Getting Started With WebObjects

This manual describes WebObjects 4.5.

Writing: Clif Liu, Ron Karr, and Kelly Toshach
Editing: Laurel Rezeau and Jeanne Woodward
With help from: Andy Belk, Craig Federighi, John Graziano, Ben Haller, Kenny Leung, Charles Lloyd, Jean Ostrem, Becky Willrich, Greg Wilson
Graphic Design: Karin Stroud
Production: Gerri Gray

Contents

Table of Contents

Preface

About WebObjects

WebObjects is an object-oriented environment for developing and deploying World Wide Web applications. A WebObjects application runs on a server machine and receives requests from a user's web browser on a client machine. It dynamically generates HTML pages in response to the user's requests. WebObjects provides a suite of tools for rapid application development, as well as prebuilt application components and a web application server.

WebObjects is flexible enough to suit the needs of any web programmer. You can write code using one of three programming languages: Java, Objective-C, or WebScript. You can write simple WebObjects applications in a matter of minutes. For more complex projects, WebObjects makes it easy by performing common web application tasks automatically and by allowing you to reuse objects you've written for other applications.

About This Book

This book contains three tutorials that help you learn what WebObjects is and how to use it:

- Chapter 1, "Creating a Simple WebObjects Application" (page 13), teaches you the basic concepts and steps involved in creating a WebObjects project, using the Project Builder and WebObjects Builder tools. You'll create a simple application that takes input from a user and displays it.

- Chapter 2, "Enhancing Your Application" (page 39), extends the capabilities of your application and shows you additional techniques you use when working with WebObjects.

- Chapter 3, "Creating a WebObjects Database Application" (page 63), teaches you how to create a more complex application, one that accesses a database.

WebObjects can run on several platforms. Screen shots in this book are for Windows NT systems; if you are running on a different platform, the look of your windows may vary slightly.

Where to Go From Here

After you have worked through the tutorials in this book, you should have a good working knowledge of WebObjects. For more in-depth information about how WebObjects works, read the *WebObjects Developer's Guide*.

Other valuable information about WebObject is available on the WebObjects CD. You can access all online information through the WebObjects home page. In particular, the WebObjects home page gives you access to some books that are available only on the WebObjects CD:

- *WebObjects Tools and Techniques* is a more comprehensive guide to using Project Builder and WebObjects Builder to develop WebObjects applications. It also discusses Direct to Web, a framework that instantly creates a WebObjects database application.
- *Serving WebObjects* describes how to administer and deploy WebObjects applications after you've written them.
- The *Dynamic Elements Reference* documents the dynamic elements provided with WebObjects and provides examples of how to use them.
- The *WebObjects Framework Reference* provides a complete reference to the classes in the WebObjects framework. Reference material is provided for both the Java and Objective-C languages.

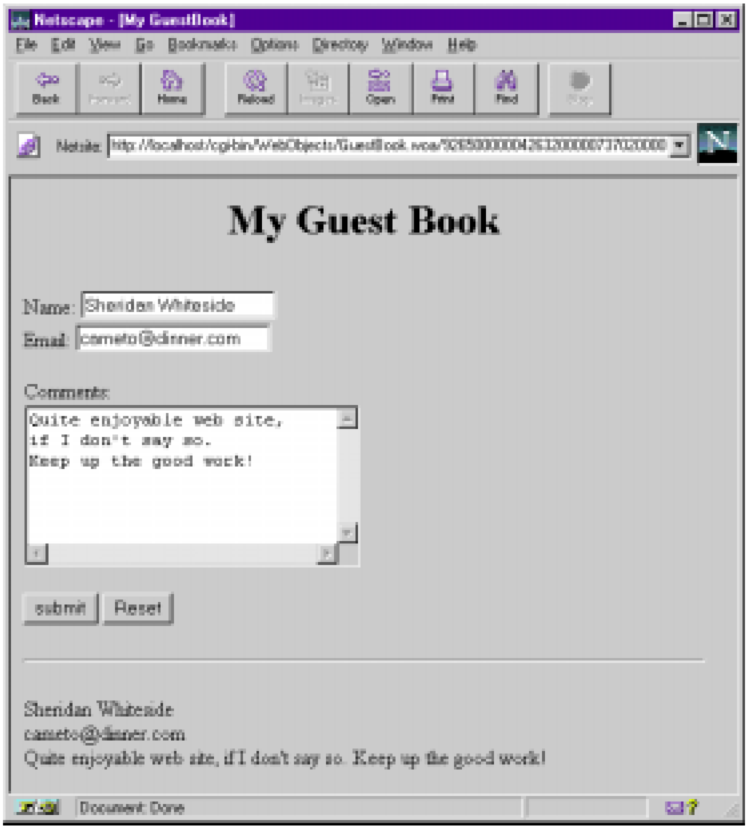
Additionally, for more information on Enterprise Objects Framework, read the *Enterprise Objects Framework Developer's Guide*. This book provides in-depth information about how Enterprise Objects Framework works and about techniques for developing database applications with it.

Chapter 1

Creating a Simple WebObjects Application

This chapter introduces you to the basic concepts and procedures of developing WebObjects applications. You'll develop, in stages, a simple application for the World Wide Web. The application you'll write is called GuestBook.

When you've finished the steps in this chapter, your application will have a single web page containing a form that allows users to enter their names, e-mail addresses, and comments. When the form is submitted, the application redraws the page with the user's information at the bottom.


Netscape - [My GuestBook]
File Edit View Go Bookmarks Options Directory Window Help
Back
Forward
Home
Reload
Images
Open
Print
Find
Stop
Netsite:
http://localhost/cgi-bin/WebObjects/GuestBook.woa/
Document: Done

My Guest Book

Name: Sheridan Whiteside

Email: cameto@dinner.com

Comments:

```
Quite enjoyable web site,
if I don't say so.
Keep up the good work!
```

submit Reset

Sheridan Whiteside
cameto@dinner.com
Quite enjoyable web site, if I don't say so. Keep up the good work!

In Chapter 2, "Enhancing Your Application" (page 39), you will add features to the application, including a second page, a table that displays information from multiple users, and hyperlinks.

This application illustrates the basic techniques you use to create a WebObjects application. You'll use two primary tools, Project Builder and WebObjects Builder.

Project Builder is an integrated software-development application. It contains a project browser, a code editor, build and debugging support, and many other features needed to develop an application. In this tutorial, you'll learn to use Project Builder to:

- Create a new WebObjects application project.
- Write scripts or compiled code to provide behavior in your application.
- Build and launch your application.

WebObjects Builder is an application that provides graphical tools for creating dynamic web pages (*components*). A web page consists of *elements*. WebObjects Builder allows you to add most of the common HTML elements to a component by using its graphical editing tools. In addition, WebObjects allows you to create *dynamic elements*, whose look and behavior are determined at runtime. You'll learn to use WebObjects Builder to:

- Create static content for your pages.
- Add dynamic elements to your pages.
- Bind the dynamic elements to variables and methods in your code.

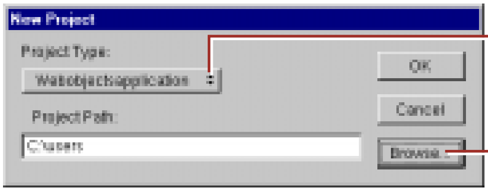
Creating a WebObjects Application Project

A WebObjects application project contains all the files needed to build and maintain your application. You use Project Builder to create a new project.

1. Launch Project Builder.

   On Mac OS X Server, choose Project Builder from the Apple menu under WebObjects. On Windows NT, you can launch Project Builder from the WebObjects program group in the Start menu.

2. Choose Project ▸ New.



New Project

Project Type: Webobjectsapplication

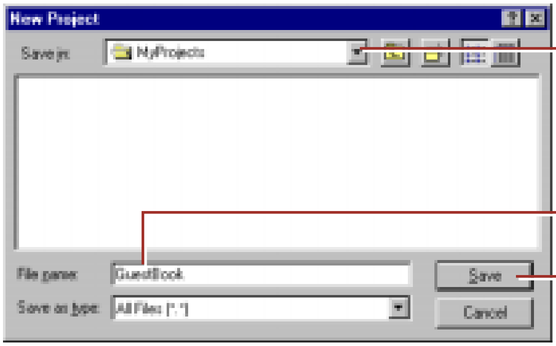
Project Path: C:\users

Set project type here.

Click to choose directory in which to create your project.

3. In the New Project panel, select Webobjectsapplication from the Project Type pop-up list.

4. Click Browse.



New Project

Save in: MyProjects

File name: GuestBook

Save as type: All Files (*.*)

Choose a directory here.

Type project name here.

Click when finished.

5. Navigate to the directory where you want to save the project.

6. Type the name of the project you want to create (`GuestBook`).

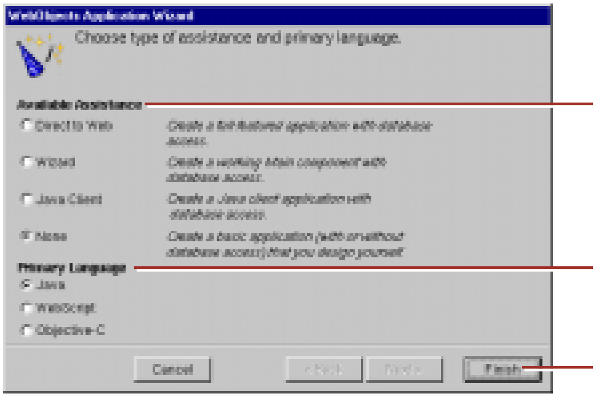
7. Click Save.

   The New Project panel shows the path you specified.

8. Click OK.

Note: On Mac OS X Server, the file browser and the Project Type pop-up list both appear on the New Project panel.

The WebObjects Application Wizard starts.



WebObjects Application Wizard

Choose type of assistance and primary language.

Available Assistance

○ Direct to Web — *Create a full-featured application with database access.*

○ Wizard — *Create a working Main component with database access.*

○ Java Client — *Create a Java client application with database access.*

● None — *Create a basic application (with or without database access) that you design yourself.*

Primary Language

● Java

○ WebScript

○ Objective-C

Choose level of assistance.

Choose programming language.

Click to proceed.

9. For Available Assistance, choose None.

   If you are developing an application that accesses a database, you may wish to use one of the levels of assistance that WebObjects provides. For more information on these options, see Chapter 3, “Creating a WebObjects Database Application” (page 63).

Choosing the Programming Language

WebObjects supports three languages:

- Java
- Objective-C
- WebScript

Java and Objective-C are *compiled* languages. They require you to build your application before running it. WebScript, which is based on Objective-C, is a *scripted* language. It allows you to make changes to your application while it is running.

When you create a new project, Project Builder provides you with a *component* called Main. In WebObjects terminology, a component represents a page in your application (or possibly part of a page).

In the wizard, you specify the language you’ll use to program your Main component, as well as the *application* and *session* code files (which will be described later).

1. For the primary language, select Java.

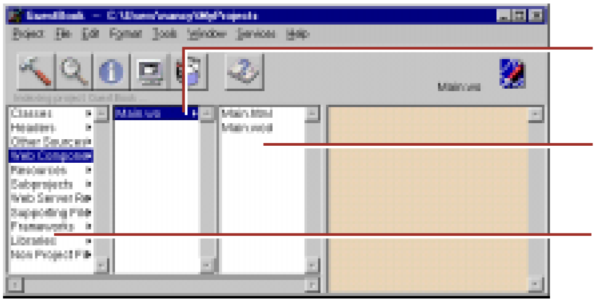
   Later, you'll create an additional component for your application and write its code in WebScript.

2. Click Finish.

   Project Builder creates a new application directory called **GuestBook**. This directory contains the files you work with in both Project Builder and WebObjects Builder.

Examining Your Project

Project Builder displays a browser showing the contents of your project. The first column lists several categories of files that your project may contain. This section describes some of the most important files you'll use.



Your project's components.

Files in the selected component.

Categories ("suitcases") of project resources.

1. Select Web Components.

   The next column displays a list with one element, **Main.wo**, which is a directory containing the first *component* in your application. Every application starts with a component called Main.

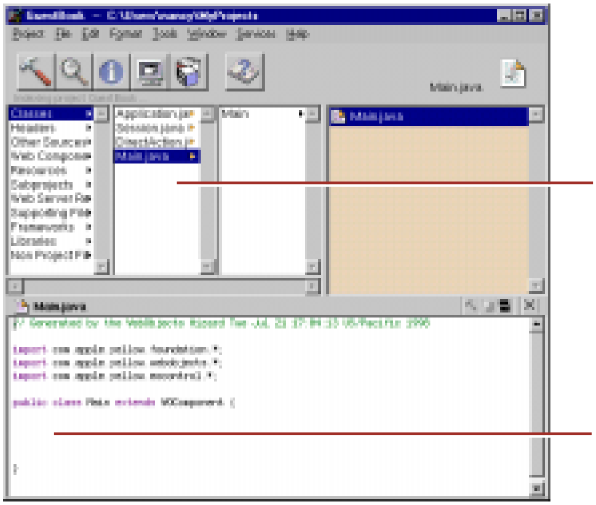
2. Select **Main.wo**.

   The files you see displayed in the next column are some of the files you work with when developing your component:

   - **Main.html** is the HTML template for your page. It can include tags for dynamic WebObjects elements as well as regular HTML. Typically, you do not edit this file directly; you create your page's elements graphically using WebObjects Builder.

- **Main.wod** is the declarations file that specifies bindings between the dynamic elements and variables or methods in your scripts. Normally, you don't edit this file directly; you use WebObjects Builder to generate the bindings for you.

3. Select Classes in the first column of the browser.



```
// Generated by the WebObjects Wizard Tue Jul 21 17:04:33 US/Pacific 1998

import com.apple.yellow.foundation.*;
import com.apple.yellow.webobjects.*;
import com.apple.yellow.eocontrol.*;

public class Main extends WOComponent {



}
```

Your application's Java classes.

The Main component's code goes here.

You'll see these files listed in the second column:

- **Main.java** is a file that allows you to specify behavior associated with the component. You do this by writing code in Java (since you specified Java as the language when you created the project). You use Project Builder to edit this file.

- **Application.java** and **Session.java** are other Java files that you may want to work with. **Application.java** defines *application variables* that live as long as the application does. **Session.java** defines *session variables* that exist for the lifetime of one user's session. In Chapter 2, you'll add code to **Application.java** and learn more about application and session variables.

- **DirectAction.java** defines a subclass of WODirectAction that you use as a container class for your action methods. You can rename this class or create multiple subclasses of WODirectAction depending on your application needs.

Launching WebObjects Builder

Now that you've created your project, you'll edit the Main component with WebObjects Builder.

1. Select Web Components in the first column of the browser.

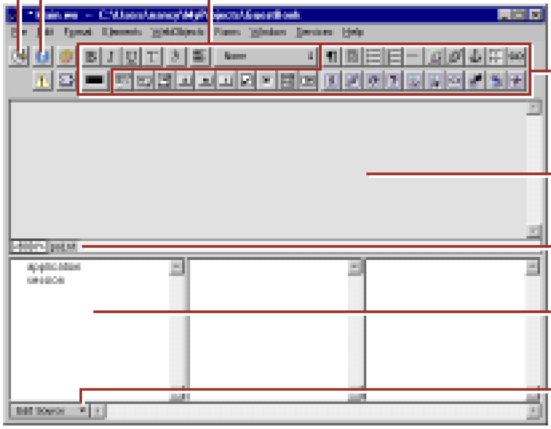
2. Double-click **Main.wo** in the second column.

   The WebObjects Builder tool launches and displays a window titled **Main.wo**. This represents your application's Main component.

Pop-up list switches editing modes.

Click to inspect selected element.

These buttons change properties of selected elements, or text.



Click one of these buttons to create a specific element.

This window displays your component's elements graphically.

Click here to select another element in the hierarchy.

Object browser shows variables and methods in your application's code

Pull-down list lets you add variables, and methods to

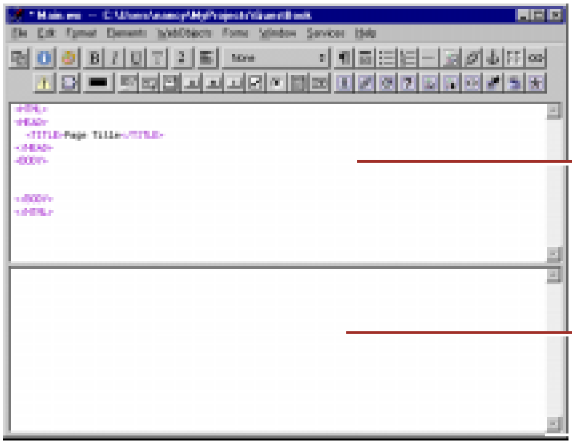
You create your component graphically in the upper pane of the component window. The browser at the bottom of the window (known as the *object browser)* is used to display variables and methods your component uses. Note that there are two variables already defined, **application** and **session**. You'll create others later.

The *path view* lies between the upper pane and the object browser and shows 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. Here, the path view shows the page element, which is the top level of the hierarchy. By clicking the tags in the path view, you can easily choose different elements in the hierarchy.

The toolbar at the top of the window contains several buttons that allow you to create the content of your component. WebObjects Builder also has menu commands corresponding to these buttons.

3. From the pop-up list at the left of the toolbar, choose .

   This pop-up list allows you to switch between graphical editing mode and source editing mode. When you choose source editing mode, the text of your HTML template (**Main.html**) appears. It is a skeleton at this point, since the page is empty. As you add elements graphically, their corresponding HTML tags appear in this file.


*Main.wo — C:\Users\nancy\MyProjects\GuestBook
File Edit Format Elements WebObjects Forms Window Services Help
None

```
<HTML>
<HEAD>
  <TITLE>Page Title</TITLE>
</HEAD>
<BODY>


</BODY>
</HTML>
```

The HTML source for your component.

Information about bindings is displayed here.

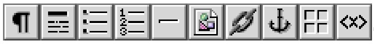
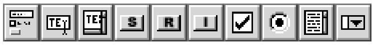
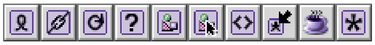
The bottom pane shows your declarations (**Main.wod**) file. Later, when you bind variables to your dynamic elements, this file stores the information. Normally, you do not type directly in this file. You can add elements using the toolbar in either source or graphical editing mode.

4. Switch back to graphical editing mode. For the rest of the tutorial, you’ll work in this mode.

Creating the Page's Content

A web page consists of *elements*. In addition to the standard static HTML elements found in all web pages, WebObjects allows you to create *dynamic elements*, whose look and behavior are determined at runtime.

To create elements, you use the toolbar buttons. There are three groups of buttons:

- **Structures** . Use these buttons to create paragraphs, lists, images, and other static HTML elements.
- **Dynamic form elements** . Use these buttons to create form elements in which users enter information. WebObjects gives your application access to the data entered by users by allowing you to associate, or *bind*, these elements to variables in your application.
- **Other WebObjects** . Use these buttons to create other dynamic elements, which you can bind to variables and methods in your program to control how they are displayed. Some of these (such as hyperlinks) have direct HTML equivalents. Others are *abstract dynamic elements*, such as repetitions and conditionals, which determine how many times an element is displayed or whether it is displayed at all.

Entering Static Text

The simplest way to add text to a page is to type it directly into the component's window. To demonstrate this, add a title for the GuestBook's page.

1. Type `My Guest Book` and press Shift-Enter (on the keyboard).

   The text is displayed at the insertion point, in this case at the beginning of the page.

2. Select the text you just typed.

3. Click the button in the toolbar. This converts the selected text to a heading element and displays it in a larger font.

4. From the pop-up list in the toolbar, choose center justification.

The toolbar also has buttons that allow you to apply text styles such as bold, underline, and italics.

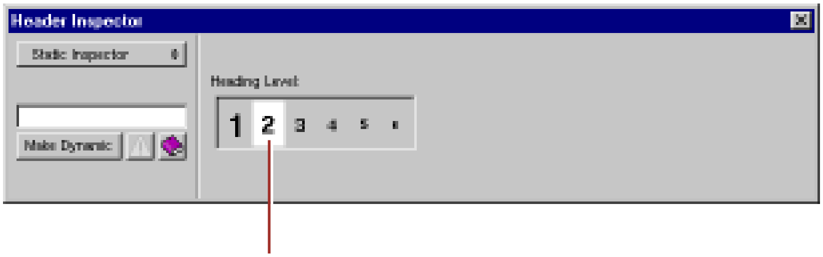
HTML provides several levels of headings. To change the level, you use the Inspector panel. You'll use this panel frequently throughout these tutorials.

Using the Inspector

You use the Inspector panel to set properties of the elements in your component. The Inspector's title and contents reflect the element you've selected in the component window.

1. Click ⓘ.

   A panel titled Heading Inspector appears. It allows you to set the level of the heading.



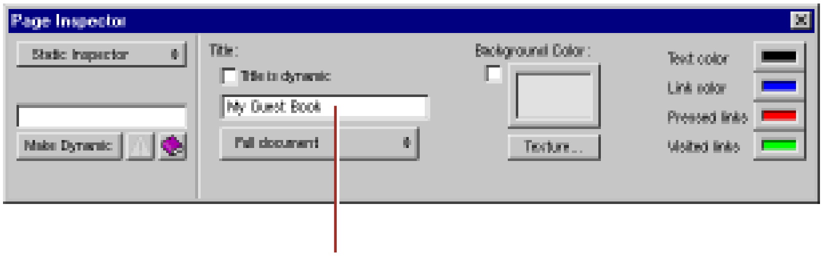
Click here to set the heading level.

2. Click "2".

   The text is now part of an <H2> tag, and is displayed in a smaller font.

3. Click <BODY> in the path view.

   Each element has its own Inspector that allows you to set properties appropriate for the element. The Page Inspector allows you to set properties such as the page's title and its text color.



Title: My Guest Book

Enter page's title here.

4. Type a title (such as "My Guest Book", or something else of your choosing) in the Title text field and press Enter. This is the title of the window that appears in your web browser when you run the application.

