



Witango 5 Tutorial

OS X

July 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



I Learning Witango I

Things You Need to Know and Do Before You Begin

Terminology	1
Overview of Lessons	3
Getting Started	5
About Your Web Server Document Root	5
Saving Witango Application Files	7
Executing Witango Application Files	7
Tutorial Samples and Resources	8
Other Resources	8
How To Use Witango: A Summary	8
Witango at a Glance	11
You're Hired!	12

2 Application File Execution Flow, Control Actions, and Passing Values 13

Introducing the Dynamic Web Site; Post Arguments and Search Arguments on the Web and in Witango

Generating a Web Page With Witango	14
Adding the Welcome Page for Suppliers	18
Adding Control Actions to Show Only One Page at a Time	21
Adding a Menu Page	29
Every Good Witango URL Has...	36
Passing Values Using HTML Forms	45

3 Data Sources 53

Creating a Data Source to Use for the Music Store

Creating a Data Source	54
Displaying Sale Items From the Database	58

4	Witango Builders and Projects	69
	<i>Working With the Search Builder, the New Record Builder, and Witango Projects</i>	
	Search Builder for Customer Page	70
	Setting the Number of Matching Records Returned	81
	Inserting Records Into the Database	85
	Creating a Pop-up Menu & Setting Required Fields	92
	Creating a Project	97
5	Collecting Values and Assigning Variables	101
	<i>Collecting, Storing, and Tracking User Information</i>	
	<i>Using an Assign Action</i>	
	Creating a Personalized Welcome Page	102
	Storing and Tracking User Information	108
6	Enhancing Your Web Site	115
	<i>Creating Additional Enhancements for the Music Web Site</i>	
	Displaying the Current Time and Date	116
	E-mailing Sales Items to Customer	119
	Using the <@INCLUDE> Meta Tag to Enhance the Menu Page	131
7	Login Model	135
	<i>Creating a Login Model for Music Club Members</i>	
	Using Search Builder to Create a Framework for the	
	 Login Application File	136
	Setting the Business Logic of the Login Solution	145
	Storing and Tracking User Information	153
	Preventing Unauthorized Access to Application Files in the	
	 Options Menu	157
8	Witango Class Files	161
	<i>Creating a Witango Class File and Using it in Tango Application Files</i>	
	Creating a Witango Class File	162
	Calling the Login Method in the Witango File	171
9	What's Next	179
	<i>Concluding Remarks</i>	

Learning Witango

Things You Need to Know and Do Before You Begin

Welcome to the Witango Tutorials! All tutorials in this book follow a common theme: the creation of functional web applications for an online music store. Because the lessons are based on typical, realistic applications, working through them will help you to understand the key functions of Witango Studio and how they apply in a web application.

When you have completed the tutorials, you should have a good grasp of the basics required to create web applications with Witango. These tutorials are designed for users who are familiar with the following:

- the basics of HTML—there are many excellent tutorials on HTML available on the World Wide Web and in print. One of the best sources for HTML references is the World Wide Web Consortium web site at <http://www.w3.org/>.
- the basics of web server operation—please consult your web server documentation.

Terminology

Some of the common terms you will encounter in these tutorials are briefly described here:

- **Witango Studio** is the development environment of Witango, featuring a complete graphical user interface in which to develop application files suitable for execution by Witango Server.
- **Witango Server** is an application server that works in conjunction with a web server (or more specifically, an HTTP server) to process Witango application files.
- **Witango CGI** and **Witango plug-in** each communicate a user's request from the web server to Witango Server, and return the results to the web server. The Witango CGI runs as a separate program on your web server. The Witango plug-in, as the name implies, plugs directly into your web server. You may use either; both the Witango CGI and Witango plug-in perform the same function.
- You create **Witango application files** (also called **taf files**) with Witango Studio. Application files are like programs or scripts that determine what operations Witango Server performs. These dynamic applications run on your web server and interact with

databases, other applications, objects, and users accessing the Witango application using a web browser.

- Witango **builders** help you create and generate a sequence of actions in an application file to perform specific tasks. Witango Studio provides two builders. The **Search Builder** builds the actions required to search database records, as well as to update and delete them. The **New Record Builder** builds the actions required to add a new record to a database.
- A **data source** contains all the information needed to connect to a particular database. You use data sources to tell your Witango applications what databases to connect to.

Both Witango Studio and Witango Server need to have access to data sources. Witango Studio uses a data source to show you database information in the form of database layouts and their fields. Witango Server requires the same data source to access the layouts and fields specified within the application file.

- An **IP address** (Internet Protocol Number) is a unique number consisting of four parts separated by dots, such as 207.105.84.106. Every machine on the Internet has a unique IP address. Most machines also have one or more *domain names*, which are easier for people to remember. These are more generally used and are part of the URL.
- **URL** (Uniform Resource Locator) is a standard way to give the address of any resource on the Internet, most commonly used for the web. Here is an example of a URL:

`http://www.witango.com/`

The most common way to use an *http*-type URL is to enter it into a web browser program, such as *Navigator* or *Microsoft Internet Explorer*.

Overview of Lessons

This book is divided into seven tutorials. Key terms are defined as they are introduced in the lessons. Many key terms are also defined in the glossary on page 209 of this book.

Tutorial A

The lessons in this tutorial introduce you to the basics of developing a Witango application file. The concepts you learn include creating a dynamic web page, passing values within an application file, and setting up a database search.

Key terms: Witango application file, Results action, meta tag, value, variable, If action, Else If action, Else action, search argument, post argument, snippet, user key, cookie, state, form method.

Tutorial B

The lessons in this tutorial show you how to create a data source. You need to create this particular data source to complete subsequent tutorials.

Key terms: ODBC data source, Search action, Results HTML, column snippet, <@ROWS> meta tag, resultSet, Data Sources Workspace.

Tutorial C

The lessons in this tutorial teach you about Witango builders and projects. You work with the Search Builder and New Record Builder to create Witango applications in which you insert new records, search for them, and display results. You also learn how to organize your Witango application files into projects.

Key terms: action-based metaphor, Search Builder, drilling down, building actions, New Record Builder, column options, field type, error codes in Witango Server, Project Workspace.

Tutorial D

The lessons in this tutorial teach you about personalization and saving user information. You learn how to create form fields for collecting user information. Then, you assign this information to variables so that it can be stored and referenced over multiple web pages.

Key terms: Assign action, user scope, configuration variable.

Tutorial E

The lessons in this tutorial show you how to make functional and stylistic enhancements to your application.

Key terms: Mail action, Simple Mail Transport Protocol, <@CURRENTTIMESTAMP> and <@INCLUDE> meta tags.

Tutorial F

The lessons in this tutorial teach you how to create a secure login model for users of your web site.

Key terms: login model, Form Results action, RecordList action.

Tutorial G

The lessons in this tutorial show you how to create a Witango class file, which can be used in a number of different Witango application files. You

then learn how to call a Witango class file's methods into your login model to search the record list.

Key terms: Witango class file, object, method, parameters, Parameter List, method call, instance, Create Object Instance action, Call Method action, <@GETPARAM> meta tag, <@SETPARAM> meta tag, Objects Workspace.

Getting Started

Before you begin the tutorials, install Witango Development Studio on a computer that has a web server installed or on a computer that has a network link to a web server. If you do not have a web server on your machine, you can install Apache web server, which is a free download from <http://httpd.apache.org>

About Your Web Server

Both Witango Server and your web server should be installed and running correctly on your system once you have completed the Witango installation.

You save the files you create in the Tutorial folder within the Witango folder in your web server document root; that is, the folder from which the web server serves pages up to web browsers that request them.

Before you begin the tutorials, there are some locations that you should identify. The tutorials refer to these locations in general terms, so you must be aware of them as you proceed through the lessons. These locations are:

- Your web server, and the location of your web server document root
- Where to save your Witango application files.

Your Web Server Document Root

Files to be served on the web must be located in your *web server document root* folder. When you call the IP address or domain name of a web server in the URL line of a web browser, the web server looks for the files in the web server document root folder or its sub-folders.

When you install Witango, a Witango folder is created within the web server document root that you selected during installation. You should be familiar with the location of the web server document root folder because you save the application files you create in Witango Tutorials in the Tutorial folder within the Witango folder of the web server document root.

The following are the default web server document root folders for some common web servers:

Web server	The default web server document root folder is...
Apache Server	<code>\Library\WebServer\Documents</code>



Note If you installed these web servers in somewhere other than their default location, you must use the appropriate path.

Saving Witango Application Files

These tutorials require you to save the application files you create within the `Tutorial` folder within the `Witango` folder of the web server document root.

To save an application file for the first time

- 1 In Witango Studio, choose **Save As...** from the **File** menu.
- 2 Enter the appropriate file name (with the `.taf` extension). For more information, see “Your Web Server Document Root” on page 4.
- 3 In the **Save in** field, navigate to the web server document root folder appropriate to your web server, or type in the path in the **File name** field.

Executing Witango Application Files

To execute an application file

- 1 Save the Witango application file on which you have been working. Remember the file name, as you will need to enter it in the URL.
- 2 Open your web browser.
- 3 Do one of the following:
 - Type in the URL to the specific file you wish to execute, and press ENTER.
 - If the appropriate URL already exists from a previous execution (since, in the lessons, you will be toggling back and forth

between Witango Studio and your web browser), simply reload or refresh your file to execute the latest saved version of it.



Note Both Witango Server and your web server must be running in order to execute application files and send the information back to your web browser. Make sure both of these applications are running.

“What is the URL for my application file?”

The information that you type in the URL line of your web browser is slightly different depending on whether you are using the Witango CGI or plug-in.

To execute an application file, use the URL that corresponds to the Witango Server you are running on you machine (replacing each italicized part below with the information it specifies):

If you are using...	The URL is...
Witango plug-in (for Apache, Microsoft servers)	<code>http://yourwebserver/path to application file/yourfile.taf</code>
Witango CGI	<code>http://yourwebserver/Witango-bin/ wsgi.exe/path to application file/ yourfile.taf</code>



Note Witango-bin is an alias to your cgi-bin directory that is created by Witango Installer during installation of Witango.

On Windows, you can use localhost or 127.0.0.1 instead of the name or IP address of yourwebserver for certain web servers. These special names are a signal to your web server to access the web server on the current machine.

Witango Tutorial Samples and Resources

Completed examples of the application files you create in these tutorials are provided in subfolders of the Tutorial folder within the Witango folder of the web server document root. Use these to help you complete the tutorials.

Resource files that you need as you work through the lessons are also provided. For the names and locations of these files, refer to the `ReadMe.txt` file in the Tutorial folder within the Witango folder of your web server document root.

Database for Tutorial

For these tutorials, you will require access to an SQL database. In the `Resources/Database/Music` folder provided with this tutorial, you will find a script which can be run with OpenBase SQL (which is available for download from <http://www.openbase.com>) to define our sample **MUSIC** database together with sample data.

You may use any database which you can connect to through ODBC or JDBC. CSV files containing sample data has been provides in the `Resources/Database/Music` folder provided with this tutorial. For the Tutorials you will need a database called MUSIC with the following structure:

PRODUCTS TABLE	
product_key	int(11) unsigned NOT NULL auto_increment (primary key)
title	varchar(40) default NULL
price	decimal(10,0) default NULL
sale_price	decimal(10,0) default NULL
onsale	enum('True','False') default NULL
type	varchar(10) default NULL
category	varchar(15) default NULL
supplier	varchar(30) default NULL
artist	varchar(35) default NULL

SITEUSERS TABLE	
siteusers_key	smallint(6) NOT NULL auto_increment (primary key)
login_id	varchar(20) default NULL
passwd	varchar(20) default NULL
ulevel	smallint(6) default NULL
firstname	varchar(20) default NULL
lastname	varchar(20) default NULL
company	varchar(25) default NULL
address	varchar(30) default NULL

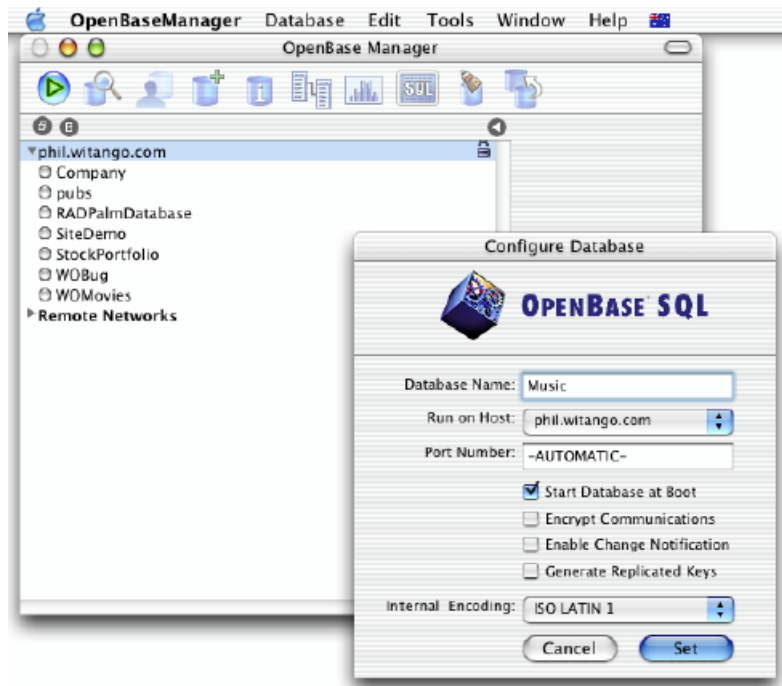
SITEUSERS TABLE	
city	varchar(25) default NULL
state_prov	char(3) default NULL
country	varchar(20) default NULL
zip_post_code	varchar(10) default NULL
email	varchar(45) default NULL
groupname	varchar(15) default NULL

Creating and OpenBase Database

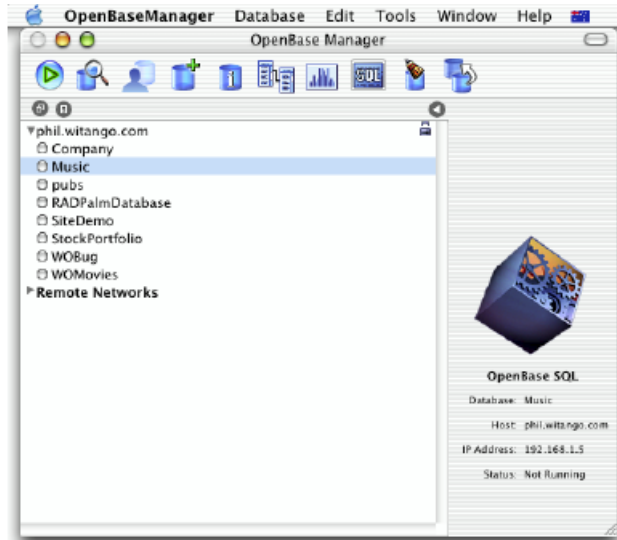
- 1 Open the **OpenBase Manager** application, and click on the create database icon in the toolbar.



- 2 This will open the **Configure Database** Window. Name the database `Music` and press **Set**.



- 3 Now the `Music` database is created it will now appear in the list of databases as shown below. Highlight the `Music` database by single-clicking on it.



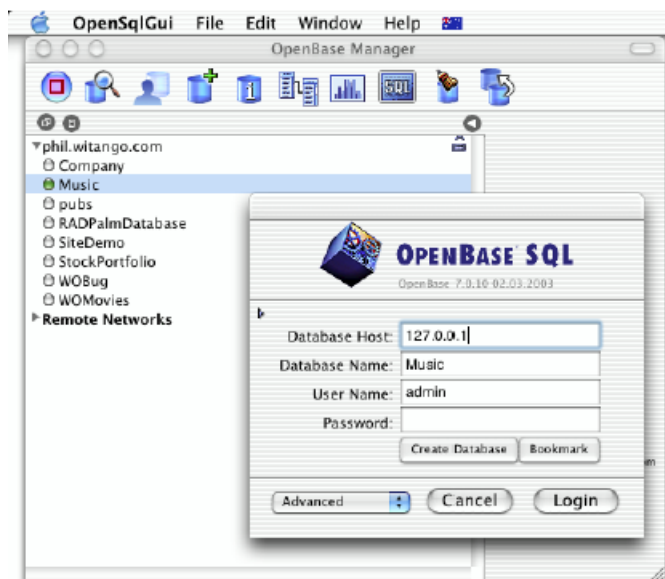
- 4 Click on the **Start Database** Icon (shown below), this will make the database available to you for creation of tables and loading of data.



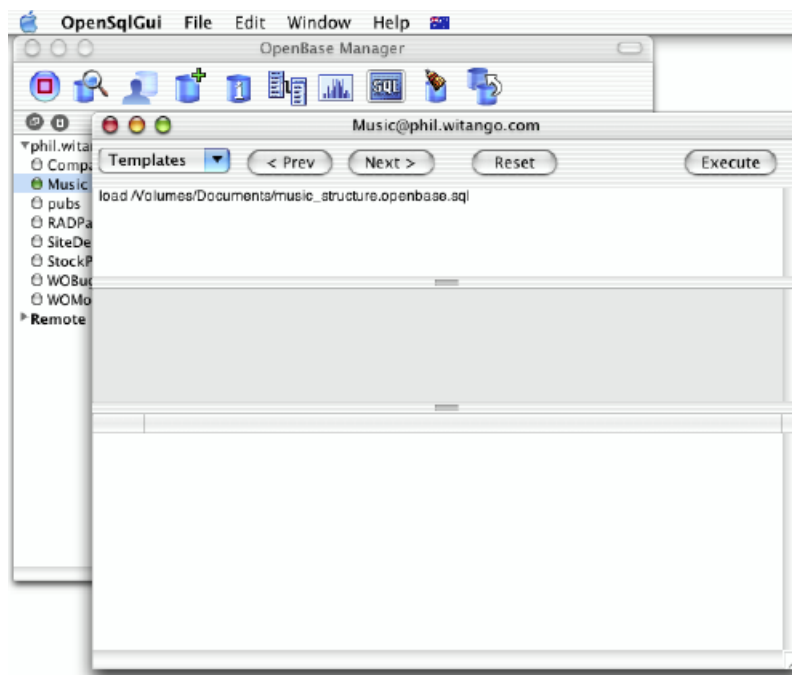
- 5 Once the `Music` database is started, click on the **SQL** icon to bring up the SQL query window to allow the load of data structure.



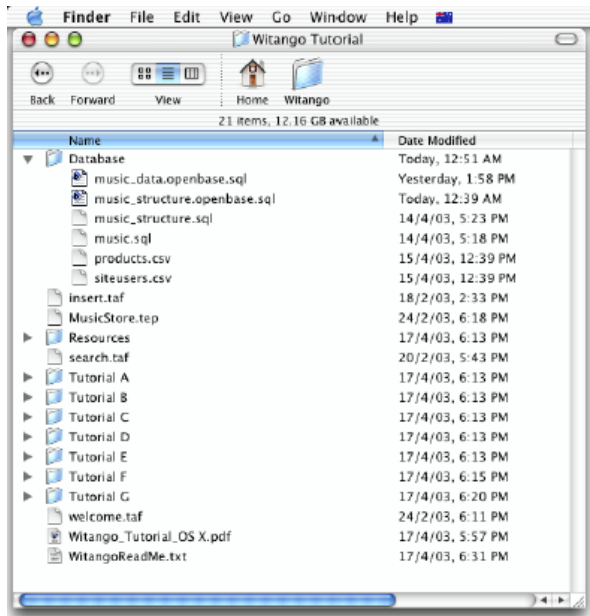
- 6 You will be required to login to the database. Login with the user details you have set up.



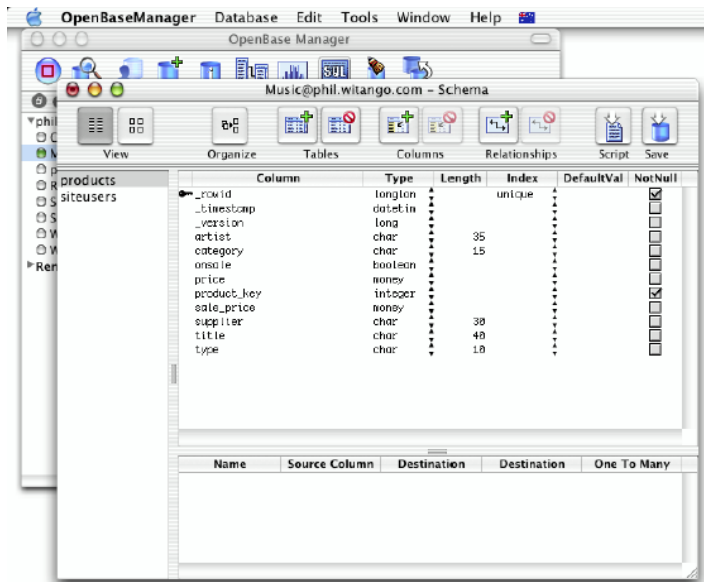
7 An interactive SQL window will appear,



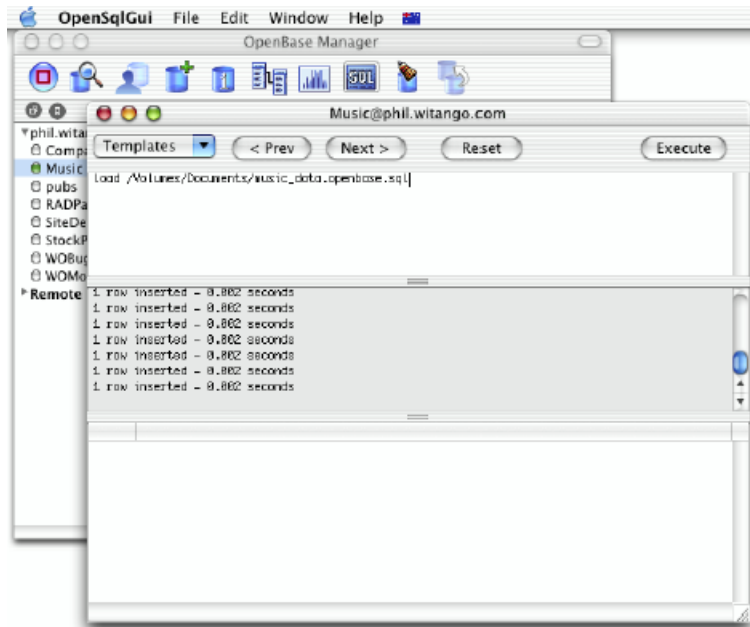
- 8 Use the **LOAD** command to load music_structure.openbase.sql file (which is available in the database directory of the Witango Tutorial Resources).



- 9 The datastructure will now be available.



- 10 Use the **LOAD** command to load `music_data.openbase.sql` file (which is available in the database directory of the Witango Tutorial Resources).



- 11 The database should now be defined and contain the sample data as provided.

Other

Ensure that you have a web browser installed on your machine. How To Use Witango: An Overview

"How do I start using Witango after installation?"

Create Witango application files in Witango Studio.

- 1 Save them in a folder within the web server document root (in these tutorials, save your files in the `Tutorial` folder within the Witango folder in your web server document root).
- 2 Call your application files in your web browser using the appropriate URL.

Both Witango Server and your web server must be running in order to execute application files and send the information back to your web browser. Make sure both of these applications are running.

Witango application files contain a series of instructions for the Witango Server to carry out when the file is called in the web browser. These instructions take the form of actions in the application file.

"How do Witango application files differ from typical HTML files?"

One HTML file produces a single web page. However, one Witango application file may produce multiple web pages. A Witango application file is, in effect, a program, and, as such, it can be set up to produce different HTML end results. For example, the Search Builder can be used to create a file that contains the form page, the record list page, the record detail page, the update page, and the delete page—all in one file. Therefore, when you are learning Witango, it is important to understand how Witango Server navigates through the different sections of the file. The first few lessons in Tutorial A teach you the flow of a Witango application file.

"I want to use Witango to search my database. What do I do?"

Use Witango's Search Builder action to build an application file with a search form (which allows the user to specify the criteria on which they will search), a record list page (a summary list of results which match your search criteria), and a record detail page (a page which links to full detail of a particular record from your summary list of results). This application file also has the ability to update and delete records when a user has navigated through to the detail page. Witango builders provide a web page-based interface for building actions in an application file. To learn more, please complete Tutorial C.

"I want to use Witango to enter information into my database. What do I do?"

Use Witango's New Record Builder action to build a file that creates an insert form—perhaps with required fields—to enter new records in the database. To learn more, please complete Tutorial D.

"I want to use Witango to do other things with my database. What do I do?"

You may also build your database functionality from scratch, without the aid of the builders. Drag one or more of Witango's five database-related actions into an application file and set them appropriately: Search, Insert, Update, Delete, or Direct DBMS (which allows you to type in your own SQL statements). To set these actions, drag columns in from the Data Sources Workspace.

"What else do I need to know to use Witango effectively?"

With Witango, you build dynamic, data-driven web sites that interact with users and present them with “live” data from your database.

Therefore, you need to understand the following:

- how to collect information from the user, and how to reference that information with Witango meta tags
- how to retrieve information from a database, and how to reference that information with Witango meta tags
- how to assign user and database values to variables in Witango Server's memory, and how to reference the variables with Witango meta tags.

Witango at a Glance

This diagram outlines how Witango components work together to create and execute applications.

Browser/Client

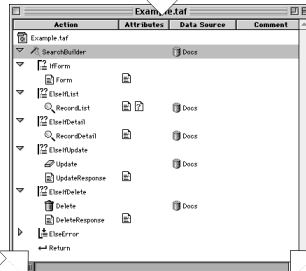
End user values are passed from one HTML page to another in an HTML form (button) or by being appended to a hyperlink (anchor reference).

A form field value, or **post argument**, is passed in an HTML form with METHOD=POST.

A URL argument, or **search argument**, is passed in a URL.



Witango
Application
File

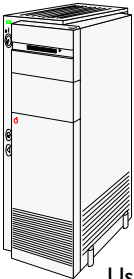


To reference post and search arguments in Witango, use:

<@SEARCHARG argument_name> for search arguments only.

<@POSTARG argument_name> for post arguments only.

<@ARG argument_name> for both search and post arguments.



Witango Server Variables

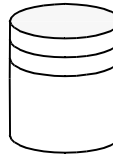
End-user values and results from the database only last for one execution of an application file or web page. To keep values for as long as you want, assign them to Witango variables.

Use <@DEFINE> to define the variable type.



Use <@ASSIGN> or the Assign action to assign values.

Use <@VAR NAME=variable_name> to reference variables.



Database

To capture values from the database, use:

<@COLUMN table_name.column_name> in the Results HTML attribute of the

database action producing the results, or <@COL column_number> anywhere in an application file.

<@VAR NAME=resultSet[x,y]> anywhere after a database action to return an array with all the database results.

You're Hired!

Here is the scenario: you have just landed a new job in which you will be using Witango Studio to set up a web site for an online music CD retailer. Your new boss needs to have the site up and running as soon as humanly possible, so he wants you to begin creating Witango application files right away.



This may seem like quite a challenge, but you are not alone. Along the way you will be prompted and advised by questions and answers relating to your current task. As well, you will be given the *ABCs*—expert advice on how Witango operates. Look for the building block icon.

It is time to begin. Good luck!

Witango Studio Basics

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

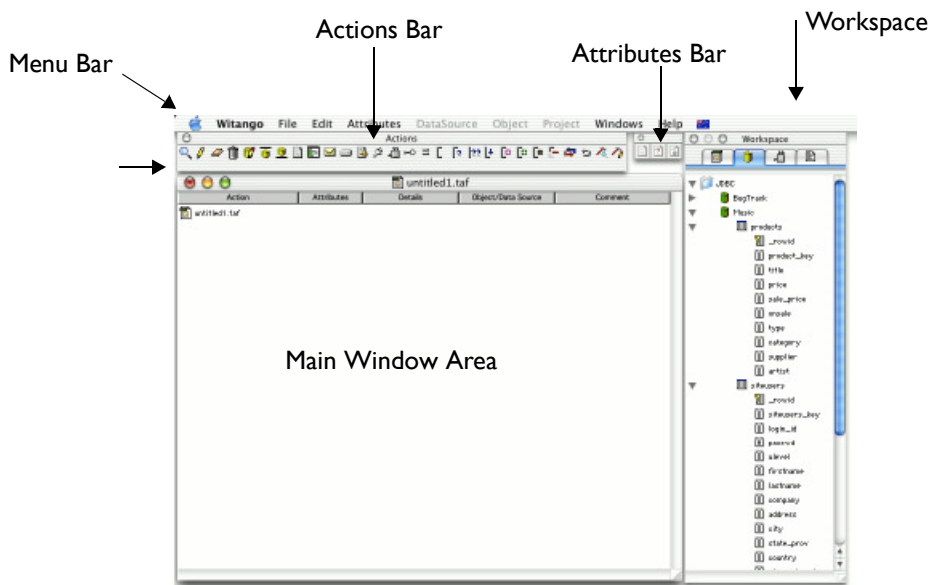


This chapter helps you orient yourself to the Witango Studio interface.

Witango Studio Window Components

To start Witango Studio double-click the Witango Dev Studio 5.0 icon which can be found in the Applications/Witango/Dev Studio 5.0 folder.

The main Witango Studio window appears:



- 3 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.
- 4 The **main window area** displays one or more Witango application file windows, action editing windows, attribute editing windows, or the SQL Query window.

- 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 Click icons on the **Actions bar** and drag them into an open Witango application file to add them to the file.
- 7 Click icons on the **Attributes bar** to assign attributes to selected actions.

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 dealing with Business Logic

The Business Logic Actions control how the application flow. They are listed in the table below:

Icon	Action	Function
	Assign	Makes specified value assignments.
	Group	Groups related actions.
	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.

Icon	Action	Function
	Return	Ends execution of an application file and returns the accumulated Results HTML to the browser.

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:

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.

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:

Icon	Action	Function
	Results	Performs no special function of its own, this action allows HTML to be appended to the Results HTML.

Icon	Action	Function
	Presentation	Allows user to reference presentation pages (external files).

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 Witango Server machine.
	Script	Allows you to specify server side JavaScript code to execute.
	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

Application *FileExecutionFlow, Control Actions, and Passing Values*

*Introducing the Dynamic Web Site; Post Arguments and Search
Arguments on the Web and in Witango*

There are profound differences between static web sites and Witango web sites.

- A static web site is *fixed* in what it can show. It is most commonly a series of pages linked together by HTML hyperlinks that must be manually updated whenever there is new information.
- A Witango web site is dynamic because it is *driven* by the data in your database; that is, it can access your data and bring it to the web. This also means that a Witango web site can display any number of pages—it simply depends on what information it is *asked* to show.

This tutorial introduces you to the dynamic web site as created with Witango and the very important concept of *passing values*. Passing values is an integral Witango concept as passed values, such as user inputs, are used to generate dynamic web pages.

There are six lessons in this tutorial:

A-1: Generating a web page with Witango

A-2: Adding the Welcome Page for Suppliers

A-3: Adding control actions to show only one page at a time

A-4: Adding a Menu Page

A-5: Every Good Witango URL Has...

A-6: Passing Values Using HTML Forms

LESSON A-1

Generating a web page with Witango

Purpose

To create an application file with Witango Studio, using a *Results* action.

Context

In web sites built with Witango, the user does not call and display static HTML files; they execute *Witango application files*. An application file is executed simply by specifying the name of the file in the URL, just like you would an HTML page. The difference is that the static HTML file displays only one web page, while a single application file can execute any number and variety of web pages, depending upon user input or database results.

Exercise



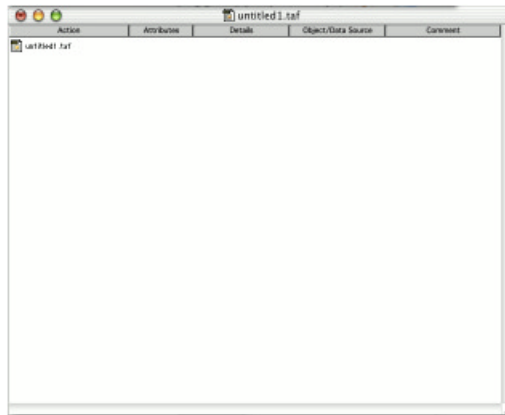
The Boss



The Boss leads you into a closet-sized office and points you to your new desk. He scoops up the nameplate of your predecessor and tosses it into the trash can. Before you have a chance to express your gratitude, he says gruffly, “Let’s see what you can do. Start by building a web page that produces a message to welcome our customers.”

Well, let’s get started!

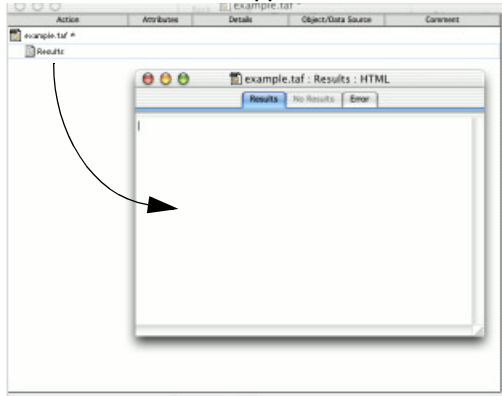
- 1 Click the **New Witango Application File** button (located at the left end of the toolbar), or choose **New**, then choose **Witango Application File** from the **File** menu. A new, untitled Witango application file window appears:



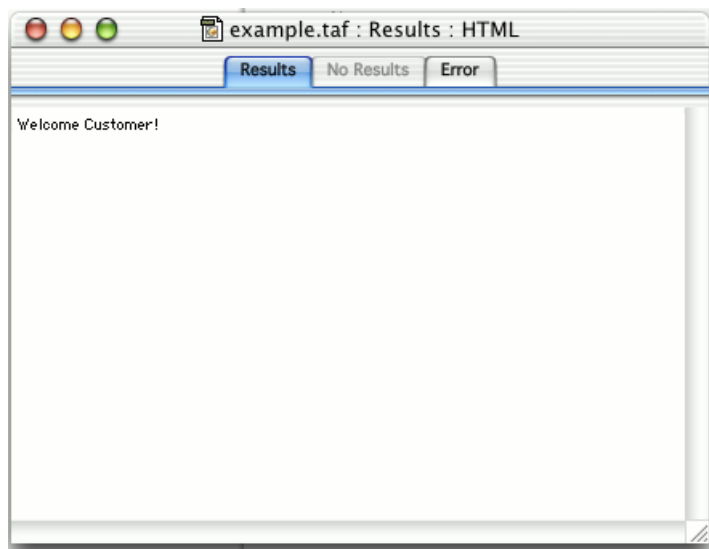
Results action

- 2 Drag a **Results** action into the application file window to begin building the Customer Page.

A new Results HTML window appears:

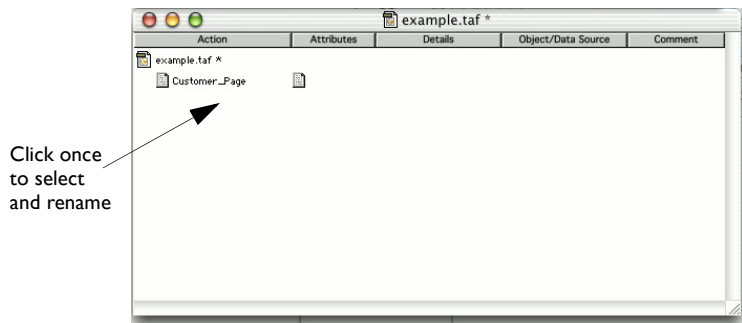


3 Type `Welcome Customer!` in the Results HTML window.:



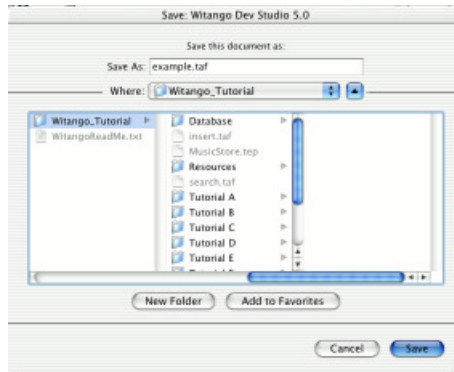
4 Close the Results action window. Select the Results action by clicking on it. A cursor appears, allowing you to type a new name. Rename

the Results action to **Customer_Page**, as shown.



For details, see “Saving Witango Application Files” on page 5.

- 5 From the **File** menu, choose **Save As....** Save the file as `welcome.taf` in the `Witango_Tutorial` folder within your web server document root.



For details, see “Executing Witango Application Files” on page 5.

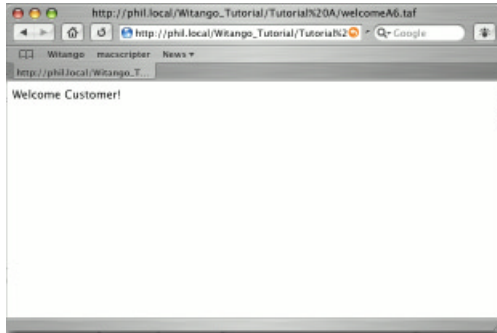
- 6 Open your web browser and execute `welcome.taf`.

Remember to include the path to your Witango CGI in the URL line if you are not using the Witango plug-in.



Note Both Witango Server and your web server must be running in order to execute application files and send the information back to your web browser.

Your welcome message to your customers appears in the web browser:



You have just created your first web page in Witango!

Witango Server executed `welcome.taf`, starting from the top because **execution always flows from the top down**. Witango came to the Results action (which holds HTML) and collected the HTML found there: `Welcome Customer!`. Having run out of actions after the Results action, Witango stopped executing the file and sent all collected HTML back to the web browser. The result is the “Welcome Customer!” message that appeared in the web browser.

LESSON A-2

Adding the Welcome Page for Suppliers

Purpose

To include a welcome message to suppliers using a second Results action.

Context

In typical web sites, one HTML file constitutes one web page. In Witango, however, one application file may contain more than one web page; application files may have more than one result, depending on what the user visiting your Witango web site asks to see. To demonstrate this, you will add the Supplier Page to your existing `welcome.taf` application file. Once you have done this, you will determine a way to show one page or the other.

Exercise



The Boss

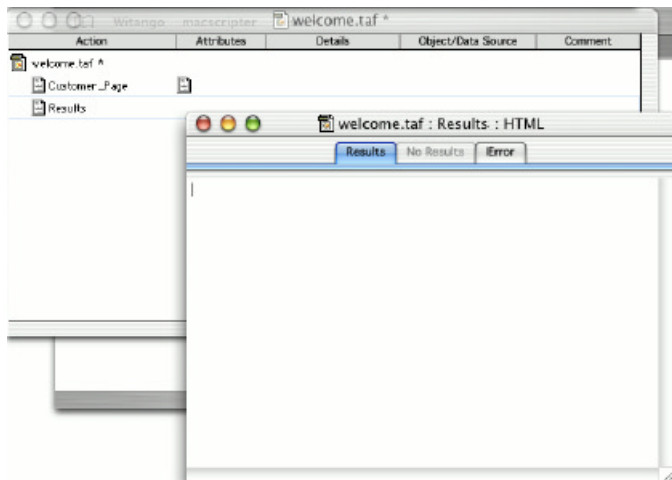
The Boss says, “I see the welcome message for the customer, but what about the welcome message for the supplier? Don’t you realize that we’d be out of business without our suppliers?!”

You learned long ago that sometimes the best response is no response, so you nod your head and get to work.

- I Click the Open icon on the Studio toolbar and return to `welcome.taf` in Witango Studio, and drag in a second **Results** action, placing it below **Customer_Page**.

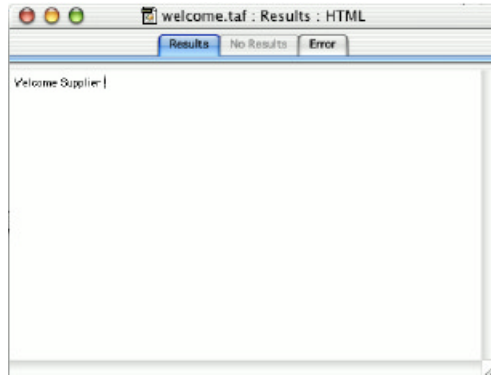


Results action

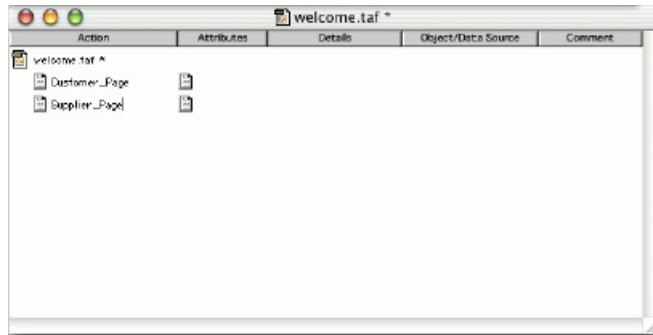


- 2 Type `Welcome Supplier!` as the message in the new Results action

window.

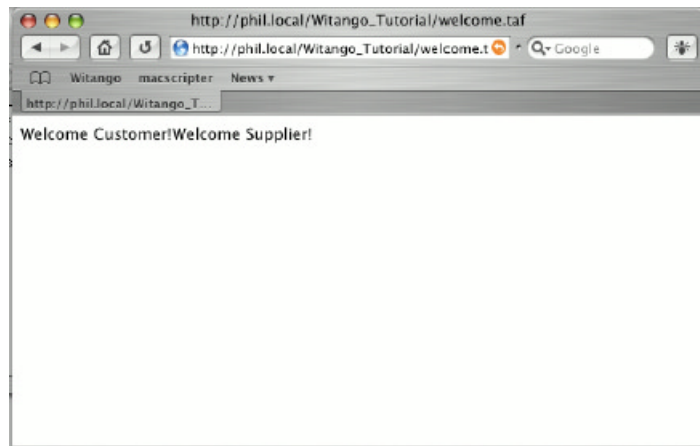


- 3 Close the second Results action window. Rename it **Supplier_Page** in the application file window



For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 4 Save welcome.taf (in the Tutorial folder), return to the web browser, and reload or refresh welcome.taf.





You may have noticed that when Witango executed the `welcome.taf`, it appended the HTML of both actions and displayed them on one page.

