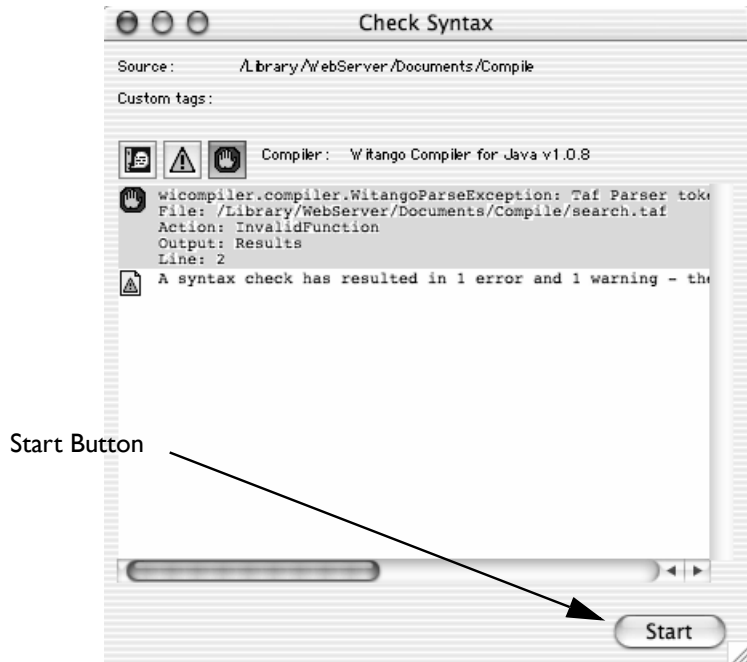
The image below shows an error which has been selected. By double clicking on this error Witango will open the associated taf file at put the cursor at Line 1 of the Results HTML screen of action Redirect.

Double
Click
Here



Rechecking the Syntax

Once a syntax issue has been fixed and saved within your taf or tcf file you can simply push the start button to execute the Syntax Check again.

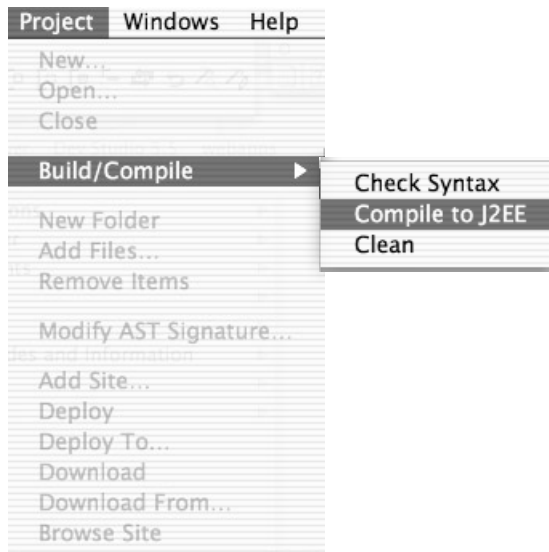


Compiling your Witango Application

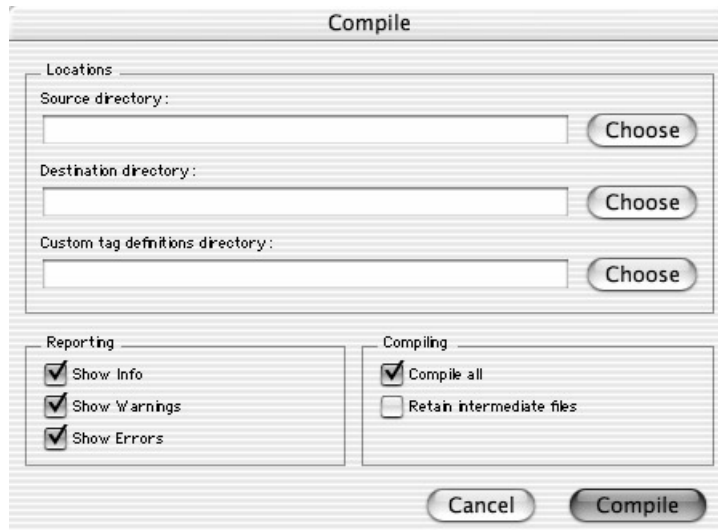
Witango Studio Professional has the functionality to compile your Witango applications for deployment on J2EE compliant web servers. This function compiles all the `taf` and `tcf` files which are located in the specified source directory (and sub-directories contained therein) into java servlets which can be executed on J2EE compliant web servers which have the Witango runtime library installed.

Executing a compile for J2EE

- 1 Select **Build/Compile** from the **Project** menu, a sub-menu of compiling options as shown below will appear.



- 2 Select the **Compile for J2EE** option and the Compile for J2EE window will appear.



- 3 Complete the following information in the Compile for J2EE Window and select the **COMPILE** button.

Location Information

- **Source Directory**

Use the **BROWSE** button to locate the source directory you wish to compile to J2EE. That is, the directory which contains the `taf` and `tcf` files you wish to compile.



Note You cannot specify just one `taf/tcf` file, all `taf/tcf` files in the directory and subdirectories will be compiled.



Note If the `taf` or `tcf` files reference include files (`images`, `.html`, `.html`, `.inc`, `.tml` etc), these files will NOT be included in the compile. They will be included at execution time, and will therefore need to be manually deployed to the deployment directory before execution.

- **Destination Directory**

Use the **BROWSE** button to locate the destination directory you wish Witango to place the resulting servlets in. If such a directory does not exist it will be created on behalf of the current user.

- **Custom Tags Directory**

Use the **BROWSE** button to locate the directory which contains definition files (.xml files) of any custom meta tags which may be referenced in the `taf` and `tcf` files you are compiling.

Report Information - prior to the source being compiled a syntax check is run against them. This syntax check results in a syntax report. The syntax report can be filtered to show any combination of general information, warnings and errors.

Show Information

- Check this checkbox if you want the Syntax Report to show information such as the status of the syntax check, ie, which file it is currently working on.

Show warnings

- Check this checkbox if you want the Syntax Report to show warnings. Warnings are issues which will not stop the compile process but may cause issues when the resulting servlets are executed.

Show Errors

- Check this checkbox if you want the Syntax Report to show errors. Errors are issues which will definitely cause problems during runtime, therefore, if errors are encountered during the syntax check, the compile process does not continue.



Note These checkboxes will not prevent you accessing the information once in the syntax report, they only affect which part of the report is immediately visible to the user.

The default values for the settings can be customised in user preferences. For more information, see "Compile" on page 138

Compile Information - the compile information settings allow the user to customise the compile function such that.

• Compile All

Check this checkbox if you want the entire directory of source files to be compiled. If the checkbox is not checked, then only those files which have been modified since the last compile was run will be compiled.