   Note: Be sure to press Enter after typing in the title; otherwise, it won't "stick."

5. Choose File ▸ Save to save the Main component.

   Although WebObjects Builder supports undo, it is always a good idea to save your work frequently.

Creating Form-Based Dynamic HTML Elements

In this section, you'll create a form with several elements to capture input from a guest. The Submit and Reset buttons you add to the form will apply to all other elements in the form. These elements look and act like HTML form elements but are actually dynamic WebObjects elements, which enable your code to receive and manipulate the data entered by the user. Refer to the screen shot that follows these steps to see how the window should look.

1. Place the cursor on the second line after the "My Guest Book" text.

2. Click ▤.

   WebObjects Builder adds a form element to your component. The triangle at the upper-left corner indicates that it is a dynamic form, as opposed to a static form.The gray border indicates the extent of the form. You can increase its size by adding elements inside it.

3. Type the text "`Name:` " and press Shift-Enter.

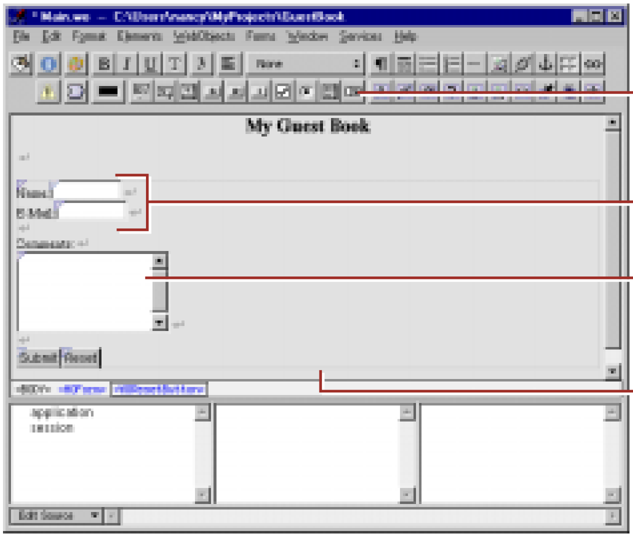
   This text replaces the word "Form" that was displayed by default.

4. Type "`E-mail: `" and press Shift-Enter twice.
5. Type "`Comments: `" followed by Shift-Enter.

   You have just entered three lines (and a blank line) of static text inside the form. Now you'll enter some dynamic elements to receive input from the user: two text fields and a multi-line text area.

6. Place the cursor to the right of the text "Name: ".
7. Click to create a dynamic text field element (WOTextField).
8. Repeat steps 6 and 7 for "E-mail: ".
9. Use the button to create a multi-line text area below the "Comments: " line.
10. Press Shift-Enter twice to create two blank lines.
11. Click S to create a Submit button, which is used to send the data in the form to the server.
12. Click R to create a Reset button, which is used to clear the data in the form.

    The window should now look like this:


* Main.wo – C:\Users\nancy\MyProjects\GuestBook
File Edit Format Elements WebObjects Forms Window Services Help
My Guest Book
Name:
E-Mail:
Comments:
Submit
Reset
application
session
Edit Source

Dynamic form elements buttons.

Dynamic text field elements.

Dynamic text area element.

Rectangle indicates extent of form.

Resizing the Form Elements

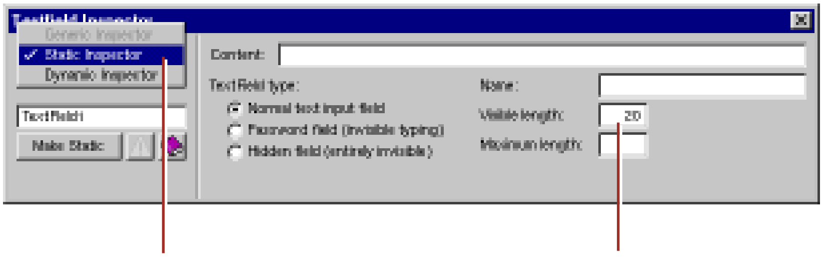
The text fields and text area are a bit small, so you'll resize them using the Inspector panel.

To inspect an element, you must first select it. Some elements (such as text fields and text areas) can be selected simply by clicking them; they appear shaded.

You select text elements as you would in most text-editing applications (by dragging, or by double-clicking words, or by triple-clicking lines); they appear highlighted when selected.

1. Select the Name text field.

2. In the Textfield Inspector, change the setting of the pop-up list at the upper left of the panel from Dynamic Inspector to Static Inspector.



TextField1

TextField type:
● Normal text input field
○ Password field (invisible typing)
○ Hidden field (entirely invisible)

Visible length: 20

Choose Static Inspector from this pop-up list.

Enter size here.

All WebObjects elements have a *dynamic inspector*, that is, one that allows you to set bindings (you'll work with bindings in the next section). In addition, many WebObjects elements (those with direct counterparts in static HTML) also have a *static inspector*. This inspector allows you to set the standard HTML attributes for that type of element.

In this panel, you can set various attributes of the static counterpart of a WOTextField, which is an HTML <INPUT TYPE=TEXT> element.

3. In the Visible length field, enter `20` to set the width of the text field to 20 characters.

4. Repeat steps 1 through 3 for the E-mail field.

5. Select the multi-line text area.

   In the Text Area Inspector, you can set various attributes corresponding to those of a <TEXTAREA> element.

6. Increase the size of the element by specifying the number of columns and number of rows to, say, 30 and 6.

7. Save the Main component.

Binding Elements

When a user enters information in form elements, your application needs a way of accessing that information. This is done by *binding* the form elements to variables in your application. When the user submits the form, WebObjects puts the data into the variables you've specified.

Your application typically processes the data and returns a new page (or the same page) displaying information that makes sense based on the user's input. The information displayed is usually represented by other dynamic elements that are bound to variables and methods in your code.

This process of receiving a request (triggered by actions such as submitting a form or clicking a hyperlink) and responding by returning a page is known as the *request-response loop*. This loop is at the heart of WebObjects programming.

In this tutorial, you'll have WebObjects return the same page, with the information you received from the user displayed in a slightly different format at the bottom.

Creating Variables

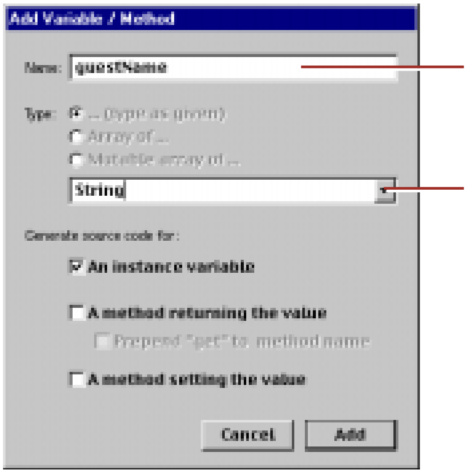
In this section, you'll declare individual variables in your code file (**Main.java**) to hold the name, e-mail address, and comments entered by a single guest. Later on, you'll structure this information differently in order to work with data from multiple users.

WebObjects Builder allows you to declare variables without having to edit your source file directly. At the bottom of the panel there is a pull-down menu titled Edit Source. It has five items:

- **Add Key** allows you to add a *key* to your source file. A key can be either an instance variable or a method that returns a value.
- **Add Action** allows you to add the template for an *action method*, which is a method that takes no parameters and returns a component (the next page to be displayed).
- **Delete Key** allows you to delete a key from your source file by deleting the instance variable or the method that returns a value.
- **Rename Key** allows you to rename a key in your source file by renaming the instance variable or the method that returns a value.
- **View Source File** opens the source file in a Project Builder window.

1. Choose Add Key from the pull-down menu.

   The Add Variable/Method panel opens.



Name: guestName

Type: ● ... (type as given)
○ Array of ...
○ Mutable array of ...

String

Generate source code for:

☑ An instance variable

□ A method returning the value

□ Prepend "get" to method name

□ A method setting the value

Type variable name here.

Choose variable's type
from this combo box.

2. Type `guestName` in the Name field.
3. To specify the variable's type, select String from the combo box (or you can type `String` directly in the box).
4. Click Add.

You have just created a variable called **guestName** of type String. It appears in the first column of the object browser. A declaration for **guestName** also appears in **Main.java**, which you'll edit later.

5. Create the variables **email** and **comments** in the same way (they are also of type String.)

Note: You may also add variables by editing the source file in Project Builder. You will need to use Project Builder to remove or modify a variable. Remember to save the file after editing in Project Builder to update WebObjects Builder.

Binding the Input Elements

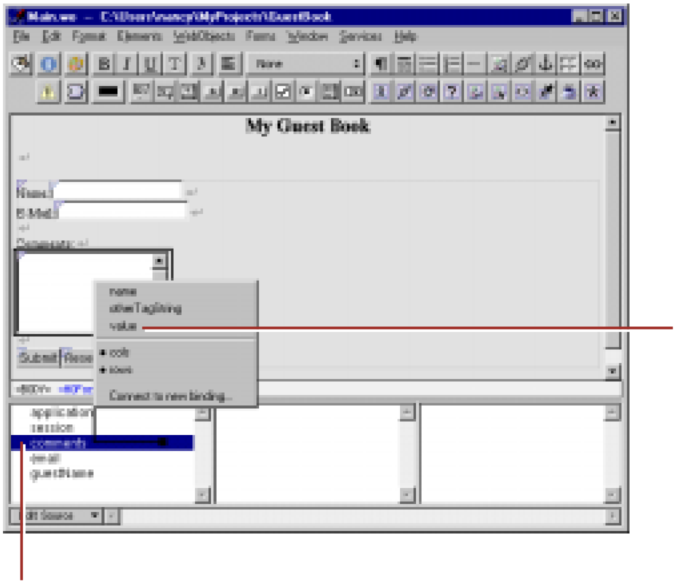
Each dynamic element contains several *attributes*. These attributes determine what happens when the element is displayed or when a form element is submitted. When you bind an element, you actually bind one or more of its attributes.

For example, a WOText element (which represents a multi-line text area) is defined as having two attributes:

- **value** specifies the string the user enters in the text area.
- **name** specifies a unique identifier for the text area.

In this tutorial, the only attribute you are concerned with is **value**, which represents the string entered by the user in the comments field. You'll bind this to the **comments** variable. You don't need to bind the **name** attribute in this application. In a later example, you'll bind more than one attribute of an element.

1. In the object browser, make a connection by pressing on the **comments** variable and holding down the mouse button while dragging to the Comments text area. Then release the mouse button.


Main.wo — C:\Users\nancy\MyProjects\GuestBook
File Edit Format Elements WebObjects Forms Window Services Help
None
My Guest Book
Name:
E-Mail:
Comments:
Submit
Reset
name
otherTagString
value
cols
rows
Connect to new binding...
application
session
comments
email
guestName
Edit Source

Click here to complete binding.

Press the mouse down on the variable name and drag to element to begin binding.

A menu appears, displaying the attributes for the text area.

2. Choose value.

   In the Dynamic Inspector, **comments** appears in the Binding column next to the **value** attribute of the text area, indicating that the binding has been made. Also, the text `comments` appears in the text field to show that it has been bound.

3. We'll bind the **guestName** variable using another technique. Select the Name WOTextField element. In the Inspector, select the Dynamic Inspector.

   The Inspector displays the **value** attribute in red, indicating that this attribute must be bound; otherwise, WebObjects displays an error message when you try to run your application.

4. In the Inspector, double-click in the Binding column next to **value.** Type `g` and press Enter. The Inspector fills in the rest of the "guestName" key for you.

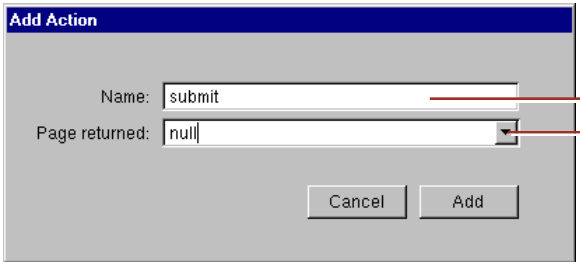
5. Bind the **email** variable to the corresponding text field using one of the methods above.

6. Save the Main component.

Implementing an Action Method

When the user clicks the Submit button, your application will respond by redisplaying the page with the submitted information shown at the bottom. To make this happen, you implement an *action method* and bind that method to the **action** attribute of the WOSubmitButton.

1. From the Edit Source menu at the bottom of the object browser, choose Add Action.



Name: submit

Page returned: null

Enter action name here.

Select response page name from pop-up menu (use null to return same page).

2. Enter `submit` as the name of your action method.

3. From the “Page returned” combo box, select **null**.

   The value returned by an action method represents the next page (component) to be displayed. When you return **null** (or **nil** if using WebScript), the current page is redrawn. In a later task, you’ll see how to return a new component.

4. Click Add.

   The **submit** action appears below a horizontal line in the first column of the object browser.

5. Make a connection from the **submit** action in the object browser to the Submit button (press the mouse button down on the action, drag to the button, and release the mouse button).

   A menu appears with the Submit button’s attributes.

6. Choose action.

   You just bound the **submit** method you created to the **action** attribute of the WOSubmitButton. You don't need to write any additional code, so your application is now ready to run. However, you may want to look at your source file.

7. From the pull-down list at the bottom of the window, choose View Source File.

   Project Builder becomes active and displays the code for your component (in **Main.java**). You'll notice that this file contains declarations for the variables you created earlier, as well as a declaration for the **submit** action method.

```
// Generated by the WebObjects Wizard ...

import com.apple.yellow.foundation.*;
import com.apple.yellow.webobjects.*;
import com.apple.yellow.eocontrol.*;
import com.apple.yellow.eoaccess.*;

public class Main extends WOComponent {
   protected String guestName;
   protected String email;
   protected String comments;

   public WOComponent submit() {
      return null;
   }
}
```

Creating the Application's Output

So far, you have a way for the guest to enter information and a way for the application to store that information. Now, the application needs to do something with the information.

For now, you'll have the application simply display the same information the user entered, in a slightly different format. This allows you to verify that you have correctly received the data. To do this, you'll add dynamic string elements (WOStrings) to the main page and bind them. In the next chapter, you'll use more complex forms of output.

1. In WebObjects Builder, place the cursor at the end of the document, making sure that it is *outside* the gray rectangle that represents the form, and press Shift-Enter.

2. Click — to create a horizontal rule (an <HR> element).

3. Press Shift-Enter to add a blank line.

4. Add a WOString element by clicking ▣.

   A WOString is a dynamic element whose value is determined at runtime. It is shown as a small rectangle surrounded by two icons. ▣□▣

5. In the object browser, make a connection from the **guestName** variable to the center rectangle of the WOString.

   Notice that the name **guestName** appears inside the WOString, and the attribute pop-up menu doesn't appear. The message "Connected guestName to value" appears in the upper-right corner of the panel.

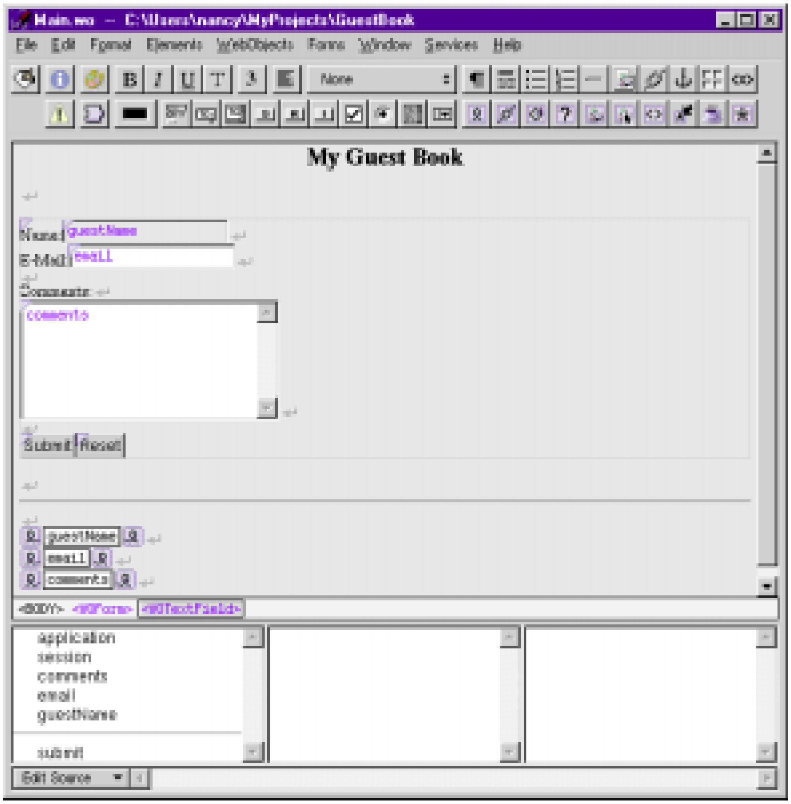
   WebObjects provides this shortcut for binding to the **value** attribute of WOStrings because it is the attribute you most often want to bind. The **value** attribute signifies the string that will be displayed when the page is drawn. If you want to bind a different attribute, you make a connection to the left or right icon, and the attribute pop-up menu appears as usual.

6. Click to the right of the WOString and press Shift-Enter.

7. Create two more WOStrings and bind them to **email** and **comments**, respectively.

   Note that it isn't necessary to resize the WOStrings as you did with the text fields. They expand at runtime to display the value of the variables to which they are bound.

8. Save your component. It should now look like this:



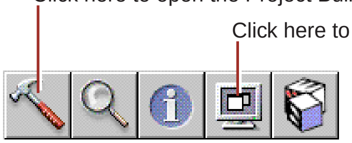
In summary, when the user clicks the Submit button, a new request-response cycle begins. WebObjects stores the data entered in the dynamic form elements in the variables they are bound to (**guestName** contains the value in the Name field, **email** contains the value in the E-mail field, and **comments** contains the value in the Comments field). It then triggers the action method bound to the **action** attribute of the WOSubmitButton. The action method returns a page (which, in this example, is the same page). When the page is redrawn, the dynamic strings at the bottom show the values entered by the user.

Now you are ready to test your application.

Building and Running Your Application

1. Make Project Builder active. A quick way to do this from WebObjects Builder is to choose View Source File from the pull-down list at the bottom of the window.

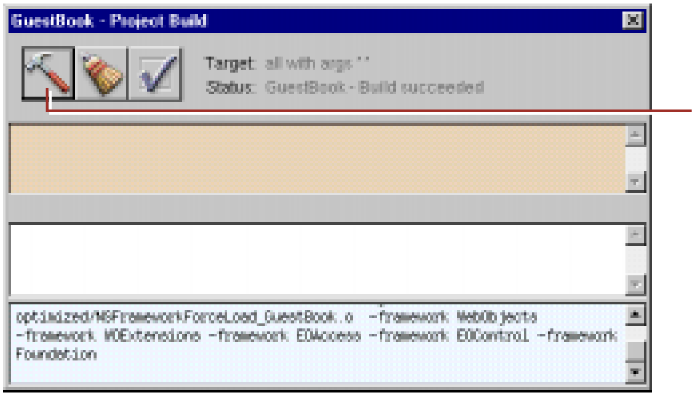
   To build and launch your application, you use buttons in Project Builder's toolbar.



Click here to open the Project Build panel.

Click here to open the Launch panel.

2. Click 🔨 in the toolbar to open the Project Build panel.
3. Click 🔨 in the Project Build panel.



GuestBook - Project Build

Target: all with args ''
Status: GuestBook - Build succeeded

```
optimized/NSFrameworkForceLoad_GuestBook.o  -framework WebObjects
-framework WOExtensions -framework EOAccess -framework EOControl -framework
Foundation
```

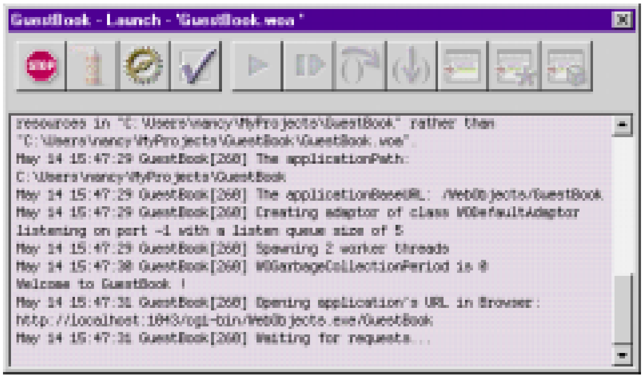
Click here to build your application.

   The Project Build panel displays the commands that are being executed to build your project. If all goes well, it displays the status message "Build succeeded."

4. Close the panel.
5. Click 🖥 in the toolbar to open the Launch panel.

6. Click 🖥 in the Launch panel to launch your application.

   The Launch panel displays a series of messages. If all goes well, you should see messages such as the following, which mean that your application is running successfully.



```
resources in "C:\Users\nancy\MyProjects\GuestBook" rather than
"C:\Users\nancy\MyProjects\GuestBook\GuestBook.woa".
May 14 15:47:29 GuestBook[260] The applicationPath:
C:\Users\nancy\MyProjects\GuestBook
May 14 15:47:29 GuestBook[260] The applicationBaseURL: /WebObjects/GuestBook
May 14 15:47:29 GuestBook[260] Creating adaptor of class WODefaultAdaptor
listening on port -1 with a listen queue size of 5
May 14 15:47:29 GuestBook[260] Spawning 2 worker threads
May 14 15:47:30 GuestBook[260] WOGarbageCollectionPeriod is 0
Welcome to GuestBook !
May 14 15:47:31 GuestBook[260] Opening application's URL in Browser:
http://localhost:1043/cgi-bin/WebObjects.exe/GuestBook
May 14 15:47:31 GuestBook[260] Waiting for requests...
```

   Your web browser (such as Netscape Navigator or Internet Explorer) should launch automatically and load the correct URL for your application.

7. Test your application by entering information and submitting the form.

   If all goes well, your page should look like the one shown at the beginning of this chapter (page 13).