This happens because application files are executed in order from the top down unless control actions alter this path. In your current application file, Witango executed your first action, `Customer_Page`, then dropped down and executed `Supplier_Page`. When Witango ran out of actions, it stopped execution of the file and sent the collected HTML from both actions to the web browser. The result is that both messages are displayed. This is an important point. **Witango concatenates, or joins together, the HTML it collects across multiple actions during a single execution of an application file.**

LESSON A-3

Adding control actions to show only one page at a time

Purpose

To display the message to either the customer or the supplier, based on a condition or expression.

Context

In order to display the message to either the customer or the supplier, a condition or expression must be used and evaluated. The *If action* evaluates expressions; if the expression resolves to true, the items grouped within the *If action* are executed by Witango Server.

In the customer/supplier scenario, you are checking for the *value* for a *variable* named `visitor`:

- If `visitor=customer`, execute the **Customer_Page** action.
- If `visitor=supplier`, execute the **Supplier_Page** action.

The user sends the value for `visitor` by typing it in the URL and sending it as a *search argument* to Witango Server. A search argument, which includes the variable and its value, is passed in the URL line of the web browser, and then to Witango Server.

Values passed from the web browser typically take the form of name-value pairs. The name refers to the name of the argument being passed, which in this case is `visitor`. The value refers to the value assigned to the argument, which in this case is `customer`.

Therefore, the name-value pair is `visitor=customer`.

In programming terms, `visitor` is like the variable name, and `customer` is the value being assigned to that variable. `Visitor` is the container, if you will, that holds the value, `customer`. Once the argument is passed, simply making a reference to the variable—or argument name—can reference its value.

The name-value pair methodology allows a web developer to reference end-user values without having to worry about what the actual values are.

Exercise



The Boss

The Boss seems satisfied, but he grumbles something that sounds suspiciously like, “Beginner’s luck!” Coming from him, you consider it a compliment.

Now he wants you to modify your application file so that it can display two separate pages, one to welcome the customer, and another to welcome the supplier.

You need to set up an application file that can determine whether the visitor is a customer or a supplier. Therefore, Witango must check for a *condition*.

Q. What Witango action do you drag in to the application file to allow for a condition to be checked?

A. An *If* action.

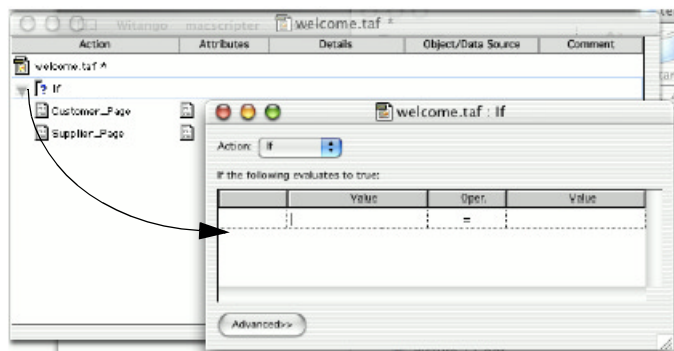
I Return to `welcome.taf` in Witango Studio and, from the **Actions** toolbar, drag in an **If** action, placing it just above **Customer_Page**.

 If action



Tip Once you have dragged in an action, you can align it vertically using the horizontal gray bar that appears when you click and move the action.

The If action window appears:



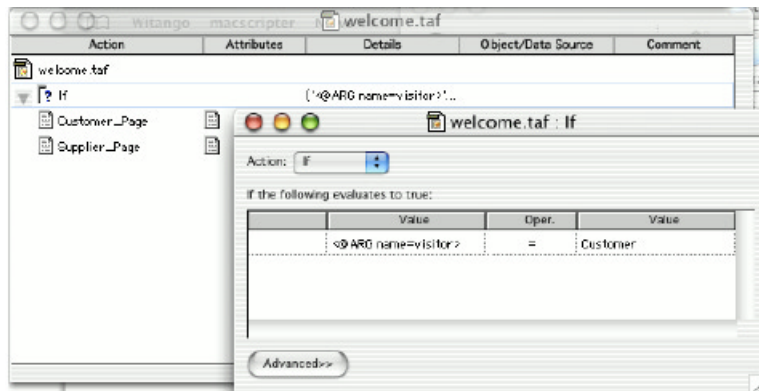
When Witango Server encounters an If action in the application file, it evaluates the condition set within it. If the condition is true, Witango Server proceeds to execute the indented list of actions within the If action. In the If action window, the drop-down menu allows you to choose between different conditions: *If*, *Elseif*, or *Else*. In this case, the default action is set at *If*.

- 2 Leave the **Action** set at **If**, and in the first **Value** field, type `<@ARG NAME=visitor>`.

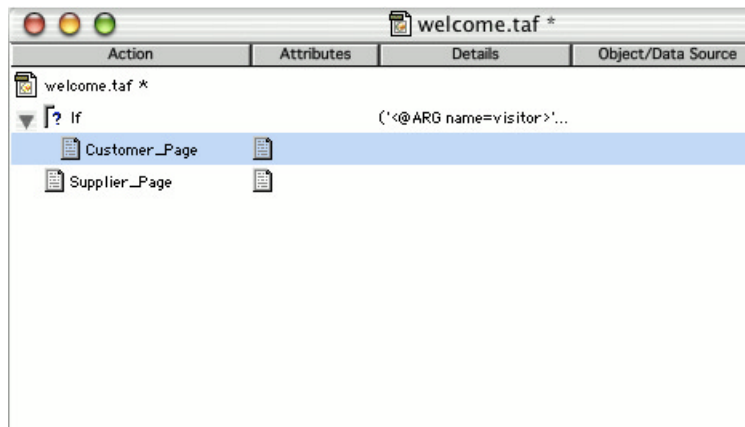


Note The window that appears for a new If action automatically displays a new value row with the first Value field ready for entering a value.

- 3 In the **Oper.** (operator) field, make sure `=` is selected.
- 4 In the **Value** field, type `customer`.



- 5 Close the If action window.
- 6 In the application file window, drag the **Customer_Page** action into the **If** action. **Customer_Page** becomes an indented action within the **If** action, as shown here:





By setting the If action with these values, you are setting the condition that *if* the search argument named `visitor` equals `customer` *then* execute the indented `Customer_Page` Results action. You include the search argument in the URL after the name of the application file.

7 Rename the If action to **If_Visitor_Is_Customer**.



- Q.** If you save `welcome.taf` as it is and execute it in the web browser without saving a value for `visitor`, what HTML message appears?
- A.** The result of the execution would be `Welcome Supplier!` from the `Supplier_Page` action, because the condition in the If action would not resolve to true during execution.

However, if `visitor=customer` was sent to Witango Server during execution, both messages would appear. `Welcome Customer!` would appear because the condition in the If action would resolve to true, and `Welcome Supplier!` would appear because `Supplier_Page` would be the next action in the order of execution.

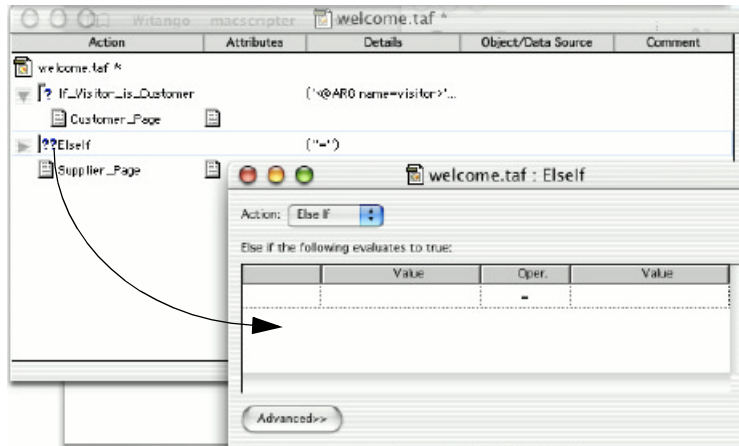
You need to set a condition so that “Welcome Supplier!” appears only if `visitor=supplier`.



Elseif action

8 Drag in an **Elseif** action, placing it just above **Supplier_Page**.

The Elself window opens.



Note The Elself action looks the same as the If action; changing the choice in the Action drop-down menu changes the type of condition.



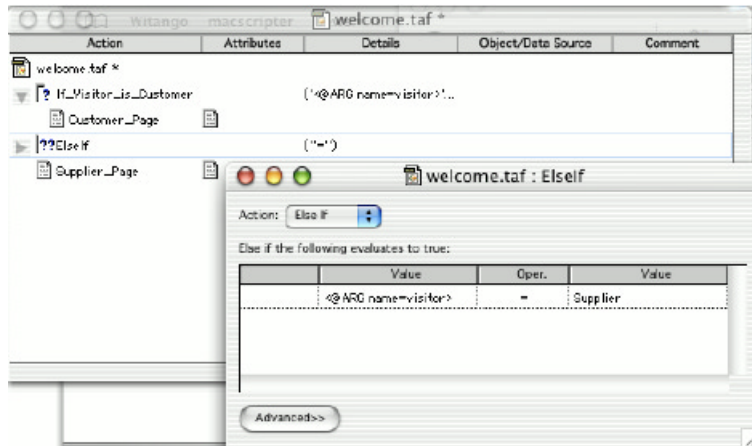
If the condition set within the If action is false, Witango simply drops down to the next action, the Elself action. Now, if the condition in the Elself action is true, Witango proceeds to execute the action(s) within the Elself action.

Furthermore, if the Elself action is false, Witango looks for an Else action (to be used shortly), which evaluates to true when neither the If nor the Elself condition is met. This is because the Else does not specify any condition.

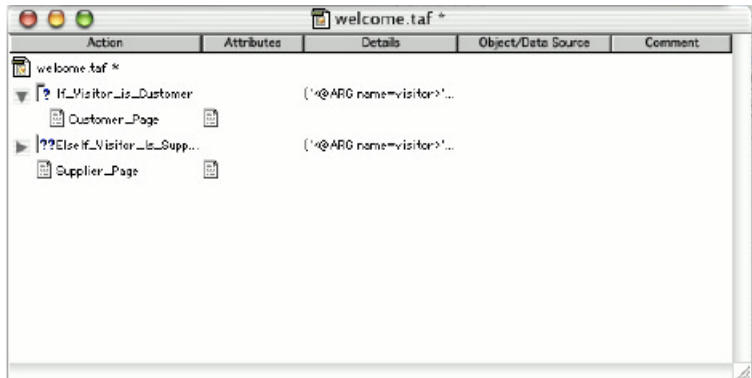
9 Enter the following values and operator as shown:

- Value:<@ARG Name=Visitor>
- Oper.: =

- Value:Supplier:



- 10 Close the Elself action window.
- 11 Drag the **Supplier_Page** action into the **Elself** action. Rename the **Elself** action to **Else_If_Visitor_Is_Supplier**.

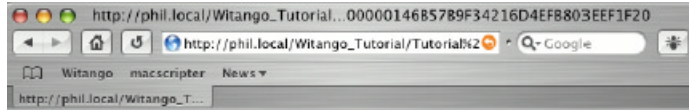


Note Supplier_Page becomes an indented action within the Elself action and is only executed if the Elself condition is met.

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 12 Save your changes to welcome.taf.
- 13 Return to your web browser and type `?visitor=customer` following the name of the application file (at the end of the URL): `welcome.taf?visitor=customer`.

I4 Press ENTER.

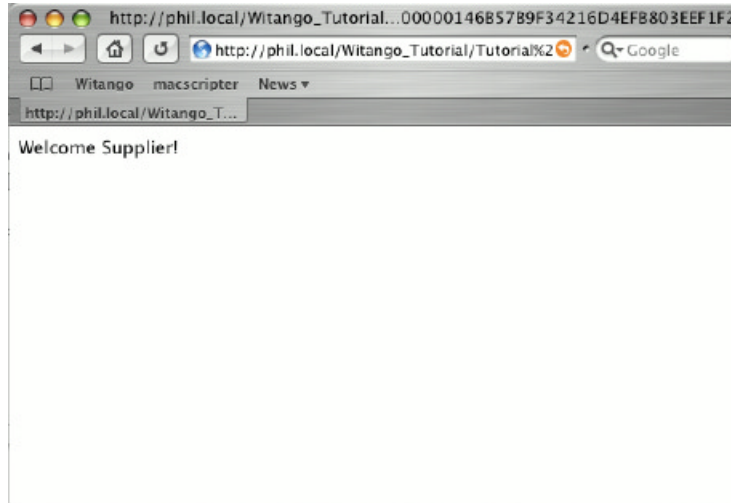


Q. What do you see?

A. Only the Customer Page appears. The search argument “visitor”, which you typed in the URL, was equal to customer, and so the Customer_Page action was executed. The Supplier_Page action was not executed because the Elself condition was not true — visitor did not equal supplier.

I5 Now, replace `?visitor=customer` at the end of the URL with `?visitor=supplier`.

16 Press ENTER.



Q. What do you see?

A. Only the Supplier Page appears. The search argument “visitor”, which you typed in the URL, was equal to supplier, and so the Supplier_Page action was executed. The Customer_Page action was not executed because the If condition was not true—visitor did not equal customer.

17 Delete ?visitor=supplier from the URL, and press ENTER.

Q. What do you see?

A. No page is displayed because neither of the conditions for generating a message was true, so there was no HTML to display.

You have now developed some logical processes to determine which HTML pages—found within one application file—are displayed. This logic is based on the condition or value for a certain variable (the search argument named `visitor`). This is an arbitrary variable name.



`<@ARG NAME=visitor>` is an example of a *Witango meta tag*. Witango meta tags are special tags that are interpreted by Witango Server when your application files are executed. They can do many different things in an application file, such as controlling the flow of information, performing an action on a data source, or setting or receiving variable values.

Meta tags look like HTML tags, with a beginning and ending angle bracket (`<` and `>`), but the character after the starting angle bracket is always `@`, 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 generally does not see the meta tags, as they are

interpreted by Witango Server before being sent to the web browser.

The `<@ARG>` meta tag returns the value of the *search argument* or *post argument* in the HTTP request that calls the application file. In the meta tag `<@ARG NAME=visitor>`, used in this example, the `NAME` attribute of the Witango meta tag `<@ARG>` specifies the value “visitor”.

You use the `<@ARG>` meta tag so that you have the flexibility of passing a value to an application file via either a search or post argument. So, what are search and post arguments?

A search argument (which can be referenced with the meta tag `<@SEARCHARG>`) refers to an argument passed in the URL. A post argument (which can be referenced with the meta tag `<@POSTARG>`) refers to an argument passed in the HTTP header. These will be discussed in greater detail in the next lesson.

Now that your application file is working, you may check in with the Boss.

LESSON A-4

Adding a Menu Page

Purpose

To create hyperlinks to pages within one application file.

Context

The user sends values for either customer or supplier by passing them in the URL, which in turn displays the corresponding pages using a Results action.

Exercise



The Boss



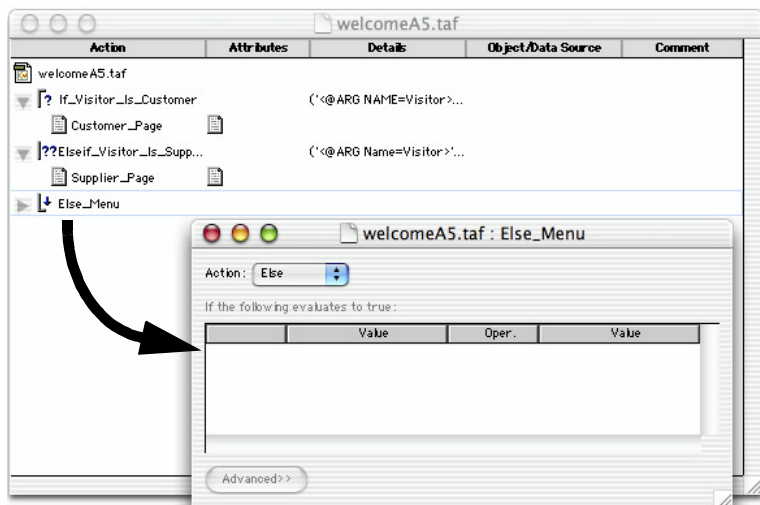
Else action

The corner of the Boss's lip turns slightly upward. Could that be a smile? "Not bad," he says. "Now, if we don't know whether a visitor is a customer or a supplier, I want them to be able to choose which page to go to. Show the user a menu of the two choices: customer or supplier."

- Q.** What action do you drag in to display a menu when the visitor has any value other than customer or supplier, including the condition that the visitor is empty, having not been passed at all.

A. An **Else** action.

- I** Drag an **Else** action into `welcome.taf` after the existing **Elseif** action.



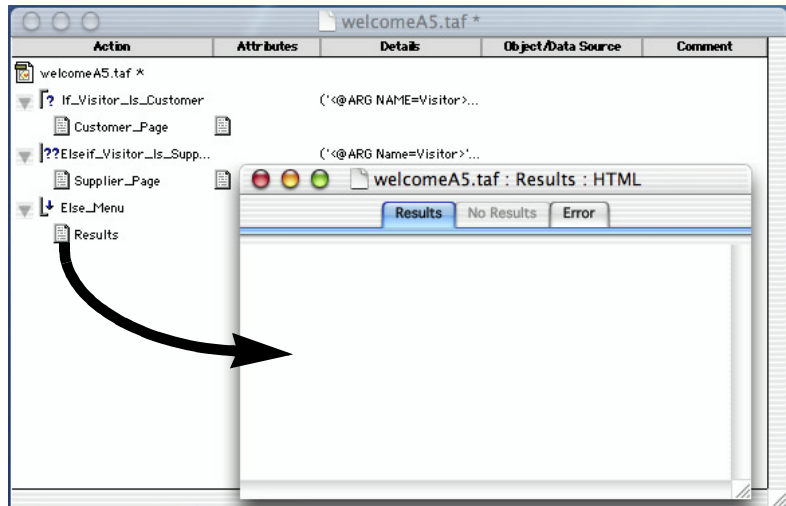
Note There is no place to enter values or expressions in the Else action. This is because it evaluates to true whenever the *If* or the *Elseif* condition is not met.



Results action

2 Rename the **Else** action to **Else_Menu**.

3 Drag a **Results** action into the **Else_Menu** action to create the menu.

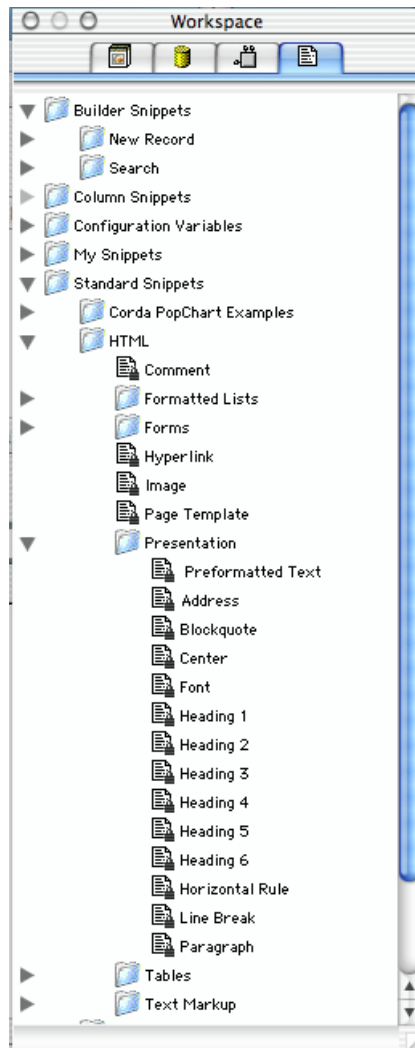


In the next few steps, you will add an HTML heading for this page using Witango's HTML snippets, so it is not necessary to type the HTML.



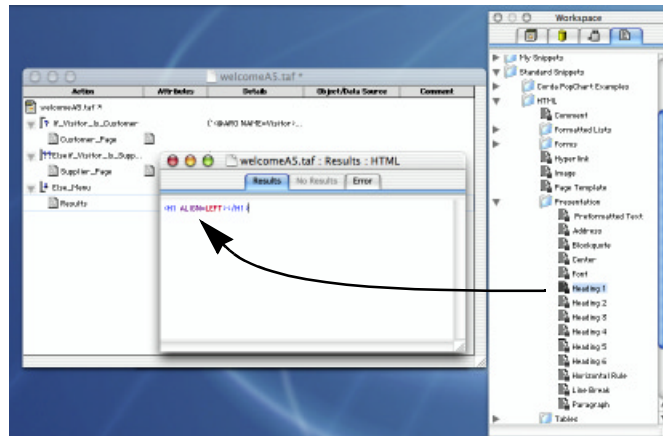
Snippets tab

4 Click the **Snippets** tab at the bottom right corner of the Workspace to display the Snippets Workspace. Now, expand the **Standard Snippets** folder, the **HTML** folder, and finally, the **Presentation** folder.

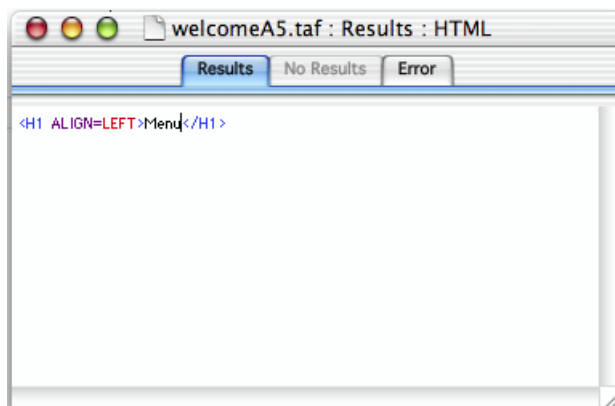


Snippets enable you to quickly access text, meta tags, and HTML that you commonly use. You drag snippets into the desired text field, such as the Results window, or you can place your cursor where you want the snippet inserted and simply double-click the snippet.

- 5 Double-click or drag over the **Heading 1** snippet to insert it into the Results HTML window.



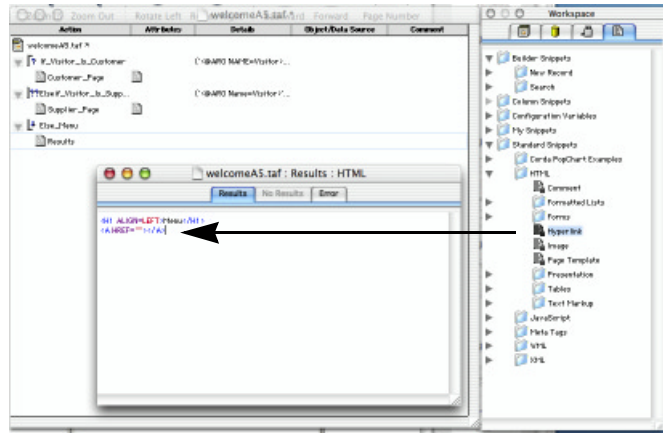
- 6 Type **Menu** between the <H1> and </H1> HTML tags in the **Results** action.



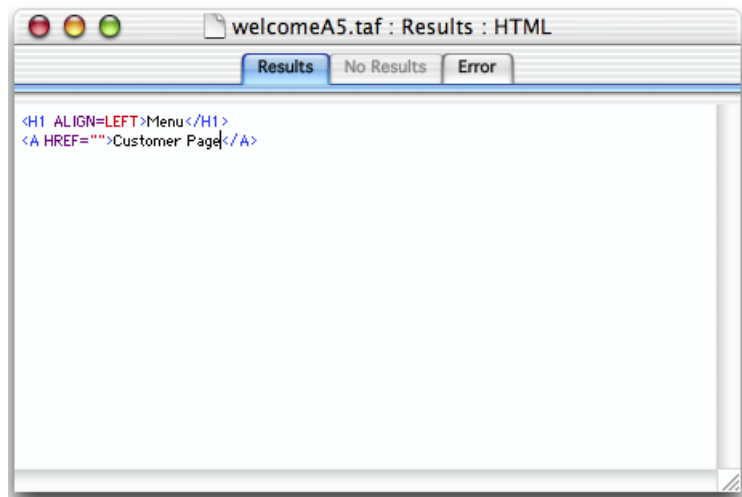
Now, let's add two hyperlinks: one linking to the Customer Page, and one linking to the Supplier Page. You'll use Witango's HTML Snippets to get the hyperlink code.

- 7 Move your cursor to a line beneath the heading. Scroll down the **Standard Snippets** in the Snippets Workspace until you find the

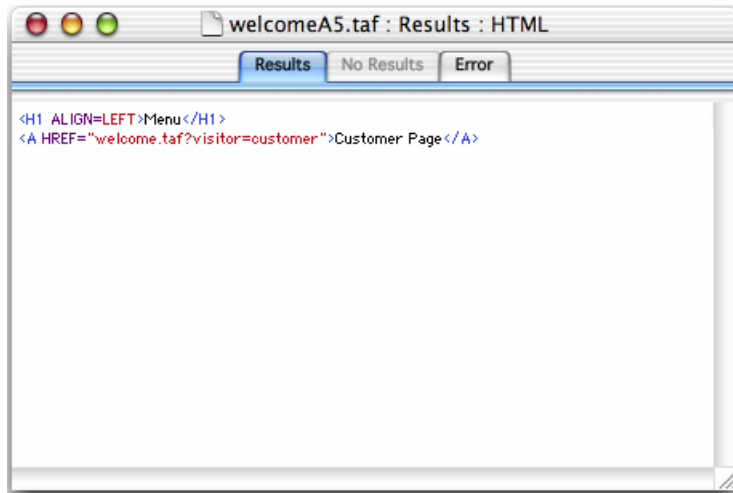
Hyperlink snippet, and add your first hyperlink by double-clicking it or dragging it over.



- 8 Type `Customer Page` between the `<A>` and `</A>` HTML tags. This is the text the user views as the link on the web page.



- 9 Place your cursor between the quotation marks after `HREF=`. Type `welcome.taf?visitor=customer`, which is the relative URL to the Customer Page.



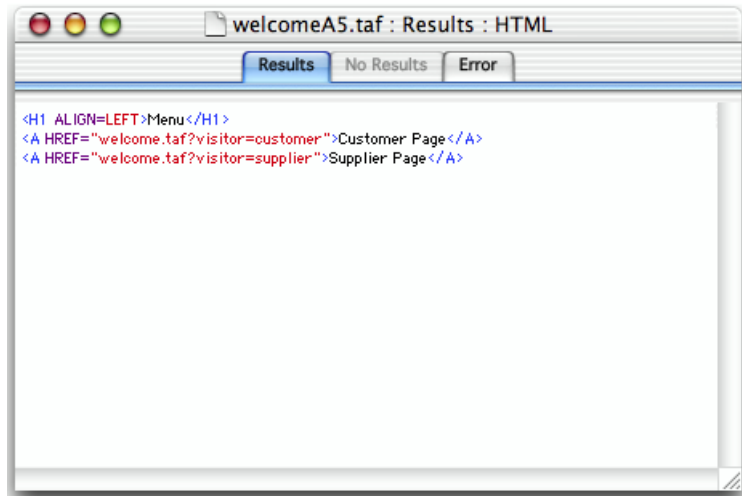
When the user clicks on the hyperlink Customer Page, Witango finds the Customer Page by following the relative URL referenced in the HREF attribute, which in this case is:

`welcome.taf?visitor=customer`.

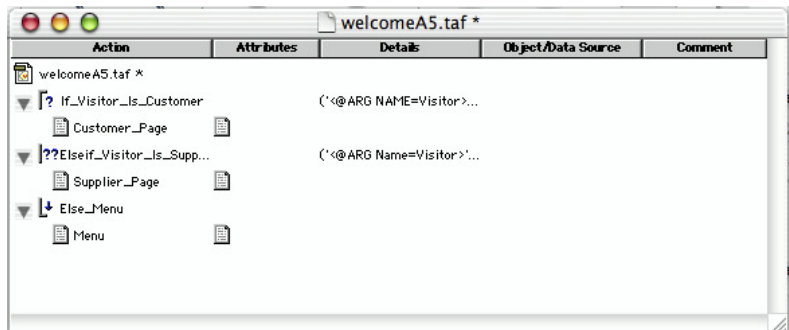
This is exactly how you called the file in the earlier lessons in order to see the Customer Page. The only difference now is that you do not have to type `?visitor=customer` in the URL. Instead, the hyperlink takes you there.

- 10 Copy the customer hyperlink tag that you just created, and paste it below on a blank line.
- 11 Change `customer`, which appears in two places on the new line, to `supplier`, so that this link selects the Supplier Page. Simply replace `customer` (in two locations) with `supplier`.

The HTML should look like this:

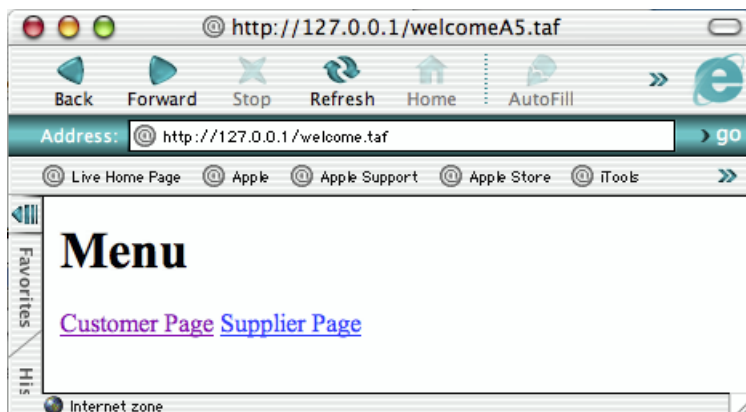


12 Close the window, and rename this **Results** action to **Menu**.



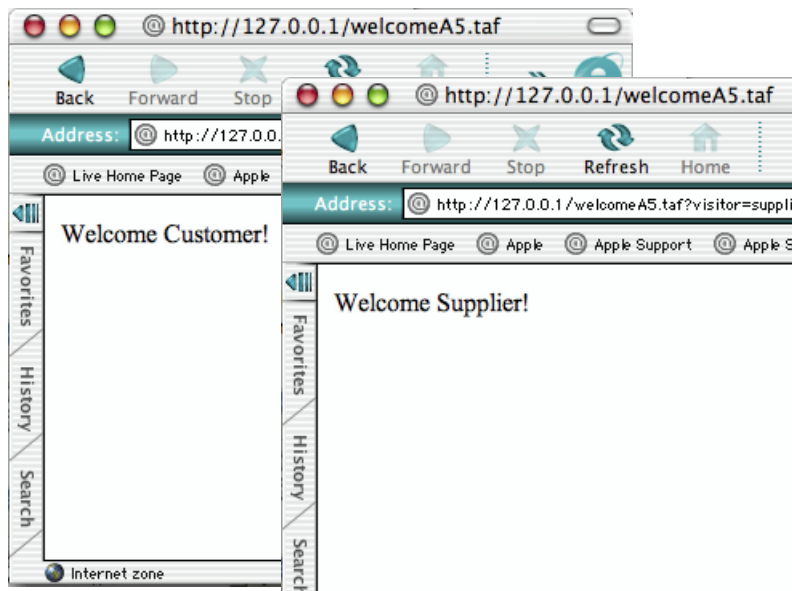
For details, see "Saving Witango Application Files" on page 5 and "Executing Witango Application Files" on page 5.

13 Save the `welcome.taf`, and return to the web browser. Delete `?visitor=supplier` from the end of the URL, and press ENTER.



14 Finally, test the links to ensure they work.

Clicking on **Customer Page** should take you to the Customer Page; clicking on **Supplier Page** should take you to the Supplier Page.



Note Notice that clicking on a hyperlink on the Menu Page appends either `?visitor=customer` or `?visitor=supplier` to the URL; this is passing values of the variable called *visitor*.

Now that your application file is working, you may check in with the Boss.

LESSON A-5

Every Good Witango URL Has...

Purpose

To explain and use the following meta tags and components in a Witango URL:

- `<@APPFILE>` or `<@APPFILEPATH>`
- `<@CGI>`
- a navigation argument
- `<@USERREFERENCEARGUMENT>`.

For example:

```
<A HREF="<@CGI><@APPFILE>?_function=list&
<@USERREFERENCEARGUMENT>">Record List</A>
```

Context

In the previous lesson you created two hyperlinks. In both links, you *hard-coded* the path and name of the Witango application file, meaning that you typed the specific route and file name into the `welcome.taf` file. You also hard-coded a call to the Witango plug-in by not adding anything in before `welcome.taf`. You could also have hard-coded a call to the Witango CGI by adding `witango-bin`—or whatever the path to your CGI folder is—plus `wcgi.exe` (the name of the Witango CGI) before `welcome.taf`.

Hard-coding file paths, file names and the use of the Witango plug-in or Witango CGI is not a good idea because changes to paths, names, or calls to the Witango plug-in or CGI may become difficult to replace in all of your Witango application files and absorb a lot of development resources.

In order to solve this problem, Witango has a number of meta tags that make it possible for you to avoid hard-coding file names, paths, and calls to the CGI or Witango plug-in. This lesson teaches you which meta tags to use to make proper Witango hyperlinks or HTML form action references.

Exercise



The Boss

The Boss summons you in to an official meeting to review your work thus far. You sit down in the tiny chair across the room from the Boss, who is elevated behind his gigantic desk. The Boss speaks:

“I noticed you hard-coded a number of things in the two hyperlinks you created. First, you hard-coded the name of the Witango application file in the hyperlink. The links really only reference the file that we are currently running, `welcome.taf`. Is there a meta tag you can use that will

reference this? That way, we can change the name of the file and not have to worry about changing the hyperlink code.

I'm also worried about switching back and forth between our two Witango Servers, one of which is running the Witango CGI, and one of which is not. Can you make the hyperlinks dynamic so that you can run your file here or there without having to make any changes to the code?"

<@APPFILE>

On your current Menu Page, both hyperlinks are set to link back to the same Witango application file, `welcome.taf`. This is because the application file has different sections to it (Customer_Page and Supplier_Page). You are linking to the same file, but going to a different section of that file, determined by the value of the argument named `visitor`.

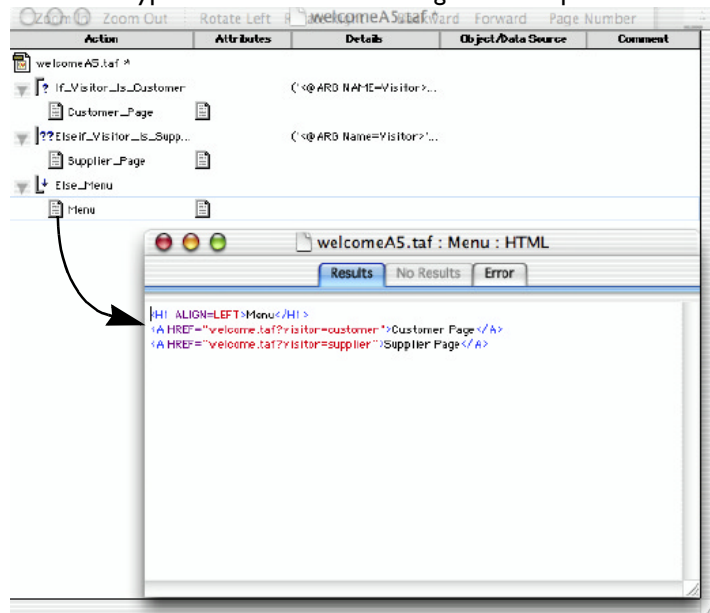
In Witango, there is a meta tag that returns the file path and name of the current application file being executed. The tag is `<@APPFILE>`. During execution of a Witango application file, `<@APPFILE>` will resolve to the path to, and the name of, the file being executed relative to the web server document root. For example, if you executed `welcome.taf` (from the Tutorial folder), during execution `<@APPFILE>` would resolve to `Witango\Tutorial\welcome.taf`.

If you renamed the file to `welcome2.taf`, during execution `<@APPFILE>` would resolve to `Witango\Tutorial\welcome2.taf`.

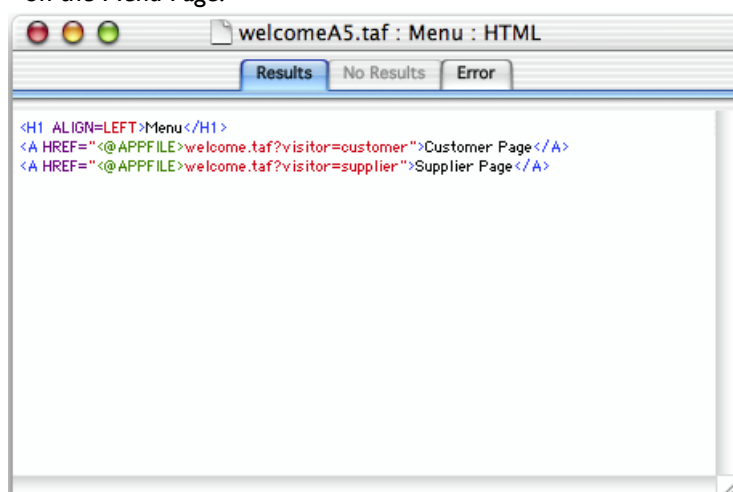
In the menu hyperlinks, replacing `welcome.taf` with `<@APPFILE>` results in an application file that can change locations or names, without the hyperlink code needing to be changed. Witango creates the hyperlink file references dynamically.

- I Open `welcome.taf` in your Witango Studio, if it is not already open.

- 2 Double-click the Results HTML action named **Menu**, which contains the menu hyperlinks. The HTML editing window opens.



- 3 Replace the text `welcome.taf` with `<@APPFILE>` in both hyperlinks on the Menu Page.



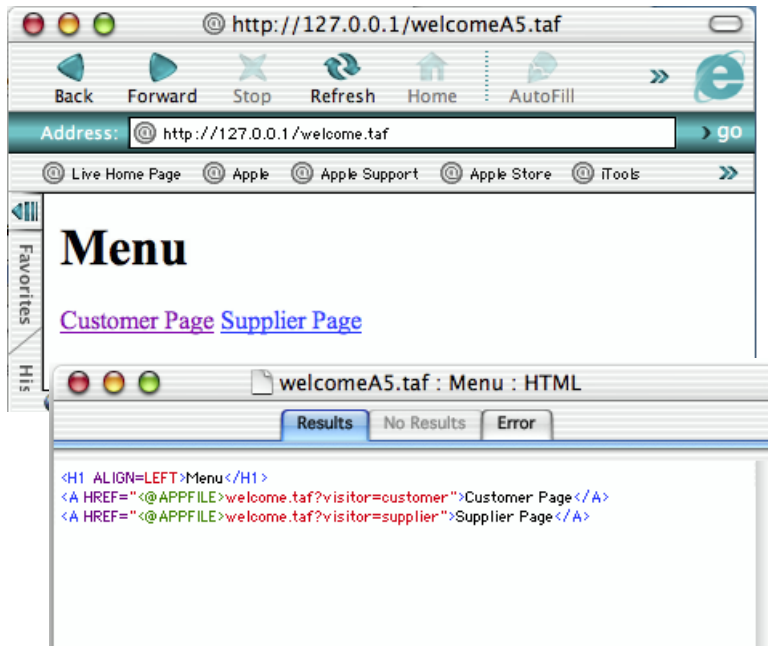


Witango Studio's syntax coloring helps you see the difference between meta tags, HTML tags, and plain text.

Using the default syntax coloring, meta tags appear green and HTML markup appears blue; attributes of either meta tags or HTML markup appear purple, and the attribute values are in red. Plain text is black.

If you are using the Witango CGI, the next step will add a Witango meta tag that accesses the CGI. See “<@CGI>” on page 50.

- 4 Save the file and execute it in the web browser. The hyperlinks should function just as they did before, if you are using the Witango plug-in.
- 5 If you check the HTML source code of the Menu Page, you will see that <@APPFILE> properly resolved to the path and name of the Witango application file you executed.



If you needed to create a hyperlink to another Witango application file—not a link to the current one—you could use the <@APPFILEPATH> meta tag. <@APPFILEPATH> resolves to the path of the current Witango application file, but does not include the name of the current Witango application file. An example of a hyperlink using this meta tag is

```
<A HREF="<@APPFILEPATH>search.taf">Search for
```

Products

The name of the new file you wish to reference is appended to <@APPFILEPATH>. <@APPFILEPATH> resolves to the path of the current file, such as Witango/Tutorial/. If the current file and the one you are referencing are in the same folder, then this meta tag allows you to avoid hard-coding the file path in the hyperlink.

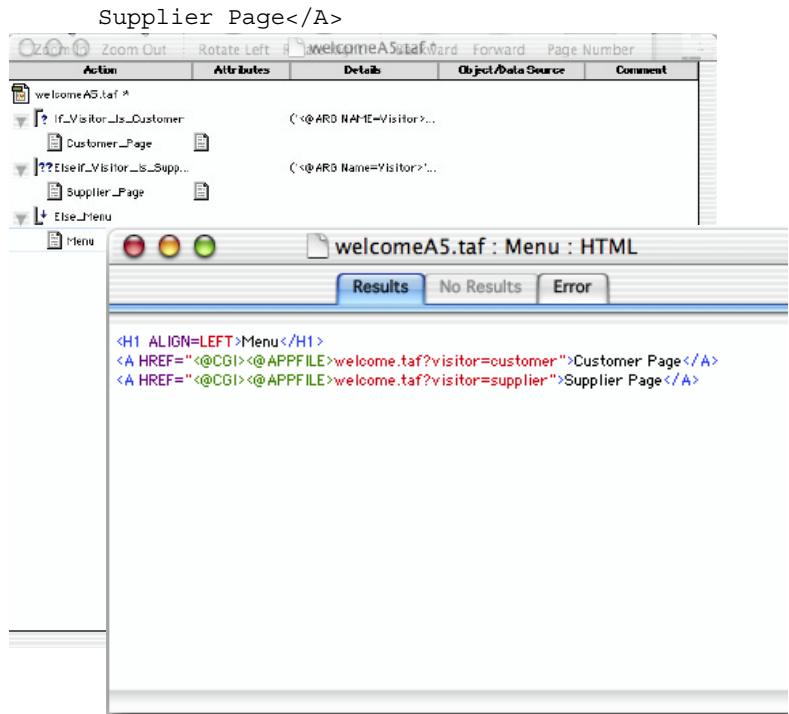
<@CGI>

In order for you to avoid hard-coding the use of the Witango CGI or the Witango plug-in, Witango must determine whether the CGI or the plug-in is being used during execution, and build hyperlinks that continue to use the appropriate item. The meta tag that enables you to do this is <@CGI>.