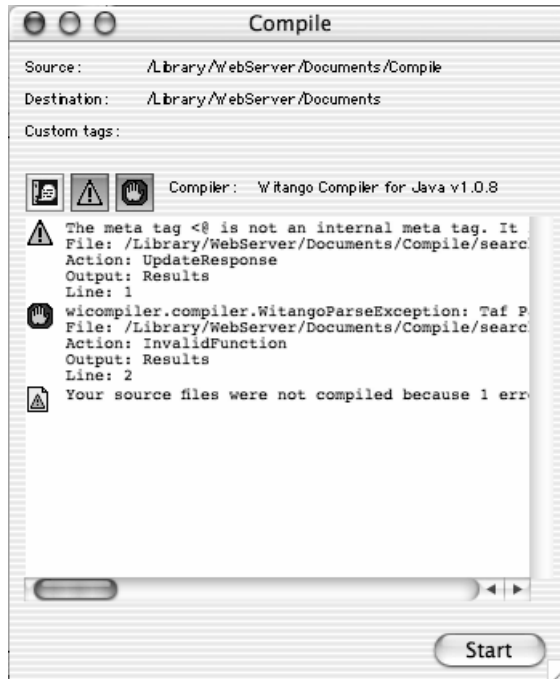
• Retain Intermediate Files

Once a successful syntax check is run on the source directory, there are two steps which the compile facility undertakes to generate the servlets. The first step is to take the `.taf` and

.tcf files to .java files. The second step is to take the .java files to .class files. If the user wishes to retain the .java files, this checkbox should be checked. The more usual approach would be to run the compile without this option checked.

The default values for the settings can be customised in user preferences. For more information, see "Compile" on page 138.

4 The Compile for J2EE window as pictured below will appear.



You will receive a report as to whether the compile process was successful. There are 3 possible outcomes:

- 1 **Successful compilation** - this is where no issues were encountered and the resulting servlets will now exist in your destination directory ready for deployment.
- 2 **Successful compile with warnings** - this is where the compile process is complete, but issues have been flagged to the user. These issues may result in errors when executed and should therefore be carefully reviewed by the user. See Correcting Issues located in a Syntax Check Reportpage 428 for more information on how to correct these issues.
- 3 **Unsuccessful compile** - this is where the compile process was not completed because errors were encountered in the source file which would definitely cause runtime errors . See Correcting Issues

located in a Syntax Check Reportpage 428 for more information on how to correct these issues.

Glossary of Terms

An Alphabetical Reference of Common Witango and Internet Terms

- action** This is short for *Witango action*. Witango Studio has a suite of actions which do many different tasks, including: getting data from or sending data to a database, invoking external actions (such as reading and writing files and sending email), and controlling application file execution. Actions form the basis of an application file in Witango Studio.
- applet** A small Java program that can be embedded in an HTML page. Applets differ from full-fledged Java applications in that they are not allowed to access certain resources on the local computer, such as files and serial devices, and are prohibited from communicating with most other computers across a network. The current rule is that an applet can only make an Internet connection to the computer from which the applet was sent.
- application** In the software industry, an application generally means a program for end-users. In Witango, application has specific meanings depending on the context. See also *scope*, *Witango application* and *Witango application file*.
- application file** See *Witango application file*.
- ASCII** **American Standard Code for Information Interchange**
This is the *de facto* world-wide standard for the code numbers used by computers to represent all the upper- and lower-case Latin letters, numbers, punctuation, and related data. There are 128 standard ASCII codes each of which can be represented by a seven-digit binary number.
- AST** **Application-Specific Witango**
An AST is a Witango Server that you can distribute with a branded Witango application. This gives an end-user the ability to execute your solution without having to purchase a full Witango Server for that single

application. All files that are accessed this way must have an AST signature. Other Witango application files can not be run with an AST.

attribute

In the context of an HTML window, an attribute is the Results HTML, No Results HTML, or Error HTML. These windows allow you to enter messages for the various outcomes.

In the context of meta tags, an attribute is a name/value pair that specifies certain required or optional criteria.

CGI

Common Gateway Interface

A set of rules that describes how a Web server communicates with another piece of software, often on the same machine, and how the other piece of software (the “CGI program”) talks to the Web server. Any piece of software can be a CGI program if it handles input and output according to the CGI standard.

Usually a CGI program is a small program that takes data from a Web server and does something with it, like putting the content of a form into an e-mail message, or turning the data into a database query.

You can often see that a CGI program is being used when “cgi-bin” appears in a URL.

cgi-bin

The most common name of a directory on a Web server in which CGI programs are stored.

The “bin” part of “cgi-bin” is a shorthand for “binary”, because once upon a time, most programs were referred to as “binaries”. In real life, most programs found in cgi-bin directories are text files—scripts that are executed by binaries located elsewhere on the same machine.

class

A category of objects defined by all the common properties of the different objects that belong to that category.

client

A software program that is used to contact and get data from a server software program on another computer, often across a network or the Internet. Each client program is designed to work with one or more specific kinds of server programs, and each server requires a specific kind of client. A Web browser is a specific kind of client.

COM object

Component Object Model object

Objects that conform to the COM objects specifications developed by Microsoft. COM objects can run only on the Windows platform. Witango

Studio and Witango Server support the use of COM objects on Windows.

configuration variables

Special values that control aspects of Witango behavior. System variables affect system wide settings. They apply to all users of Witango Server.

cookie

The most common meaning of *cookie* on the Internet refers to a piece of information sent by a Web server to a Web browser that the browser software is expected to save and to send back to the Web server whenever the Web browser makes additional requests from the Web server.

Depending on the type of cookie used and the Web browser's settings, the Web browser may accept or not accept the cookie, and may save the cookie for either a short time or a long time.

Cookies might contain login or registration information, on-line shopping cart information, or user preferences.

When a Web server receives a request from a Web browser that includes a cookie, the Web server is able to use the information stored in the cookie. For example, the Web server might customize what is sent back to the user, or keep a log of particular users' requests.

Cookies are usually set to expire after a predetermined amount of time and are usually saved in memory until the Web browser software is closed down, at which time they may be saved to disk if their expire time has not been reached.

data source

An abstraction or description of the database that Witango Studio and Witango Server are referencing.

data type

In programming, a data type is a classification of data based on certain characteristics. You normally do not have to be concerned with data types when you develop Witango application files. However, you encounter data types when you use objects because object vendors often specify data requirements for using their object. Witango facilitates the use of COM objects (Windows-only), JavaBeans, and Witango class files with the same application file by converting the various data types to Witango data types, whenever possible.

DCOM object

Distributed COM object

The DCOM environment deploys COM objects on machines other than

the one running Witango Server. Witango supports the DCOM environment on Windows.

document instance

An XML document represented using DOM. Once an XML document has been converted to a document instance, you can manipulate the document instance using Witango meta tags.

DOM

Document Object Model

A World Wide Web Consortium (W3C) standard for the manipulation of structured data, including XML.

DOM, as the name implies, allows Witango developers to manipulate the elements of a structured document (for example, XML) as if they were objects. Developers can build document instances, navigate their structure, and add, modify, or delete elements and content. DOM creates a representation of an XML document that is an object tree, and gives you the tools to create and manipulate the object tree in Witango using Witango variables and meta tags.

domain name

The unique name that identifies an Internet site. Domain names always have two or more parts separated by dots. The part on the left is the most specific, and the part on the right is the most general. A given machine may have more than one domain name, but a given domain name points to only one machine. For example, the domain names

```
example.com  
training.example.com  
mail.example.com
```

can all refer to the same machine, but each domain name can refer to no more than one machine.