Chapter 2

Enhancing Your Application

In the previous tutorial, you learned how to create a web component that has input and output elements and how to bind these elements to variables and methods in your code.

Now you'll add some additional features to your project that move it a bit more in the direction of being a real-world web application. The application will:

- Use a custom Java class to represent the data for a guest, rather than using three separate variables.
- Maintain a guest list, which keeps track of all guest data (whether entered by you or multiple users of your application), rather than just the current guest.
- Have a second component, so that the guest list is displayed in a new page rather than the same page. You'll use WebScript rather than Java to implement this component's behavior.
- Make use of additional interface elements (such as HTML tables).

Duplicating Your Project

Before proceeding, you'll create a new project by copying the old one and renaming it. This way, you can make changes and still retain your previous version.

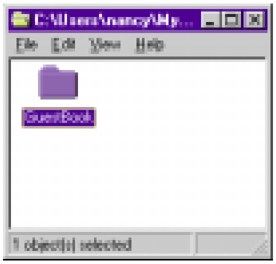
1. In WebObjects Builder, close the component window.

   If there are any unsaved files, you are prompted to save them.

2. In Project Builder, close GuestBook's project window.

   If there are any unsaved files, you are prompted to save them.

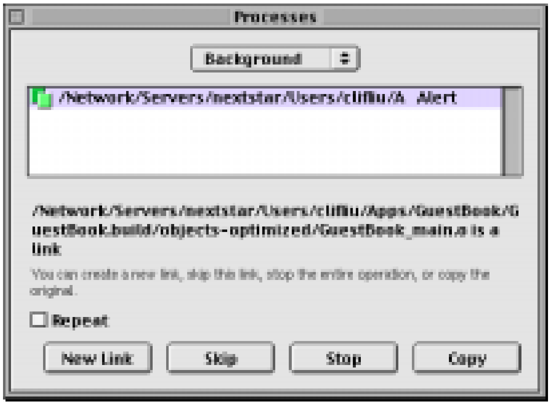
3. In your computer's file system, navigate to the directory where your project is located.



4. Duplicate the GuestBook folder.

   On Rhapsody, select the folder and press Command-D. On Windows NT, select the folder, choose Edit ► Copy, then Edit ► Paste.

   On Mac OS X Server, you may get an alert panel regarding a symbolic link; click Skip if it appears.



/Network/Servers/nextstar/Users/cliflu/Apps/GuestBook/GuestBook.build/objects-optimized/GuestBook_main.a is a link

You can create a new link, skip this link, stop the entire operation, or copy the original.

5. Open the new folder (Copy of GuestBook) and double-click the project file **PB.project**.

   Project Builder opens a new browser window for this project. (Alternatively, you could have opened the project from within Project Builder by choosing Project ► Open, then navigating to the project folder and selecting **PB.project**.)

6. Click in the toolbar to bring up the Project Build panel.

7. Click in the Project Build panel.

This command deletes all the files that were generated when you built the project previously.

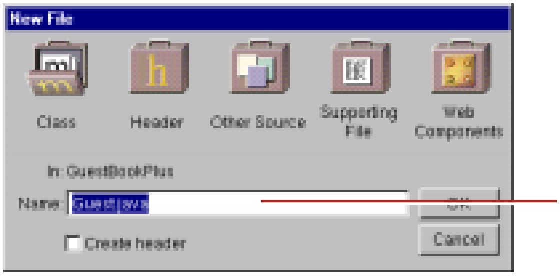
8. Click ⓘ to open the Project Inspector.
9. Choose Project Attributes from the pop-up list at the top of the window.
10. In the Name field, enter `GuestBookPlus` and press Enter.
11. Respond Yes to the prompt that asks if you want to rename the folder.

You now have a new project called GuestBookPlus.

Creating a Custom Guest Class

In the first chapter, you created individual variables to store a guest's name, e-mail address, and comments. When keeping track of multiple guests, it's more useful to encapsulate all the data for a guest as a single entity. You'll do this by creating a Java class that contains the data for a single guest.

1. In Project Builder's browser, select Classes in the first column.
2. Choose File ▶ New in Project.



In: GuestBookPlus

Name: Guest.java

□ Create header

Type name of class here.

3. Type `Guest.java` as the name of the file.
4. Click OK.

The newly created file contains a skeleton for a class called Guest.

5. Modify the code so it looks like this:

```
// Generated by the WebObjects Wizard ...

import com.apple.yellow.foundation.*;
import com.apple.yellow.eocontrol.*;
import com.apple.yellow.webobjects.*;

public class Guest extends EOCustomObject {
   protected String guestName;
   protected String email;
   protected String comments;

   Guest() {
      guestName = "";
      email = "";
      comments = "";
   }
}
```

A class stores information in its *instance variables* (also referred to as *data members*). Here you're declaring three instance variables for Guest: **guestName**, **email**, and **comments**. Note that these declarations are the same as those that appeared in the code for **Main.java** when you added the three variables using WebObjects Builder. In WebObjects, a component is also a class, specifically a subclass of the class WOComponent.

Java classes require a *constructor* to initialize an instance (or *object*) of a particular class whenever one is created. A constructor has the same name as the class and returns no value.

Whenever your application creates a new Guest class, its instance variables are initialized with empty strings, which is the default value if the user enters no data. (If you prefer, you can use different strings for these initial values.)

6. Save **Guest.java** by choosing Save from the File menu.

Saving the file lets WebObjects Builder know about your newly created Guest class.

Binding the Class's Instance Variables to the Form Elements

In the first chapter, you bound the input elements to variables in Main's code. Now you'll modify the bindings to use the class you just created.

1. Select Web Components in the first column of the browser.

2. Double-click **Main.wo** in the second column of the browser to open the component in WebObjects Builder.

3. Using the Add Key panel, add a variable called **currentGuest** to your component and specify its type as Guest. (Note that you can now choose Guest from the Type combo box.)

   An entry for **currentGuest** appears in the object browser. Notice the ">" symbol to the right of its name. This means that there is additional data to be displayed in the second column.

4. Select **currentGuest** in the object browser.

   The second column displays the three fields of **currentGuest**, as determined by the definition of its class, Guest.

5. Make a connection from **guestName** in the second column of the object browser (next to **currentGuest**) to the Name text field (press the mouse button down on the variable, drag to the element, and release the mouse button), and click **value** in the pop-up menu.

   This time, when the pop-up menu appears, there is a dot next to the **value** attribute because you bound it in the first tutorial.

6. Bind the other two input elements to **currentGuest.email** and **currentGuest.comments**.

Creating a Table to Display the Output

In the first chapter, you created three WOString elements to display the information the guest entered. In this tutorial, you'll create a different type of element, an HTML table, to display the information. In later tasks, you'll display data for multiple users in the table.

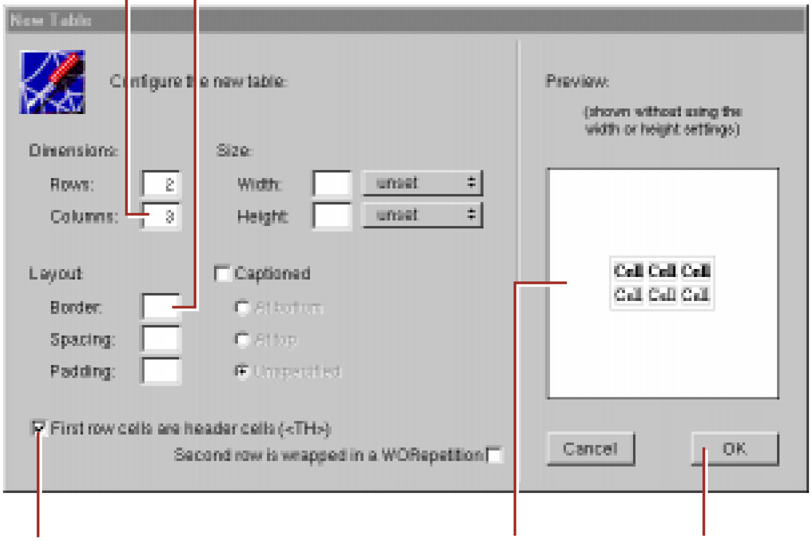
1. Delete the WOString elements below the horizontal line in the Main component, because you'll be replacing them with a table. Select the WOString elements and choose Cut from the Edit menu to delete them.

2. Click the ⊞ button.

   The New Table panel appears. On the right is the Preview pane, which displays what your table will look like.

Enter the number of columns here.

Enter the border width here.



New Table

Configure the new table:

Dimensions:

Rows: 2

Columns: 3

Size:

Width: unset

Height: unset

Layout

Border:

Spacing:

Padding:

□ Captioned

○ At bottom

○ At top

● Unspecified

☑ First row cells are header cells (<TH>)

Second row is wrapped in a WORepetition □

Preview:

(shown without using the width or height settings)

Cell	Cell	Cell
Cell	Cell	Cell

Cancel

OK

Click here to turn the first row into header cells.

Preview pane

Click here to put the table in your page.

3. In the Columns field, type 3 and press Enter. A third column appears in the preview pane.

4. In the Border field, enter 1. A border appears around the table in the preview pane.

5. Click "First row cells are header cells (<TH>)". The top row text becomes bold in the preview pane.

6. Click OK. The table appears in your page.

7. Select the upper-left cell of the table by clicking it.

8. Change the text in the cell to `Name`.

9. Open the Inspector if it is not already open.

   The Inspector presents a number of modifiable settings that apply to the table cell you've selected.

Enter table width here.

Select pixels here.

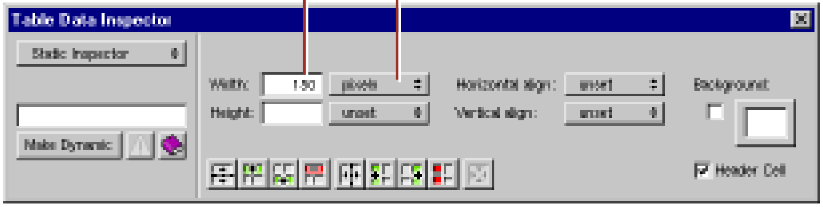


Table Data Inspector

Static Inspector

Width: 150 pixels

Height: unset

Horizontal align: unset

Vertical align: unset

Background ☐

☑ Header Cell

10. Select pixels from the Width pop-up list. Enter 150 in the Width field.

    The width of the column is set to 150 pixels.

11. Click in the component window, then press Tab.

    Pressing Tab when editing a table causes the contents of the next cell to the right to be selected (or the first cell of the next row if in the rightmost column). Pressing Shift-Tab moves in the opposite direction through the table.

12. Repeat steps 8 through 11 for the second and third cells of the top row. Label the middle column `E-mail` and set its width to 150 pixels. Label the third column `Comments` and leave its width unset. (The comments field takes up the remainder of the width of the table.)

    Note: It isn't necessary to adjust the height of the columns—if left unset, they'll expand at runtime to accommodate the size of the text being displayed.

Adding Dynamic Elements to Table Cells

Tables and cells are static HTML elements, so you can't bind them to variables or methods. To display dynamic information in cells, you add dynamic elements, such as WOStrings, to the cells.

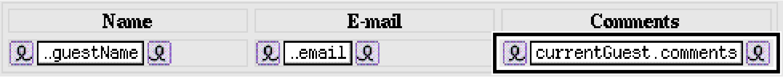
1. Click 👤 to add a WOString to the cell.
2. Press the Tab key.

   The contents of the next cell to the right are selected.
3. Repeat steps 1 and 2 for the other two cells in the second row.

Binding the Dynamic Elements in the Table

1. Make a connection from **currentGuest.guestName** in the object browser to the center of the WOString in the first column to bind its **value** attribute.
2. Similarly, bind **currentGuest.email** and **currentGuest.comments** to the WOStrings in the second and third columns.

   The table should now look like this:



Name	E-mail	Comments
..guestName	..email	currentGuest.comments

3. Save the Main component.

Creating the Guest Object

Earlier in this chapter, you created a Java class of type Guest and wrote a constructor for it. You also added a variable of that class, **currentGuest**, to the Main component. However, adding a variable to the component doesn't actually create a new Guest object; you need to create one explicitly at some point in your code.

You'll create the Guest object in the constructor method for your component. This method is called when the component is first created; that is, the first time the user accesses the component.

Note: In WebScript or Objective-C, you use a method called **init** for this purpose.

1. Choose View Source File from the pull-down list at the bottom of the window.

   Project Builder becomes active and displays the code for **Main.java**. Notice the following declaration that was added to your code when you added the **currentGuest** variable:

   ```
   protected Guest currentGuest;
   ```

2. Delete the declarations of **guestName**, **email**, and **comments** since you aren't using them anymore.

3. Add the constructor method inside the Main class definition:

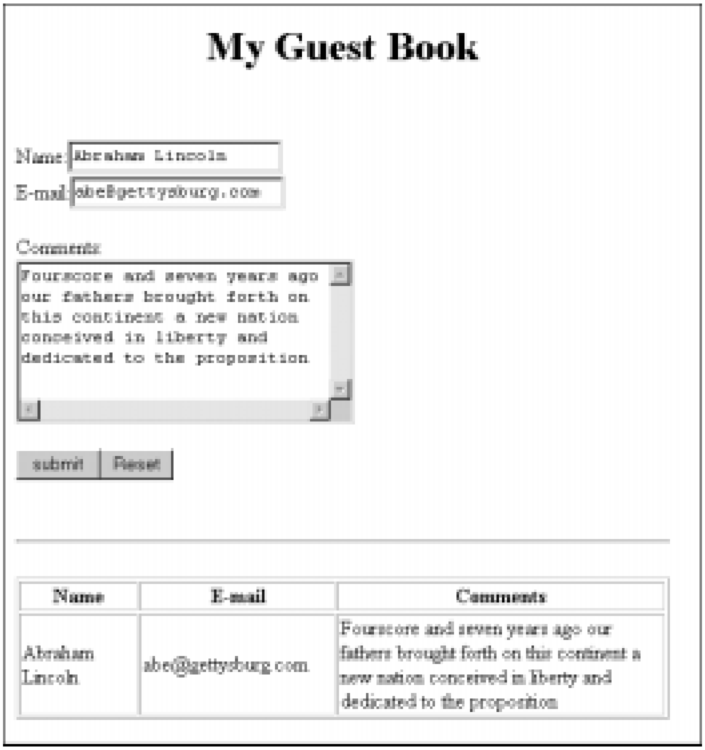
   ```
   public Main() {
       super();
       currentGuest = new Guest();
   }
   ```

   The first statement calls the constructor of Main's superclass (which is **com.apple.yellow.webobjects.WOComponent**). The second statement allocates a new empty Guest object and calls Guest's constructor to initialize its instance variables.

4. Save **Main.java**.

5. Build and run your application.

   The application should work similarly to that in the first chapter, except that the guest's data is displayed in a table at the bottom of the page instead of as plain text.



My Guest Book

Name: Abraham Lincoln

E-mail: abe@gettysburg.com

Comments:

```
Fourscore and seven years ago
our fathers brought forth on
this continent a new nation
conceived in liberty and
dedicated to the proposition
```

Name	E-mail	Comments
Abraham Lincoln	abe@gettysburg.com	Fourscore and seven years ago our fathers brought forth on this continent a new nation conceived in liberty and dedicated to the proposition

At this point, your application still handles information from a single guest only; in the next section, you'll modify the application so that it can keep track of multiple guests.

Keeping Track of Multiple Guests

You've been using the variable **currentGuest** in the Main component to hold the information entered by the user. You'll need another variable (an array) to store the list of all the guests who have registered.

Before doing this, it is important to understand the scope and life span of variables in WebObjects:

- *Component variables*, such as **currentGuest**, exist for the lifetime of the component. These variables are defined in the component (in this case, **Main.java**) and are accessible only by its methods. Each user that uses a component gets a separate instance of the variable.

- *Session variables* exist for the lifetime of one user's session and are accessible by all code in the session. They are defined in **Session.java**. An instance of each session variable is created for each user.
- *Application variables* live as long as the application does and are accessible by all code in the application. They are defined in **Application.java**. A single instance of an application variable is shared by all users of the application.

Creating a Guest List

To store the information from all guests that have accessed the application, you'll create an application variable called **allGuests**, which exists for the life of the application.

1. In Project Builder, select Classes in the first column of the browser. Then select **Application.java** from the second column.

   The application's code appears in the window. The following listing shows the code generated by the Wizard, along with code you will add.

```
// Generated by the WebObjects Wizard ...

import com.apple.yellow.foundation.*;
import com.apple.yellow.webobjects.*;

public class Application extends WOApplication {

    protected NSMutableArray allGuests;
    public Application() {
       super();
       allGuests = new NSMutableArray();
       System.out.println("Welcome to " + this.name() + " !");
       /* ** put your initialization code in here ** */
    }

    public void addGuest(Guest aGuest){
       allGuests.addObject(aGuest);
    }

    public void clearGuests(){
       allGuests.removeAllObjects();
    }
}
```

Note that there is one method already defined: **Application**, which is the constructor for the Application object. The first line calls the constructor for Application's superclass (which is the class WOApplication). The second line prints a message, which you see in the Launch panel when you launch your application.

2. After the call to **super**, enter this code:

```
allGuests = new NSMutableArray();
```

This statement initializes **allGuests** to be a new object of class NSMutableArray. This class is the Java equivalent of the Objective-C class NSMutableArray, which provides an interface that allows you to add, change, and delete objects from an array.

3. At the top of the Application class definition, enter this declaration:

```
protected NSMutableArray allGuests;
```

This declares **allGuests** to be of type NSMutableArray. Declaring it `protected` means that it is accessible only from this class or one of its subclasses. It is standard object-oriented practice for a class to prevent other classes from directly manipulating its instance variables. Instead, you provide *accessor methods* that other objects use to read or modify the instance variables.

4. Add the accessor methods **addGuest** and **clearGuests**, as shown in the listing.

The **addGuest** method adds an object of class Guest to the end of the **allGuests** array, using the NSMutableArray method **addObject**.

The **clearGuests** method removes all the objects from the array using the NSMutableArray method **removeAllObjects**.

5. Save **Application.java**.

Adding Guests to the Guest List

Now, when the user submits the form, you'll add the information to the **allGuests** array rather than displaying it directly.

1. Switch to the code for **Main.java**.

2. In the **submit** method, add the following code before the `return` statement:

   ```
   ((Application)application()).addGuest(currentGuest);
   currentGuest = new Guest();
   ```

   This code calls the application's **addGuest** method, which adds an object (in this case, **currentGuest**) to the end of the array. Then it creates a new Guest object to hold the next guest's data.

   Note: The **addGuest** method is defined in the class Application, which is a subclass of WOApplication. The component's **application** method (called in the above statement) returns an object of type WOApplication, so you must cast it to Application in order to access its **addGuest** method.

Your next step is to create a new component to display the list of guests that **allGuests** stores.

Adding a Second Component

In this section, you'll create a new component. Instead of Java, you'll implement its code using WebScript. This section demonstrates the quick turnaround between development cycles when using WebScript.

1. In Project Builder's browser, click Web Components in the first column.

2. Choose File ► New in Project.

   Note that the Web Components suitcase is selected.

3. Type `GuestList` as the name of the new component, then click OK.

   The WebObjects Component Wizard appears.

4. Choose None for Available Assistance and WebScript for Component Language.

5. Click Finish.

6. In the second column of the browser, double-click **GuestList.wo** to bring up the component window in WebObjects Builder.

7. Create a heading for this page, as you did for the Main component. Call it "Guest List" (or something else of your choosing), then press Enter twice.

8. Add a WOString below the heading. After the WOString, type the text " `guests have signed this guestbook`." Press Shift-Enter twice.

   You're going to bind this WOString so that it reflects the number of guests who have submitted this form.

9. In the object browser, click **application**.

   There is an entry in the second column for the **allGuests** application variable you created. This entry appears in the Main component as well, since application variables are accessible from anywhere in the code.

   If you click **allGuests**, you'll see in the third column an entry for **count**. This is a standard method that returns the number of objects in the array.

10. Make a connection from **count** to the center rectangle to bind it to the WOString's **value** attribute.

11. Save the GuestList component.

    You need to do one more thing so that the GuestList page now displays when the user submits the form.

12. Go back to Project Builder and view the source code for **Main.java**. Replace the `return` statement in the **submit** method with the following code:

    ```
    return pageWithName("GuestList");
    ```

    pageWithName is a standard WebObjects method (defined in the WOApplication class) that allows you to specify a new page to display.

At this point, the code for **Main.java** looks like this:

```
// Generated by the WebObjects Wizard ...



import com.apple.yellow.foundation.*;
import com.apple.yellow.webobjects.*;
import com.apple.yellow.eocontrol.*;
import com.apple.yellow.eoaccess.*;

public class Main extends WOComponent {
   protected Guest currentGuest;

   public Main() {
      super();
      currentGuest = new Guest();
   }

   public WOComponent submit() {
      ((Application)application()).addGuest(currentGuest);
      currentGuest = new Guest();
      return pageWithName("GuestList");
   }
}
```

13. Save **Main.java**.

14. Build and run your application.

    Each time you submit the form, the number of guests displayed in the WOString should increase.

    To return to the Main page, you'll have to use your browser's backtrack button. Later in the tutorial, you'll add a hyperlink to return to the Main page.

Using a Repetition

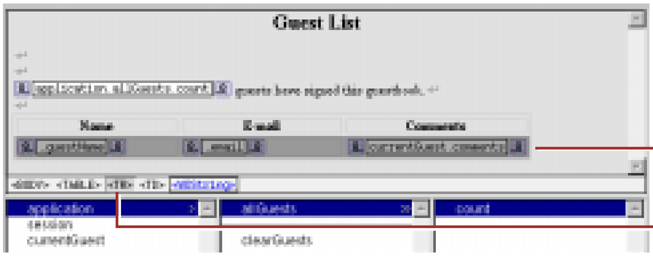
Now you'll create a table to display the entire list of guests in the GuestList component. To do so, you'll use a dynamic element called a *repetition* (an instance of the WORepetition class). Repetitions are one of the most important elements in WebObjects, since it is quite common for applications to display repeated data (often from databases) when the amount of data to be displayed isn't known until runtime. Typically, a repetition is used to generate items in a list or a browser, multiple rows in a table, or multiple tables.