During execution of a Witango application file, <@CGI> resolves to the call to the Witango CGI (that is, the path to the CGI folder and the name of the Witango CGI) if the call is being made, and resolves to nothing if the plug-in is being used. Hence, this meta tag should be included in all hyperlinks and form action attributes that call Witango application files.

I Add <@CGI> to each hyperlink in the Menu Page as follows:

```
<A HREF="<@CGI><@APPFILE>?visitor=customer">
Customer Page</A>
<A HREF="<@CGI><@APPFILE>?visitor=supplier">
```



Note Do not place any slashes between `<@CGI>` and `<@APPPFILE>`; Witango inserts the appropriate slashes (folder separators) in the right places.

2 Save the file and execute it in the web browser.

First, call the file using the Witango CGI. For example:

```
http://yourserver/Witango-bin/wcgi/Witango/
Tutorial/welcome.taf
```

The Menu Page appears. Click one of the hyperlinks. Note in the URL that the call to the Witango CGI was included in the hyperlink. This is because the `<@CGI>` tag in the hyperlink resolved to your CGI directory (for example, `cgi-bin/`), and the CGI program, `wcgi.exe`, ensuring that the use of the CGI was continued.

Second, call the file using the Witango plug-in, if available for your web server. For example:

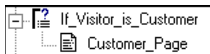
```
http://yourserver/Witango/Tutorial/welcome.taf
```

The Menu Page appears. Click one of the hyperlinks. Note in the URL that the Witango plug-in was used; there is no call to the Witango CGI. This is because the `<@CGI>` tag in the hyperlink resolved to nothing, ensuring that the use of the Witango plug-in was continued.

Navigation Argument

An example of a navigation argument is `visitor=customer`. The only reason for passing this argument in a menu hyperlink is so that Witango can navigate to the appropriate section of the Witango application file, that is, the Customer Page, when that hyperlink is clicked.

Navigation arguments are necessary because typical Witango application files have multiple end-results. If, Elseif, and Else actions are used to keep sections separate, and these actions check for certain conditions. If the conditions are true, the sub-actions are executed.



In `welcome.taf`, the If action checks whether or not the argument “visitor” is equal to “customer”. If this is true, the Customer Page is executed. Hence, if you wish to go to the Customer Page, you must pass `visitor=customer` in your hyperlink or Form Action attribute.

Witango’s builders use `_function` as their navigation argument. It does not matter what you use, as long as you are consistent. If you are checking for `_function`, then pass `_function`. The underscore, by the way, has no magical importance; it is simply part of the name of the argument.

Since the menu hyperlinks already contain a navigation argument, there are no steps to be done here. Simply keep in mind that a typical Witango URL contains a navigation argument.

<@USERREFERENCEARGUMENT>

In Witango, you can assign values to Witango’s memory, thereby keeping those values for longer than one execution of a Witango application file. Values in memory are called Witango *variables*. Because this is a rather advanced discussion for so early in the tutorials, keep in mind that variables will be examined in greater detail in Tutorial D.

Witango variables can be assigned for *specific users* by assigning the values to the *user scope*. The actual value that Witango can use to tie to each user variable is called the *user key*. So, what does Witango use as its user key?

The default setting in Witango provides a user reference number for the user key. The user reference number is a hexadecimal number that

Witango generates for each new user. This user reference is sent as a *cookie* to the user's web browser, and each time the user comes back to the Witango web site, Witango asks the web browser for the cookie and identifies the user. This way, Witango is maintaining a user's session; Witango is maintaining *state*.

If a user comes to the site and there is no user reference in the cookie, Witango assumes this person is someone new, and assigns them a user reference.

What if a user has turned cookies off in their web browser? How can Witango track that user? If Witango cannot find the user reference in the cookie, it looks to see if there is a user reference argument that contains the hexadecimal number. When Witango finds one, then Witango does not assign a new user reference. The name of the argument it looks for is `_UserReference`.

An argument is passed either by a hyperlink or a form. Therefore, if you wish to track users using the user reference, and you do not wish to rely on cookies to pass the user reference, you must append a user reference argument and its hexadecimal value to every hyperlink and form action attribute on your web site. This way, whenever the user clicks on a button or link on your site, you can be assured that the user reference will be passed, guaranteeing a way to track the end user.

Use the `<@USERREFERENCEARGUMENT>` meta tag to append the user reference to each hyperlink and form action attribute. During execution, this meta tag resolves to

```
_UserReference=hexadecimal_number
```

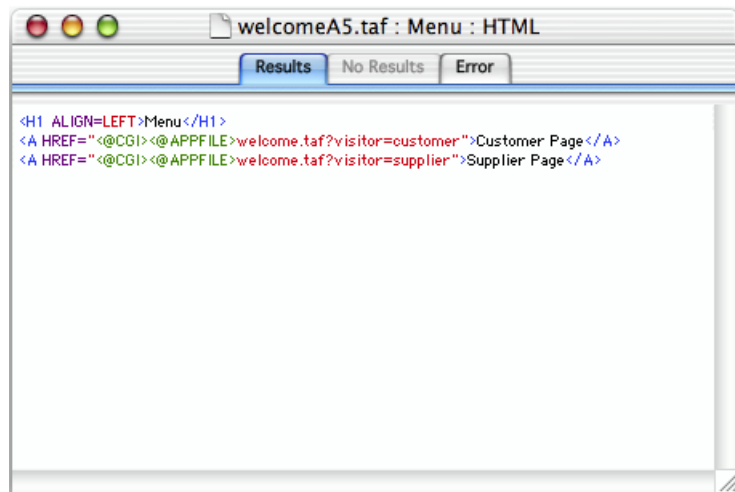
For example:

```
_UserReference=52694DB25AB8AF7D37728219
```

I Append `<@USERREFERENCEARGUMENT>` to each hyperlink in the Menu Page as follows:

```
<A HREF="<@CGI><@APPFILE>?visitor=customer&
<@USERREFERENCEARGUMENT>">Customer Page</A>
```

```
<A HREF="@CGI><@APPFILE>?visitor=supplier&
<@USERREFERENCEARGUMENT">">Supplier Page</A>
```

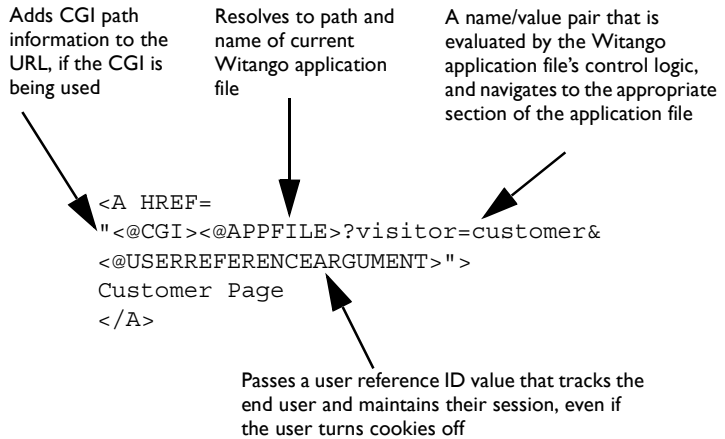


Note The ampersand (&) before <@USERREFERENCEARGUMENT> serves as a separator between arguments.

- 2 Save the file and execute it in the web browser, calling up the Menu Page.
- 3 Click one of the hyperlinks. The `_UserReference` argument was passed in the URL with its long hexadecimal value.

Summary

Every good Witango URL has the following components:



Finally, here is an example of a form that uses the above components:

```
<FORM METHOD=POST
ACTION="<@CGI><@APPFILE>?visitor=customer&
<@USERREFERENCEARGUMENT>">

Email: <INPUT NAME=Email TYPE=TEXT>

</FORM>
```

This creates a form on a web page where the user can enter their e-mail address. Clicking the submit button calls the Witango application file specified in the URL with the appropriate name-value pairs and user ID passed through.

LESSON A-6

Passing Values Using HTML Forms

Purpose

To pass information to other pages within the same application file using an HTML form.

Context

HTML hyperlinks can pass values in the URL (known as the *Get* method). There are two possible disadvantages of passing values via the *Get* method: they are visible, and there is a space limitation of 255 characters only. If you were collecting a user's user identification and password, you would not want it to appear in the URL. Also, if you were collecting a lot of data, you would not want it passed in the URL because it may be truncated due to space limitations. Hence, there is another way to pass values: the *Post* method.

The *Post* method passes values in the HTTP header; the values are invisible to the end-user, and more information can be passed.

In this lesson, you replace your hyperlinks with a form in order to demonstrate how to pass the argument `visitor` using the *Post* method.

This lesson teaches how to create variables in HTML, and demonstrates how values are passed using these variables.

Exercise



The Boss



Results action

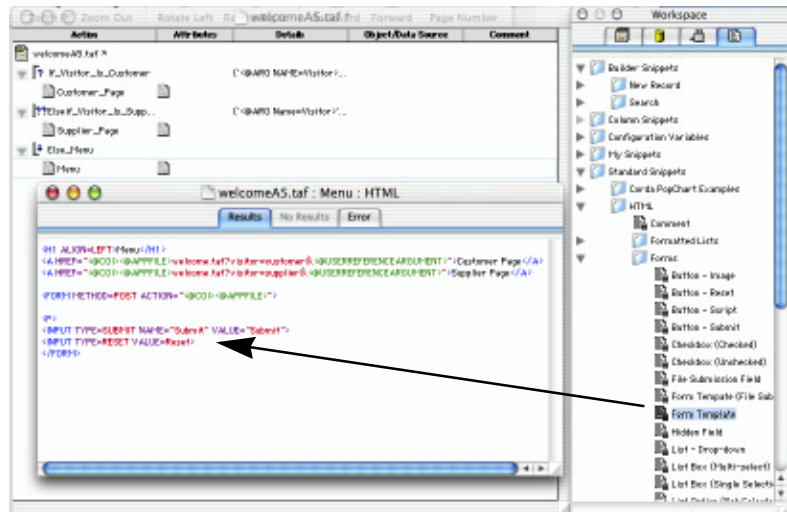
The Boss just may be impressed, because he is now calling you by your first name. However, he's mispronouncing it, perhaps just to keep you alert.

Now he wants you to use an HTML form to create a web page with a more interesting user interface.

- 1 Delete your current Menu Page by selecting it and pressing **DELETE**. Now, drag a new **Results** action into **Else_Menu** (renaming it **Menu**) so that you may begin working on the form solution.
- 2 Under **Standard Snippets**, in the **HTML** snippet folder, open the **Presentation** folder, and drag in or double-click the **Heading 1** snippet.

Type `Menu` between the tags, and place your cursor on a line beneath the heading.

- 3 Again, in the **HTML** snippet folder, open the **Forms** folder. Double-click or drag over the **Form Template** snippet to insert it.



An HTML *form* has an opening tag, `<FORM>`, and a closing tag, `</FORM>`. There are METHOD and ACTION attributes in the opening tag. The value of the METHOD attribute is POST, and as you may recall, the Post method passes arguments invisibly through the HTTP header. The ACTION attribute specifies the two meta tags, `<@CGI>` and `<@APPFILE>`. These represent the default path to the current Witango application file. `<@CGI>` returns the full path and name of the Witango CGI. With server plug-in/extension versions of Witango, this meta tag returns nothing. `<@APPFILE>` returns the path to the current application file, including the file name. See the previous lesson for more information.

Within the form tags, there are HTML input elements defined with specific attributes. Examine the parts of the first input:

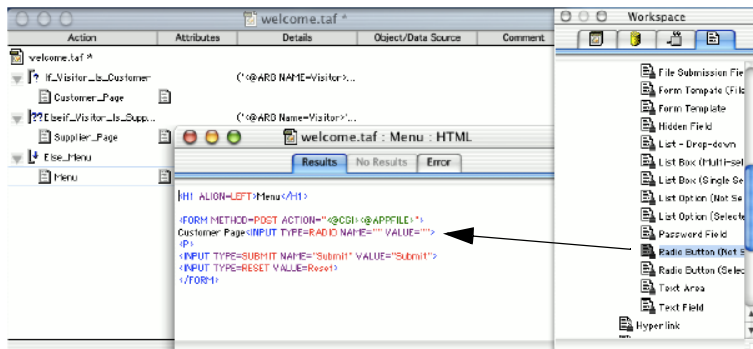
```
<INPUT TYPE=SUBMIT NAME="Submit" VALUE="Submit">
```

An input with TYPE=SUBMIT creates a button on the web page. When clicked, the button collects and sends (or submits) the values entered by the user to Witango Server, which then returns the appropriate web page. The NAME attribute establishes and names a variable, in this case, Submit (the name of the button). The VALUE attribute determines the value of the variable, which in this case is the same as the NAME.

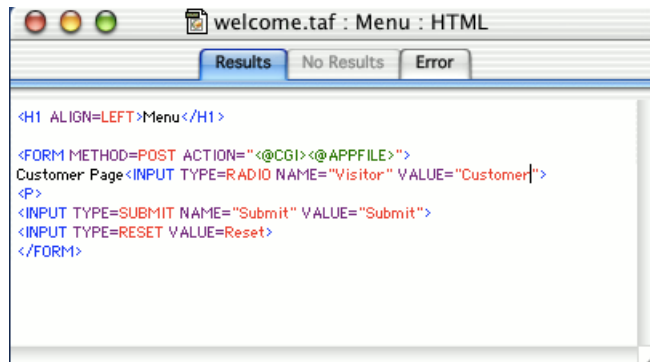
- 4 Without moving your cursor, type Customer Page, followed by a space. This is the text the user will see on the web page.



- 5 Scroll down the **Form Snippet** list until you find the **Radio Button (Not Selected)** snippet. Now, insert your first radio button.



- 6 Type visitor between the quotes after NAME=. Place your cursor between the quotes after VALUE=, and type customer.





Examine the parts of this new line. `Customer Page` is the text that is displayed beside the form field, so that the user knows what he or she is choosing. The Input Type in this case is `RADIO`, which creates a radio button on the web page. The Name variable is `visitor`, and its Value is `customer`. If the user selects this radio button, and then clicks the Submit button, he or she passes this argument to the Witango Server, which then returns the corresponding web page, "Welcome Customer!". The result is the same as when `visitor=customer` was passed in the URL as part of a hyperlink. In this case, the only thing that has changed is *where* the argument is being passed.

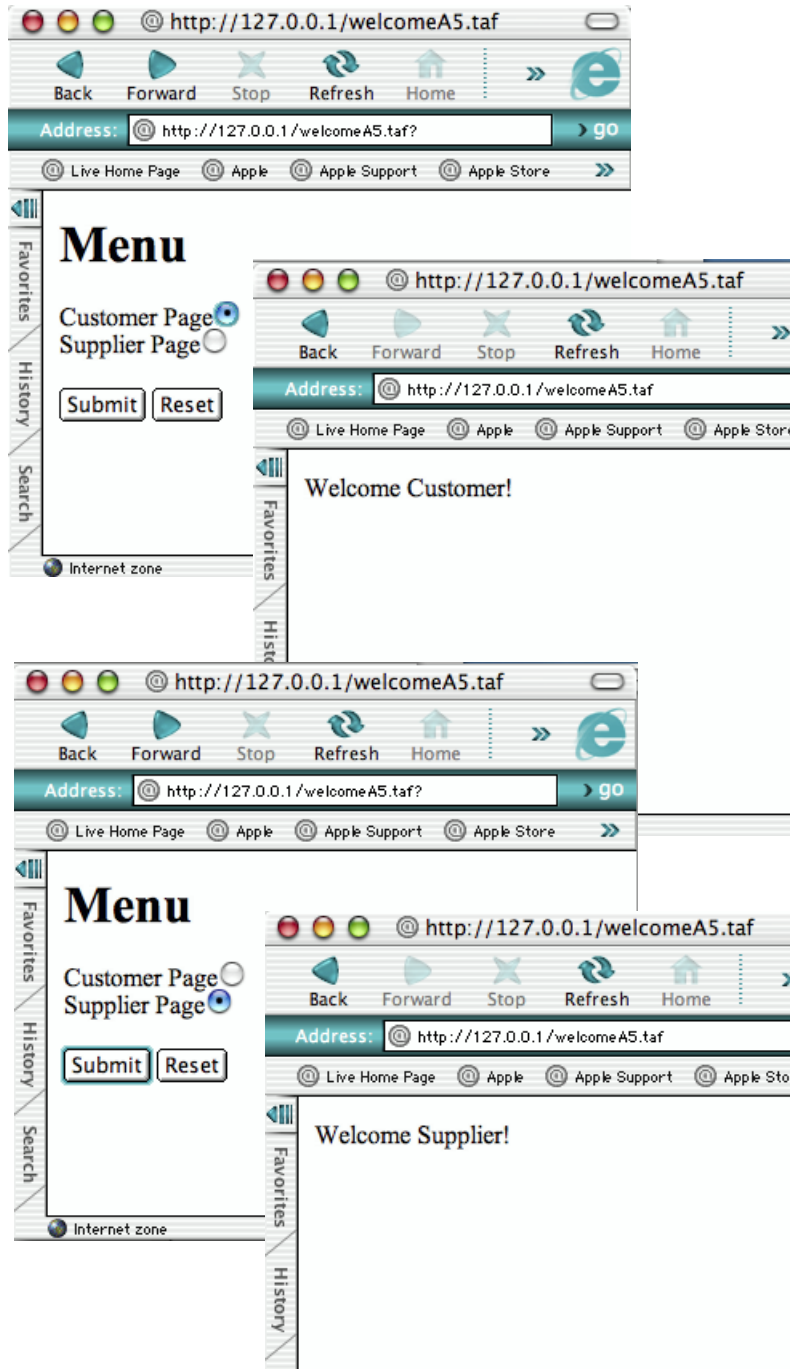
- 7 Copy the entire line containing the `customer` radio button, and paste it below on a blank line.
- 8 Change the word `customer`, which appears in two places on the new line, so that this link selects the Supplier Page. Simply replace `customer` with `supplier`, as shown here:

```
<H1 ALIGN=LEFT>Menu</H1>

<FORM METHOD=POST ACTION="@CGI"><@APPFILE>
Customer Page<INPUT TYPE=RADIO NAME="visitor" VALUE="customer"><BR>
Supplier Page<INPUT TYPE=RADIO NAME="visitor" VALUE="supplier">
<P>
<INPUT TYPE=SUBMIT NAME="Submit" VALUE="Submit">
<INPUT TYPE=RESET VALUE=Reset>
</FORM>
```

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- Save `welcome.taf`, return to the web browser, and execute `welcome.taf`. Select a radio button, and click **Submit** to make sure the appropriate page is displayed.





Examine the URL for the Welcome Supplier page. Notice that the name-value pair `visitor=supplier` was not passed in the URL. However, in order to arrive at the “Welcome Supplier!” page, it must have been passed somehow.

Where and how the Submit button sends the name-value pairs are specified by the `FORM METHOD` and `ACTION` attributes in the opening form tag.

The Form Method attribute specifies how the values will be passed. There are two form methods, *get* and *post*. In this case, the form method is *post*. As mentioned at the beginning of the lesson, the *post* method passes values invisibly in the HTTP header, which is passed with every web page—this is why you cannot see the name-value pair in the URL. The values are being posted to the CGI or other server-side program, such as Witango. Variables and values passed with the *post* method are termed *post arguments*. Similarly, values passed with the *Get* method are termed *search arguments*. Both of these types of arguments are important to knowing and understanding Witango.

The form’s action specifies the page the user goes to when he or she clicks the button. In this case, the action uses meta tags, `<@CGI><@APPFIL>`, which act as place holders for the file path currently being used. `<@CGI>` specifies the CGI server-side program, but simply resolves to nothing if CGI is not being used. `<@APPFIL>` is resolved by the server to the full folder path and name of the current application file.

Now that your application file is working, you may check in with the Boss.

Data Sources

B

Creating a Data Source to Use for the Music Store

A *data source* contains all the information Witango needs to find and access a database.

This tutorial covers how to create and use a data source.

There are two lessons in this tutorial:

B-1: Creating a Data Source

B-2: Displaying Sale Items From the Database

This tutorial assumes that you have installed your database, defined the sample MUSIC database and imported the sample data provided. This process is described on page 7 in the introduction to this tutorial.

LESSON B-I

Creating a Data Source

Purpose

To create a data source that contains information about how to connect to a specific database.



Note Witango needs a data source to interact with a database.

Context

The database contains your products: the names, prices, and sale prices of the CDs, the artists, and the music categories. To connect to a database, you must set up a *data source*. This lesson sets up a data source through Witango.

An ODBC / JDBC system data source stores information about how to connect to the specified database. A system data source is visible to all users of your computer, including NT services. The data source contains all the information Witango needs to find and access a database.



Note You can also set up a data source using the ODBC (Open Database Connectivity) or JDBC (Java Database Connectivity).

Exercise



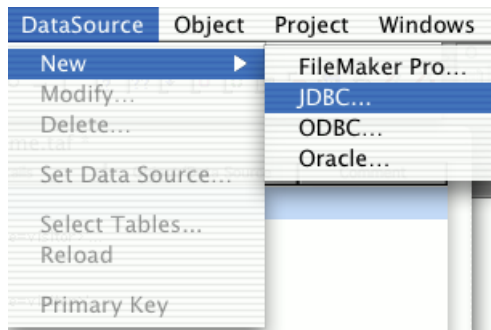
Because of your good work, the Boss has definitely warmed up to you. In fact, he has instructed the custodial staff to store the mops and pails someplace other than your office.

Now, he would like you to create a data source called *Music* that links to the CD inventory.

- 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.



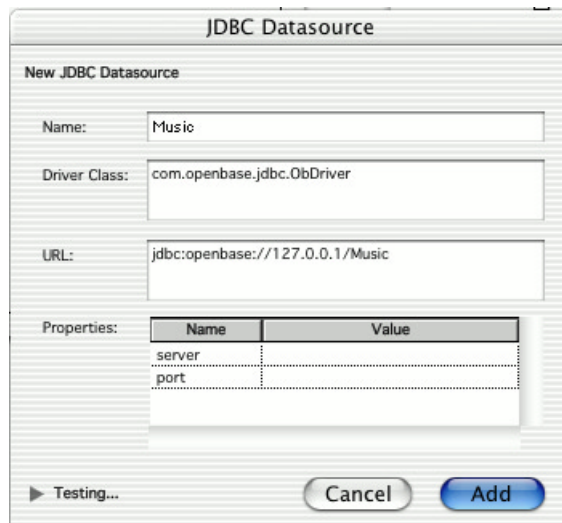
- 3 From the **DataSource** menu, choose **New**, then choose **JDBC...** from the submenu.



The **JDBC Data Source** window appears:

A screenshot of the 'JDBC Datasource' window. The title bar says 'JDBC Datasource'. Below the title bar, it says 'New JDBC Datasource'. There are three text input fields: 'Name:', 'Driver Class:', and 'URL:'. Below these is a 'Properties:' section with a table. The table has two columns: 'Name' and 'Value'. The first row has 'server' in the 'Name' column and an empty 'Value' column. The second row has 'port' in the 'Name' column and an empty 'Value' column. At the bottom left is a 'Testing...' button with a play icon. At the bottom right are 'Cancel' and 'Add' buttons.

4 Complete the details in the JDBC Data Source Window:

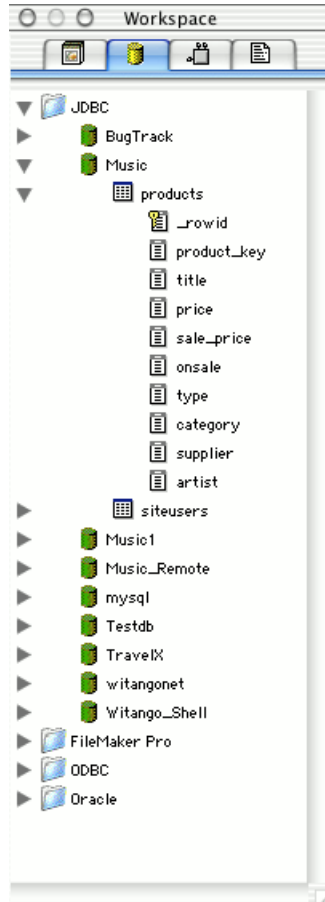


The image shows a 'JDBC Datasource' configuration window. It has a title bar 'JDBC Datasource' and a subtitle 'New JDBC Datasource'. There are three text input fields: 'Name' with the value 'Music', 'Driver Class' with the value 'com.openbase.jdbc.ObDriver', and 'URL' with the value 'jdbc:openbase://127.0.0.1/Music'. Below these is a 'Properties' section containing a table with two columns: 'Name' and 'Value'. The table has two rows: 'server' and 'port', both with empty value cells. At the bottom left is a 'Testing...' button with a right-pointing arrow. At the bottom right are 'Cancel' and 'Add' buttons.

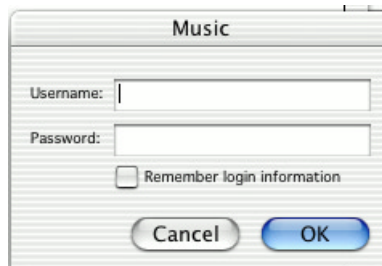
Name	Value
server	
port	

Click the **Add** button:

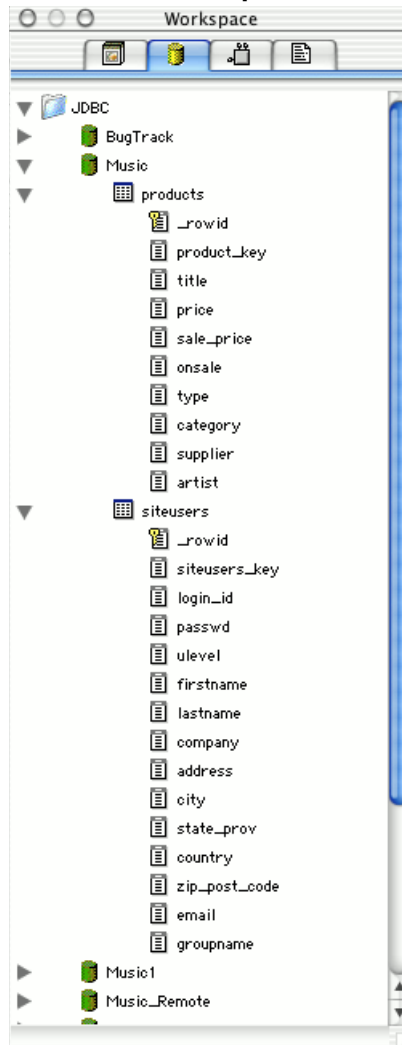
- 5 Open the **JDBC** folder in the **Workspace** area. You should now see a **Music** folder.



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



- 7 You did not setup a login during the set up process so just leave the **name** and **password** fields blank and click **OK**. You should now see two tables in the **Music** database: **products** and **siteusers**.



Note You have set up the **Music** data source that points to the database files in the database folder. Witango Studio and Witango Server can now access the database. You use the **Music** data source for the rest of the tutorials.

Now that your data source is working, you may check in with the Boss.

LESSON B-2

Displaying Sale Items From the Database

Purpose

To display records from the database using a *Search action*.

Context

A dynamic web site often involves retrieving information from a database. In this lesson, you discover how to retrieve data from a database, and you determine what meta tags to use to reference that data.

Exercise



The Boss

The Boss is actually looking at you now when he talks to you. Could this be *respect*? He says, “I want the customer to know what CDs are on sale. Can you display this information from the database?” “Yes Boss,” you say confidently, “yes I can!”

Q. What Witango action do you need to drag in to retrieve the sale item(s) from the *products* table for display on the Customer Page?

A. A *Search action*.



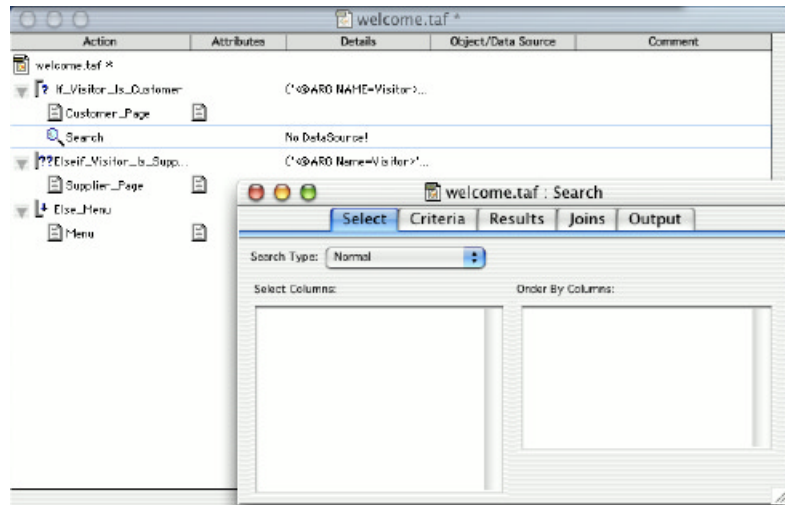
In order to search for records in a database, you need to set up a Search action. The Search action allows you to specify the fields (or columns) and rows within the database that you want to view. These fields are determined using your data sources, which define the fields in the databases and also link to them.



Search action

I Open `welcome.taf`, and drag in a **Search** action, placing it within

the **If** action and below **Customer_Page**.



The function of a Search action is to retrieve records from a database.

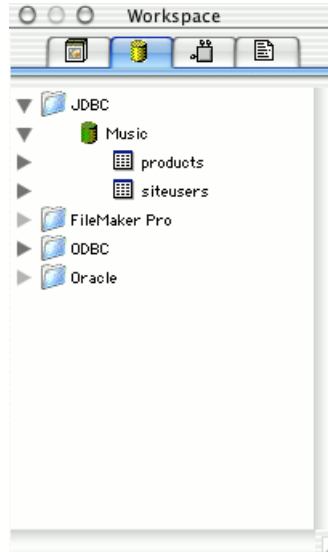
There are four main groups of settings (or tabs) for a Search action: *Select*, *Criteria*, *Results*, and *Joins*. At this point, we are concerned with the first two.

The *Select* option allows you to select the fields to retrieve, and, if you wish, to determine the order in which the information will be returned.

The *Criteria* option allows you to determine what rows from the database are returned by the action. If no criteria are specified, all rows are returned; otherwise, each row in the database (or specifically, in this case, the *products* table) is compared to your criteria, and only those meeting them are returned.

The first step of setting up the **Search** action is to select the field(s) to retrieve.

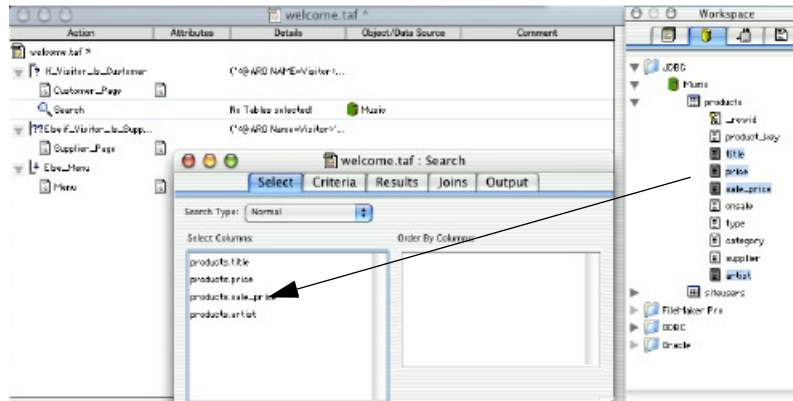
- 2 In the **Data Sources** workspace, open the **ODBC** folder and expand the **Music** data source .



Data Sources contain all the information needed to connect to a particular database. In this case, you use the Music data source to tell this application to connect to the database containing all of the Products information. Witango Studio allows you to create and manage—modify and delete—data sources.

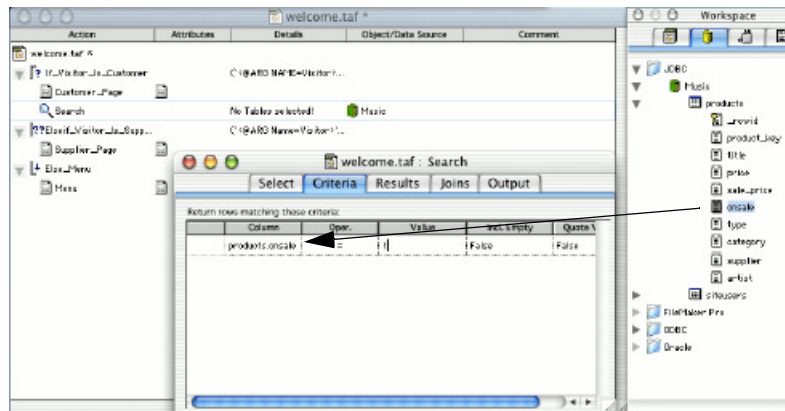
Both Witango Studio and Witango Server need to have access to data sources. Witango Studio uses a data source to show you database information in the form of database layouts and their fields. Witango Server requires the same data source so it can access the layouts and fields specified within the application file.

- 3 View the columns in the **products** table. Drag the **Title**, **Price**, **Sale_Price** and **Artist** columns into the **Select Columns** area of the **Search** action.

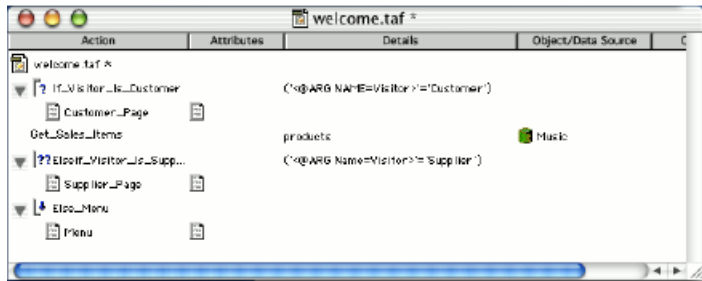


The next step is to determine which items are on sale. Do this by setting up a criteria for the search.

- 4 Switch to the **Criteria** tab of the **Search** action and drag the **onsale** column into the Criteria window. Type **True** into the **Value** column.

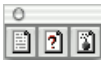


5 Close the **Search** action, and rename it to **Get_Sales_Items**.



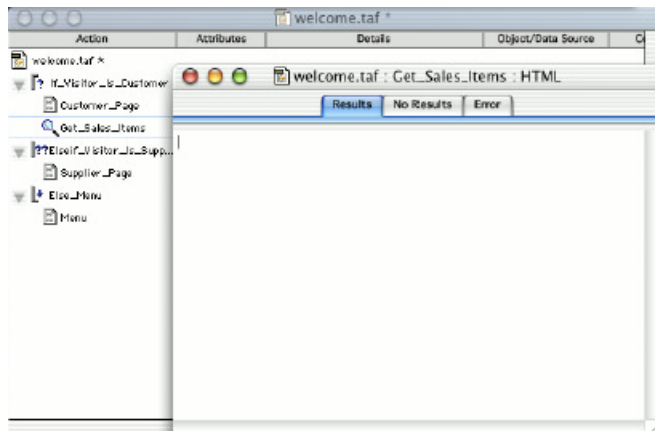
You have now set up the Search action to *retrieve* data from your database based on particular criteria. The next step is to reference that data in an HTML window to display it to the customer. You use Witango meta tags to reference the data.

Regarding the HTML, there are two options: it can be placed in another Results action below the Search action, or it can be placed in the Results HTML attribute of the Search action. In this lesson, you place the HTML in the Results HTML attribute of the Search action.



Attributes Toolbar

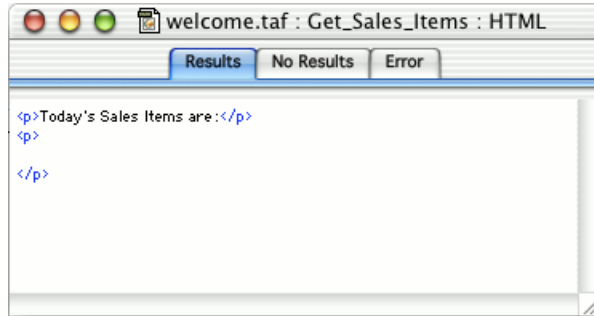
6 On the **Attributes** toolbar, click the **Results HTML** attribute button to open the Results HTML window for the **Get_Sales_Items** action.



Before setting up the display of the actual sales information, you need to add an appropriate heading.

7 Type `<p>Today's sale items are:</p>` and press Enter.

- 8 Now type `<p></p>` and then place the cursor between the two tags that you just typed as shown below.



You now need to add some HTML to display the results of the search that you created earlier in this lesson.

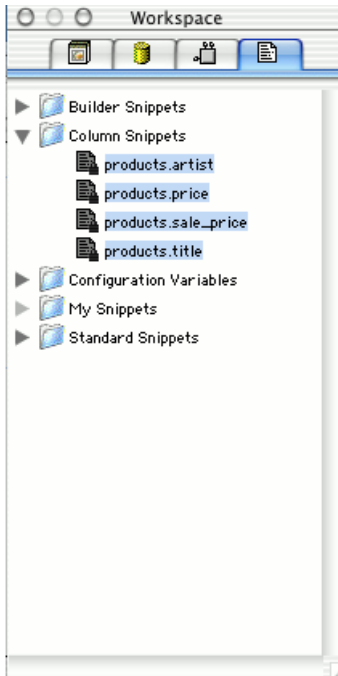
- Q.** Which type of meta tag would you use in this case to retrieve and display the items on sale?
- A.** The `<@COLUMN>` meta tag.



Like a Results action, Results HTML can contain meta tags that Witango Server processes. While the HTML, JavaScript, or regular 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 the value of the specified column to be placed in the HTML.

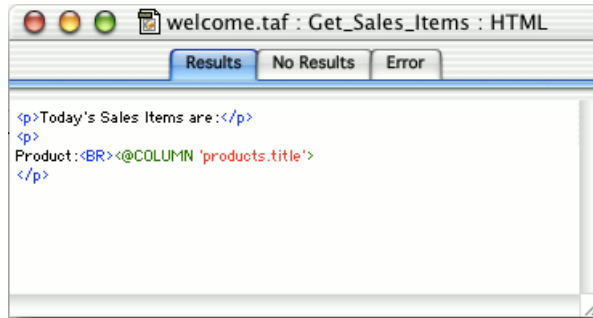
- 9 To insert the appropriate `<@COLUMN>` tags, switch back to the Snippets Workspace, and open the **Column Snippets** folder.



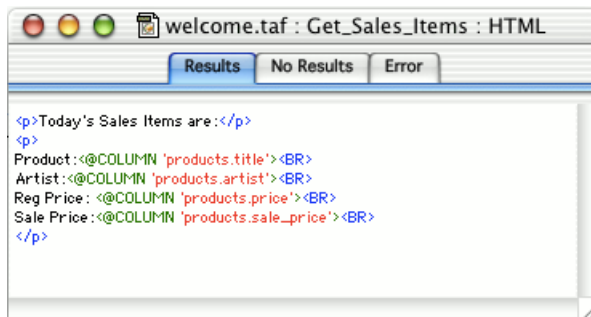
Note The column snippets that appear correspond directly to the columns you dragged into the Select Columns window of the Search action. If you had dragged a fifth column into the Search action, a snippet for that column would also appear here.

The Column Snippets folder of the Snippets Workspace becomes active when Results HTML (or New Record Response HTML in the New Record Builder) is open.

- 10 In the **Results HTML**, type `Product :` and then leave a space. Double-click or drag in the **products.title** column snippet, then type `<BR>`, and press Enter.



- 11 Type `Artist:` and leave a space. Double-click or drag in the **products.price** column snippet, type `<BR>`, and press Enter.
- 12 Type `Reg.Price :` and leave a space. Double-click or drag in the **products.price** column snippet, type `<BR>`, and press Enter.
- 13 Type `Sale Price:` and leave a space. Insert the **Products.Sale_Price** column snippet, type `<BR>`.



What you have just created will only display the details of the first record in the search results. The first record is contained in the first row of the data in the search results array. In order to display further records you need to have the information contained in these three columns repeated for each record/row of data in the search results. Witango has a metatag `<@ROWS>` that simplifies this task considerably.

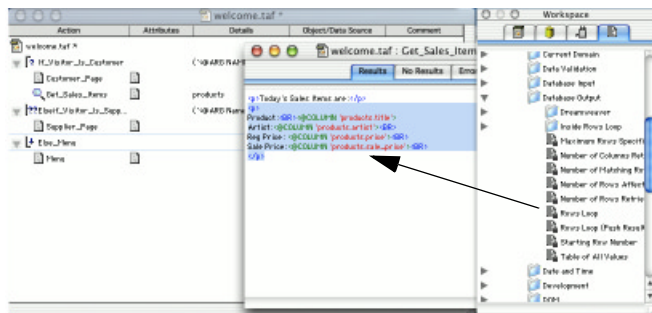


The Results HTML appearing between a `<@ROWS>``</@ROWS>` tag pair is processed once for each row of the result set generated by the action within.

An `<@ROWS>` loop allows iteration over the rows of a given array. It places a copy of the text between the opening and closing tags for each row of the array. In this action, the array to loop over defaults to the value of the result set (the products on sale).

Arrays are a special type of variable that allow you to store many different values in a structured format. This is distinct from the standard variable, which only stores one value. Arrays are structured as a table with rows and columns; values are saved at each row and column intersection. This is similar to the way values are saved in a database table: as rows and columns.

- 14 Select all the `<@column>` lines and the `<p>``</p>` tags that surround them. Open the **Meta Tags** folder, and the **Database Output** folder. Double-click or drag in the **Rows Loop** meta tag snippet.

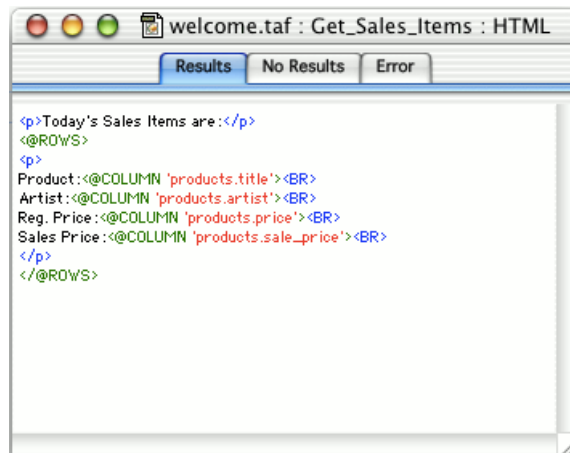


The **Rows Loop** is inserted both above and below the selected text.