Usually, all of the machines on a given network share the right-hand portion of their domain names (*example.com* in the examples above). It is also possible for a domain name to exist but not be connected to an actual machine. This is often done so that a group or business can have an Internet e-mail address without having to establish a real Internet site. In these cases, some real Internet machine must handle the mail on behalf of the listed domain name.

DNS

Domain Name Server

A machine on the Internet that converts (“resolves”) domain names to IP address numbers.

DTD

Document Type Definition

SGML and XML specifications require a DTD, which defines the structure of the various elements that make up an XML document, and ensures that all applications that read from and write to it do so in a consistent way. It is, in effect, the schema of the document.

firewall

A combination of hardware and software that separates a LAN into two or more parts for security purposes. See also *proxy server*.

FTP

File Transfer Protocol

A standard method for transferring files between machines or between a client machine and a file server on the Internet. FTP allows a client machine to log in to a server machine to send or retrieve files.

Within a Witango project, you can define an FTP site and deploy (upload) files defined in your project to another computer while preserving the hierarchy structure of your project files. You can also download files from a remote site to replicate a project or share projects with other developers.

gateway

A hardware or software setup that translates between two dissimilar protocols. For example, Prodigy has a gateway that translates between its internal, proprietary e-mail format and Internet e-mail format. Another meaning of gateway is to describe any mechanism for providing access to another system, for example, AOL might be called a gateway to the Internet.

hit

Each time a Web server sends a file to a Web browser, it is recorded in the Web server log file as a *hit*. Hits are generated for every element of a requested page (including graphics, text and interactive items). If a page containing two graphics is viewed by a user, three hits are recorded (one for the page itself and one for each graphic).

Hits are often used as a rough measure of load on a Web server, such as 300,000 hits per month. Because each hit can represent anything from a request for a tiny document (or even a request for a missing document) all the way to a request that requires some significant extra processing (such as a complex search request), the actual load on a machine from one hit is almost impossible to define.

HTML

HyperText Markup Language

The coding language used to create hypertext documents for use on the World Wide Web. HTML looks a lot like old-fashioned typesetting code,

where you surround a block of text with codes that indicate how it should appear. Additionally, in HTML you can specify that a block of text, or a word, is linked to another file on the Internet. HTML files are meant to be viewed using a World Wide Web client program, such as Netscape Navigator or Microsoft Internet Explorer.

HTTP

HyperText Transfer Protocol

The protocol for moving hypertext files across the Internet. Requires a HTTP client program on one end (Web browser), and an HTTP server (Web server) program on the other end. HTTP is the most important protocol used in the World Wide Web.

HTTP header

Header fields in HTTP requests and responses. A request header contains information about the request and about the client itself (such as e-mail address, Web browser type, and platform). A response header contains information about the Web server and the HTML document returned to the client. Headers can also contain cookies.

HTTP method

HTTP methods are used by Web browsers to request and submit information on the World Wide Web (WWW). Web browsers request information from a Web server when they want to display information (such as pages). Web browsers can submit information as well. For example, a visitor may fill out a form on a Web site and submit this information to the Web server. Common methods include:

- `GET`, which is used to retrieve a page from a Web server
- `HEAD`, which is used to check whether a page has been changed
- `POST`, which is used to submit form data.

HTTP request

Sent by a client, typically a Web browser, to a Web server asking the Web server to retrieve some unit of content (for example, HTML pages, images, or files).

HTTP response

Sent by a Web server in response to a request by a client, typically a Web browser. For example, a Web server may return HTML pages, images, sound files or video files to a client.

HTTP result code

HTTP result codes indicate whether a Web transaction is successful. Transactions occur whenever visitors request information from a Web server and the Web server returns this information. For example, a typical transaction occurs when a visitor clicks on a link to access a Web page and the Web server returns that page to the visitor.

Every transaction has a result code. If the transaction is successful, the visitor never sees the result code. If there is an error, the visitor may see the result code, however. A common result code seen by visitors is “404 Page Not Found”. HTTP success codes begin with 2 or 3, and error codes begin with 4 or 5.

intranet

A private network inside a company or organization that uses the same kinds of software that you would find on the public Internet, but that is only for internal use.

As the Internet has become more popular many of the tools used on the Internet are being used in private networks, for example, many companies have Web servers that are available only to employees.

IP address

Internet Protocol Address

A unique number consisting of four parts separated by dots, such as 207.107.95.106. Each of the four sections is a number from 0 to 255. Every system connected to the Internet has a unique IP address. Most people use domain names in addition to IP addresses, and the resolution between domain names and IP addresses is handled by Domain Name Servers.

It is difficult to use IP addresses to accurately identify visitors. IP addresses are reused and redistributed to visitors who use Internet Server Providers and dial-up servers to access the Internet. This is called *dynamic IP addressing*. However, visitors can be recognized persistently with cookies even if they use different IP addresses.

ISDN

Integrated Services Digital Network

A way to move more data over existing regular phone lines. It can provide speeds of roughly 128,000 bits per second over regular phone lines. In practice, most people will be limited to 56,000 or 64,000 bits per second.

JAS

Java Application Server

A JAS is an application that accepts requests from Witango to execute Java class files, and returns the results of that execution back to Witango. Witango comes with a JAS.

Java

Java is a network-oriented programming language invented by Sun Microsystems that is specifically designed for writing programs that can be safely downloaded to your computer through the Internet and immediately run without fear of viruses or other harm to your computer

or files. Using small Java programs (called *applets*), Web pages can include functions such as animations, calculators, and other fancy tricks.

JavaBean

A component technology for Java that lets developers create reusable software objects. These objects can be shared. A database vendor can create a Java bean to support its software, and other developers can easily drop the bean into their own projects. You can incorporate JavaBeans in Witango application files on all platforms.

Java class

In Java, a type that defines the implementation of a particular kind of object. A class definition defines instance and class variables and methods, as well as specifying the interfaces the class implements and the immediate superclass of the class. If the superclass is not explicitly specified, the superclass will implicitly be `Object`.

JRE

Java Runtime Environment

A type of virtual machine, a JRE allows the running of Java classes or JavaBeans. See also *JVM* and *virtual machine*.

JVM

Java Virtual Machine

An interface between the Java language and the hardware platform processing the data. Once a specific platform has a JVM, it can run any Java program. In Witango, a JVM is an environment where you can run Java classes or JavaBeans. See also *virtual machine*.

link

Any text on a Web site that can be chosen by a visitor and which causes another document to be retrieved and displayed. Also known as a “hot link” or “hypertext link”.

Linux

A freely-distributed implementation of UNIX that runs on a number of hardware platforms.

load-splitting

Splitting a Web site amongst two or more Witango servers in order to improve processing speed and performance as the site’s size and traffic volume increase.

