# WebObjects Builder User Guide

## (Legacy)

Internet & Web > WebObjects

2005-11-09

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010



Simultaneously published in the United States and Canada.

# Contents

# Figures

# Introduction to WebObjects Builder

**Important:** The information in this document is obsolete and should not be used for new development.

WebObjects Builder is an application that provides graphical tools for creating dynamic webpages for HTML-based web applications. Each page, or part of a page, is represented by a web component in your application. An application can have one or more web components. For example, most web applications have at least one component, called the main component, used to display the first webpage. This document describes how to use WebObjects Builder to create web components.

## Who Should Read This Document?

You should read this document if you are creating a WebObjects HTML-based web application. You do not use WebObjects Builder if you are creating a Java Client or web service application. This document assumes that you are already familiar with Xcode and EOModeler. Read *Xcode 2.2 User Guide*and *EOModeler User Guide* for more information on these other tools. Read *WebObjects Web Applications Programming Guide* for general information on creating web applications.

## Organization of This Document

Before creating a web component using WebObjects Builder you should be familiar with the breadth of tools available, the types of elements and components you can use as building blocks, and the location of all these features.

If you are new to WebObjects Builder, read "Getting Started With WebObjects Builder" (page 13) and "Editing Components" (page 23)first. If you are new to HTML and web application development, read "Working With Static Elements" (page 45) next. Otherwise, if you are new to WebObjects, read "Working With Dynamic Elements" (page 51), "Using Display Groups" (page 81), and "Custom Components" (page 103).

These introductory chapters lay the groundwork for using the application:

- "Getting Started With WebObjects Builder" (page 13) introduces the common features and functions, explains the layout of the web component window, and describes the organization of elements and components you use to build other components.
- "Editing Components" (page 23) describes basic HTML editing and formatting functions that can be applied to most web content. It also covers editing specific types of elements—for example, tables and frames.

The following chapters explain how to use specific types of elements:

- "Working With Static Elements" (page 45) describes how to use static HTML elements to build your component.
- "Working With Dynamic Elements" (page 51) describes how to add dynamic elements to your web component and bind them to your application objects.
- "Using Display Groups" (page 81) describes how to use display groups, in combination with tables and repetitions, to display large amounts of data that come from a database. It also contains instructions on implementing more complex pages such as a master-detail interface.

The remaining chapters cover other important features of WebObjects Builder:

- "Working With Palettes" (page 95) describes the collection of existing palettes and how to create your custom palettes. You can save time constructing your component by using the pre-built components on palettes.
- "Custom Components" (page 103) explains how to build custom components and reuse them in your applications.
- "Validating" (page 109) describes the process of validating your web component.

# See Also

For details on using companion tools, read:

- *Xcode 2.2 User Guide*
- *EOModeler User Guide*

For an in-depth description of the WebObjects dynamic elements and extensions, read:

- *WebObjects Dynamic Elements Reference*
- *WebObjects Extensions Reference*

For an in-depth discussion of web application development, read:

- *WebObjects Web Applications Programming Guide*
- *WebObjects Enterprise Objects Programming Guide*

The `/Developer/Examples/JavaWebObjects` folder contains more in-depth code examples.

# Getting Started With WebObjects Builder

WebObjects Builder is a graphical tool for creating web components for WebObjects applications. A web component represents a webpage or part of a webpage—typically, webpages that contain dynamic content. Web components are HTML-based—therefore, they are used only by web applications—you do not use WebObjects Builder if you are developing other types of WebObjects applications such as web services and Java Client applications. For example, a web service application created using the Direct To Web Services Application template in Xcode does not generate HTML.

WebObjects Builder is not a complete integrated development environment (IDE) either. You use Xcode to manage the rest of your WebObjects project files, and use Xcode to build and test your application. Read *Xcode 2.2 User Guide* if you are not familiar with Xcode.

Before editing your web component, you should be familiar with the layout of the web component window and common functions and features, such as setting preferences, opening the inspector, and switching between editing modes. WebObjects also provides a wealth of elements and prebuilt components you use to construct your web component. It helps to understand the breadth, organization and location of these windows, menus, and controls before you begin editing your component.

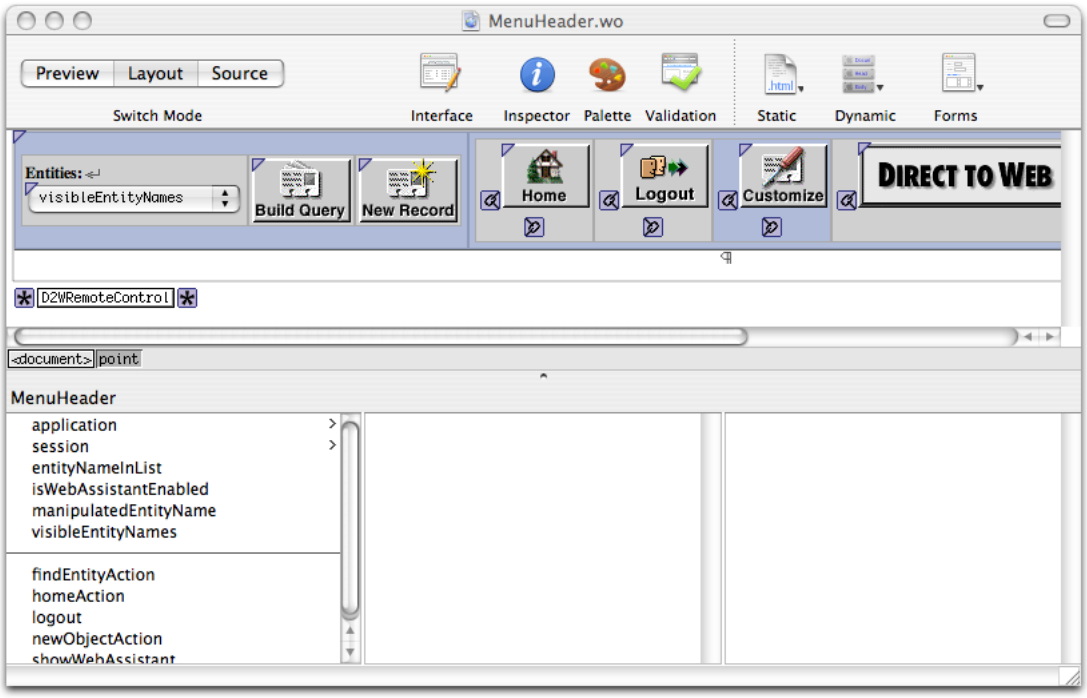
## Creating Web Components

There are several ways to create web components—some web components are created automatically for you and others you can create manually.

Typically, standard web components, like `Main.wo`, `MenuHeader.wo` and `PageWrapper.wo`, are created for you by the Xcode assistant when you select a WebObjects template. The Main component represents the first page displayed by your web application. The MenuHeader and PageWrapper components are specific to Direct to Web applications. For example, you edit the PageWrapper component to add or remove information from every webpage—this component is a wrapper for every page in your application.

Follow the steps in *WebObjects Web Applications Programming Guide* to create a Xcode project using one of the web application templates. Double-click any of the web components that appear in the Web Components group in Xcode to launch WebObjects Builder. For example, double-click the `MenuHeader.wo` file in a Direct to Web application to open it as shown in Figure 1-1.

**Figure 1-1** The MenuHeader component window




To create a new component and add it to your existing Xcode project, follow these steps:

1. Launch WebObjects Builder, located in the `/Developer/Applications/WebObjects` folder.
2. Choose File > New to create a new web component.

   A blank web component window appears.
3. Choose File > Save to save the component.
4. Enter a filename and location—for example, your project directory.
5. Click "Save as."

   A panel appears asking if you want to add the web component to the open Xcode project.
6. Click Yes.

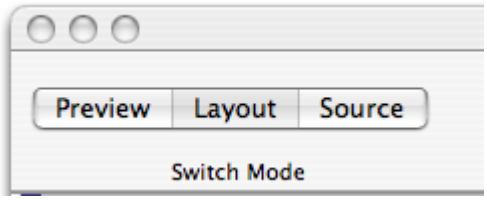
   The new component is added to the Web Components group in your Xcode project.

The rest of this chapter explains the layout of the web component window and the main WebObjects Builder menus and buttons.

# Editing Modes

When you open a web component, WebObjects Builder displays it in a **component window**, similar to the window shown in Figure 1-1 (page 14). The component window has a toolbar at the top and panes below that depend on the editing mode you select. Use the buttons in the upper-left corner of the component window, shown in Figure 1-2, to switch among the following edit modes: Preview, Layout, and Source mode.

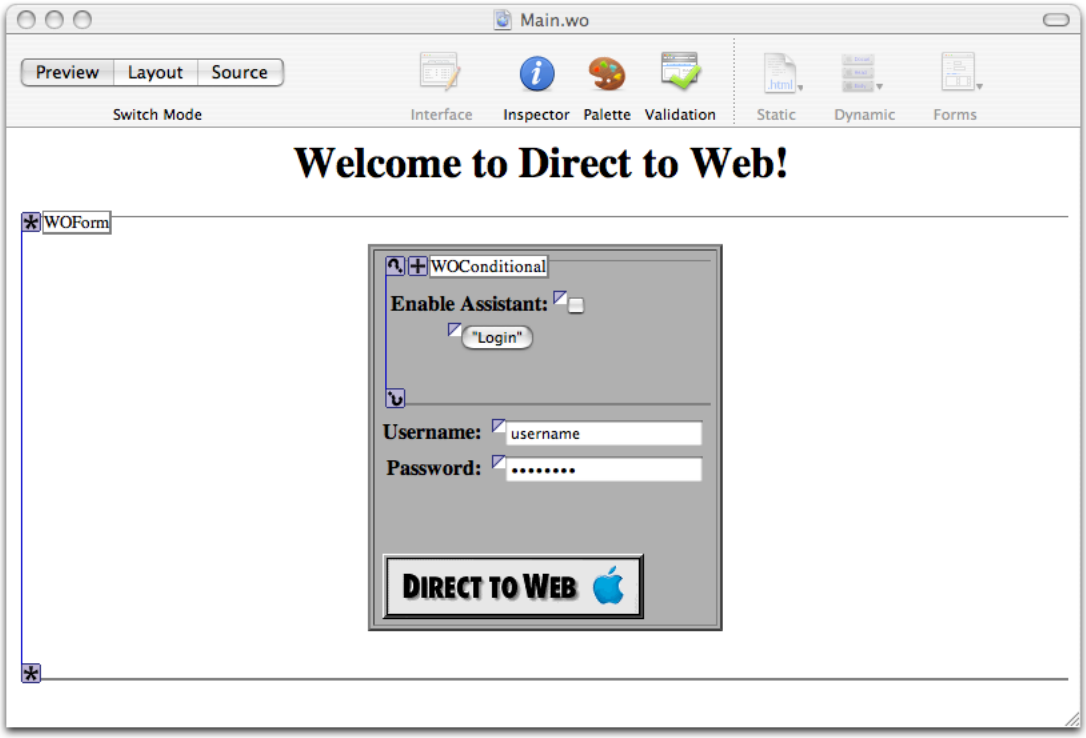
**Figure 1-2** The Switch Mode buttons




## Preview Mode

The **Preview mode** shows a visual representation of your component. This view displays an approximation of what the user sees in the browser and is not editable. The elements are collapsed as much as possible and bindings are not displayed. Figure 1-3 shows the Preview mode of a Direct to Web component.

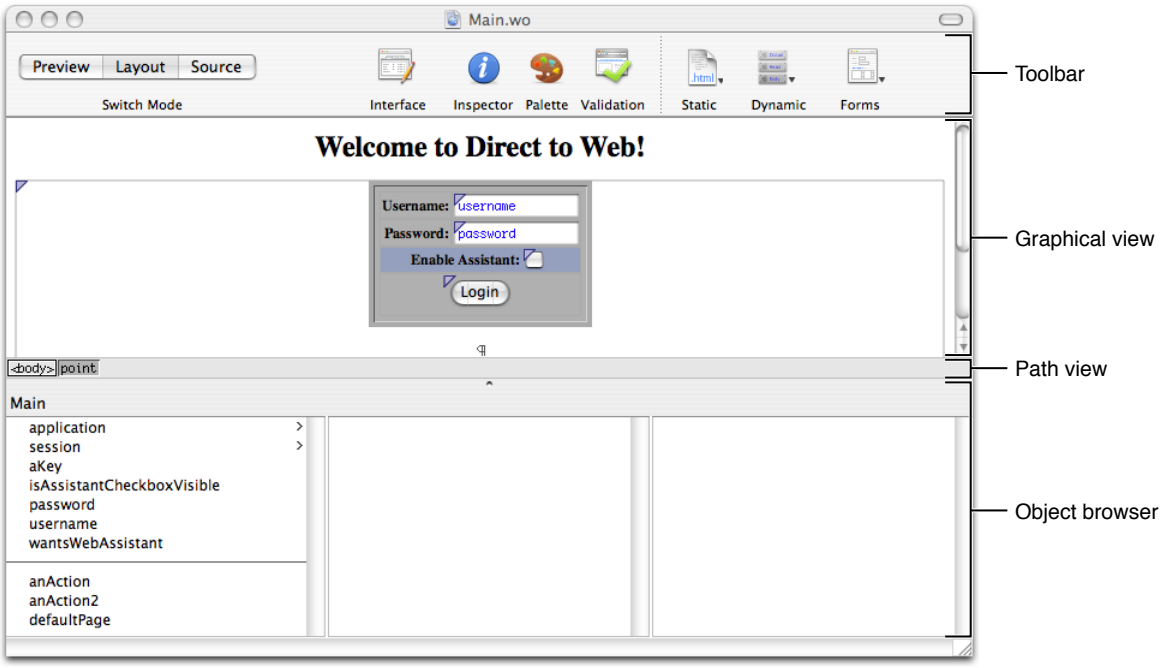
**Figure 1-3** The Preview mode



# Layout Mode

The **Layout mode** shows an editable visual representation of your component, including its dynamic elements. The visual representation is similar to that of the preview mode except that elements are expanded and bindings are displayed as shown in Figure 1-4.

**Figure 1-4** The Layout mode



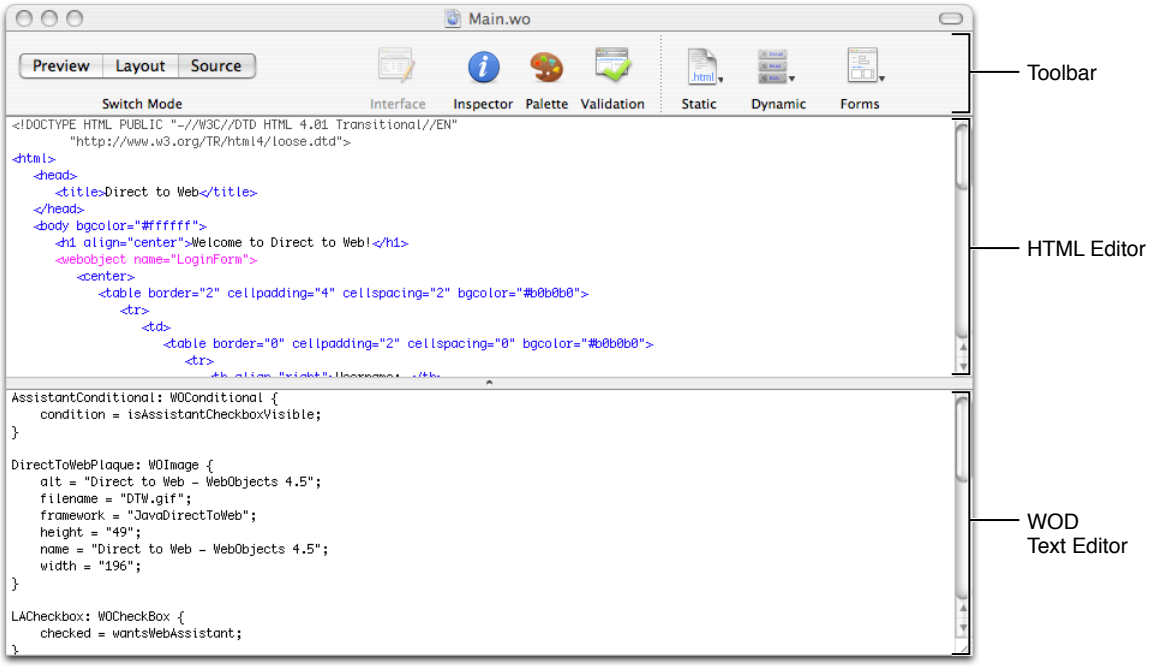

When you select Layout, you create your component's appearance graphically in the upper pane of the component window called the **graphical view**. The browser at the bottom of the window, called the **object browser**, displays variables and methods you use to bind elements to your application objects.

The **path view**, which is visible in the Layout mode only, lies between the two panes. It displays the element path to the selected element. Any element can be contained in a hierarchy of several levels of elements and can in turn contain other elements. For example, the main component is typically enclosed by an HTML `<body>` element tag, which is the top level of the hierarchy. You can click an element in the path view to select it in the graphical view.

# Source Mode

The **Source mode** allows you to view and edit your HTML source file directly as shown in Figure 1-5.

**Figure 1-5** The Source mode




When you select Source, the HTML source for your component (the `.html` file) is displayed in the upper pane and the text of your declarations (the `.wod` file) is displayed in the bottom pane. Both panes are editable views. You can enter any HTML code in the upper-pane HTML source editor. For example, you can include HTML elements that are not directly supported by WebObjects Builder's layout tools. You can also add elements using the toolbar and menus.

When you add elements in the Layout mode, their corresponding HTML tags appear in this HTML source file.

# The Toolbar

The toolbar at the top of the component window contains additional buttons and menus as shown in Figure 1-6.

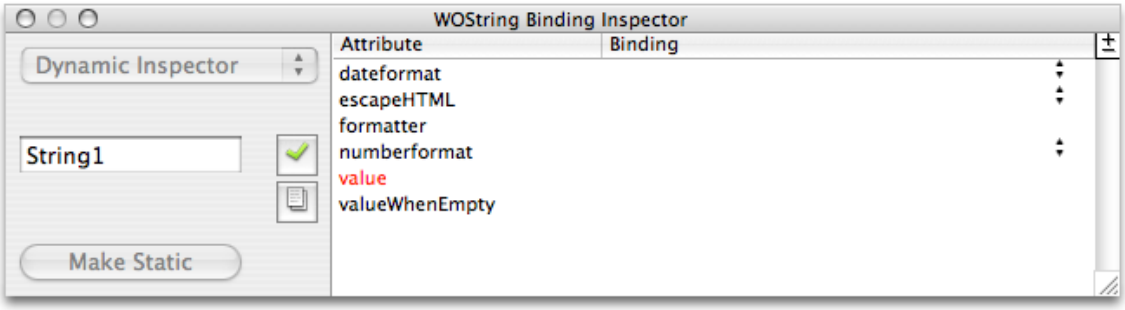
**Figure 1-6** The Toolbar


The first four buttons on the toolbar to the right of the Switch Mode buttons perform specific tasks and open additional tools. Some of the menu equivalents for these buttons are in the Window menu (where you can find the keyboard equivalents too).

- The **Interface button** contains menu items that modify the interface of your component's Java source. You use this menu to add variables and actions to your component.
- The **Inspector button** opens the **inspector window**, which allows you to set properties of the selected element.

  You use the inspector window to set both HTML and dynamic element attributes. You can open the inspector window by selecting an element in either Layout or Source mode and clicking the Inspector button, or choosing Window > Inspector. The Inspector window is unique for each element but has some common controls. For example, Figure 1-7 shows the inspector for a WOString element. Read "Inspecting Elements" (page 26) for more information on inspectors and editing elements.

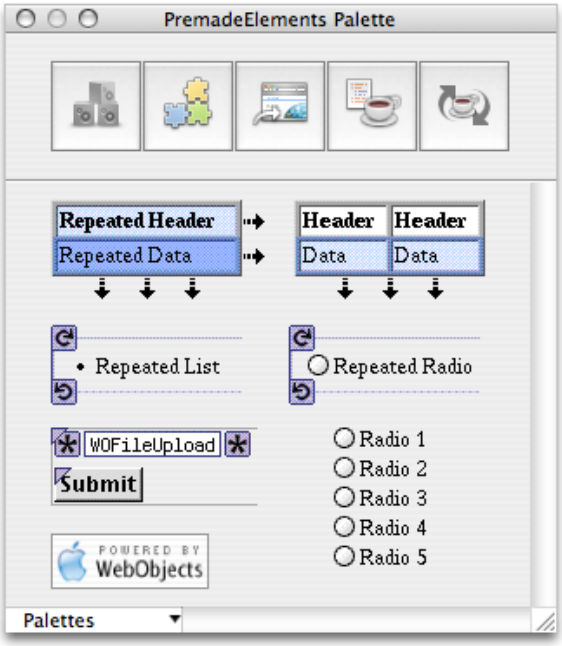
**Figure 1-7** The WOString Binding Inspector




- The **Palette button** opens the **Palette window**, which contains additional elements and components that you use to construct your web component as shown in Figure 1-8. You can also open the Palette window by choosing Window > Palette.

  WebObjects Builder contains some standard palettes including palettes containing components specifically for Direct to Web and Java Client. There's also a JavaScript palette containing various flyover and panel components. You can also create your own palettes and add them to the Palette window. Read "Working With Palettes" (page 95) for more information on this.

**Figure 1-8** The Palette window




- The **Validation button** opens the **Validation window**, which displays any binding validation errors on the page. Read "Validating" (page 109) for more information on validating your web component.

The next three buttons on the toolbar are convenience pop-up menus for adding elements to your component—you can also add the elements using the menus in the menu bar which have the same names.

- The **Static** pop-up menu contains common static HTML elements.

  Read "Working With Static Elements" (page 45) for how to use static elements.

- The **Dynamic** pop-up menu contains dynamic HTML elements that are WebObjects-specific.

  Some of these elements are concrete—have HTML counterparts—and others are abstract, such as WOConditional. Read "Working With Dynamic Elements" (page 51) for details on these elements.

- The **Forms** pop-up menu contains dynamic elements that you typically use in a form—a user interface that obtains information from the user and contains a submit button.

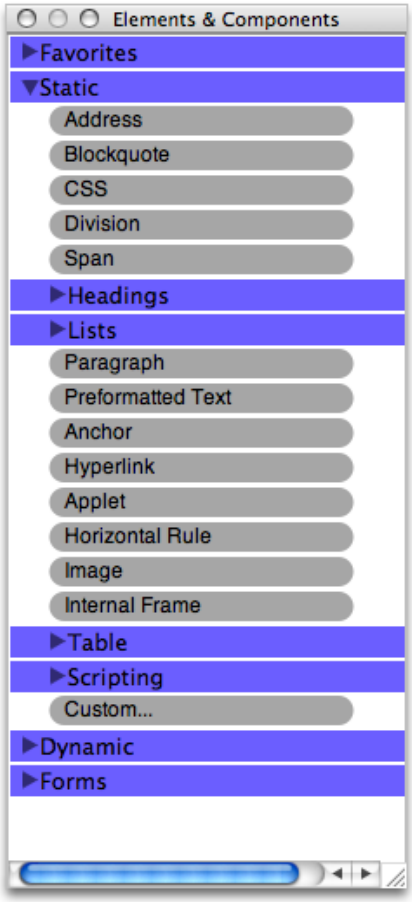
  Read "Using Dynamic Form Elements" (page 62) for details on using form elements.

Optionally, you can use the Elements & Components window to add elements to your component with a single click as described in "The Elements & Components Window" (page 19).

# The Elements & Components Window

The **Elements & Components window**, shown in Figure 1-9, provides a convenient, single-click approach to adding elements to your component.

**Figure 1-9** The Elements & Components window




The Elements & Components window contains an outline view with folders for each type of element: Static, Dynamic, and Forms. You click to the left of a folder name to reveal its contents. The folders contain the same elements that are available in the pop-up menus on the toolbar and the element menus on the menu bar.

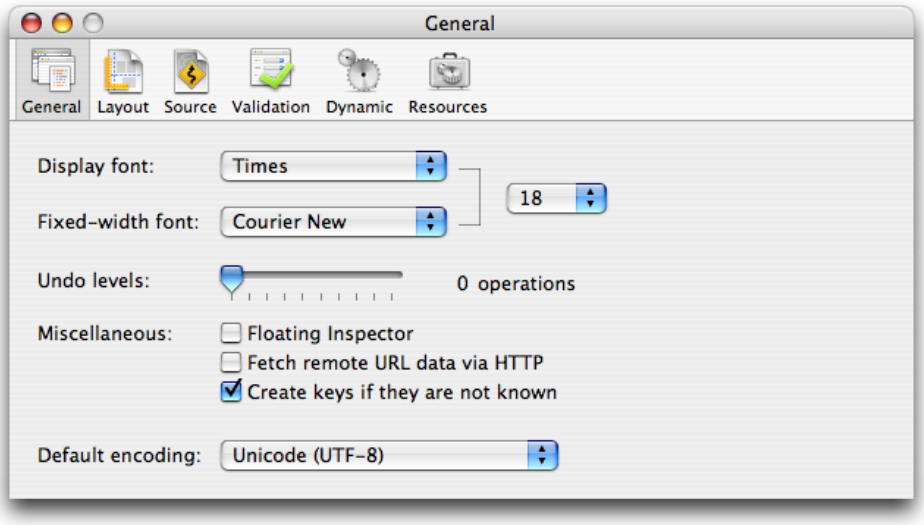
However, using the Elements & Components window, you click on an element name once to add it to your component. The element is added at the insertion point in either the Layout or Source mode. All of the same editing rules that apply to selecting elements from the menus apply to the Elements & Components window. Read "Editing Components" (page 23) for details on these editing rules.

Just drag elements from the Static, Dynamic, and Forms folders to the Favorites folder. You can use the Favorites folder at the top of the window to quickly locate frequently used elements.

# The Preferences Window

There are many ways to customize the views and behavior of WebObjects Builder. Choose Preferences from the application menu to open the Preferences window as show in Figure 1-10.

**Figure 1-10** The Preferences window




The preferences window has three panes:

- **General preferences** apply to the entire application and include preferences for setting display font, undo, and encoding attributes.
- **Layout preferences** affect the appearance of the graphical view in Layout mode. Use these preferences to set colors and turn graphical tags on and off. Read "Changing the Appearance of the Graphical Editor" (page 31)for details on these preferences.
- **Source preferences** affect the appearance of the HTML source editor, HTML source generation, and text editor containing the `.wod` file. Read "Changing the Appearance of the HTML Source Editor" (page 32) for details on the Fonts & Colors preferences, "Reformatting HTML" (page 34) for details on the HTML Generation preferences, and "Changing the Appearance of the WOD Text Editor" (page 34) for details on the Dynamic Element Definitions preferences.
- **Validation preferences** affect the behavior of the validation functions. Read "Validating" (page 109) for more information on validation and these preferences.
- **Dynamic preferences** apply to the graphical view in the Layout mode. Read "Setting Dynamic Element Preferences" (page 80)to learn more about using dynamic elements.
- **Resources preferences** apply to files that you import into your web component. Read "Adding Elements From the File System" (page 25)for details on these preferences.

# The Menu Bar

There are menu and keyboard equivalents, located on the menu bar, for the toolbar buttons, toolbar pop-up menus and the Elements & Components window described earlier in the chapter. The toolbar and Elements & Components window are designed for your convenience. Optionally, you can use the menus on the menu bar.

The menu bar also contains the typical File, Edit, Window, and Help menus. (Power users may wish to memorize the keyboard equivalents for some of these menu items.) These are the menus in the WebObjects Builder menu bar:

- The **File menu** contains commands for creating and opening web components. Read "Creating Web Components" (page 13) for details on using this menu.
- The **Edit menu** contains typical menu items for undo, cut, copy and paste, searching, and spell checking. You can also use this menu to edit your source and switch between views. Read "Modifying the Web Component Interface" (page 53) for details on the Edit Source submenu.
- The **Format menu** contains menu items to format text, tables, and frames. Read "Editing Text in Layout Mode" (page 30), "Editing Tables" (page 38), and "Editing Frames" (page 43) for details on using these menu items.
- The **Static menu** is equivalent to the Static pop-up menu on the toolbar. The same elements are also available in the Elements & Components window. Read "Working With Static Elements" (page 45) for how to use static elements.
- The **Dynamic menu** is equivalent to the Dynamic pop-up menu on the toolbar. The same elements are also available in the Elements & Components window. Read "Working With Dynamic Elements" (page 51) for details on these elements.
- The **Forms menu** is equivalent to the Forms pop-up menu on the toolbar. The same elements are also available in the Elements & Components window. Read "Using Dynamic Form Elements" (page 62) for details on using form elements.
- The **Window menu** contains commands for opening the tools and windows described in this chapter.
- Use the **Help menu** to access WebObjects Builder documentation.