A repetition can contain any other elements—either static HTML or dynamic WebObjects elements. In the GuestList component, you'll create a repetition that contains a table row.

You'll bind the **allGuests** array to the WORepetition's **list** attribute. This tells WebObjects to generate the elements in the repetition once for each item in the array. Each time WebObjects iterates through the array, it sets the repetition's **item** attribute to the current array object. You bind **item** to the variable **currentGuest** and use **currentGuest**'s fields to bind the elements inside the repetition (such as WOStrings). At runtime, the table will consist of one row (displaying name, e-mail address, and comments) for each guest.

1. In WebObjects Builder, make the Main component window active (double-click **Main.wo**).
2. Select the table at the bottom of the page by pressing outside it and dragging across it.
3. Choose Edit ► Copy.
4. Make the GuestList component active.
5. Place the cursor at the bottom of the page and choose Edit ► Paste.

   You have just copied the table from Main into GuestList. It has all the same properties, including the bindings. The WOStrings in the table are still bound to instance variables of **currentGuest**. Since **currentGuest** is a component variable defined in Main, it isn't accessible from GuestList. Therefore, you need to declare it here.
6. From the pull-down list at the bottom of the window, choose Add Key. Enter `currentGuest` as the name of the variable and `Guest` as its type, and click Add.
7. Click in one of the cells in the second row. Click <TR> in the path view to select the entire row.



Click in any cell in this row...

... and click here to select the entire row.

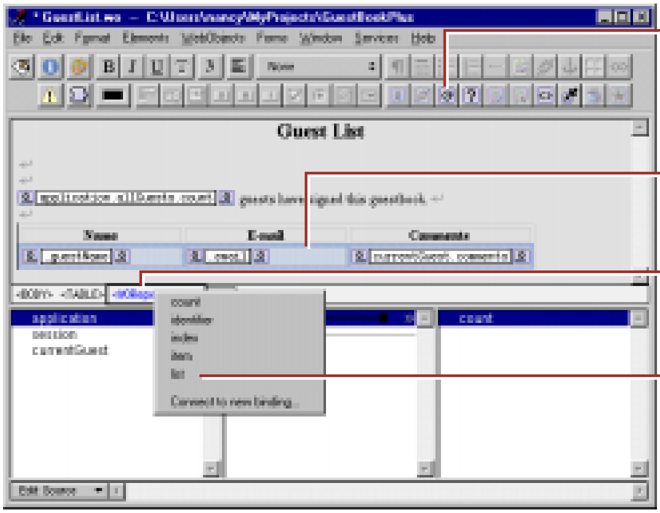
8. Click ↻ in the toolbar.

   When you wrap a repetition around a table row in this way, the WORepetition symbol ↻ doesn’t appear in the table. Instead, the row appears in blue. For additional examples of using repetitions, see Chapter 3, “Creating a WebObjects Database Application” (page 63).

9. In the object browser, select **application** in the first column.

10. In the second column, make a connection from **allGuests** to the <WORepetition> tag in the path view.

11. In the attribute menu that appears, click **list**. This binds **application.allGuests** to the WORepetition’s **list** attribute.



Select table row, then click here to create repetition around row.

Blue border and background means row is in a repetition.

Drag **allGuests** here to bind to repetition.

Click here to bind **allGuests** to the repetition’s list attribute.

12. Bind **currentGuest** to the repetition's **item** attribute.

    By using the name **currentGuest** for the **item** attribute, you are taking advantage of the fact that the strings in your table are already bound to the fields of **currentGuest**.

    You now have finished implementing the repetition. When the table is generated, it will have one row for each item in the **allGuests** array.

13. Save the GuestList component.

14. Delete the table from the Main component, since you no longer need it.

15. Test your application.

    Note: In this case, you don't have to rebuild or relaunch your application in order to test it. Building is required only when you have made changes to Java or Objective-C code. If you modify a component or WebScript code only, the changes take effect even if the application is already running.

16. Try entering data for multiple guests and verifying that each guest appears in the table.

Adding the Finishing Touches

There are a few additional things left to do to make your application a bit more user friendly:

- Add a button that, when clicked, clears the guest list.
- Add a hyperlink to the GuestList page that allows users to return to the Main page.

Clearing the Guest List

While developing your application, you may find it useful to be able to remove all guests from the list. (Typically, you wouldn't allow users to remove all guests from the list.)

1. In WebObjects Builder, make the GuestList component window active.

2. Choose Add Action from the pull-down list at the bottom of the window. In the panel, enter `clearGuestList` as the name of the action and set the page returned to `nil`. Click Add.
3. Choose View Source File from the pull-down list.

   Project Builder displays the code for **GuestList.wos**. **GuestList.wos** is your script file, the WebScript equivalent of **Main.java** in the Main component. You'll notice that there is a skeleton of the **clearGuestList** action method, using WebScript syntax, as well as the declaration for **currentGuest** that you created previously.
4. Enter the following code before the return statement in **clearGuestList**:

   ```
   [[self application] clearGuests];
   ```

   This code calls the application's **clearGuests** method, which removes all the Guest objects from the array.
5. Save **GuestList.wos** by choosing Save from the File menu.
6. Go back to WebObjects Builder.
7. Place the cursor below the table and press shift-Enter.
8. Click ▤ to add a WOForm element to contain the button you'll create in the next step.
9. Click S.

   This creates a submit button that the user will click to clear the guest list.
10. Using the Inspector, double-click in the binding column next to the **value** attribute and type `"Clear Guest List"`.

    This changes the title of the button. Note that the quotes are necessary to indicate that you're binding a string, not a variable.
11. Bind the **action** attribute to **clearGuestList**.

    When the user clicks the button, the **clearGuestList** action method is called, which causes the guest list to be cleared and the page to be redrawn.

Adding a Dynamic Hyperlink

Now you'll create a hyperlink that returns the user to the Main page.

1. Place the cursor below the submit button (outside the rectangle of its containing form).

2. Click 🔗.

3. Type `Return to Sign-in Page`, replacing the selected text.

4. Inspect the hyperlink.

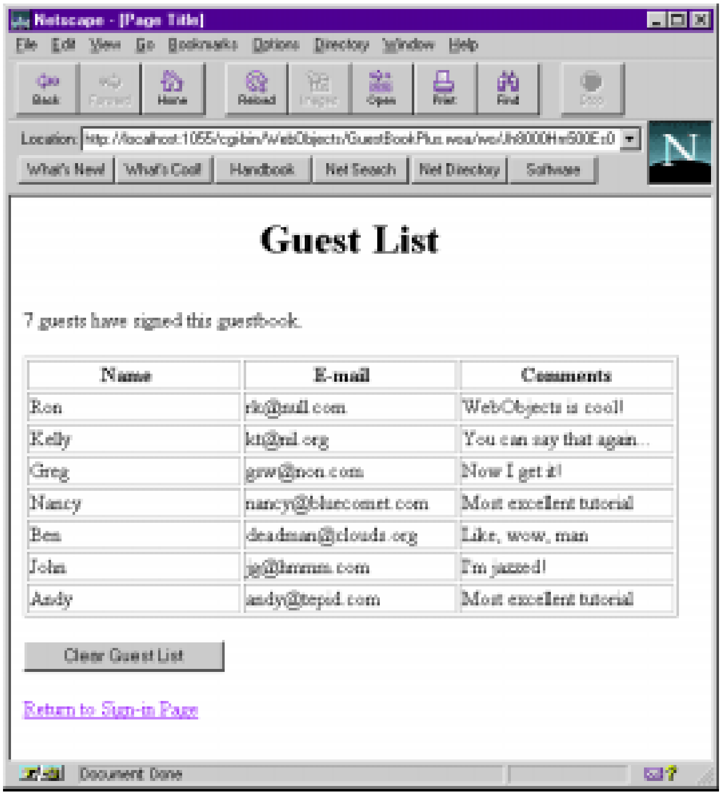
5. Select the **pageName** attribute, then double-click in the Binding column and type `"Main"` (including the quotes).

   Note: You must specifically type the quotation marks in "`Main`", because you are specifying a string representing the name of the page to be returned. If you left off the quotes, you would be specifying a variable or method called **Main**.

6. Save the GuestList component.

7. Test your application. Since you didn't modify any Java or Objective-C code, you don't have to rebuild and relaunch your application.

The GuestList page should now look like this:


Netscape - [Page Title]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
What's New! What's Cool! Handbook Net Search Net Directory Software
Document: Done

Location: http://localhost:1055/cgi-bin/WebObjects/GuestBookPlus.woa/wo/Jh8000Hn500E;0

Guest List

7 guests have signed this guestbook.

Name	E-mail	Comments
Ron	rk@null.com	WebObjects is cool!
Kelly	kt@rel.org	You can say that again...
Greg	grw@non.com	Now I get it!
Nancy	nancy@bluecomet.com	Most excellent tutorial
Ben	deadman@clouds.org	Like, wow, man
John	jg@hmmm.com	I'm jazzed!
Andy	andy@tepid.com	Most excellent tutorial

Clear Guest List

Return to Sign-in Page

Chapter 3

Creating a WebObjects Database Application

One of the most powerful features of WebObjects is its ability to provide access to databases. To do so, it uses a framework called the Enterprise Objects Framework. This chapter introduces you to the Enterprise Objects Framework by showing you how to create a simple database application. The steps you take in creating this application demonstrate the principles you'll use in every other application you develop with the WebObjects and Enterprise Objects frameworks.

The application you'll create in this tutorial is called Movies. It makes use of a sample database, the Movies database, that contains information about movies. In this tutorial we'll use the OpenBase Lite database that comes with WebObjects. If you wish to use another database, you need to set up the Movies database as described in the *Post-Installation Instructions*. Also, if you aren't familiar with Project Builder and WebObjects Builder, read the first tutorials in this book, "Creating a Simple WebObjects Application" (page 13) and "Enhancing Your Application" (page 39), which introduce basic concepts and procedures you should know before you go on.

In this tutorial, you will:

- Use the WebObjects Application Wizard to create a fully functional Main component that reads and writes from the Movies database.
- Create and configure *display groups* for interacting with a database in terms of objects.
- Create bindings between display groups and a user interface.
- Write code to manipulate display groups' selected objects.
- Set up display groups in a master-detail configuration.
- Use EOModeler to maintain a model file.
- Create custom enterprise object classes.

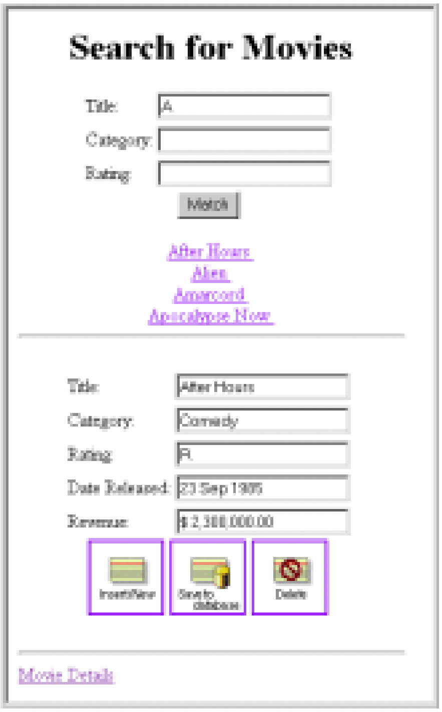
Along the way, you'll learn basic Enterprise Objects Framework concepts you can use to design your own database applications.

Note: You can also develop database applications using Direct to Web, a high-level framework based on WebObjects. Direct to Web instantly generates a generic database application and allows you to modify its user interface, which makes it a useful starting point for simple projects without very specific user interface requirements. See *WebObjects Tools and Techniques* and *Developing WebObjects Applications With Direct to Web* for more information.

The Movies Application

The Movies application has two pages, each of which allows you to access information from the database in different ways:

- *MovieSearch* (the main page) lets you search for movies that match user-specified criteria. For example, you can search for all comedies starting with the letter "A". Once you find the movie you're looking for, you can make changes to its data or delete it. You can also use this page to insert new movies into the database.
- *MovieDetails* displays the actors who star in a selected movie and the roles those actors play. You can add new roles, change the name of a role, and assign a different actor to a role.



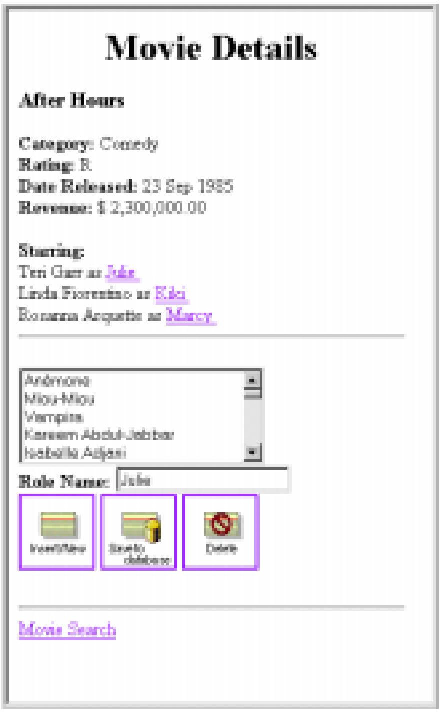
Search for Movies

Title: A
Category:
Rating:

After Hours
Alien
Amarcord
Apocalypse Now

Title: After Hours
Category: Comedy
Rating: R
Date Released: 23 Sep 1985
Revenue: $ 2,300,000.00

Movie Details



Movie Details

After Hours

Category: Comedy
Rating: R
Date Released: 23 Sep 1985
Revenue: $ 2,300,000.00

Starring:
Teri Garr as Julie
Linda Fiorentino as Kiki
Rosanna Arquette as Marcy

Anémona
Miou-Miou
Vampira
Kareem Abdul-Jabbar
Isabelle Adjani

Role Name: Julie

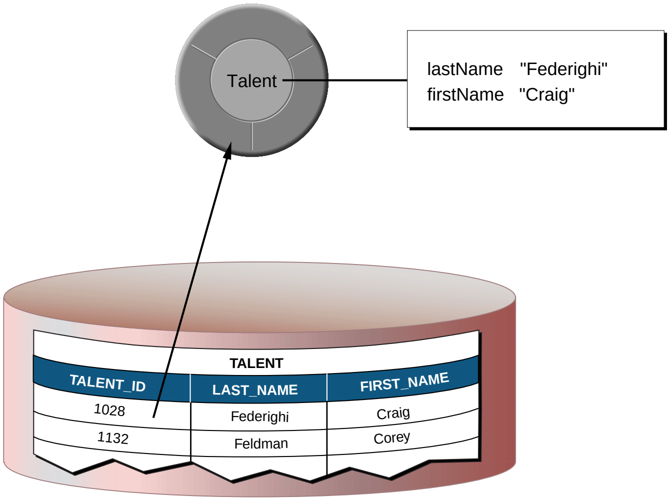
Movie Search

Enterprise Objects and the Movies Database

Enterprise Objects Framework manages the interaction between the database and objects in the Movies application. Its primary responsibility is to fetch data from relational databases into *enterprise objects*. An enterprise object, like any other object, couples data with methods for operating on that data. In addition, an enterprise object has properties that map to stored data. Enterprise object classes typically correspond to database tables. An enterprise object instance corresponds to a single row or record in a database table.

The Movies application centers around three kinds of enterprise objects: Movies, MovieRoles, and Talents. A movie has many roles, and talents (or actors) play those roles.

The Movie, MovieRole, and Talent enterprise objects in the Movies application correspond to tables in a relational database. For example, the Talent enterprise object corresponds to the TALENT table in the database, which has LAST_NAME and FIRST_NAME columns. The Talent enterprise object class in turn has **lastName** and **firstName** instance variables. In an application, Talent objects are instantiated using the data from a corresponding database row, as shown in the following figure:



TALENT

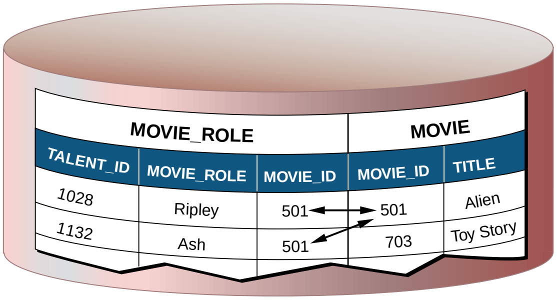
TALENT_ID	LAST_NAME	FIRST_NAME
1028	Federighi	Craig
1132	Feldman	Corey

Enterprise Objects and Relationships

Relational databases model not just individual entities, but entities' relationships to one another. For example, a movie has zero, one, or more roles. This is modeled in the database by both the MOVIE table and MOVIE_ROLE table having a MOVIE_ID column. In the MOVIE table, MOVIE_ID is a *primary key*, while in MOVIE_ROLE it's a *foreign key*.

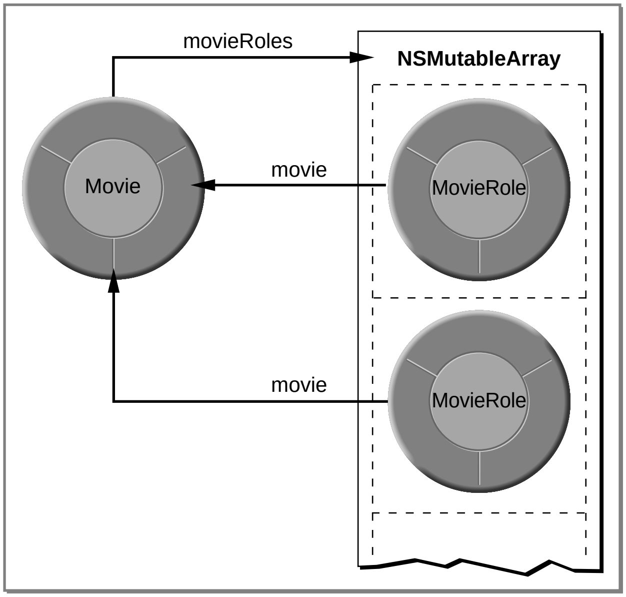
A *primary key* is a column or combination of columns whose values are guaranteed to uniquely identify each row in that table. For example, each row in the MOVIE table has a different value in the MOVIE_ID column, which uniquely identifies that row. Two movies could have the same name but still be distinguished from each other by their MOVIE_IDs.

A *foreign key* matches the value of a primary key in another table. The purpose of a foreign key is to identify a relationship from a source table to a destination table. In the following diagram, notice that the value in the MOVIE_ID column for both MOVIE_ROLE rows is 501. This matches the value in the MOVIE_ID column of the "Alien" MOVIE row. In other words, "Ripley" and "Ash" are both roles in the movie "Alien."



MOVIE_ROLE			MOVIE	
TALENT_ID	MOVIE_ROLE	MOVIE_ID	MOVIE_ID	TITLE
1028	Ripley	501	501	Alien
1132	Ash	501	703	Toy Story

Suppose you fetch a Movie object. Enterprise Objects Framework takes the value for the movie's MOVIE_ID attribute and looks up movie roles with the corresponding MOVIE_ID foreign key. The framework then assembles a network of enterprise objects that connects a Movie object with its MovieRole objects. As shown below, a Movie object has an array of its MovieRoles, and the MovieRoles each have a Movie.



Designing the Main Page

Every WebObjects application has at least one component—usually named Main—that represents the first page the application displays. In Movies, the Main component represents the MovieSearch page.

To design the Main component, you'll use the WebObjects Application Wizard. The wizard performs all the setup that's necessary to fetch database records and display them in a web page. Specifying different wizard options yields different pages: The MovieSearch page is an example of one of the many different layouts you can generate with the wizard.

Starting the WebObjects Application Wizard

1. In Project Builder, choose Project ▶ New.

2. In the New Project panel, select Webobjectsapplication from the Project Type pop-up list.

3. Click Browse.

4. Navigate to a directory where you want to create your new project.

5. Type `Movies` in the “File name” field.

6. Click Save.

7. In the New Project panel, click OK.

   This starts the WebObjects Application Wizard.

Note: On Mac OS X Server, the file browser and the Project Type pop-up list both appear on the New Project panel.

8. Choose Wizard under Available Assistance.

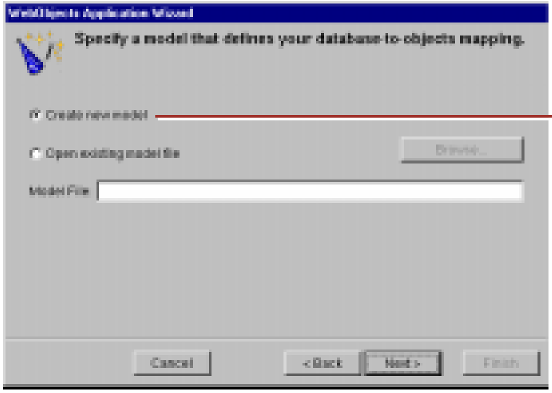
   With this option, the wizard guides you through the creation of a Main component for your application. When you finish, you can immediately build and run your application without performing any additional steps and without adding any code.

9. Choose Java as the primary language.

10. Click Next.

Specifying a Model File

A *model* associates database columns with instance variables of objects. It also specifies relationships between objects in terms of database join criteria. You typically create model files using the EOModeler application, but the wizard can create a first cut at a model as a starting point. Later on, you’ll use EOModeler to modify the model created by the wizard.



WebObjects Application Wizard

Specify a model that defines your database-to-objects mapping.

● Create new model

○ Open existing model file

Model File:

Select this option.