Witango is scalable; it allows you to do load-splitting without having to alter your Witango application files. As your Web site grows, you do not have to go through another cycle of development and testing of your applications every time you add a Witango Server.

meta tag

The basic component of a tag language unique to Witango Server. Meta tags communicate with Witango Server in the same way that HTML communicates with a Web server.

method

The interface of an object consists of one or more methods. A method allows you to tell the object to input data, get data, or carry out any other action.

MIME

Multipurpose Internet Mail Extensions

A standard for attaching non-text files to standard Internet mail messages. Non-text files include graphics, spreadsheets, formatted word-processor documents, sound files, and most other files.

An e-mail program is said to be MIME Compliant if it can both send and receive files using the MIME standard.

Generally speaking, the MIME standard is a way of specifying both the type of file being sent (for example, a Quicktime video file), and the method that should be used to turn it back into its original form.

Besides e-mail software, the MIME standard is also universally used by Web Servers to identify the files they are sending to Web clients. In this way, new file formats can be accommodated simply by updating the Web browsers' list of pairs of MIME-types and appropriate software for handling each type.

object

A reusable software component. Witango supports the use of objects in Witango application files. The use of objects can simplify the development process and reduce development time.

Witango supports different object types:

- COM objects (Windows-only)
- JavaBeans
- Witango class files.

object instance

When Witango Server executes a Call Method action in a Witango application file, it creates an *object instance* (or simply, instance) for a class—such as a COM object—as soon as it encounters a Call Method action associated with that class.

ODBC

Open Database Connectivity

A standard set by Microsoft that allows applications to communicate with a variety of databases from different vendors. An ODBC client application

talks to the ODBC driver manager, which in turn talks to a database driver for a specific type of database.

parameter

The basic data elements of a method. A parameter defines what the object takes as input, output, or both. Each method consists of one or more parameters.

plug-in

A (usually small) piece of software that adds features to a larger piece of software. A common example of a plug-in is for the Netscape Navigator Web browser and Web server. The idea behind plug-ins is that a small piece of software is loaded into memory by the larger program, adding a new feature, and that users need only install the few plug-ins that they need, out of a much larger pool of possibilities. Plug-ins are usually created by people other than the publishers of the software the plug-in works with.

port

In TCP/IP and networks, it is an endpoint to a logical connection. The port number identifies what type of port it is. For example, port 80 is generally used for HTTP traffic.

proxy server

A server that sits between a client application, such as a Web browser and a real server. It intercepts all requests to the real server to see if it can fulfill the request itself. If not, it forwards the request to the real server. Witango allows you to route files through a TIS (Trusted Information Server) proxy server. See also *firewall*.

scope

In Witango, a scope refers to a characteristic of variables that determines where they are valid. The following scopes are available for variables in Witango application files and Witango class files: local, user, cookie, application, domain, system, and custom. In addition, the following scopes are available for variables in Witango class files only: method and instance. See Chapter 8 for details.

search argument

A search argument is the part of a URL used when a set of arguments are sent to an executing program, such as a CGI, or Web server. Search arguments are commonly used for forms and searches. The search argument follows a question mark (?) in the URL and can be used to track dynamic content or CGI variables being passed between client and server.

For example, in this URL:

```
http://www.example.com/test.taf?function=form
```

the search argument is `function=form`.

security certificate

Information (often stored as a text file) that is used by the SSL protocol to establish a secure connection.

Security certificates contain information about who it belongs to, who it was issued by, a unique serial number or other unique identification, valid dates, and an encrypted fingerprint that can be used to verify the contents of the certificate.

For an SSL connection to be created, both sides must have a valid Security Certificate.

server

A computer, or a software package, that provides a specific kind of service to client software running on other computers. The term can refer to a particular piece of software, such as a Web server, or to the machine on which the software is running. A single server machine could have several different server software packages running on it, thus providing many different servers to clients on the network.

server push

A way to deliver information from a Web server to a Web browser. The information is sent to the Web browser without a client request; for example, a site that automatically updates a Web browser with the latest news.

Server Watcher

A watching process that works with Windows- and UNIX-based Witango Servers to ensure that Witango is always running. While Witango attempts to recover from a fatal error automatically, it does not always succeed for various reasons. When this happens, the watching process provides a simple and robust external process to relaunch Witango automatically.

SGML

Standard Generalized Markup Language

The International Organization for Standardization (ISO) chose SGML as the tool used to organize and tag elements (for example, titles, sections, and paragraphs) of a document. SGML specifies the rules for tagging elements, but not the formatting of documents. These tags can then be interpreted to format elements in different ways.

SGML is useful for managing large documents that are subject to frequent revisions and that need to be printed in different formats.

SMTP

Simple Mail Transfer Protocol

The main protocol used to send mail on the Internet. SMTP consists of a set of rules for how a program sending mail and a program receiving mail should interact.

Almost all Internet mail is sent and received by clients and servers using SMTP, thus if one wanted to set up an e-mail server on the Internet one would look for e-mail server software that supports SMTP.

Solaris

A UNIX-based operating system developed by Sun Microsystems. Originally developed to run on Sun's SPARC workstations, Solaris now runs on many workstations from other vendors.

SQL

Structured Query Language

A unified language for defining, querying, modifying and controlling the data in a relational database. Most industrial-strength relational databases and many smaller database applications are addressed using SQL. Each specific application has its own version of SQL implementing features unique to that application, but all SQL-capable databases support a common subset of SQL.

SSL

Secure Sockets Layer

A protocol designed by Netscape to enable encrypted, authenticated communications across the Internet.

SSL is used mostly (but not exclusively) in communications between Web browsers and Web servers. URLs that begin with "https" indicate that an SSL connection will be used.

SSL provides three important features: privacy, authentication, and message integrity.

In an SSL connection each side of the connection must have a Security Certificate, which each side's software sends to the other. Each side then encrypts what it sends using information from both its own and the other side's Certificate, ensuring that only the intended recipient can decrypt it, and that the other side can be sure the data came from the place it claims to have come from, and that the message has not been tampered with.

Witango application

A group of Witango application files in a particular application folder that can share variables in an application scope.

Witango application file

Witango application files are written using Witango Studio and are composed of one or a series of actions that are executed by Witango

Server. Each action's reaction or response from a database, server, external program, and so on, can be posted in HTML. When a Witango application file is completed, the results are returned to the client. Witango application files are sometimes called application files. They generally have a `.taf` file extension.

Witango CGI

The CGI that links Witango Server and your Web server. Not necessary if you use one of the Web server plug-ins.

Witango class file

Reusable software components that you can incorporate in Witango application files. You can create and edit Witango class files using Witango Studio. Witango class files generally have a `.tcf` file extension.

Witango client

The Witango client receives requests made by a Web browser, then redirects the request to a specified Witango Server. This allows your Witango configuration to perform load-splitting. The Witango client is either a CGI or a plug-in, and is located on the Web server machine. See also *load-splitting*.

Witango Server

Executes and serves up the application files by which clients can interact with HTML pages to perform a variety of tasks; for example, querying the data source.

TCP/IP**Transmission Control Protocol/Internet Protocol**

This is the suite of protocols that defines the Internet. Originally designed for the UNIX operating system, TCP/IP software is now available for every major kind of computer operating system. To be truly on the Internet, your computer must have TCP/IP software.