# Editing Components

You can edit your web component in WebObjects Builder using either the graphical editor in Layout mode or the HTML source editor in Source mode. For your convenience, you can easily switch from Layout mode to Source mode. Read "Getting Started With WebObjects Builder" (page 13) for a description of the component window, commands, menus and different types of modes. This chapter describes how to add elements to your component and use other basic editing functions.

## Types of Elements

A dynamic web page or part of a web page is represented by a web component. A web component consists of elements, some static, some dynamic, some with HTML counterparts, and others that are completely unique to WebObjects.

WebObjects Builder allows you to add most of the common HTML elements to a component by using its graphical editing tools. You can type text directly into a component window and you can add additional HTML elements using the buttons and menus in WebObjects Builder. HTML elements are called **static elements**.

In addition, you can add dynamic elements, whose look and behavior are determined at runtime, to a web component. Dynamic elements that have HTML counterparts are called **concrete elements**.

You can also use **abstract elements** in a web component that do not have an HTML counterpart but control the generation of HTML content. Conditionals and repetitions are examples of abstract elements.

Elements that contain other elements are called **container elements**. Container elements can be static or dynamic.

WebObjects also provides a wealth of reusable components on palettes. Components represent partial web pages and are constructed from one or more elements. You can create your own components specific to your enterprise and add them to a custom palette.

## Inserting Elements

You can add a variety of elements to your web component including static HTML elements, dynamic elements, and other form elements. Read "Working With Static Elements" (page 45) and "Working With Dynamic Elements" (page 51) to learn more about specific types of elements. The steps for adding these elements to your component are the same:

1. Switch to Layout or Source mode.
2. Move the insertion point to where you want to insert an element.

3. Choose the element from either the menus in the menu bar, the toolbar, or the Elements & Components window. For example, you can choose WOString from the Dynamic folder in the Elements & Components window (choose Windows > Elements & Components to open the window).

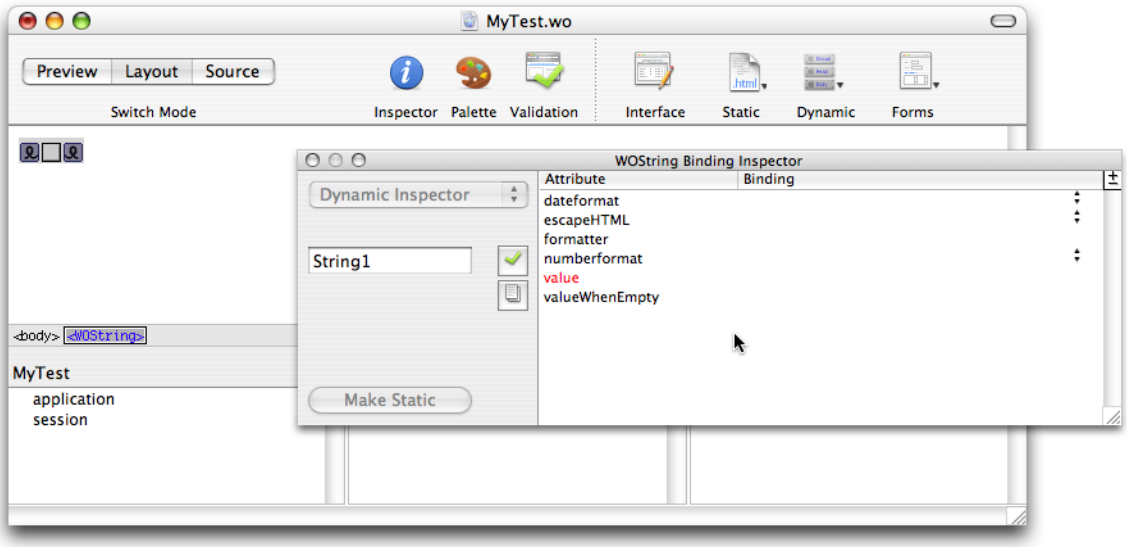
   The element appears at the insertion point. Typically, the new element is selected unless it is a container element. If it is a container element, the default content is selected or the insertion point is inside the container, so you can begin typing the contents.

4. If the element is not selected, select it by clicking it in either the graphical or path view in Layout mode, or selecting the HTML in Source mode. (Read "Selecting Elements in Layout Mode" (page 29) and "Selecting Elements in Source Mode" (page 32) for details.)

5. Open the inspector window by clicking the Inspector button in the toolbar or choosing Window > Inspector.

   Use the inspector to set element-specific properties and bind dynamic elements. For example, you can change a paragraph's type from plain to preformatted or bind the `value` attribute of a WOString to an application variable as shown in Figure 2-1.

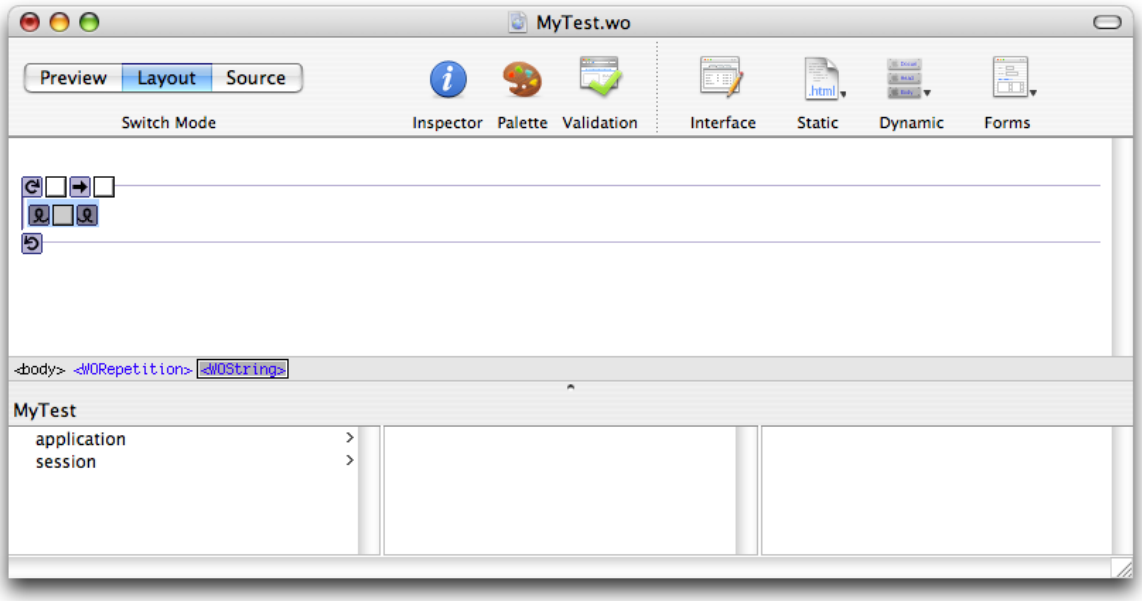
**Figure 2-1** Binding value attribute of a WOString




This is the behavior if you have text or other elements selected when you add a new element:

- If the new element is a container element, the selected elements are wrapped or contained inside the new element. For example, Figure 2-2 shows the result of selecting WORepetition when a WOString element is selected—the WOString is wrapped with a WORepetition.

**Figure 2-2** Wrapping a WOString with a WORepetition




- If the new element cannot contain other elements (for example, a horizontal rule or image), the new element *replaces* the selection.

# Adding Elements From the File System

Some elements can be added to your component by dragging them from the Finder or the desktop into your component window. Types of objects that can be dragged into a web component are:

- Components
- Client-side Java components
- EO model files (of type `.eomodeld`)
- Image files and image maps

Certain file types (such as `.gif`, `.jpeg`, `.tif`, `.eps`, and `.bmp`) are automatically recognized by WebObjects Builder. Use the WebObjects Builder Preferences window to view and edit the list of filename extensions that WebObjects Builder accepts. You can drag any item with one of those extensions into a component window, and the item will be added to your project. If the extension of your file is in the Application Resources table, Xcode adds the file to the Resources group in your project. If the extension of your file is in the Web Server Resources table, Xcode adds the file to the Web Server Resources group in your project. Read *WebObjects Deployment Guide Using JavaMonitor*for a description of these different groups.

If you need to, you can add additional file types to the lists as follows:

1. Choose WebObjects Builder > Preferences.

   The WebObjects Builder Preferences window appears.

2. Click the Resources button.

   The preferences window displays a table containing the application resources and another table containing the web server resources.

3. Click the plus button below either the Resources or Web Server Resources table to add an extension.

   A blank row is inserted into the table.

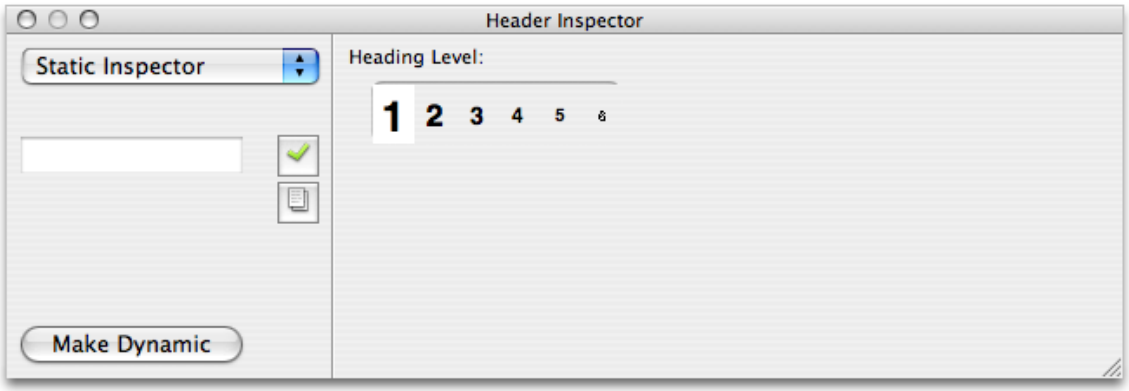
4. Double-click the new row and enter the new filename extension.

# Inspecting Elements

You use the inspector to configure individual elements in your component.

Select an element and open the inspector to view its properties. To open the inspector, click the Inspector button on the toolbar in the component window, choose Window > Inspector, or choose Inspect from an element's contextual menu (Control-click an element to display its contextual menu). For example, if you select a Heading element and click the Inspector button on the toolbar, the Header Inspector window appears as shown in Figure 2-3.

**Figure 2-3** The Header Inspector




The inspector window title and contents reflect the selected element. Each element has its own inspector that allows you to set properties appropriate for that element. For example, the Heading Inspector in Figure 2-3 (page 26) allows you to set the level of a heading element. Other elements have different properties that you can set.

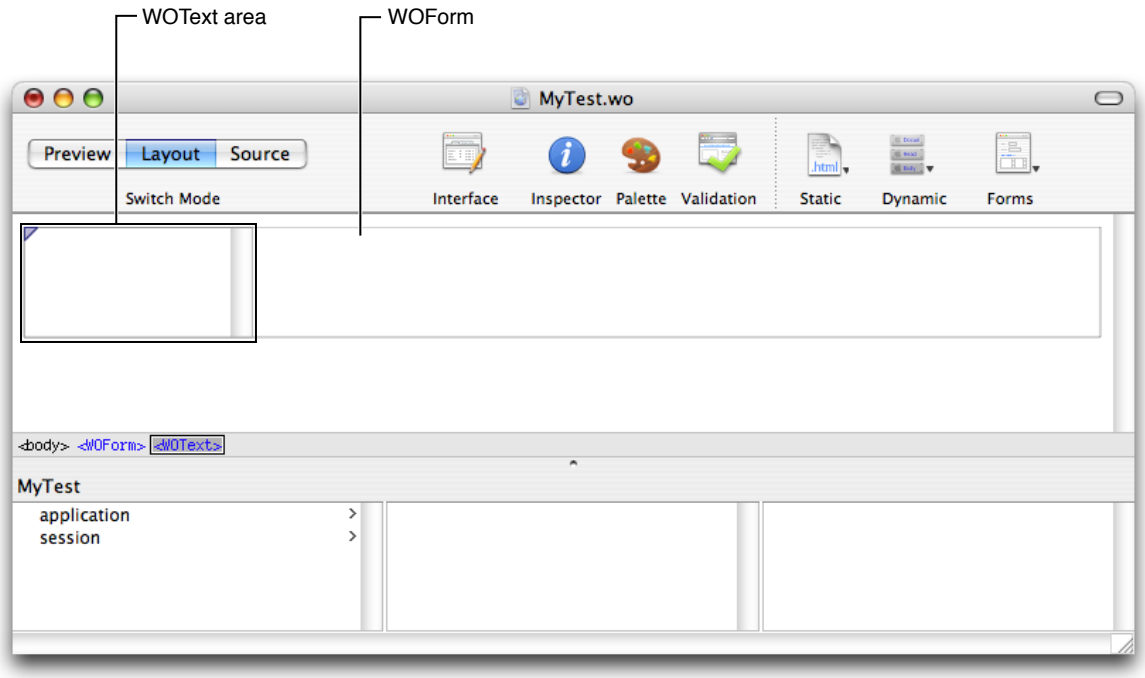
Most dynamic elements have generic, dynamic, and static properties that you can set. You can set the static properties of concrete dynamic elements that have HTML counterparts. You can also change a dynamic element to a static element and vice versa as explained in "Making Elements Dynamic or Static" (page 28).

## Setting Dynamic and Static Properties

The pop-up menu in the upper-left corner of an element inspector is used to switch between generic, static, and dynamic properties. This menu may be disabled if an element does not have static properties.

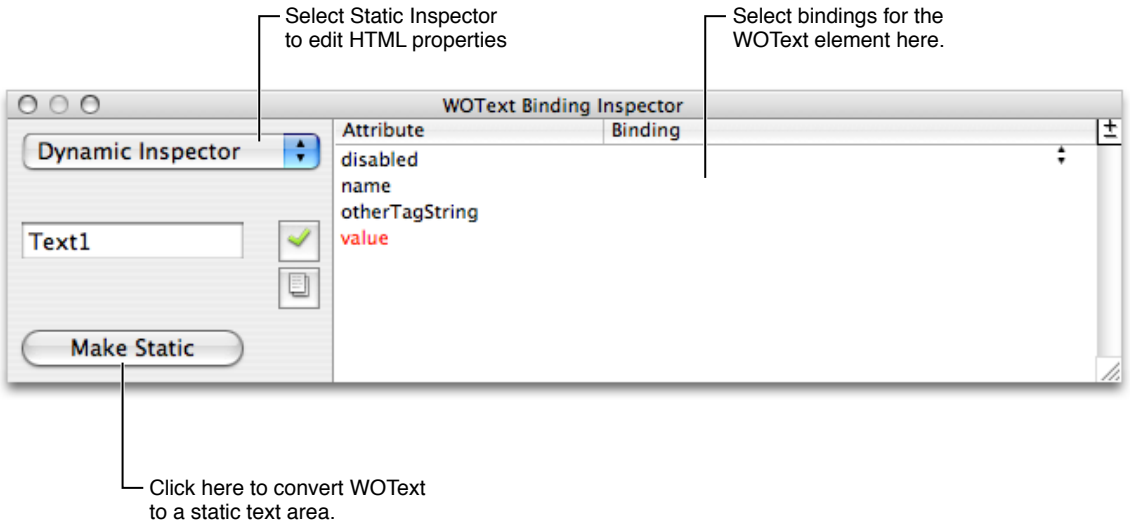
For example, choose Forms > WOForm to create a form element, and choose Forms > WOTextArea to insert a text area element in the form element as shown in Figure 2-4.

**Figure 2-4** Adding a WOTextArea element



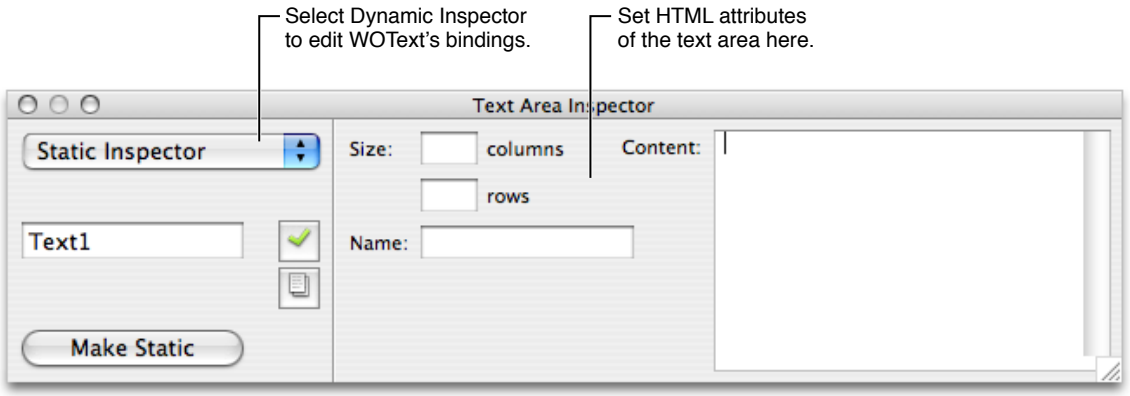

Click inspector on the toolbar to open the inspector for the dynamic text area element. The WOText Binding Inspector displays the attributes that you can bind for this element as shown in Figure 2-5.

**Figure 2-5** A Dynamic element Inspector




If you choose Static Inspector from the pop-up menu, the Text Area Inspector appears as shown in Figure 2-6. This is the same inspector you would see for a static text area element (`<textarea>`) and allows you to set its HTML properties (such as the `COLS` and `ROWS` properties). You can also add bindings to the dynamic inspector to set HTML properties. (The static inspector is provided for your convenience only.)

**Figure 2-6** A Static element Inspector




Choose Dynamic Inspector from the pop-up menu to switch back to the WOText Binding Inspector.

## Making Elements Dynamic or Static

A static element has a Make Dynamic button in the lower-left corner that converts it to the corresponding dynamic element. For example, if you click the Make Dynamic button in an Anchor Inspector window, the selected element is converted to a WOHyperlink dynamic element. Similarly, a concrete dynamic element has a Make Static button in its inspector that converts it to its HTML counterpart.

The following table shows the dynamic counterparts for each of the static HTML elements shown:

| | |
|---|---|
| Image | WOImage, WOActiveImage |
| Form | WOForm |
| Textfield | WOTextField |
| Text area | WOText |
| Button | WOSubmitButton, WOResetButton, WOImageButton |
| Checkbox | WOCheckBox |
| Radio button | WORadioButton |
| Select | WOBrowser, WOPopupButton |
| Hyperlink | WOHyperlink |
| Applet | WOApplet |
| Other | Generic WebObjects |

If you convert a static element to its dynamic counterpart by clicking Make Dynamic and no direct counterpart exists, the element becomes a WOGenericElement whose element name is the HTML tag for the static element. Read "Generic WebObjects Elements" (page 71)for more information on generic elements.

# Working in Layout Mode

Typically, you use the graphical editor in Layout mode to create your web component. This editor is easy to use and behaves like the editors in other Apple applications. However, because it is more than an HTML editor—you edit dynamic WebObjects elements too—there are some different features.

## Selecting Elements in Layout Mode

There are several operations you can perform in WebObjects Builder that require you to select an element, such as copying, deleting, inspecting, or wrapping elements (placing one element inside another).

You select text elements in the graphical view in Layout mode as you would in most text-editing applications: by dragging, by double-clicking words, by triple-clicking lines, or by Shift-clicking to make a multiple selection. The selected text appears highlighted.

Some elements, such as text fields and text areas, can be selected simply by clicking them.

Other elements, such as tables, require you to click outside the element and then drag across the element in order to select it.

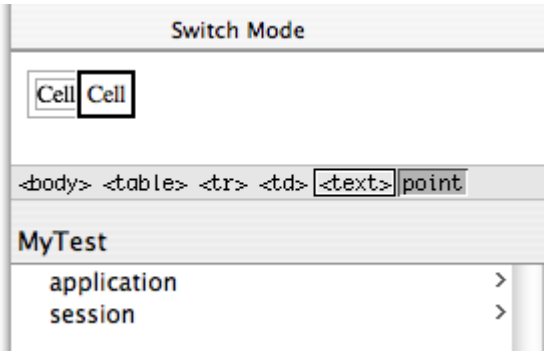
To select a range of elements, drag across them, or press the Shift key while clicking at the end of the range.

In Layout mode, you can use the path view to verify your selection or to select the parent element. The selected element appears highlighted at the end of the path. You select elements and their containing elements by clicking the HTML tag corresponding to the parent element. The selected element is highlighted in the graphical view.

For example, if you want to select a particular row in a table, click anywhere in the row and the path view displays the enclosing elements as show in Figure 2-7. Click the `<td>` tag to select the entire cell or the `<tr>` tag in the path view to select the entire row.

Read "Editing Tables" (page 34) for more tips on selecting table parts

**Figure 2-7** Selecting a table row




## Editing Text in Layout Mode

If you choose New from the File menu to create a new web component, the cursor appears in the upper-left corner of the graphical view in Layout mode. If you begin typing text directly into this pane, the text appears in the default font and size. Here are some convenient keyboard shortcuts for editing text:

- Press Shift-Return to insert a line break, `<br>` tag.
- Press Return to insert a paragraph element, `<p>`tag.

You use the Format menu to format the selected text. If there is no selection, the format changes apply to any text typed at the insertion point. Items in the Format menu have direct HTML equivalents. You can switch to Source mode to check the formating changes you make to the HTML file, and switch to preview mode to see the effect of those changes.

The following Format > Font submenu items make these HTML edits:

- Choose Show Fonts, select a font from the list, and click Apply to change the font of the current selection—wrap the selection with a Font element, `<font>`.
- Choose Bold to change the style to bold—wrap the selection with a bold element, `<b>`.
- Choose Italic to change the style to italic—wrap the selection with an italic element, `<i>`.
- Choose Underline to change the style to underline—wrap the selection with an underline element, `<u>`.
- Choose Typewriter to change the style to typewriter (fixed-width font)—wrap the selection with a typewriter element, `<tt>`.

The Format > Alignment submenu items allow you to center or justify text by setting the align property of the enclosing paragraph. For example, if you choose Format > Alignment > Center when the insertion point is on a paragraph element, the paragraph element's align property is set to `center`.

The Format > Size submenu items change the font size of selected text—wraps the selection with a font element and sets the font element's size property to the selected size. For example, if you select some text and choose Format > Size > 6, the selection is wrapped with a font element with the size set to `6`. In the HTML source, `<font size="6">` is inserted before the selection and `</font>` is inserted after the selection.

The Format > Table submenu is used to format tables. Read "Editing Tables" (page 38) for details on using tables.

The Format > Frame submenu is used to format frames. Read "Editing Frames" (page 43) for details on using frames.

Choose Window > Show Colors to open the Colors window and change the color of the selection—wrap the selection with a font element and set the color property. The colors can also be applied to entire pages and hyperlinks.

Choose Format > Indent to indent the selection—wrap the current selection with a block quote element, `<blockquote>`. Select Format > Outdent to move indented text to the left—remove the enclosing block quote element.

## Using Contextual Menus

Most elements displayed in the graphical and path views in Layout mode have contextual menus that are displayed when you Control-click the elements or a selection. The contextual menus are element specific. For example, the contextual menu of a table allows you to split cells and add or remove columns and rows as discussed in "Editing Tables" (page 38). Container elements have menu items for deleting and unwrapping their contents.

Objects in the object browser also have contextual menus that are accessed by Control-clicking the object. These contextual menus are discussed in "Binding Dynamic Elements" (page 51).

## Changing the Appearance of the Graphical Editor

You can also change the appearance of the graphical editor by changing the preferences. For example, you can turn on table tags in the graphical view to make it easier to bind dynamic elements in tables or wrap table cells or rows with a repetition as discussed in "Wrapping Table Rows and Cells With Repetitions" (page 78). To turn on the table tags, open the Preferences window, click the Layout button, and select tables from the "Show inline graphical tags for" options.

# Working in Source Mode

The Source mode displays the `.html` file in an HTML source editor and the `.wod` file for your component in a text editor. If you are familiar with HTML syntax, you might prefer using the HTML source editor to change your component. Otherwise, you can continue to edit your component using the graphical view in Layout mode and view your changes in Source mode. The editing commands and preferences are different in Source mode.

## Selecting Elements in Source Mode

You select text elements in the HTML source editor as you would in most text-editing applications: by dragging, by double-clicking words, or by Shift-clicking to make a multiple selection. Selected text appears highlighted.

When you triple-click a tag in the HTML source editor, you select the tag, the matching tag, and the HTML in between. For example, if you triple-click `<title>` in a new page, WebObjects Builder selects `<title>`, `</title>`, and the "Page Title" text. You can also drag a tag after it has been selected to move the entire section of HTML source to a new location.

The bottom pane in Source mode displays your component's declarations file (a `.wod` file). This file stores the binding information when you bind variables and methods to your dynamic elements. Editing is enabled in this window but normally, you don't edit this file directly. Read "Binding Elements" (page 57) for how to bind elements in Layout mode.

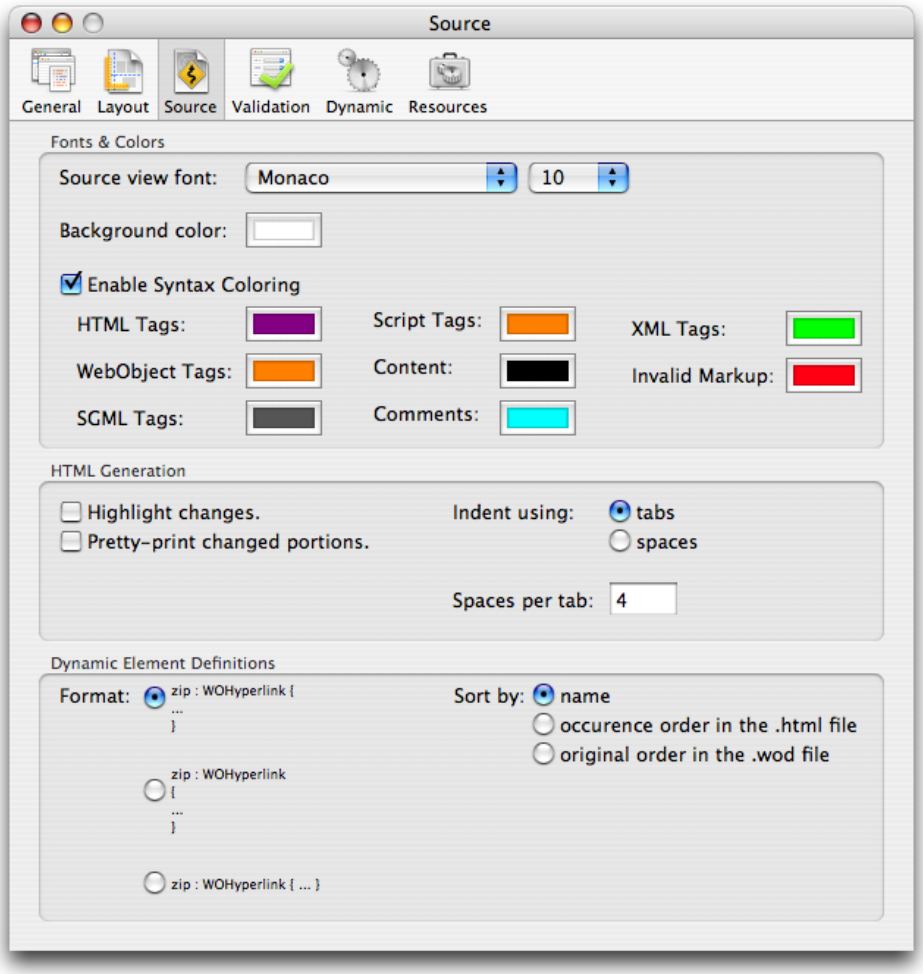
## Changing the Appearance of the HTML Source Editor

WebObjects Builder automatically inserts necessary elements such as `<html>`, `<head>` and `<body>` for you. However, the HTML source file is not an ordinary HTML file. It can contain static, dynamic, concrete, and abstract elements specific to WebObjects. To distinguish these different types of elements, you can change the colors of the tags and text in Source mode using the Preferences window as follows:

1. Choose WebObjects Builder > Preferences.
2. Click Source.

The Source preferences pane appears as show in Figure 2-8.

**Figure 2-8** The Source preferences




3. Select Enable Syntax Coloring to turn on the colors in Source mode.

   The default colors appear in the HTML source editor.

4. Set the desired colors for the tags and text by clicking the specific color wells for each tag.

5. If you want to turn off the colors, deselect Enable Syntax Coloring.

   The HTML in the source editor appears in black text with a white background.

Similarly, use the "Background color" colorwell to change the background color in the HTML source editor. You can also choose a different font and font size from the "Source view font" pop-up menu.

## Reformatting HTML

To reformat HTML entered in the HTML source editor, choose Format > Reformat HTML. This menu item is enabled only in Source mode. Choosing this menu item reformats the HTML source for your component using HTML preferences you can change, such as number of spaces per tab.