1. Choose “Create new model.”

2. Click Next.

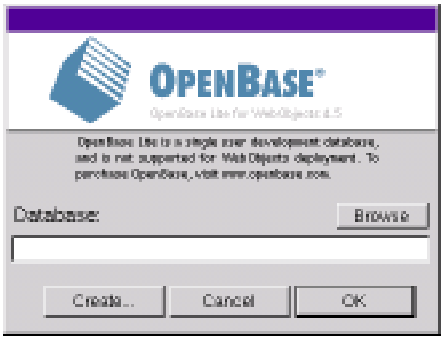
Choosing an Adaptor

An *adaptor* is a mechanism that connects your application to a particular database server. For each type of server you use, you need a separate adaptor. WebObjects provides adaptors for OpenBase Lite, Informix, Oracle, and Sybase servers. If you’re working on a Windows platform, WebObjects also provides an ODBC adaptor for use with ODBC-compliant database sources.

1. In the wizard panel, choose the adaptor for your database.

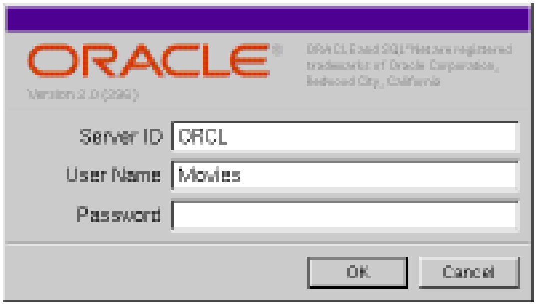
2. Click Next.

   A login panel for the selected adaptor opens. Different databases require different login information, so each database’s login panel looks different. Shown below are the login panels for the OpenBase Lite, Oracle, and ODBC adaptors.



OpenBase Lite is a single user development database, and is not supported for WebObjects deployment. To purchase OpenBase, visit www.openbase.com.

Database:

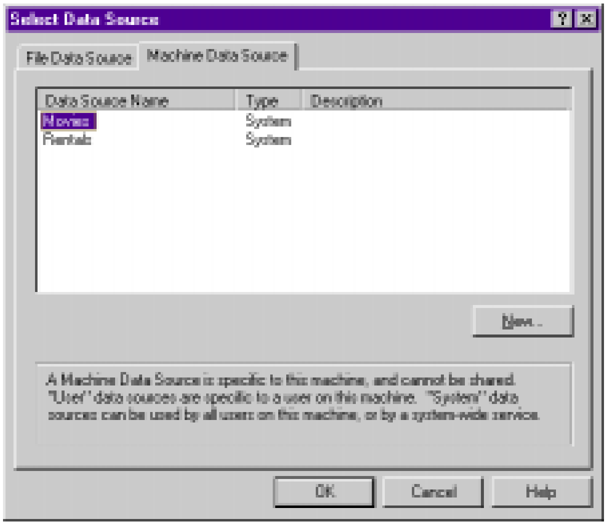


ORACLE and SQL*Net are registered trademarks of Oracle Corporation, Redwood City, California

Server ID: ORCL

User Name: Movies

Password:



Data Source Name	Type	Description
Movies	System	
Rentals	System	

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

3. Complete the login panel.

   If you are using the preinstalled OpenBase Lite database, click “Browse”, browse to the **/Local/Library/Databases/Movies.db** file

(**\Apple\Local\Library\Databases\Movies.db** in Windows NT) and click Open. The filename now appears in the Database field.

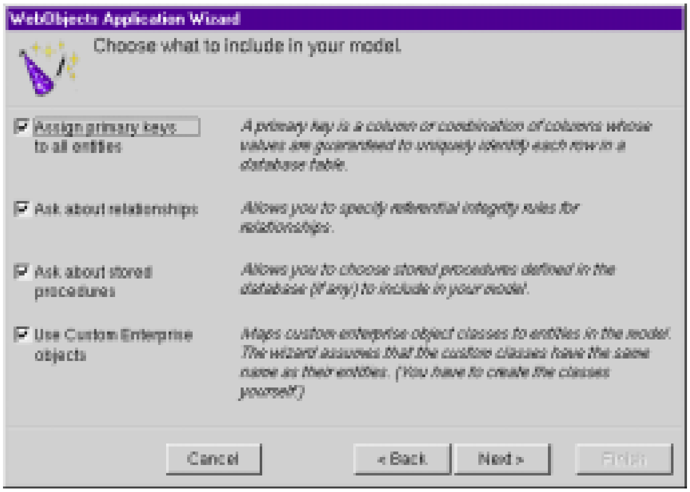
If you are not using OpenBase Lite, specify the connection information you provided when you created and populated the Movies database. *Post-Installation Instructions* provides more information.

4. Click OK.

When you use the wizard to create a model file, the wizard uses the adaptor you specify to connect to your database. With the information you specified in the adaptor's login panel, the adaptor logs in, reads the database's schema information, and creates a model. The wizard uses your answers to the questions in the next several pages to configure that model.

Choosing What to Include in Your Model

In this next wizard page, you can specify the degree to which the wizard configures your model.



WebObjects Application Wizard

Choose what to include in your model.

☑ Assign primary keys to all entities — *A primary key is a column or combination of columns whose values are guaranteed to uniquely identify each row in a database table.*

☑ Ask about relationships — *Allows you to specify referential integrity rules for relationships.*

☑ Ask about stored procedures — *Allows you to choose stored procedures defined in the database (if any) to include in your model.*

☑ Use Custom Enterprise objects — *Maps custom enterprise object classes to entities in the model. The wizard assumes that the custom classes have the same name as their entities. (You have to create the classes yourself.)*

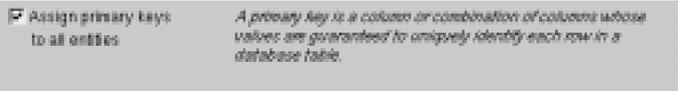
The basic model the wizard creates contains *entities*, *attributes*, and *relationships*. An *entity* is the part of the database-to-object mapping that associates a database table with an enterprise object class. For example, the Movie entity maps rows from the MOVIE table to Movie objects. Similarly, an *attribute* associates a database column with an instance variable. For example, the **title** attribute in the Movie entity maps the TITLE column of the MOVIE table to the **title** instance variable of Movie objects.

A *relationship* is a link between two entities that's based on attributes of the entities. For example, the Movie entity has a relationship to the MovieRole entity based on the entities' **movieId** attributes (although the attributes in this example have the same name in both entities, they don't have to). This relationship makes it possible to find all of a Movie's MovieRoles.

How complete the basic model is depends on how completely the schema information is inside your database server. For example, the wizard includes relationships in your model only if the server's schema information specifies foreign key definitions.

Using the options in this page, you can supplement the basic model with additional information. (Note that the wizard doesn't modify the underlying database.)

1. Check the "Assign primary keys to all entities" box.



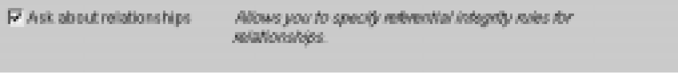
☑ Assign primary keys to all entities

A primary key is a column or combination of columns whose values are guaranteed to uniquely identify each row in a database table.

Enterprise Objects Framework uses primary keys to uniquely identify enterprise objects and to map them to the appropriate database row. Therefore, you must assign a primary key to each entity you use in your application. The wizard automatically assigns primary keys to the model if it finds primary key information in the database's schema information.

Checking this box causes the wizard to prompt you to choose primary keys that aren't defined in the database's schema information. If your database doesn't define them, the wizard later prompts you to choose primary keys.

2. Check the "Ask about relationships" box.

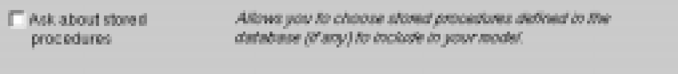


☑ Ask about relationships

Allows you to specify referential integrity rules for relationships.

If there are foreign key definitions in the database's schema information, the wizard includes the corresponding relationships in the basic model. However, a definition in the schema information doesn't provide enough information for the wizard to set all of a relationship's options. Checking this box causes the wizard to prompt you to provide the additional information it needs to complete the relationship configurations.

3. Uncheck the "Ask about stored procedures" box.

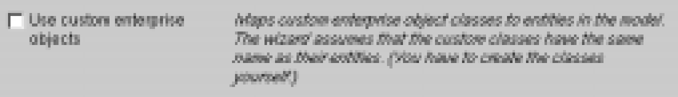


□ Ask about stored procedures

Allows you to choose stored procedures defined in the database (if any) to include in your model.

Checking this box causes the wizard to read stored procedures from the database's schema information, display them, and allow you to choose which to include in your model. Because the Movies application doesn't require the use of any stored procedures, don't check this box.

4. Uncheck the "Use custom enterprise objects" box.



□ Use custom enterprise objects

Maps custom enterprise object classes to entities in the model. The wizard assumes that the custom classes have the same name as their entities. (You have to create the classes yourself.)

An entity maps a table to enterprise objects by storing the name of a database table (MOVIE, for example) and the name of the corresponding enterprise object class (a Java class, Movie, for example). When deciding what class to map a table to, you have two choices: EOGenericRecord or a custom class. EOGenericRecord is a class whose instances store key-value pairs that correspond to an entity's properties and the data associated with each property.

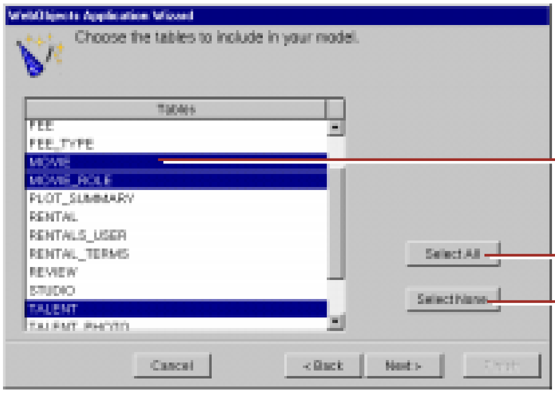
If you don't check the "Use custom enterprise objects" box, the wizard maps all your database tables to EOGenericRecord. If you do check this box, the wizard maps all your database tables to custom classes. The wizard assumes that each entity is to be represented by a custom class with the same name. For example, a table named MOVIE has an entity named Movie, whose corresponding custom class is also named Movie.

Use a custom enterprise object class only when you need to add business logic; otherwise use EOGenericRecord. The Movies application uses EOGenericRecord for the Movie entity and custom classes for the Talent and MovieRole entities. Later on, you'll use EOModeler to specify the custom classes.

5. Click Next.

Choosing the Tables to Include

1. In the wizard panel, select MOVIE, MOVIE_ROLE, and TALENT in the Tables browser.



WebObjects Application Wizard

Choose the tables to include in your model.

Command-click (Control-click in Windows NT) to select more than one table.

Click to select all the tables.

Click to deselect all the tables.

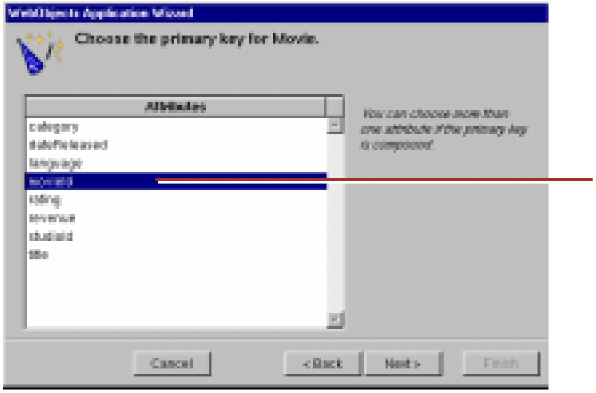
The wizard creates entities only for the tables you select. Since the Movies application doesn't interact with any of the other tables (for example, DIRECTOR, PLOT_SUMMARY, STUDIO, and TALENT_PHOTO), you don't need to include them in the model.

2. Click Next.

Specifying Primary Keys

If you are using a database that stores primary key information in its database server's schema information, the wizard skips this step. The wizard has already successfully read primary key information from the schema information and assigned primary keys to your model.

However, if primary key information isn't specified in your database server's schema information (as with Microsoft Access), the wizard now asks you to specify a primary key for each entity.



WebObjects Application Wizard

Choose the primary key for Movie.

Attributes

category
dateReleased
language
movieId
rating
revenue
studioId
title

You can choose more than one attribute if the primary key is compound.

Command-click (Control-click in Windows NT) to select more than one attribute.

1. Select **movieId** as the primary key for the Movie entity.

2. Click Next.

3. Select both **movieId** and **talentId** as the primary key for the MovieRole entity.

   MovieRole's primary key is *compound*; that is, it's composed of more than one attribute. Use a compound primary key when any single attribute isn't sufficient to uniquely identify a row. For MovieRole, the combination of the **movieId** and **talentId** attributes is guaranteed to uniquely identify a row.

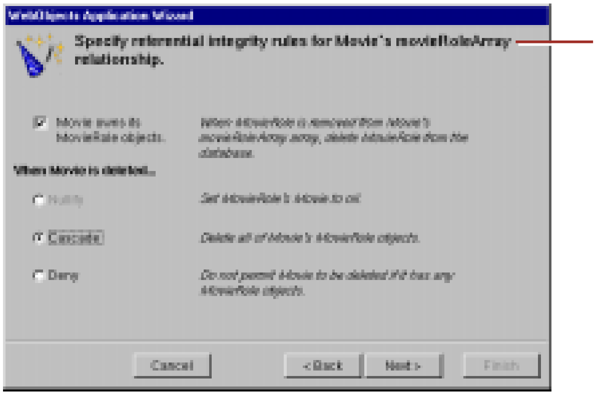
4. Click Next.

5. Select **talentId** as the primary key for the Talent entity.

6. Click Next.

Specifying Referential Integrity Rules

If you're using a database that stores foreign key definitions in its database server's schema information, the wizard reads them and creates corresponding relationships in your model. For example, Movie has a to-many relationship to MovieRole (that is, a Movie has an array of MovieRoles), and Talent has a to-many relationship to MovieRole. The wizard now asks you to provide additional information about the relationships so it can further configure them.



Specify referential integrity rules for Movie's movieRoleArray relationship.

☑ Movie owns its MovieRole objects. — When MovieRole is removed from Movie's movieRoleArray array, delete MovieRole from the database.

When Movie is deleted...

○ Nullify — Set MovieRole's Movie to nil.

● Cascade — Delete all of Movie's MovieRole objects.

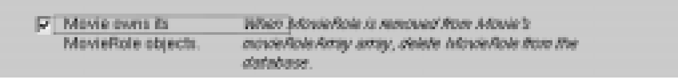
○ Deny — Do not permit Movie to be deleted if it has any MovieRole objects.

In this example, the relationship name is **movieRoleArray**, but the name is dependent on the adaptor you're using.

If foreign key definitions aren't specified in your database server's schema information (as with Microsoft Access), the wizard hasn't created any relationships at all, and it skips this step. You'll add relationships to your model using EOModeler later in this tutorial.

In the first relationship configuration page, the wizard asks you about Movie's relationship to MovieRole. The name of the relationship is dependent on the adaptor you're using.

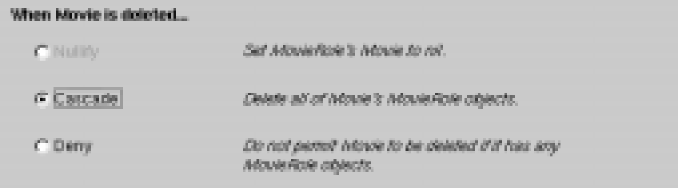
1. Check the "Movie owns its MovieRole objects" box.



☑ Movie owns its MovieRole objects. — When MovieRole is removed from Movie's movieRoleArray array, delete MovieRole from the database.

This option specifies that a MovieRole can't exist without its Movie. Consequently, when a MovieRole is removed from its Movie's array of MovieRoles, the MovieRole is deleted—deleted in memory and deleted in the database.

2. Choose Cascade.



When Movie is deleted...

○ Nullify *Set MovieRole's Movie to nil.*

● Cascade *Delete all of Movie's MovieRole objects.*

○ Deny *Do not permit Movie to be deleted if it has any MovieRole objects.*

This option specifies what to do when the source object (the Movie) is deleted. The cascade delete rule specifies that when a source object is deleted, the source's destination objects should also be deleted—again, deleted in memory and correspondingly in the database.

3. Click Next.

   Now the wizard asks you about Talent's relationship to MovieRole.

4. Check the "Talent owns its MovieRole objects" box.

5. Choose Deny.

   The deny delete rule specifies that if the relationship source (a Talent) has any destination objects (MovieRoles), then the source object can't be deleted.

6. Click Next.

You're done with the model configuration part of the wizard. The rest of the wizard pages are to help you configure your application's user interface.

Choosing an Entity

In this page, the wizard asks you to choose the entity around which the Main component will be centered. Your Main component centers around the Movie entity.

1. Select the Movie entity.

2. Click Next.

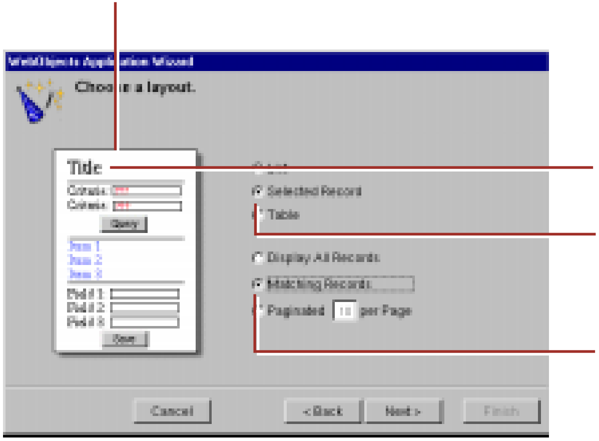
Choosing a Layout

The wizard provides several page layout options for formatting objects fetched from the database.

1. Choose Selected Record.

2. Choose Matching Records.

A preview of the page is an approximation of what the finished page will look like given your choices. (The number of fields and items isn't necessarily the exact number that will be in the finished page.)



WebObjects Application Wizard

Choose a layout.

○ List
● Selected Record
○ Table

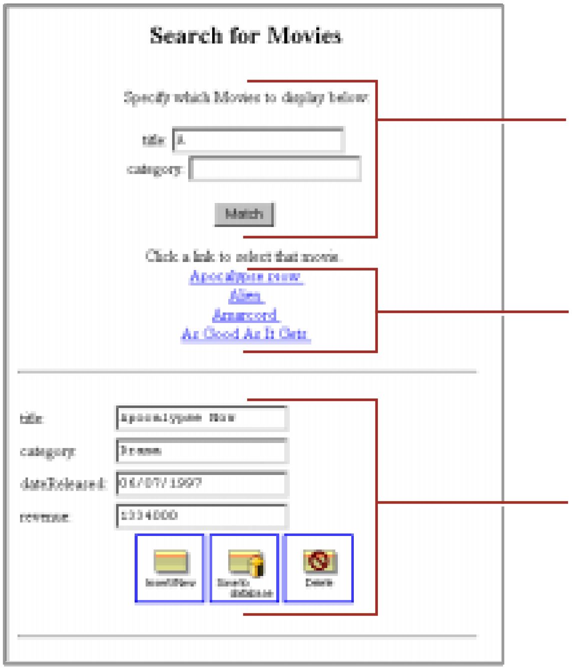
○ Display All Records
● Matching Records
○ Paginated 10 per Page

The wizard generates a title based on your chosen entity.

Specifies that the page will have a way to select a record from a list and a way to edit that selected record.

Specifies that the page will have a way to specify search criteria.

Based on your specifications, the wizard shows you a preview of the page it will generate. To see how the wizard's preview corresponds with the actual page the wizard will create, the finished page is shown below.



Search for Movies

Specify which Movies to display below:

title: A

category:

Click a link to select that movie.

Apocalypse now

Alien

Amarcord

As Good As It Gets

title: Apocalypse Now

category: Drama

dateReleased: 06/07/1997

revenue: 1334000

This is the **query part**, where users type search criteria. Clicking Match fetches movies that meet the criteria and displays their titles in the repetition part in the middle of the page.

This is the **repetition part**. Clicking a movie title selects the movie and displays it in the editing part at the bottom of the page.

This is the **editing part**, which displays information about the selected movie. You can use this part to edit or delete the selected movie, to create a new movie, and to save your work.

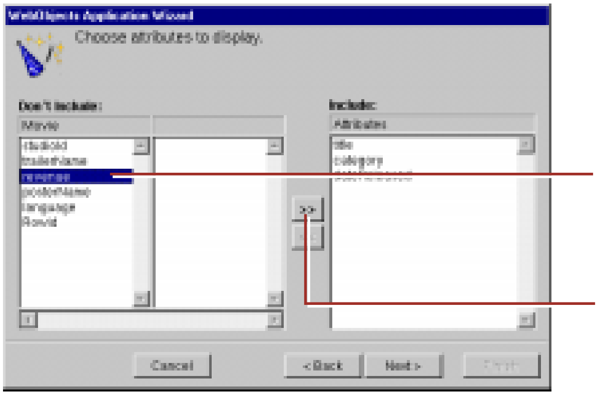
There are three parts to this page: the query part (at the top of the page), which contains fields in which users provide search criteria; the repetition part (in the middle of the page), which contains a list of matching records fetched from the database; and the editing part (at the bottom of the page), which allows you to make changes to the selected record.

3. In the wizard panel, click Next.

Choosing Attributes to Display

The next step is to choose which of the Movie entity's attributes to display in the editing part at the bottom of the page.

1. Move attributes from the Don't Include list to the Include list.



Double-click an attribute to move it to the Include list.
OR
Select an attribute...

...and click here to move it.

The order in which you add the attributes determines the order in which they appear on the page, so add them in the following order: **title**, **category**, **dateReleased**, and **revenue**.

Don't add any of the remaining attributes (for example, **trailerName**, **studioId**, **posterName**, and **language**)—they aren't used in this tutorial.

2. Click Next.

Choosing an Attribute to Display as a Hyperlink

You now need to specify the attribute used in the repetition part of the page to identify each record. This attribute will be displayed as a hyperlink. Clicking the hyperlink displays the corresponding record in the detail part of the page.

1. Add the **title** attribute to the Include browser.

2. Click Next.

Choosing Attributes to Query On

Specify the attributes to display in the query part of the page. The wizard creates search criteria fields for each of the attributes you choose.