Witango snippets have a useful property—they can be used to surround text, as you have done here. The special `¥` in the snippet tells Witango Studio to insert the snippet either with the cursor at that point or, if there is a current selection, to surround the selected text with the snippet; that is, putting what is before the `¥` symbol before the selected text, and putting what is after the `¥` symbol after the selected text.

- 15** Finally, below the `</@ROWS>` tag, type `<HR>`. Press Enter.

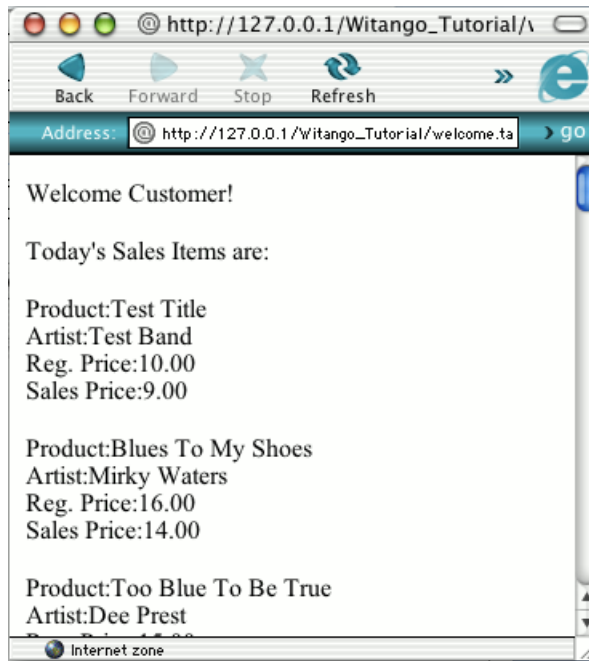


Note Typing `<HR>` and pressing Enter places a line across the bottom of the web page. This will be used to separate information to come later.

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 16** Save the file, return to the web browser, and reload `welcome.taf`. Choose **Customer Page**. The page should display with the

customer message followed by the sale item message, a list of sale items, ending with a line across the page.



In this lesson, you used the `<@COLUMN "table.column">` meta tag within an `<@ROWS>` block to reference the database values returned by an action in the application file, where `table` and `column` represent the names of your table and column, respectively. There are other meta tags that can be used to return results.

`<@COL NUM=__>` returns the value of the column `NUM` in the current record of a result rowset or array. This tag may be used in any Results HTML. For instance, `<@COL NUM=2>` refers to the second column in the current row.

`<@VAR NAME=resultset [r,c]>`, where `r` and `c` are row and column numbers, references an array variable called `resultSet`. The `resultSet` variable is generated automatically by most actions, and contains the results of that action. This variable array is stored in Witango's request scope, which means it is purged at the end of the file execution. Also, if you do two searches in a row, for example, the second search will produce a `resultSet` array that overwrites the `resultSet` of the first search.

Now that your application file is working, you may check in with the Boss.

Witango Builders and Projects



Working With the Search Builder, the New Record Builder, and Witango Projects

This tutorial teaches you more about working with the Witango builder tools: New Record Builder, and Search Builder. They are easy-to-use shortcuts that generate a series of actions that perform several functions, either inserting a new record or searching for a record and returning results. Once you have created files using the builders, you organize them into a project.

There are five lessons in this tutorial:

C-1: Search Builder for Customer Page

C-2: Setting the Number of Matching Records Returned

C-3: Inserting Records Into the Database

C-4: Controlling User Input: Creating a Drop-down Menu, Radio Buttons & Setting Required Fields

C-5: Creating a Project

LESSON C-I

Search Builder for Customer Page

Purpose

To use the Search Builder to build an application file that searches for records in the database and returns results.

Context

It is important to keep in mind that Witango application files work using an *action-based metaphor*. Each action carries out a specified function and may be combined with other actions to produce a web page. *Witango builders*, on the other hand, are useful for new Witango users, because they allow you to think in terms of web pages for searching, getting record details, inserting and deleting records, as opposed to actions in a programming flow.

Builders generate Witango actions that carry out certain common functions in an application file. For example, you use the *Search Builder* to very quickly generate actions that produce a search form page, which leads to a record list page with hyperlinks to a record detail page where records can be updated or deleted.

Builders are not separate from an application file. They are part of it, and so are the actions that they produce.

In this lesson, you use the Search Builder to build actions that produce three web pages when the file is executed in the web browser. The three tabs of the Search Builder correspond to three pages that it creates:

- *Search*, which produces a Search Form page
- *Record List*, which produces a Record List, or *Results* page
- *Record Detail*, which produces a Record Detail page.

The Search Builder allows you to control the look of these web pages.

The *Search Form* page displays a form into which the user can enter search criteria. The *Record List* page lists the results of the search, but is only displayed if there are matching records. If there are no matching records, Witango does not display the *Record List* page; it displays the *No Results HTML* page. On the *Record List* page, you can make one of the fields a hot-link to the *Record Detail* page, which displays more details about a specific matching record.

Drilling down into the database is the process of moving through the three pages in the web browser to a specific record. In this lesson, you determine those fields the user searches on, those fields that will be

returned for each matching record in the Record List, and those which will be displayed for a specific record on the Record Detail page.

Exercise



The Boss

The Boss has been kind enough to send someone in to replace that annoying, flickering neon light tube in your office. Things are looking brighter! Then, the Boss calls. He wants the customer to be able to perform a product search on a linked file.

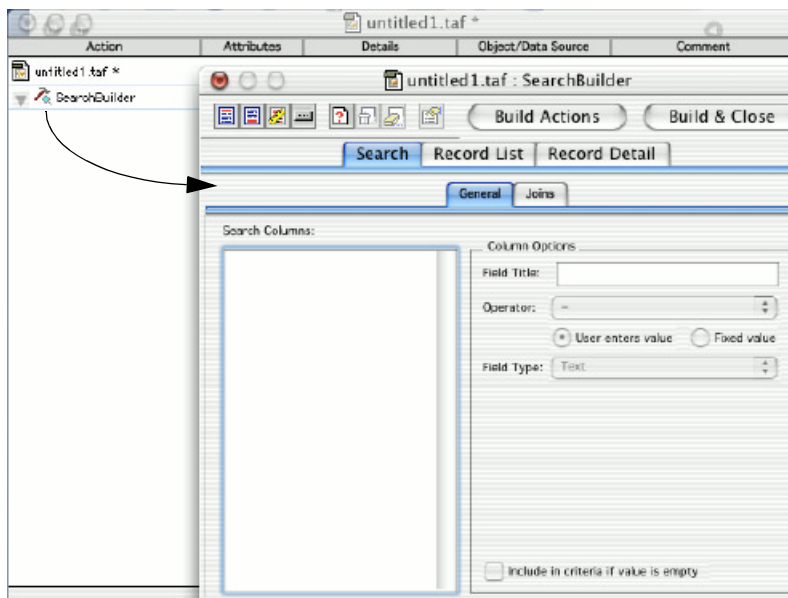
You need to place the customer search functionality into a new application file.

- Q.** What Witango action do you drag in to provide a search form page, a record list page, and a detail page for the database?
- A.** A *Search Builder* action.

- I** Open a new application file, and drag the **Search Builder** icon into the application file window from the **Actions** toolbar.



Search Builder



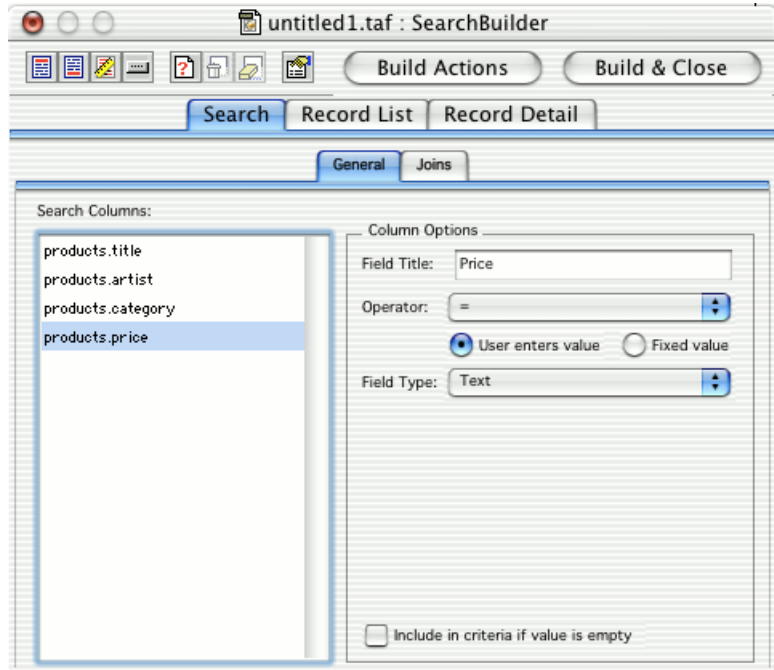
If you have not done so already, you will have to complete the lesson on “Creating a Data Source” on page 54 in order to complete the lessons in this tutorial.

- 2** In the Data Sources Workspace, expand the **Music** data source to view the columns of the **products** table. Drag the **products.title**, **products.artist**, **products.category**, and **products.price** columns into the **Search Columns** area of the Search Builder.



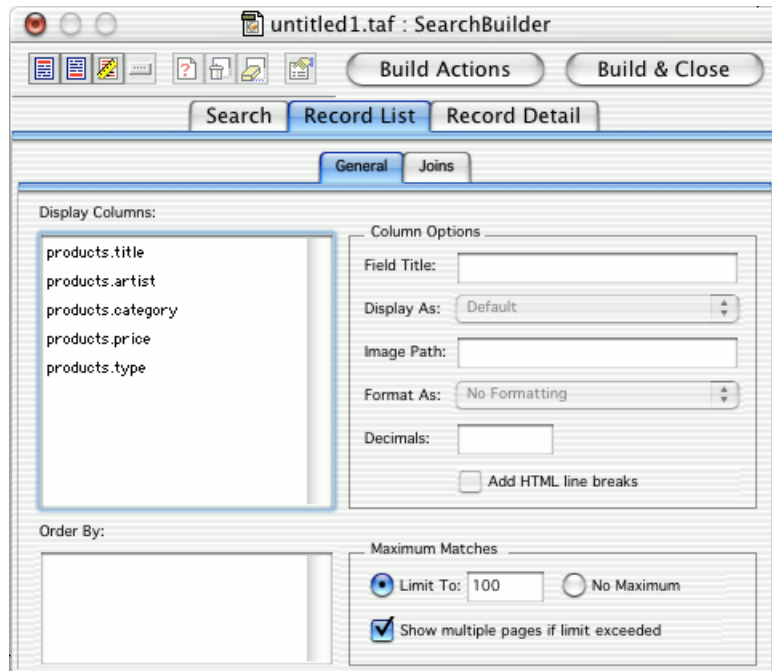
Tip The order of the columns in the Search Columns list determines the order of the fields on the resulting search form. From this list, you can reorder columns or delete them.

- 3 In the **Search Columns** area, select **products.price**. In the **Column Options** area, choose **User Enters** from the **Operator** drop-down menu to let the user enter the search operator on the search form.



- 4 Click the **Record List** tab of the Search Builder, and drag in the **products.title**, **products.artist**, **products.category**, and

products.price columns once again. Also, drag **products.type** into the **Display Columns** area.

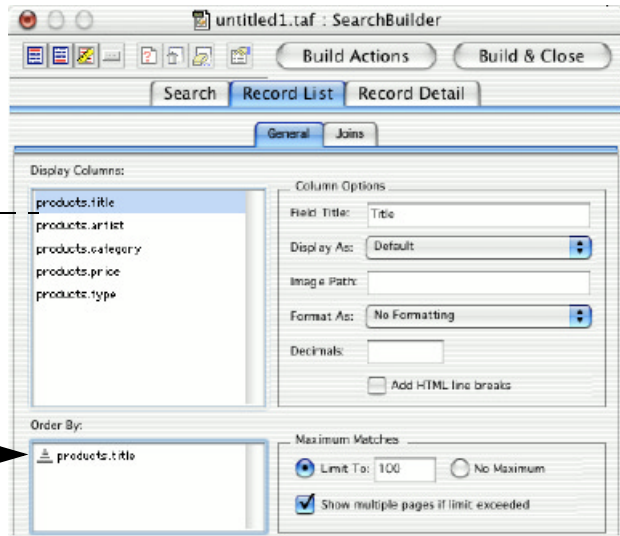


You do not necessarily want to return *all* the fields for each matching record in the database on the Record List page. If there were a large number of records being returned with full details, the user would have to wait unnecessarily while Witango Server downloads all the fields for all the matching records.

It is unlikely that the user wants the details of every matching record. Instead, the user requires full details only when they have narrowed their search to one matching record, that is, on the Record Detail page.

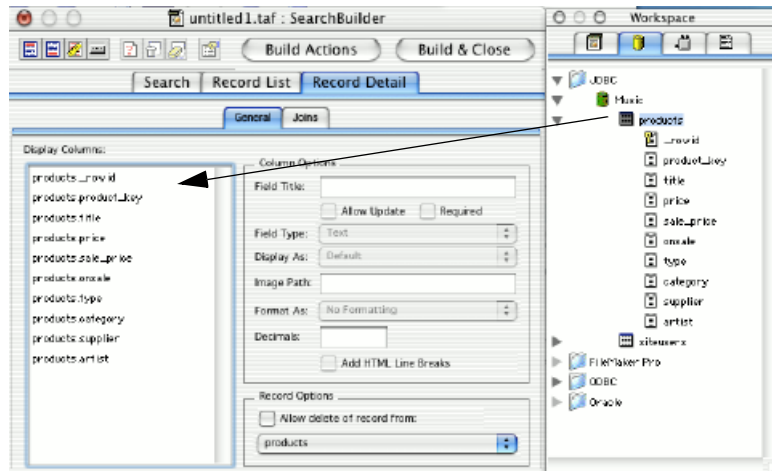
- 5 From the **Display Columns** area, drag **products.title** into the **Order By** area to sort the returned record list by the product name.

Drag the column into the **Order By** window to sort the returned record list by the product name.



Note In the *Order By* area, the triangle to the left of the column name determines whether the ordering is in ascending or descending order. To change the order direction for a field, click the triangle. 

- 6 Click the **Record Detail** tab of the Search Builder. Display all columns of the **products** table by dragging the **products** table into the **Display Columns** area.

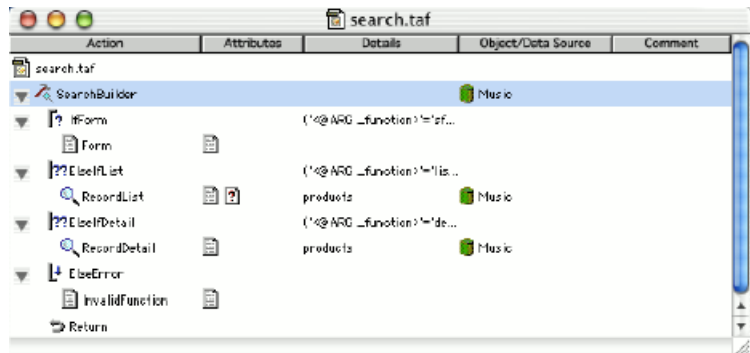


Here you display all the fields for a particular CD. Instead of dragging each field into the Display Columns list individually, or highlighting them and dragging the group over, highlight the *products* table and drag it over. This brings all the fields within that table into the Display Columns list.

Of course, in your own Witango applications you can drag over any fields that are relevant to your needs. In fact, the only requirement for the Search Builder is that *any* two of the three tabs of the Search Builder each have at least one field in them.

Do nothing else in the builder at the moment. The purpose of this lesson is to build a quick application file that searches the database. When you have accomplished the functionality, you may return to the builder and make adjustments or additions to suit the product search.

- 7 Click **Build Actions** to build the actions and close the Search Builder. Your application file should look like this:



Clicking *Build Actions* generates the series of actions required to perform the task of searching the database.

Build Actions

These actions appear in the application file window, grouped under the Search Builder icon. You may expand or close the group to hide or reveal the actions within it.

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 8 Save the file as `search.taf`.
- 9 Return to the web browser, and access `search.taf` using the appropriate URL. This will be the same URL that you used in Tutorials A and B, but substituting `search.taf` for `welcome.taf`.

Search for all Country music under \$20. Your search should produce a **Matching Records** page of the Country music CD titles under \$20, as shown below:

The screenshot shows a web browser window titled "@ Search" with the address bar set to "http://127.0.0.1/search.taf". Below the address bar are navigation buttons: Back, Forward, Stop, Refresh, and a "go" button. The search form contains the following fields:

- Title:
- Artist:
- Category:
- Price: (with a dropdown arrow)

Below the form are two buttons: "Find" and "Reset Values".

The second screenshot shows the browser window titled "@ Matching Records" with the address bar set to "http://127.0.0.1/search.taf?_function=list&_J". The page content indicates there are 7 matching records and displays matches 1 through 5 in a table.

Title	Artist	Category	Price	Type
When Will The Cheatin' End?	Slim Chance	Country	17.00	CD
Tumbleweeds and Tears	Loretta Lonesome	Country	14.00	CD
Stop Your Cheatin'	Slim Chance	Country	14.00	CD
Still Cheatin'	Slim Chance	Country	11.00	CD
My Dog Left And So Did You	Backroad Boys	Country	18.00	CD

The browser status bar at the bottom indicates "Internet zone".

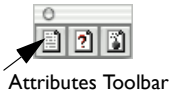


If there is a matching record in the database, Witango displays the record (or records if there is more than one match). The displayed page corresponds to the Record List page. The first field or column value is hot-linked, allowing you to click it to receive the Record Detail page

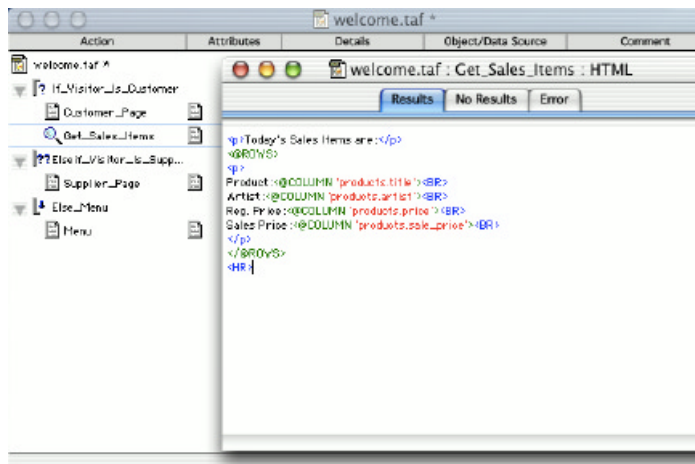
containing all the available information for the product you searched.

If you were to enter a search criterion in the Category field for which there were no matching records, a message stating “No matching records were found” would appear. You may try this now. This message can be changed to something more friendly later, but it will do for now.

Now you need to return to `welcome.taf` and add a link to `search.taf` on the Customer Page.

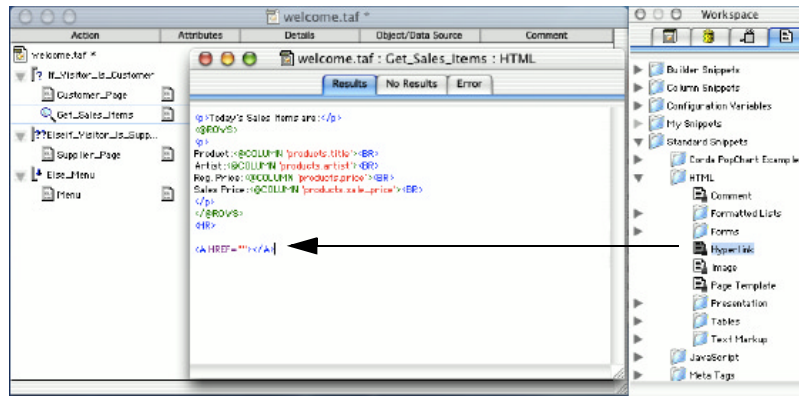


- I0 Open `welcome.taf` in Witango Studio.
- I1 Reopen the Results HTML for the **Get_Sales_Items** action by selecting **Get_Sales_Items** and clicking the **Results HTML** icon on the **Attributes** toolbar.

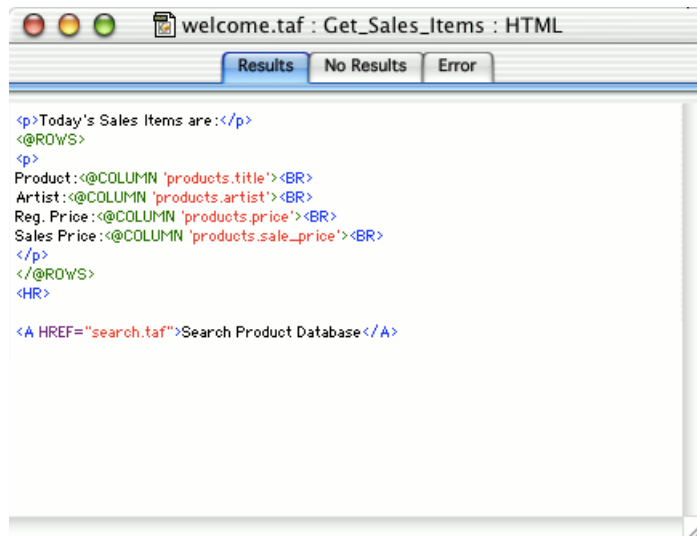


Place your cursor at the bottom of the page after the `<HR>`.

- 12 In the **Standard Snippets** folder, open the **HTML** folder, and double-click or drag the **Hyperlink** snippet into the Results HTML window for the **Get_Sales_Items** action.

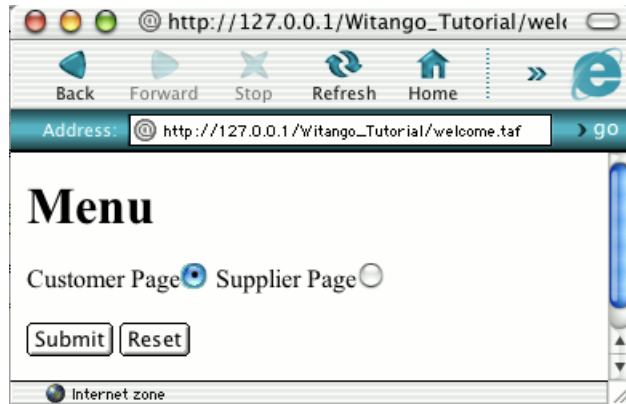


- 13 Type **Search Product Database** between the **<A>** tags. This is the highlighted text the user sees. Type **search.taf** between the quotations after **HREF=**, as the target of the link.

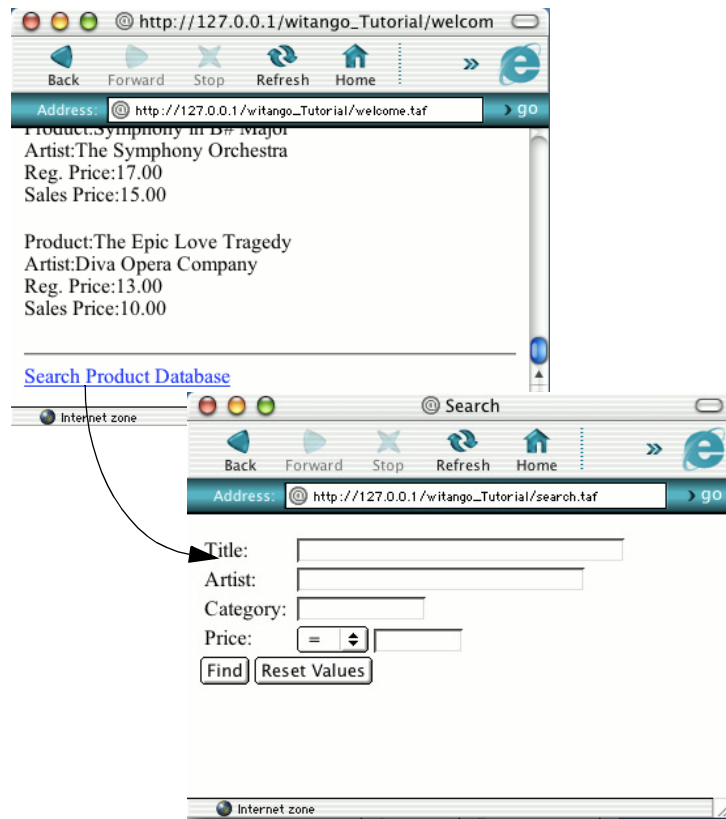


For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 14** Save the file, return to the web browser, and execute `welcome.taf` using the appropriate URL. Select **Customer Page** and click **Submit**.



- 15** Click **Search Product Database** to search the database.



Now that your application file is working, you may check in with the Boss.

LESSON C-2

Setting the Number of Matching Records Returned

Purpose

To set the number of matching records to return on the Record List page.

Context

The Search Builder has many features and functions that you can set to improve or enhance your search functionality. In this lesson, you learn how to set the number of matching records to return to the user at one time.

In some cases, the list of matching records could be lengthy. For example, if you searched for the last name “Smith” in a phone directory database, hundreds of matching records would be found. To provide all records to the user at one time may impose an unreasonably long delay.

In the Search Builder, you can set a reasonable number of records to return to the user on the Record List page, as well as decide whether or not to offer *Next* and *Previous* buttons.

Exercise



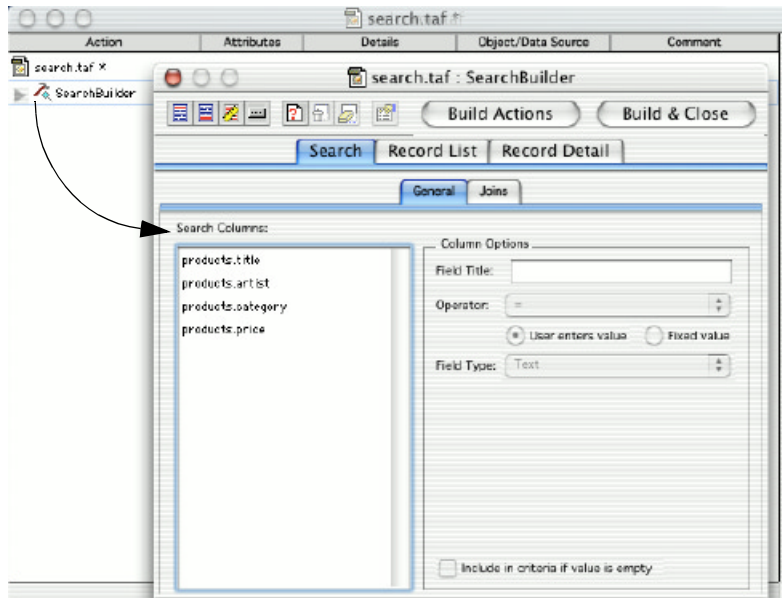
The Boss

The light in your office no longer flickers. Instead, it hums briefly every thirty-seven seconds—and you’ve discovered on the drive home at the end of the day, so do you.

To avoid displaying too many Sale Price records on a single page, the Boss would like you to limit the number of returned records to five, allowing for remaining records to be displayed on subsequent pages

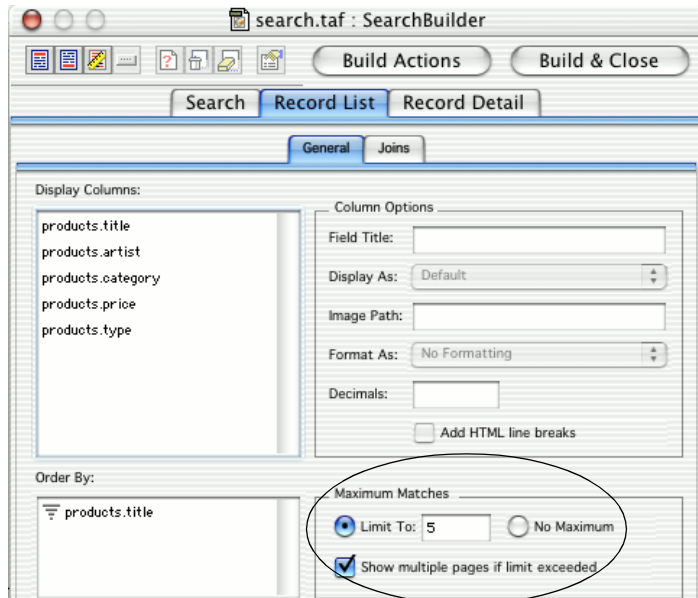
I Return to `search.taf` and double-click the Search Builder to open

it.



- Click the **Record List** tab. In the **Maximum Matches** area, select the **Limit to** radio button. Enter 5 in the **Limit to:** text field.

Now, check the **Show Multiple Pages If Limit Exceeded** check box.





In the *Maximum Matches* section, you have three options available: 1) a set maximum number of matches *without* Next/Previous Matches buttons available if the limit is exceeded; 2) a set maximum number of matches *with* Next/Previous buttons available if the number is exceeded; and 3) no limit to the number of matching records returned on the Record List page.

The default builder setting does not limit the maximum matches returned. You can disable *Show Multiple Pages If Limit Exceeded*, so the user is not presented with Next/Previous buttons in the event there are more than 100 matches.

The *Limit To* option lets you limit the number of records to be returned by the search. Limiting the number to 5 means that only the first five records matching the search criteria will appear. If you were to select the *No Maximum* option, all records matching the search criteria would be retrieved and displayed on the record list page.

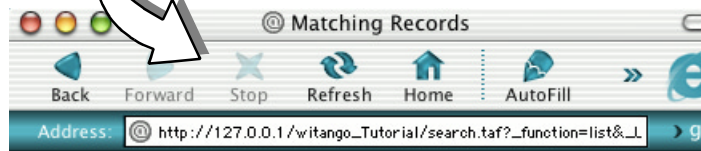
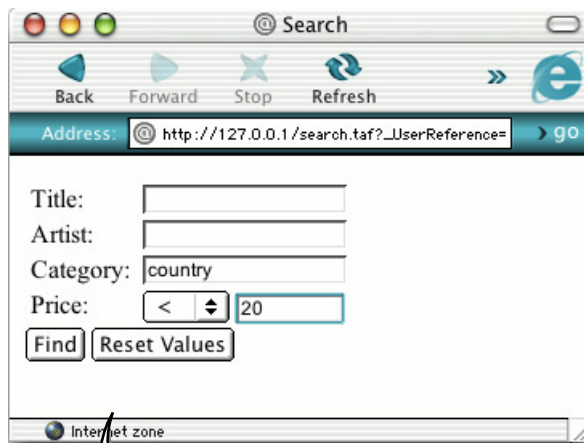
The *Show Multiple Pages If Limit Exceeded* option is available only when you specify a maximum number of matches in the *Limit To* field. Since you have selected this option, a *Next* button will appear on the record list page on the web browser when the number of matching records exceeds the limit entered. Other information to appear will be an indication of the total number of matching records, as well as which ones will be displayed. When the user clicks the Next button, the next group of matching records appears. A *Previous* button appears on record list pages beyond the first one, allowing the user to go backwards in the list of matching records.

- 3 Click **Build Actions** and close the Search Builder window to rebuild all the Witango actions.



For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 4 Save the file, return to the web browser, and reload `search.taf`. Again, search for all Country music CDs under \$20.



Here are the results of your search, tess Duezer

There are 7 matching records.

Displaying matches 1 through 5.

Title	Artist	Category	Price	Type
Cheatin' On Me	Slim Chance	Country	12.00	CD
Honky Tonk Harry	Hot Hands	Country	15.00	CD
My Dog Left And So Did You	Harry	Country	18.00	CD
Still Cheatin'	Backroad Boys	Country	11.00	CD
Stop Your Cheatin'	Slim Chance	Country	14.00	CD

Next 2 Matches



- 5 Click **Next** once to display the next set of matching records.



Here are the results of your search, tess Duezer

There are 7 matching records.

Displaying matches 6 through 7.

Title	Artist	Category	Price	Type
Tumbleweeds and Tears	Loretta Lonesome	Country	14.00	CD
When Will The Cheatin' End?	Slim Chance	Country	17.00	CD

Previous 5 Matches



Note On the second record list page, there are only two additional records (beyond the first five) that matches the search criteria; therefore, a *Next* button does not appear on this page.

Now that your application file is working, you may check in with the Boss.

LESSON C-3

Inserting Records Into the Database

Purpose

To learn the basic functionality of the New Record Builder as you use it to create an application file that can insert records into a specified database.

Context

You use the New Record Builder to build actions that, when executed, display a form allowing users to enter data for a new record, and return a result message after the record is added to the database. On the web page, the user enters the values for the fields in the new record, and clicks Save to save the record. Witango then saves the record to the data source, and returns the HTML response you specify in the New Record Response HTML.

Exercise



The Boss

To celebrate the big sale on the surplus country music CDs, the Boss has invited a number of his best employees to his dude ranch for a good old-fashioned hoedown on the weekend—and you're included! First, however, there is work to be done.

The Boss would like his suppliers to be able to insert new products in the **products** table of the database.

Place the Supplier insert functionality in a new application file.

Q. What Witango action do you drag in to provide an insert form page for the database?

A. A *New Record Builder* action.

I From the **File** menu, choose **New**, or click the **New** icon on the main toolbar.

A blank application file appears.

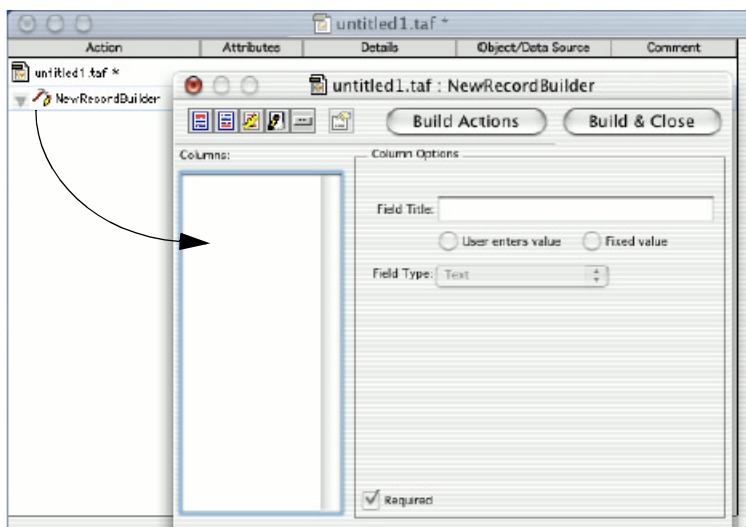


New Witango
Application File

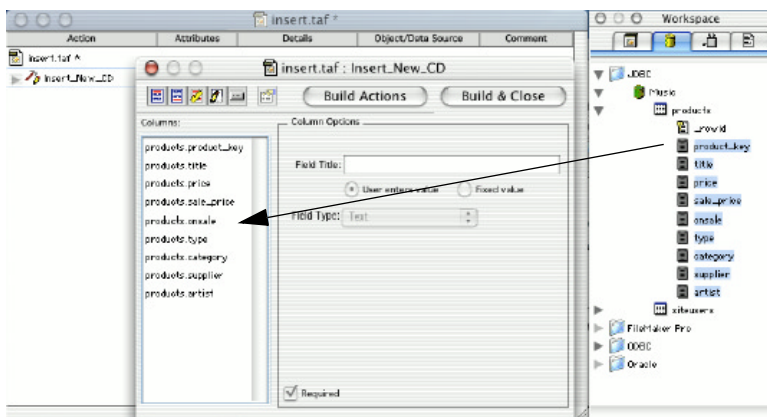


New Record Builder

2 Drag in a **New Record Builder** action.



3 From the Data Sources Workspace, expand the **Music** data source. Drag the **products** table into the **Columns** area of the New Record Builder window to add all of the columns from **products** table.



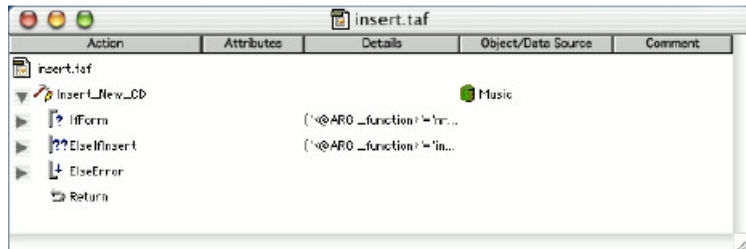
The columns you include in the Columns section are fields the user assigns values to in the new record. You can add columns from only one table, which in this case is *products*. Remember, the order in which the fields appear in the Columns list determines their order on the resulting

new record entry form.

In the Column Options area, you can configure each field 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 now, leave the options in this area at their default settings.

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

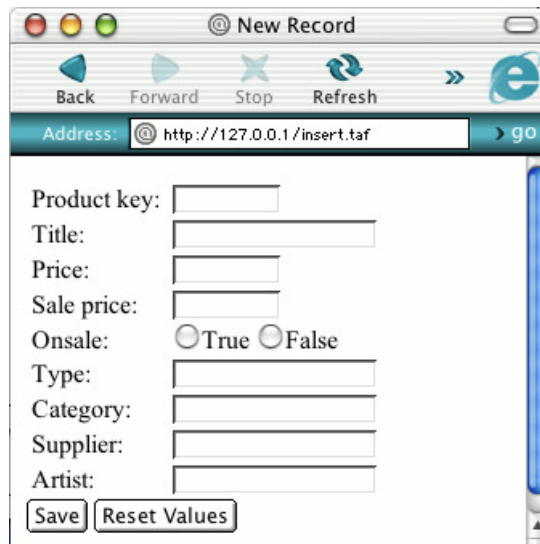
- 4 Click on **New Record Builder** to highlight it. Rename it to **Insert New CD**.



Build Actions

- 5 Click **Build Actions** and close the window. Save the file as `insert.taf`.

Return to the web browser and access your file using the appropriate URL. Use the URL that you used in the previous lessons, substituting `insert.taf` for the application file.



- 6 Type the following information in the appropriate fields on the form:

Product key = 12345 **Type** = CD
Title = TestTitle **Category** = Rock
Price = 10.00 **Supplier** = TestLabel
Sale price = 9.00 **Artist** = TestBand
Onsale = false

The screenshot shows a web browser window titled "New Record". The address bar shows "http://127.0.0.1/insert.taf". The form contains the following fields and values:

- Product key: 123456
- Title: Test Title
- Price: 10.00
- Sale price: 9.00
- Onsale: ☐ True ☒ False
- Type: CD
- Category: Rock
- Supplier: TestLabel
- Artist: Test Band

At the bottom of the form are two buttons: "Save" and "Reset Values".

- 7 Click **Save**.

The screenshot shows a web browser window titled "Record Added". The address bar shows "http://127.0.0.1/insert.taf?_function=ii". The main content area displays:

Record Added

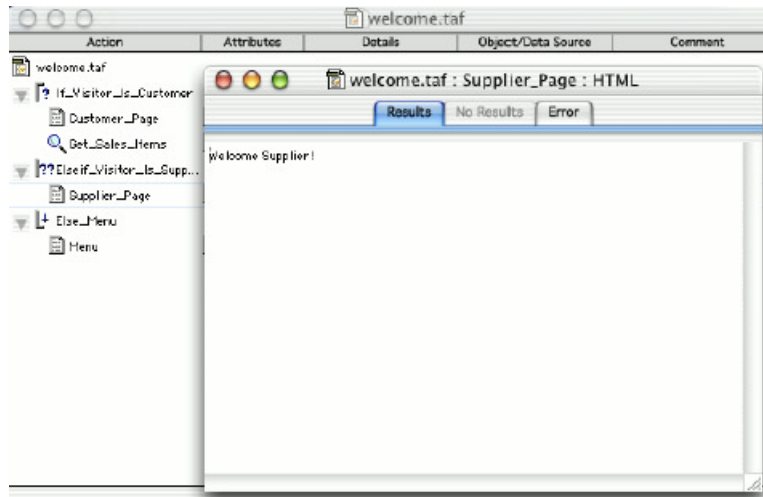
The record was added successfully.

[Add another record](#)

- 8 Use your web browser to access search.taf and check that the new record is now included in the Music database by searching for it.

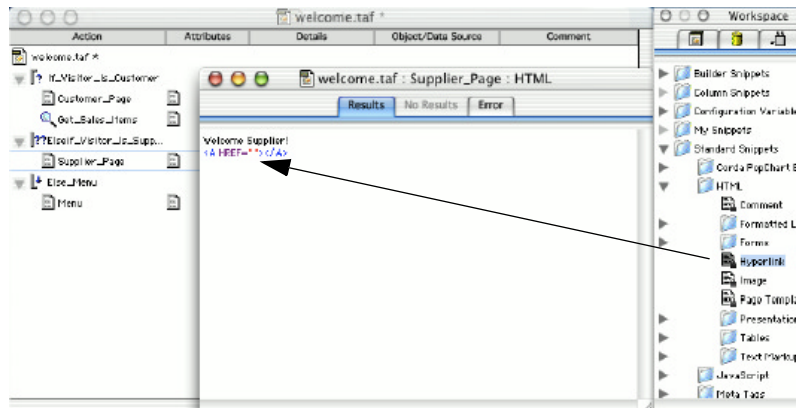
Now, return to the welcome.taf file and add a link to insert.taf on the Supplier Page.

- 9 Reopen **Supplier_Page** and place your cursor on a new line under Welcome Supplier.

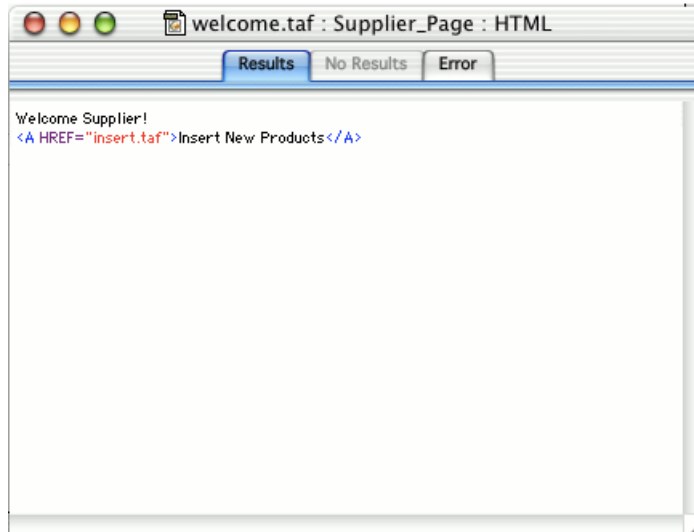


- 10 In the Snippets Workspace, expand the **HTML** folder within the **Standard Snippets** folder and double-click or drag in a **Hyperlink** snippet.