To change Source mode preferences, open the Preferences window and click the Source button. The HTML preferences are in the HTML Generation box, as shown in Figure 2-8 (page 33), and allow you to highlight changes, pretty-print changes, and specify tabs or spaces for indentation.

## Changing the Appearance of the WOD Text Editor

You can also change the appearance of declarations in the `.wod` file that appear in the lower pane in Source mode. Typically, you do not edit the `.wod` file directly, but you might want to format it differently and change the order of declarations. Open the Preferences window and click the Source button. The `.wod` file preferences are in the Dynamic Element Definitions box, shown in Figure 2-8 (page 33), and allow you to select a format style and sort order.

# Editing Tables

Tables are a common tool for organizing and formatting large amounts of information on a webpage. Tables can be used to implement menu bars and even incrementally display large graphics in small pieces. This section describes how to create, edit, and format tables using WebObjects Builder.

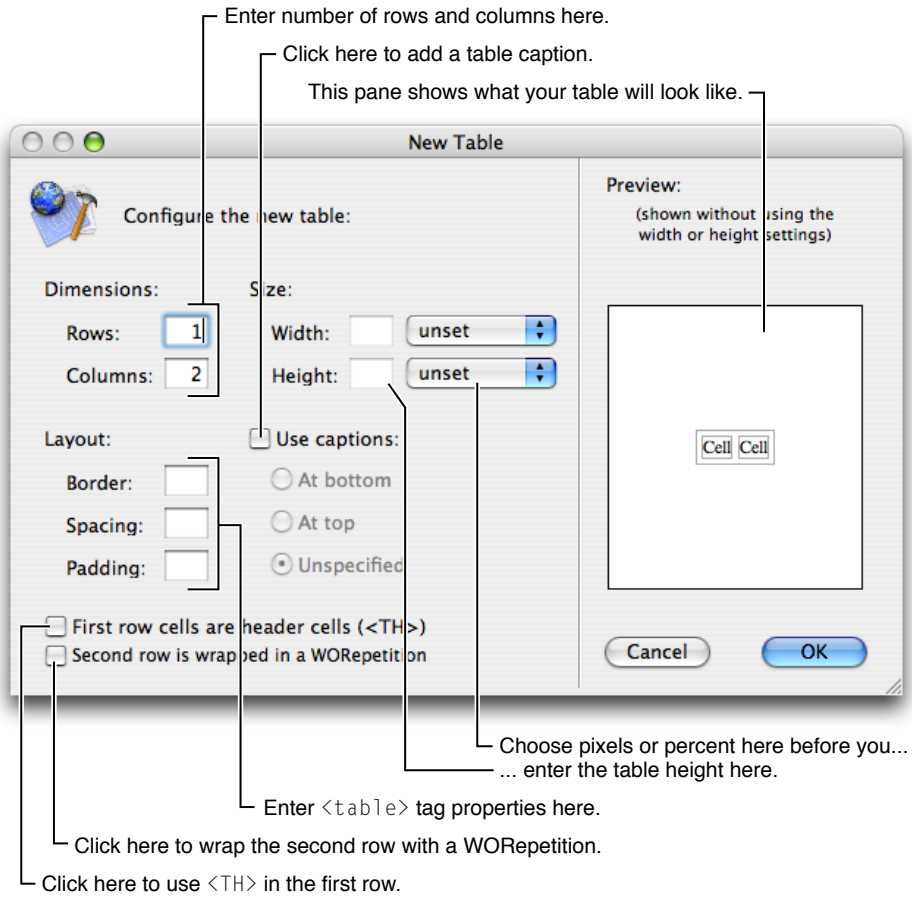
## Creating Tables

Follow these steps to create a table:

1. Select Table from the Static menu to add a table at the insertion point.

The New Table panel appears allowing you to configure the table before you insert it as shown in Figure 2-9.

**Figure 2-9** Creating tables




2. Use the Dimensions fields to change the number of rows or columns in the table.

   You can add rows and columns later but its more convenient to to use the New Table panel. When you are done entering the dimensions, the Preview pane in the New Table panel shows approximately what your table looks like.

3. Optionally, select a unit for the width (pixels or percent) in the Width pop-up menu, and enter the table width in the Width text field. Similarly, set the table height.

4. To add a caption to the table, select "Use captions", and set the location of the caption by clicking either "At bottom", "At top" or Unspecified.

   If you select "Unspecified" then the user browser decides where to put the caption.

5. To change the table's border, spacing (corresponding to the HTML `cellspacing` table property), and padding (corresponding to the HTML `cellpadding` table property), enter the appropriate sizes (in pixels) in the Layout fields.

6. To add a header to the table, click the checkbox labeled "First row cells are header cells."

The first row cells are wrapped in an HTML `<th>` tag instead of a `<td>` tag, which causes the cells to appear in bold in most browsers.

7. To create a table whose contents depend on a repetition, click the checkbox marked "Second row is wrapped in a WORepetition."

   The Preview pane renders the repeated row in blue. See "Using Abstract Dynamic Elements" (page 73) for more information on repetitions.

8. Click OK to insert the table; otherwise click Cancel.

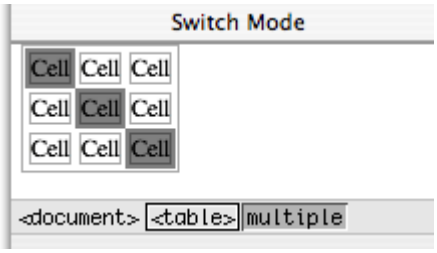
The PremadeElements palette contains some common tables that are wrapped in repetitions—tables that can expand in both the horizontal and vertical directions. Read "PremadeElements Palette" (page 97) for details on these tables.

## Selecting Table Parts in the Graphical View

Typically, you select a cell, row, or multiple parts of a table and open the inspector to edit its properties. For example, you might select a cell and use the inspector to change the cell's width or height. In the graphical view, you can use a combination of the mouse and keyboard to select and create table parts:

- To begin entering content in a cell, click the cell and enter the content.
- To add content to a cell, type text in the cell or add elements—for example, select an element from one of the toolbar menus.
- To move to the cell to the right, or to the first cell on the next row (if you are in the rightmost column), press Tab.
- To create a new row when you are at the end of a table, press Tab.
- To move backwards through a table (for example, select the cell to the left), press Shift-Tab.
- To move to the cell at the end of the table when you are at the beginning of a table, press Shift-Tab.
- To make a contiguous multiple cell selection, click in a cell and drag across adjacent cells, or select a cell and Shift-click another cell.
- To make a noncontiguous multiple cell selection, Command-click multiple cells as shown in Figure 2-10.

**Figure 2-10** Selecting noncontiguous multiple cells

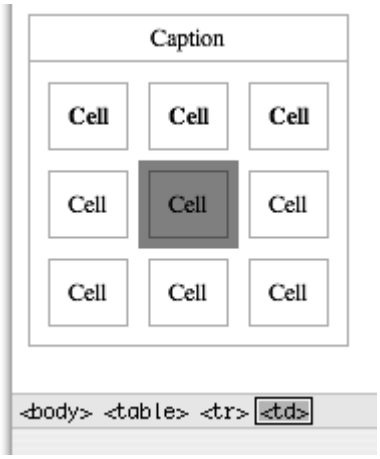


## Selecting Table Parts in the Path View

It may be more convenient to use the path view to select table parts for editing. An HTML table (a `<table>` tag) is a hierarchical structure that contains rows (`<tr>` tags). Rows in turn contain cells (`<th>` or `<td>` tags). When you select a cell, the path view shows the path from the cell up to the page (a `<body>` tag). You can inspect any element in the path by clicking its tag in the path view. For example, if you select a table cell, you can inspect the cell by clicking the `<td>` tag, the row by clicking the `<tr>` tag, or the table by clicking the `<table>` tag.

Figure 2-11 shows a table in graphical view with the cell tag, `<td>`, selected in the path view.

**Figure 2-11** Selecting a cell using the path view

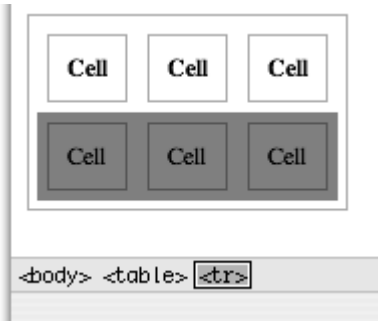



## Selecting Rows and Tables

Selecting all the cells in a row or table is not the same as selecting the row or table. You need to make selections in both the graphical and path views to select an entire row or table.

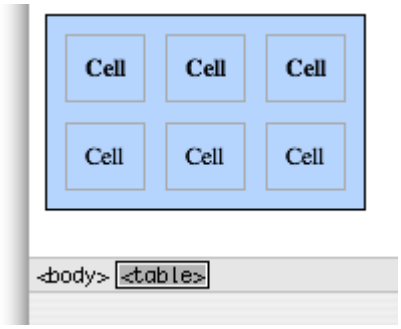
- *To select a row*, select one or more cells in a row and click the `<tr>` element in the path view, as shown in Figure 2-12.

**Figure 2-12** Selecting a row



- *To select a table*, select one or more cells or rows in the table and click the `<table>` element in the path view, as shown in Figure 2-13.

**Figure 2-13** Selecting a table

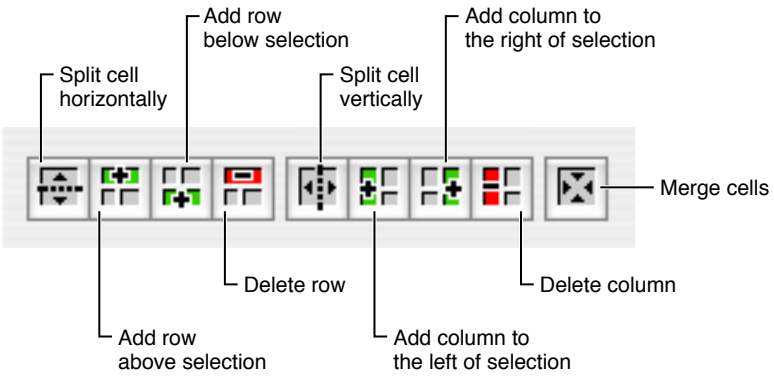



## Editing Tables

There are several ways to edit tables: split and merge cells, add and delete columns, and add and delete rows. These actions are the same whether you choose a menu item or click a button to execute them.

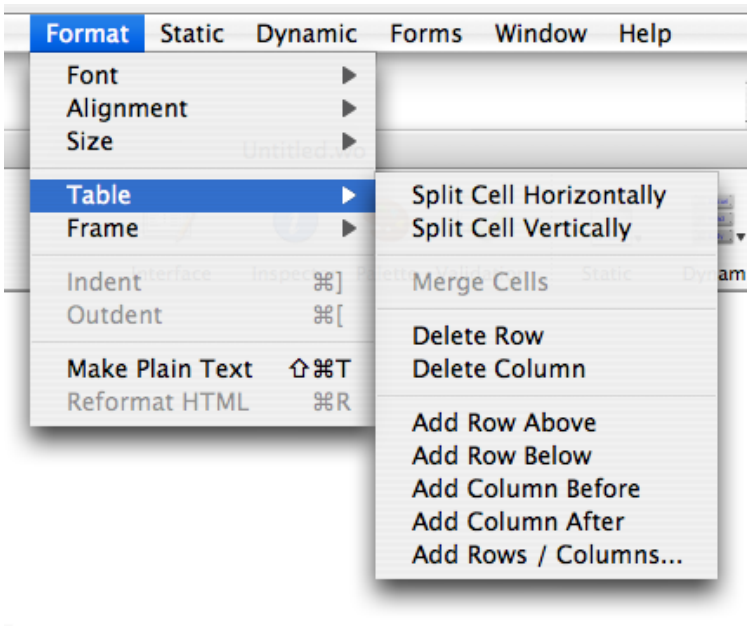
- The most convenient method is to select a table, row or cell and use the control buttons on the Inspector, show in Figure 2-14, to edit the table.

**Figure 2-14** Editing a table




- Otherwise, you can select a table, row, or cell and use the contextual menu to edit the table (Control-click the element to open the contextual menu).
- Finally, you can choose an item from the Format > Tables menu to change the structure of a table as shown in Figure 2-15.

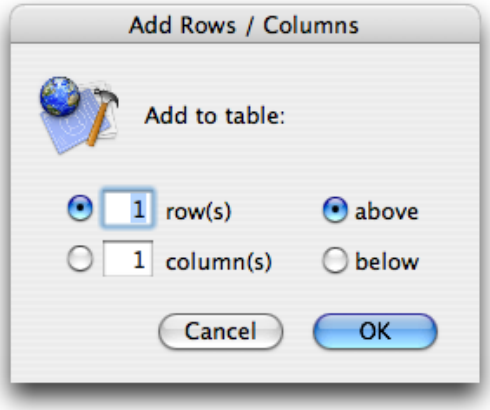
**Figure 2-15** Editing a table using the Table menu




The table editing commands below are in the Format > Table menu.

- To split a cell, select it and choose Split Cell Vertically to split it vertically. Choose Split Cell Horizontally to split it horizontally.
- To merge multiple cells, select a contiguous region of cells and choose Merge Cells. This menu item isn't enabled unless the selected cells make up a group that could logically be merged into one cell.
- To delete a row, select the row you want to delete (or a cell within that row) and choose Delete Row.
- To delete a column, select a cell within the column you want to delete and choose Delete Column.
- To add a row, select a cell or row and choose Add Row Above to add a row above your selection. Choose Add Row Below to add a row below your selection.
- To add a column, select a cell and choose Add Column Before to add a column to the left of your selection. Choose Add Column After to add a column to the right of your selection.
- To add multiple rows or columns, choose Add Rows / Columns. The Add Rows / Columns panel appears as shown in Figure 2-16. Select row(s) or column(s), enter the number of rows or columns, select above or below for rows and before or after for columns (when you select column(s) the radio button labels on the right change from above and below to before and after). Click OK.

**Figure 2-16** Adding multiple table rows or columns




You modify other properties of a table or part of a table by selecting it and opening the Table, Table Row, or the Table Data Inspector. For example, read "Resizing Tables and Cells" (page 40) for how to change the size of cells and rows.

## Wrapping Rows and Cells

You can wrap a row or cell with an abstract dynamic element such as a conditional or repetition. You use a conditional if you want to optionally display a row or cell. You use a repetition if the content of a table is dynamic.

To wrap an abstract dynamic element around a selected row or cell, select the row or cell, and add a dynamic element. For example, select a row and choose WORepetition from the Dynamic menu. A row or cell that is wrapped appears highlighted. If you click in the row or cell, the dynamic element appears highlighted in the path view, too.

See "Using Abstract Dynamic Elements" (page 73) for more information on configuring conditionals and repetitions.

## Resizing Tables and Cells

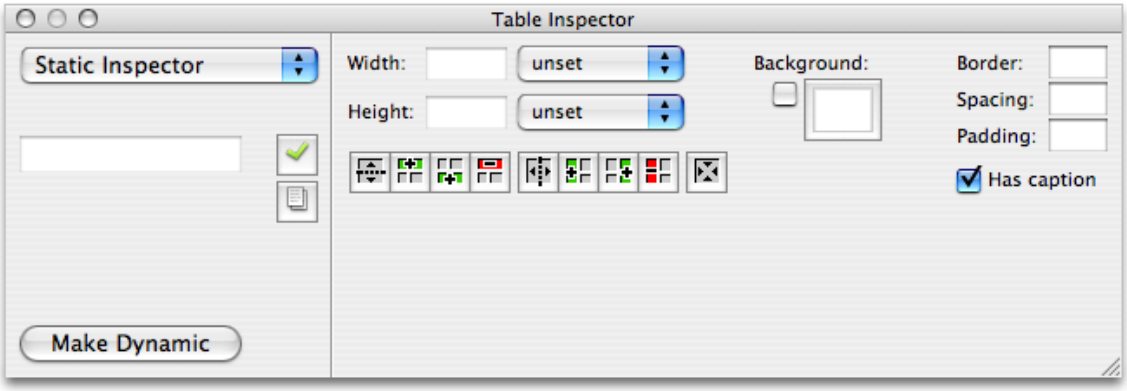
By default, the size of an HTML table is determined by the contents of the table's cells. If you type text (or insert other elements) inside a table cell, the cell's width expands as necessary to fit the data. The width of any column, therefore, will be the width of the widest cell in the column.

In WebObjects Builder, a cell does not resize until you finish editing the cell and either move to another cell or move out of the table. To update the cell immediately, press the Escape key.

Use the inspector to set the size of a cell or table directly:

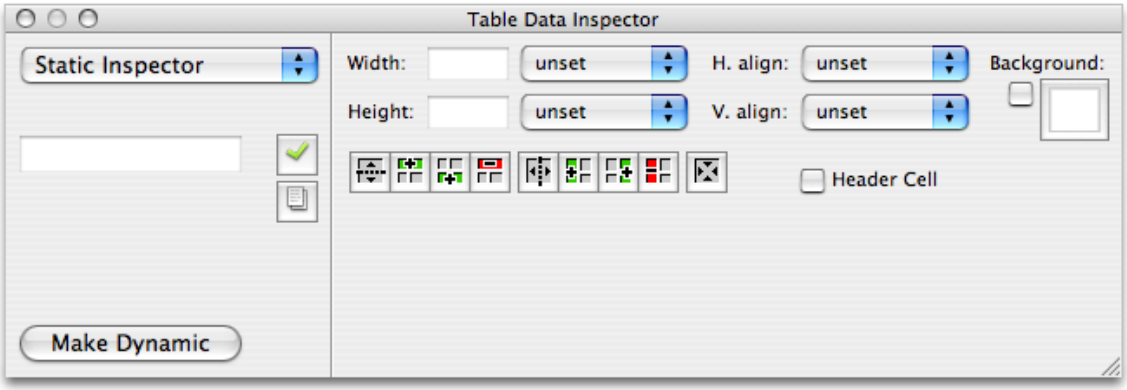
- To set the width or height of a table, select the table and open the Table Inspector as shown in Figure 2-17. Enter the table size in the Width and Height text fields.

**Figure 2-17** Setting table width and height




- To set the width or height of a cell, select the cell and open the Table Data Inspector. Read "Selecting Table Parts in the Path View" (page 37) for how to select a cell. Figure 2-18 shows what the inspector looks like. Enter the cell size in the Width and Height text fields.

**Figure 2-18** Setting cell width and height




**Warning:** WebObjects Builder allows you to set sizing specifications that may be impossible—for example, percentages that add to more than 100% or cells in the same column with different widths. How the tables appear to the user depends on the browser.

## Setting Row and Cell Alignments

Use the inspector to set the row and cell alignment properties as follows:

1. Select the row or cell—for example, click in a cell and select the `<td>` tag in the path view.
2. Open the inspector—for example, click Inspector on the toolbar.

The Table Data Inspector appears if a cell is selected (see Figure 2-18 (page 41)), and the Table Row Inspector appears if a row is selected (see Figure 2-19 (page 42)). Both inspectors contain the same controls for setting the alignment.

3. To set the horizontal alignment, choose left, center, or right from the "Horizontal align" pop-up menu.

   If the value is unset, the browser determines the horizontal alignment.

4. To set the vertical alignment, choose top, middle, bottom, or baseline from the "Vertical align" pop-up menu.

   If the value is unset, the browser determines the vertical alignment.

## Setting Other Table Properties

Use the Table Inspector, shown in Figure 2-17 (page 41), to set remaining table properties, such as the background color, border thickness, spacing, and padding properties. You can also choose to include a caption with your table.

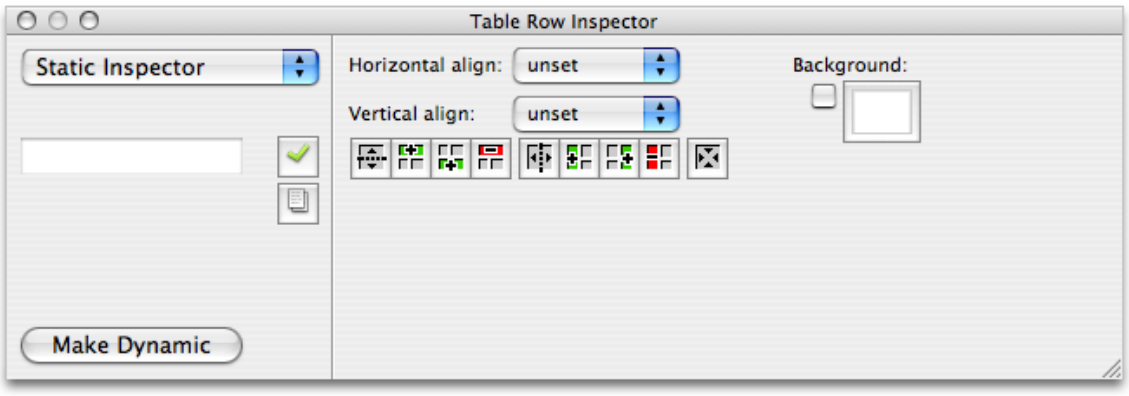
## Setting Other Cell Properties

Use the Table Data Inspector, shown in Figure 2-18 (page 41), to set remaining cell properties, such as the background color of a cell and whether or not a cell is a header cell.

## Setting Other Row Properties

Use the Table Row Inspector, shown in Figure 2-19, to set remaining row properties such as the background color of all cells in a row.

**Figure 2-19** Setting row properties



# Editing Frames

Before editing frames you need to create a new web component based on a frameset. A **frameset** element is a container of frame elements that specifies the layout of the frames. Choose File > New Frameset to create a web component with frames. A new component window appears containing a single `<frame>` element with a root `<frameset>` element

Follow these steps to edit individual frames. Select a frame element inside a frameset and perform one of the following actions:

- To split a frame in two horizontally, choose Edit > Frame > Split Horizontally.
- To split a frame in two vertically, choose Edit > Frame > Split Vertically.
- To specify the frame width, choose Absolute Width, Percentage Width, or Proportional Width from the Edit > Frame menu.
- To specify the frame height, choose Absolute Height, Percentage Height, or Proportional Height from the Edit > Frame menu.
- To connect the frame with a URL, choose Edit > Frame > Connect To Source.

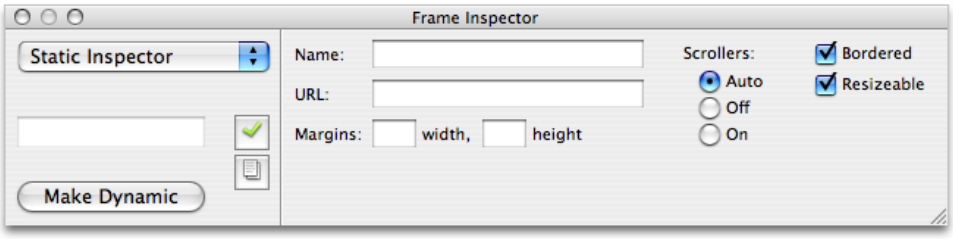
  Select a file from the "Connect frame to" window and click Open.

You can also configure properties using the inspector as follows:

1. Select the frame and click the Inspector button on the toolbar.

   The Frame Inspector appears as shown in Figure 2-20.

   **Figure 2-20** The Frame Inspector




2. Optionally, enter a name in the Name field.
3. Optionally, enter a URL for the initial contents of the frame in the URL field.
4. Optionally, enter the margins, using pixel values, in the width and height text fields.
5. Select Auto to leave the decision to use scroll bars to the browser, select Off to remove scroll bars, and select On to add scroll bars.

   If you select Auto, the browser or user agent on the client side determines whether or not scrollers are used.
6. Select Bordered if you want a border around the frame.

7. Select Resizable to allow the frame to be resized.
8. Click Make Dynamic to convert the frame element into a WOFrame.

   If you use a WOFrame, you can specify the content of the frame dynamically.

# Formatting with Cascading Style Sheets

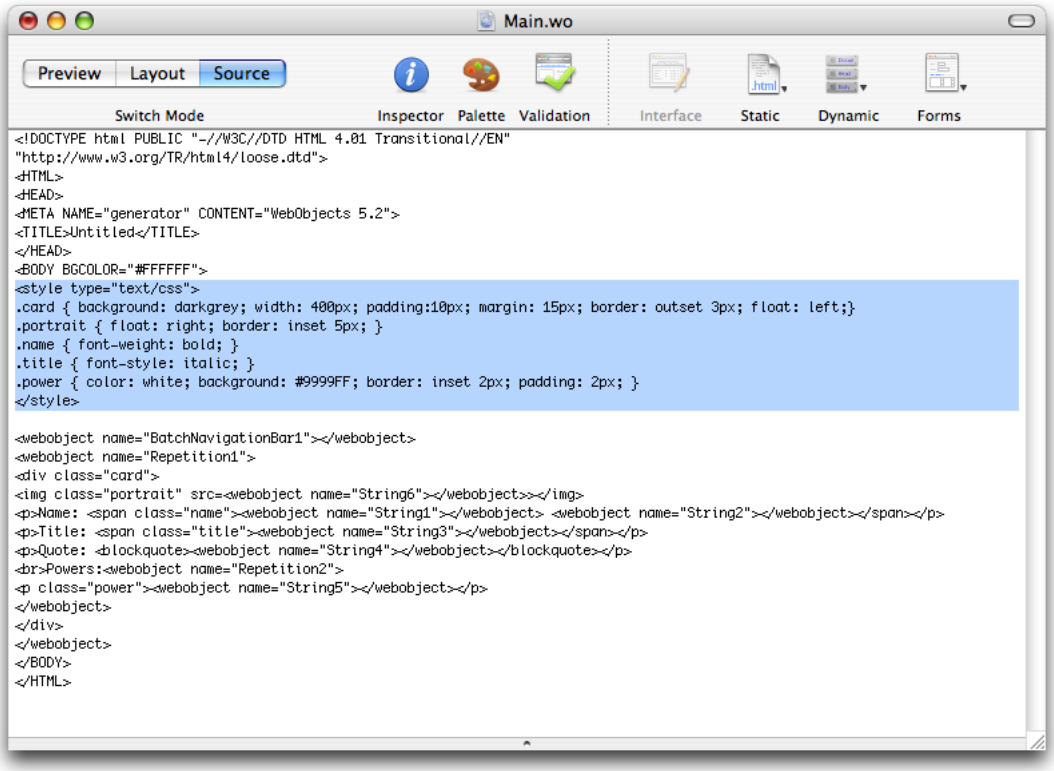
This section describes how to use cascading style sheets (CSS) in your web component to format content.

Choose Static > CSS to add a `<style>` element to your component. The following HTML is added to your component: `<style type="text/css">CSS Content</style>`. Replace `CSS Content` with your style instructions.

For example, Figure 2-21 shows the `Main.wo` component for the TradingCards sample code. The highlighted section specifies the format of elements tagged with the names: `card`, `portrait`, `name`, `title` and `power`. Note the use of the span and division elements in the HTML that appears after the style element that marks sections to apply the formatting.

See www.w3.org/Style/CSS for a complete description of cascading style sheets.

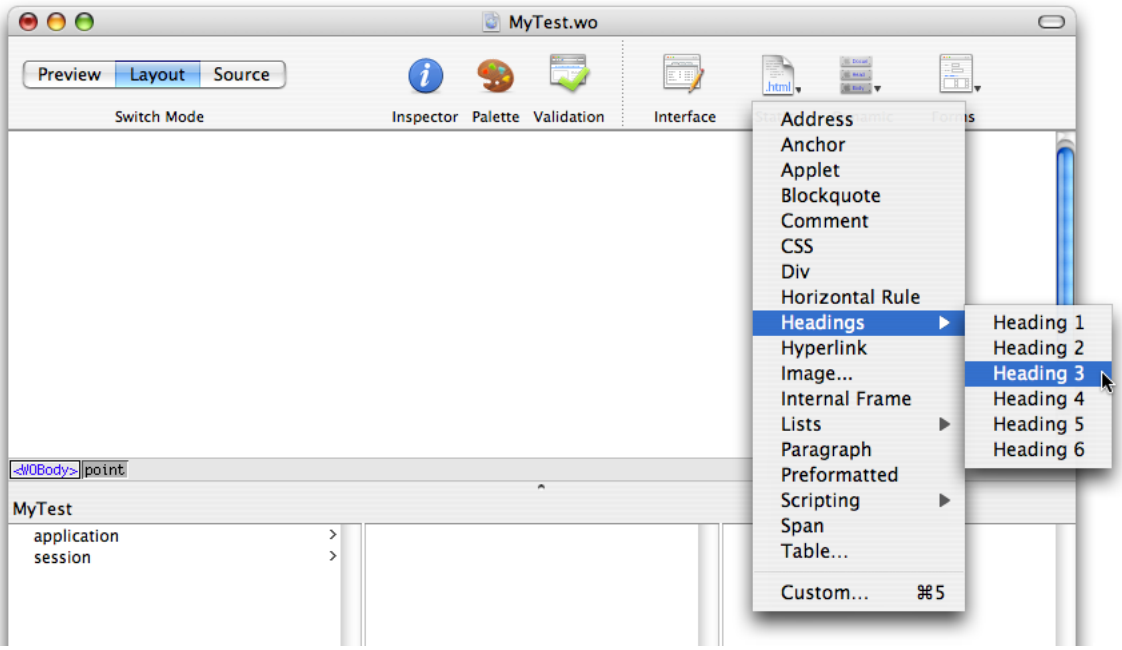
**Figure 2-21** A CSS Sample



# Working With Static Elements

WebObjects Builder allows you to add common static HTML elements to your web component—for example, paragraphs, lists, images, and other static HTML elements that add structure to your web component. You can add static HTML elements using the Elements menu or the Elements pop-up menu on the toolbar in the component window as shown in Figure 3-1.