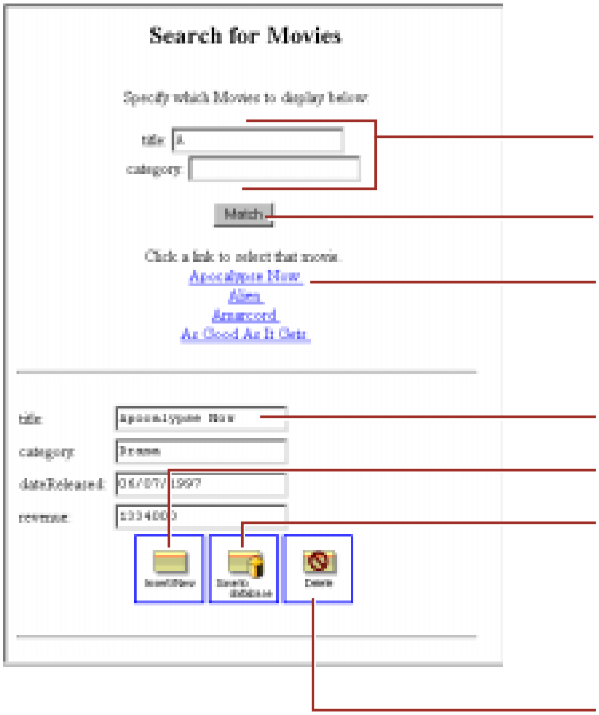
1. Add the **title** and **category** attributes to the Include browser.

2. Click Finish.

When the wizard finishes, your new project is displayed in Project Builder. The wizard has produced all the files and resources for a fully functional, one-page application. All you need to do before running your Movies application is build it.

Running Movies

Build and run the application as you did in the previous tutorials.



Search for Movies

Specify which Movies to display below:

title: A

category:

Match

Click a link to select that movie.

Apocalypse Now

Alien

Amarcord

As Good As It Gets

title: Apocalypse Now

category: Drama

dateReleased: 06/07/1997

revenue: 1334000

Type matching criteria. A database string matches if it begins with the string in the text field. For example, strings match “The” if they start with the string “The”.

Click here to fetch and return matching movies.

Click a movie to select it and display its information below.

Use these text fields to edit the information about a movie.

Click here to create a new, empty movie.

Click here to save your work in the database (add the new movies you inserted, remove the movies you deleted, and save changes you made to existing movies).

Click here to delete the selected movie.

Experiment with the application by entering different search criteria and inserting, updating, and deleting movies. For example:

1. Search for all movies beginning with the letter “A”.

   Type `A` in the title field, and click Match.

2. Change the attributes of one of the movies and click “Save to database.”

When you're done, perform another search to verify that your change was saved.

3. Try entering dates with different formats.

   In the same movie, try changing the date released using different formats (for example, "6/7/97," "June 7, 1997," and "today"). Save each time after changing the date.

4. Create a new movie.

   Click Insert/New to create a new, empty movie. Fill out all the fields, and click "Save to database." Search for your movie to verify that it was saved successfully.

5. Delete your movie.

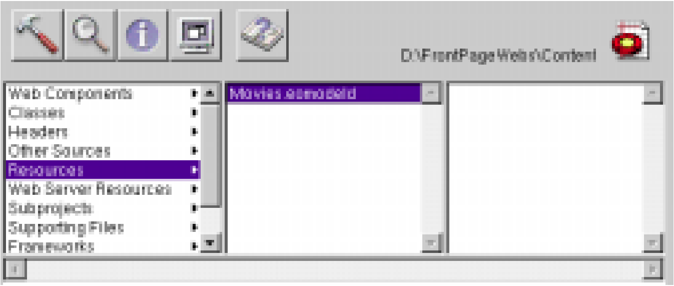
   With your movie selected, click Delete and then click "Save to database." When you're done, search for the movie again to verify that it's been deleted.

Examining Your Project

Whenever you create a new project, Project Builder populates the project with ready-made files and directories. What it includes depends on the choices you make in the wizard, so this project has a set of files different from those of the GuestBook project.

Like GuestBook, the Movies project contains a Main component (**Main.wo**). It also includes some files that the GuestBook doesn't have: classes (**Application.java**, **Session.java**, **DirectAction.java**, and **Main.java**), a model file, and images used by the Main component.

In Project Builder, navigate to the Movie project's Resources category. This is where the model, named **Movies.eomodeld**, is located. Later in this tutorial you'll use EOModeler to open the model and enhance it.



Navigate to the Web Server Resources category. This is where your project's images are located: **DBWizardUpdate.gif**, **DBWizardDelete.gif**, and **DBWizardInsert.gif**, for the "Save to database," "Delete", and "Insert/New" buttons, respectively.

The biggest difference between the GuestBook and Movies projects is their Main components. Whereas the Main component you created for the GuestBook project was empty, the Main component for the Movies project contains a fully functional user interface. Also, the **Main.java** class already contains code that supplies the component with behavior. In the next sections, you'll examine the Movies project's **Main.wo** component and its **Main.java** class.

Examining the Variables

1. Double-click **Main.wo** in Project Builder's Web Components category to open the Main component in WebObjects Builder.

   There are four variables in the object browser: the **application** and **session** variables that are available in all components and two others, **movie** and **movieDisplayGroup**.

   The **movie** variable is an enterprise object that represents a row fetched from the MOVIE table. **movieDisplayGroup** is a *display group*—an object that interacts with a database, indirectly through classes in the Enterprise Objects Framework. Display groups are used to fetch, insert, update, and delete enterprise objects that are associated with a single entity. The entity of **movieDisplayGroup** is Movie, which you specified in the wizard's "Choose an entity" page.

2. In Project Builder, look at the class file **Main.java** to see how **movie** is declared.

   The **movie** declaration (shown below) declares **movie** to be an EOEnterpriseObject—a Java interface that describes the general behavior that all enterprise objects must have.

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

   At runtime, **movie** is a EOGenericRecord object. Recall that EOGenericRecord is used to represent enterprise objects unless you specify a custom class. Since you didn't check the "Use custom enterprise objects" box in the wizard's "Choose what to include in your model" page, your application uses EOGenericRecord for all its entities.

   The comment (`/** @TypeInfo Movie */`) is used by WebObjects Builder to identify **movie**'s entity (Movie). Knowing the entity allows WebObjects Builder to display **movie**'s attributes (**category**, **dateReleased**, and so on). You can see **movie**'s attributes if you select the **movie** variable in the WebObjects Builder's object browser.

3. In Project Builder, examine the **movieDisplayGroup** declaration in **Main.java**.

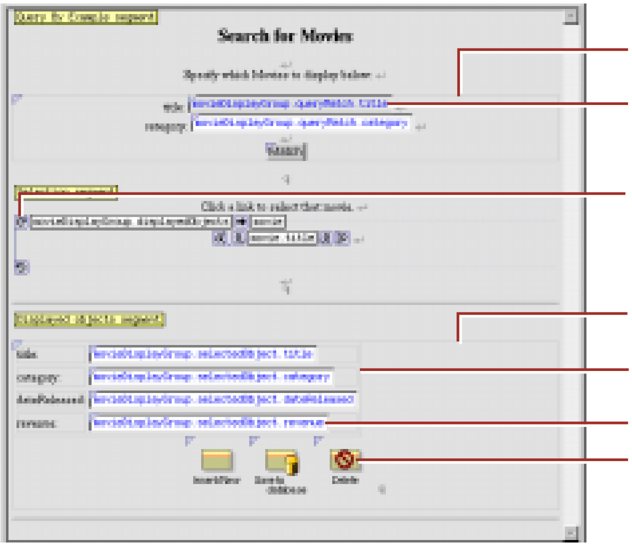
   The declaration (shown below) declares **movieDisplayGroup** to be a WODisplayGroup.

   ```
   protected WODisplayGroup movieDisplayGroup;
   ```

   Also note the comment explaining how **movieDisplayGroup** is initialized. The **Main.java** class doesn't have any code to create and initialize the display group. Instead, it's instantiated from an archive file, **Main.woo**, that's stored in the **Main.wo** component. You shouldn't edit **woo** files by hand; they're maintained by WebObjects Builder. The **woo** file archiving mechanism is described in more detail later in "Specifying a Sort Order" (page 91).

Examining the Bindings

Now examine the bindings of your Main component in WebObjects Builder.


Query By Example segment
Search for Movies
Specify which Movies to display below:
title:
movieDisplayGroup.queryMatch.title
category:
movieDisplayGroup.queryMatch.category
Match
Click a link to select that movie.
movieDisplayGroup.displayedObjects
movie
movie.title
title
category
dateReleased
revenue
movieDisplayGroup.selectedObject.title
movieDisplayGroup.selectedObject.category
movieDisplayGroup.selectedObject.dateReleased
movieDisplayGroup.selectedObject.revenue
Insert/New
Save to Database
Delete

Everything within this gray box is in a form.

This text field appears shaded because it's selected.

This is a repetition.

This gray box identifies another form.

This is a table with four rows and two columns.

This text field is in a table cell.

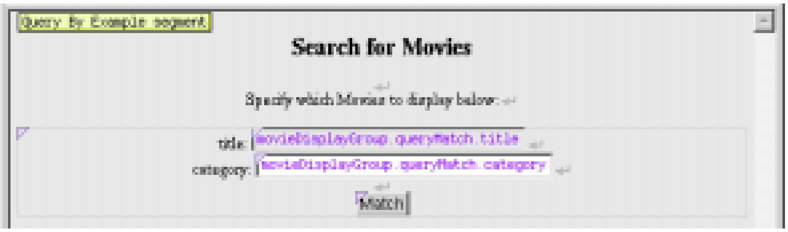
This is a WOImageButton.

Remember that you can use WebObjects Builder's Inspector to see the bindings for an element's attributes. Simply select the element to inspect, and click the ⓘ button to open the Inspector.

Bindings in the Query Part

In the query part of the component, **movieDisplayGroup.queryMatch.title** is bound to the title text field. There are similar bindings to the category text fields. The **queryMatch** bindings allow users to specify search criteria to use when **movieDisplayGroup** next fetches movies. The Match button is bound to **movieDisplayGroup.qualifyDataSource**, which actually performs the fetch.

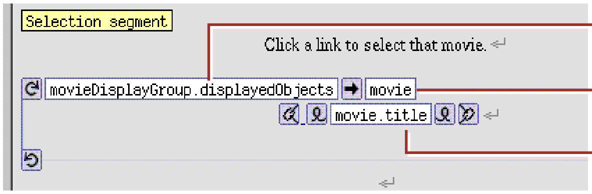
For example, to display all comedies, a user types "Comedy" in the Category text field, and clicks the Match button. **movieDisplayGroup** then refetches, selecting only movies whose **category** values are set to Comedy.


Query By Example segment
Search for Movies
Specify which Movies to display below:
title:
movieDisplayGroup.queryMatch.title
category:
movieDisplayGroup.queryMatch.category
Match

Bindings in the Repetition Part

In the repetition part of the component where matching movies are listed, **movieDisplayGroup.displayedObjects** is bound to a repetition. More specifically, **displayedObjects** is bound to the repetition's **list** attribute, providing an array of movies for the repetition to iterate over.

The **movie** variable is bound to the repetition's **item** attribute to hold each movie in turn, and **movie.title** is bound to the string element inside the repetition. These bindings produce a list of movie titles.



Click a link to select that movie.

movieDisplayGroup.displayedObjects → movie

movie.title

Displays the binding for the repetition's list attribute.

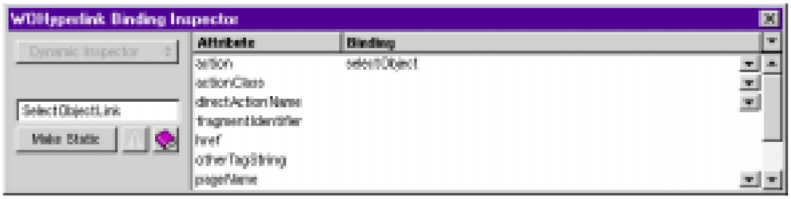
Displays the binding for the repetition's item attribute.

Displays the binding for the string's value attribute.

The repetition's string element is enclosed in a hyperlink. By clicking a movie title, the user *selects* the corresponding movie.

1. Inspect the hyperlink.

   Its **action** attribute is bound to the action method **selectObject**.



SelectObjectLink

Attribute	Binding
action	selectObject
actionClass	
directActionName	
fragmentIdentifier	
href	
otherTagString	
pageName	

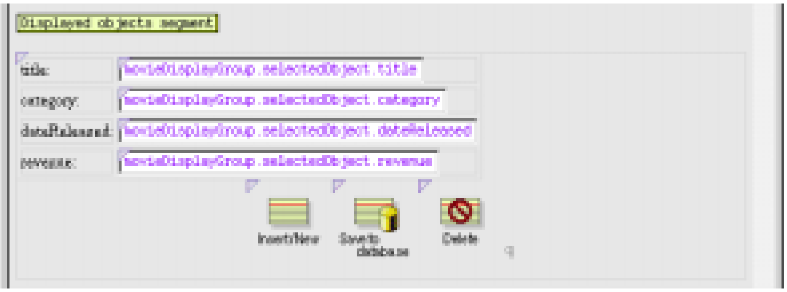
2. Look in the **Main.java** class to see how the **selectObject** method is implemented.

   The method (shown below) simply sets the selected object of **movieDisplayGroup** to the movie the user clicked.

```
public void selectObject() {
movieDisplayGroup.selectObject(movie);
}
```

Bindings in the Editing Part

The text fields in the editing part are all bound to attributes of the **movieDisplayGroup** object's **selectedObject**—the movie on which the user clicked. Typing new values into these fields updates the Movie enterprise object. To actually save the updated values to the database, the user must click the "Save to database" button.



Displayed objects segment

title: movieDisplayGroup.selectedObject.title

category: movieDisplayGroup.selectedObject.category

dateReleased: movieDisplayGroup.selectedObject.dateReleased

revenue: movieDisplayGroup.selectedObject.revenue

1. Inspect the middle image button.

   Its **action** attribute is bound to the action method **saveChanges**.

2. Look in the **Main.java** class to see how **saveChanges** is implemented.

   The method (shown below with comments omitted) simply saves any changes that have been made to **movieDisplayGroup**'s objects to the database.

```
public void saveChanges() throws Exception {
    try {
       this.session().defaultEditingContext().saveChanges();
    }
    catch (Exception exception) {
       System.err.println("Cannot save changes ");
       throw exception;
    }
}
```

   this.session() returns a Session object that represents a connection to the application by a single user. A Session object provides access to an EOEditingContext object. The expression

```
this.session().defaultEditingContext().saveChanges();
```

sends a **saveChanges** message to the Session's **defaultEditingContext**. This default EOEditingContext object manages graphs of objects fetched from the database, and all changes to the database are saved through it. For more information, see the EOEditingContext class specification in the *Enterprise Objects Framework Reference*.

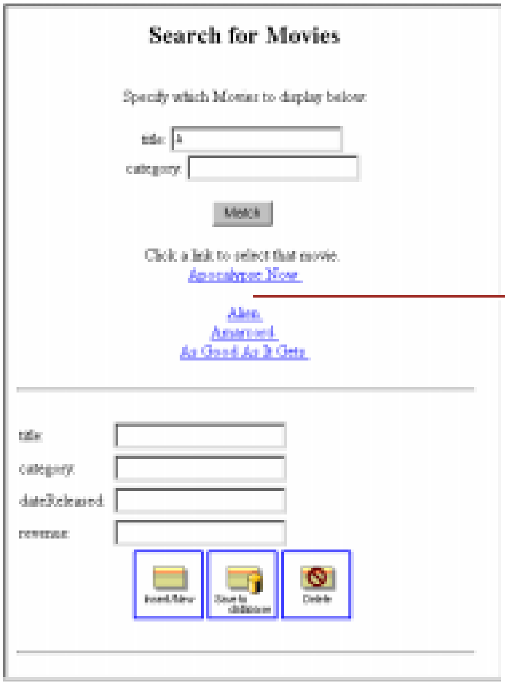
An EOEditingContext's **saveChanges** method uses other Enterprise Objects Framework objects to analyze its network of enterprise objects (Movie objects referenced by the application) for changes and then to perform a set of corresponding operations in the database. If an error occurs during this process, **saveChanges** throws an exception. The **Main.java saveChanges** method simply raises the exception, having the effect of returning a diagnostic page. You could return an error page that explains the reason for the save failure instead, but the application in this tutorial uses the default behavior.

3. Inspect the first and third image buttons to see what their **action** attributes are bound to.

   They are bound to the **movieDisplayGroup.insert** and **movieDisplayGroup.delete** methods respectively. The WODisplayGroup **insert** method creates a new enterprise object, then inserts it into the display group's list of objects just past the current selection. The WODisplayGroup **delete** method deletes the display group's selected object. These changes happen only in memory—not in the database. To actually insert a new row in the database (or delete a row), the user must click the "Save to database" button, invoking **saveChanges** on the session's EOEditingContext. The editing context analyzes the enterprise objects in memory; determines if any objects have been added, updated, or deleted; and then executes database operations to sync the database with the application.

Refining Main.wo

You may have noticed that your application doesn't list fetched movies in any particular order. Also, when you insert a new movie, it appears in the list of movies as a blank line.



Search for Movies

Specify which Movies to display below:

title: A

category:

Click a link to select that movie.

Apocalypse Now

Alien

Amarcord

As Good As It Gets

title:

category:

dateReleased:

revenue:

A newly inserted movie doesn't have a title set, so it appears in the list of movies as a blank line.

In this section you'll tidy up the user interface to fix these things and a few others. Specifically, you'll:

- Configure **movieDisplayGroup** to sort the movies it displays.
- Assign default values to new Movie objects.
- Change the way that dates and numbers are displayed.

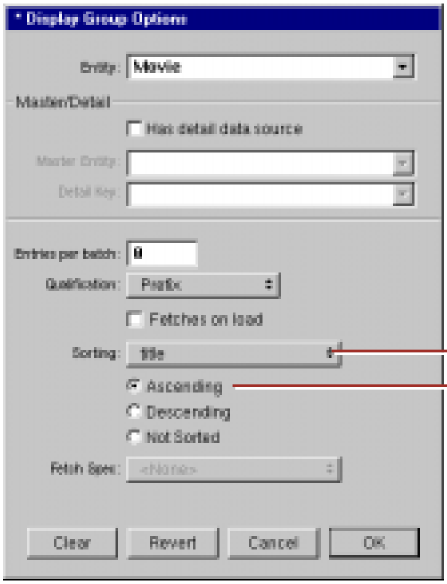
You can also put the query part of the page in a table and capitalize **Main.wo**'s text field labels—for example, use "Title" instead of "title" and "Date Released" instead of "dateReleased."

Specifying a Sort Order

You can change your application to sort movies alphabetically without writing any code. Display groups manage sorting behavior, and WebObjects Builder provides a Display Group Options panel for configuring this and other characteristics of display groups.

1. Double-click the **movieDisplayGroup** variable in the object browser.

The Display Group Options panel opens for configuring **movieDisplayGroup**.



Entity: Movie

□ Has detail data source

Master Entity:

Detail Key:

Entries per batch: 0

Qualification: Prefix

□ Fetches on load

Sorting: title

● Ascending

○ Descending

○ Not Sorted

Fetch Spec: <None>

Choose an attribute to sort on.

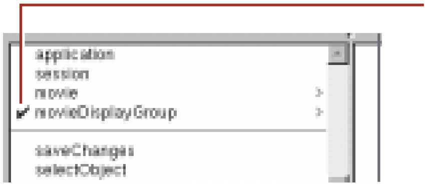
Select this option to sort from ‘A’ to ‘Z’.

2. Select the **title** attribute in the Sorting pop-up list.

3. Select Ascending.

4. Click OK.

WebObjects Builder stores your settings in an archive that specifies how to create and configure **movieDisplayGroup** at runtime. The archive is stored inside your Main component in a file named **Main.woo**. You can’t see the file from Project Builder because you’re not meant to edit it directly, but WebObjects Builder’s object browser shows you which of your component’s variables are initialized from the archive (or **woo** file) so you don’t have to view its contents directly.



An image in this column means that the variable can be initialized from the component’s archive.

A ✔ means that initialization parameters are already set. The variable is created and initialized from the archive as a part of the component’s initialization.

A ⊟ means that no initialization parameters have been set, and so the variable isn’t automatically created. Double-click the variable to configure it and add it to the archive.

Specifying Default Values for New Enterprise Objects

When new enterprise objects are created in your application, it’s common to assign default values to some of their properties. For example, in your Movies application it makes sense to assign a default value for the **title** attribute so a new movie won’t be displayed in the list of movies as a blank line.

You could write an action method for the Insert/New button instead of binding it directly to the display group **insert** action method. In the custom action, you would create a new Movie object, assign default values to it, and then insert the new object into the display group. However, there are two additional ways to specify default values for new enterprise objects, without making explicit assignments:

- Assign default values in the enterprise object class.
- Specify default values using a display group.

For a particular situation, one of the approaches is usually better than the other. If the default values are intrinsic to the enterprise object, assign them in the enterprise object class. For example, consider a Member class with a **memberSince** property. It’s likely that you would automatically assign the current date to **memberSince** instead of forcing a user to supply a value. You’ll see how to use this technique in “Adding Behavior to Your Enterprise Objects” (page 123).

On the other hand, if the default values are specific to an application or to a particular user interface, explicitly initialize the object in code or specify the default values using a display group. In the Movies application, the need for default values is motivated by Main’s user interface: you need to provide a default value so users can tell when a newly inserted record is selected. In another situation, you might not want a new movie to have a default title; you might instead want a new movie's title to be blank.

The Movies application specifies default values for newly created Movie objects using the display group, **movieDisplayGroup**.