Unix

A computer operating system (the basic software running on a computer, underneath programs such as word processors and spreadsheets). Unix is designed to be used by many people at the same time (it is multi-user) and has TCP/IP built-in. It is the most common operating system for servers on the Internet.

URL**Uniform Resource Locator**

The “address system” used by the World Wide Web (WWW). A URL is the address of a piece of information stored on a Web server. This information can include HTML pages, graphics, multimedia files, Java classes, downloadable files or any other type of file you have stored.

For example, this URL:

`http://www.example.com/path/subdir/file.htm`

specifies the resource `file.htm` located on the server
`www.example.com`.

virtual hosting With virtual hosting, one physical host is actually many virtual hosts. With hardware virtual hosting, a single machine can act like multiple machines (with multiple domain names and IP addresses). With software virtual hosting, a single machine can act as multiple servers, but only use one IP address.

virtual machine A virtual machine acts as an interface between a code and microprocessor. It is program that functions as a machine without having any physical properties. The machine is an abstract, as opposed to a physical, entity. A Java Virtual Machine (JVM) is an example of a virtual machine.

Web browser A software program that can request, load and display documents available on the World Wide Web. Web browsers are typically operated by people, but can also be run by robots, such as Web crawlers and intelligent agents.

Web server A computer that stores the files comprising a Web site. It sends information to, and accepts information from, Web browsers.

Web Server document root All files that are served on the Web server must be placed in the Web Server document root. When the files are needed, the Web Server looks in the root or its subfolders to access them. A `Witango 2000` folder is created inside the Web server document root when Witango is installed on your machine.

Web site The collection of elements (such as HTML files, images, video or audio files) the comprise an entity on the Internet. A Web site is equivalent to a Web domain. It is identified by its domain name, for example `www.example.com`. One or more Web sites can run on a physical machine under one Web server process.

XML **Extensible Markup Language**
XML is a text-based and widely-endorsed standard markup language, similar to HTML, but much more flexible and robust. It is a subset of SGML. Its goal is to enable generic SGML (that is, structured documents) to be served, received, and processed on the Web in the way that is now

possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Index

Symbols

@ and /@ 143
¥ 123

A

`absolutePathPrefix` 302, 308, 314, 315,
319, 320, 322, 323

action

See *also* Group action and builder
about 225, 226
adding 40, 229
assigning data source to 101, 115
attribute 40, 229
 See *also* results HTML, no results HTML,
 error HTML, and push
 assigning 46, 236
 indicator icon 47, 237
copying 43, 44, 232, 233
deleting 42, 231
dragging into SQL query text window 25
editing 42, 232
generated by builder 167, 206, 220
jumping to another
 See Branch action
moving 43, 232
multi-column list 19
naming and renaming 41, 230
nested 282, 290
organizing
 See Group action
properties 45, 107, 234
redirecting flow of
 See control action
repeating
 See loop action
setting development or deployment data
 source 106

action, name of

See *also the names of the specific actions*
Assign 155, 227
Begin Transaction 226, 325
Branch 228, 274
Break 228, 294
Call Method 227, 350

conditional action 280
control action 273
Create Object Instance 227, 349
database 251, 325
Delete 226, 270
Direct DBMS 226, 325
Else 227, 280
Else If 227, 280
End Transaction 226, 325
External 227, 297
File 227, 317
For Loop 227, 289
Group 227, 245
If 227, 280
Insert 226, 266
loop action 288
Mail 227, 309
Objects Loop 227, 391
Presentation 227
Results 227
Return 228, 295
Script 227, 297
Search 226, 252
transaction 325
Update 226, 268
While Loop 227, 289

ACTIONRESULT meta tag 153

actions bar 226, 228

alias

Oracle data source 114

and operator 258, 284

application file

See *also* action, Group action, builder, and
project
about 1, 55
assigning AST signature to file 76
changed but not saved 57
creating 57
debugging 46, 61, 236
dirty file indicator 57
dragging column into 21
inserting meta tag 148
run-only 59
saving 58
specifying URL of 60

- window 39, 56, 229
 - XML format 55
- ARG meta tag 149, 150, 206, 220, 261
- array
 - about 155
- Assign action 156, 227
 - defining variable 155
 - in Witango class file 403
- ASSIGN meta tag 144, 155
- AST signature
 - for application file 76
 - for project 79, 82
 - overwriting 76, 79
 - valid characters 80
- attribute
 - contextual menu 13
- attribute, associated with action 46, 236
- attribute, associated with meta tag 144, 147
- attribute, associated with object 373, 375, 384
- Attributes menu
 - attribute HTML 13
- automation server 355, 366
- AVG function 255

B

- beanpaths.ini 363, 369
- Begin Transaction action 226, 325, 326
- begins with operator 259, 260
- BIND meta tag 335
- Branch action 228, 274
 - destination rules 274
 - executing 276
 - in Witango class file 274, 402
 - selecting destination 278
 - setting up 277
 - to action group 248
 - to another application file 274, 276
- Break action 228, 294
- builder
 - See also Search Builder and New Record Builder
 - about 165
 - adding to application file 166
 - behaving like action group 167
 - generating actions 169
 - naming 166
 - page format 131, 167
 - replacing existing actions 170

- snippet 119
- business logic 52, 243

C

- cache
 - object information 375
- CALC meta tag 286
- Call Method action 227, 347, 350
 - adding 364, 383, 384
 - completing information for 385
 - nested method call 404
 - object instance variable 386
 - parameter list 387
 - result variable 386
 - self-referencing in Witango class file 404
 - window, example of 385
- CALLMETHOD meta tag 347, 350, 361, 393
- CGIPARAM meta tag 152
- class and object 348, 396
- CLASSPATH environment variable 363, 369
- COL meta tag 49, 239, 265
 - for non-ODBC data source 334
- collection object 391, 393
- column
 - See also Search action, Search Builder, Search page, Record List page, Record Detail page, primary key, and join
 - custom 272
 - grouping 255
 - ordering 253, 255
 - properties 112
 - selecting 253, 254, 257
 - from multiple tables 338
 - snippet 119, 128
- COLUMN meta tag 47, 49, 128, 144, 218, 237, 239, 265
 - for ODBC data source 334
- COM object
 - about 348, 355
 - adding to workspace 365
 - automation server 355, 366
 - executing 362
 - initialization string 381
 - library 366, 373
 - licensing 356
 - non-programmable 367
 - optional parameter 369
 - referenced object 367