**Figure 3-1** Static elements menu



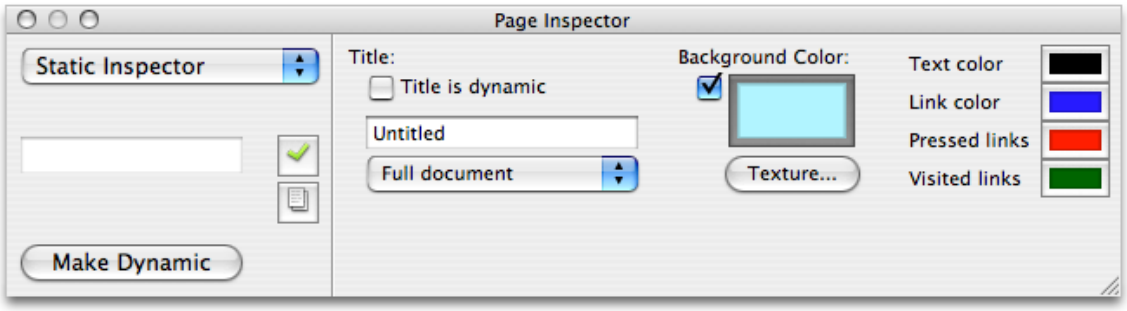

This chapter describes only static HTML elements. Read "Using Dynamic Form Elements" (page 62) if you are creating a form and "Using Other Concrete Dynamic Elements" (page 68) if you want to use other dynamic elements. Read "Editing Tables" (page 34) for details on using tables, and read "Editing Frames" (page 43) for details on using frames.

## Body and Document

The **body element** is the root element used for a full document and the **document element** is the root element used for a partial document. Both elements are containers representing the entire web component. For example, the main component has a body element as the root element. You edit the body attributes to change the overall appearance of your page. However, unlike other elements, the body and document elements don't appear in the Static menu. These elements are automatically added when you create your component.

To inspect a page's attributes, click either the body tag (<body>) or the document tag (<document>) in the path view, and open the Inspector (click the Inspector button on the toolbar). Using the inspector shown in Figure 3-2, you can switch from a full document to a partial document by selecting Full Document or Partial Document from the pop-up menu. This action simply changes the HTML tag from <body> to <document>.

**Figure 3-2** Switching from a full to a partial document




The Title text field allows you to set the title of the document. If you select the "Title is dynamic" checkbox, the title becomes a dynamic string whose value is determined at runtime. You enter its bindings in the Title text field.

You can also set the background color or texture of the page. You can set the colors of specific text objects: text, links, pressed links, and visited links. To change a color, click the border of the color well and then select a color from the Colors window.

The dynamic element counterpart for the body element is WOBody. See "Working With Dynamic Elements" (page 51) for more information on WOBody and binding dynamic elements.

# Address, Blockquote, Division, Paragraph, and Preformatted

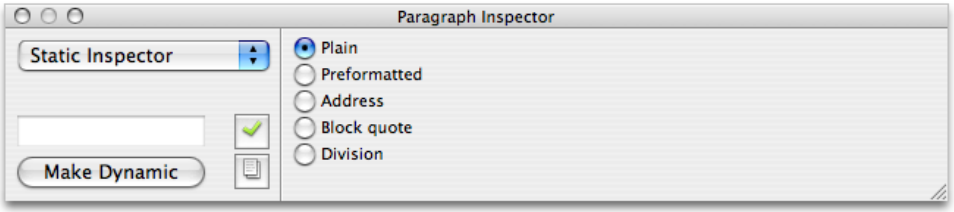
Use the **paragraph element** if you want a plain paragraph. Use the **address element** if you want to wrap author contact information. Use the **blockquote element** if you want browsers to indent a long quotation. Use the **division element** if you want to add structure to your document—browsers usually place a line break before and after a division element. Use the **preformatted element** if you don't want browsers to modify the format of your paragraph.

To add a new paragraph element (<p>), choose Paragraph from the Static menu or simply press Return in the graphical view in Layout mode. If there is a selection, it is wrapped in a paragraph element.

To add an address, blockquote, division, or preformatted element choose the element from the Static menu. If there is a selection, it is wrapped in the chosen element.

The address, blockquote, division, paragraph, and preformatted static elements are all related and share the same Paragraph Inspector shown in Figure 3-3. Select the element and click Inspector on the toolbar to open the Paragraph Inspector.

**Figure 3-3** The Paragraph Inspector




You can use the Paragraph Inspector to convert any of these elements to the following elements with the corresponding HTML tags:

- Plain (`<p>`)
- Preformatted (`<pre>`)
- Address (`<address>`)
- Blockquote (`<blockquote>`)
- Division (`<div>`)

Read "Formatting with Cascading Style Sheets" (page 44) for examples of using division element with cascading style sheets.

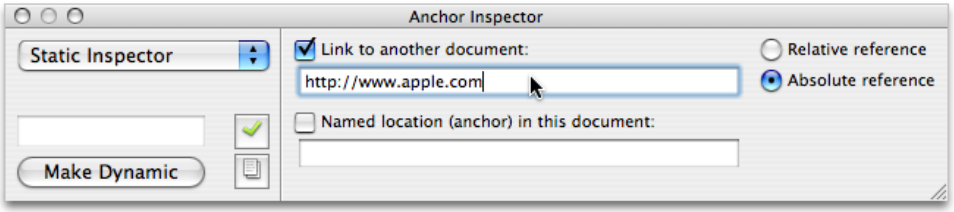
# Anchors and Hyperlinks

An **anchor** is a named location within a document that can be referenced by a hyperlink. A **hyperlink** links to another location. Anchors and hyperlinks are both represented by an **anchor element**, `<a>`, and have the same Anchor Inspector.

Choose Anchor or Hyperlink from the Elements menu to add an anchor element. If text is selected, it is wrapped in an anchor element. Otherwise, you can begin typing the text you want contained in the anchor element. The text is underlined as you type.

If you are adding a hyperlink, open the Anchor Inspector as shown in Figure 3-4. Select the "Link to another document" checkbox and enter the destination of the link (the URL) in the text field below the checkbox. You can also specify if the path is relative or absolute by clicking the "Relative reference" or "Absolute reference" radio buttons.

**Figure 3-4** The Anchor Inspector



If you are adding an anchor, select the "Named location (anchor) in this document" checkbox, and enter the named location within a document in the text below the checkbox.

# Applet

Use the **applet element**, `<applet>`, if you want to embed a Java applet in an HTML document. Choose Static > Applet to add an applet element.

# Comments

Use a **comment element** if you want to enter text into the document that is ignored when rendering the HTML. For example, in Layout mode, choose Static > Comment and enter some text. Switch to the Preview mode and you see that the comments are not displayed.

# Horizontal Rule

A **horizontal rule** is a line across the page. Choose Static > Horizontal Rule to add a horizontal rule element (`<hr>`). You can change the height, width, and style of a horizontal rule element using the Horizontal Rule Inspector.

# Headings

**Headings** are the titles of sections and are typically displayed in larger font sizes and bold. HTML supports six levels of headings. Choose a submenu from the Static > Headings menu to add a header element (`<h1>` thru `<h6>`). If there is a selection, it is wrapped with the header element. You can change the header level (select a number from 1 to 6) using the Header Inspector.

# Images

You can also embed images in your HTML document. Choose Static > Image to add an image element (`<img>`). A Select Image panel appears, allowing you to select an image file to display at the insertion point. Use the Image Inspector to change image properties, including its size, file path, and whether it uses an absolute or relative reference. See "Body and Document" (page 45)to set the background image of a page.

# Frames

Frames support multiple views of content using independent windows or subwindows. An **inline frame element** allows you to insert a frame within a block of text. Choose Static > Internal Frame to add an inline frame element at the insertion point. Read "Editing Frames" (page 43) for details on creating and editing frames.

# Lists

Lists provide support for enumerating information. All lists must contain one or more list items. WebObjects Builder supports ordered and unordered lists.

Choose Static > Lists > Ordered List menu to add an ordered list (`<ol>`), and choose Unordered List to add an unordered list (`<ul>`). If there is a selection, each line in the selection becomes a list item (`<li>`). You can use the inspector to change the list to an unordered or ordered list. You can also change the numbering style and the starting number.

When typing in a list:

- Press Return to create a new list item.
- Press Shift-Return to create a line break but no new list item.
- Press Tab to create a new list nested inside the original list.
- Press Shift-Tab to move the nesting back one level.

Read "PremadeElements Palette" (page 97) for how to create multiple list items wrapped in a repetition.

# Scripting

Use a **script element** to embed a script within a document. Choose JavaScript or VBScript from the Script submenu to add a script element. Use the Script Inspector to enter the script, select the type of scripting language, and set other script properties.

# Span

Use a **span element** to help structure your document. The span element defines content to be inline versus block-level (the division element described in "Address, Blockquote, Division, Paragraph, and Preformatted" (page 46)). Choose Static > Span to add a span element.

Read "Formatting with Cascading Style Sheets" (page 44)for examples on using the division element with cascading style sheets.

# Tables

Use a **table element** to display content in column and row format. Choose Static > Table to add a table at the insertion point. Read "Editing Tables" (page 38) for details on creating and editing tables.

# Custom

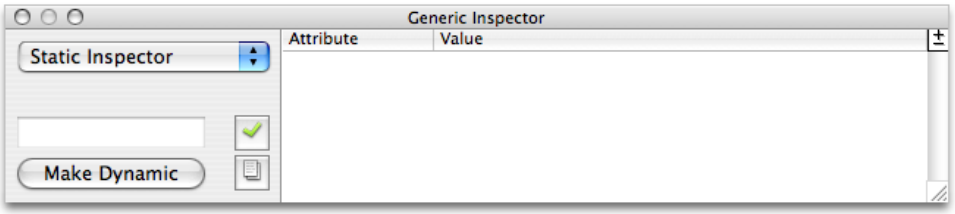
Use a **custom element** to add an HTML element that is not provided in one of the WebObjects Builder menus. Follow these steps to create a custom element:

1. Move the insertion point to where you want to add the custom element.
2. Choose Static > Custom.
3. Enter the custom marker in the "Tag to use" text field.
4. If the element doesn't require an end tag, the "New element is a container" option should not be selected.
5. Click OK.

   The marker is inserted in the component and HTML source—for example, `<marker>Custom Marker</marker>` is inserted in the HTML source.
6. If the element has attributes that you want to specify, select the custom tag in the path view and open the inspector window.

   A Generic Inspector window appears as shown in Figure 3-5.

**Figure 3-5** The Generic Inspector




7. To set attribute values, choose "Add attribute" from the +/- button menu and type the attribute's name and value in the table below.
8. To remove an attribute, select it in the table and choose "Delete attribute" from the +/- button menu.

Alternatively, you can set the attributes in source mode by editing the HTML text directly.

If you want to reuse the custom element, you can save it on a palette. Read "Working With Palettes" (page 95)for details on custom palettes.

# Working With Dynamic Elements

You use dynamic elements to build dynamic web pages whose content is determined at runtime by your application. You should understand conceptually how dynamic elements work before using them in your component. Dynamic elements use a Model-View-Controller design pattern and rely on key-value coding to work. Binding dynamic elements involves adding keys to your objects and editing Java code.

"Binding Dynamic Elements" (page 51) explains how dynamic elements work, describes the techniques you use to add dynamic elements to your components and describes how to bind them to variables and methods in your code. The rest of this chapter briefly covers the different types of dynamic elements—forms, concrete, and abstract—and provides examples of each.

Read *WebObjects Dynamic Elements Reference* for reference details of each dynamic element.

## Binding Dynamic Elements

Before you begin using dynamic elements, you should understand the concepts that make dynamic elements work. WebObjects uses the Model-View-Controller design pattern and key-value coding to encapsulate and reuse objects that have distinct roles. This section describes this design pattern and explains how to bind dynamic elements through examples.

### Model-View-Controller Design Pattern

Although WebObjects doesn't use the same terminology, it loosely follows a Model-View-Controller (MVC) design pattern. Understanding how this design pattern works helps you understand dynamic elements and bindings.

In the MVC design pattern, **models** encapsulate application data and the logic, or methods, that operate on that data. Models are typically the objects that you want to live beyond the lifetime of an application instance—they are the persistent objects that you store on disk. In WebObjects, the models are **enterprise objects**.

**Views** display a representation of model data and often allow users to edit that data. In WebObjects, views are **dynamic elements**.

**Controllers** mediate between the two, allowing for a clear separation of functionality between model and view. In WebObjects, controllers are your web component object and any other objects you can access via your web component, such as the application and session objects and any display groups you might add. A display group is analogous to an NSArrayController in Cocoa.

Essentially, controllers are the objects that appear in the object browser pane in Layout mode. You add keys and actions to controller objects and bind them to dynamic elements. For example, a display group is a typical controller that manages a collection of enterprise objects that are displayed via a repetition in a table. Read "Binding Elements" (page 57) for more advanced topics.

## Types of Dynamic Elements

A **dynamic element** is an element whose exact HTML representation isn't determined until runtime. Dynamic elements are represented in the HTML template by the `<webobject>` tag.

There are several types of dynamic elements that you can use in your WebObjects applications. Some of them—for example, dynamic forms or images—have counterparts in standard HTML—for example, `<form>` and `<img>`—and are always translated into those counterparts at runtime. These are called **concrete dynamic elements**.

Others don't translate directly into HTML but control the generation of other elements—for example, conditionals and repetitions. These are called **abstract dynamic elements**.

However, all dynamic elements have similar characteristics and the technique for binding them is the same. In general, dynamic elements have one or more attributes whose purpose is to either:

- Determine the exact HTML to generate when the element is displayed
- Capture information from users—for example, dynamic form elements
- Specify actions to take when an event occurs

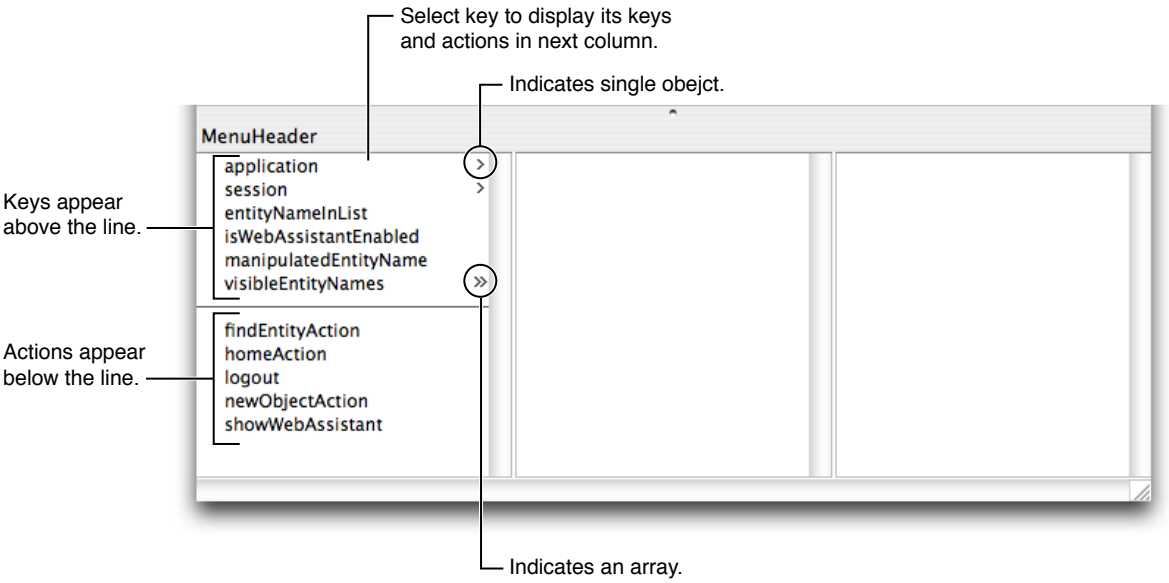
The process of associating a dynamic element attribute with a variable or method in your code is called **binding**. WebObjects Builder provides tools that make it easy to create bindings. Information about the bindings you make are stored in the declarations (`.wod`) file in your component.

Most dynamic elements have a number of attributes that you can bind. Some are required and others are optional. Read *WebObjects Dynamic Elements Reference* for a complete description of each dynamic element.

## Browsing Objects

You bind objects using the object browser, the bottom pane of the component window in Layout mode. The object browser displays your application objects and their variables and methods. The object browser allows you to visually bind objects in your code to dynamic elements in the component. For example, Figure 4-1shows the object browser for the `MenuHeader.wo` component of a Direct To Web application.

**Figure 4-1** The MenuHeader object browser




The first column in the object browser displays both keys and actions that belong to your component. **Keys** are displayed above the horizontal line and can be either an instance variable or method that returns a value. **Actions** are displayed below the line—they are methods that take no parameters and return a component that is the next page. If an action returns `null`, the current page is redisplayed.

You use the object browser to essentially navigate your web component's object graph. A greater-than character (>) appears next to an object's name in the browser if it contains additional keys and actions (if it has children). When you select an object, its keys and actions are displayed in the next column. If a key is an array, then two greater-than characters appear next to its name (>>). When you select an array, it's `count` attribute along with other attributes is displayed in the next column. For example, Figure 4-1 (page 53) shows that the web component has a key called `visibleEntityNames` that is an array.

All components have `application` and `session` keys because they are keys inherited from WOComponent. You can subclass WOApplication and WOSession to add additional keys and actions to these objects.

## Modifying the Web Component Interface

You can add keys and actions to your web component interface using either WebObjects Builder or by editing the component source in Xcode. When you save your changes in Xcode, WebObjects Builder parses the file, detects items that have been added, deleted, or changed and updates the object browser to reflect those changes.

Alternatively, you can add a display group key by dragging an EO model entity or relationship from EOModeler or Xcode to your component window. Read "Using Display Groups" (page 81) for more information on adding display groups in this manner.

Follow the instructions in this section to modify your web component interface using WebObjects Builder. The instructions use the Interface pop-up menu on the toolbar but all of these actions are also available from either the Edit Source submenu in the Edit menu or a contextual menu in the object browser (Control-click an object in the object browser to display the contextual menu).

Note that your component needs to be saved and added to your Xcode project before you can modify the interface—the component's Java source file needs to be on disk. If you created a new component, the Interface menu may be disabled. Save your component to enable this menu.

## Adding Keys

Keys are the variables in your code that contain application-specific data. You can bind keys to dynamic elements to either push data to the user or pull data from the user. Use the Interface menu to add a key as follows:

1. Choose Interface > Add Key.

   A sheet appears prompting you for more information.

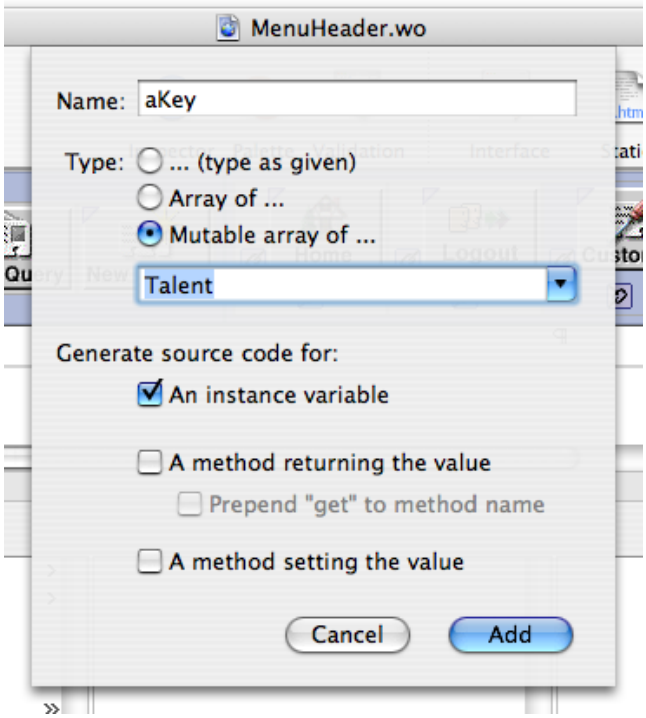
2. Type the name of the key in the Name field.

3. If the key is a mutable array, select "Mutable array of" as the type and choose the contained type from the pop-up menu or enter the type in the combo box directly.

   WebObjects Builder needs to know the type of objects that will be contained in the array so you can make additional bindings to those objects in the object browser. The pop-up menu also contains the entity names in your models. If you are unsure which type to use, choose Object.

   For example, in Figure 4-2 (page 54)the key is a mutable array of type Talent.

**Figure 4-2** Add key sheet




4. If the key is a non-mutable array, select "Array of" as the type and choose the contained type from the pop-up menu or enter the type in the combo box directly.

5. If the key is not an array, select "(type as given)" and choose the type from the pop-up menu or enter the type in the combo box directly.

6. Select the appropriate objects under "Generate source code for." For example, select "An instance variable" if you want an instance variable created in the source code.

   The key can be an instance variable whose value is accessed directly or a method that returns a value (not necessarily associated with an instance variable). You can also create a method that sets the value of an instance variable.

   Note that at least one source code option must be selected. You cannot deselect all of the source code options.

7. Click Add when done.

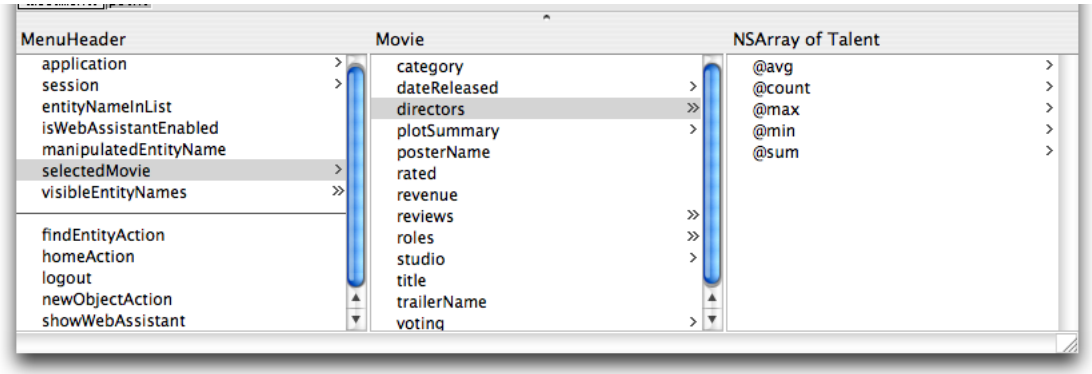
   The new key appears in the object browser (below `application` and `session`) and the corresponding source code is updated.

If you choose an entity name from the Type pop-up menu—for example, if you enter `selectedMovie` as the name and choose Movie as the type—the following code is added to your source file:

```
/** @TypeInfo Movie */
protected EOEnterpriseObject selectedMovie;
```

The instance variable `selectedMovie` is declared as type EOEnterpriseObject. The comment `/** @TypeInfo Movie */` is a structured comment that WebObjects Builder uses to identify the entity associated with the object—don't edit it. The structured comment enables WebObjects Builder to display the attributes of `selectedMovie` in the object browser as shown in Figure 4-3.

**Figure 4-3** Viewing object properties




## Adding Actions

Add actions to your interface to perform some business logic or processing when, for example, a button is clicked. The action needs to return the next component or webpage that will be displayed. Follow these steps to add an action to your component:

1. Choose Interface > Add Action.

   A sheet appears prompting you for more information.

2. Enter the name of the action in the Name field.
3. Select the response page from the Component pop-up menu or type in the name of the component in the text field.

   The Component pop-up menu that appears contains standard JavaScript components used for action buttons.

   If you type in a component name, make sure the component is loaded in your project; otherwise the action will fail at runtime.
4. Click Add when done.

   The new action appears below the line in the object browser and the corresponding source code is updated.

## Deleting Keys and Actions

Follow these steps to delete a key or action:

1. Select the key or action to delete in the object browser.
2. Choose Interface > Delete Key.

   Any variables and methods associated with the key or action are deleted from the source file. You can restore the deleted key or action by choosing Edit > Undo.

## Renaming Keys and Actions

Follow these steps to rename a key or action:

1. Select the key or action to rename.
2. Choose Interface > Rename Key.

   A sheet appears to obtain the new name.
3. Enter the new name in the Name field.
4. Click Rename.

## Viewing Source Files

Choose Interface > View Source to open the web component Java source file in Xcode.

## Modifying the Application and Session Interface

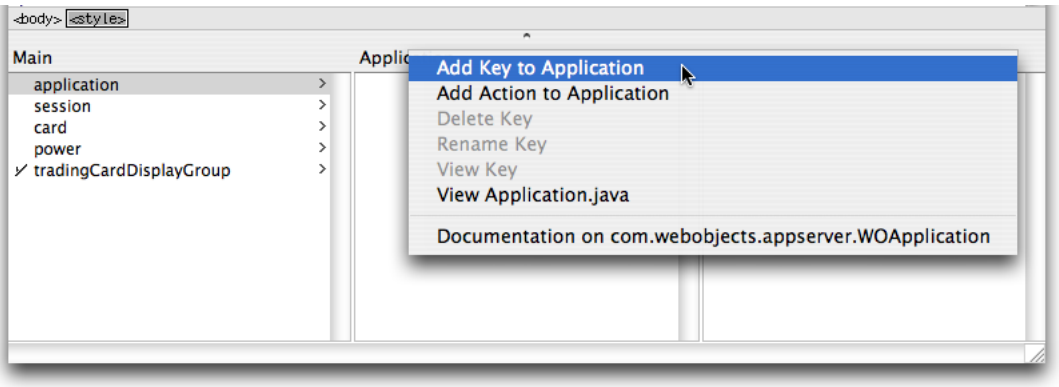
Similarly, you can add keys and actions to your `application` and `session` objects. Your WebObjects application should contain `Application.java` and `Session.java` files located in the Classes group. These classes are subclasses of WOApplication and WOSession respectively. When you add keys and actions to the `application` and `session` variables in your component, you are modifying the Application and Session class interfaces respectively.

Follow these steps to modify the application interface:

1. Select `application` in the object browser and Control-click anywhere in the second column including the heading.

   A menu appears containing various options to modify the application interface as shown in Figure 4-4.

   **Figure 4-4** The Interface menu




2. To add a key, choose Add Key to Application.
3. To add an action, choose Add Action to Application.

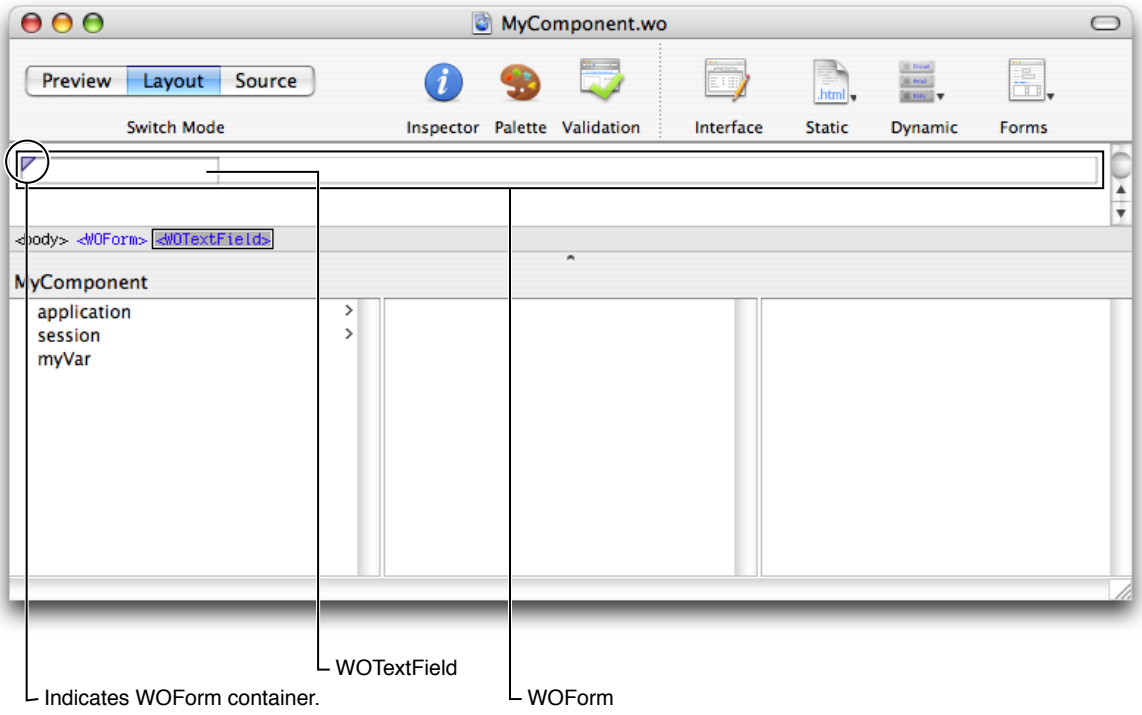
Follow the same steps to modify `session` except select `session` in the object browser.