1. Open **Main.java** in Project Builder.

2. Add the following constructor:

```
public Main() {
    super();
    NSMutableDictionary defaultValues = new NSMutableDictionary();
    defaultValues.setObjectForKey("New Movie Title", "title");
    movieDisplayGroup.setInsertedObjectDefaultValues(defaultValues);
}
```

   This method assigns the value "New Movie Title" as the default value for a new movie's **title** attribute. When **movieDisplayGroup** inserts a new movie (as it does when a user clicks the Insert/New button), it creates a new movie and assigns this default value to that movie.

Setting a Date Format

To change the way that dates are displayed, you assign a date format to the element that displays the dates.

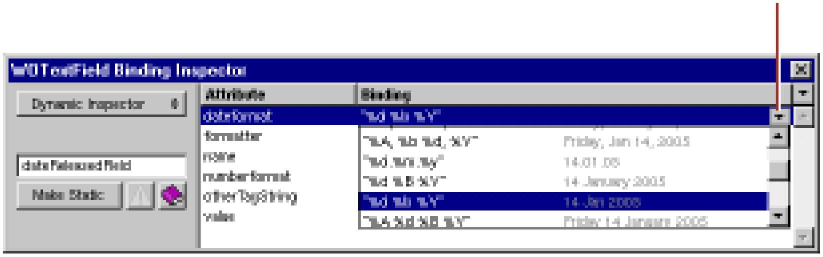
1. Using WebObjects Builder, inspect the **dateReleased** text field, which is near the bottom of the Main component window.

   Notice that the text field has a **dateformat** attribute that is bound to the string "%m/%d/%y". This binding tells the text field that it's displaying dates and describes how to format them. The %m conversion specifier stands for month as a decimal number, %d stands for day of the month, and %y stands for year without century.

2. Click the combo box on the right side of the binding column. Choose `"%d %B %Y"`.

   This date format displays dates as 14 Jan 2005. The %b conversion specifier stands for abbreviated month name, and %Y stands for year with century. You can create your own date formats with any of the conversion specifiers defined for dates. For more information, see the

NSGregorianDate class specification in the *Foundation Framework Reference*.



dateReleasedField

Attribute	Binding
dateformat	"%d %b %Y"
formatter	
name	
numberformat	
otherTagString	
value	

"%A, %b %d, %Y"	Friday, Jan 14, 2005
"%d.%m.%y"	14.01.05
"%d %B %Y"	14 January 2005
"%d %b %Y"	14 Jan 2005
"%A %d %B %Y"	Friday 14 January 2005

Click here to choose date format

Setting a Number Format

In addition to a **dateformat** attribute, text field elements also have a **numberformat** attribute.

1. Inspect the **revenue** text field.

   The **revenue** text field's **numberformat** attribute has no binding.

2. Using the combo box, change the text field's **numberformat** binding value to `"$###,##0"`.

   Using this number format, the Movies application formats the number 1750000 as $1,750,000. For more information on creating number formats, see the NSNumberFormatter class specification in the *Foundation Framework Reference*.

Optional Exercise

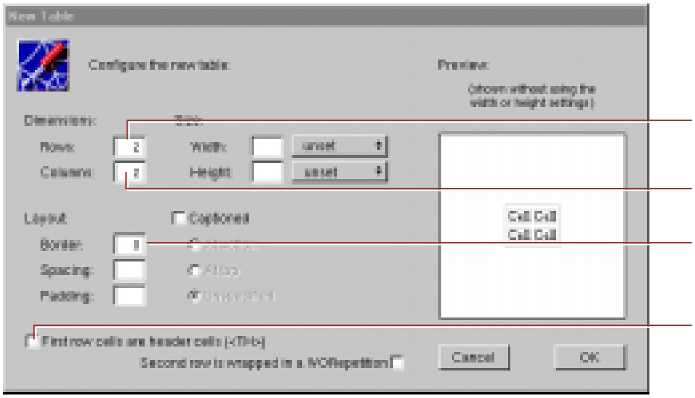
You can tidy up the user interface even further by putting the query part of the page in a table to match the editing part of the page. Also, you should consider capitalizing **Main.wo**'s text field labels.

To put the query part of the page in a table, follow these steps:

1. Put the cursor inside the form element before the "title" text field (in the Query By Example segment).

2. Click the ⊞ button in the toolbar to add a table.

The table panel appears.



New Table

Configure the new table:

Preview: (shown without using the width or height settings)

Dimensions:
Rows: 2
Columns: 2

Size:
Width: unset
Height: unset

Layout:
Border: 0
Spacing:
Padding:

□ Captioned

□ First row cells are header cells (<TH>)

Second row is wrapped in a WORepetition □

Enter the number of rows here.

Enter hte number of columns here.

Enter the border width here.

Uncheck this checkbox.

3. Enter 2 in the Rows field and 2 in the Columns field.

4. Enter 0 for in the Border field to remove the appearance of a border.

5. Uncheck “First row cells are header cells.” The first row text will not appear in bold.

6. Click OK. The table appears in your page.

7. Type the labels `Title:` and `Category:` in the cells in the first column.

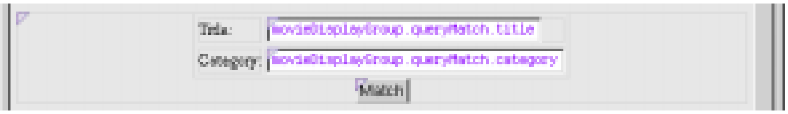
   The table doesn’t resize to accommodate new cell content until you’re done typing; that is, until you move the cursor out of the edited cell.

8. Cut and paste the query text fields into their corresponding table cells.

   Just click a text field to select it. When a text field is selected, it appears shaded with a box around it. Choose Cut from the Edit menu, double-click the cell to select its text, and choose Paste from the Edit menu.

9. Delete the old query field labels.

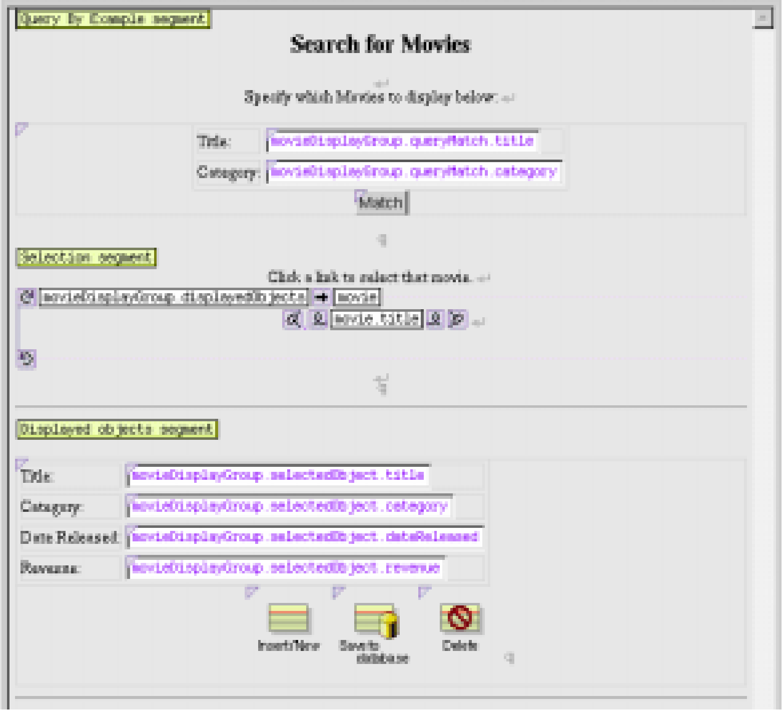
When you’re done, the query part should look like this:



Title: movieDisplayGroup.queryMatch.title

Category: movieDisplayGroup.queryMatch.category

Now edit the text labels in the editing part of the page and put any other finishing touches on the page that you want. The finished component might look something like this:



Search for Movies

Specify which Movies to display below:

Title: movieDisplayGroup.queryMatch.title
Category: movieDisplayGroup.queryMatch.category

Click a link to select that movie.

movieDisplayGroup.displayedObjects → movie
movie.title

Title: movieDisplayGroup.selectedObject.title
Category: movieDisplayGroup.selectedObject.category
Date Released: movieDisplayGroup.selectedObject.dateReleased
Revenue: movieDisplayGroup.selectedObject.revenue

Adding the MovieDetails Page

The MovieDetails page shows you the detailed information about a movie you select in the Main page. For this to work, the Main page has to tell the MovieDetails page which movie the user selected. The MovieDetails page keeps track of the selected movie in its own instance variable. In this section, you'll:

- Create a new component whose interface you'll create yourself.
- Assign Main's selected movie to a variable in the MovieDetails page.
- Create a way to navigate from Main to MovieDetails and back.

In the sections following this one, you'll extend the MovieDetails page to display movie roles and the starring actors.

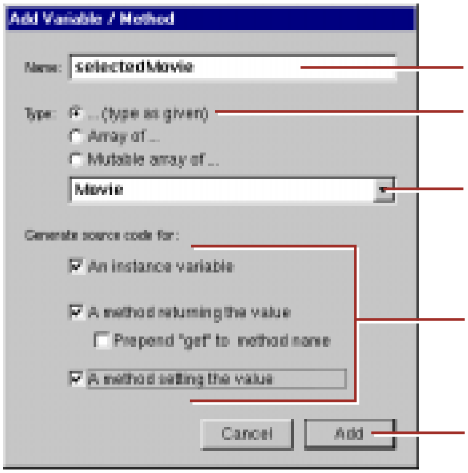
Creating the MovieDetails Component

1. In Project Builder, choose File ► New in Project.
2. In the New File panel, click the Web Components suitcase.
3. Type `MovieDetails` in the Name field.
4. Click OK.
5. In the wizard panel, choose None for available assistance.
6. Choose Java as the component language.
7. Click Finish.
8. Open the new component, **MovieDetails.wo**, in WebObjects Builder.

Storing the Selected Movie

Now, in the MovieDetails component, create a variable that holds the application's selected movie. Later on, you'll add code to the **Main.java** class that assigns Main's selected movie to this variable.

1. Choose Add Key from the pull-down list.



Name: selectedMovie

Type: ● ...(type as given)
○ Array of ...
○ Mutable array of ...

Movie

Generate source code for:

☑ An instance variable

☑ A method returning the value

□ Prepend "get" to method name

☑ A method setting the value

Type the variable name here.

Select this.

Choose Movie.

Check each of these boxes.

Click here when you're done.

2. Name the variable `selectedMovie`.

3. Set the variable's type to Movie.

   Movie isn't actually a class; it's an entity. It's listed in the combo box as a type along with entries for all the entities in your model. When you choose an entity as the type for your variable, WebObjects Builder recognizes that the variable is an enterprise object. Using information in the model, WebObjects Builder can determine the entity's corresponding enterprise object class and the properties of that class.

4. Check the "An instance variable" box.

5. Check the "A method returning the value" box.

6. Check the "A method setting the value" box.

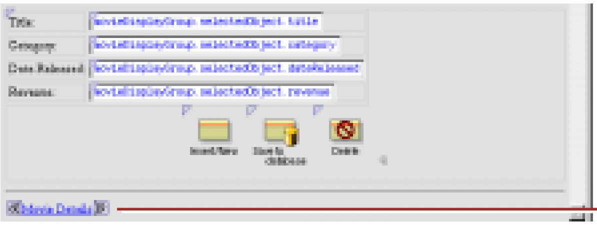
7. Click Add.

Navigating from Main to MovieDetails

To get to the MovieDetails page from the Main page, users use a hyperlink. Clicking the hyperlink should set MovieDetail's **selectedMovie** variable and then open the MovieDetails page.

1. Add a hyperlink at the bottom of the Main component.

2. Replace the text “Hyperlink” with “Movie Details.”


Title: movieDisplayGroup.selectedObject.title
Category: movieDisplayGroup.selectedObject.category
Date Released: movieDisplayGroup.selectedObject.dateReleased
Revenue: movieDisplayGroup.selectedObject.revenue

Add the hyperlink below the horizontal rule.

3. Choose Add Action from the pull-down list.

4. In the Add Action panel, type `showDetails` in the Name field.

5. Select `MovieDetails` from the “Page returned” combo box.

6. Click Add.

7. Bind the **showDetails** action to the hyperlink’s **action** attribute.

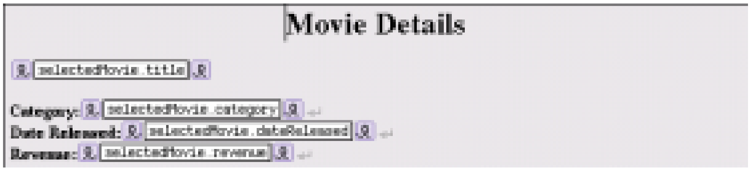
8. In Project Builder, modify the **showDetails** action in **Main.java** to look like the following:

```
public MovieDetails showDetails() {
    MovieDetails nextPage =
(MovieDetails)pageWithName("MovieDetails");

    // Initialize your component here
    EOEnterpriseObject selection =
        (EOEnterpriseObject)movieDisplayGroup.selectedObject();
    nextPage.setSelectedMovie(selection);

    return nextPage;
}
```

This method creates the MovieDetails page and then invokes its **setSelectedMovie** method with the movie that’s selected in the Main page. The display group method **selectedObject** returns its selected object, which, in the Main component, is set when a user clicks a movie title hyperlink.

Designing MovieDetails' User Interface

Now lay out the user interface for MovieDetails. When you're done, your component should look like the following:



Movie Details

selectedMovie.title

Category: selectedMovie.category

Date Released: selectedMovie.dateReleased

Revenue: selectedMovie.revenue

1. Create a top-level heading with the text `Movie Details`.

   Recall that to create a top-level heading, you type the text of the heading, select the text, click the button to add a heading element around the text, and then use the Inspector to set the heading's level, as you did in "Using the Inspector" (page 24).

2. Below the heading, add a string element.

3. With the string element selected, add a heading.

   This adds a new level-1 heading element around the string. The MovieDetails page will show the title of the selected movie in this heading.

4. Click `<H1>` in the path view. The Inspector now displays the Heading Level.

5. Click 3 in the Heading Inspector.

6. Add labels and string elements to display the selected movie's category, date released, and revenue.

7. Bold the labels.

8. Bind **selectedMovie.title** to the **value** attribute of the first string element (the one in the heading).

9. Similarly, create bindings for the Category, Date Released, and Revenue strings.

10. At the bottom of the page, add a horizontal rule.

Adding Date and Number Formats

String elements have **dateformat** and **numberformat** attributes just like text field elements. Create bindings for the Date Released and Revenue strings so that **dateReleased** and **revenue** values are displayed the way they are in the Main page.

1. Add the date format `"%d %b %Y"` to the Date Released string. You can select the format from the combo box in the Inspector's binding column.

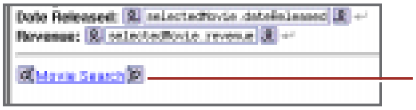
2. Add the number format `"$###,##0"` to the Revenue string. You can select the format from the combo box in the Inspector's binding column.

Navigating from MovieDetails to Main

Now add a hyperlink to the MovieDetails page so users can navigate back to the Main page from MovieDetails.

1. Add a hyperlink to the bottom of the page.

2. Label it `Movie Search`.



Date Released: selectedMovie.dateReleased
Revenue: selectedMovie.revenue

Movie Search

Add the hyperlink here.

3. Bind the hyperlink's **pageName** attribute to the text (including the quotes) `"Main"`. You can select "Main" from the combo box in the inspector's binding column.

   Recall that the **pageName** attribute is a mechanism for navigating to another page without writing code. By setting the attribute to "Main", you're telling the application to open the MovieSearch page when the hyperlink is clicked.

Running Movies

Be sure that all your project's files are saved (including the components in WebObjects Builder), and build and run your application. In the Main page, select a movie and click the Movie Details link. The MovieDetails page should display all the movie's information.

Refining Your Model

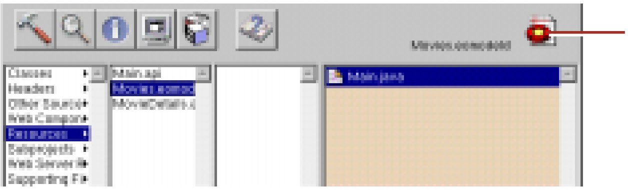
The model created for you by the wizard is just a starting point. For most applications, you need to do some additional work to your model to make it useful in your application. To refine your model so that it can be used in the Movies application, you'll ultimately need to do all of the following:

- Remove primary and foreign keys as class properties.
- Add relationships to your model if the wizard didn't have enough information to add them for you.
- Configure your model's relationships in the Advanced Relationship Inspector.
- Generate source files for the Talent class.

These steps are described in more detail throughout the rest of this tutorial.

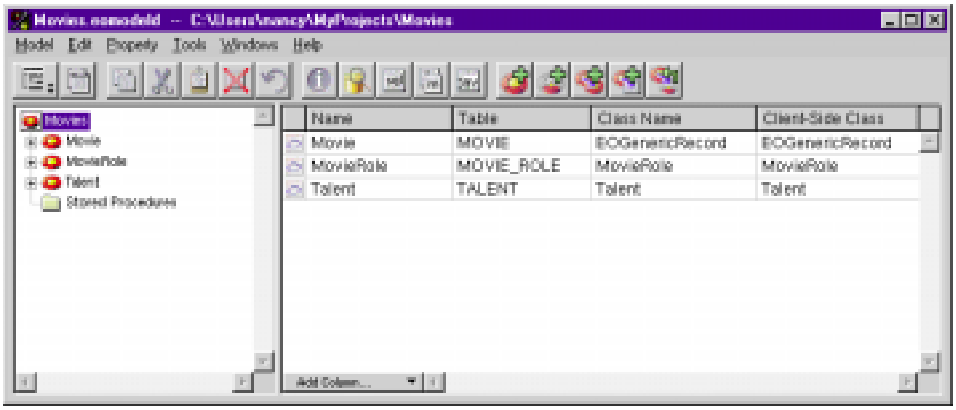
Opening Your Model

1. In Project Builder, click the Resources category.
2. Select **Movies.eomodeld**.
3. Double-click the model icon.



Double-click to open the model.

Project Builder opens your model file in EOModeler, launching EOModeler first if it isn't already running. EOModeler displays your model in the Model Editor. It lists the entities for the tables you specified in the wizard—Movie, MovieRole, and Talent.



Name	Table	Class Name	Client-Side Class
Movie	MOVIE	EOGenericRecord	EOGenericRecord
MovieRole	MOVIE_ROLE	MovieRole	MovieRole
Talent	TALENT	Talent	Talent

Removing Foreign Keys as Class Properties

By default, the wizard makes all of an entity's attributes, except primary keys, *class properties*. When an attribute is a class property, it means that the property is a part of your enterprise object, usually as an instance variable.

You should mark as class properties only those attributes whose values are meaningful in the objects that are created when you fetch from the database. Attributes that are essentially database artifacts, such as primary and foreign keys, shouldn't be marked as class properties unless the key has meaning to the user and must be displayed in the user interface.

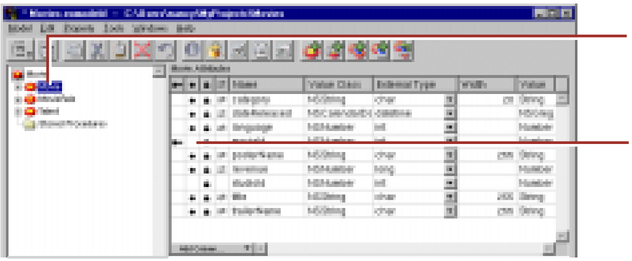
Eliminating primary and foreign keys as class properties has no adverse effect on how Enterprise Objects Framework manages enterprise objects in your application.

1. In the left frame (or *tree view*), click the Movie entity.

   The right frame switches from a view of the entities in the model to a view of Movie's attributes.

   A 🔑 symbol in the first column means that the attribute is a primary key for the selected entity. A ◆ symbol in the second column means that the attribute is a class property.

2. Click in the Class Property column to remove the ◆ symbol for the **studioId** attribute, which is a foreign key. The wizard didn't make **movieId** a class property because it is a primary key.



Click an entity in this frame to select the entity.

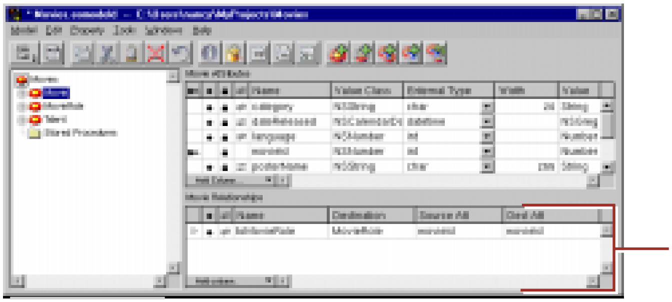
Click in an attribute's Class Property column to remove it as a class property.

Adding Relationships to Your Model

The Movies application uses two pairs of inverse relationships. The first pair defines the relationship between the Movie and MovieRole entities, while the second pair defines the relationship between the MovieRole and Talent entities. An Enterprise Objects Framework relationship is *directed*; that is, a relationship has a source and a destination. Generally models define a relationship for each direction.

1. Select the Movie entity.

   The right frame of the Model Editor shows the Movie's relationships as well as its attributes.



The selected entity's relationships are displayed here.

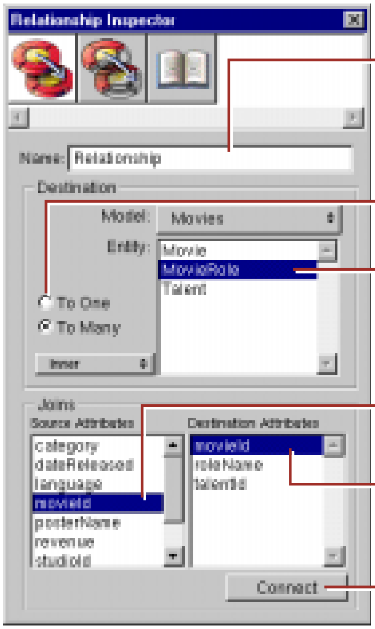
Your model's Movie entity might have a different name than the **toMovieRole** relationship shown above. That's because the wizard created your relationship, and the relationship's name is dependent on the adaptor the wizard used. Adaptors don't all have the same naming convention for to-many relationships. For example, the Oracle adaptor names Movie's relationship **movieRoleArray** instead of **toMovieRole**.