- username and password **382**
- variant parameter **388**
- command, in menu **6**
 - See *also* menu
- COMMIT command **326**
- Component Object Model
 - See COM object
- component, Witango
 - See Witango component
- conditional action **280**
 - See *also* If action, Else If action, and Else action
 - nested **282**
- configuration variable
 - snippet **119**
- configuration variable, name of
 - absolutePathPrefix **302, 308, 314, 315, 319, 320, 322, 323**
 - mailDefaultFrom **310**
 - mailPort **314**
 - mailServer **314**
 - passThroughSwitch **103, 105, 109**
 - stripChars **173**
 - TCFSearchPath **363**
- contains operator **259, 260**
- context-sensitive menu
 - for action **234**
 - for action attribute **236**
 - for Branch action **278**
 - project **68**
 - showing Insert Meta Tag **145**
- contextual menu
 - attribute **13**
 - for action **45**
 - for action attribute **47**
- control action **273**
 - See *also* conditional action, loop action, Branch action, Break action, and Return action
- cookie
 - properties **159**
 - setting up **159**
- cookie scope **158**
- COUNT function **255**
- Create Object Instance action **227, 346, 349**
 - adding **364, 378**
 - completing information for **379**
 - initialization string **381**
 - object instance variable **380**
 - window, example of **379**
- CREATEOBJECT meta tag **346, 347, 349, 361**
- CURRENTDATE meta tag **151**

- CURRENTTIME meta tag **151**
- CURRENTTIMESTAMP meta tag **151**

D

- data source
 - See *also* development data source and deployment data source
 - about **83**
 - assigning to action **101, 115**
 - connecting **113**
 - deleting **99**
 - editing **111**
 - handling unknown **100**
 - icon **102**
 - information stored in application file **114**
 - logging on **102**
 - modifying **98, 99**
 - properties **102, 110**
 - reloading **100**
 - selecting column **21**
 - selecting table **113, 132**
 - types supported by Witango **84**
 - using on different computers **114**
 - workspace **21, 85, 110**
- data source, name of
 - See *also the names of the specific data sources*
 - JDBC **84**
 - ODBC **84**
 - Oracle **84**
- data type, of object
 - about **358**
 - converting **358**
 - in parameter list **387, 407**
- database
 - joining tables **325, 336**
 - searching record and retrieving data **252**
- database action **251**
 - See *also* Search action, Insert action, Update action, Delete action, transaction action, and column
 - advanced **325**
 - commit **326, 329**
 - rollback **326**
- database transaction
 - See transaction action
- DCOM environment **356**
- debugging
 - application file **46, 61, 236**

- icon 62
- Witango class file 415
- default method, in Witango class file 401
- Delete action 226, 270
 - executing 271
 - setting up 270
- delete response HTML 202
- dependencies
 - about 74, 82
 - referenced by application file 76
 - resolving 74, 75
- deployment data source 103
 - setting parameters for action 106
 - using meta tag 103, 105
- development data source 102, 103
 - setting parameters for action 106
- Direct DBMS action 226, 325, 330
 - calling stored SQL procedure 335
 - executing 334
 - ODBC and non-ODBC sources 334
 - results HTML 334
 - setting result option 333
 - setting up 330
 - SQL encoding on meta tag value 332
 - using meta tags within 331
- dirty file indicator 57
- Distributed Component Object Model
 - See DCOM environment
- DLL 301
 - calling with External action 297
- document type definition
 - See DTD
- DTD 56
- dynamic linked library
 - See DLL

E

- editing
 - command 11
 - error HTML 13
 - file in project 73
 - finding and replacing 27, 73
 - no results HTML 13
 - results HTML 13
 - selecting text editor 131
 - snippet
 - See snippet, editing
 - using tab character 12
 - word wrap 12
- editing window
 - INCLUDE meta tag 16
- Else action 227, 280
- Else If action 227, 280
 - setting up 283
- e-mail address syntax 311
- e-mail, sending from Witango
 - See Mail action
- End Transaction action 226, 325
 - setting up 328
- ending file processing
 - See Return action
- ends with operator 259, 260
- error HTML
 - See also error message, custom
 - associating with an action 46, 50, 236, 240
 - creating or editing 50, 240
 - using meta tag 50, 240
 - window 13
- error message, custom 51, 241
- ERROR meta tag 50, 240
- ERRORS meta tag 50, 240
- executing
 - application file, using plug-in and CGI
 - application file
 - executing, using plug-in and CGI 60
 - Branch action 276
 - Delete action 271
 - Direct DBMS action 334
 - For Loop action 293
 - Group action 249
 - Insert action 267
 - JavaScript 297, 298
 - Script action 299
 - Search action 264
 - SQL statements 330
 - transaction action 329
 - Update action 269
 - While Loop action 293
 - Witango class file 402
- exiting loop
 - See Break action
- expanding and collapsing parent objects 35
- expiry URL 382
- Extensible Markup Language
 - See XML
- External action 227, 297
 - assigning attribute 305
 - deleting parameter 306

- enabling and disabling 308
- error HTML 305
- executing 306, 308
- no results HTML 305, 307
- results HTML 305, 306
- setting up 301
 - command line 302
 - DLL call 301
 - Java 304

F

- field properties
 - in New Record Builder 214
 - in Search Builder 180
- file
 - creating 15
 - opening 16
 - reading, writing, and deleting
 - See File action
 - saving 15
- File action 227, 317, 318
 - adding 318
 - enabling and disabling 323
 - security 323
 - setting
 - for deleting 321
 - for reading 319
 - for writing 319
 - using meta tag 318
- file, application
 - See application file
- file, special
 - beanpaths.ini 363, 369
 - error.htx 51, 241
 - objects.ini 359
- file, under project
 - See project and presentation page
- find and replace text or regular expression
 - See editing
- folder, under project
 - See project
- footer HTML
 - for New Record Builder 218
 - for Record Detail page 202
 - for Record List page 196
 - for Search page 187
- For Loop action 227, 289, 291
 - executing 293
- function, name of

- AVG, COUNT, MAX, MIN, SUM, and none 255

G

- GETPARAM meta tag 413
- getter method 373
- graphical user interface 6
 - viewing component 7
- greater than operator 259, 260, 285
- greater than or equal to operator 259, 260, 285
- Group action 227, 246
 - adding action to group 247
 - adding group to application file 247
 - branching to 248
 - deleting group 248
 - executing 249
 - removing action from group 247
 - ungrouping actions 248
- GUI
 - See graphical user interface

H

- header HTML
 - for New Record Builder 218
 - for Record Detail page 202
 - for Record List page 196
 - for Search page 187
- help, on-line 132
- HTML
 - Attribute menu 13
 - color-coding 12
 - editing window 9
- HTTP
 - server
 - See Web server

I

- If action 227, 280
 - parameter
 - logical operator 284
 - operator 284
 - setting up 283
 - specifying advanced parameters 285
 - specifying basic parameters 284

- IF meta tag 144, 332
- IFEMPTY meta tag 144, 332
- IFEQUAL meta tag 144, 332
- INCLUDE
 - HTML editing window 16
 - meta tag 16
- include empty
 - optional parameter 389
 - search criteria 261
- INCLUDE meta tag 145, 299
- included files
 - opening in HTML editing window 16
- initialization string 381
- Insert action 226, 266
 - executing 267
 - results returned 267
 - setting up 266
- instance scope 403, 404
- interface
 - viewing component 7
- interface, Tango Editor 6
- introspection of object 347
- is empty operator 285
- is equal to operator 259, 260, 285
- is in operator 259, 260, 261
- is not empty operator 285
- is not equal to operator 259, 260, 285
- is not null operator 259, 260
- is null operator 259, 260
- item variable, Objects Loop action 393