## Binding Elements

No matter what dynamic elements you use, the procedure to bind them to your objects—for example, application, session, web component, or display group object—is the same. This section discusses the basic procedure for binding elements. Read "Using Dynamic Form Elements" (page 62), "Using Other Concrete Dynamic Elements" (page 68), and "Using Abstract Dynamic Elements" (page 73) for element-specific details and more examples.

Figure 4-5 shows a form element (WOForm) wrapped around a dynamic text element (WOTextField). The triangle in the top-left corner of the WOTextField distinguishes a dynamic text element from a static HTML text element. The long rectangle surrounding the text element represents the containing form element.

**Figure 4-5** Binding a WOTextField




Follow these steps to bind the dynamic text element to an application object—for example, bind the WOTextField to `myVar` (follow the steps in "Adding Keys" (page 54) to add `myVar` to your component).

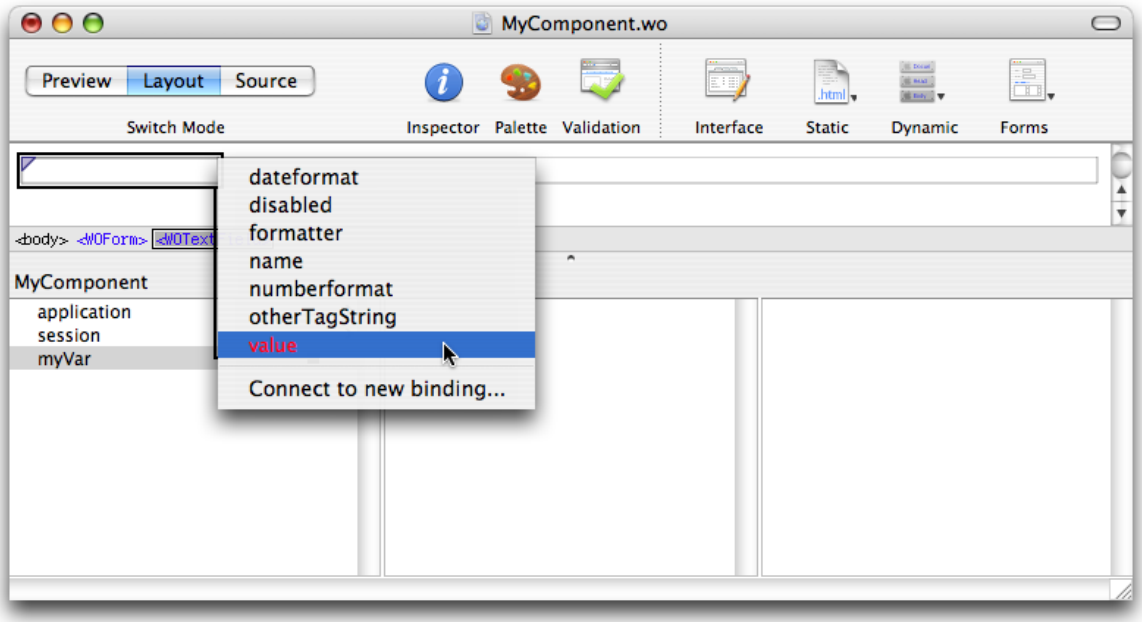
1. In the object browser, drag from `myVar` to inside the text field element in Layout mode.

   A black line appears from `myVar` to the current mouse location. As you drag over an element, a black box appears around the element indicating that you can bind to that element.

2. Release the mouse button.

A menu containing the attributes for that element appears as shown in Figure 4-6.

**Figure 4-6** Binding the value attribute

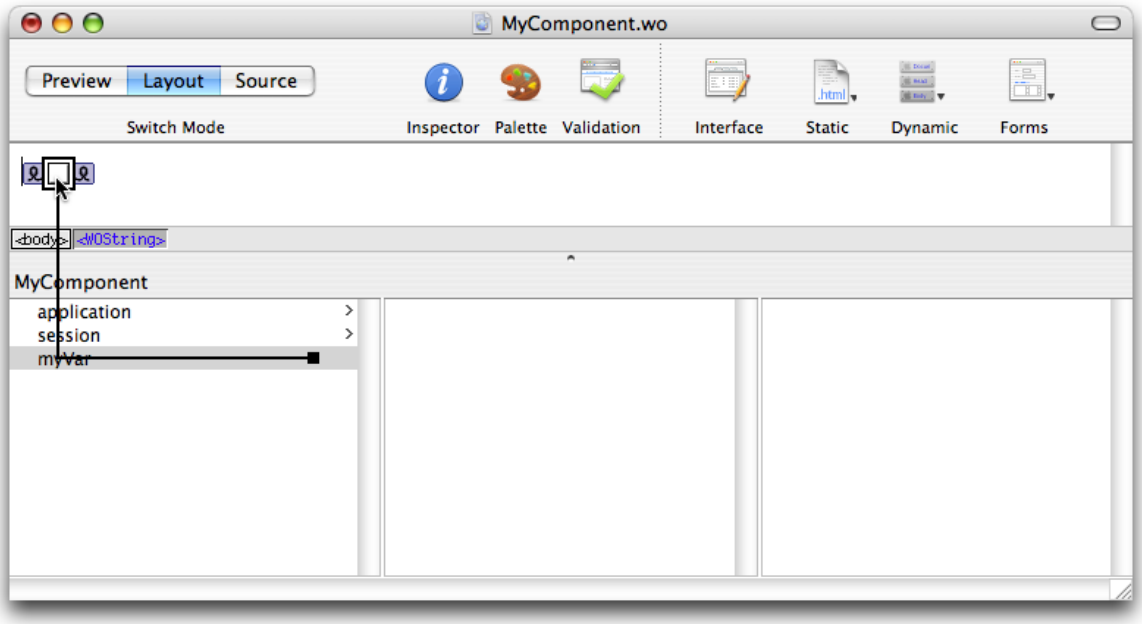



3. To complete the binding, choose the attribute you want to bind to from the menu—for example, choose `value` if the element is a WOTextField.

   The name of the binding appears in the element—for example, `myVar` appears in the WOTextField. The bindings available depend on the element. Read *WebObjects Dynamic Elements Reference* for a detailed description of each dynamic element and its bindings.

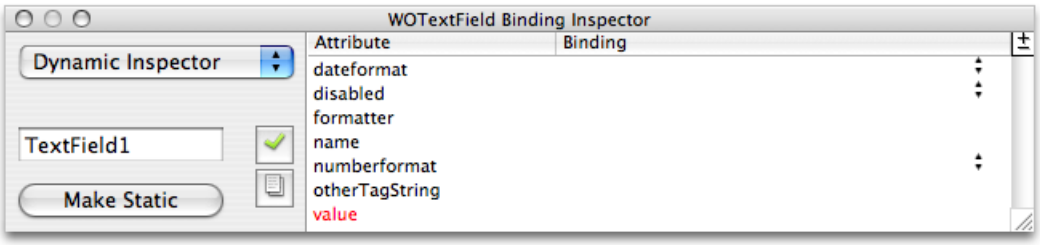
Some dynamic elements have icons that appear at the beginning and end of the element in Layout mode. You can optionally drag to the icon to create a binding. Some elements, like WOString elements, have shortcuts—for example, you can drag from a key to the center of a WOString (as shown in Figure 4-7) to bind to the `value` attribute of the WOString. The binding appears directly in the box, and the attribute menu doesn't appear.

**Figure 4-7** Binding a WOString




You can also view the bindings using the inspector. Select the element in either the graphical view or the path view and click the Inspector button on the toolbar to open the Inspector. The name of the variable appears in the Binding column next to the attribute as show in Figure 4-8.

**Figure 4-8** The WOTextField Binding Inspector




You can also bind an element's attributes by typing in the inspector directly as follows:

1. Double-click in the Binding column of the row for the attribute you want to set.

   A cursor appears in the Binding column, allowing you to type.

2. Type the binding in the text field.

   As you type, WebObjects Builder attempts to complete the binding using keys in the object browser.

3. When the desired key appears in the binding column, press Return.

These rules apply when entering bindings directly in the inspector:

- Constant strings (such as "MyVar") must be in double quotes.
- Variable and method names (such as `myVar`) must not be in quotes.
- Symbolic constants (such as `true` and `false`) must not be in quotes.
- You must specify the full key path when entering a binding.

## Deleting Bindings

Use the inspector to delete a binding as follows:

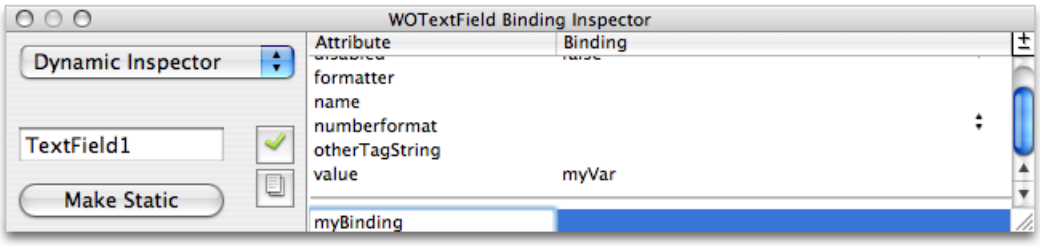
1. Select the element.
2. Click Inspector on the toolbar to open the inspector window.
3. Click the row that contains the binding you wish to delete.
4. Choose "Delete binding" from the +/- menu in the top-right corner of the inspector window. Optionally, press the Delete key.

## Adding Bindings

You can also add bindings to a dynamic element. Typically, you add bindings to a custom element that may not appear in the inspector—WebObjects Builder already knows about the bindings for existing elements. Follow these steps to add a binding to an element:

1. Select the element.
2. Click Inspector on the toolbar to open the inspector window.
3. Choose "Add binding" from the +/- menu in the top-right corner of the inspector window.
4. Enter the name of the binding in the Attribute column as shown in Figure 4-9.

**Figure 4-9** Adding bindings



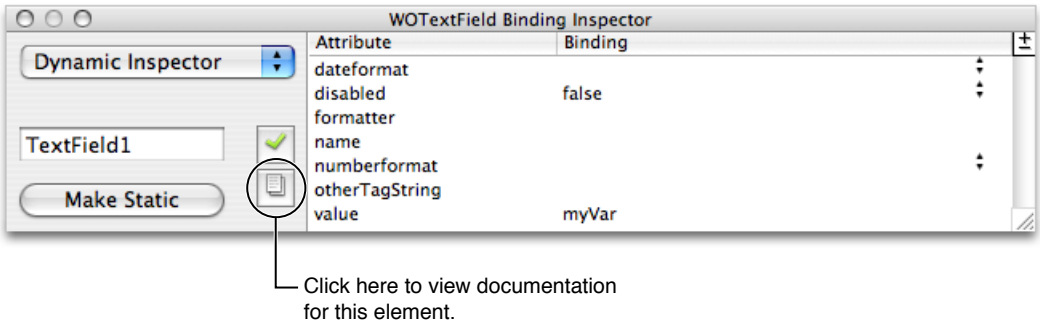
## Viewing Documentation

You can use the inspector to view documentation about an element and its bindings as follows:

1. Select the element.
2. Click Inspector on the toolbar to open the inspector window.
3. Click the documentation button.

   A window opens displaying the documentation for the element and its bindings as shown in Figure 4-10.

**Figure 4-10** Viewing documentation




# Using Dynamic Form Elements

This section explains how to use the dynamic form elements to create forms where users enter information. A form is a page or portion of a page containing fields that the user edits. Typically, the user clicks a button or types Return when done entering information in a form. WebObjects gives your application access to the data entered by users by allowing you to associate, or **bind**, these elements to variables and actions in your application.

In WebObjects Builder, you create form elements by choosing one of the dynamic elements from the Forms pop-up menu on the toolbar or Forms menu in the menu bar. All the form elements in this menu are *concrete*—they have static HTML counterparts. You can convert any dynamic form element to its static equivalent (and vice versa) by using the inspector. Read "Inspecting Elements" (page 26) for how to convert dynamic elements to their static counterparts.

In HTML, a form is a container element—an element that contains other elements. Typically, forms contain input elements–for example, text fields, radio buttons and checkboxes—to capture user information, a button or active image to submit the form data, as well as display elements such as text and images.

Most form elements have a `value` binding that represents the information entered by the user. You bind this attribute to a key in your application so that your application can use it. Other elements, such as WOSubmitButton, WOImageButton, or WOForm, don't receive information but have an `action` binding representing an action to be taken when the form is submitted. Follow the general procedure for binding elements as described in "Binding Elements" (page 57).

Typically, you create a WOForm element to contain other form elements, including buttons. If you add submit and reset buttons to the form, they apply to all elements in the form. Read "Login Panel Example" (page 63) to understand how to create a form. The rest of this section introduces the form elements.

# Login Panel Example

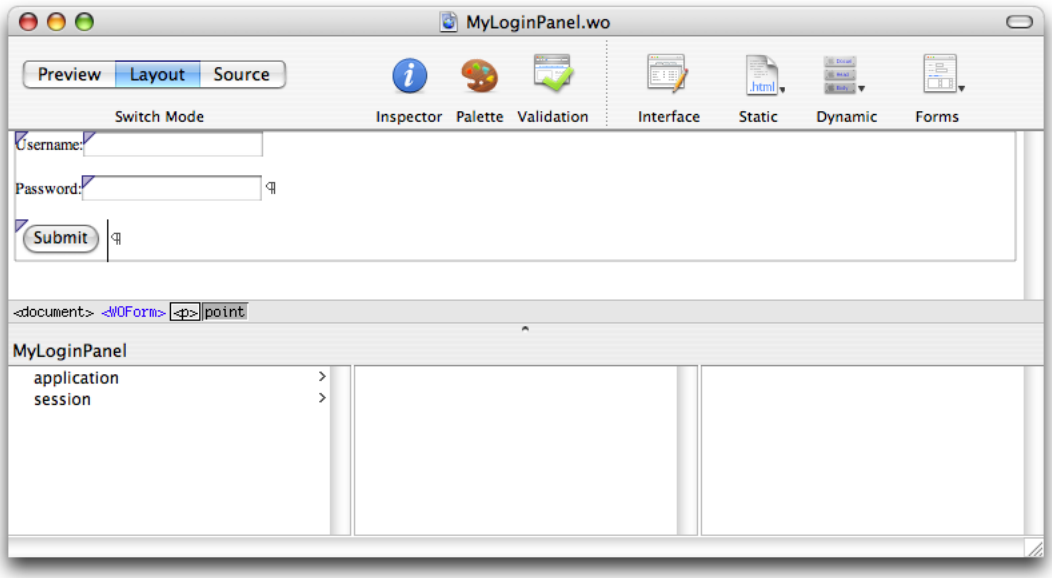
Follow the steps below to implement a simple login panel where the user enters a user name and password and then clicks Submit. This example covers the general procedure for creating a form: create form elements, add keys and actions, create bindings, and implement methods. You can extend this example to create your own form.

## Create Form Elements

Follow these steps to create the form for a login panel shown in Figure 4-11:

1. Choose Forms > WOForm to add a WOForm element to a new component.
2. Type `Username:` in the WOForm.
3. Choose Forms > WOTextField to create a username text field.
4. Press Return to create a new paragraph.
5. Type `Password:` in the WOForm.
6. Choose Forms > WOPasswordField to create a text field with hidden text.
7. Press Return to create a new paragraph.
8. Choose Forms > WOSubmitButton to add a submit button

**Figure 4-11** Creating login elements



You can extend this example by using a table to format the login panel. For example, use one column in a table to right-justify the labels and another column to left-justify the text fields. Use the last row in the table to center the Submit button under the text fields.

## Add Keys

Follow these steps to add keys needed by the login panel to your web component source:

1. Choose Interface > Add Key.

   A sheet appears prompting you for more information as shown in Figure 4-12.

2. Type `username` in the Name text field.

3. Type `String` in the Type text field or choose String from the menu.

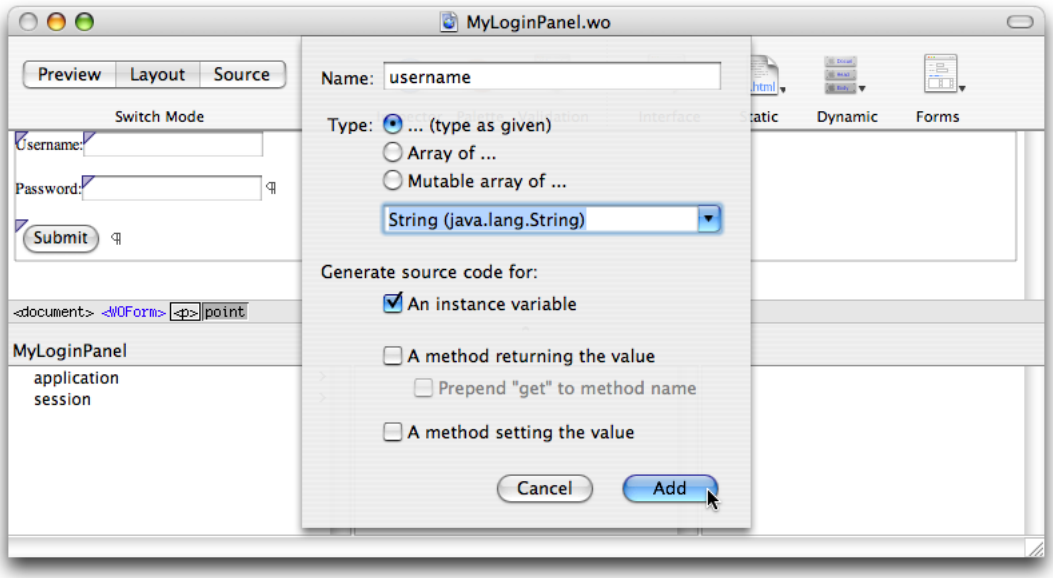
4. Select "An instance variable."

5. Click Add.

   A `username` key appears in the object browser.

6. Repeat steps above to add a `password` key.

**Figure 4-12** Adding login keys




## Add Actions

Follow these steps to add a login action to your web component source:

1. Choose Interface > Add Action.

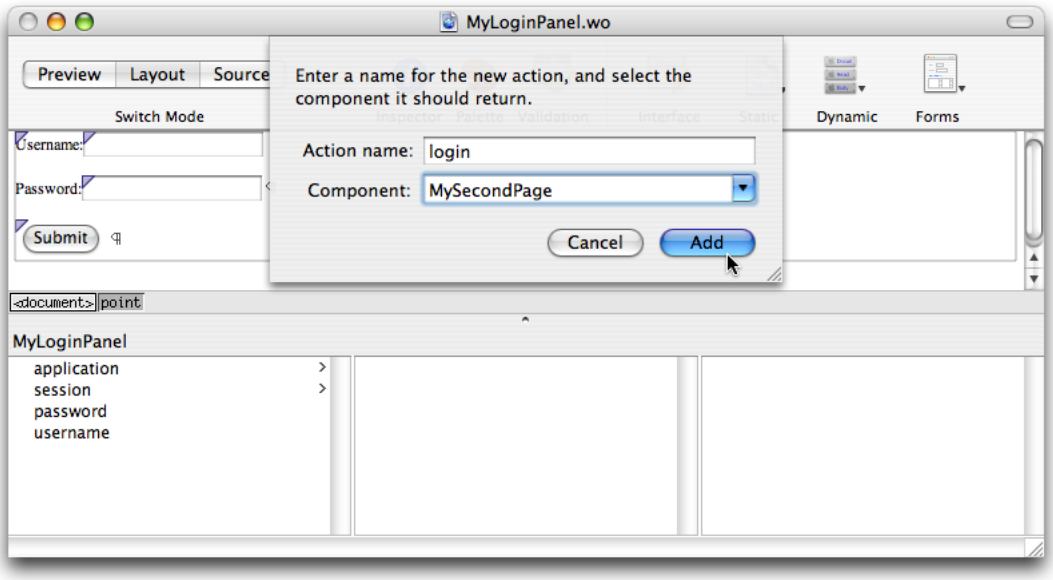
   A sheet appears prompting you for more information as shown in Figure 4-13.

2. Type `login` in the "Action name" text field.
3. Choose a component or type the name of the component in the Component combo box.

   Typically, you have a component representing a default page—the page that appears after the user logs in. The Component menu contains all the components in your project. Choose the appropriate component for your application from this menu.
4. Click Add.

   A `login` action appears in the object browser.

**Figure 4-13** Adding login actions




## Create Bindings

Follow these steps to bind the Username and Password text fields to your component:

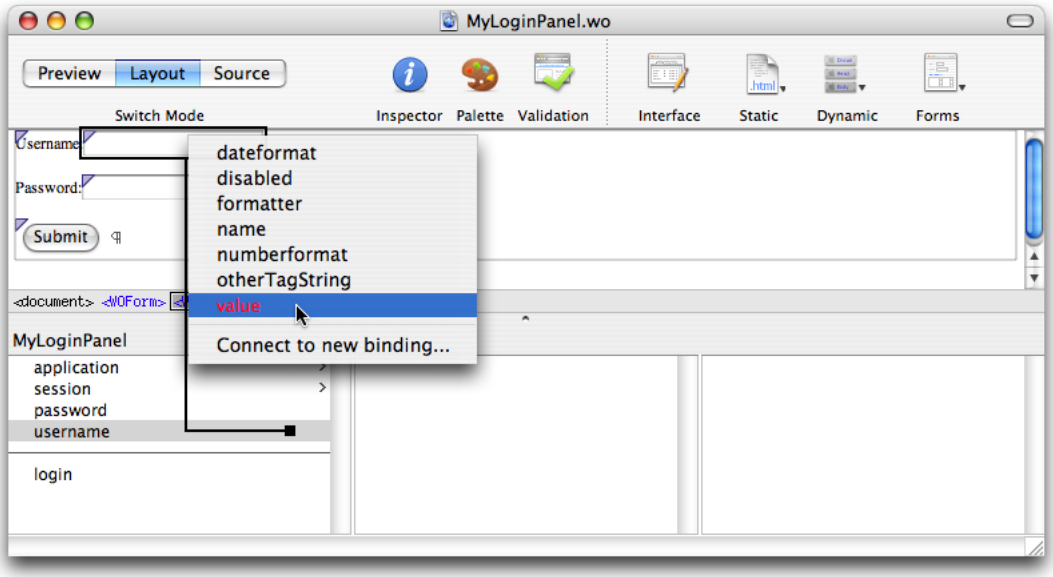
1. Drag from the `username` key in the object browser to the Username text field in the layout mode.
2. Release the mouse button while on the WOTextField element.

   A menu appears containing the bindings for the WOTextField as shown in Figure 4-14.
3. Choose `value` from the menu.

   The text `username` appears in the WOTextField.
4. Follow the same steps above to bind the `password` key to the Password WOTextField.

   The text `password` appears in the WOTextField.

**Figure 4-14** Creating login bindings




Follow these steps to bind the Submit button to the `login` action:

1. Drag from the `login` action in the object browser to the Submit button in the layout mode.
2. Release the mouse button while on the WOSubmitButton.

   A menu appears containing the bindings for the WOSubmitButton.
3. Choose action from the menu.

### Implement Methods

Finish by implementing the login action in your web component source to verify the `username` and `password` provided by the user (how you verify the login fields is application-dependent). The method should return the default page if the `username` and `password` are correct; otherwise, return `null` to show the same page.

Optionally, you can add an error message using a WOString element to the page. The WOString can display an error message when the login action failed—for example, display "Your password is incorrect. Please try again." You can wrap the WOString in a conditional so it appears only when an error occurs.

## Text Fields

You use the following text elements to display or input text:

- WOText is a multiple-line text field for input or display.
- WOTextField is a single-line text field for input and display.
- WOPasswordField is similar to WOTextField but hides the text entered by the user.

- WOHiddenField adds hidden text to an HTML page. You can use this element to store application state data in a page.

## Browser

Use WOBrowser to display a list of selectable items in a scroll view. Use the related element, WOPopUpButton, if you want to display the selected item in a pop-up menu.

## Checkboxes, Pop-up Menus, and Radio Buttons

Use checkboxes, pop-up menus, and radio buttons for selecting items from a list. You use checkboxes for multiple selections of `boolean` values. Use pop-up menus and radio buttons for selecting one of many items. The specific elements are:

- WOCheckBox is a single checkbox with a label that you can use to turn on/off a flag. Use the WOCheckBoxList element if you want to create a list of checkboxes.
- WOPopUpButton is a menu button that pops up a list when selected. The user can select only one item in the list.
- WORadioButton is a single on/off switch you can use for a selection. Use WORadioButtonList if you want a group of WORadioButton elements where the user selects no more than one of several choices.

Read "PremadeElements Palette" (page 97) for how to create multiple radio buttons wrapped in a repetition.

## Buttons

There are three button elements:

- WOImageButton is a submit button where you specify the button image. Clicking the image of a WOImageButton has the same behavior as clicking a WOSubmitButton.
- WOSubmitButton submits the enclosing form's values to the WebObjects application.
- WOResetButton resets the form to the original state.

## File Upload

Use the WOFileUpload element if you want to allow the user to upload a file to the web server. Read "PremadeElements Palette" (page 97) for how to create a file upload element with a submit button.

# Using Other Concrete Dynamic Elements

In addition to dynamic form elements, WebObjects provides a number of concrete dynamic elements—elements that have HTML counterparts at runtime and display dynamic content. For example, a hyperlink element whose address is determined at runtime. You can bind these dynamic elements to variables and methods in your application to control how they are displayed.

The dynamic elements described in this section are concrete with HTML counterparts but do not need to be in a form. For example, use WOActiveImage if you need a button outside a form, and use WOImageButton if you need a button inside a WOForm.

Read "Using Dynamic Form Elements" (page 62) for more information on form elements. Read "Using Abstract Dynamic Elements" (page 73) to learn more about abstract dynamic elements—for example, conditionals and repetitions—that control how dynamic web content is generated.

## Strings

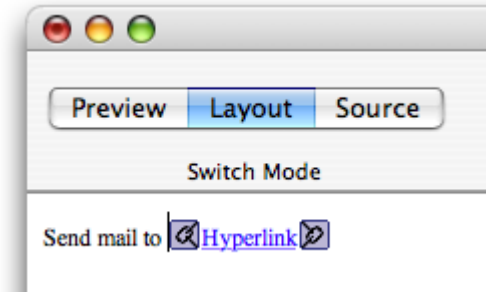
WOString elements dynamically generate string content. You bind the `value` attribute of a WOString to a variable or method in your web component that returns a string at runtime. The counterpart to a WOString element is not an HTML element but plain text. Therefore, WOString elements are treated like plain text—they can be contained in any HTML element that contains text and can be formatted in a way similar to the way text is formatted.

WebObjects Builder provides a shortcut for binding the `value` attribute of commonly used elements such as WOString. Instead of dragging to one of the icons on either side of the WOString element in the layout mode, drag to the center box as shown in Figure 4-7 (page 60). The binding appears directly in the box, and the attribute menu doesn't appear.

## Hyperlinks

WOHyperlink elements allow you to specify a hyperlink where the link's name and destination are dynamic. You can even specify your own action method to be invoked when the link is clicked. Read *WebObjects Dynamic Elements Reference* for a complete list of hyperlink bindings.

**Figure 4-15** An email hyperlink




For example, follow these steps to create a hyperlink at the bottom of your page to send email as shown in Figure 4-15:

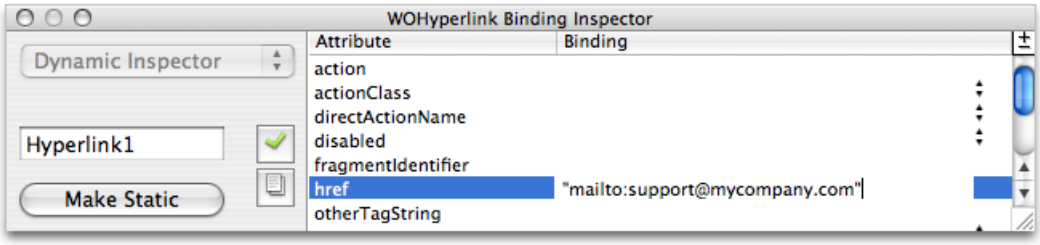
1. Type `Send mail to` in layout mode followed by a space.
2. Choose Dynamic > WOHyperlink.

   A hyperlink element appears with the word `Hyperlink` highlighted.
3. Type `support@mycompany.com` into the WOHyperlink text area (this text replaces the word `Hyperlink`).