In the **Supplier_Page** action, type Insert New Product between the tags. This is the highlighted text the user sees.



- 11 Type `insert.taf` between the quotation marks after `HREF=` to specify the target of the link.



For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 12 Save the file, return to the web browser, and reload `welcome.taf`.
- 13 Click **Supplier Page**, then click **Submit**.

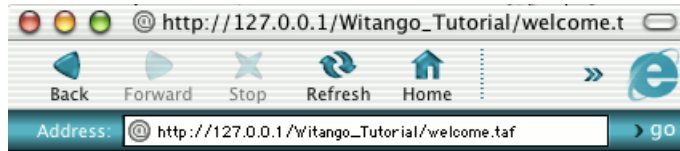


Menu

Customer Page ☐ Supplier Page ☒



- 14** Click the **Insert New Product** link and, if necessary, click **Reset Values**.



Welcome Supplier! [Insert New Products](#)

Product key:

Title:

Price:

Sale price:

Onsale: ☐ True ☐ False

Type:

Category:

Supplier:

Artist:

Now that your application file is working, you may check in with the Boss.

LESSON C-4

Controlling User Input: Creating a Drop-down Menu, Radio Buttons & Setting Required Fields

Purpose

To exercise some control over the quality and completeness of user entered data collected via the insert form by making several modifications including:

- setting some form fields as *required*;
- replacing the Onsale form field with radio buttons;
- replacing the Category form field with a Categories drop-down menu.

Context

This lesson demonstrates how easy it is to make three significant data control changes to the insert form using the New Record Builder (now now renamed to Insert New CD).

The first change is setting required fields so that users cannot leave them blank. This simply involves checking a box in the Insert New CD window. Once required fields are set, Witango checks to see whether the user entered values for these fields. If those fields were left empty, Witango returns a message informing the user which required fields were left empty.

The second change is to use Radio buttons for the Onsale field. Radio buttons are useful as a form of data validation as they ensure that the user enters a valid value for the particular database field or column.

The third change is to add a drop-down menu for the product categories. Drop-down menus can also be used as a form of data validation to ensure that a user can only make a valid data choice but without the page space sacrifice associated with radio buttons.

Exercise



The Boss slaps you on the back for a job well done. He was also quite pleased with the way you handled his prize bull on the weekend.

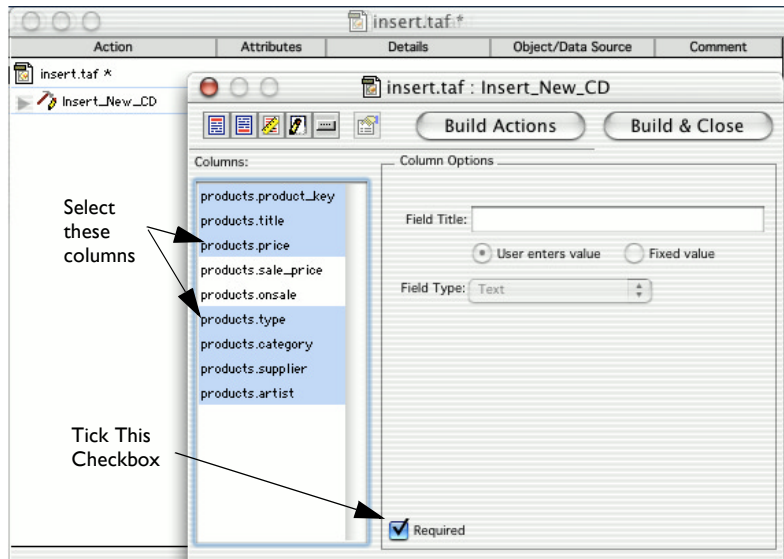
Now he wants you to make some improvements to your `insert.taf` file. He's noticed that some visitors to the web site are not entering all the information that is required and that other visitors have entered incorrect or inconsistent information to the Music database. He wants you to prevent this from happening.

To ensure that values must be entered by the user before a record can be inserted you will need to set the following fields as “required”: Product Key, Title, Price, Type, Category, Supplier and Artist.

The Onsale field only has two possible values - true or false - as an item is either “on sale” or “not on sale”. Any other value would be invalid and should not be entered into the field. Additionally, the valid values for this field are mutually exclusive, that is, only one of the values can be valid at any time. To ensure that the user must choose a single value from an existing list of valid values, rather than making up their own data you will turn the Onsale field into radio buttons.

Although the Category field will function no matter what is value is entered into it, you really only want users to enter one of the category names that you have chosen. To force the user to choose from an existing list of approved categories in the system, rather than making up their own category you will build a drop-down menu for the music categories.

- 1 Return to the `insert.taf` and reopen **Insert New CD** by double-clicking on it.
- 2 Select all columns in the **Columns** area except for **products.sale_price** and **products.onsale** by holding down the CTRL key and selecting the desired columns. Click the **Required** check box at the bottom of the **Column Options** area of the Insert New CD window.

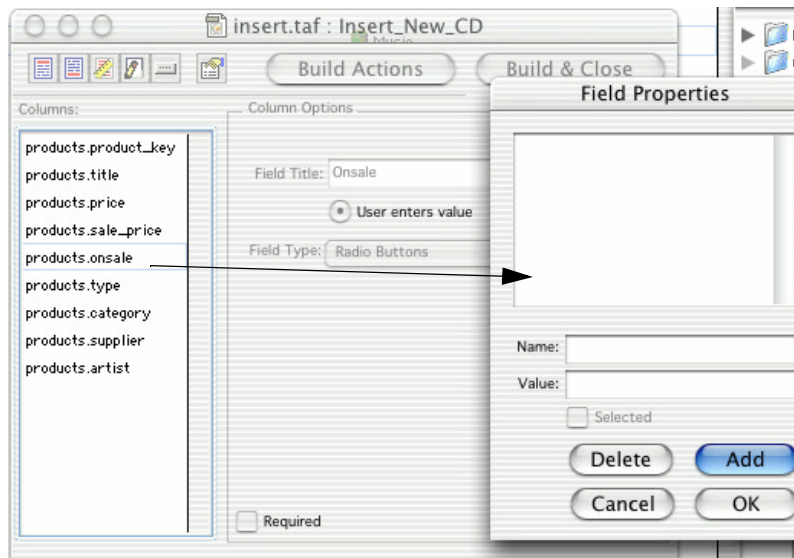




Checking the *Required* check box prevents the user from adding the new record without specifying a value for this column. Instead, if the supplier tries to add a new record without indicating such standard things as the product key, title, price, type, category, artist, and name of the supplier, they will get an error message.

- 3 Select **products.onsale** and , in the **Field Type**, choose Radio Button.

The Field Properties dialog box opens.



The **Music** database is expecting to use the values **True** and **False** in the **Onsale** field to indicated whether an item is on sale or not. This is fine for the database but meaningless for a human being who would expect something more like **Yes** and **No** to be used. To accommodate for this need Witango builders allow the display name to be different to the value that is being passed to the database. To demonstrate this:

- 4 Add the following to the **Field Properties** window:

Name	Value
Yes	True

Name	Value
No	False

Click the **New** button. In the **Name** field type `Yes` and in the **Value** field, type `True` but do not tick the **Selected** checkbox.

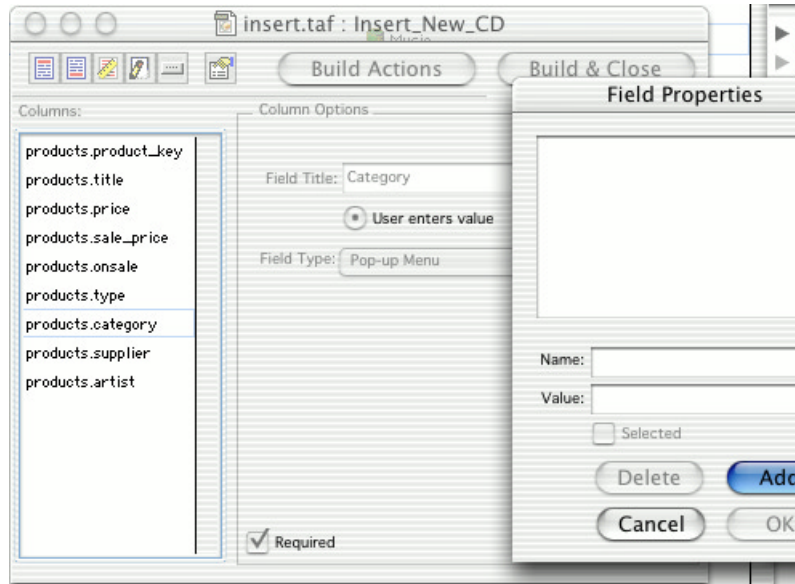
Click the **New** button. Type `No` in the **Name** field and `False` in the **Value** field and click the **Selected** checkbox to make it the default value.

Click **OK** when done.

The image shows a 'Field Properties' dialog box. It contains a list with two entries: 'Yes' with value '1' and 'No' with value '0'. The 'No' entry is selected. Below the list are input fields for 'Name' (containing 'No') and 'Value' (containing '0'). There is a 'Selected' checkbox which is currently unchecked. At the bottom are four buttons: 'Delete', 'Add', 'Cancel', and 'OK'.

- 5 Select **products.category** and, in the **Field Type**, choose Drop-down list.

The Field Properties dialog box opens.



- 6 Add the following categories to the **Field Properties** window:
Blues, Classical, Country, Jazz, Polka, Pop, Rock and R&B.

Click the **New** button. In the **Name** and **Value** fields, type **Rock**, and tick the **Selected** checkbox. This makes **Rock** the default value in the dropdown list. .



Click the **New** button and repeat this for the next music category but do not click the **Selected** checkbox.

When all categories are entered, click **OK**.

- 7 Click **Build Actions** in the Insert New CD window to rebuild your Witango actions.
- 8 Save the file, return to the web browser, and reload `insert.taf`



For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

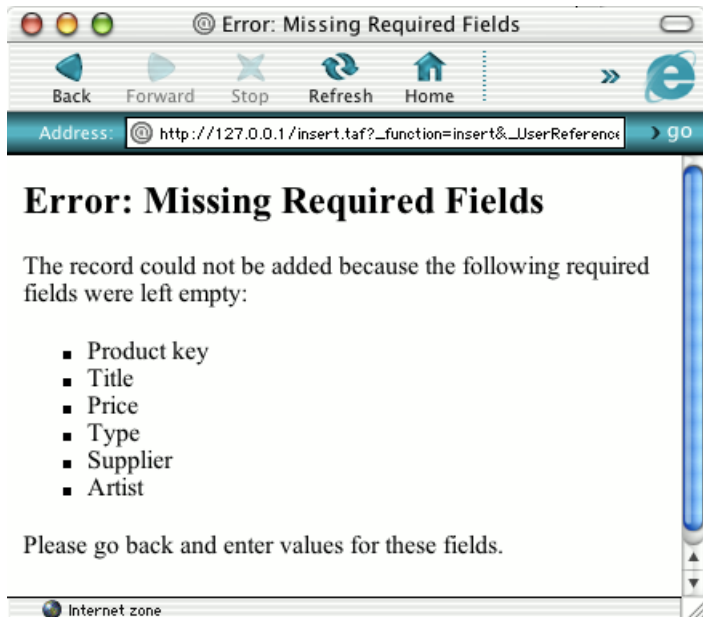
Note The **Category** field is now a drop-down menu listing all the options you entered in the **Insert New CD** window and showing **Rock** as the default value.

The **Onsale** field is now two radio buttons, each one displaying one of the names (**Yes/No**) that you listed in the **Insert New CD** window but passing the different values (**True/False**) to the Music database. **No/False** is the default.

Finally, the fields that you set as *Required* are now shown in bold type on the form to indicate this.

- 9 Now, select an option other than **Rock** from the new **Category** drop-down menu. and select **Yes** from the **Onsale** radio buttons.

Leave the required fields empty and click **Save**.



An error message is displayed. Witango Server returns an error message when the required fields are submitted as empty. Witango Server knows which required fields were left blank and lists them in the error message.

Click the *Back* button of the web browser and fill out one of the required fields. The list on the Error page will change accordingly.

Now, go back and enter values for all the required fields. Witango inserts the record and provides a "Record Added" message.

Now that your application file is working, you may check in with the Boss.

LESSON C-5

Creating a Project

Purpose

To create and use Witango projects.

Context

You now have three application files for the music store: `welcome.taf`, `search.taf`, and `insert.taf`. Witango Studio allows you to group related files such as these into a *Witango project*.

A Witango project file contains information about a project you create, including a list of application files, graphic or HTML files, and data sources grouped in this project.

Projects allow you to logically group and work on a set of related application files.

Exercise



The Boss

The Boss is very pleased - company legend has it that he even smiled the day he saw the improved web site!! In recognition of your hard work, your face has appeared on the front page of the company newsletter as *employee of the month*! It is a proud moment for you.

The Boss would like you to create a project called `MusicStore`, incorporating all the files you have created so far.

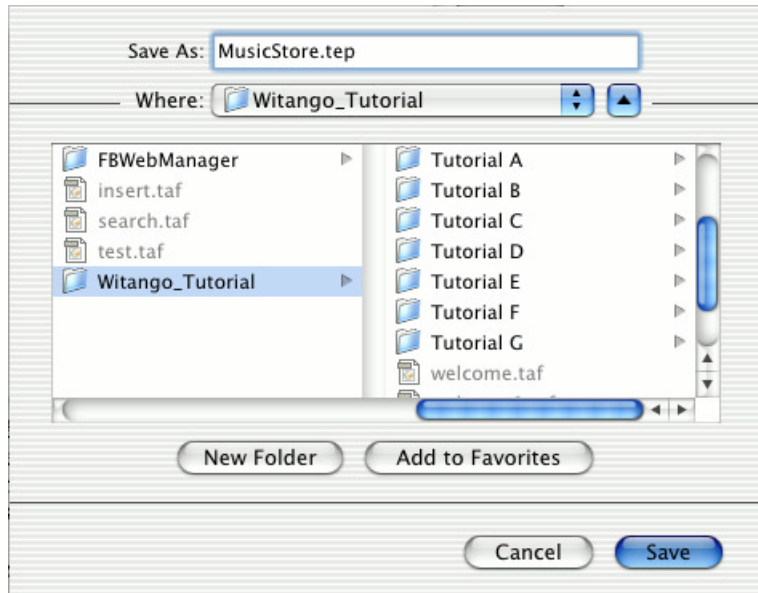
I From the **Project** menu, choose **New**.



Note You cannot create an *untitled* project. When choosing New, Witango prompts you for a file name to save the project under.

For details, see “Your Web Server Document Root” on page 4.

- 2 Navigate to the `Tutorial` folder within the `Witango` folder of the web server document root.



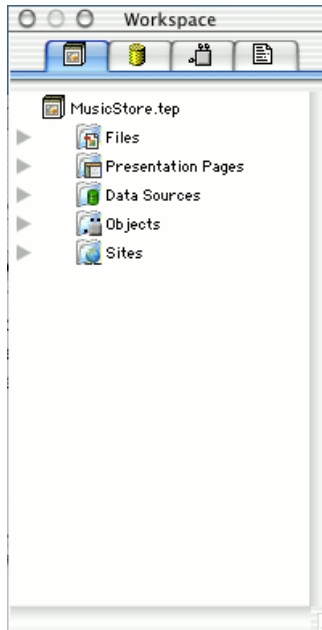
- 3 Name the project file `MusicStore`, and click **Save**.

Witango automatically adds the file extension `.tep`, short for (Wi)Tango Studio Project, to the project file.



Project Workspace

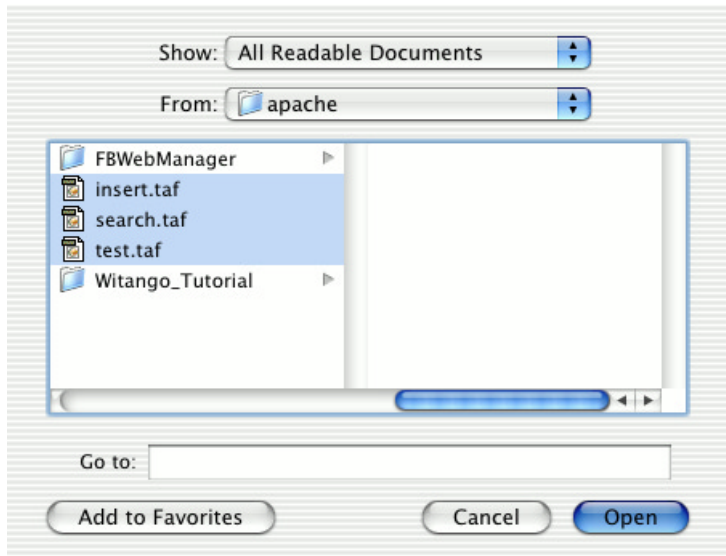
A *Project Workspace* tab is added to the Workspace, and your new MusicStore project is displayed in the Project Workspace. You can work on only one Witango Studio project at a time.



When you create a project, it is not necessary to have any application file open. Also, and you do not need to open the files that will be assigned to the project. If you have any application files open, you will be asked if you would like to add them to the project.

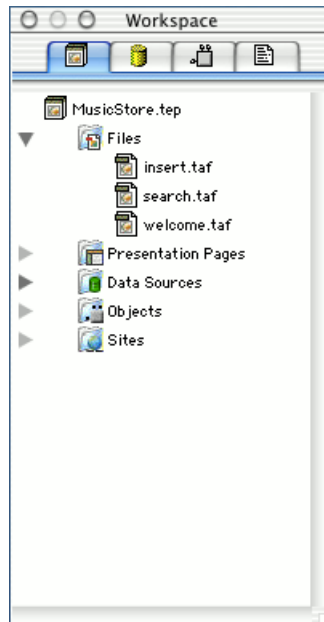
- 4 To add your application files to the **MusicStore** project, choose **Add Files...** from the **Project** menu.

The Add Files into Project dialog box appears.



5 Click **Open**.

The Add Files into Project dialog box closes and `welcome.taf`, `search.taf`, and `insert.taf` now appear as items within the Files folder of the **MusicStore** project.





The *project file* is written to disk whenever you close the project or quit Witango Studio. In the future, you simply open the MusicStore project file and open MusicStore project application files you want to work on from there by selecting them in the Project Workspace, right-clicking and choosing Open from the context-sensitive menu that appears. You remove files from a project in the same manner. Again, the purpose of the MusicStore project is to organize logically a set of MusicStore application files.

Now that your application file is working, you may check in with the Boss.

Collecting Values and Assigning Variables

Collecting, Storing, and Tracking User Information Using an Assign Action

A useful Witango solution saves user information over a number of executions by using a form to collect information, such as the user's name and e-mail address, when someone visits your web site. This information is stored so that it can be used again in subsequent executions by the same user.

In Witango, any piece of user information is passed as a value. The only way in which a value can be remembered and referenced for more than one execution of an application file, however, is to store it as a variable.

There are two lessons in this tutorial:

D-1: Creating a Personalized Welcome Page

D-2: Storing and Tracking User Information

LESSON D-I

Creating a Personalized Welcome Page

Purpose

To create a welcome page for the user who has just logged in successfully.

Context

In the previous tutorials, you learned how to pass and reference values (arguments) from the web browser, and how to retrieve and reference database values. The problem with arguments and database values is that the values only last for one execution of an application file. The values are received or generated by Witango Server during an application file execution and are wiped out or lost at the end of execution when Witango Server returns HTML to the web browser. In Witango terms the *scope* of the variable is too short.

In many cases, it is necessary to keep these values for longer than an application file execution. For example, in a login model, a user's access level retrieved from the database should be remembered by Witango so that throughout the rest of the user's session, Witango can control which application files or functions the user can execute. The user access level must be remembered beyond one file execution. Witango variables are how Witango Server *remembers* values.

For the music web site, you will personalize each page viewed by the customer. This involves two steps: collecting the customer's name and e-mail information (passed to Witango Server as web browser arguments via a form), and assigning that information to Witango user variables so they can be referenced, or called, on multiple pages throughout the site.

Variables are name-value pairs, and can be assigned to one of several scopes. Depending on scope, each variable is stored in different areas of Witango Server's memory (as is the case for request, user, application, domain and system scopes) or saved on the user's machine (as for cookie scope).

This first lesson teaches you how to collect the user's name and e-mail address. The second lesson shows you how to assign variables.



Note If you have not done so already, you will have to complete the lesson on “Creating a Data Source” on page 64 in order to complete the lessons in this tutorial.

Exercise



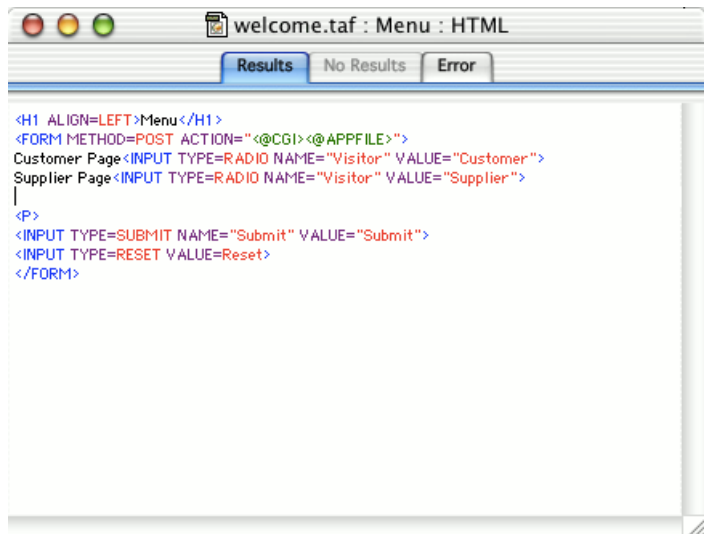
The Boss

If you have your project open, you can just click on `welcome.taf` to open it. See “Creating a Project” on page 114.

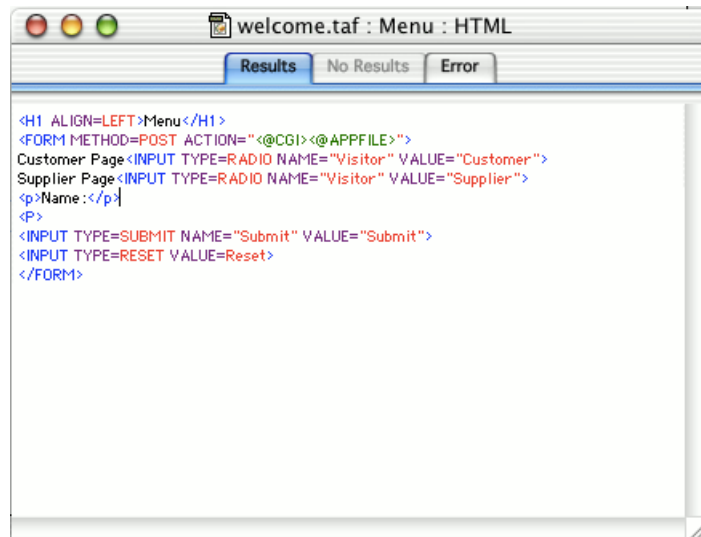
The Boss has moved you into a new, larger office that’s above ground! It has a magnificent view of the carpark and a couch and a coffee table that were obviously retrieved from an era gone by.

As you sit down on the couch the Boss walks in and says, “I didn’t give you this couch so that you could go to sleep on it. I’d like you to create a file that displays a personalized greeting using the customer’s name. Have the customer enter his or her name and e-mail address on the form.”

- 1 Open `welcome.taf`, and double-click the **Menu Results** action to open it.
- 2 Position your cursor between the radio button tags and the `<P>` tag that precedes the Submit and Reset button tags.



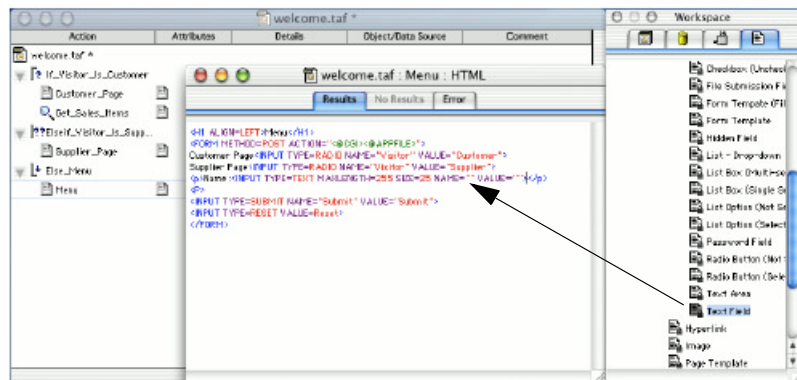
- 3 Type `<P>Name: </P>` and press Enter. Make sure you leave a space after the colon.



Q. What is the appropriate Form snippet for creating a form field for Name : ?

A. A *Text Field snippet*.

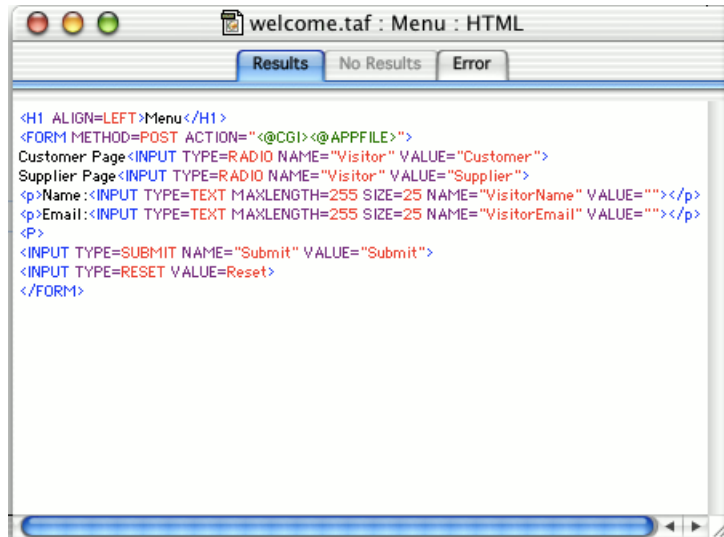
- 4 In the Snippets Workspace, open the **HTML** folder, and the **Forms** folder. Double-click or drag in a **Text Field** snippet.



- 5 In the Text Field form tag, fill in the NAME= attribute by typing VisitorName between the quotation marks.
- 6 Highlight and copy the Name : form field line of code you just created. Place your cursor to the right of the `</P>` tag at the end of the

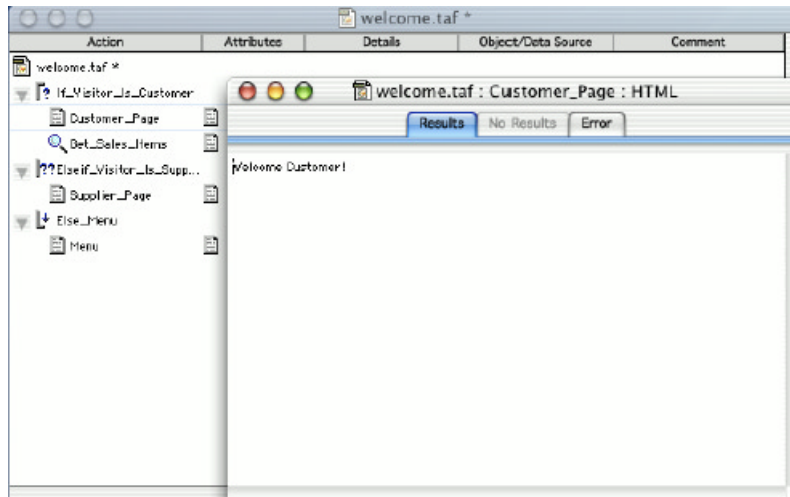
Name : form field line of code and press Enter. Paste the Name : form field tag line of code to the blank line you just inserted.

- 7 Replace Name : with Email : and also replace NAME=VisitorName with NAME=VisitorEmail).

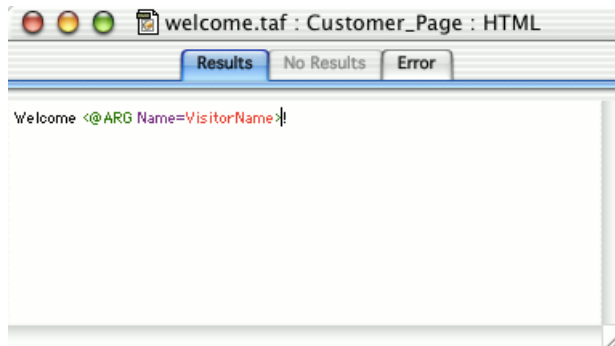


- 8 Close the **Menu** Results HTML window.

- 9 Open **Customer_Page**.



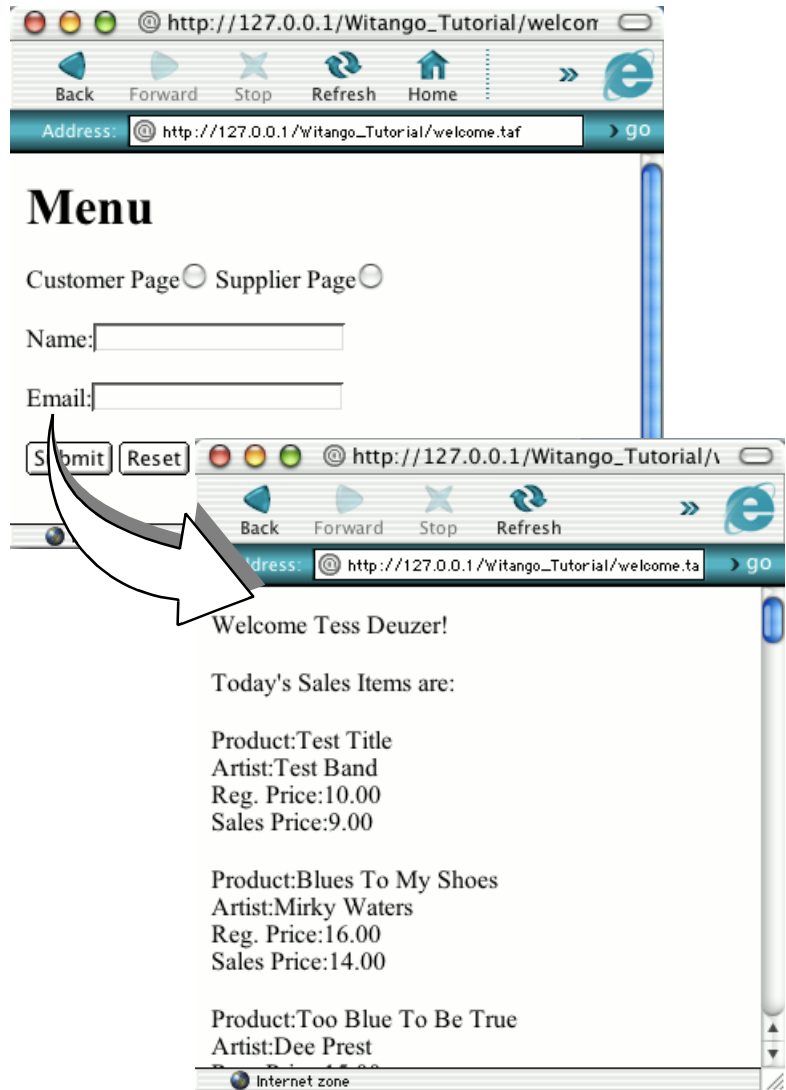
- I0** Replace `Customer` in `Welcome Customer!` with `<@ARG Name=VisitorName>.`



Recall that the `<@ARG>` meta tag references the value of `NAME`. Specifically, it passes the value of the argument `VisitorName`, which is equal to the name that the user enters in the `Name` field on the form.

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- I1** Close the **Customer_Page**. Save `welcome.taf`. Return to the web browser, and reload `welcome.taf`.
- I2** Test `welcome.taf` by selecting **Customer Page** in the Menu, and type some test data e.g. `Tess Deuzer` in the **Name** field and `tessd@abcd.com` in the **Email** field.

13 Click Submit to see the results.

Before you added the *Name* form field, the user could choose to be identified as either customer or supplier by clicking the appropriate radio button. However, the only actual knowledge you had regarding the user's identity was the IP address, which identifies the computer they are using. The problem with this is that there could be a number of different people using the same computer.

There are two benefits to allowing the user to enter their name in the

Name field on the form. First, you can greet the user personally when the name value is passed to the Customer Page. Secondly, you may need to use the name to allow or restrict database queries performed by that user. In order to do this, though, you need to assign a variable to the name value. You will do this next.

Now that your application file is working, you may check in with the Boss.

LESSON D-2

Storing and Tracking User Information

Purpose

To learn how to assign and reference Witango variables.

Context

This lesson teaches you how to use the customer information you collected in Lesson D-1 (name and e-mail) to assign them as variables that can be referenced on multiple pages.

Exercise



The Boss



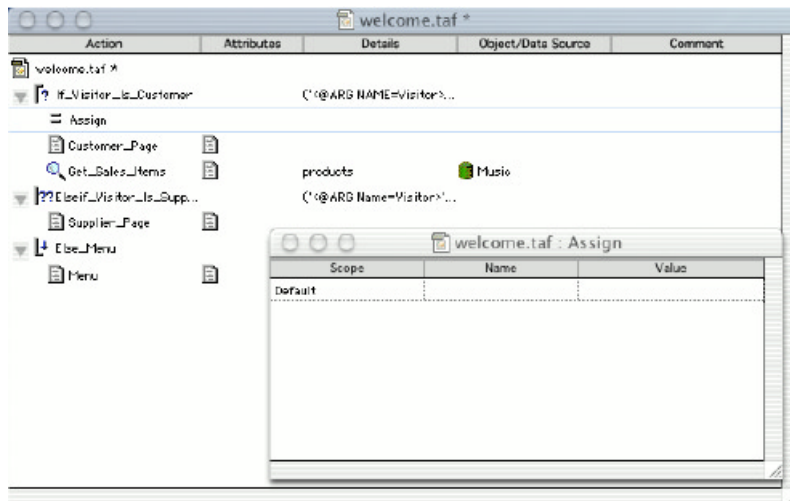
Assign action

You wake up the Boss — he has been sleeping on your couch — and you ask him what he wants you to do next. You need to set up the user's name and e-mail address as variables so that they can be used throughout the application file execution.

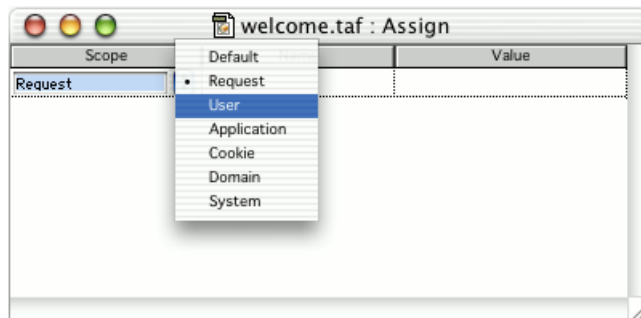
Q. What kind of Witango action do you use to assign a variable to the customer name and e-mail address?

A. An *Assign* action.

I Open `welcome.taf`. Drag an **Assign** action into **If_Visitor_is_Customer**, placing it above **Customer_Page**.



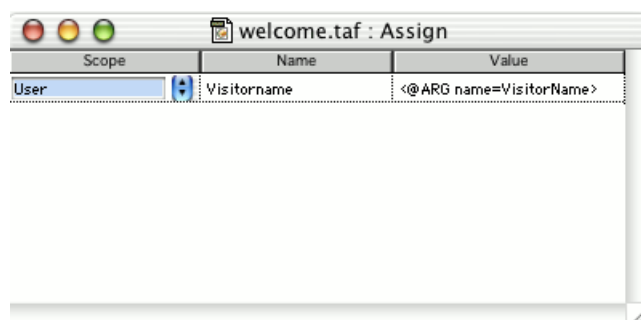
- 2 Double-click on the **Scope** field, and from the drop-down menu that appears, set **Scope** equal to **User** for the first variable.



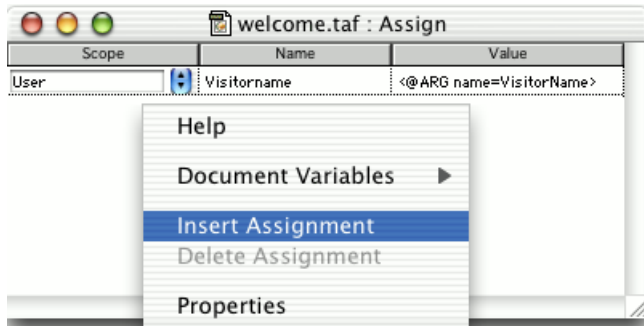
An *Assign* action is used to assign variables, such as the user's name and e-mail address. As you know, variables are placeholders in memory that you can assign a value to. Every variable has a *scope*, which tells Witango if the variable is to be used only for the particular application file execution, for a user, or for a particular domain being served with Witango. In this case, you set the *Scope* to *User* so that you can store and access values associated with a particular user.

Once an assignment has been made to a user variable, its value is available in subsequent actions within the same application file execution (in this case, `welcome.taf`) and in any application files executed afterward. Keep in mind that user variables expire 30 minutes after the user associated with them last accesses Witango (this, however, can be altered by changing the value of the Witango configuration variable `variableTimeout`).

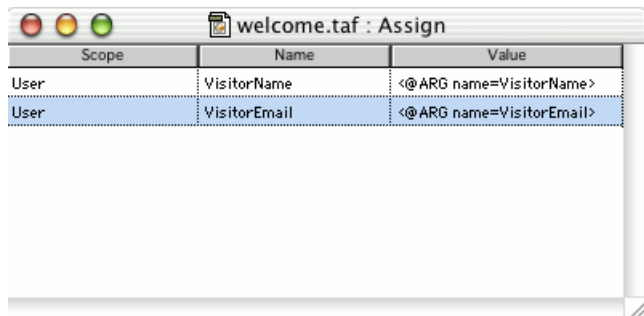
- 3 Type `VisitorName` in the **Name** column to name the new variable. Type `<@ARG NAME=VisitorName>` in the **Value** column to indicate the value of the `VisitorName` variable.



- 4 Right-click within the Assign action window and choose **Insert Assignment** from the context-sensitive menu that appears to input the Email variable.

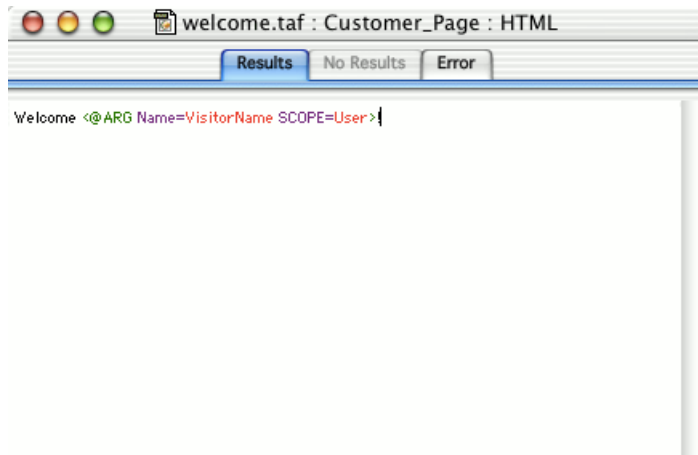


- 5 To set an VisitorEmail variable, once again set **Scope** equal to **User**, and type VisitorEmail in the **Name** column to name the new variable. Now type <@ARG Name=VisitorEmail> into the **Value** column to indicate the value of the VisitorEmail variable.



- 6 Close the **Assign** action, and open the **Customer_Page** Result HTML.

- 7 Replace the `<@ARG>` tag in the welcome message of the **Customer_Page** Results HTML with `<@VAR NAME=VisitorName SCOPE=User>`, as shown:



For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.



Search Builder

- 8 Save and close `welcome.taf`.

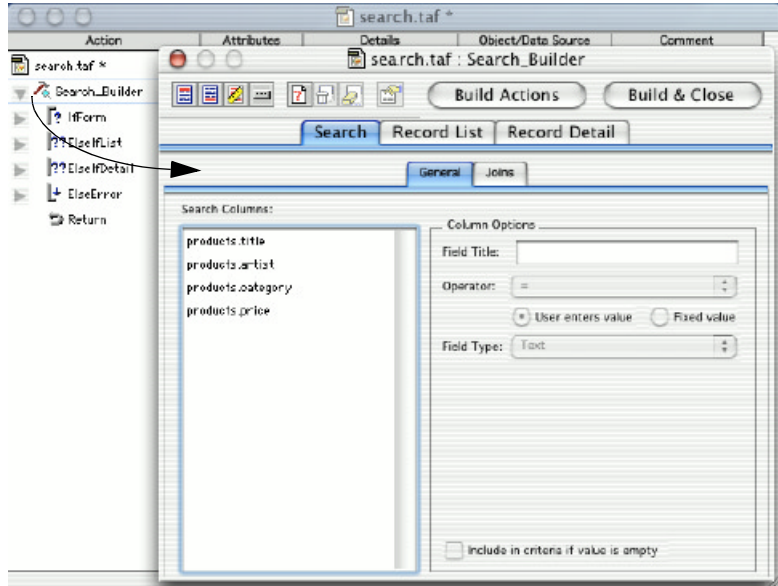
Now, follow these next steps to add the visitor name and message to the Search Form and Record List.

- 9 In Witango Studio, open `search.taf`, and double-click the **Search Builder** action to open it.



Tip If you have your project open, you can just click on `search.taf`

to open it. See “Creating a Project” on page 114.