J

- Java 304
 - enabling and disabling 308
 - executing 306
- Java action
 - configuring 304
- Java class file
 - calling with External action 297
- Java server 304, 305
- JavaBean
 - about 348, 356
 - adding to workspace 363, 367
 - executing 363
 - Java archive file 368, 369, 373
- JavaScript 297
 - enabling and disabling 308
 - executing 298

- object and variable scope 298
- using meta tag 299

join

- about 336
- and data sources 338
- creating 337
 - in Search action 338
 - in Search Builder 205, 341
- deleting 340
- editing 337, 340
- standard and outer 325, 336

join operator 339

K

keyboard shortcut 34

L

- less than operator 259, 260, 285
- less than or equal to operator 259, 260, 285
- logical operator 258
 - See *also* operator
- loop action 288
 - See *also* For Loop action, While Loop action, and Objects Loop action
 - nested 290

M

- Mail action 227, 309
 - attaching file 313
 - enabling and disabling 315
 - setting up 310
 - specifying options 312
 - using meta tag 310, 314
- mailDefaultFrom 310
- mailPort 314
- mailServer 314
- MAX function 255
- menu
 - See *also* keyboard shortcut
 - Attributes 13
 - Edit 10
 - File 15
 - View 7
 - Windows 7

meta tag
 about 143, 144
 attribute 147
 case sensitivity 144
 category
 action result item 152
 current date/time value 151
 form field or URL argument value 149
 request parameter 152
 variable value 150
 color-coding 12
 combining with other meta tags 146
 in data source 109
 in deployment data source 105
 in error HTML 50, 240
 in External action 305
 in JavaScript 299
 in no results HTML 49, 239
 in results HTML 47, 49, 237, 239
 inserting 105, 148
 using with Direct DBMS action 331
 where to use 145
meta tag, name of
 ACTIONRESULT 153
 ARG 149, 150, 206, 220, 261
 ASSIGN 144, 155
 BIND 335
 CALC 286
 CALLMETHOD 347, 350, 361, 393
 CGIPARAM 152
 COL 49, 239, 265, 334
 COLUMN 47, 49, 128, 144, 218, 237, 239, 265, 334
 CREATEOBJECT 346, 347, 349, 361
 CURRENTDATE 151
 CURRENTTIME 151
 CURRENTTIMESTAMP 151
 ERROR 50, 240
 ERRORS 50, 240
 GETPARAM 413
 IF 144, 332
 IFEMPTY 144, 332
 IFEQUAL 144, 332
 INCLUDE 16, 145, 299
 POSTARG 144, 149, 150
 PURGE 349, 380
 ROWS 49, 144, 239, 265, 334
 SCRIPT 298, 299
 SEARCHARG 149, 150
 SETPARAM 403, 413

TOTALROWS 264

VAR 150

method

See *also* object and Call Method action

about 347

actions included 402

default and user-created 401

defining and editing 405, 410, 411, 412

getter 373

list of, in Witango class file 401

on_create 401

on_destroy 401

parameter

See parameter, in method

properties 414

renaming 410

return value 406, 412

setter 374

method definition window 400, 405, 406

method scope 403

this, in Witango class file 404

MIN function 255

multi-column list 19

N

naming

action 41, 230

builder 166

nested action

Call Method 404

conditional 282

loop 290

New Record Builder 55, 228

See *also* builder

about 165, 209, 210, 212

adding to application file 212

customizing messages 218

formatting new record entry form 217

generating actions 220

including HTML snippet 222

relationship to Web browser 210

specifying columns 212

new record entry form 209, 210

See *also* New Record Builder

new record response HTML

for New Record Builder 218

no results HTML

associating with an action 46, 49, 236, 239

- creating or editing 49, 239
- Search action 265
- Search page 187
- using meta tag 49, 239
- window 13

O

object

- See *also* method, Create Object Instance action, Call Method action, and workspace, objects

- about 346, 349, 355, 396
- adding to workspace 364, 365
- as black box 346
- attributes folder 373, 384
- benefits of using 354
- caching information 375
- collection object 391, 393
- example of using 351, 352
- expiry URL 382
- general requirements 357
- installing 362
- introspection 347
- method
 - See method
- object instance
 - See object instance
- properties 374
- refreshing 376
- removing from workspace 371
- security 359
- steps in using 364
- thread safety 375
- types supported in Witango 355
- when to use 354

- object instance
 - about 346, 348
 - availability 350
 - creating 349

- object instance variable
 - for Call Method action 386
 - for Create Object Instance action 380
 - for self-referencing 404

- Objects Loop action 227
 - about 391
 - collection object 393
 - item variable 393
 - using 392

- objects.ini 359

ODBC data source

- about 84
- creating 88
- information in application file 114

- on_create method, in Witango class file 401

- on_destroy method, in Witango class file 401

opening

- included files 16

operator 259, 260

- See *also* logical operator and join operator

operator, name of

- and 284

- begins with 259, 260

- contains 259, 260

- ends with 259, 260

- greater than 259, 260, 285

- greater than or equal to 259, 260, 285

- is empty 285

- is equal to 259, 260, 285

- is in 259, 260, 261

- is not empty 285

- is not equal to 259, 260, 285

- is not null 259, 260

- is null 259, 260

- less than 259, 260, 285

- less than or equal to 259, 260, 285

- or 284

option, in using Witango Studio

- See Witango Studio, setting preference

optional parameter 389

or operator 258, 284

Oracle data source

- about 84

- alias 114

- creating 97

- information in application file 114

P

parameter, in method

- about 347, 348

- adding, deleting, and editing 412, 413

- data type 387, 407

- getting and setting value 413

- input and output 373, 387, 406

- list of

- for Witango class file 406
 - in Call Method action window 387
- optional 389
- variable and value 388
- variant 388
- passThroughSwitch 103, 105, 109
- POSTARG meta tag 144, 149, 150
- preference, in using Witango Studio
 - See Witango Studio, setting preference
- Presentation action 227
 - about 53, 243
 - setting up 53, 244
- presentation logic 52, 243
- presentation page
 - about 53, 77, 243
 - adding file 77
 - in Presentation action 53, 244
 - removing file 78
- primary key
 - about 87
 - using to create new record 218
- project
 - See also workspace, project
 - about 65
 - adding
 - file 71
 - folder 70
 - AST signature 79, 82
 - closing 72
 - context-sensitive menu 68
 - creating new 68
 - dependencies
 - See dependencies
 - editing file 73
 - file type supported 71
 - find and replace text 73
 - opening 72
 - path name 67
 - project file 67
 - removing file or folder 72
 - renaming folder 71
- properties
 - action 45, 107, 234
 - application file 75
 - column 112
 - cookie 159
 - data source 102, 110
 - file 77
 - method 414
 - object 374

- project FTP sites 79
 - project root 81
 - snippet 125
 - snippet folder 125
 - table 112
 - window 8
- PURGE meta tag 349, 380
- push
 - associating with an action 46, 51, 236, 241
 - disabled in Witango class file 403