   The text that appears inside of a WOHyperlink is displayed at runtime—it doesn't need to be a URL, it can be a logical name, such as the name of a page. If you want the name to be dynamic, you can put a WOString inside of a WOHyperlink and bind the WOString to your variable.
4. Select the WOHyperlink in the graphical view or path view.
5. Click Inspector in the toolbar.

   The WOHyperlink Binding Inspector appears.
6. Double-click the cell, in the Binding column, to the right of `href`.
7. Type `"mailto:support@mycompany.com"` in the cell (include the quotes) as shown in Figure 4-16.

**Figure 4-16** The WOHyperlink Binding Inspector




8. Press Return.

When the user clicks `support@mycompany.com` on the webpage, the default mail program is launched and the email message is addressed to `support@mycompany.com`.

## Images

The WOImage and WOActiveImage elements are dynamic images. At runtime, WOImage is rendered as a passive image and WOActiveImage as a mapped, active image (behaves like a button). You bind the dynamic image's `filename` or `src` attribute to a key (variable or method) in your application that returns a file URL to your image. The `filename` attribute is a relative URL and `src` is an absolute URL. You can optionally bind the `width` and `height` of the image, too. Hence the content of an image can be dynamic.

This section contains some examples of using dynamic images. Read *WebObjects Dynamic Elements Reference* for a complete list of image bindings.

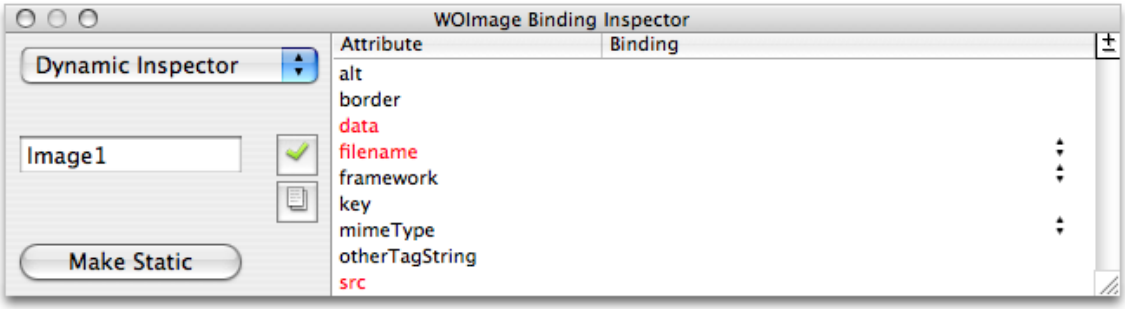
## WOImage Example

If you want to just display an image on a page, use WOImage and bind the `filename` or `src` attributes to a variable or method in your application. For example, follow these steps to add an image to your component that displays content returned by a method in your component. Assume that the `currentImage` method already exists in your component source.

1. Choose Dynamic > WOImage.
2. Select the new WOImage element and click Inspector on the toolbar.

   The WOImage Bindings Inspector appears. Mandatory bindings appear in red text as shown in Figure 4-17. You can bind either `data`, `filename`, or `src` although all of these bindings appear in red text when no bindings are set.

   **Figure 4-17** The WOImage Binding Inspector




3. Drag from `currentImage` in the object browser to the WOImage in the graphical view or path view.
4. Choose `src` from the pop-up menu.

   You are done when none of the bindings in the WOImage Bindings Inspector appear in red text. Otherwise, a validation error will occur.

## WOActiveImage Example

Typically, you use a WOActiveImage when you need a button outside of a form. The buttons described in "Buttons" (page 67) work only inside of a WOForm element. WOActiveImage has the same `filename` and `src` attributes, to specify the image, that you used to create a WOImage. After binding either `filename` or `src`, you just need to bind `action` to the method in your component that you want invoked when the button is clicked.

# Containers

Containers wrap around other types of elements. Use WOBody if you want a dynamic image as the background. Use WONestedList if you want to display hierarchical content. Use WOEmbeddedObject to add a Netscape plug-in. Use a WOFrame to specify the content of a frame dynamically. Read "Editing Frames" (page 43)for more information on using frames.

## JavaScript

Use WOJavaScript to embed a script written in JavaScript in a dynamically generated page. Use WOActionURL and WOResoourceURL to facilitate communication between JavaScript and your web application. Use WOActionURL to add a URL that invokes methods or specific pages. Use WOResourceURL to add a URL that returns resources such as images and sounds.

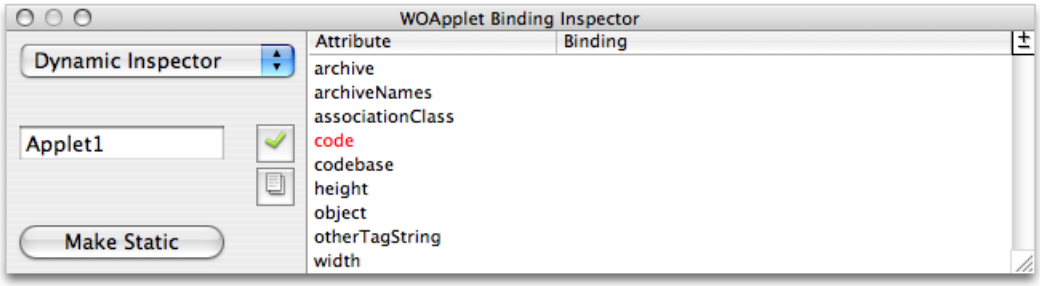
## Applets

The WOApplet dynamic element represents a Java applet or client-side component. There are two ways to create a WOApplet:

- Choose Dynamic > WOApplet.

  A WOApplet element is added to your component and you must open the inspector to set its bindings as shown in Figure 4-18. Set the `code` binding to the corresponding Java class—for example, `"MyApplet.class"`—and set the `codebase` binding—for example, set it to `"/Java/"`. By default, the Applet executable should be placed in `/Library/WebServer/Documents/Java`.

  **Figure 4-18** The WOApplet Binding Inspector




- Drag a file of type `.class` into your component.

  You are asked whether you want to add the `.class` file to your project. If you click Yes, it is added to the Web Server Resources group. A WOApplet appears in your component, with its `code` attribute set to the name of the file.

## Generic WebObjects Elements

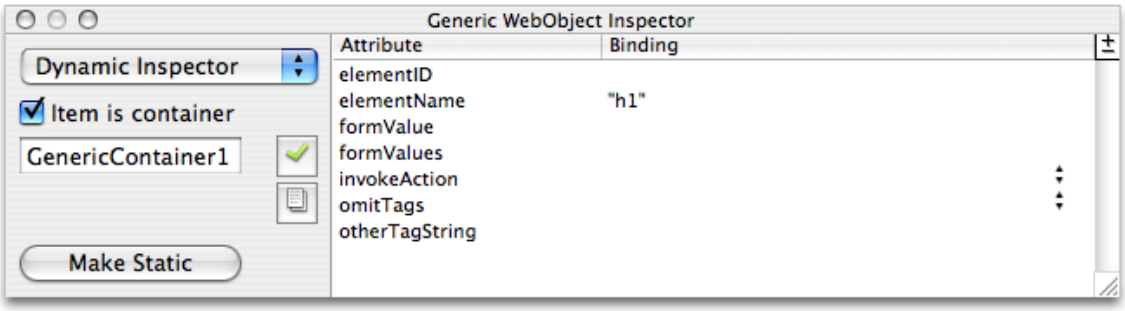
You use a generic WebObject element to create a dynamic version of any HTML element.

Follow these steps to create a generic WebObjects element from a supported HTML element:

1. Add an HTML element to your component as described in "Inserting Elements" (page 23).

2. While the element is selected, open the inspector as shown in Figure 4-19and click Make Dynamic.

**Figure 4-19** The Generic WebObject Inspector




If the element has no dynamic counterpart in WebObjects, it becomes a generic WebObjects element—that is, a WOGenericContainer or a WOGenericElement.

Follow these steps to create a generic WebObjects element from HTML that doesn't appear in WebObjects Builder:

1. Choose Dynamic > WOGenericContainer.
2. Click Inspector on the toolbar.

   The Generic WebObject Inspector appears with the `elementName` attribute in red text. The `elementName` attribute specifies the type of element that is generated at runtime in place of this generic element.
3. Double-click in the binding column next to `elementName` and type the name of the element in quotes.

   If the name isn't in quotes, WebObjects assumes it is a binding that should be resolved at runtime. You might use a binding if you want to choose the type of element programmatically rather than specifying it in advance.
4. Use the "Add binding" item in the +/- menu to specify any additional properties of the element that don't appear in the inspector.

If you create a WOGenericElement and want to change it to a WOGenericContainer, select "Item is container" in the inspector.

## WOXMLNode

Use WOXMLNode to build web components that are complex XML documents (not HTML). WOXMLNode has a single binding, `elementName`, that should be set to the name of the element you want wrapped around the content.

For example, choose Dynamic > WOXMLNode to add a WOXMLNode element. The HTML added to your component is `<webobject name="XMLNode1"></webobject>`. Whatever content you insert in this element in your component file will be wrapped in the `elementName` tag when rendered.

# Using Abstract Dynamic Elements

WebObjects also provides abstract dynamic elements that do not have HTML counterparts. These abstract elements, such as conditionals and repetitions, are used to control the generation of other elements. Using these abstract elements is similar to programming. You can wrap content with abstract dynamic elements `N` levels deep. Hence, you can generate very sophisticated HTML webpages using abstract elements.

## Conditionals

A WOConditional element is a dynamic container element that displays its contents only if a particular condition is `true`.

### Binding Conditionals

The main binding of a WOConditional element is `condition` which is a `boolean` value. If `condition` is `true`, the contents is displayed. If `condition` is `false`, the contents is not displayed.

Follow these steps to add a conditional:

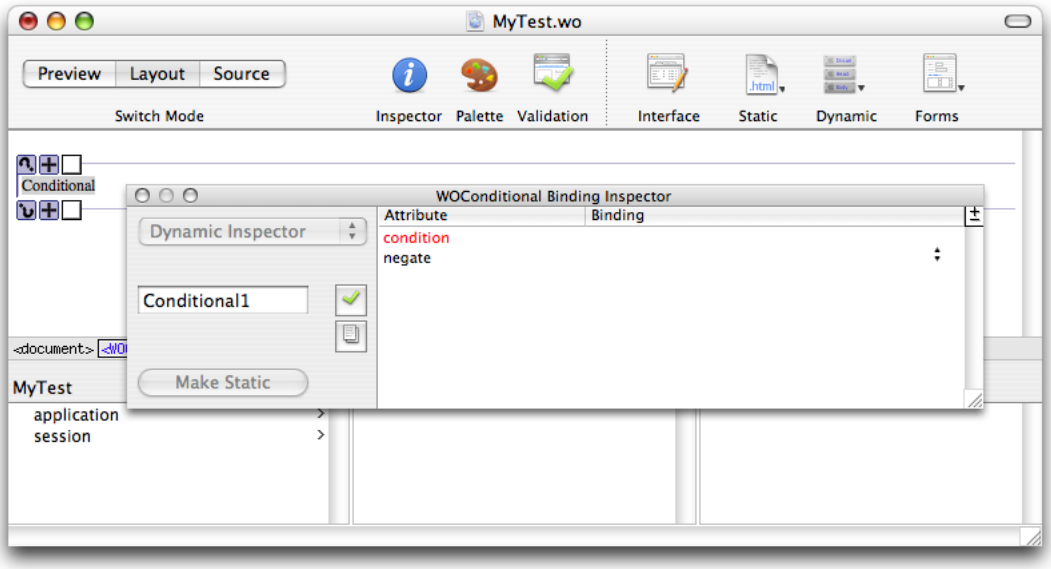
1. Choose Dynamic > WOConditional.

   Any selected elements will be contained within the conditional. Figure 4-20 (page 73)shows a new conditional with the default content when nothing is selected.

2. Select the conditional and click Inspector on the toolbar.

   The WOConditional Binding Inspector window appears with the `condition` binding in red text as shown in Figure 4-20.

**Figure 4-20** Adding a conditional



3. Bind `condition` to a variable or method in your application that returns a `boolean` value.
4. Set the `negate` attribute to `true` if you want the content displayed when `condition` is `false`.

## Using Conditionals to Create If-Then-Else Structures

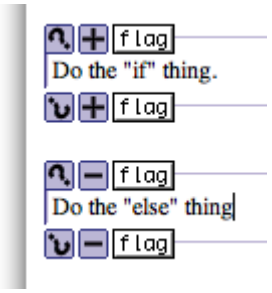
You can use conditionals to implement the equivalent of an "if-then-else" structure in a programming language; that is, "if `condition` is `true`, display this text; if not, display this other text." You use the `negate` binding of a conditional to do this. If `negate` is `true`, then the contents of the conditional is displayed only if `condition` is `false`. Follow these steps to create an if-then-else structure:

1. Choose Dynamic > WOConditional twice to create two adjacent conditionals.
2. Bind `condition` of both conditionals to the same variable or method.

   If using a variable, it must be a `boolean` type. If using a method, it must return a `boolean` value.
3. Click the plus icon on the second conditional to change `negate` to `true`.

   The button changes from a plus to a minus. By default `negate` is `false`, so you do not need to bind the first conditional's `negate` attribute unless you changed it. Your if-then-else structure should look like the one in Figure 4-21.

**Figure 4-21** An if-then-else structure




## Using Conditional Shortcuts

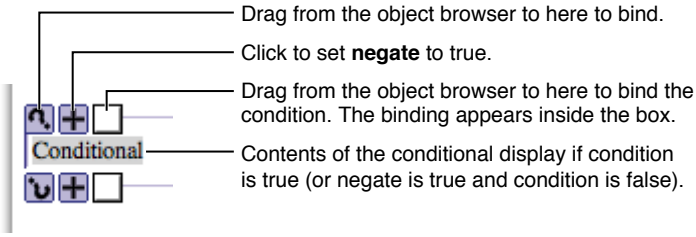
WebObjects provides these shortcuts for binding conditionals, as show in Figure 4-22:

- Drag from the object browser to the question mark icon to bind to any attribute.

  Choose a binding from the pop-up menu.
- Click the +/- button to toggle the `negate` state.
- Drag from the object browser to the first box to bind to `condition` directly.

  The conditional binding appears inside the box.

**Figure 4-22** Conditional shortcuts




### Wrapping Table Rows and Cells With Conditionals

You can wrap a conditional around a table row or cell to optionally display that row or cell.

To wrap a table row or cell with a conditional, select the row or cell—for example, click `<tr>` or `<td>` in the path view—and choose Dynamic > WOConditional. The selection is wrapped with the conditional. The conditional symbol doesn't appear by default in the graphical view but the row appears with a dark blue border and the cell or cells appears with a light blue background. (Read "Editing Tables" (page 34)for more details on selecting rows and cells.)

To select a conditional that is inside a table, click in the table and select the `<WOConditional>` element from the path view.

To bind a conditional that is inside a table, select inside the row or cell, and drag from your application variable or method to the `<WOConditional>` element in the path view. Or select the `<WOConditional>` element in the path view and open the inspector. Use the inspector to bind `condition` and `negate` to your application variables and methods.

Optionally, you can turn on the table tags to see the elements that are wrapped around rows and cells. If you do this, you can bind your elements using the shortcuts. To turn on the table tags, open the Preferences window, click the Layout button, and select tables from the "Show inline graphical tags for" options.

## Repetitions

A WORepetition element is a container element that repeats its content the specified number of times. It is like a loop in a structured programming language. Repetitions are one of the most important elements in WebObjects, since it is common for applications with database backends to display numerous records, and the amount of data to be displayed, is not known until runtime. For example, a repetition is used to generate items in a list, multiple rows in a table, or multiple tables.

This section describes how to create and bind your own repetitions. However, the PremadeElements palette contains some reusable components that use repetitions. There are examples of tables, list items, and radio buttons wrapped in repetitions. Read "PremadeElements Palette" (page 97)for more information on using this palette.

### Binding Repetitions

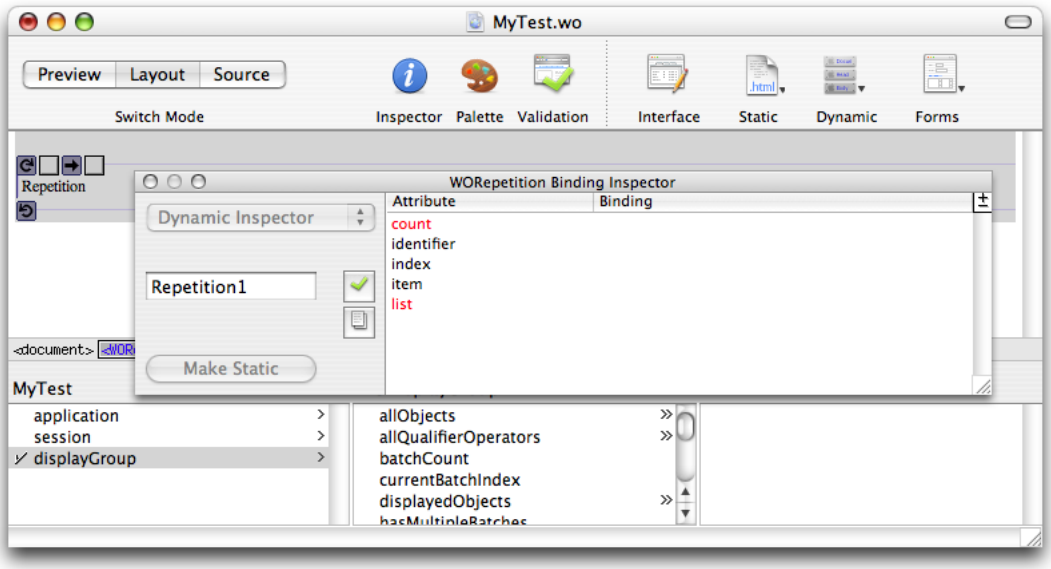
Typically, you bind these two attributes of a repetition: `list` and `item`. The `list` attribute must be bound to an array. WebObjects generates the elements in the repetition once for each item in the array. Each time through the array, the `item` attribute points to the current array object. Typically, you bind `item` to an application variable and then use that variable in the contents of a repetition.

Follow these steps to create and bind a repetition:

1. Choose Dynamic > WORepetition.

   The repetition appears in the component window at the insertion point as shown in Figure 4-23.

   **Figure 4-23** Adding a repetition




2. Add content to be repeated inside the repetition, replacing the word `Repetition`.

   A repetition can contain any other elements, either static HTML or dynamic WebObjects elements.

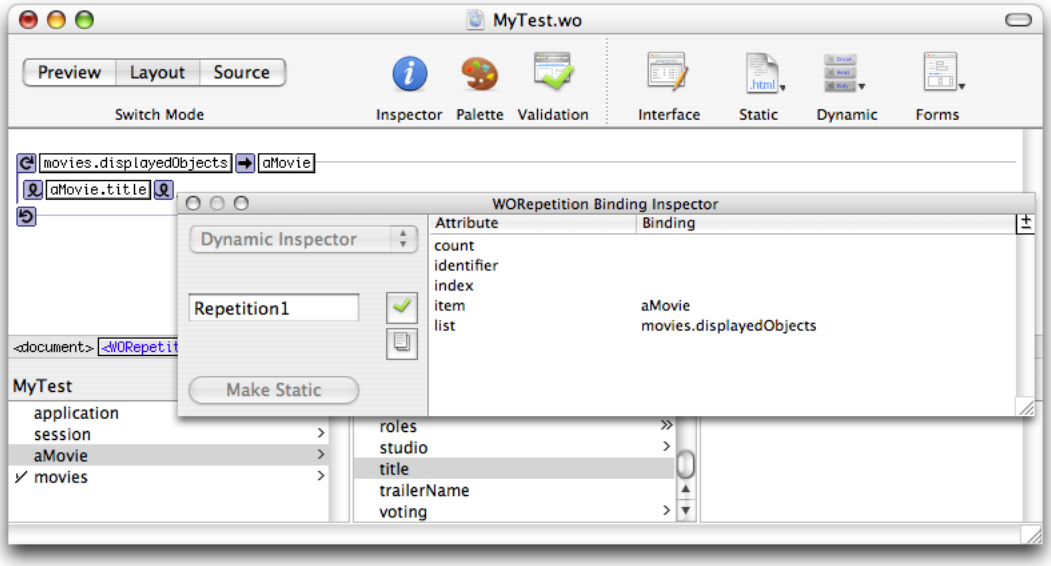
3. Select the repetition and click Inspector on the toolbar to open the inspector.

   The `count` and `list` bindings appear in red text. Typically, you bind `item` and `list`.

4. Bind `list` to an array in your application—for example, the `displayedObjects` attribute of a display group.

The array appears in the first text box between the repetition icon and right arrow icon as shown in Figure 4-24. If the inspector is open, `count` changes to black text and `item` changes to red text. If you bind `list` you must also bind `item`; otherwise a validation error will occur.

**Figure 4-24** Binding a repetition



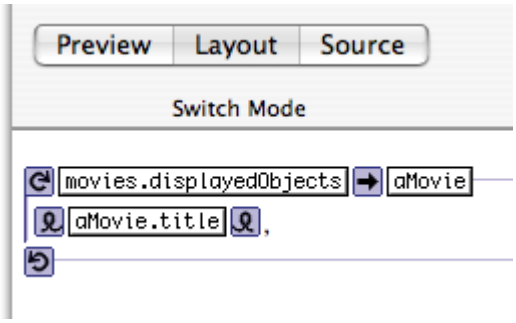

5. Bind `item` to a variable in your application that can be used within the repetition to reference the current item.

   The variable appears in the second text box after the right arrow icon as shown in Figure 4-25.

6. Finally, you add content inside the repetition that displays something about the item.

For example, using `Movies.eomodeld`, you can add a display group to your interface, called `movies`, that manages all the Movie objects, and a variable, called `aMovie`, that you use within the repetition to reference the current item. Figure 4-25 (page 77) shows what the repetition looks like in Layout mode when you bind `item` and `list`. The content of the repetition is a WOString that displays the `title` of `aMovie`. This repetition displays all the movie titles delimited by a comma. Read "Wrapping Table Rows and Cells With Repetitions" (page 78)for how to use a table to format large amounts of data.

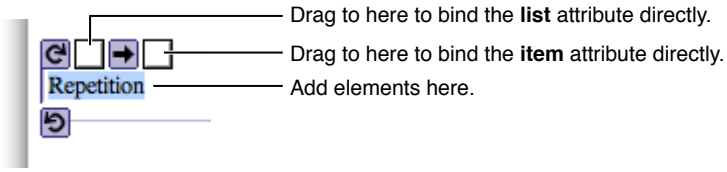
**Figure 4-25** A bound repetition



## Using Repetition Shortcuts

WebObjects Builder also provides shortcuts for binding repetitions so that you don't have to use the inspector. Drag to the first binding box to bind the `list` attribute, and drag to the second box to bind to the `item` attribute as shown in Figure 4-26.

**Figure 4-26** Repetition shortcuts

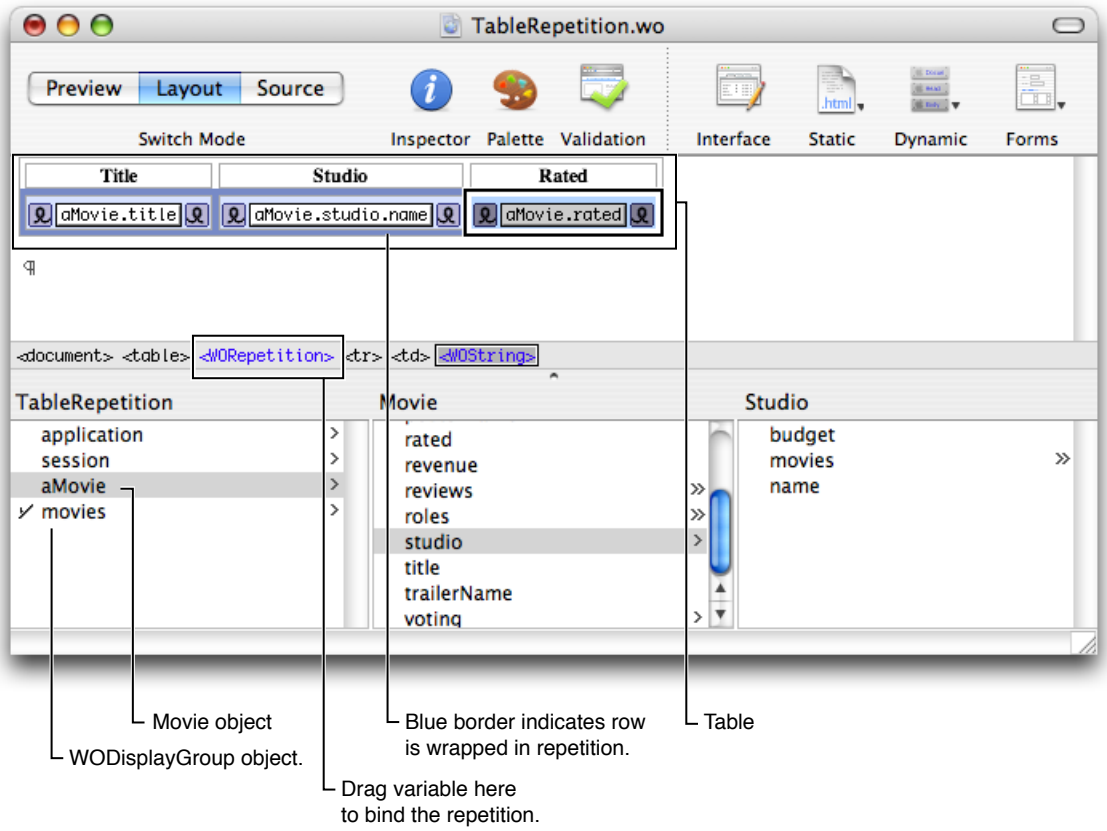



## Wrapping Table Rows and Cells With Repetitions

You can use tables to format large amounts of data where the quantity and content of the data is not known until runtime. Typically, you use tables, repetitions, and display groups to implement lists of your entity objects. Each row in the table corresponds to an enterprise object and each column corresponds to a property you want to display about that object. The display group is used to provide the array of objects. Because you can wrap both rows and cells with a repetition, the properties you display about enterprise objects in these tables can be determined at runtime too.

For example, Figure 4-27 shows a repetition wrapped around a row in a table. At runtime, this table will display information about Movie objects. The first column displays the title of the movie, the second displays the name of the studio who produced the movie, and the last displays the rating of the movie. The size of the table expands at runtime to contain all the records provided by the `movies` display group.

**Figure 4-27** Wrapping table rows with repetitions




To wrap a table row or cell with a repetition, select the row or cell and choose Dynamic > WORepetition. The repetition symbol doesn't appear by default in the graphical view but the row appears with a dark blue border and the cell or cells appear with a light blue background. (Read "Editing Tables" (page 34)for more details on selecting rows and cells.)

To select a repetition that is inside a table, click in the table and select the `<WORepetition>` element from the path view.

To bind the repetition that is inside a table, drag from the object browser to the `<WORepetition>` element in the path view. The attribute menu appears, allowing you to bind the element as usual. Otherwise, you can select the `<WORepetition>` element in the path view and open the inspector to bind the element.

Optionally, you can turn on the table tags to see the elements that are wrapped around rows and cells. If you do this, you can bind your elements using the shortcuts. *To turn on the table tags*, open the Preferences window, click the Layout button, and select tables from the "Show inline graphical tags for" options.

## Component Content

WOComponentContent is another abstract dynamic element that helps support another programming concept, reusability. You can build reusable web components and add them to other components or place them on a palette. You use the WOComponentContent element to group other elements and components together. Like the WORepetition and WOConditional dynamic elements, reusable components can wrap

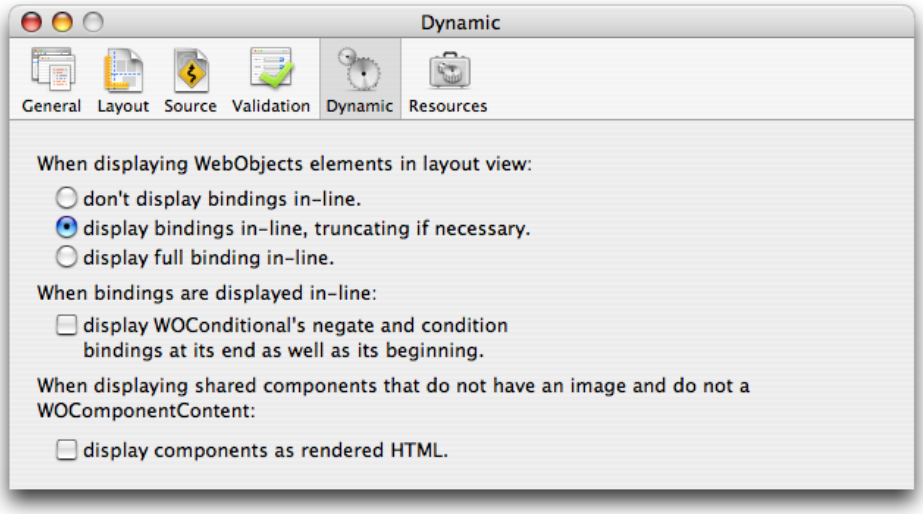
HTML elements. When you are editing a reusable component, the wrapped HTML elements are collectively called the component content. Use the WOComponentContent dynamic element to refer to it. Read "Custom Components" (page 103) for details on using WOComponentContent.