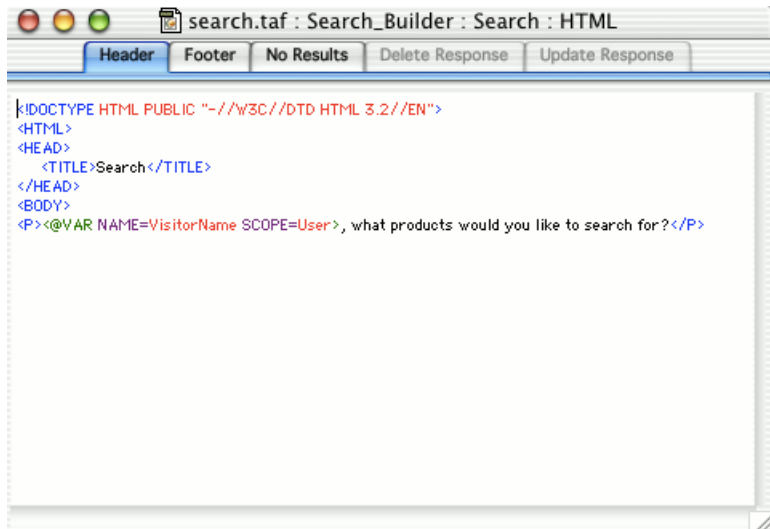


Header HTML

10 Click the **Header HTML** button.

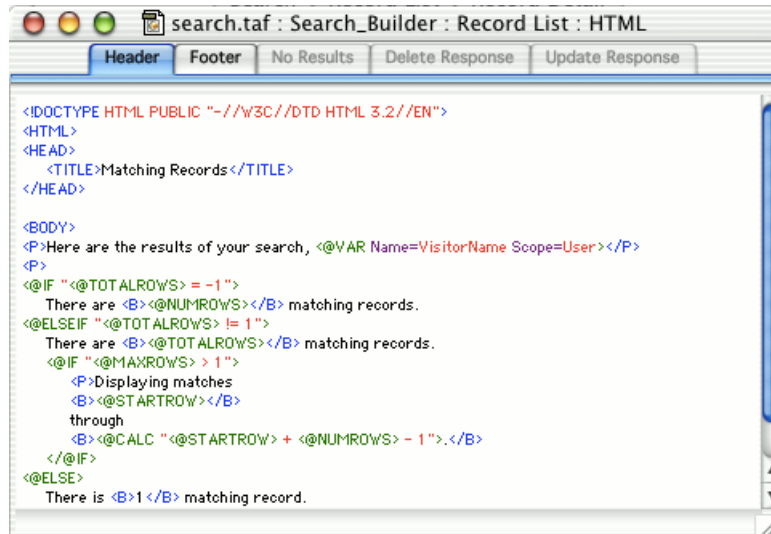
Just below <BODY> type:

```
<P><@VAR NAME=VisitorName SCOPE=User>, what products would you like to search for?</P>
```



Close the Header HTML window.

- 11 Click the **Record List** tab of the Search Builder. Once again, click the **Header HTML** button and, directly after `<BODY>` type
`<P>Here are the results of your search,`
`<@VAR Name=VisitorName Scope=User></P>`



Close the Header HTML window for the Record List.

- 12 Click the **Build Actions** button to redo all the Witango actions, and close the Search Builder.
- 13 Save `search.taf`.

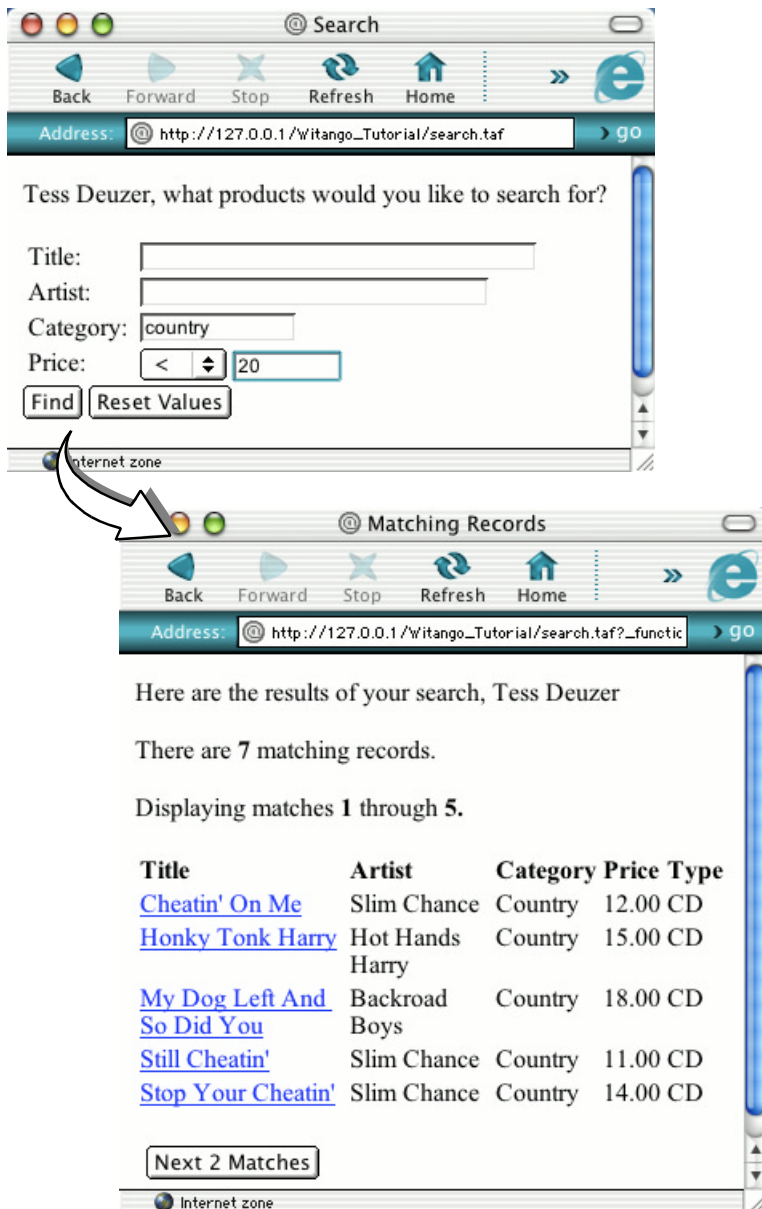
Now you will see, as you navigate through the sequence of web pages, how assigning variables stores the user's information and displays it on each web page.

- 14 Return to the web browser, and reload `welcome.taf`. Fill in the form as follows: **Name** = Tess Deuzer, and **Email** = `tessd@abcd.com`.

- 15 Select the **Customer Page** radio button, and click **Submit** to see the results.



- 16 On the Customer Page in your web browser, click the link to search the product database and perform the search choices as you did before.





The variable Name, “Tess Deuzer”, has been referenced in the three pages after the form page in which its value was entered and after its value was put into the user variable using the Assign action. Each time you execute a new page in the application file, Witango displays the Name value specified by the `<@VAR>` meta tag by referencing it. Essentially, Witango Server *keeps track* of the user’s information throughout the user’s session; when the user has completed the session, this information is no longer stored in Witango Server’s memory (specifically, after the user has not accessed Witango for 30 minutes—this value is set with the `variableTimeout` configuration variable).

Now that your application file is working, you may check in with the Boss.

Enhancing Your Web Site



Creating Additional Enhancements for the Music Web Site

This tutorial provides some examples of dynamic web page design to help enhance both the appearance and function of your Witango web site. The lessons in this tutorial are intended to familiarize you with a few of the many convenient Witango meta tags that can be used to create special functions or design features on your web site.

There are three lessons in this tutorial:

E-1: Displaying the Current Time and Date

E-2: E-mailing Sales Items to Customers

E-3: Using the `<@INCLUDE>` Meta Tag to Enhance the Menu Page

LESSON E-1

Displaying the Current Time and Date

Purpose

To display the time and date automatically on the Menu Page each time the user logs in.

Context

The `<@CURRENTTIMESTAMP>` meta tag, used in this lesson, indicates the current time and date when the user logs in and sees the main Menu Page.

Exercise



The Boss is quite impressed with your progress, and has taken to slapping you on the back so often that you automatically flinch whenever he gets to within a foot of you.

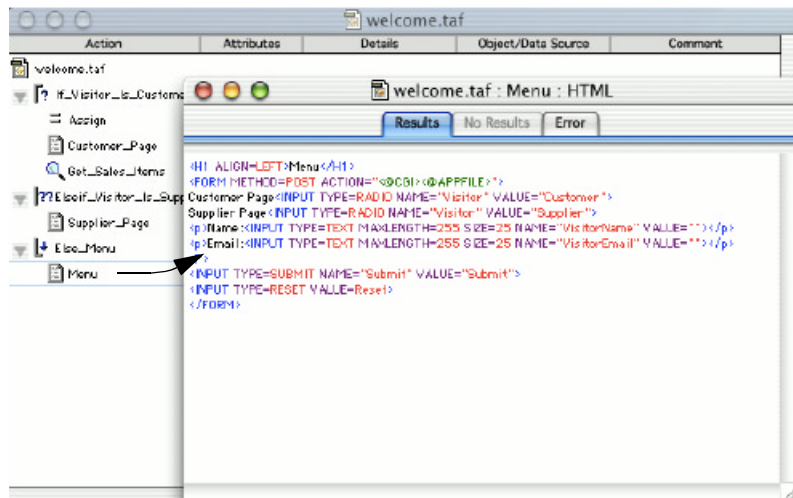
He says, “I’d like our visitors to see the current time and date when they log in. This should help to give the impression that we are keeping our web site current.”



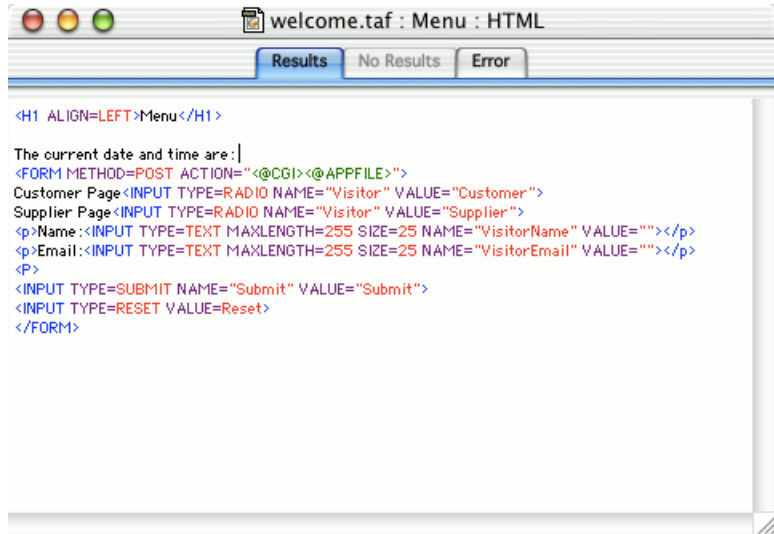
Note If you have not done so already, you must complete the first lesson on “Creating a Data Source” on page 64 in order to complete the lessons in this tutorial.

If you have your project open, you can just click on the `welcome.taf` to open it. See “Creating a Project” on page 114.

- I Open `welcome.taf`, and double-click **Menu** to open it.

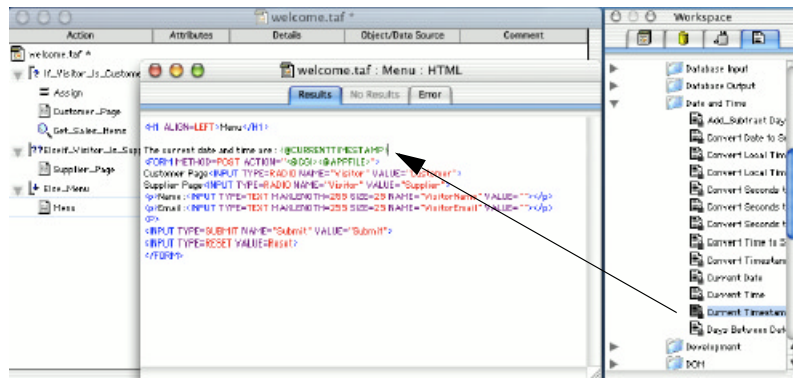


- 2 Leave a space under the main heading, and type The current date and time are:, followed by a space.



- 3 In the Snippets Workspace, open the **Date and Time** folder within the **Meta Tags** snippets folder.

Double-click or drag in the **Current Timestamp** meta tag snippet.

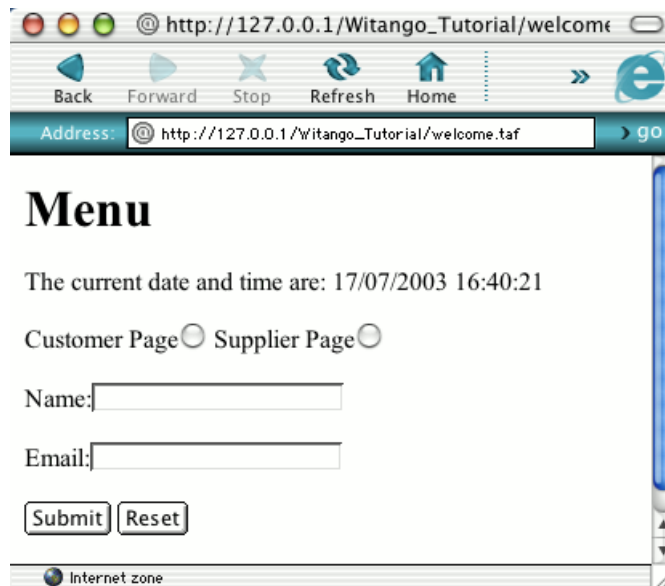


The `<@CURRENTTIMESTAMP>` meta tag displays the date and time together. Unless otherwise specified, the date and time values returned by this meta tag reflect the setting of the clock on the computer where Witango Server is installed.

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 4 Save the file, return to the web browser, and reload `welcome.taf`.

The Menu Page that you see in your web browser now has a current date and time. Reloading the page updates the current date and time.



You can change the way that the date and time are formatted by using different formatting codes in the `<@CURRENTTIMESTAMP>` meta tag. You can add a `FORMAT=` attribute to that meta tag, and put various formatting codes in the quotes.

For example:

In the current webpage you didn't specify any formatting codes so the default was used resulting in "02/21/03 15:25:04".

```
<@CURRENTTIMESTAMP FORMAT="%m/%d/%Y %H:%M:%S">
```

Date
Time

If you wanted to format the above date and time as "Thursday, July 29, 1999, 5:47 PM", you would add the following to your application file:

```
<@CURRENTTIMESTAMP FORMAT="%A, %B %d, %Y, %I:%M %p">
```

Date
Time

Now that your application file is working, you may check in with the Boss.

LESSON E-2

E-mailing Sales Items to Customer

Purpose

To set up a Mail action that sends a list of sales items to the customer.

Context

A *Mail action* is beneficial for many reasons; for the AB CD's site, it can provide the customer with a record of information that he or she can refer to when not currently viewing your site. The Mail action sends out electronic mail using the Simple Mail Transfer Protocol (*SMTP*).

SMTP is the main protocol used to send e-mail on the Internet. When the Mail action is executed, Witango Server connects to the SMTP server. Witango then sends the e-mail message to the specified recipient, which in this case is determined by a variable that contains the e-mail address of the user.

You can send an e-mail message about any subject to anyone with an e-mail address. You simply add the Mail action to an application file, prepare your message, and execute the file to send the message. In this lesson, the message will provide the customer with a list of CDs that are on sale.



Note To create an e-mail message when your application file is executed, you need a mail server that can send e-mail somewhere accessible to your site. You set this up by doing one of the following:

- a) During the installation of Witango, setting your mail server appropriately when the installer asks for it.
- b) Setting the `mailServer` and `mailPort` configuration variables.

If you do not have a mail server set up, you can still follow this lesson, but Witango Server will not generate an e-mail when the application file is executed.

Exercise



The Boss

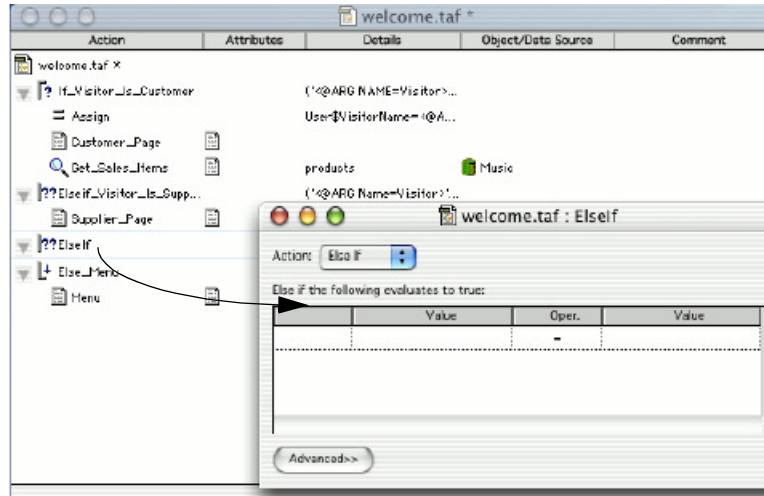


Elself action

While showing the Boss the current date displayed on the form, you slap him on the back and say, “So what d’ya think, big guy?” No response. Wait...—Yes! The Boss smiles, and says, “Well done.” In your nervous excitement, you slap him once again.

The Boss would like you to set up an e-mail response, at the customer’s request, to provide a list of items that are on sale.

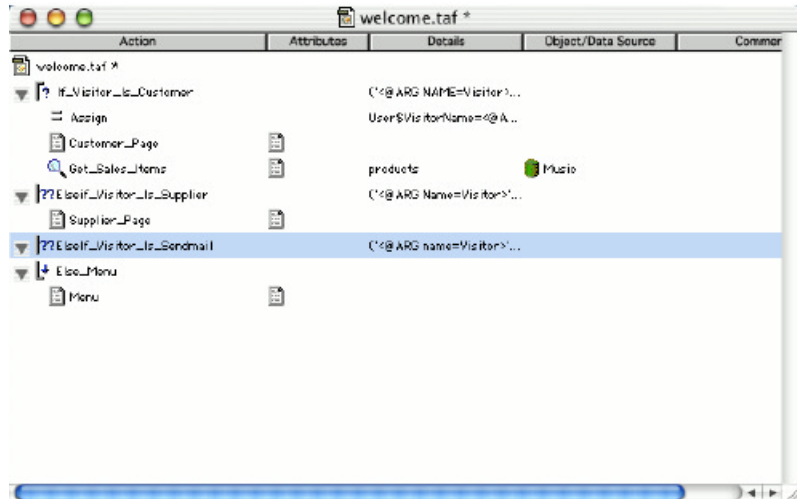
- 1 Reopen `welcome.taf`. Add another condition section by dragging an **Elself** action into the file, placing it above the **Else_Menu** action.



- 2 In the **Value** column of the Elself window, type `<@ARG Name=Visitor>`, which is the meta tag you used as your navigation argument. In the second **Value** column, type `Sendmail`.

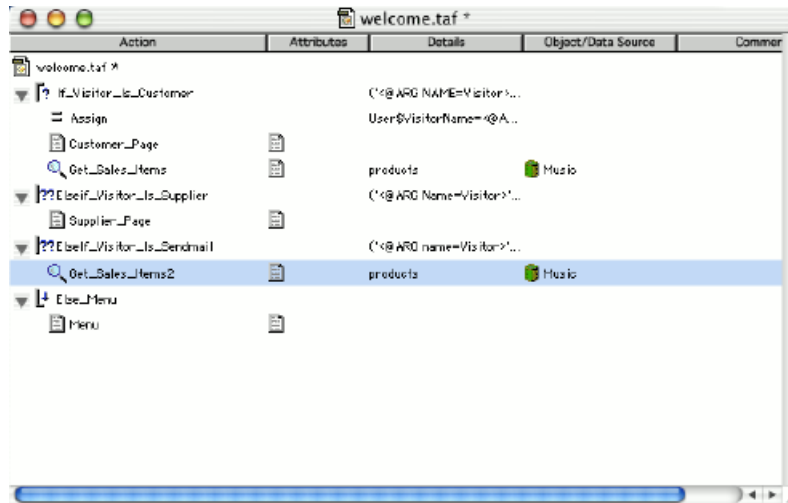


3 Close the Elself window and rename the **Elself** action to **Elself_Visitor_is_Sendmail**.



Now you need to copy the **Get_Sales_Items** action from the **If_Visitor_is_Customer** action, so that you can repeat the search of the sales items, calling up the most current list before sending the e-mail.

- 4 Select **Get_Sales_Items** and copy it by holding down the **OPTION** key and dragging it into the new **Elself_Visitor_is_Sendmail** action.



- 5 Double-click the Results HTML for **Get_Sales_Items2**, or right-click **Get_Sales_Items2** and choose Results HTML from the context-sensitive menu that appears.

The Results HTML editing window appears. Select the HTML, right-click, and choose **Delete** from the context-sensitive menu that appears to delete the contents of the window.



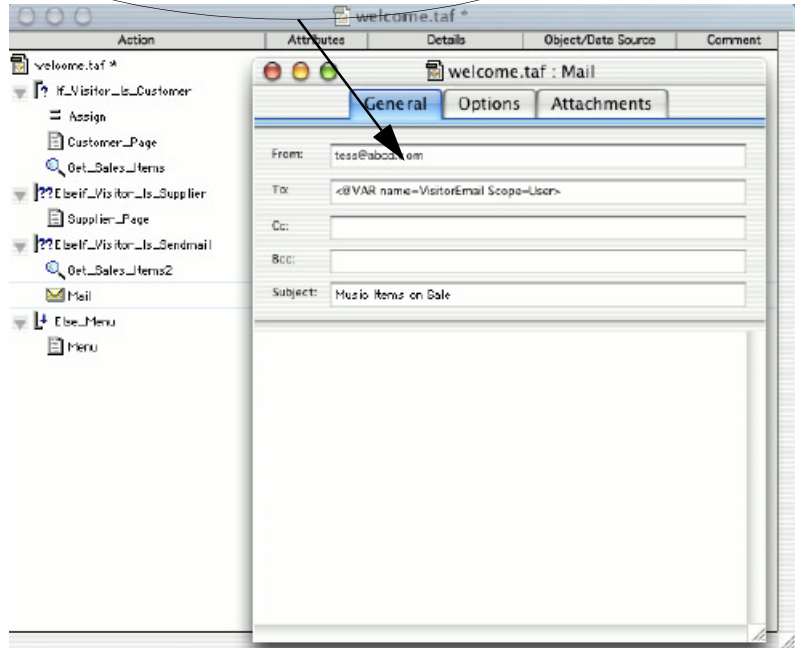
Mail action

- 6 Drag a **Mail** action below **Get_Sale_Items2**.

The Mail action dialog box appears.

7 Fill in the open **Mail** action dialog box as follows:

From: tessd@abcd.com
To: <@VAR Name=VisitorEmail Scope=User>

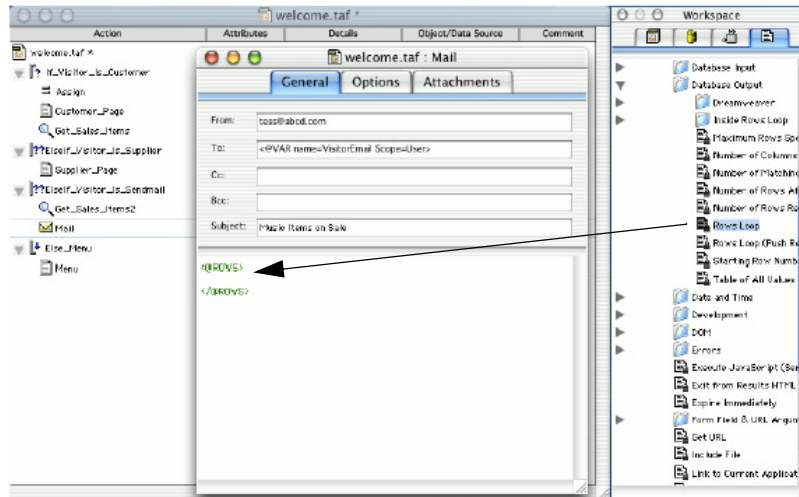


Now, you need to reference the Search action's results by using a meta tag that allows you to use database values.

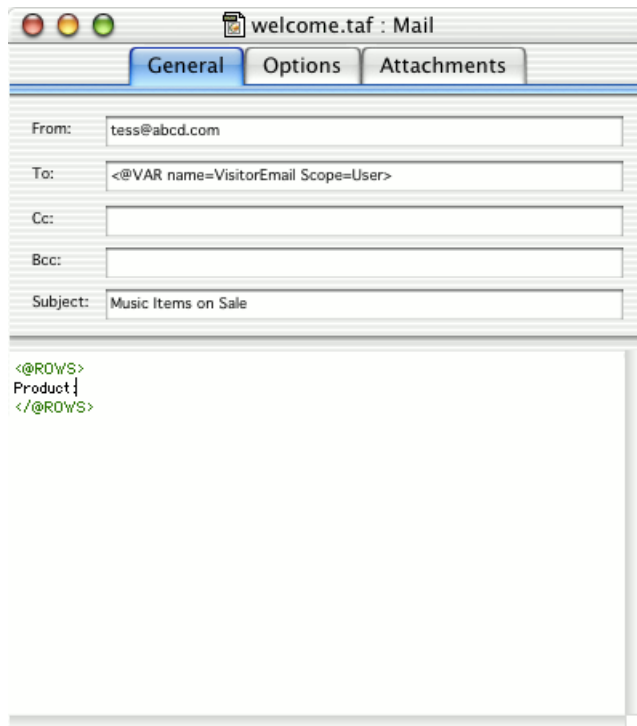
Q. Which Witango meta tag must you use in this case?

A. A *Rows Loop* (<@ROWS>) meta tag.

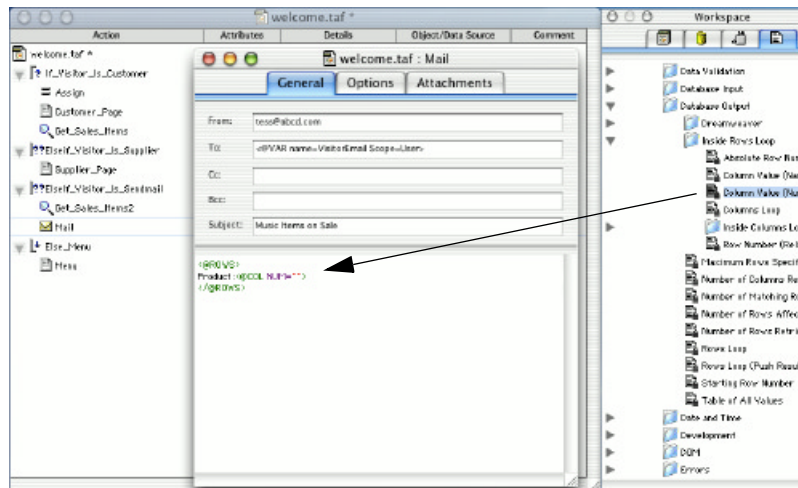
- 8 In the Snippets Workspace, open the **Database Output** folder within the **Meta Tags** snippets folder. Insert a **Rows Loop** meta tag into the **message** area of the Mail action window.



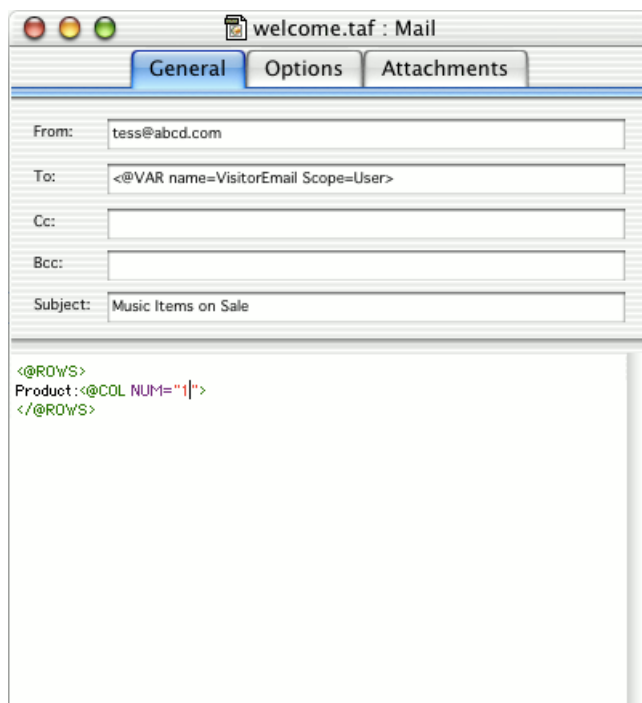
- 9 Between `<@ROWS>` and `</@ROWS>`, type `Product :`, and leave a space.



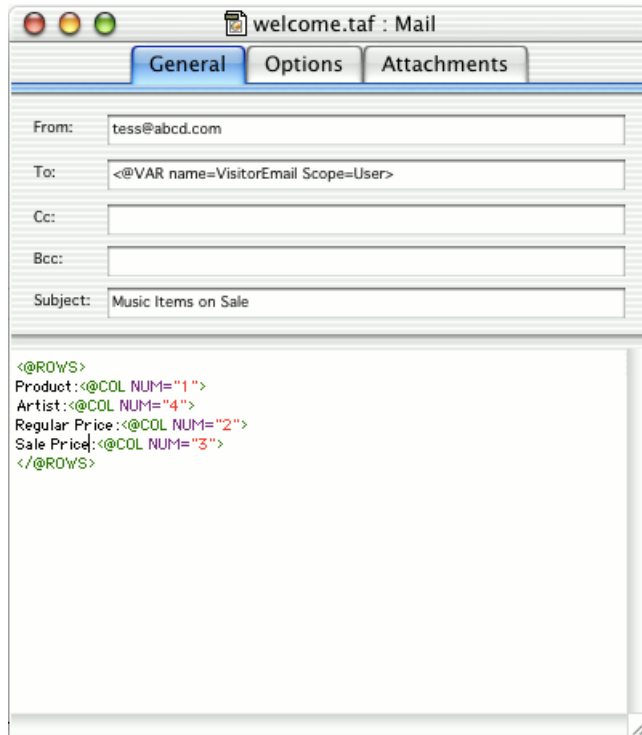
- 10 Now open the **Inside Rows Loop** folder (located above the Rows Loop snippet), and double-click or drag in a **Column Value (Numbered)** tag after Product :



- 11 The **Get_Sales_Items** action returns three columns from the database. For the first <@COL> tag, you need to enter 1 in between the quotations to the right of the NUM= attribute.

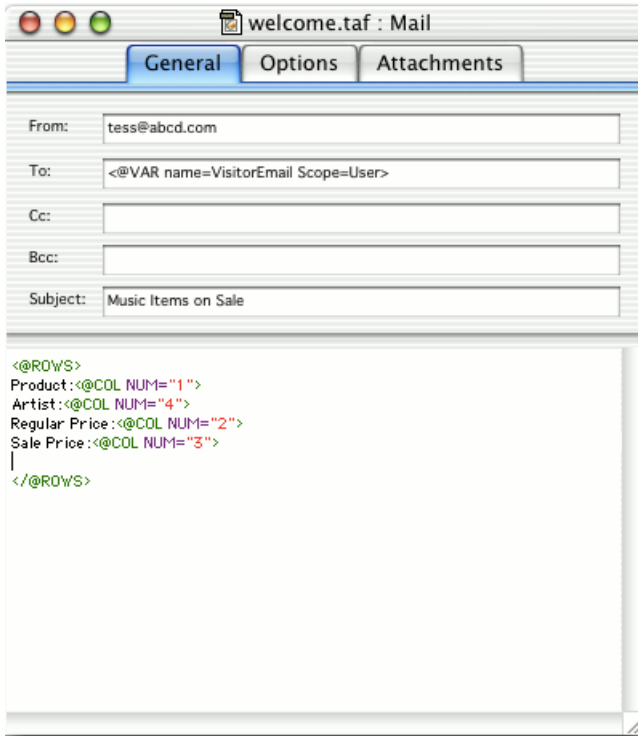


- 12** Copy and paste the `<@COL>` tag three times, changing the numbers to 4, 2 and 3 respectively. Add the labels Artist: , Regular Price: and Sale Price:, respectively.

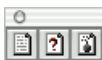
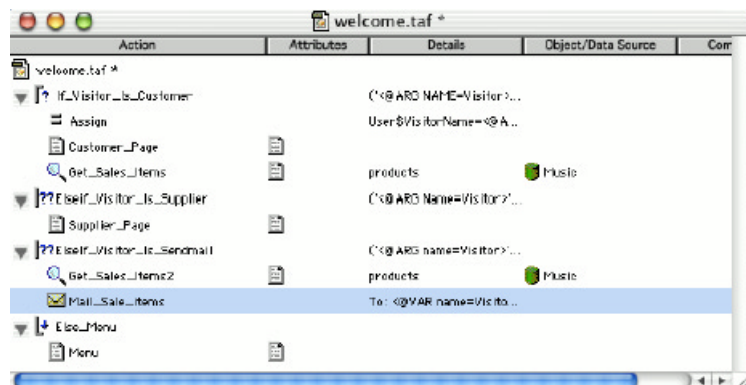


You may recall that the text contained between the `<@ROWS>` `</@ROWS>` pair of meta tags defines the result set to be returned from the database. The `<@COL NUM=>` meta tag returns the values of the column NUM in the current record of the result set (or array). The three columns—Product, Regular Price, and Sale Price—display the corresponding information for each sales item record, which are, in turn, determined by the `Get_Sales_Items` action.

- 13 Press ENTER after the last `<@COL>` and before the `</@ROWS>` to format the e-mail.



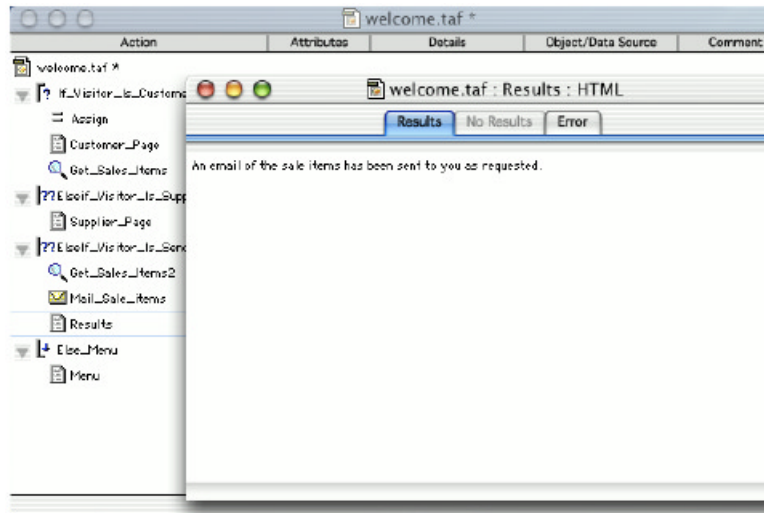
- 14 Close the Mail action window and rename the **Mail** action to **Mail_Sale_Items**.



Attributes Toolbar

- 15 Finally, click the **Results HTML** attribute button on the **Attributes** toolbar and type An email of the sale items has been

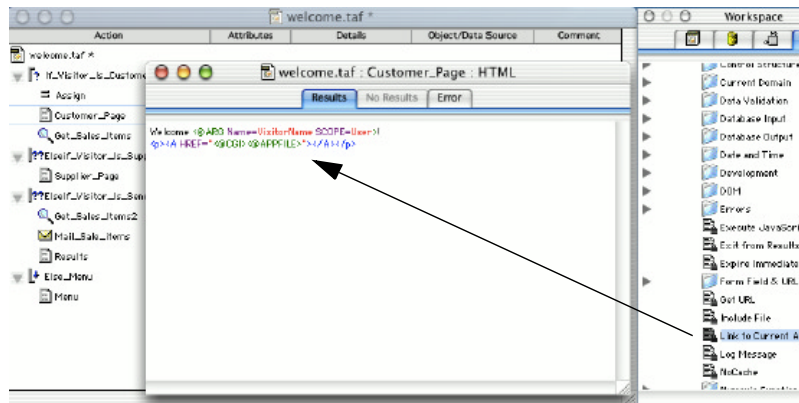
sent to you as requested. in the Results HTML editing window.



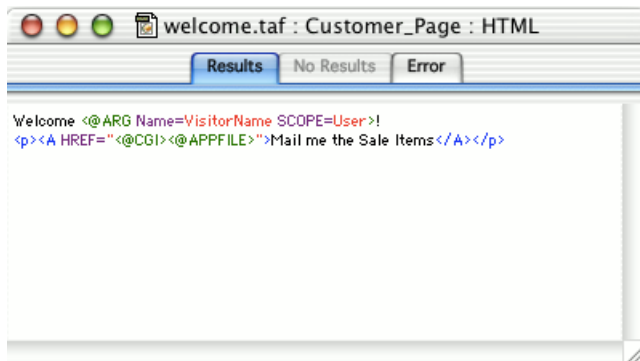
The message you type in the Results HTML is what the user sees as the response message to his or her request, made on the Customer Page, to be sent the sale items by e-mail. To create such a request mechanism, you need to create a hyperlink to the Mail action.

- 16** Open **Customer_Page**. Below the welcome message in the Results HTML editing window, add a `<P>`. Press ENTER.

In the Snippets Workspace, open the **Meta Tags** folder. Double-click or drag in the **Link to Current Application File** meta tag snippet. Type `</P>` at the end of the line.



- 17** Between the tags, type Mail me the Sale Items and place the cursor after the `<@APPFILE>` tag and before the quotation mark.

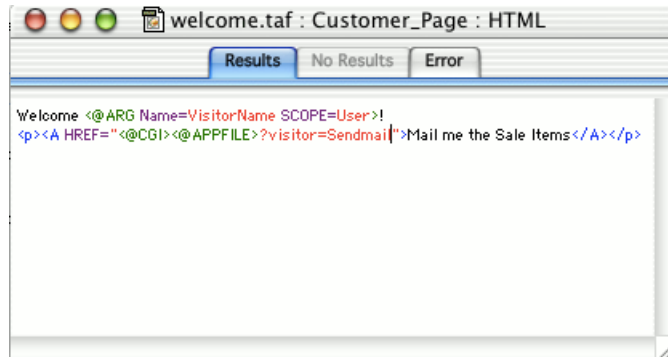


Here, you will add the necessary navigational argument needed to call the **Mail** action within `welcome.taf`.

Q. Which name-value argument pair do you use?

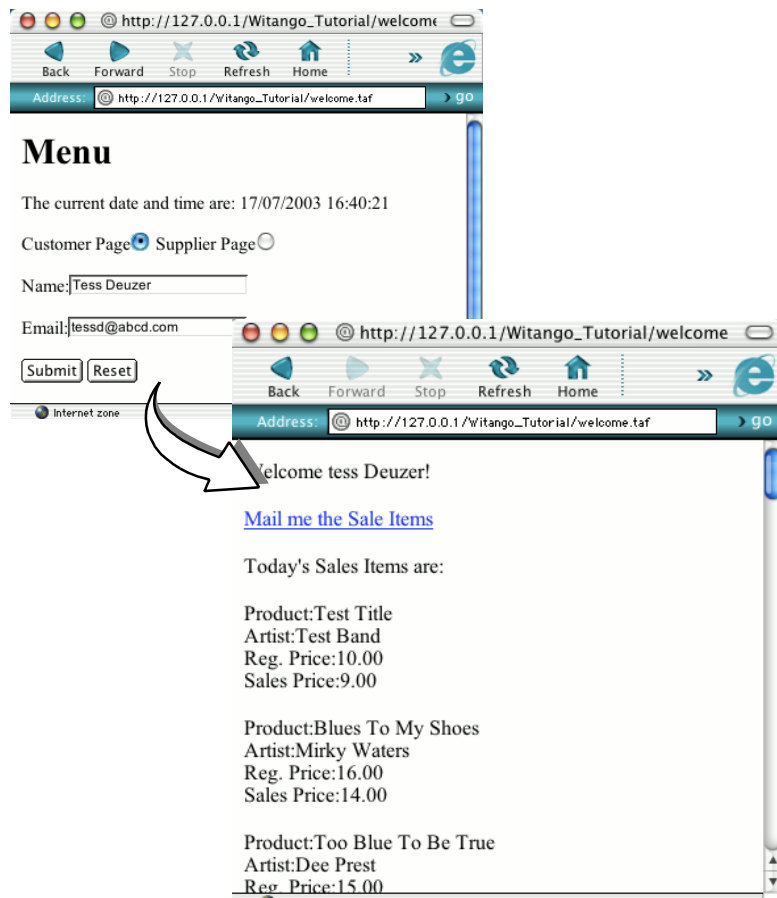
A. Visitor=Sendmail

- 18 Enter `?visitor=sendmail` (remember to include the `?` delimiter) after the `<@APPFILE>` meta tag and before the closing quotation mark.



For details, see "Saving Witango Application Files" on page 5 and "Executing Witango Application Files" on page 5.

- 19 Save the file, return to the web browser, and reload `welcome.taf`. Fill in Tess Deuzer's information again and go to the Customer Page.



To create an e-mail message when your application file is executed, you need a mail server that can send e-mail somewhere accessible to your site. For information on setting up a mail server, see page 141.

20 Finally, click the **Mail me the Sale Items** link.



An email of the sale items has been sent to you as requested.



There are many more functions of the Mail action that you can use within a Witango application. You can attach files to Mail actions, sending documents to your users, for example.

Now that your application file is working, you may check in with the Boss.

LESSON E-3

Using the `<@INCLUDE>` Meta Tag to Enhance the Menu Page

Purpose

To include files in your Witango application file using the `<@INCLUDE>` meta tag.

Context

The `<@INCLUDE>` meta tag references the contents of any file you specify, including HTML or graphics. One of the many benefits of using this meta tag is that it allows you to avoid duplicating information that you use more than once. For example, if you have some specific HTML that must be used over and over again within a Witango application file, or over a number of applications—such as a header or footer on a web page—you can place the HTML markup and text in an external file, and reference it when you need it using `<@INCLUDE>`.

The `<@INCLUDE>` meta tag also helps you to separate *presentation logic*—how pages look, and how the user navigates through the site—from Witango *business logic*—the actions, meta tags, and commands that get and organize data. That is what you are doing in this lesson: referencing marketing templates that provide distinguishing page design elements for your music Menu Page.