R

- record
 - adding to a table 266
 - deleting 200
 - modifying 268
 - removing 270
 - updating 198
- Record Detail page
 - about 176, 197
 - customizing messages 202
 - formatting Record Detail Web page 201
 - record maintenance 200
 - relationship to Web browser 176
 - specifying columns 197
- record detail Web page 174, 176
 - See also Record Detail page
- Record List page
 - about 175, 189
 - customizing header HTML and footer HTML 196
 - formatting record list Web page 195
 - relationship to Web browser 176
 - specifying columns 189
 - specifying number of matches 193
- record list Web page 173, 176
 - See also Record List page
- referenced object 367
- regular expression 28, 29, 31
 - See also editing
- renaming
 - action 41, 231
 - method, in Witango class file 410
 - snippet 125
- result
 - returned by Insert action 267
 - returned by Update action 269
 - returning to Web browser 51, 241

- sorting 253, 255
- Results action 227
 - See *also* results HTML and no results HTML
 - adding HTML 52, 242
- results HTML
 - associating with an action 46, 47, 236, 237
 - creating or editing 47, 237
 - Direct DBMS action 334
 - in Witango class file 402
 - Search action 265
 - using meta tag 47, 49, 237, 239
 - window 13
- resultSet
 - for Mail action 314
 - in External action 307
 - in File action 319, 321
 - in Script action 300
- retrieving data
 - See Search action
- Return action 228, 295
- ROLLBACK command 326
- row
 - See record
- ROWS meta tag 49, 144, 239, 265, 334
- run-only file, creating
 - application file 59
 - Witango class file 408

S

- scope
 - See *also* variable
 - about 155
 - for JavaScript 298
- scope, name of
 - See *also the names of the specific scopes*
 - instance 403
 - method 403
- Script action 227, 297, 298
 - executing 299
 - setting up 298
- SCRIPT meta tag 298, 299
- Search action 226, 252
 - See *also* column and join
 - creating join 338
 - criteria
 - about 257
 - column 258
 - grouping 255

- include empty 261
 - logical operator 258
 - operator 259
 - quote value 262
 - separator 259
 - value 261
- executing 264
- joining tables 338
- no results HTML 265
- result returning option 263
- results HTML 265
- search type 252
 - normal 253
 - summaries of groups 254
 - summary of all rows 256
- using multiple tables
 - See join
- Search Builder 55, 228
 - See *also* builder, Search page, Record List page, and Record Detail page
 - about 165, 171, 172
 - adding to application file 178
 - creating join 205, 341
 - generating actions 206
 - including HTML snippet 208
 - relationship to Web browser 176
 - using multiple tables
 - See join
 - using simplified steps 204
 - using standard steps 176
- search form 172, 176
 - See *also* Search page
- Search page
 - about 175, 178
 - customizing messages 187
 - formatting search form 186
 - relationship to Web browser 176
 - specifying columns 178
- SEARCHARG meta tag 149, 150
- security
 - in File action 323
 - object 359
- self-referencing, in Witango class file 404
- server, HTTP
 - See Web server
- SETPARAM meta tag 403, 413
- setter method 374
- simple mail transfer protocol
 - See SMTP
- SMTP 309
- SMTP server 314

- snippet
 - about 117, 118
 - copying, deleting, duplicating, and moving 126
 - creating
 - folder 124
 - snippet 121
 - editable and non-editable 121
 - editing 121, 123
 - for New Record Builder 222
 - for Search Builder 208
 - inserting 120
 - organizing 124
 - placeholder or yen symbol 123
 - properties 125
 - renaming 125
 - workspace 118
 - snippet, name of folder
 - builder snippets 119, 121
 - column snippets 119, 121, 126, 128
 - configuration variables 119, 121, 126
 - my snippets 119, 121, 124, 126
 - standard snippets 119, 121, 126
 - SQL
 - calling stored procedure 335
 - SQL keyword
 - See *also* operator and function
 - COMMIT 326
 - DISTINCT 264
 - HAVING 255
 - ROLLBACK 326
 - SQL query
 - executing 330
 - performing 26
 - setting up 23
 - using meta tag 332
 - window 22
 - string, find and replace 28
 - studio, Witango
 - See Witango Studio
 - SUM function 255
- ## T
- tab, in editing text
 - See editing
 - table
 - See *also* join
 - adding records to 266
 - filtering 114
 - modifying record in 268
 - page format used in builder 167
 - properties 112
 - removing record from 270
 - selecting from data source 113
 - setting primary key 87
 - Tango Editor
 - interface 6
 - TCF
 - See Witango class file
 - TCFSearchPath 363
 - text file
 - font, size, and color option 133
 - text, editing
 - See editing
 - TOTALROWS meta tag 264
 - transaction action 325
 - See *also* Begin Transaction action, End Transaction action, and database action
 - commit and rollback 328
 - executing 329
 - impact on database connection 326
- ## U
- Update action 226, 268
 - executing 269
 - results returned by 269
 - setting up 268
 - update response HTML 202
 - user-created method, in Witango class file 401
- ## V
- value
 - of operator 260
 - search criteria 261
 - VAR meta tag 150
 - variable
 - See *also* scope, array, and configuration variable
 - about 155
 - adding variable assignment 157
 - assigning 156
 - deleting variable assignment 157
 - editing assignment 156
 - moving variable assignment 157
 - selecting variable assignment 157

W

While Loop action 227, 289, 290
 executing 293
 pitfalls to avoid 291

window

 application file 56
 error HTML 13
 HTML editing 9
 method definition 400, 406
 no results HTML 13
 properties 8
 results HTML 13
 SQL query 22
 Witango class file 400

Windows menu shortcut 34

Witango

 about 1

Witango builder

 See builder

Witango application file

 See application file

Witango application server

 See Witango Server

Witango class file 55

 See *also* application file, method, and parameter,
 in method

 about 348, 356, 396

 adding to workspace 369

 Assign action 403

 benefits of using 397

 Branch action 402

 creating 408

 debugging 415

 developing 400

 differences from application file 402

 editing 409

 error handling 403

 executing 363, 402

 list of actions 402

 list of instance variables 404

 push attribute disabled 403

 recursion 403

 results HTML 402

 scope 403

 setting search path 137, 363, 416

 steps in using 399

 when to use 398

 window 400

Witango component

 See *also* Witango Studio, Witango Server,
 Witango CGI, and Witango plug-in

Witango Server

 about 1

Witango Studio

 about 1

 setting preference 129, 130

 builder page format 131

 on-line help 132

 selecting table from data source 132

 selecting text editor 131

 text font. size, and color 133

word wrap 12

workspace

 about 6

 data sources 85, 110

 objects

 adding 364, 365

 COM object 365

 example of 372

 JavaBean 367

 removing 371

 viewing information 372

 Witango class file 369

 project 66

 moving file or folder 68

 opening file 67

 presentation pages 77

 snippets 118

X

XML

 See *also* Document Object Model and
 document instance

 about 55

 DTD 56

 folder 56

 format 55

 advantages 55