Choose Dynamic > WOComponentContent to create a component content. This dynamic element has no bindings, and you can only have one WOComponentContent element in a given component.

# Setting Dynamic Element Preferences

You can also configure the look of dynamic elements in the graphical view in Layout mode. Choose WebObjects Builder > Preferences to open the window and click Dynamic to view the dynamic element preferences as shown in Figure 4-28.

**Figure 4-28** Dynamic element preferences



# Using Display Groups

Display groups are special controller objects that manage a collection of enterprise objects. They fetch, insert, delete, display, update, and search records in a database. Using display groups in combination with dynamic elements, such as tables, and abstract elements, such as repetitions, is very powerful.

This section describes the mechanics of adding display groups to a web component using WebObjects Builder, how to implement a master-detail interface, and how to use a detail-display group. Read *WebObjects 5.3 Reference* for details about the WODisplayGroup class. Read *WebObjects Enterprise Objects Programming Guide* for concepts on enterprise objects that make display groups work.

## EO Models

WebObjects applications access databases through **enterprise objects**, which are represented by database rows. Enterprise object classes typically correspond to database tables, and an enterprise object instance corresponds to a single row or record in a table.

In a database application, you use an entity-relationship model, called an **EO model**, to associate database columns with instance variables of objects. You create a model with either the EOModeler application or the Xcode EO Model design tool. Alternatively, you can add an existing EO model using the assistant when creating an Xcode project. The EO model appears in the Resources group in Xcode. Read *EOModeler User Guide* or *Xcode 2.2 User Guide* for more information on creating EO models.

The EO model contains **entities**, **attributes**, and **relationships**. An entity associates a database table with an enterprise object class. An attribute associates a database column with an instance variable. A relationship is a link between two entities. In database terms, a relationship is a join between tables.

Display groups use an EO model to get information about the type of objects they manage. Display groups manage objects associated with a single entity.

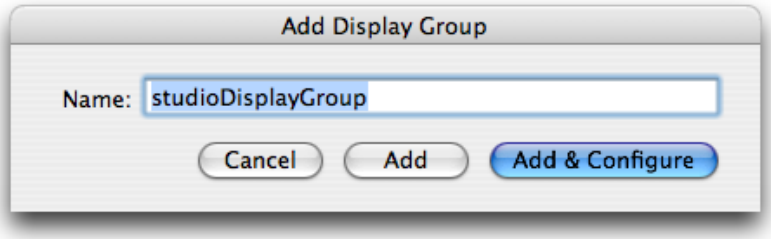
## Adding Display Groups

You may already have a display group added to your component that was created by the assistant when you created your WebObjects project. Otherwise, you can add a display group by dragging a model from the Finder or entity from EOModeler as follows:

1. Drag a model (a folder with the extension `.eomodeld`) from the file system to the object browser in your component window, or drag an entity from the EOModeler application, or Option-drag an entity from the Xcode EO Model design tool into the object browser.

An Add Display Group panel appears, as shown in Figure 5-1, asking if you want to add the model to your project.

**Figure 5-1** The Add Display Group panel




2. Enter the name of the display group in the Name field.
3. Click Add & Configure.

   A Display Group Options window appears prompting you for more information. (Alternatively, you can click Add on the Add Display Group panel and configure the display group later.)
4. Follow the steps in "Configuring Display Groups" (page 83) to configure the display group and click OK when done.

You can also add a display group to your web component directly and hook it up to a model manually as follows:

1. Choose Add Key from the Interface pop-up menu on the toolbar.

   A sheet appears prompting you for more information.
2. Type the name of the key in the Name field.
3. Choose WODisplayGroup from the Type pop-up menu.
4. Select the appropriate source-code generation options.
5. Click Add.

Alternatively, you can declare the display group directly in the code:

```
protected WODisplayGroup myDisplayGroup;
```

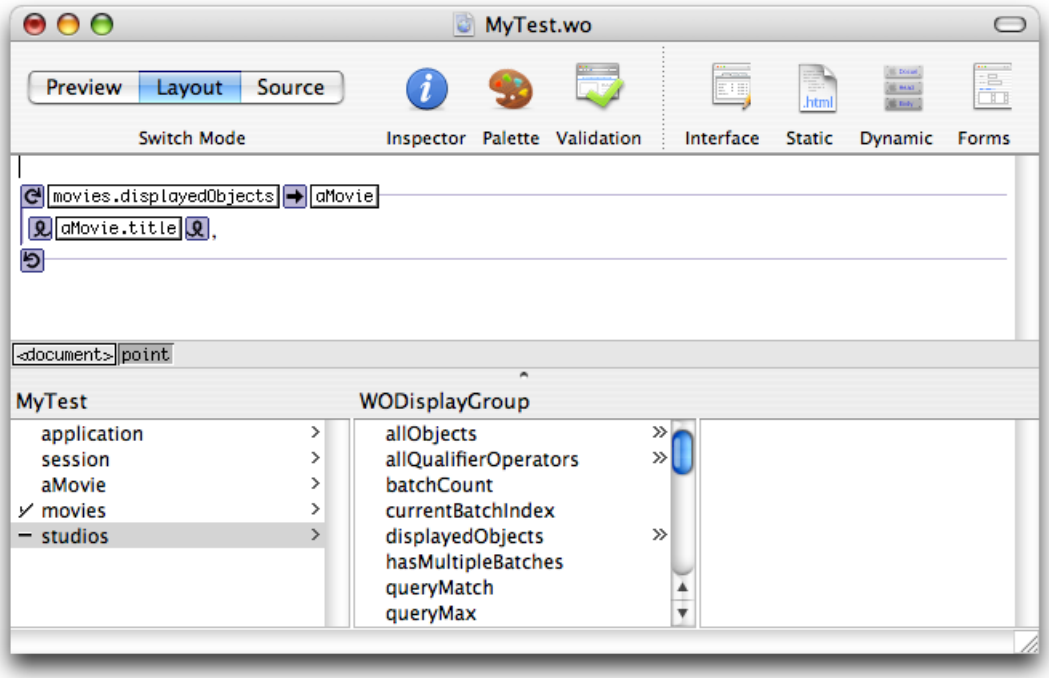
When you add a display group directly, you are responsible for ensuring that the EO model containing your entity is added to your Xcode project. You also need to configure your display group as described in "Configuring Display Groups" (page 83).

# Configuring Display Groups

A display group must be configured in order for it to be created and initialized automatically at runtime. Display groups are instantiated from an archive file (with the extension `.woo`) that is stored in the component folder. You shouldn't edit the `.woo` file directly—it is maintained by WebObjects Builder.

A checkmark appears next to a display group in the object browser if it has been configured. A minus sign next to a display groups means that it has not been configured. In Figure 5-2, the `movies` display group is configured and the `studios` display group is not. If a display group is not configured, it is not created at runtime. If you select a configured display group, its keys and actions are displayed in the next column in the object browser. You can bind these keys and actions to elements in your program.

**Figure 5-2** Configuring display groups

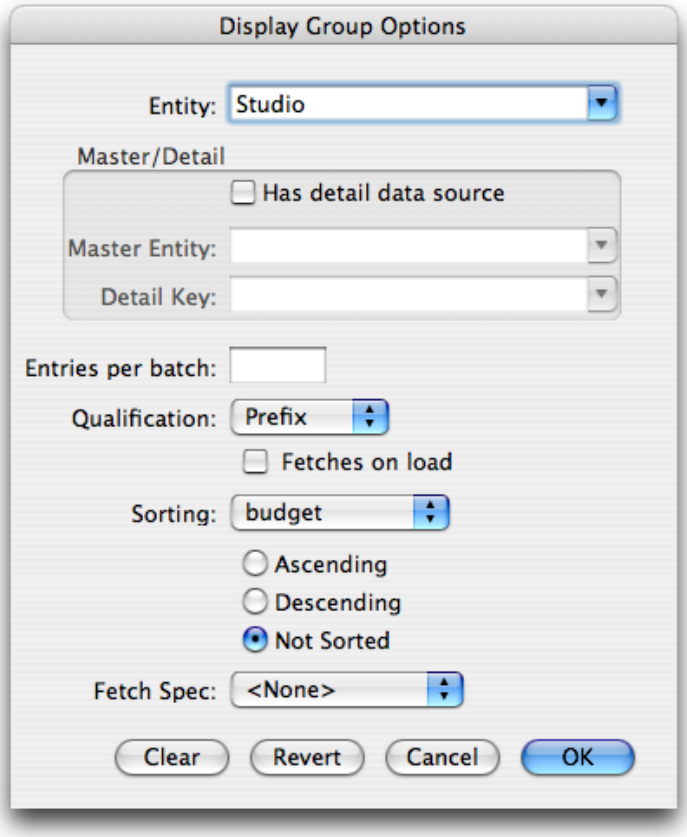



Follow these steps to configure a display group or change its configuration:

1. Double-click the display group in the object browser.

The Display Group Options panel appears as shown in Figure 5-3.

**Figure 5-3** The Display Group Options panel




2. Choose an entity name from the Entity pop-up menu or enter an entity name in the text field.

   The Entity combo box contains entities from the models in your project.

3. Click "Has detail data source" if you are creating a master-detail interface.

   Read "Creating a Detail Display Group" (page 90) for steps to create this type of interface.

4. If you want to batch fetched objects, enter the number of objects to fetch in a single batch in the "Entries per batch" text field.

   Batching is turned on if you enter a nonzero value in this text field. Otherwise, all enterprise objects are displayed.

5. When fetching objects that match a query, choose a qualifier from the Qualification pop-up menu to narrow the query.

   Choose Prefix to match the beginning of an attribute, choose Contains to match any part of an attribute, and choose Suffix to match the ending of an attribute.

6. Click "Fetches on load" if you want to fetch all records when the component is loaded.

7. If you want to sort the objects by attribute, choose an attribute from the Sorting pop-up menu, and select sort order: Ascending, Descending, or Not Sorted.

8. If you are using a fetch specification, select it from the Fetch Spec pop-up menu.

   The Fetch Spec pop-up menu contains all the fetch specifications defined in your model. A fetch specification is a predefined query that you create using EOModeler or the Xcode EO Model design tool. For example, you can create a fetch specification that fetches all records with a date attribute between some start and end date. Read *EOModeler User Guide* or *Xcode 2.2 User Guide*for more information on fetch specifications.

9. Click OK.

   The attribute is added to the object browser and a checkmark appears next it its name. If you select a configured display group, you will see its attributes, such as `allObjects`, displayed in the next column.

# Creating a Master-Detail Interface

Typically, you use display groups to create a master-detail interface. In a **master-detail interface**, the user can select objects from a list of objects and inspect the selected objects. The master interface displays the collection of objects, and the detail interface implements an inspector of the selected object. Whenever the user changes the selection in the master interface, the detail interface is updated to show the new selection. If no object is selected, the detail interface displays nothing or disables itself (if it is editable). If multiple objects are selected (assuming multiple selection is allowed), the detail interface is disabled or it applies to all selected objects. Typically, such an interface also allows users to add and remove objects from the collection.

This section assumes you created a project using the Movies example EO model. If you have not already done so, follow these steps to create it:

1. Create a simple web application using Xcode by choosing File > New Project in Xcode.

2. Select Web Application under WebObjects as the Xcode template.

3. When prompted for existing EO model files, enter `Movies.eomodeld` located in `/Developer/Examples/JavaWebObjects/Frameworks/JavaBusinessLogic`.

4. After your Xcode project is created, double-click `Main.wo` located in the Web Components suitcase to launch WebObjects Builder and begin creating your component.

The steps to create a master-detail interface involve creating a display group (similar to a NSArrayController object in Cocoa), creating views (the web component interface), and binding the views to the controller. In addition, you need to implement a mechanism for changing the selected object in the master interface.

## Creating a Display Group

You use a display group to manage your collection of model objects—your collection of enterprise objects. It's easy to do this if you drag your entity from Xcode or EOModeler to WebObjects Builder as follows:

1. Open `Movies.eomodeld` in Xcode and Option-drag the Talent entity from the EO Model design tool to the graphical view in Layout mode in WebObjects Builder.

   A panel appears asking if you want to add a display group to your component.

2. Click Add.

   A configured `talentDisplayGroup` key is added to your component's interface. The entity is set to Talent.

3. Double-click `talentDisplayGroup` to open the Display Group Options window and configure other settings. For example, you might select "Fetches on load."

4. Click OK.

## Creating a Master Interface

Typically, you use a table with its row wrapped in a repetition to display an array of enterprise objects. For example, you use a repetition to display the objects in `talentDisplayGroup.displayedObjects`. First you create the table and then you bind the contents to the display group.

Follow these steps to create the table:

1. Choose Static > Table to create the table of Talent records.

   The New Table panel appears allowing you to configure the table before inserting it in your component.

2. Enter `2` in the Rows and Columns text fields, check "First row cells are header cells," and "Second row is wrapped in a WORepetition." Click OK.

   A table with two columns and two rows is added to your component. The second row is wrapped in a repetition.

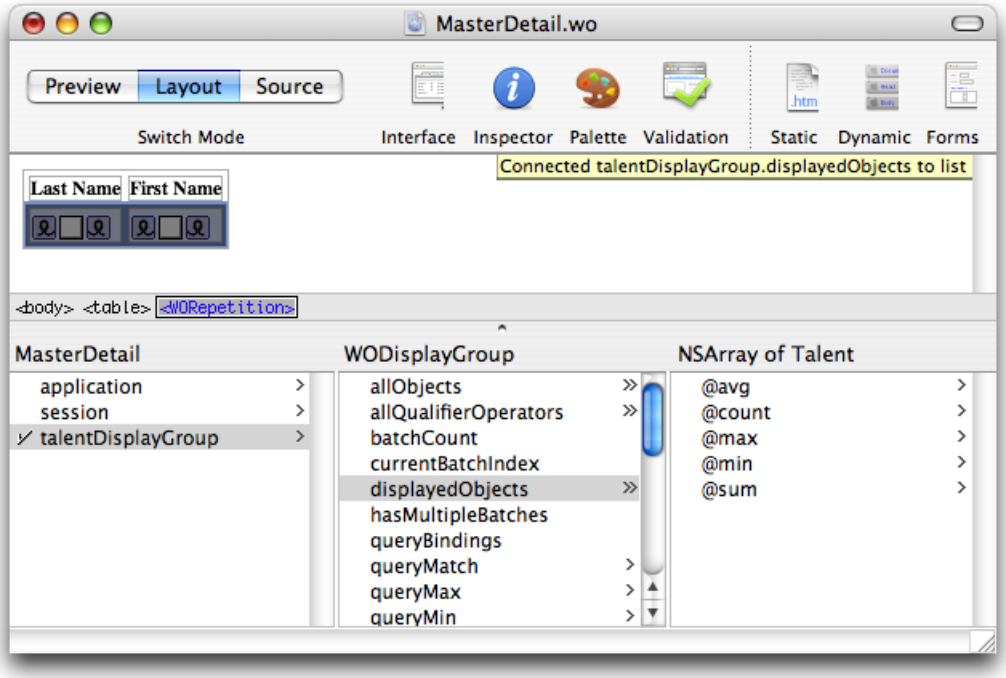
3. Type `Last Name` and `First Name` for the headings in the first and second columns of the table.

4. Double-click Cell in the second row of the first column and choose Dynamic > WOString.

5. Double-click Cell in the second row and second column and choose Dynamic > WOString.

Your web component window should look like Figure 5-4. Next you bind the repetition and dynamic elements.

**Figure 5-4** Master-detail interface




Follow these steps to bind the repetition and table content to your objects:

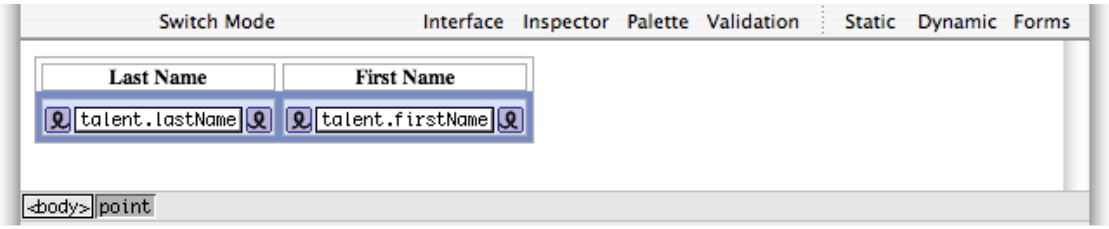
1. Click in a cell in the second row to show the WORepetition in the path view.
2. Drag from `talentDisplayGroup.displayedObjects` in the object browser to the `<WORepetition>` element in the path view and choose `list` from the menu to bind the rows in the table to the Talent records.
3. Add a `talent` key to your interface to reference the current object in the repetition by choosing Add Key from the Interface menu.

   A sheet appears prompting for more information.
4. Enter `talent` as the name, enter `Talent` as the type, and click Add.
5. Connect the `talent` key to the WORepetition by dragging from `talent` in the object browser to `<WORepetition>` in the path view and choosing `item` from the menu.
6. Bind the content of the table to your objects by dragging from `talent.lastName` in the object browser to the center box of the WOString in the graphical view.

   Dragging to the center box is a shortcut for binding the `value` attribute of the WOString.
7. Similarly, bind `talent.firstName` to the WOString in the second row and second column of the table.

When done the graphical view in Layout mode should look like the table in Figure 5-5.

**Figure 5-5** Master interface



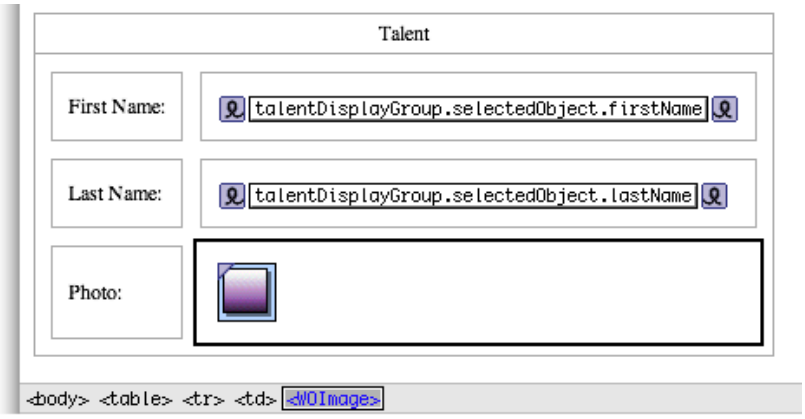

## Creating the Detail Interface

The detail interface displays information about the selected object. For example, create a detail interface that displays the first name, last name, and photo of the selected Talent record. Follow these steps to create this detail interface:

1. Insert a horizontal line after the master interface by choosing Static > Horizontal Rule.
2. Insert a table with three rows and two columns after the horizontal line by choosing Static > Table.
3. Change the text in the first column to these labels: `First Name`, `Last Name` and `Photo`.
4. Insert a WOString in the first and second row of the second column to display the talent's first and last name.
5. Insert a WOImage in the third row of the second column to display the talent's photo.
6. Bind `talentDisplayGroup.selectedObject.firstName` to the first WOString and `talentDisplayGroup.selectedObject.lastName` to the second WOString.
7. Bind `talentDisplayGroup.selectedObject.photo.photo` (an NSData object) to the `data` attribute of the WOImage.
8. Select the WOImage and open the inspector to set the `mimeType`. Set the `mimeType` to `"image/jpeg"`.

Figure 5-6 shows what your detail interface might look like.

**Figure 5-6** A detail interface




## Implementing a Select Action

At runtime, each row in the master interface table in Figure 5-4 (page 87) contains a Talent record. The `selectedObject` of the display group is expected to reference the currently selected object. However, since this interface is HTML, not a Java Client or Cocoa application, you have to implement a mechanism for selecting a row and setting `talentDisplayGroup.selectedObject`.

When you click in a row, either on a hyperlink or button, WebObjects stores the selected item, corresponding to that row, in the `item` attribute of the WORepetition. Hence, the application variable you bind to `item` stores the selected enterprise object. However, the value is only valid within the scope of the action method. After the action method returns, `item` may be used for something else—for example, to traverse the rest of the objects in the array when generating the HTML for the table. Therefore, your action method needs to record the selected object. You can record the selected object by setting the display group's `selectedObject` property to the value of `item`.

Follow these steps to implement a select action for the master-detail interface:

1. Choose Interface > Add Action.
2. Enter `selectAction` as the action name and click Add.

   If you don't select a component from the Component combo box menu, the action returns `null` and the component page is redisplayed with the selected talent details.
3. Control-click `selectAction` in the object browser and select "View selectAction" from the context menu.

   Xcode displays your component source code—for example, `MasterDetail.java`.
4. Using Xcode, implement the `selectAction` method as follows:

```
public WOComponent selectAction(){
        talentDisplayGroup.selectObject(talent);
```

```
            return null;
    }
```

Now you need to implement some mechanism for the action method to be invoked—implement a user interface for selecting a row in the table. The master-detail interfaces in Direct to Web applications use a inspect or edit button column in the table to do this. Alternatively, you could wrap the WOString that displays the last name with a WOHyperlink so that when the user clicks the last name, the action method is invoked. Follow these steps to wrap the last name with a WOHyperlink and bind it to the action method:

1. Select the last name WOString in the master interface.
2. Choose Dynamic > WOHyperlink to wrap the WOString with a hyperlink element.
3. Drag from `selectAction` in the object browser to the new WOHyperlink in the graphical or path view.

   A menu appears containing the WOHyperlink bindings.
4. Choose action from the menu.
5. Build and test your application.

# Creating a Detail Display Group

This interface is similar to a master-detail interface except that you use a second display group to manage the destination objects of a to-many relationship.

For example, in the Talent entity, there is a to-many relationship from Talent to MovieRole. If you want your interface to display the associated movie roles in a detail interface, you need to create a detail display group to manage the movie roles—you need two display groups, one per collection.

What other components you use to implement a detail interface depends on your application. Typically, you use a table and repetition to display a collection of objects in both the master and detail interfaces. The example in this section uses the same interface as shown in "Creating a Master-Detail Interface" (page 85) with the addition of movie roles added to the detail interface.

If you are following along, you can skip some of the steps below by modifying the previous example to use a detail display group.

## Creating a Master Display Group and Interface

Follow the same steps in "Creating a Display Group" (page 85) and "Creating a Master Interface" (page 86) to create your master display group and interface. When done, you should have `talent` and `talentDisplayGroup` variables added to your component. Also, follow the same steps in "Implementing a Select Action" (page 89) to implement the action method. Optionally, you can extend the master-detail example in "Creating a Master-Detail Interface" (page 85) to display the movie roles in the detail interface.

## Creating a Detail Interface for the To-Many Relationship

Follow the same steps in "Creating the Detail Interface" (page 88) to create your detail interface except replace the row that displays the photo with a row that displays the movie roles.

1. Enter `Movie Roles:` in the first column of the last row as a label.
2. Remove content from the second column of the last row and choose Static > Table to insert a table into that cell.

   A New Table panel appears prompting you for more information.
3. Enter `2` in the Rows and Columns text fields, select "First row cells are header cells" and "Second row is wrapped in a WORepetition," and click OK.
4. Enter `Role` as the heading for the first column and `Movie` as the heading for the second column.
5. Insert a WOString in each cell in the second row to display the role name and movie title respectively.

## Creating a Detail Display Group

Follow these steps to create a detail display group—for example, to display a collection of movie roles:

1. Double-click the `talentDisplayGroup`display group in the object browser.

   The Display Group Options panel appears as shown in Figure 5-3 (page 84) except that the entity is Talent.
2. Select "Has detail data source."

   The Master Entity pop-up list is enabled. It lists all entities in the models contained in your project.
3. Select the Master Entity from the pop-up list or type it in the text field—for example, select Talent.

   The Detail Key pop-up list now contains all the to-many keys belonging to the master entity.
4. Select the Detail Key from the pop-up list or type the key name in the text field—for example, select `roles`, the to-many relationship from Talent to MovieRole.
5. Click OK.

   Display groups that have detail data sources have a `masterObject` attribute. Select `talentDisplayGroup.masterObject` in the object browser to see properties of the master object. You use the master object to bind elements in the detail interface. For example, bind elements to the collection of movie roles.

Follow these steps to bind the detail repetition to the to-many `roles` relationship in the master object:

1. Click in a cell in the second row of the detail table to show the WORepetition in the path view.

2. Drag from `talentDisplayGroup.masterObject.roles` in the object browser to the `<WORepetition>` element in the path view and choose `list` from the menu to bind the rows in the table to the MovieRole records.

3. Add a `role` key to your interface to reference the current object in the repetition by choosing Add Key from the Interface menu.

   A sheet appears prompting for more information.

4. Enter `role` as the name, enter `MovieRole` as the type, and click Add.

5. Connect the `role` key to the WORepetition by dragging from `role` in the object browser to `<WORepetition>` in the path view and choosing `item` from the menu.

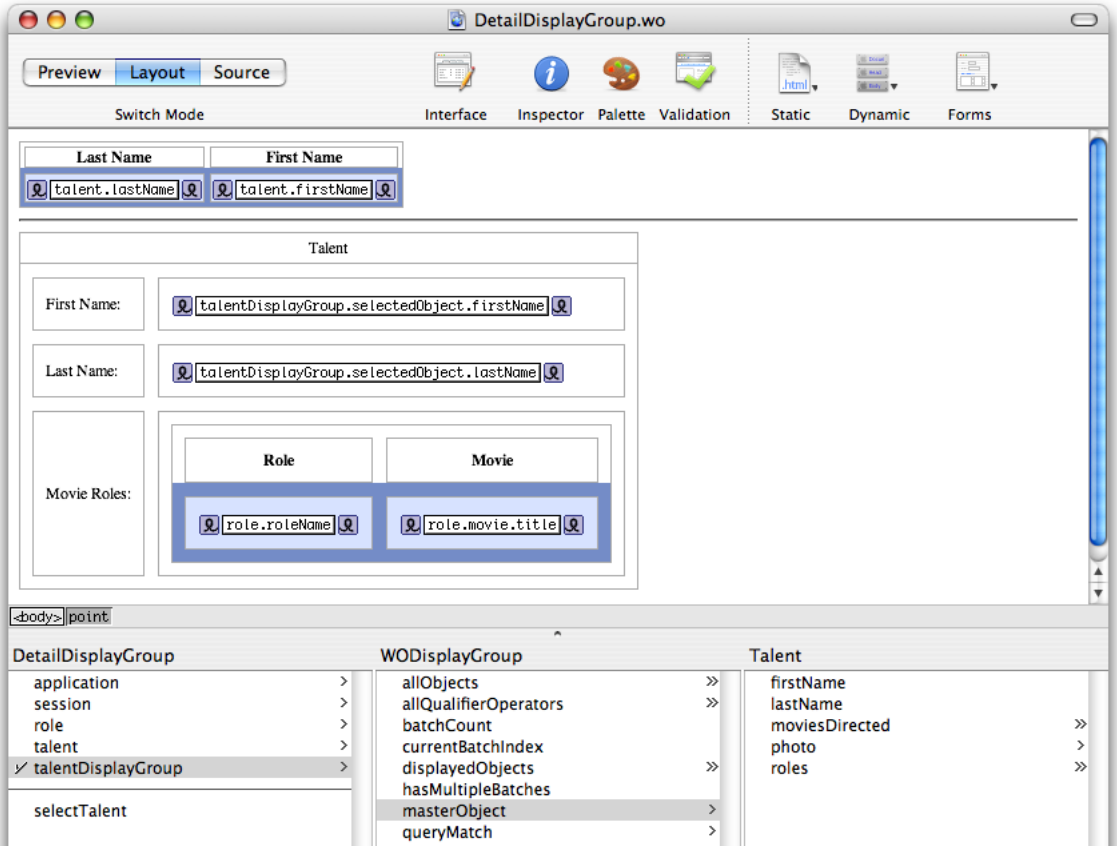
6. Bind the content of the table to your objects by dragging from `role.roleName` in the object browser to the center box of the WOString in the graphical view.

   Dragging to the center box is a shortcut for binding the `value` attribute of the WOString.

7. Similarly, bind `role.movie.title` to the WOString in the second row and second column of the table.

   When done the graphical view in Layout mode should look like Figure 5-7.