Exercise



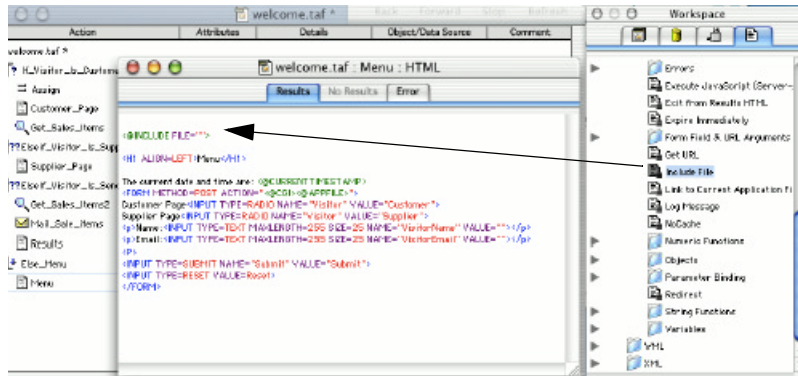
The Boss

The Boss will be away all day for an emergency dental appointment and, believe it or not, he has put you in charge! It is, however, a national holiday, but there are still a couple of people around to lord this over.

While he is away, the Boss would like you to make sure that the Menu Page reflects the company name and style.

- 1 Return to `welcome.taf`. Double-click **Menu** to open the Results HTML editing window.
- 2 Place your cursor at the very top of the page and press Enter and then place the cursor in the blank line you just created. In the

Snippets Workspace, open the **Meta Tags** snippets folder and double-click or drag in the **Include File** meta tag snippet.

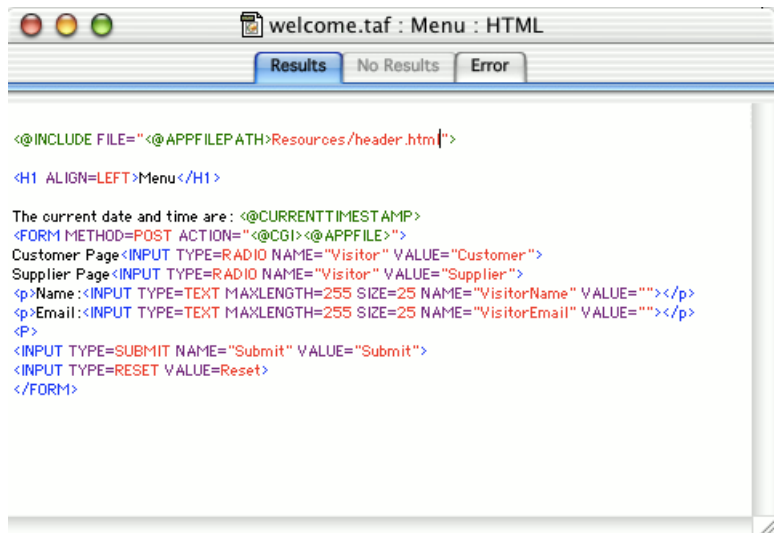


3 Between the quotation marks, type the following:

```
<@APPFILEPATH>Resources/header.html
```

For information about resource files provided with Witango Tutorials, please refer to the README in the Tutorial folder.

The header.html file displays the provided marketing templates for AB CD's. This file is located in the Resources folder, inside the Tutorial folder.



The <@INCLUDE FILE> meta tag returns the contents of whatever file is specified (in this case, header.html). The FILE attribute indicates a path from the web server document root. The value of the

FILE attribute may be literal text (like the name of a file), meta tags (such as `<@VAR NAME=myFile>`), or a combination of the two.

The `<@APPFILEPATH>` meta tag returns the path to the current application file. Because `header.html` is located in a subfolder of the Tutorial folder, you must specify the subfolder name after the meta tag.

- 4 At the very bottom of the Results HTML editing window, insert (double-click or drag in) another **Include File** meta tag to display the provided marketing templates for AB CD's. The filename for this file is `footer.html`.
- 5 Between the quotation marks, type `<@APPFILEPATH>Resources/footer.html`.

Your Results HTML should look like this:

For information about resource files provided with Witango Tutorials, please refer to the README in the Tutorial folder.

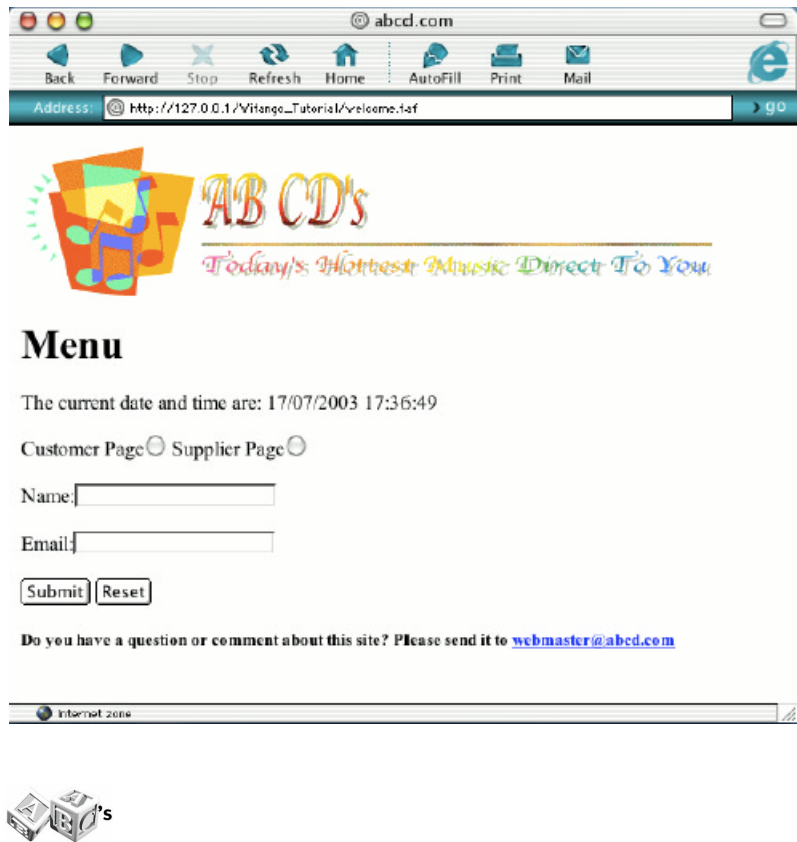
```
<@INCLUDE FILE="<@APPFILEPATH>Resources/header.html">

<H1 ALIGN=LEFT>Menu</H1>

The current date and time are: <@CURRENTTIMESTAMP>
<FORM METHOD=POST ACTION="<@CGI><@APPFILE>">
Customer Page <INPUT TYPE=RADIO NAME="Visitor" VALUE="Customer">
Supplier Page <INPUT TYPE=RADIO NAME="Visitor" VALUE="Supplier">
<p>Name:<INPUT TYPE=TEXT MAXLENGTH=255 SIZE=25 NAME="VisitorName" VALUE=""></p>
<p>Email:<INPUT TYPE=TEXT MAXLENGTH=255 SIZE=25 NAME="VisitorEmail" VALUE=""></p>
<P>
<INPUT TYPE=SUBMIT NAME="Submit" VALUE="Submit">
<INPUT TYPE=RESET VALUE=Reset>
</FORM>
<@INCLUDE FILE="<@APPFILEPATH>Resources/footer.html">
```

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 6 Close the **Menu Results** HTML window, and save the file. Return to the web browser, and reload `welcome.taf`. In order to see the effects of the <@INCLUDE> tags, do not pass in any values.



As we have seen, the <@INCLUDE> meta tag provides very powerful functionality to include all sorts of files in your Witango application files, from headers and footers and graphics to files of any sort of format. As you build Witango applications, make sure you remember to use <@INCLUDE>—it helps make your application both easier to develop and a richer experience for the user.

Now that your application file is working, you may check in with the Boss.

Login Model

Creating a Login Model for Music Club Members

One of the most common Witango web solutions is the *login model*. The login model involves a page on which a user enters their username and password. If these values match values in the database, the user is allowed into the system, and a welcome screen is displayed—that is, the user logs in to the system.

The displayed menu options after the user logs in link to other application files, which are executed, or made available, if the user has the appropriate permissions; that is, permissions information is stored for each user and, depending on the user's access level, different options are displayed.

The Witango login model is easy to build and provides a secure means for users with different access levels to enter your site.

There are four lessons in this tutorial:

F-1: Using Search Builder to Create a Framework for the Login Application File

F-2: Setting the Business Logic of the Login Solution

F-3: Storing and Tracking User Information

F-4: Preventing Unauthorized Access to Application Files in the Options Menu

LESSON F-I

Using Search Builder to Create a Framework for the Login Application File

Purpose

To use the Search Builder to build a large part of the functioning login solution.

Context

In general terms, the login model is simply a search of a database. The database contains all the valid users in the system, including their user ID and password.

When a user logs in by entering a user ID and password, Witango searches to see if these values match any of those in the database. If a match is found (indicating the user is valid), the user is permitted to see the next page (the welcome screen), which presents a list of menu options. The menu options are links that allow the user to perform other functions or activities, such as fan club registration, searching for special pricing for customers, and updating or deleting memberships. If the user ID and password do not have a match in the database of valid users, an invalid login response is displayed.

Exercise



The Boss

The Boss escorts you down to the garage and presents you with a company car. To your surprise, it is no ordinary car. It is an especially sentimental moment for the Boss as he wipes a tear from his eye and tells you that this is the very same car that was presented to him by *his* Boss, many, many years ago. “With a little work,” he says, slapping you on the back, “you’ll have it off those blocks and on the road in no time!”

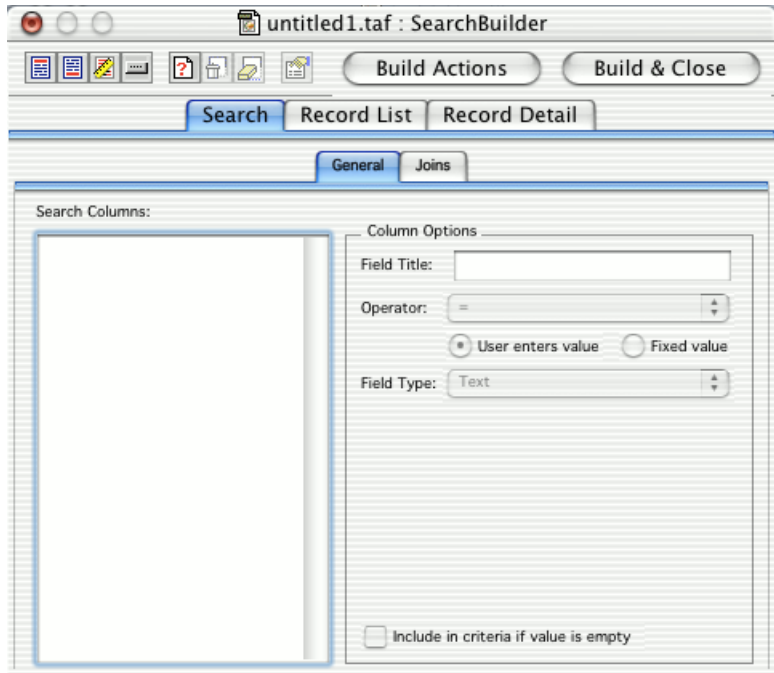
Back in the office, he asks you to create a login screen for the “Music Club” that is offered as a special service to customers. The login screen is presented first to the user, containing two form fields into which the user enters a user ID and password. If the login is successful, a personalized welcome screen should appear with a set of menu options. If the login is unsuccessful, an invalid login message should appear.

- 1 Open a new application file in Witango Studio.
- 2 Drag the **Search Builder** icon into the application file.

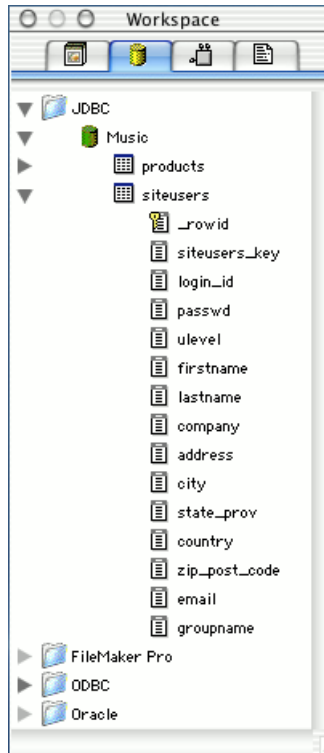


Search Builder

The Search Builder window appears.

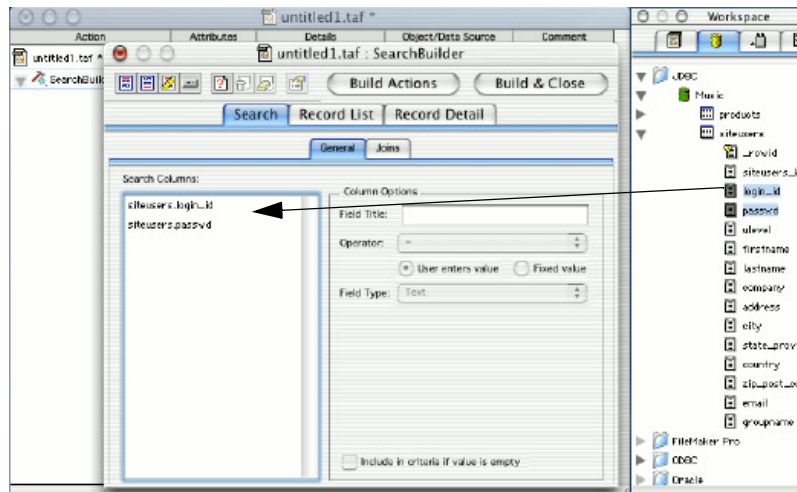


- 3 In the Data Sources Workspace, select the **Music** data source, and expand the **siteusers** table.



Note The *siteusers* table holds all the Music Club members, as well as fan club registration and product specials for members. These members will now become valid users in the login system. Many may not yet have passwords. This tutorial provides you with members with valid login ids and passwords where necessary. They already exist in the database.

- 4 Drag the **login_id** and **passwd** columns to the **Search Columns** list.

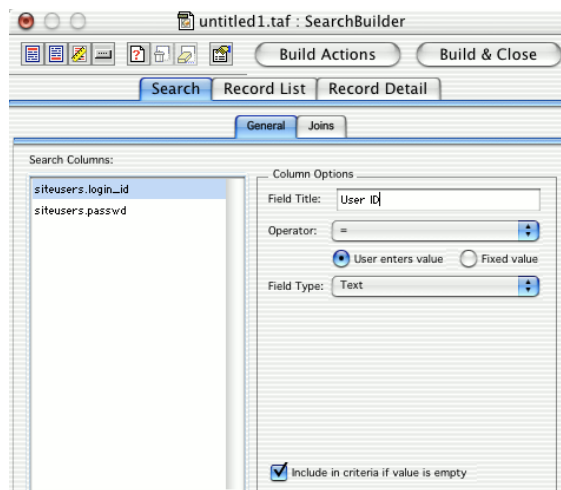


- 5 Select the **siteusers.login_id** column in the **Search Columns** list.
- 6 In **Column Options**, change the default **Field Title** to **User ID**.
- 7 In **Operator**, select **=**, if not already selected.

The check box for **Include in criteria if value is empty** becomes available when **Operator** is set to **=**.

- 8 Check **Include in criteria if value is empty**.

Your **Column Option** settings within the Search Builder should look like this:



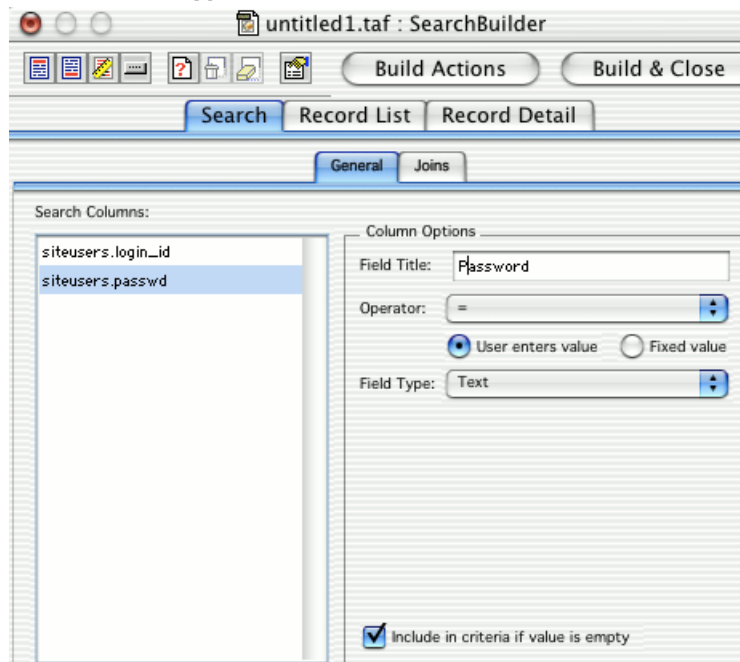


The *Include in criteria if value is empty* option determines whether Witango returns all the records in a database if the search criteria is left empty by the user. Enabling *Include in criteria if value is empty* forces Witango to use the criteria and the operator no matter what the search criteria value is, even if it is a blank or empty. If the criteria is blank, Witango goes off into the database and searches for a blank record that matches. If it does not find one, it returns a no matching records message; if one is found, Witango returns the blank record.

This is very important for Witango applications such as the login model! You do not want a user to leave the user ID and password form fields blank, and suddenly see a listing of all Music Club members in the database. If the user leaves it blank, they should receive a “no matching records” message.

It is your responsibility as database administrator to ensure there are no blank records in the database. Because of the nature of the *Include in criteria if value is empty* option, it is only available or applicable when you set the search operator to =. For example, if you specify a contains search and the user leaves the search criteria blank, Witango cannot search for a blank record.

- 9 Repeat steps 5 to 8 for the **siteusers.passwd** column: change the columns **Field Title** to Password, select = for the **Operator**, and set the **Field Type** to Text.





Field Properties

- 10** To make it so that the user's entered password does not appear as plain text in the web browser, you need to change the type of this HTML input element to `PASSWORD`. You can do this from the Search Builder. With the **siteusers.passwd** column selected, click the Field Properties icon on the Search Builder.

The Field Properties dialog box appears.

The Field Properties dialog box is shown. It has a title bar 'Field Properties'. Inside, there is a 'Default Value:' text box. Below that is a 'Text box' section. Within this section, there are three rows: 'Maximum Length:' with a value of '20' and 'characters'; 'Width:' with a value of '20' and 'characters'; and 'Height:' with a value of '1' and 'lines'. At the bottom of this section are two checkboxes: 'Scrolling Field' (unchecked) and 'Password Field' (checked). At the very bottom of the dialog are 'Cancel' and 'OK' buttons.

- 11** Check the Password Field check box, and click **OK**.



Header HTML

- 12** Click the **Header HTML** button, or choose **Header HTML** from the **Attributes** menu.

The Header HTML editing window opens.

- 13** Replace the existing HTML with the following:

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Login</TITLE>
```

```
</HEAD>
```

```
<H2>Please Log In</H2>
```

```
Enter your User ID and Password to gain access to  
the Music Club Member's Area.<BR> <BR>
```

```
<BODY>
```

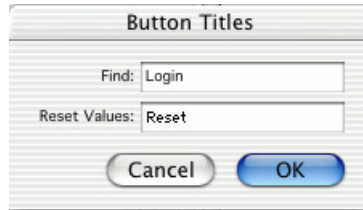
- 14** Close the window to save this information.



Button Titles

- 15** Click the **Button Titles** icon, or choose **Button Titles** from the **Attributes** menu.

The Button Titles dialog box appears.



- 16 In the **Find** field, type `Log in`.
- 17 In the **Reset Values** field, type `Reset`, and click **OK**.
- 18 Click the **No Results HTML** icon on the Attributes Toolbar, or choose **No Results HTML** from the **Attributes** menu.
- 19 Replace the existing HTML with the following:



Attributes Toolbar

```
<HTML>
<HEAD>
<TITLE>Invalid Login</TITLE>
</HEAD>
<BODY><H3>The User ID and/or Password entered are
incorrect.</H3>
Please <A
HREF="<@CGI><@APPFILE>?_function=sform">try
again.</A></BODY></HTML>
```



This HTML is displayed if no records match the user ID and password entered and provides a link back to the login form so the user can try again.

In the hot-link code, `<@CGI>` is a Witango meta tag that resolves to the file path and the filename of the Witango CGI, for example, `\cgi-bin\wcgi.exe`, which you *must* specify if the Witango plug-in is not being used. If the Witango CGI is not being used, this meta tag evaluates to nothing.

The `<@APPFILE>` meta tag resolves to the file name of the current application file, including the path to that file. You will save this application file as `login.taf` in the `\Witango\Tutorial\` folder. Hence, `<@APPFILE>` resolves to `\Witango\Tutorial\login.taf`.

As you learned in Tutorial A (“Every Good Witango URL Has...” on page 46), the purpose of using both these meta tags is to avoid hard-coding these paths. The meta tags allow you to easily change the file path or the file names of either the Witango CGI or the current

application file without the additional task of having you change any links that followed the old paths and names.

A good exercise is to create a new application file, drag in a Results action, and type out the above meta tags. Save the file in a folder in the Tutorial folder and execute the application file in the web browser. The meta tags should return the values for them at that moment in time in the browser screen. Experiment with placing the file in different folders and see what is returned.

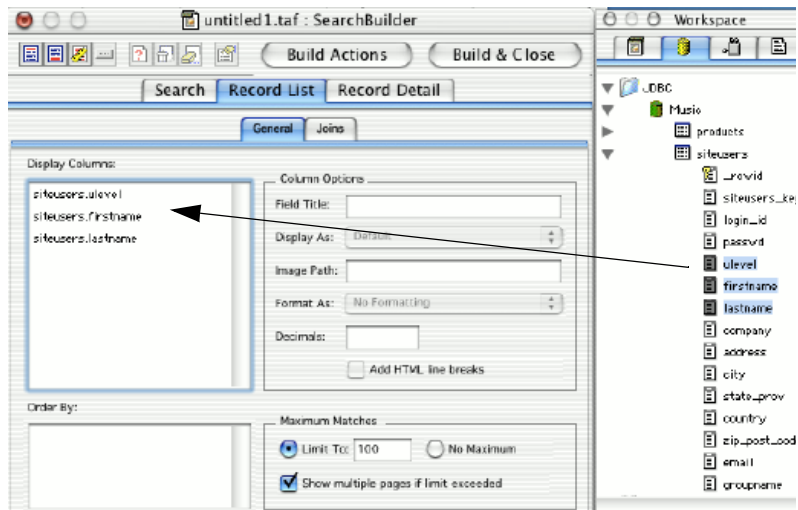
20 Close the No Results HTML window to save the information.

21 Click the **Record List** tab in the Search Builder.



The *Record List* page represents the Welcome Page in the login model. The Search Builder automatically displays a record list on this page, but this can be replaced by a list of menu items later, once you are finished using the Search Builder.

22 From the **siteusers** table, drag the **ulevel**, **firstname**, and **lastname** columns into the **Display Columns** list.



You need the values for these columns to be retrieved from the database when a user successfully logs in; you want Witango to retrieve the user's first and last names, and their user access level (ULevel). You

use these elements to create a personalized Welcome Page for the user.

(You do not need to do any page formatting or field title formatting as you will modify the Record List page considerably later in this lesson to be the welcome screen.)

- 23** Click **Build Actions** to generate the Search Builder actions.

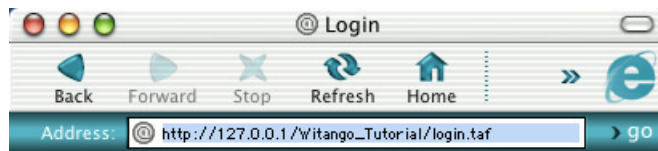
Close the Search Builder.

- 24** Save the application file as `login.taf`.

- 25** Return to your web browser and execute `login.taf`.

The opening page of the login appears, with form fields for entering **Login ID** and **Password**.

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.



Please Log In

Enter your User ID and Password to gain access to the Music Club Members Area.

User ID:

Password:



- 26** Test the login by logging in as the following valid user in the database.

Login ID: `abcd01`

Password: `public`

Asterisks appear for each character in the **Password** field.



The *Record List* page appears with the Search Builder default formatting. You will modify this page to suit the login model shortly. It will contain a personalized message to the user and list a set of menu options.

27 Click **Back** in your web browser to return to the Login Page.

28 Try logging in as a user that does not exist in the database.

The invalid login message is displayed. Test the hot-link back to the login form page to ensure it works.

Q. If ***Include in criteria if value is empty*** is enabled, and a user leaves the search criteria blank, Witango returns which of the following: all the records in the database; a database error; all blank records that exist in the database, if any exist; or the first record in the database?

A. All blank records that exist in the database, if any exist. If none exist, Witango will return an “Invalid login” page, which is exactly what you want. If you keep your database clear, a blank userid and password should always return an “Invalid login” message.

Now that your application file is working, you may check in with the Boss.

LESSON F-2

Setting the Business Logic of the Login Solution

Purpose

To learn where and how to modify the actions generated by the Search Builder to complete the login solution.

Context

You have used the Search Builder as a starting point for your login model. It has built most of the logic structure for you, but now it's time to move beyond the Search Builder and make the changes that complete the login model. At present, the soon-to-be Welcome Page and the invalid login message page are Results HTML and No Results HTML attributes of the Search action titled RecordList, which searches the database for the user ID and password entered by the user.

In this lesson, you separate these components from the Search Builder and display each according to whether there are results from the search or not. By separating the Welcome Page and invalid message page, you leave the opportunity to add other actions that may be required in either a valid or invalid login.

Exercise

Finally, the Boss has given you a much-deserved raise! In fact, it is more than enough to make up for the money you are losing in gas consumption with the “new” company car.



The Boss

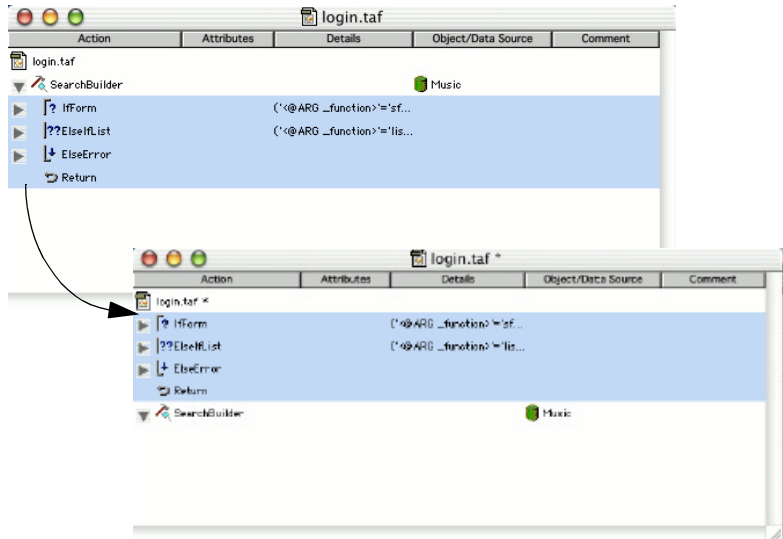
1 Return to `login.taf` in Witango Studio.



In this lesson, you will modify the actions generated by the Search Builder. If you were to return to the Search Builder and click the **Build Actions** button to rebuild the actions, your changes would be *lost*. To prevent yourself or a colleague from accidentally doing this, you must remove the actions from the builder group.

2 Hold down the **SHIFT** key and highlight all of the actions in the **Search Builder** group.

- 3 Drag the block of highlighted actions out of the builder group and drop them above the **Search Builder** action.



You may choose to leave the Search Builder after the return in case you need to regenerate some actions in the future, or you may discard it. If you rebuilt actions in the Search Builder, a new set of actions will be generated; the old ones will be left untouched. For this lesson, you do not need to keep the Search Builder.

- 4 Delete the **Search Builder** action, if you choose to.



Now, you drag in all the actions you need for the logic that determines whether Witango sends the Welcome Page or the invalid login page to the web browser. You drag in the actions all at once but set them after, to help you visualize the flow and business logic of the application file when it executes.



If action



Results action

- 5 Hold down the ctrl key and drag an **If** action into the **ElseIfList** action. Ensure it appears in sequence after the **RecordList** action.
- 6 Rename the new **If** action to **If_Invalid_Login**.
- 7 Hold down the ctrl key and drag a **Results** action into the **If_Invalid_Login** action so that it becomes a sub-action of **If_Invalid_Login**.

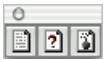
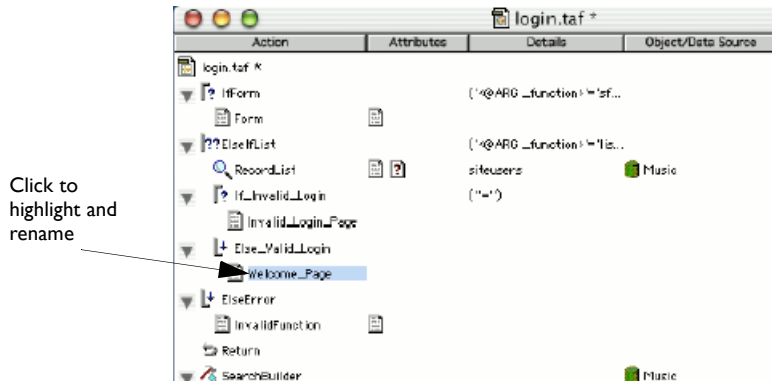


Else action



Results action

- 8 Rename the **Results** action to **Invalid_Login_Page**.
- 9 Hold down the ctrl key and drag an **Else** action into the **ElselfList** action.
- 10 Rename the **Else** action to **Else_Valid_Login**.
- 11 Hold down the ctrl key and drag a **Results** action into the **Else_Valid_Login** action so that it becomes a sub-action of **Else_Valid_Login**.
- 12 Rename the **Results** action to **Welcome_Page**.



Attributes Toolbar

- 13 Select the **RecordList** action. Choose **Results HTML** from the **Attributes** menu, or click the **Results HTML** icon on the Attributes Toolbar, to open the Results HTML editing window.
- 14 Delete the HTML shown there and close the window. (You do not need this HTML because its presentation does not suit the login model.)
- 15 Select the **RecordList** action, if not already selected, and choose **No Results HTML** from the **Attributes** menu, or click the **No Results HTML** icon on the Attributes Toolbar, to open the No Results HTML editing window.
- 16 Select and cut the HTML shown there (that is, delete it and copy it to the clipboard with the **Cut** command, available from the **Edit** menu). This is the HTML you typed in the No Results HTML window in the Search Builder for the login model. However, it should now go in the **Invalid_Login_Page** action.
- 17 Close the window.

The **No Results Attribute** icon disappears for the **RecordList** action.
- 18 Double-click the **Invalid_Login_Page** action.

A blank HTML editing window opens.

- 19** Paste in the HTML you cut from step 16, and close the window.



You now have an outline of the login model's logic that takes place after the search for a login ID and password. You now return to set the If and Elseif actions.

The criteria that determines whether Witango displays the invalid login message or the welcome message depends on whether or not there are any results from RecordList, the Search action that searches the database for the user ID and password. If there are results from the search, it means the user's User ID and Password were found in the database; it was a successful login. If there are no results from the search, it means the user's User ID and Password were not found in the database; it was not a successful login.

The number of records returned by the search is determined by the `<@NUMROWS>` meta tag.

- 20** Double-click **If_Invalid_Login** to open it.
- 21** In the first **Value** column, enter `<@NUMROWS>`. This resolves to the number of records that are retrieved from the database, and resolves to zero if no columns are retrieved from the database.
- 22** From the **Oper.** drop-down menu, select `=`. In the second **Value** column, type `0` (zero).

login.taf : If_Invalid_Login

Action: If

If the following evaluates to true:

Value	Oper.	Value
<@NUMROWS>	=	0

Advanced>>



If one or more rows are returned from the database, then `<@NUMROWS>` is greater than 0. If no rows are returned from the database, then `<@NUMROWS>` executes the invalid login page.

23 Close the **If_Invalid_Login** action.



`Else_Valid_Login` is the corresponding Else action to `If_Invalid_Login`. You do not need to set an expression to evaluate in this action because it evaluates the opposite expression found in `If_Invalid_Login`. In other words, if `<@NUMROWS>` is *not* equal to 0, the `Welcome_Page` is displayed.

24 Double-click **Welcome_Page** to open the HTML editing window.

25 In the Snippets Workspace, open the **HTML** folder and double-click **Page Template** to insert HTML code that formats Welcome Page as a proper HTML page.

Between the `<TITLE></TITLE>` tags, type `AB CD's Welcome Page`.

26 Place your cursor after the opening `<BODY>` tag. In the Snippets Workspace, double-click **Heading1 (Presentation folder)** to insert HTML code for the largest size heading possible.

27 Between the `<H1></H1>` tags, type `Welcome Page`.

28 Type the following HTML on a new line:

```
<P>Thank you for logging in,
<@VAR NAME=resultset[1,firstname]>
<@VAR NAME=resultset[1,lastname]>. You have the
following menu options:</P>
```

The user sees this as text in the web browser.



```
<@VAR NAME=resultset[1,firstname]> <@VAR
NAME=resultset[1,lastname]> references the first and last
name of the user who has just logged in.
```

29 Start a new line after `</P>`. In the Snippets Workspace, double-click on **Hyperlink** in the **HTML** snippets folder to insert the following code: `<A HREF=" "></A>`. Between the quotes, type:

```
<@CGI><@APPFILEPATH>fanclubregister.taf?
<@USERREFERENCEARGUMENT>
```

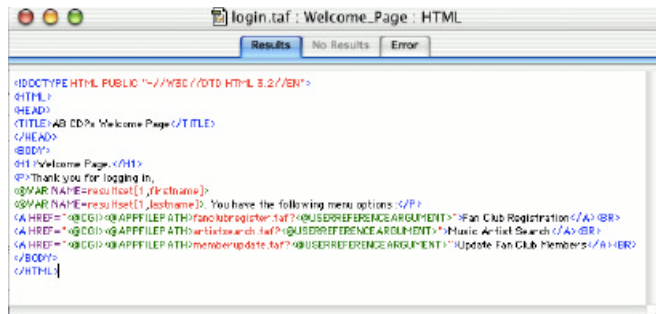
Between the opening `<A>` and `</A>` tags, type Fan Club Registration.

- 30** Copy and paste this hyperlink two more times, each on a separate line, and change the filenames to `artistsearch.taf` and `memberupdate.taf`, respectively.

Change the text between the opening and closing tags to Music Artist Search and Update Fan Club Members.

- 31** Place a `<BR>` tag between each hyperlink, so the links are displayed on separate lines.

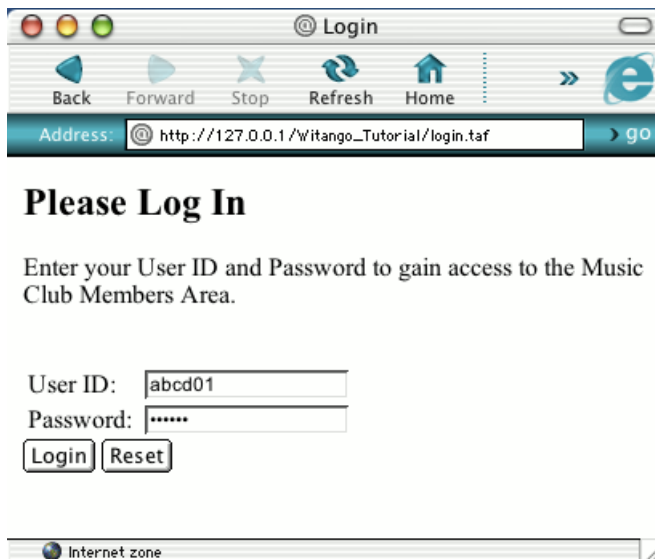
Your HTML for **Welcome_Page** should look like this:



- 32** Close the HTML editing window.

- 33** Save `login.taf`.

- 34 Return to your web browser and execute `login.taf`.

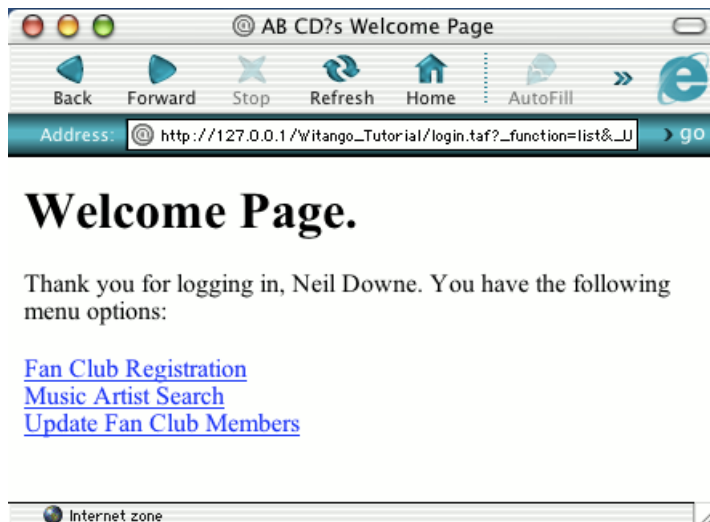


- 35 Log in with the following user ID and password:

User ID: abcd01

Password: public

Click the **Login** button. The following welcome message appears.



- 36 Now, try to log in with a user you know is *not* in the database.

The **Invalid Login** message appears.



Now that your application file is working, you may check in with the Boss.

LESSON F-3

Storing and Tracking User Information

Purpose

To keep track of a user after a successful login.

Context

In many Witango applications you create, you may need to keep track of information associated with a particular user's session on your web site and have it available to use in many different places— either across multiple executions of the same application file or in many different application files. In the login solution, for example, you want to retrieve a valid user's user level, so that throughout the rest of the user's session you can control which application files the user is allowed to execute or not execute.

As you have seen in previous lessons, this is done with the use of a Witango variable.

Exercise



The Boss

The Boss takes you golfing for the day so that he can show you off to the Board of Directors. Unfortunately, your very first shot on the very first hole ricochets off the vice-president's knee and crashes through the president's windshield.

Back at the office, the Boss sends you an e-mail (since he isn't speaking to you) telling you to add an Assign action that assigns user information to Witango variables for tracking. There are, however, to be no changes to the web pages the user sees.



During a database search for the user's user ID and password, certain values exist: two Post arguments named `UserID` and `Password` with the values the user submitted. These are used to search the database. If matching records are found, the database produces three values: `Firstname`, `Lastname`, and `ULevel`.

If you want to use these values beyond one Witango application file execution during the user's login session, you must assign them to Witango variables. In this lesson, you assign the results from the database search to Witango variables using the Assign action.

I Return to `login.taf` in Witango Studio.



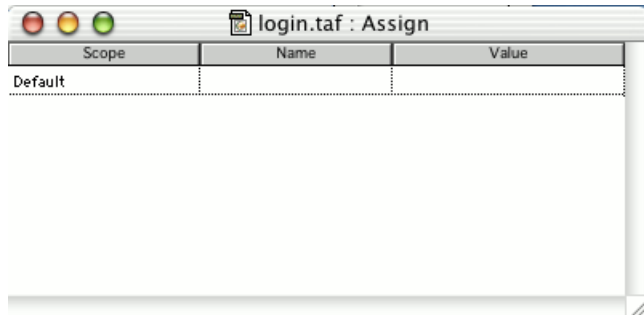
Note You want to assign the Witango variables only after a successful login, so the Assign action must be inserted in the `Else_Valid_Login` action.



Assign action

2 Drag an **Assign** action to be the first sub-action of **Else_Valid_Login**.

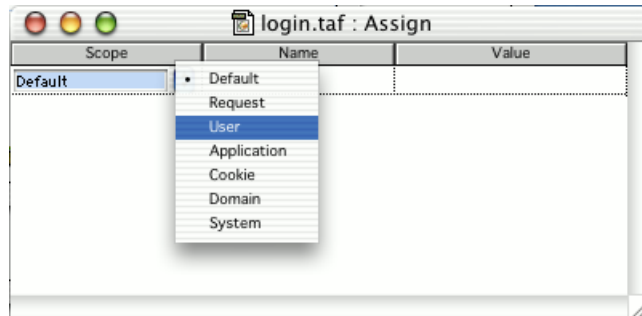
The Assign action window appears.



You have three column values returned as results from the database: `Firstname`, `Lastname`, and `ULevel`. In the next set of steps, you assign all three search results to Witango variables in the order they are retrieved from the database.

You are particularly concerned with assigning the `ULevel` value to a Witango variable because `ULevel` holds the value of the user's login system access level, and is used to determine what menu items the user is allowed to execute on the Welcome Page.

- 3 Click the **Scope** field twice. In the drop-down menu that appears, select **User**.



- 4 In the **Name** field, type `Firstname`. This names the variable `Firstname`; it is logical to name your variable the same name as the column in the database.
- 5 In the **Value** field, type `<@VAR Resultset[1,firstname]>`.



Whenever an action returns a result rowset in Witango (for example, a Search action), that rowset is assigned, in array format, to the request variable `resultSet`. `resultSet` is an array variable, which means that it is a grid of rows and columns, each of which can have a value. The columns of `resultSet` can be referred to by name or number. The above array-returning syntax `[row, column]`, when used with the `<@VAR>` meta tag, references row 1 of the array, and the column called `firstname`.

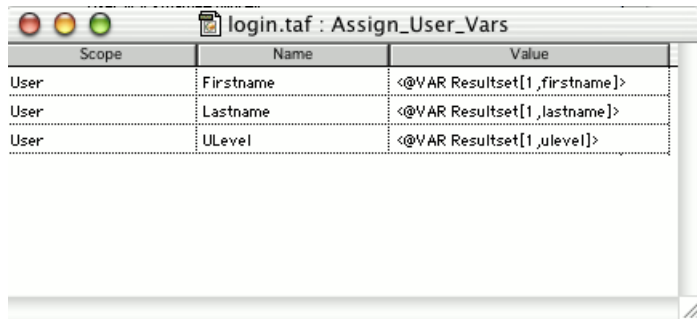
- 6 Add another row by right-clicking under the **Scope** column and choosing **Insert Assignment**.

An empty value row appears.

- 7 Repeat steps 3 to 6 using the following names and values:

<code>Lastname</code>	<code><@VAR Resultset[1,lastname]></code>
-----------------------	---

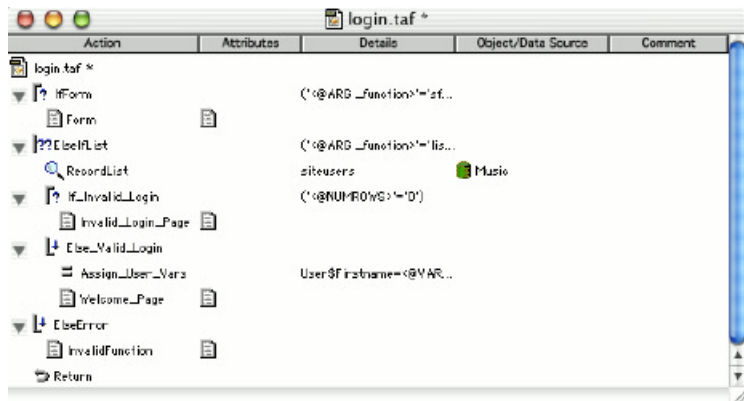
```
Ulevel      <@VAR Resultset[1,ulevel]>
```



Scope	Name	Value
User	Firstname	<@VAR Resultset[1,firstname]>
User	Lastname	<@VAR Resultset[1,lastname]>
User	Ulevel	<@VAR Resultset[1,ulevel]>

8 Close the Assign action editing window.

9 Rename the **Assign** action to **Assign_User_Vars**.



10 Save login.taf.



The user's specific variables are now set, and can be referenced using the <@VAR> meta tag as follows:

```
<@VAR NAME=firstname SCOPE=user>
<@VAR NAME=lastname SCOPE=user>
<@VAR NAME=ulevel SCOPE=user>
```

They can also be accessed using the following shorthand notation:

```
@@user$firstname
@@user$lastname
@@user$ulevel
```

Because the variable assignments take place before the display of the welcome message in the execution of login.taf, you could now

replace `<@VAR resultset[1,firstname]>` in `Welcome_Page` with `<@VAR NAME=Firstname>`. Both represent the same value during the execution of `login.taf`.

In the next lesson, you use the variable `ULevel` to determine whether other application files—specifically the welcome menu items—can be executed by the current user. The `ULevel` is checked at the top of these other application files, and the execution is ended if the user's access level `ULevel` is not high enough.

Now that your application file has been created, you may check in with the Boss.

LESSON F-4

Preventing Unauthorized Access to Application Files in the Options Menu

Purpose

To create a user access-level check at the top of application files to determine whether the files are executed or not for that particular user.

Context

In the previous lesson, you assigned the user's user access level to a Witango variable called `ULevel`. In this lesson, you insert three actions at the top of three application files (provided with Witango Tutorials) that check `ULevel` and execute the rest of the actions in the file if the user's access level is of a sufficient level. The three actions can be implemented at the top of any application files. The user's access level has been pre-determined and is a value in the database.

Exercise



The Boss

The Boss is talking to you again—he must want you to do something for him. In fact, he does, and you are the only one who can do it!

A Welcome Page with three menu options is presented to a user upon successful login: Fan Club Registration, Music Artist Search, and Fan Club Members Update. The access levels necessary to execute these application files are “1”, “5”, and “10”, respectively. If users have an access level of “1”, they are able to get to the Fan Club Registration form (allowing them to register to the fan club), but receive an “Access Denied” message when executing Music Artist Search or Fan Club Members Update. Users with an access level of “5” can get to the first form page for the first two menu options successfully, but not the last. The user (likely the system administrator) with an access level of “10” can get to the first form page for all three menu options.

- 1 Return to `login.taf` in Witango Studio.
- 2 In Witango Studio, open the application files that are the three menu options on the Welcome Page. These files (provided with the Witango Tutorials) are found in the `\Witango\Tutorial\` folder: `fanclubregister.taf`, `artistsearch.taf`, and `memberupdate.taf`.
- 3 In `fanclubregister.taf`, hold down the ctrl key and add an **If** action to the beginning of `fanclubregister.taf`.
- 4 Hold down the ctrl key and add a **Results** action to the **If** action, executed only if the expression in the **If** action resolves to true.



If action



Results action



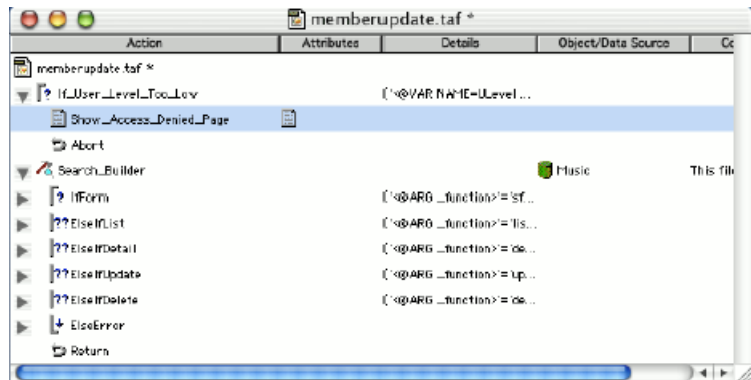
Return action

Make sure the **Results** action is within the **If** action, that is, indented and below.

- 5 Add a **Return** action to the **If** action below the **Results** action. Make sure the **Return** action is within the **If** action, that is, indented and below.
- 6 Rename the three new actions as follows:

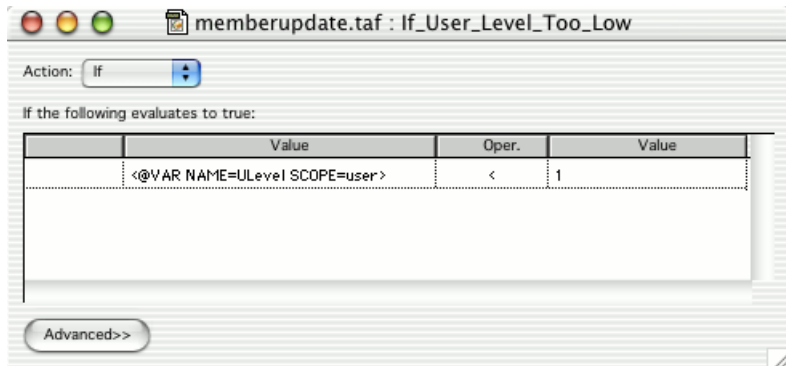
Rename...	To...
If	If_User_Level_Too_Low
Results	Show_Access_Denied_Page
Return	Abort

Now that you have the file logic outlined, you return to each action to set it or its attributes.



- 7 Double-click the **If_User_Level_Too_Low** action.
- The If action window appears.
- 8 In the first **Value** field, enter <@VAR NAME=ULevel SCOPE=user>, which is the value for the current user's access level.
- 9 From the **Oper.** drop-down menu, select < (less-than sign).

10 In the second **Value** field, enter 1.



The actions within the If action will be executed only when the user's access level is less than one, which it specifically would be if the user has not logged in at all — it would be 0.

11 Close the If action window.

12 Double-click the **Show_Access_Denied_Page** action to open the HTML editing window.

13 Type the following:

```
<TITLE>Access Denied</TITLE>
<H1>Access Denied</H1>
Either you have not logged in, or your user level
does not permit access to this function.<BR>
To log in, click
<A HREF="<@CGI>
<@APPFILEPATH>login.taf?_function=sform">here.
</A>
```



This is the HTML the user sees if the If action above resolves to true.

14 Close the HTML editing window.

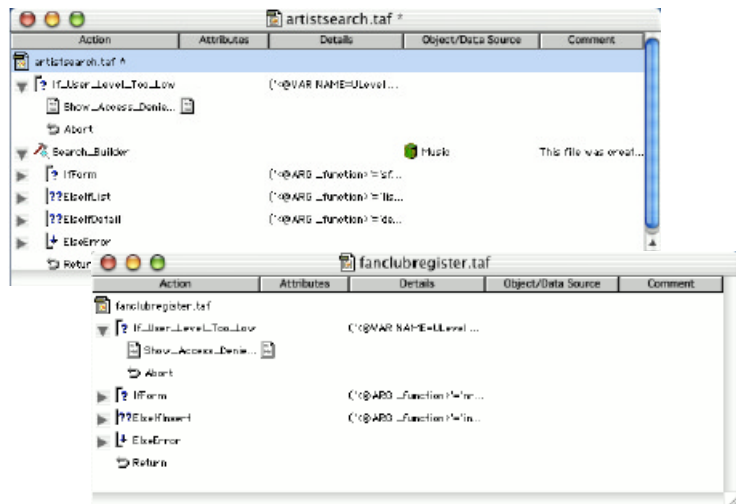
15 Save fanclubregister.taf.

For details, see "Saving Witango Application Files" on page 5.



It is easy to drag these three actions—**If_User_Level_Too_Low**, **Show_Access_Denied_Page**, and **Abort**—and use them at the top of any application file. The only item that needs to be adjusted is the access level you want to set for the application file in question.

- 16 Highlight the first three actions of `fanclubregister.taf`: **If_User_Level_Too_Low**, **Show_Access_Denied_Page**, and **Abort**.
- 17 Drag the three actions to the top of `artistsearch.taf`.
- 18 In `artistsearch.taf`, open the **If_User_Level_Too_Low** action.



- 19 Change the value in the second **Value** field to 5, so that an access level of five or greater is necessary to continue execution of the application file.
- 20 Save `artistsearch.taf`.
- 21 Repeat steps 16 to 20 for `memberupdate.taf`, setting the access level to 10 in step 19.
- 22 Return to your web browser and execute `login.taf`.

Log in as different users with different access levels. The following users exist in the database.

ULevel	Name	Login id	Passwd
1	Neil Downe	abcd01	public
5	Amanda Love	abcd04	member
10	Tess Deuzer	abcd13	administrator

- 23** On the Welcome Page, click through the menu options. The menu options available to you depend directly on the access level of the user you logged in as.



You have now completed the login solution.

The primary Witango application file is `login.taf`, which presents a login screen to the end user, then checks the User ID and password entered against the database. If the userid and password are found in the database, the user's level is selected from the database, and a Welcome Page is returned to the user, showing three hyperlinks. The hyperlinks point to three other Witango application files that serve different functions.

The user must have the appropriate user level to run each of the other three Witango application files. This type of security is set up within each of the other three files. The very first action executed by Witango in each of the Witango application files is an If action that checks the user's user level. If it is too low, an "Access Denied" message is displayed, and Witango stops execution of the Witango application file (because a Return action is encountered).

In these few lessons, you have built a very secure login model.

Now that your application file is working, you may check in with the Boss.

Witango Class Files

Creating a Witango Class File and Using it in Witango Application Files

Witango class files are reusable software components that you can incorporate into Witango application files. You can create and edit Witango class files using Witango Studio.

Witango class files are constructed in the Witango Studio in much the same way that you create Witango application files, by dragging in actions and typing information into HTML windows. However, there are several additional elements that you must create for a Witango class file; these are *methods*, each with its own *parameter list*.

A method is a distinct piece of logic within an object that does something. A parameter list is the way you send values to a method. A method may have its own return value, or it may return values through a parameter list. Together, methods and parameters make up the interface to the Witango class file. When you incorporate an object into a Witango application file, you do so by using the object's methods and parameters.

In this lesson, we turn part of an existing Witango application file (the Search action of the login model) into a Witango class file, and then change our login model to call a method of the Witango class file, incorporating reusable code into your Witango application file.

There are two lessons in this tutorial:

G-1: Creating a Witango Class File

G-2: Calling the Login Method in a Witango Application File

LESSON G-1

Creating a Witango Class File

Purpose

To demonstrate how to create a Witango class file, its methods, and parameters; also, to indicate when it is ideal to create and use Witango class files.

Context

In `login.taf`, which you created in the previous tutorial, a Search action called `RecordList` is the action that actually retrieves the information needed to validate the user. The Search action accepts user input (the user ID and password) and checks for those values in the database. The Search action then outputs a result—the user's first name, last name, and user level—if a matching record is found. If no match is found, no output is produced.

Because the Search action for the login model receives input and produces output, and because it is an action that may be used repeatedly in other applications, it makes sense to create an *object* based on this Search action. An object is a piece of modularized, reusable code that you can use in your Witango applications. To modularize the Search action, we turn it into a *Witango class file*, which is Witango's own object type.

That way, you do not need to set up the Search action again when you want to authenticate a user; you simply add a *method call* to the Witango class file.

A method call is a call to an object that passes parameters to or retrieves parameters and results from the object. Any retrieved values can be put into Witango variables and used in your Witango application file.

In this lesson, you create a Witango class file that contains the Search action for the login model and has defined input and output variables, and then you create an method call to this Witango class file.

Exercise



The Boss

New Witango Class File

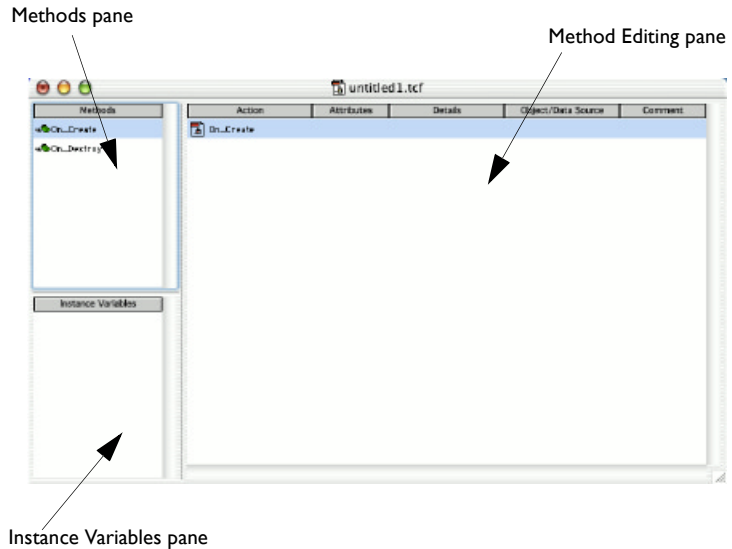
The Boss takes you aside and tells you that he is so impressed with you not only as a worker, but as a human being, that he has written you into his will. You don't ask any questions, but hope that doesn't mean you'll be left with that hideous clock in his office.

He asks you to create a reusable Witango class file that can be called in `login.taf` to replace the `RecordList` Search action.

- I Open `login.taf`.
- I Click the **New Witango Class File** icon to open a new Witango

class file.

The window that appears has three separate panes:



In the **Methods** pane, two methods appear by default: `On_Create` and `On_Destroy`.

`On_Create` directs Witango Server to execute certain actions prior to the creation of the object instance. `On_Destroy` directs Witango Server to execute certain actions prior to the destruction of the object instance.

For this lesson, you leave these methods empty.

2 Do one of the following:

- From the **Edit** menu, choose **New Method**.
- Right-click in the **Methods** pane and choose **New Method** from the context-sensitive menu that appears.

New_Method appears in the **Methods** pane.

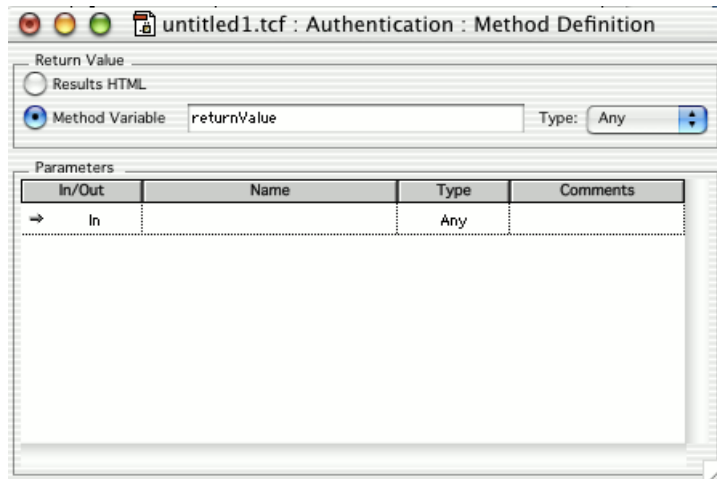
3 Rename **New_Method** to **Authentication**.

4 From `login.taf`, drag the **RecordList** Search action into the **Method Editing** pane of the untitled Witango class file window, under the **Authentication** method that appears there.



You have now created an action for the method to perform; it will search the Music database just as the Witango application file `login.taf` did. However, we still need to create the interface to that method and set up the parameters.

- 5 Double-click the **Authentication** method in the **Method Editing** pane. The Method Definition window appears:



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 Witango class file to the application file), or *input/output* (passing data from the application file to the Witango class file and putting the new value from the Witango class file into the original variable in the application file).

- 6 Right-click in the Method Definition Parameters window for the **Authentication** method, and choose Insert from the context-sensitive menu that appears. Add the following parameters to the **Authentication** method:

in/out	name
in	login_id
in	passwd
out	firstname
out	lastname
out	ulevel
out	validlogin

Leave the **Type** drop-down menu set to *Any* for all **Authentication** method parameters.

- 7 Close the Method Definition 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.

(In/Out parameters pass in a value in a variable, and then the changed value after the method is called is written to the same variable. For the sake of simplicity, we are not using this type in these tutorials.)

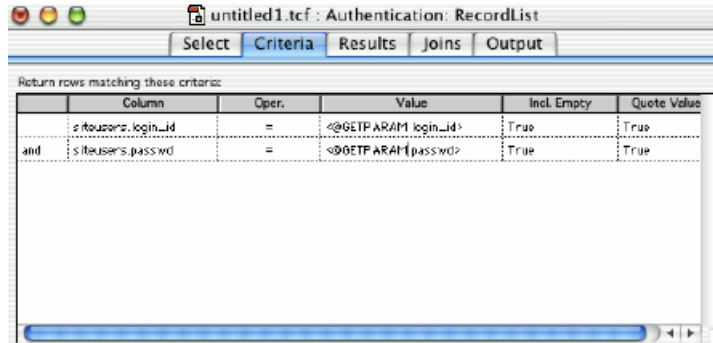
Name: This column shows the parameter names.

Type: This column shows the data type for each parameter. For this lesson, accept the default, *Any*.

Comments: This column allows you to enter your optional comments for each parameter.

- 8 Double-click the **RecordList** Search action inside the **Authentication** method to open it, and click the **Criteria** tab.

- 9 Change the meta tag in the first row of the **Value** column to `<@GETPARAM login_id>`, and change the meta tag in the second row of the **Value** column to `<@GETPARAM passwd>`.

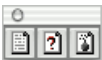


Within a Witango class file method, there are two Witango meta tags available—`<@SETPARAM>` and `<@GETPARAM>`—that set and get the value of a named parameter.

All parameters are also accessible as method variables. These tags get and set the named method variables, with the added benefit of error checking; that is, if the parameter specified is not found, an error occurs.

Accessing parameters with `<@GETPARAM>` here allows the search to be made on the parameters that we pass in from the Witango application file. However, a level of abstraction is created; that is, we do not know *how* the parameter will be passed in. It could be, as in Tutorial F, that we use arguments to pass in the parameters (`<@ARG>`). However, we do not have to; if we choose to set the parameters another way, the method call to the search action will still function correctly. This abstraction helps you to modularize the use of software components: no matter how you get the values to be passed in, the Search action in the Witango class file will be performed on them.

- 10 Close the **RecordList** action.



Attributes Toolbar

- 11 Make sure the **RecordsList** action is highlighted and open the Results HTML of this action by doing one of the following:
- choosing **Results HTML** from the **Attributes** menu
 - clicking on the **Results HTML** button on the **Attributes** toolbar.

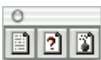
I2 Type the following into the RecordList HTML:

```
<@SETPARAM NAME=firstname VALUE=<@VAR
resultSet[1,firstname]>>
<@SETPARAM NAME=lastname VALUE=<@VAR
resultSet[1,lastname]>>
<@SETPARAM NAME=ulevel VALUE=<@VAR
resultSet[1,ulevel]>>
<@SETPARAM NAME=validlogin VALUE=1>
```



We are now setting the *out* parameters of the method within the Witango class file. These should look familiar to you: this is very similar to what we were doing before when we were taking the returned *resultSet* and creating user variables. The only difference here is that we are using `<@SETPARAM>` to set method variables, which in this case are returned to the calling Witango application file.

There is one variable you have not seen before: we need to create a value that tells the calling application file whether there has been a valid login. This new variable is called *validlogin*, and it will have the value '1' (successful login), or '0' (unsuccessful login).

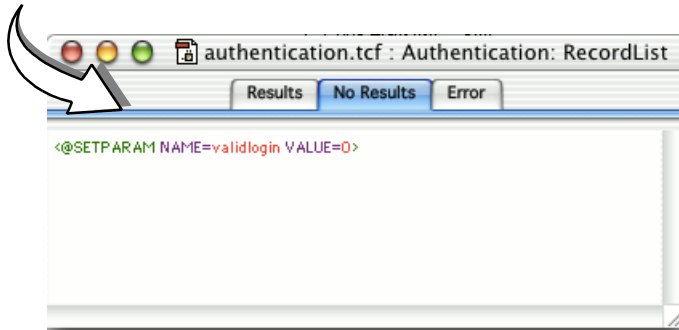


Attributes Toolbar

I3 Click the **No Results HTML** tab at the bottom of the HTML window, or click the **No Results HTML** button on the **Attributes** toolbar.

14 Type the following in the No Results HTML window:

```
<@SETPARAM NAME=validlogin VALUE=0>
```



The No Results HTML is only called when no results are found for the search action; that is, when there has not been a successful login. In this case, no parameters are set except for the `validlogin` parameter, which is set to 0.

For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

15 Close the Witango class file and save it as `authentication.tcf`.

`.tcf` is the default extension for Witango class files.



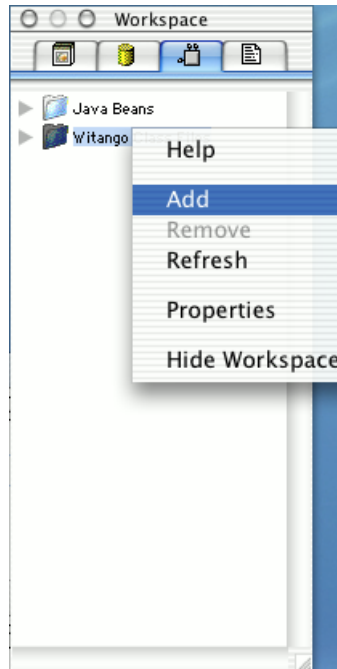
Witango Server will always search for Witango class files in certain directories, set by a configuration variable called `TCFSearchPath`.

By default, `TCFSearchPath` includes `<@APPFILEPATH>`; that is, the directory of the current Witango application file. That is where you should save your Witango class file in this case: in the same directory as `login.taf`.

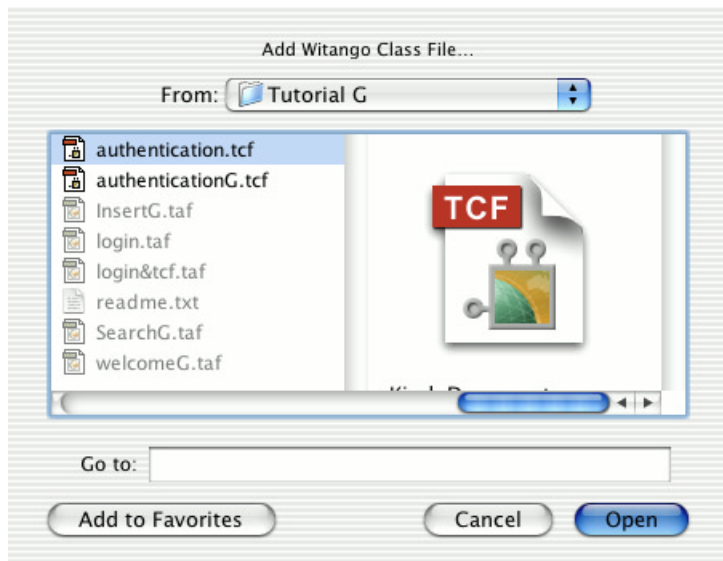
You need to add your Witango class file to the Objects Workspace so that it can be used in a Witango application file.

- I** In the Workspace, click the **Objects** tab to display the Objects Workspace.

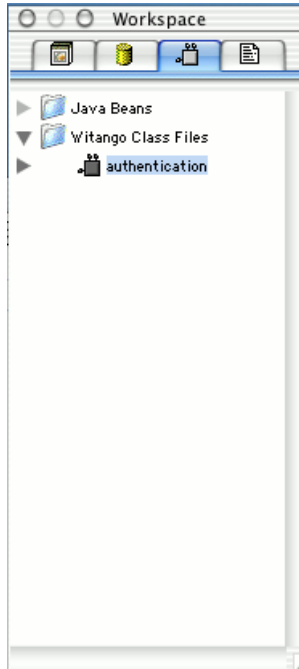
- 2 Right-click the **Witango Class Files** folder, and choose **Add...** from the context-sensitive menu that appears.



The Select Witango Class File(s) dialog box appears.



- 3 Select `authentication.tcf` (the Witango class file you saved in the Tutorial folder, which is in the Witango folder of your web server document root) and click **Open** to add it to the **Witango Class Files** folder in the Objects Workspace.



At this point, you will be able to use the `Authentication` Witango class file in any Witango application files you create with Witango Studio.

Now that your Witango class file has been created, you may check in with the Boss.

LESSON G-2

Calling the Login Method in a Witango Application File

Purpose

To demonstrate how to call an object method in a Witango application file. To call a method, you must first create an object instance.

Context

In the previous lesson, you created a Witango class file containing objects and methods. Now you can call that object method in any Witango application file. In this lesson, you return to `login.taf` and modify it so that you create an object instance and call a method of a Witango class file there.

Exercise

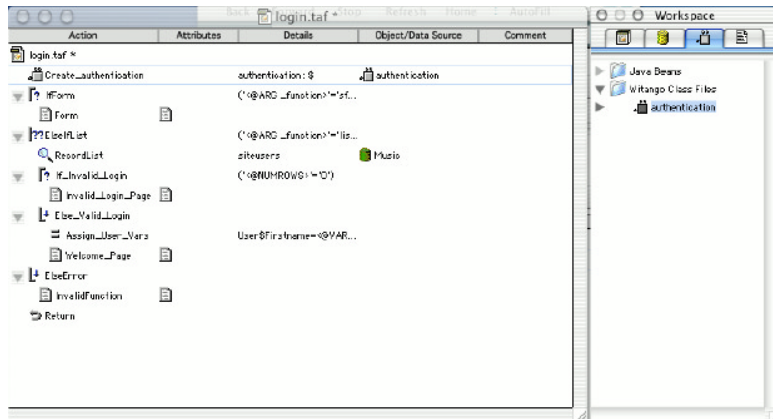


The Boss has just announced his retirement, and he is recommending you as his replacement! As your brow begins to furrow and the rest of your face slowly forms a permanent scowl, you feel confident that you are up to the challenge!

The Boss tearfully makes his final request of you. He wants you to modify `login.taf` to use a Witango class file called `authentication.tcf` that you created in the previous lesson.

- 1 Open `login.taf`.
- 2 Delete the Search action called **RecordList** and the Assign action called **Assign_User_Vars**. These two actions will be replaced by a method call to an object that carries out the login validation and takes care of assignment to the user variables.
- 3 In the Objects Workspace, double-click the **Witango Class Files** folder to open it.

- 4 Open the object instance called **Authentication** and drag it to the top of your Witango application file.

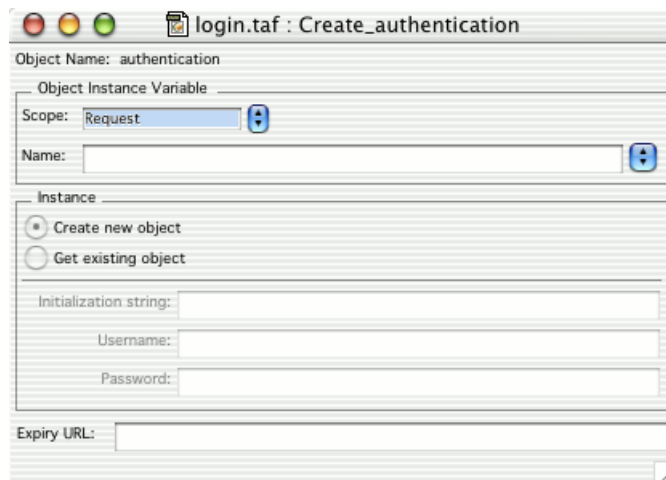


You could also drag in a Create Object Instance action and then drag the object from the Objects Workspace over top of this dialog box.



Create Object Instance

This inserts a **Create Object Instance** action, and the following window appears:



Note The name of the Create Object Instance action defaults to Create_authentication; that is, “Create_” plus the name of the object you dragged in from the Objects Workspace.



In order to use an object, you must first create an instance of the object and assign it to a Witango variable. The Witango class file itself is something like a blueprint. To make the blueprint 'come alive', you must create an instance of the object that can be used in the Witango application file. This is what is being done with the Create Object Instance action: you are creating an instance of the Witango class file, and it is that instance that values are sent to and retrieved from.

- 5 Leave the object instance variable scope set to **Request**, and type in `myObject` as the variable name.

login.taf : Create authentication2

Object Name: authentication

Object Instance Variable

Scope: Request

Name: MyObject

Instance

☒ Create new object

☐ Get existing object

Initialization string:

Username:

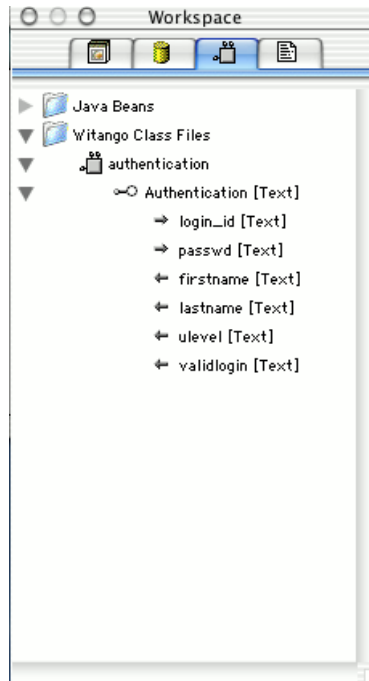
Password:

Expiry URL:

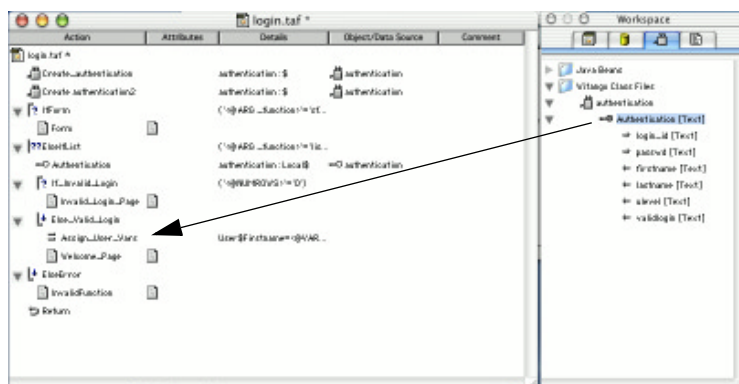
Leaving the object instance variable set at request scope means that this object variable will not be available in any other application files. When you use objects, you may want to change the scope specification of the object variable in order to reuse a particular instance of an object.

- 6 Close the Create_authentication window.
- 7 In the Objects Workspace, show the methods of `Authentication.tcf` by clicking on the **plus** sign `[+]` to the left of the Witango class file icon; then show the parameters of the

object's method by clicking on the **plus** sign [+] next to the method icon.



- 8 Drag the **authentication** method call into `login.taf`, placing it where the **Search** action used to be; that is, below the **ElseIfList** action.



You could also drag in a Call Method action and then drag the method call or any of its parameters from the Objects Workspace over top of this dialog box.



Call Method action

This inserts a **Call Method** action, and the following dialog box appears:

login.taf : Authentication

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

9 You must reference the object instance variable that was created earlier in this application file. Type `myObject` in the **Name** field of the **Object instance variable** section.

10 Now you must assign Witango variables to the parameters of this method call so that values are passed in and retrieved from this method call correctly.

Assign parameters as follows (the bold ones are the ones you should change).

in/out	name	Type	Format	Value		Incl. Empty
in	login_id	any	value	<@ARG login_id>		true
in	passwd	any	value	<@ARG passwd>		true
out	firstname	any	variable	User	firstname	true
out	lastname	any	variable	User	lastname	true
out	ulevel	any	variable	User	ulevel	true

in/out	name	Type	Format	Value		Incl. Empty
out	validlogin	any	variable	User	validlogin	true



These parameters divide up into two types: those that are passed *into* the Witango class file object (IN parameters), and those that are retrieved *from* the Witango class file (OUT parameters).

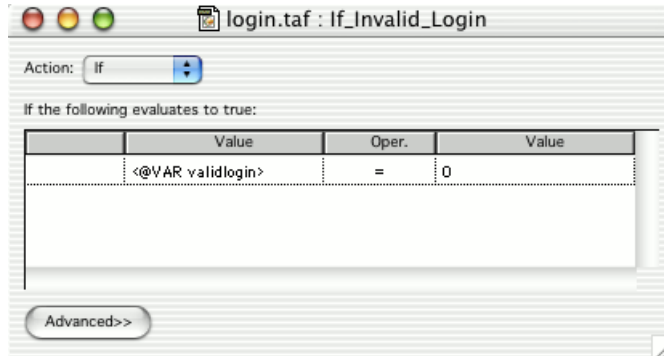
The parameters that are passed in are the `<@ARG login_id>` and `<@ARG password>` user-entered arguments that we have seen earlier in this tutorial. Instead of directly being used to search, they are being passed, via the Call Method action, to the Witango class file object, and performing the actions that it specifies; namely, performing that search.

This extra step is the *first* step to take in *modularizing* your Witango applications; that is, breaking them up into reusable pieces. If you are performing a complex search that you use often, you can create a Witango class file with that functionality, and then call that Witango class file object whenever you want that search performed. You can also change that search in the Witango class file, which would change the search every place that a Call Method action is used to call that Witango class file.

The parameters that are retrieved from Witango class file object via the Call Method action are being put directly into user variables. This skips a step that had to be done in the previous lesson: take the search results and put them into a user variable with the Assign action. In this case, the user variables are assigned by the values retrieved from the method call to the Witango class file; that is, the returned values are put directly into variables. We have chosen here to assign them directly to user variables, so the Assign action is not needed.

II Close the Call Method action window.

- 12** Open the **If_Invalid_Login** action in `login.taf`, and change the **Value** field to use the variable we set up that checks whether the user logged on successfully: `<@VAR validlogin>`.



For details, see “Saving Witango Application Files” on page 5 and “Executing Witango Application Files” on page 5.

- 13** Save the file, return to the web browser, and reload `login.taf`.
- 14** As you did in Tutorial F, log in as different users with different access levels.

The following users exist in the database:

ULevel	Name	Login id	Passwd
1	Neil Downe	abcd01	public
5	Amanda Love	abcd04	member
10	Tess Deuzer	abcd13	administrator



The Create Object Instance action and Call Method action to the Authentication method of your Witango class file have exactly the same functionality as your previous application file; the results retrieved from the database are the same. However, we have modularized the code, and made the authentication search independent of the rest of the login application file.

This has many different ramifications as you start to create and use your own Witango applications. You can start to think about the reusable actions you use over and over and ways to break them out of your application files and create Witango class files with similar functionality.

You can also add other kinds of objects to your Witango class file.

Now that your application file is working and correctly calling a Witango class file for authentication, you may sit back and relax... in the new plush chair in your office. You're the Boss!

What's Next

Concluding Remarks

Now that you have completed the Witango tutorials, you should have a good grasp of the basics required to create your own web-based applications with Witango.

To learn more about Witango development, including the developer community, go to:

`http://www.witango.com`

Glossary of Key Terms

An Alphabetical Reference of Terms used in the Tutorials

General Terms	See also “Actions” on page 212 and “Builders” on page 213.
action-based metaphor	Witango application files work by using action-based metaphors. Each action carries out a specified function and may be combined with other actions to produce Results, which are sent to a web server to return to a web page.
configuration variable	A configuration variable is a variable that controls aspects of Witango Server behavior. System configuration variables affect system-wide settings, they apply to all users of Witango Server.
cookie	A cookie is a piece of information sent by a web server to a web browser that the web browser is expected to save and send back to the web server whenever additional requests are made. Cookies are usually used to save information about the user.
data source	A data source allows Witango to connect to a particular database. Both Witango Server and Witango Studio need data sources to access databases. Witango Studio uses a data source to show a database’s information in the form of tables and their columns. Witango Studio can also be used to create and manage data sources. Witango Server requires the same data source, or a data source with the same name on the deployment machine, so it can access the specified database tables and columns when executing an application file.
drilling down	Drilling down describes the process of going through linked web pages in order to find more details about a specific record retrieved from a database. The initial web pages provide fewer details and linked pages will provide more information. Linked web pages with additional information can be set up automatically with the Search Builder.

instance	A scope that is available for variables only when using object instances in Witango class files. A variable in this scope persists across all calls to methods in the same object instance. See <i>object</i> , <i>scope</i> , <i>variable</i> .
login model	A login model is a page on which a user enters their username and password. If these values match values in the database, the user logs in to the system.
method	A method is a distinct piece of logic within an object that does something. It may have its own return value, or it may return values through a parameter list. See <i>object</i> .
method call	A method call is a call to an object that passes parameters to or retrieves parameters and results from the object. Any retrieved values can be put into Witango variables and used in your Witango application file. See <i>parameter</i> .
meta tag	Witango meta tags are special tags that are interpreted by Witango Server when your application files are executed. They can do many different things in an application file, such as controlling the flow of information, performing an action on a data source, or setting or receiving variable values. Meta tags look like HTML tags, with a beginning and ending angle bracket (< and >), but the character after the starting angle bracket is always @, 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 generally does not see the meta tags, because they are interpreted by Witango Server before being sent to the web browser.
object	An object is a reusable software component, such as a COM object, a JavaBean, or a Witango class file. Witango supports the use of objects in Witango application files. The use of objects can simplify the development process and reduce development time. See <i>method</i> .
ODBC	Open Database Connectivity is 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. See *data source*.

parameter

Parameters are the basic data elements of a method. A parameter defines what the object takes as input, output, or both. Each method includes one or more parameters. Parameters are listed in a parameter list. They are the way you send values to a method. See *method*.

post argument

A post argument is an argument passed in the HTTP header.

Results HTML

Results HTML can have two meanings in Witango. First, it is an HTML editing window that allows easy access to HTML attributes assigned to an action. Results HTML can contain meta tags that Witango Server processes. While the HTML, JavaScript, or regular text in accumulated 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.

resultSet

`resultSet` is an array variable that is generated automatically by most actions, and contains the results of that action. This variable array is stored in Witango's request scope, which means it is purged at the end of the application file execution.

scope

Scope tells Witango the relevant usage domain of a variable; that is, whether it is to be used only for a particular Witango application file execution, within the Witango application, for a user, or for a particular domain being served with Witango. The basic types of scope, from most general to most restrictive, are: System, Domain, Application, User, Cookie and Request. There are also Instance and Method scopes, which are used only with Witango class files. See *variable*.

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.

SMTP

Simple Mail Transfer Protocol is the main protocol used to send electronic 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, therefore if you wanted to set up an e-mail server on the Internet one would look for e-mail server software that supports SMTP.

Witango application file

Witango application files are written using Witango Studio and consist of one or a series of actions that are executed by Witango Server. They are also referred to as application files.

Witango class file

A Witango class file is a reusable software component native to Witango that you can incorporate in Witango application files. You can create and edit Witango class files using Witango Studio. By default, Witango class files have the `.tcf` file extension.

userKey

The `userKey` configuration variable determines the default key used for tracking user variables. The default value for `userKey` is `<@APPKEY><@USERREFERENCE><@CGI_CLIENT_IP>`.

value

A value is assigned to a variable using an Assign action or the `<@ASSIGN>` meta tag. See *scope*, *variable*.

variable

A variable is a placeholder that you can assign values to; in Witango, variables are created and assigned values using the Assign action or the `<@ASSIGN>` meta tag. You can assign any combination of literal values and meta tags to a variable. Every variable belongs to a scope which determines where they are valid. See *scope*.

Actions

Assign action

The Assign action is used to assign values to variables. See *values*, *variables*.

Call Method action

The Call Method action calls a method or an object. They are used when dealing with objects. When Witango Server executes a Witango application file that contains a Call Method action, it performs the task specified by the method. See *method*, *object*.

Create Object Instance action	The Create Object Instance action creates an instance of an object, such as a Witango class file, and it is that instance that values are sent to and retrieved from.
Else action	The Else action evaluates to true when the conditions in an If action or an Elseif action are not met. The Else action does not specify any conditions. See <i>Elseif action</i> , <i>If action</i> .
Else If action	If the condition set in an If action is not true, Witango continues to the next action. if the next action is an Elseif action, then Witango proceeds to execute the action(s) within the Elseif action if they are true. See <i>Else action</i> , <i>If action</i> .
If action	The If action evaluates expressions. If the expression resolves to true, the items grouped with the If action are executed by Witango Server. See <i>Else action</i> and <i>Else If action</i> .
Mail action	The Mail action sends out electronic mail using the Simple Mail Transport Protocol (<i>SMTP</i>). See <i>SMTP</i> .
Results action	The Results action performs no special functions on its own, but adds HTML (and evaluated Witango meta tags) to an application file's results.
Search action	The Search action retrieves records from a data source.
Builders	
New Record Builder	The New Record Builder lets you build actions to display a form on a web page, allowing users to enter data for a new database record, and returns a result message after the record is added to the database.
Search Builder	The Search Builder allows you to generate actions that produce a search form page, which leads to a record list page, with hyperlinks to a record detail page where records can optionally be updated or deleted.

The Search Builder has the following tabs:

- **Search.** Allows you to specify the columns you want users to search and define the format of the search form you want users to see in their web browsers.
- **Record List.** Allows you to specify the columns you want to display in the record list returned to the user after a search and define the format of the record list web page.
- **Record Detail.** 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, and define the format of the record detail web page. You can also set up the record detail web page to allow users to delete the record and/or edit specified columns.