**Figure 5-7** A complete detail display group interface



You can also create a detail display group by dragging a to-many relationship from EOModeler or the EO Model Xcode design tool to your component window. Follow the steps in "Adding Display Groups" (page 81) to create a display group. When the Display Group Options panel appears it will already be configured as a detail display group for the dragged relationship. Just click OK to create the display group.

# Working With Palettes

A palette is a collection of resources (such as images, static or dynamic HTML elements, and components) that you can drag to your component—items on a palette are reusable. There are several predefined palettes described in this chapter but you can also create your own custom palettes and drag elements from a component to your palette.

Regardless of which palette you use, the steps for adding palette items to your component are the same. Follow these steps to add a palette item to your component:

1. Choose Window > Palette or select Palette from the toolbar to open the Palette window.
2. Select the palette you want from the buttons at the top of the Palette window.

   The window title reflects the name of the selected palette.
3. Drag an element from the palette to your component window at the insertion point.

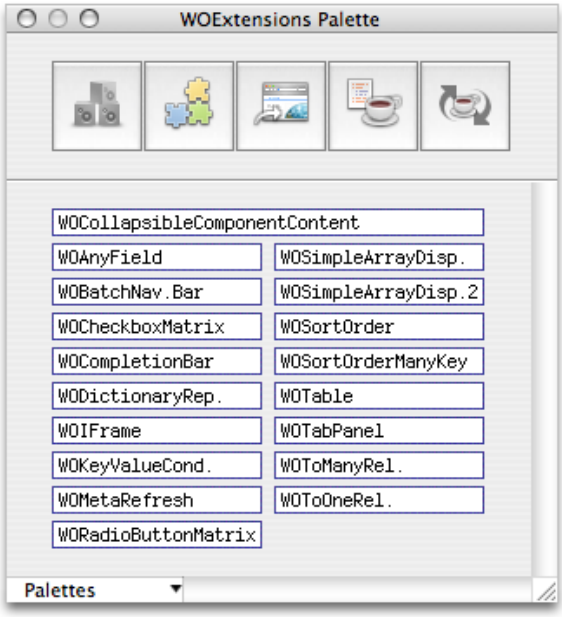
   The palette item is added to your component.
4. Use the inspector to configure any attributes—bind them to the variables and actions in your component.

## WOExtensions Palette

The WOExtensions palette, shown in Figure 6-1, contains a number of reusable components provided by the WebObjects Extensions Framework. Read *WebObjects Extensions Reference*for a detailed description of all the components in this framework.

Because these components are examples, you can also examine the source code for the WebObjects Extensions Framework, located in `/Developer/Examples/JavaWebObjects/Source/JavaWOExtensions`. Examining these components is an excellent way to learn more about creating your own reusable components. Since you have the source code, you can also extend these components. Note that some of these components or similar components are used to construct Direct to Web pages.

**Figure 6-1** The WOExtensions palette




These are the components that appear on this are:

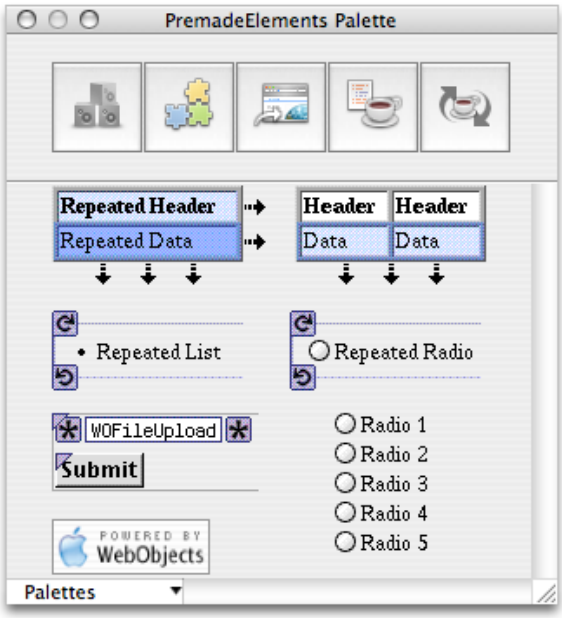
- WOCollapsibleComponentContent allows you to expand or collapse content, as in a hierarchical view of content similar to the outline view in Cocoa.
- WOAnyField can be used on a query page to filter enterprise objects by an attribute value.
- WOBatchNavigationBar can be used on a list view to control the number of enterprise objects fetched in a batch, and fetch the next and previous batches.
- WOCheckboxMatrix displays a multicolumn array of checkboxes.
- WOCompletionBar displays a progress bar on a page.
- WODictionaryRepetition is similar to WORepetition except iterates a dictionary of key-value pairs.
- WOIFrame is a dynamic element that corresponds to the `<iframe>` tag. Use WOFrame if you want a regular frame corresponding to the `<frame>`tag.
- WOKeyValueConditional is similar to WOConditional except that it displays content depending on the value of the parent component's key. This component is used in conjunction with WOTabPanel.
- WOMetaRefresh inserts a refresh tag into your page that instructs the user's browser to display a new page after a specified time interval.
- WORadioButtonMatrix displays a multicolumn array of radio buttons.
- WOSimpleArrayDisplay displays the specified number of objects in an array in a single-column table. If the entire array does not fit in the table, the last item is a hyperlink to a page that displays all the objects.
- WOSimpleArrayDisplay2 displays all the objects in an array in a single-column table. If all the objects do not fit in the table, an inspect button is displayed which links to a page that displays all the objects.
- WOSortOrder uses multiple buttons to allow the user to specify the sort order of objects in a display group—for example, unsorted, ascending, or descending.

- WOSortOrderManyKey is similar to WOSortOrder except that the user can select a destination key to sort by—for example, sort by last name or first name.
- WOTable is a container similar to WORepetition but displays the contents in a multicolumn table. This component is used to display a matrix of checkboxes or radio buttons.
- WOTabPanel displays a tab panel. When the user selects a tab the corresponding panel appears.
- WOToManyRelationship allows the user to edit the destination objects of a to-many relationship.
- WOToOneRelationship allows the user to edit the destination object of a to-one relationship.

# PremadeElements Palette

The PremadeElements palette, shown in Figure 6-2, contains some commonly used components built from other elements. For example, elements that you would use in a repetition. This palette is provided for your convenience to reduce repetitive actions when designing your component.

**Figure 6-2** The PremadeElements palette




These are the components on the palette:

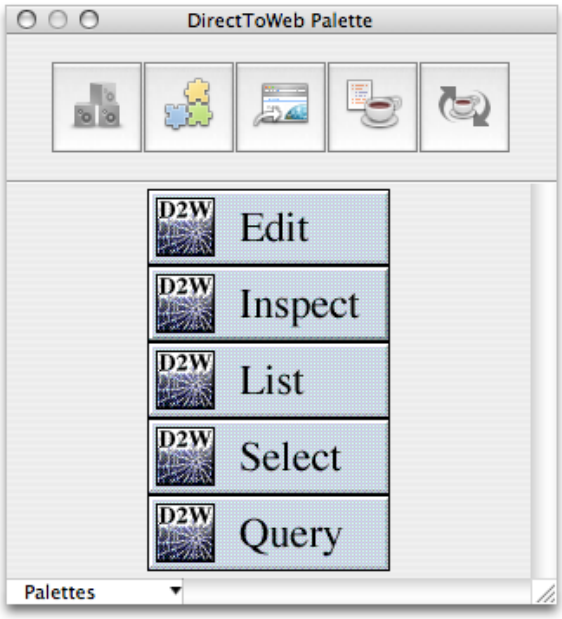
- Repeated Header is a table that uses repetitions to expand in both the horizontal and vertical directions.
- Repeated Table is a table that uses a repetition to expand in the vertical direction—the first row is a header and the second row is wrapped in a repetition.
- Repeated List is list wrapped in a repetition.
- Repeated Radio Button is a radio button wrapped in a repetition.
- Upload Form is a component containing a WOFileUpload and Submit button.

- Multiple Radio Buttons is a component containing multiple radio buttons.
- WebObjects Logo is a graphic you can add to your webpage.

# DirectToWeb Palette

The DirectToWeb palette, shown in Figure 6-3, contains Direct To Web pages. You can use these components in any page as long as your project includes the Direct To Web frameworks.

**Figure 6-3** The DirectToWeb palette



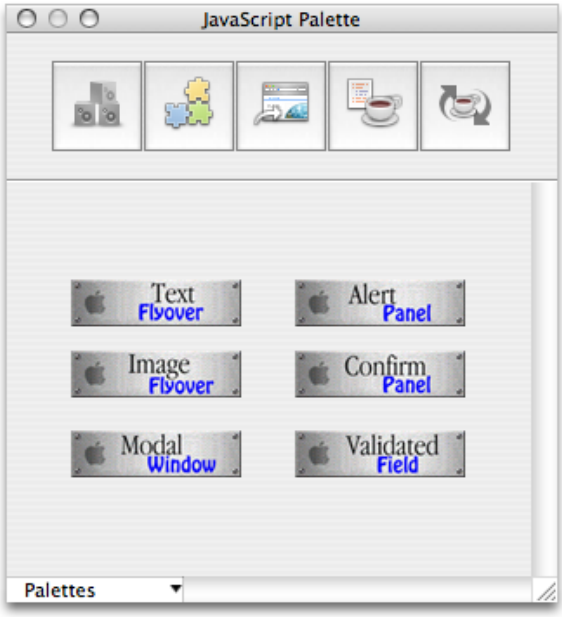

These are the components on the palette:

- D2WEdit is used to edit a single enterprise object.
- D2WInspect is used to view a single enterprise object.
- D2WList is used to view a collection of enterprise objects, as in a master interface.
- D2WSelect is used to display the selected object, as in a detail interface.
- D2WQuery is used to query or search for enterprise objects.

# JavaScript Palette

The JavaScript palette, shown in Figure 6-4, contains all the JavaScript components provided by the WebObjects Extensions Framework.

**Figure 6-4** The JavaScript palette




These are the components on the palette:

- Text Flyover displays a hyperlink containing text that changes color when the user passes the mouse over it.
- Image Flyover displays an active image that changes to another image when the user passes the pointer over it.
- Modal Window displays a hyperlink and when clicked opens a modal panel.
- Alert Panel displays a hyperlink and when clicked opens an alert panel.
- Confirm Panel displays a hyperlink and when clicked opens a confirm panel. The confirm panel displays a message and an OK button. The panel disappears when the user clicks the OK button.
- Validated Field is similar to WOTextField—it needs to be placed in a WOForm—except that the text in the field is validated before the form is submitted.

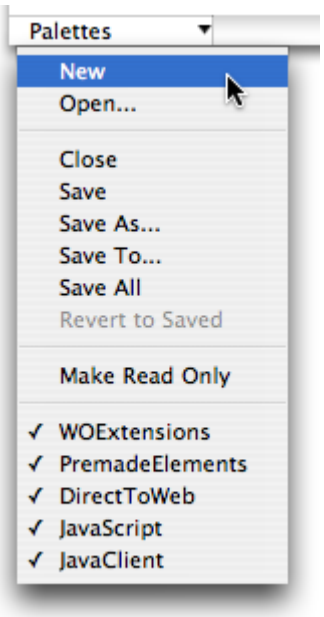
# Custom Palettes

You can create your own custom palettes to store frequently used items, such as forms, tables, or images, and you can load palettes created by third parties. This section explains how to create, add, remove, and edit custom palettes.

## Creating Palettes

*To create a new palette*, choose New from the Palettes pop-down menu at the bottom-left of the Palette window shown in Figure 6-5. The palette appears in the palette window represented by a generic palette icon. Read "Changing Palette Icons" (page 101) to change the palette's icon.

**Figure 6-5** Palettes pop-down menu




Follow these steps to add an item to a palette:

1. If your palette background is grey, choose Palettes > Make Editable to make it editable.

   A newly created palette is editable by default. If the background is white, it is editable. If the background is grey, it is read only. You cannot edit standard palettes.

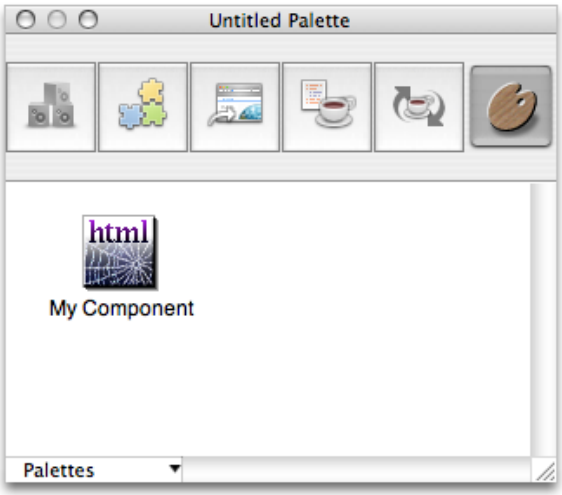
2. Select the element or elements that you want to add to the palette in the component window.

3. Option-drag the selection in the component window to the editable palette.

   A generic HTML icon appears on the palette labeled "Untitled."

4. You can change the title of the item by selecting its name and typing as show in Figure 6-6.

**Figure 6-6** Adding items to palettes




5. Read "Changing Palette Icons" (page 101)to change the item's icon.

You can also add any item from the file system to a palette (including such things as a component, an image, or an EOModel). Follow these steps to add items from the file system:

1. Make your palette editable.
2. Locate the item using the Finder.
3. Drag the item from the Finder onto the palette.

   For example, to add a component to a palette, you drag the `.wo` folder to the palette.

To remove an item from a palette, simply drag it off the palette while the palette is editable.

When you are done adding items to your palette, choose Palettes > Save, Save As, Save To, or Save All. A sheet appears prompting you for a palette name and location to save it.

To open an existing palette and add it to the Palette window, choose Palettes > Open.

To close or remove a palette from the palettes window, choose Palettes > Close.

## Changing Palette Icons

You can change the icon of any palette or item on a palette following these steps:

1. Open the palette window and select the palette whose icon you want to change.
2. Make the palette editable.

3. Drag an image from the file system onto the palette's icon or an item's icon.

   You can use any image file recognized by WebObjects Builder (such as a GIF, TIFF or JPEG files).

4. Save the palette.

## Using Palette Items

The steps for using palette items from your custom palettes are the same as using items from any type of palette. However, you might want to make your palette read only first. If the background of your palette is white, choose Palettes > Make Read Only.

# Custom Components

Custom components are reusable components that have some business logic and custom bindings. Any component that you define, whether it represents an entire page or part of a page, can be reused by any HTML-based WebObjects application and in any other WebObjects component. A reusable component can be used in multiple pages or even multiple times in the same page.

Typically, you use custom components to implement headers, footers, and any custom user interface component, such as a navigation bar, that you use repeatedly in your pages.

For example, all of the components in the WebObjects Extensions framework are custom components and are an excellent source of sample code. The source code for the `JavaWOExtensions.framework`is located in `/Developer/Examples/JavaWebObjects/Source/JavaWOExtensions`.

Custom components are partial documents defined in a single `.wo` folder. You reuse them by adding them first to your project and then to your component. You can add the `.wo` folder to your project directly or add a framework containing the component to your project.

You add custom components to your component using the custom WebObjects element in WebObjects Builder. You can also create palettes containing custom components and drag them from the palette to your component. However, not all palette items are custom components. Some palette items, like the Repeated Header on the PremadeElements palette, are just components built from existing elements.

This chapter explains how to build custom components and reuse them in your applications.

## Creating Custom Components

First you create your component in WebObjects Builder and add your business logic to the Java file. You can use the WOComponentContent and WOSwitchComponent elements to help you construct complex components. If you have custom bindings, you use the API Editor in WebObjects Builder to specify them.

### Creating Custom Components

Create a new component using WebObjects Builder and add it to your project.

Custom components that are added to other components must be partial documents. This way WebObjects Builder does not wrap `<HTML>`, `<HEAD>`, and `<BODY>` tags around the component when it is added to the parent component. Read "Body and Document" (page 45) for how to change a component to a partial document.

## Using the WOComponentContent Element

If your component is a container, use WOComponentContent to represent the content that is placed in the container. Since WOComponentContent is just a placeholder, it has no bindings. For example, `WOCheckboxMatrix.wo` in the WebObjects Extensions framework uses a WOComponentContent element to represent the content that appears in the table cells as show in Figure 7-1.

**Figure 7-1** The graphical view of WOCheckboxMatrix

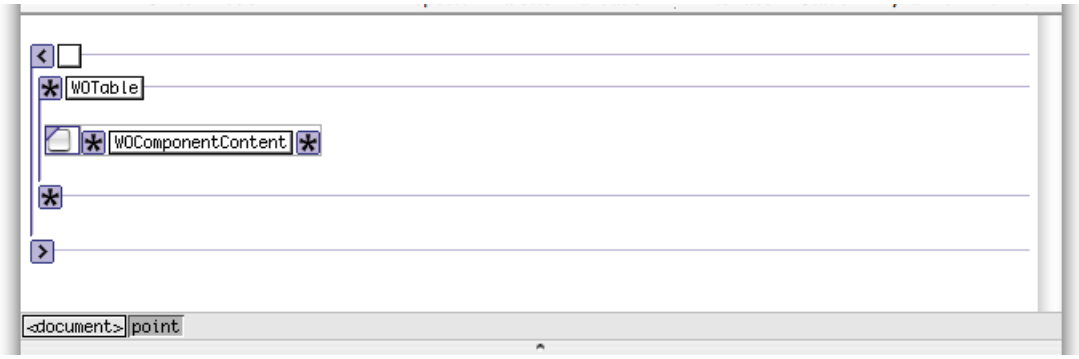
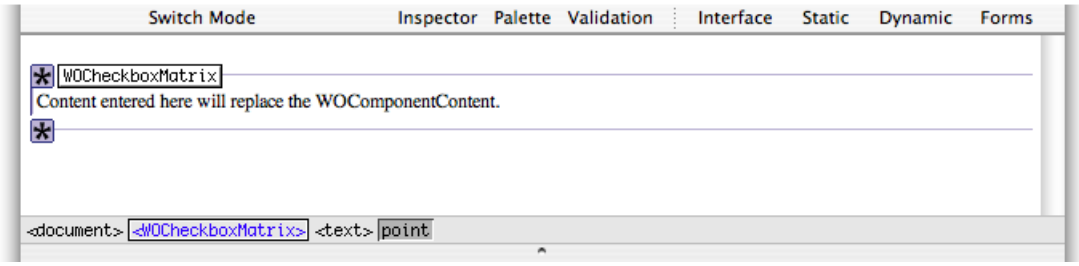



Figure 7-2 shows what a WOCheckboxMatrix looks like when added to your component. You add a WOCheckboxMatrix by dragging it from the WOExtensions palette. Any content you enter inside the WOCheckboxMatrix boundary replaces the WOComponentContent in the `WOCheckboxMatrix.wo` when rendered. Hence, you can use only one WOComponentContent per custom component.

**Figure 7-2** Adding a WOCheckboxMatrix




## Using the WOSwitchComponent Element

WOSwitchComponent is similar to WOComponentContent except that WOSwitchComponent can have dynamic content. WOSwitchComponent has a single binding called `WOComponentName`. You bind WOComponentName to a variable or method in your component that returns a String. Using a WOSwitchComponent is useful when you have more than two choices for the content of your component. If you have just two choices, you can use a WOConditional.

Another advantage of using a WOSwitchComponent is that any additional bindings added to the element are passed on to the chosen component (the component that replaces the WOSwitchComponent when rendered). Hence, all the components that might replace the WOSwitchComponent should be key-value coding compliant—respond to `takeValueforKey()` without throwing an exception.

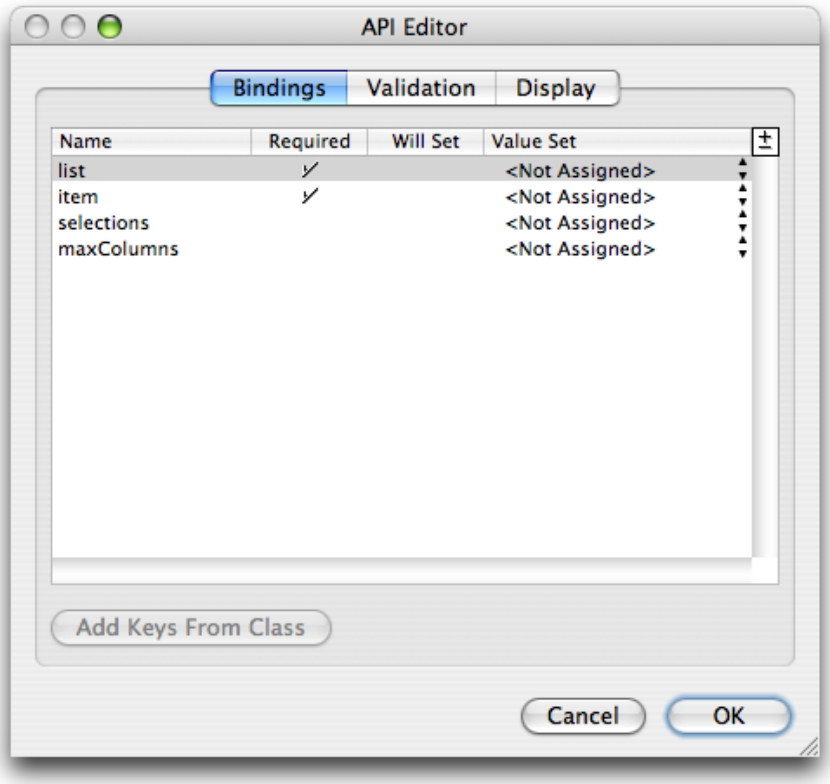
## Using the API Editor

You use the API Editor to define your custom bindings. You can specify the binding name, the value type, and validation rules, You can even specify an image to display when your component is added to another component. Follow these steps to configure your component using the API Editor:

1. Choose Window > API

   The API Editor window appears. Figure 7-3 shows what the API Editor window looks like for the WOCheckboxMatrix component.

   **Figure 7-3** The API Editor




2. Select "Add binding" from the +/- menu in the upper-right corder of the table to add a binding.
3. Enter the name of the binding in the Name column.
4. Click in the Required column if the binding is required.
5. Choose a value from the Value Set pop-up menu if you know or want to restrict the type.
6. Click Validation to configure additional validation checks.
7. Click Display to set the image representation of the component.

# Adding Custom Components to Your Project

Before using a component it needs to be added to your project. You can either add the component to a framework and include that framework in your project, or add the component directly to your project.

Some WebObjects frameworks—for example, `JavaWOExtensions.framework` and `JavaDirectToWeb.framework`—already contain custom components that you can reuse. If these frameworks are added to your project, their components are already available. Read *Xcode 2.2 User Guide*for more information on creating your own frameworks.

Follow these steps to add an existing component directly to your project:

1. Drag the component (a folder with a `.wo` extension) from the Finder or Desktop to your component window.

   A panel appears asking if you want to add the component to your project.

2. Click Yes.

   The component is copied to your project folder and added to the Web Components group in Xcode. A custom element appears in your component window at the insertion point.

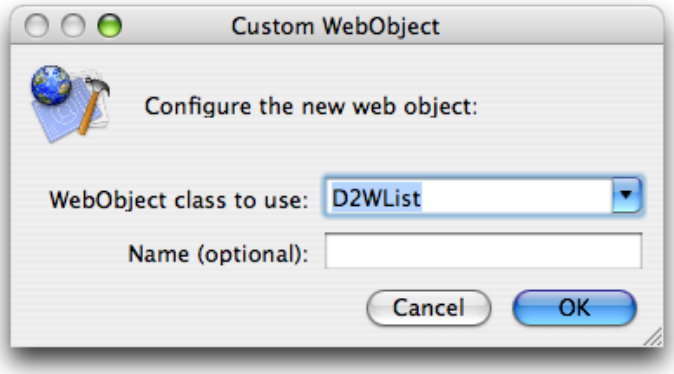
3. Select the component and open the inspector to bind its attributes.

Follow these steps to add a custom element to your component:

1. Choose Dynamic > Custom.

   A Custom WebObject panel appears prompting you for more information as show in Figure 7-4.

   **Figure 7-4** The custom WebObject panel




2. Select the web component class from the "WebObject class to use" combo box menu.

   The menu contains all components that are in the current project and its frameworks. For example, the WOAnyField and WOCheckBox components contained in the menu are defined in `JavaWOExtensions.framework`.

3. Alternatively, you can enter the component's class name in the combo box text field.

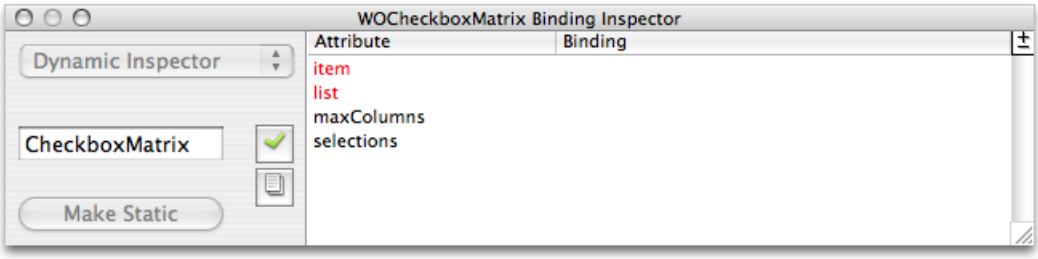
4. Optionally, enter the component's name in the Name text field.

   WebObjects Builder gives each instance of a component a unique name that appears in the `.wod` file that appears in Source mode. Enter a name in the Name field if you want to use a different logical name.

Many of the standard palettes, described in "Working With Palettes" (page 95) also contain reusable components. To add a custom element from a palette, drag the element from the palette to your component window. Read "Working With Palettes" (page 95)for how to create your own palettes.

A component that is designed for reuse can export keys and actions, which become attributes that you can bind, just as you would bind the attributes of any other dynamic element. When the component is added to your component, these attributes show up in the inspector as shown in Figure 7-5. The `item` and `list` bindings appears in red because they were specified as required using the API Editor in Figure 7-3 (page 105).

**Figure 7-5** The WOCheckboxMatrix Binding Inspector




If the custom bindings do not appear, read "Adding Bindings" (page 61) for steps to add them using the inspector. If you are adding your own custom component, then read "Using the API Editor" (page 105).

# Validating

This section describes how to validate your web component. Typically, the validating process identifies common errors in your web component such as missing bindings, wrong location of elements, and wrong tags. However, it can't catch all errors in your component, just those that don't conform to the HTML and WebObjects specifications. You can validate individual elements or an entire web component.

## Validating Elements Using the Inspector

You can validate individual dynamic elements using the inspector as follows:

1. Select the element.
2. Click Inspector on the toolbar to open the inspector window.
3. Click the verify button (the green checkmark).

   An HTML Validation window appears detailing the binding errors. The text view is blank if there are no errors.
4. If errors are listed in the text view, click them individually to view the elements or HTML that contains the errors.

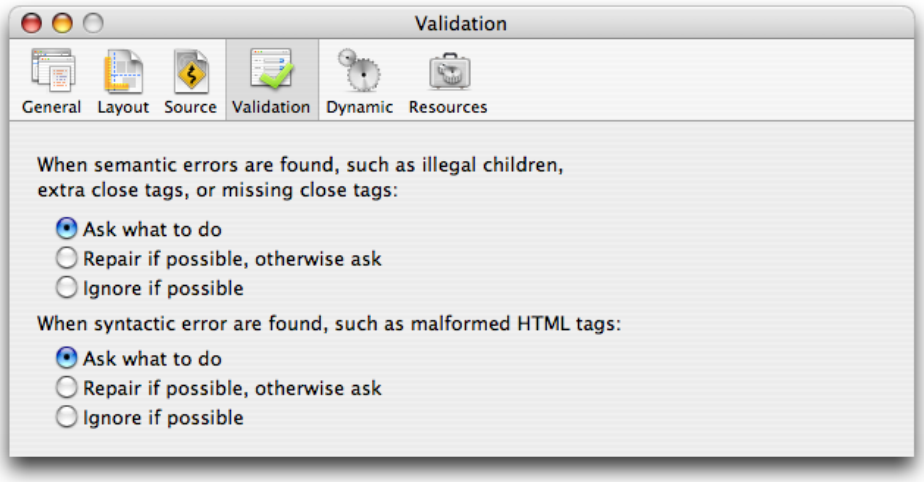
## Validating the Entire Web Component

The steps to validate the entire web component are identical to validating individual elements except that you do not need to select an element. Just choose Window > Validation to validate your component and errors will be displayed in the HTML Validation window.

## Validating Preferences

The validation process can repair some errors, if possible, or ignore them. Choose WebObjects Builder > Preferences to open the preferences window, and click Validation to view the preferences as show in Figure 8-1. Use this pane to configure exactly how you want semantic and syntactic errors handled.

**Figure 8-1** Validation preferences



# Document Revision History

This table describes the changes to *WebObjects Builder User Guide*.

| Date | Notes |
| --- | --- |
| 2005-11-09 | New document that describes WebObjects Builder, a tool for creating dynamic webpages for HTML-based web applications. |

Document Revision History