If your Movie entity doesn't have a **toMovieRole** relationship, it means that the database server's schema information for your database didn't have enough information for the wizard to create them. You need to create them by hand now. The next several steps explain how.

2. Choose Property ► Add Relationship.

   A new relationship named "Relationship" is added in the table view at the bottom of the Model Editor. The new relationship is already selected.

3. With the relationship selected in the right frame of the Model Editor, click the ⓘ button (in the toolbar) to inspect the relationship.



Relationship Inspector

Name: Relationship

Destination

Model: Movies

Entity: Movie, MovieRole, Talent

○ To One
● To Many

Inner

Joins

Source Attributes: category, dateReleased, language, movieId, posterName, revenue, studioId

Destination Attributes: movieId, roleName, talentId

Connect

Don't change the relationship's name, because EOModeler updates the name for you automatically when you connect the Destination and Join properties.

First select whether the relationship is to-one or to-many.

Then select a destination entity.

Select a source attribute...

...and a matching destination attribute.

When you're done, click here.

4. In the Inspector, select the To Many option.

5. Select MovieRole as the destination entity.

6. Select **movieId** in the Source Attributes list.

7. Select **movieId** in the Destination Attributes list.

8. Click Connect.

EOModeler automatically renames the relationship based on the name of the destination entity. For example, after connecting a to-many relationship from Movie to MovieRole, EOModeler names the relationship "toMovieRole." To-one relationships are named with the singular form of the destination entity's name. For example, EOModeler names the inverse to-one relationship (from MovieRole to Movie) "toMovie."

If the wizard created your relationship and used a name other than "toMovieRole," consider renaming the relationship. The rest of this tutorial assumes that your relationships are named using EOModeler's naming convention.

9. Repeat the steps above to create the following relationships (if they do not already exist):

   A to-one relationship named "toMovie" in the MovieRole entity where:

   - The destination entity is Movie.
   - The source attribute is **movieId**.
   - The destination attribute is **movieId**.

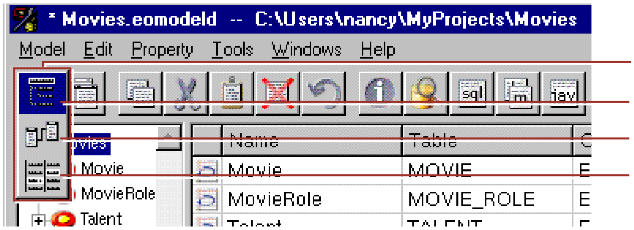
   A to-one relationship named "toTalent" in the MovieRole entity where:

   - The destination entity is Talent.
   - The source attribute is **talentId**.
   - The destination attribute is **talentId**.

   A to-many relationship named "toMovieRole" in the Talent entity where:

   - The destination entity is MovieRole.
   - The source attribute is **talentId**.
   - The destination attribute is **talentId**.

10. Choose ▤ in the toolbar pop-up list to switch the Model Editor to Diagram View.



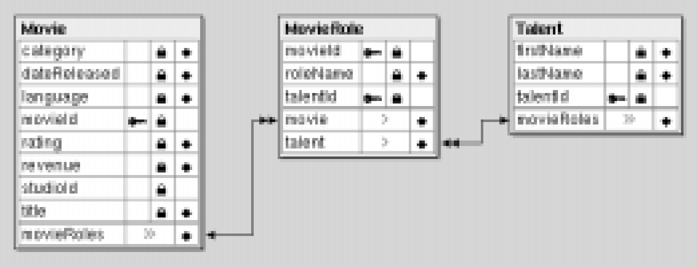
Use this pop-up list to switch to a different view.

Switches to Table View.

Switches to Diagram View.

Switches to Browser View.

At this point your model has all the relationships it needs. The Diagram View gives you an overview of the entities in the model and their relationships to other entities.



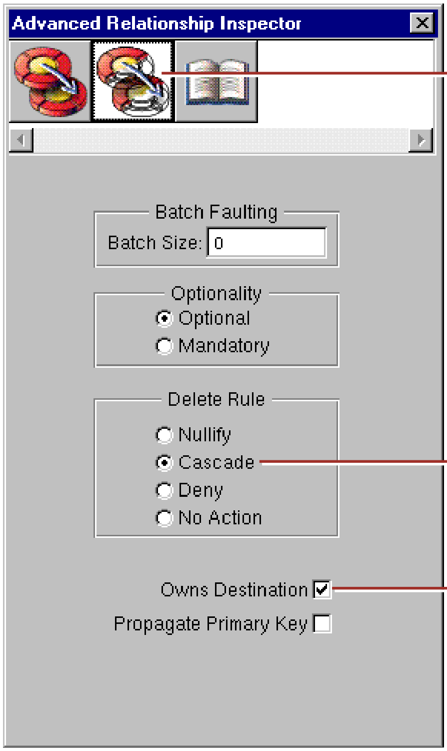
You can also use the Diagram View to edit your model. Double-click an attribute or relationship to change its name. To create a relationship and its inverse, Control-drag from the relationship's source attribute to its destination attribute.

Using the Advanced Relationship Inspector

There are several additional settings you use to configure a relationship's referential integrity rules. For these, use the Advanced Relationship Inspector.

1. Inspect Movie's **toMovieRole** relationship.

2. In the Inspector, click the Advanced Relationship button.



Batch Faulting

Batch Size: 0

Optionality

● Optional

○ Mandatory

Delete Rule

○ Nullify

● Cascade

○ Deny

○ No Action

Owns Destination ☑

Propagate Primary Key □

Advanced Relationship button.

This should be selected.

This box should be checked.

3. Ensure that the delete rule is set to Cascade.

   If the wizard created relationships for you, the relationship’s delete rule should already be set to Cascade. You specified this in the wizard. If you created your relationships by hand, you’ll have to set the delete rule yourself.

4. Ensure that the Owns Destination box is checked.

   As with the delete rule, if the wizard created relationships for you, the relationship’s Owns Destination box should already be checked. If you created your relationships by hand, you’ll have to check this box yourself.

5. Check the Propagate Primary Key box.

   A relationship that propagates its primary key *propagates* its key value to newly inserted objects in the destination of the relationship. In this case, checking the Propagate Primary Key box means that if you create a new

MovieRole and add it to a Movie's list of MovieRoles, the Movie object automatically assigns its **movieId** value as the value for the new MovieRole's **movieId** property.

This option is usually used with relationships that own their destination. For more information on propagating primary keys, see "Where Do Primary Keys Come From?" (page 110).

6. Ensure that Talent's **toMovieRole** relationship has its delete rule set to Deny.
7. Ensure that Talent's **toMovieRole** relationship owns its destination.
8. Set Talent's **toMovieRole** relationship to propagate its primary key.
9. Choose Model ► Save (File ► on Windows NT) to save your model.

Where Do Primary Keys Come From?

Enterprise Objects Framework uses primary keys to identify enterprise objects in memory, and it works best if you never change an enterprise object's primary key from its initial value. Consequently, applications usually generate and assign primary key values automatically instead of having users provide them. For example, the Movies application assigns a **movieId** value to a new movie when it's created, and the value never changes afterward. The Movies interface doesn't even display **movieId** values because they aren't meaningful to users of the application.

Enterprise Objects Framework provides several mechanisms for generating and assigning unique values to primary key attributes. By default, Enterprise Objects Framework uses a native database mechanism to assign primary key values. See the chapter "Answers to Common Design Questions" in the *Enterprise Objects Framework Developer's Guide* for more information.

The Movies application generates primary key values for Movie and Talent objects using the default mechanism, but MovieRole is a special case because:

- MovieRole's primary key is compound. The default behavior of generating a primary key value using a native database mechanism works only on simple (not compound) primary keys.

- A MovieRole's primary key attributes, **movieId** and **talentId**, must match the corresponding attributes in the MovieRole's Movie and Talent objects. The default mechanism generates new, unique values.

Instead of the default mechanism, Enterprise Objects Framework uses primary key propagation to assign primary keys to MovieRole objects. By configuring the Movie's **toMovieRole** relationship to propagate primary key, the Framework knows to assign a new MovieRole's **movieId** to the same value as the **movieId** of the MovieRole's Movie. Similarly, a new MovieRole's **talentId** is set to the same value as the **talentId** of the MovieRole's Talent.

Setting Up a Master-Detail Configuration

So far your Movies application fetches, inserts, updates, and deletes only Movie objects. Considered alone, a Movie object isn't as interesting as it is when it's related to actors and roles. In this section, you'll add MovieRole and Talent objects to the Movies application.

The relationships defined in your model now come into play. Using Movie's **toMovieRole** relationship, you can display the MovieRoles for the selected Movie. In this type of configuration, called *master-detail*, a master display group holds enterprise objects for the source of a relationship, while a detail display group holds records for the destination. As individual records are selected in the master display group, the detail display group gets a new set of enterprise objects to correspond to the selection in the master.

In the Movies application, the master-detail configuration is built around Movie's **toMovieRole** relationship. The configuration is split across two pages in the application. The master, **movieDisplayGroup**, is in the Main component, while the detail is in MovieDetails.

In this section, you'll:

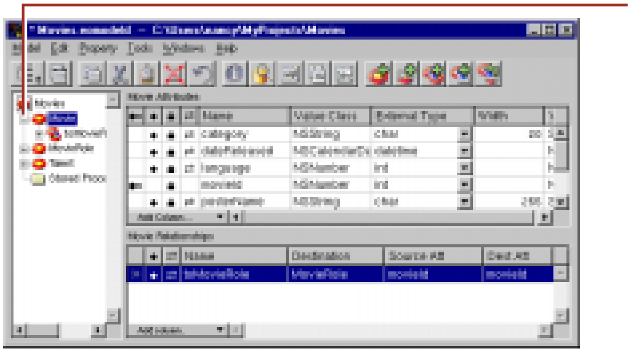
- Create and configure the detail display group.
- Extend the MovieDetails user interface to hold MovieRole and Talent information.

Creating a Detail Display Group

You can create a detail display group several different ways. You can write a declaration for it in Project Builder, or you can use WebObjects Builder's Add

Variable/Method command. But the easiest way to create a detail display group is by dragging a relationship from EOModeler into your component, as described below.

1. In EOModeler's tree view, expand the Movie entity.



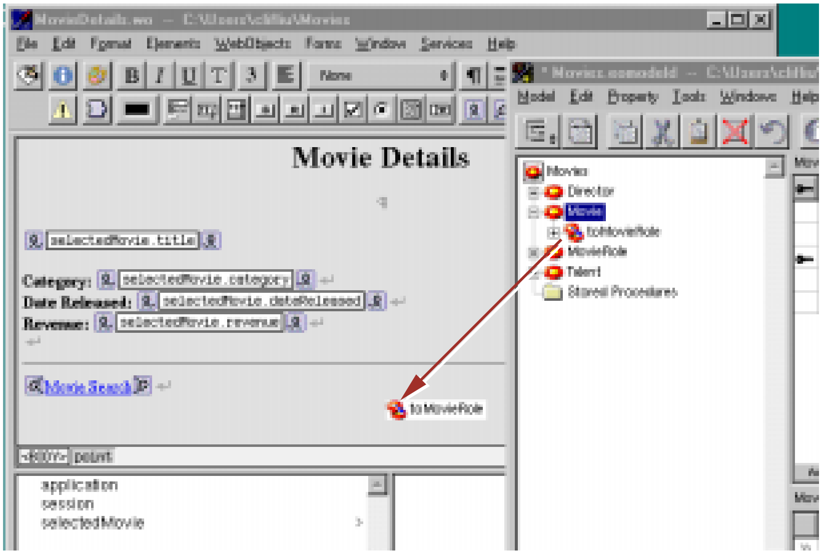
Click here to expand or contract an entity.

⊟ means that the entity is already expanded. Click the dash to contract the entity.

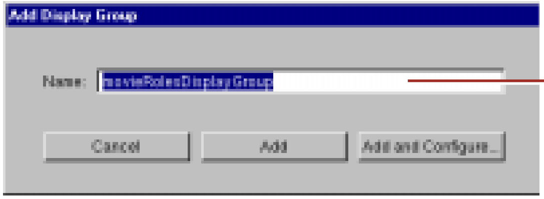
⊞ means that the entity can be expanded to display its relationships. Click the plus to expand the entity.

If an entity has neither a dash nor a plus, the entity has no relationships, and therefore can't be expanded.

2. Drag the Movie's **toMovieRole** relationship from the tree view into the MovieDetails component's object browser.



An Add Display Group panel opens.


Add Display Group

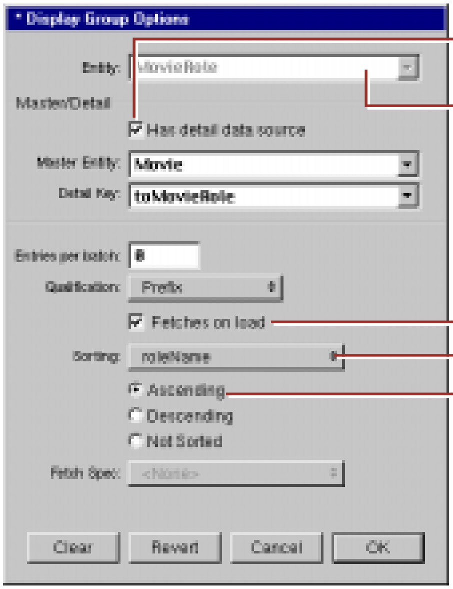
Name: movieRolesDisplayGroup

WebObjects Builder assigns a default name based on the relationship name.

3. In the Add Display Group panel, change the name to `movieRoleDisplayGroup`.

4. Click Add and Configure.

   The Display Group Options panel opens so you can immediately configure the newly created display group.


Display Group Options

Entity: MovieRole

Master/Detail

☑ Has detail data source

Master Entity: Movie

Detail Key: toMovieRole

Entries per batch: 8

Qualification: Prefix

☑ Fetches on load

Sorting: roleName

● Ascending

○ Descending

○ Not Sorted

Fetch Spec: <None>

Identifies this display group as a detail display group.

You can't set the entity of a detail display group. The entity is computed from the Master/Detail settings.

Check this box so the display group automatically fetches its objects.

Sort MovieRole objects by roleName...

...from 'A' to 'Z'.

Ensure that the "Has detail data source" box is checked. This means that **movieRoleDisplayGroup** gets its objects from a EODetailDataSource object.

All display groups use some kind of *data source* to fetch their objects. A data source is an object that exists primarily as a simple means for a WODisplayGroup to access a store of objects. It's through a data source that a display group fetches, inserts, updates, and deletes database records.

An EODetailDataSource is a subclass of EODataSource that's intended for use in master-detail configurations. A detail data source keeps track of a *master object* and a *detail key*. The master object is typically the selected object in a master display group, but a master display group isn't strictly required. The detail key is the name of the relationship on which the master-detail configuration is based. When a detail display group asks its data source to fetch, the EODetailDataSource simply gets the destination objects from the master object as follows:

```
detailObjects = masterObject.valueForKey(detailKey);
```

In your master-detail configuration, the master object is the selected Movie, and the detail key is **toMovieRole**. When **movieRoleDisplayGroup** asks its data source for its MovieRole objects, the detail WODisplayGroup returns the objects in the selected Movie's **toMovieRole** array of MovieRoles. Similarly, when MovieRole objects are inserted or deleted in **movieRoleDisplayGroup**, they are added and removed from the master object's **toMovieRole** array.

5. Set the display group to sort alphabetically by **roleName**.

6. Check the "Fetches on load" box.

   When "Fetches on load" is selected, the display group fetches its objects as soon as the component is loaded into the application. You want this feature in the MovieDetails page so that users are immediately presented with the selected movie's roles. In contrast, the Main page does not fetch on load; it shouldn't present a list of movies until the user has entered search criteria and clicked Match.

7. Click OK.

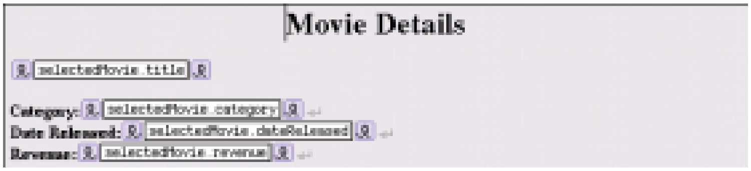
8. In Project Builder, modify MovieDetail's **setSelectedMovie** method to look like the following:

```
public void setSelectedMovie(EOEnterpriseObject newSelectedMovie) {
    selectedMovie = newSelectedMovie;
    movieRoleDisplayGroup.setMasterObject(newSelectedMovie);
}
```

With this addition, whenever a user navigates to the MovieDetails page, **setSelectedMovie** updates the master object of the **movieRoleDisplayGroup** so it displays the corresponding MovieRole objects.

Adding a Repetition

Now you'll extend the user interface of the MovieDetails component to display the actors in the selected movie. Because different movies have different numbers of roles, you need the dynamism of a repetition element. When you're done adding the repetition, your component should look like this:



Movie Details

selectedMovie.title

Category: selectedMovie.category

Date Released: selectedMovie.dateReleased

Revenue: selectedMovie.revenue

1. In the MovieDetails component window, add the bolded text **Starring:** beneath the Revenue line.

2. Below the Starring label, add a repetition.

3. Replace the "Repetition" text with three string elements.

   The strings should all be on the same line, so don't type carriage returns between them.

4. Type a space between the first two strings and the word " `as` " (with a space before and after) between the last two.

5. Add a carriage return after the last string.

Configuring a Repetition

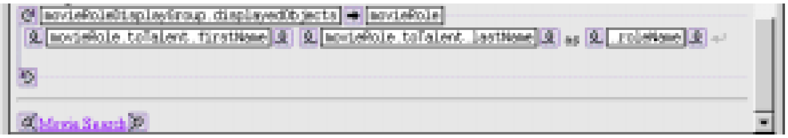
Now configure MovieDetails' repetition in a way similar to the way Main's repetition is configured. First you need to create a new variable to bind to the repetition's **item** attribute.

1. Use the Add Key command to add a new variable, **movieRole,** whose type is set to the MovieRole entity.

Don’t create set and get methods for **movieRole**. You won’t need accessor methods because the variable is used only within the MovieDetails component and shouldn’t be visible to any other classes.

2. Bind **movieRoleDisplayGroup.displayedObjects** to the repetition’s **list** attribute.
3. Bind **movieRole** to the repetition’s **item** attribute.
4. Bind **movieRole.toTalent.firstName** to the **value** attribute of the first string in the repetition.
5. Bind **movieRole.toTalent.lastName** to the **value** attribute of the second string.
6. Bind **movieRole.roleName** to the **value** attribute of the last string.

When you’re done, the repetition bindings should look like the following:



movieRoleDisplayGroup.displayedObjects → movieRole

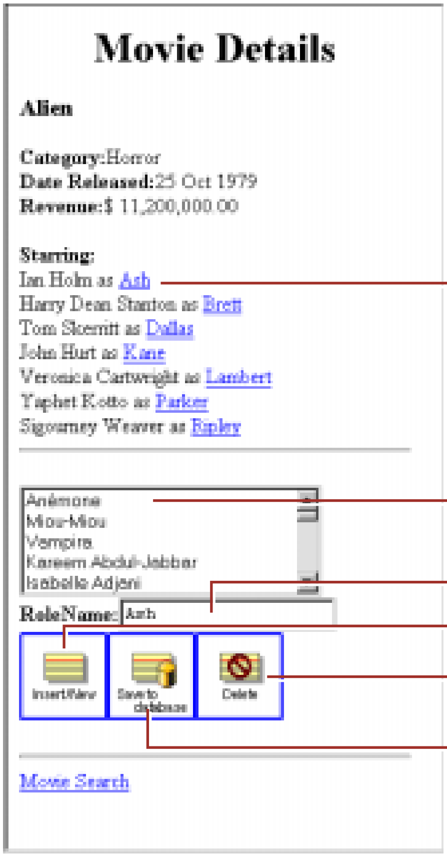
movieRole.toTalent.firstName movieRole.toTalent.lastName as ..roleName

Running Movies

Be sure that all your project’s files are saved (including the components in WebObjects Builder and the model in EOModeler), and build and run your application. In the Main page, select a movie and click the Movie Details link. Now, in addition to displaying all the movie’s information, the Movie Details page should also display the movie’s roles and actors.

Updating Objects in the Detail Display Group

In this section, you’ll add the ability to insert, update, and delete movie roles. The MovieDetails page will then look something like this:



Movie Details

Alien

Category: Horror
Date Released:25 Oct 1979
Revenue:$ 11,200,000.00

Starring:
Ian Holm as Ash
Harry Dean Stanton as Brett
Tom Skerritt as Dallas
John Hurt as Kane
Veronica Cartwright as Lambert
Yaphet Kotto as Parker
Sigourney Weaver as Ripley

Anémone
Mou-Mou
Vampira
Kareem Abdul-Jabbar
Isabelle Adjani

RoleName: Ash

Movie Search

Click a role to select it and display its information in the editing part below.

Use the browser to choose an actor for the selected role.

Edit the name of the selected role.

Click here to create a new, empty role.

Click here to delete the selected role.

Click here to save your work in the database (add the new roles you inserted, remove the roles you deleted, and save changes you made to existing roles).

Many of the features in this page are similar to features in the Main page, but in this section you perform by hand the tasks the wizard performed for you to create Main. Already you've learned how to create a WODisplayGroup variable and how to bind it to dynamic elements. In this section you'll:

- Write code to update a display group's selected object.
- Create and configure a browser.
- Create a custom enterprise object class.
- Use display group actions to configure image buttons to insert, update, and delete.

Managing a WODisplayGroup's Selection

Remember how clicking a movie title in the Main page selects the corresponding Movie object in **movieDisplayGroup**. MovieDetails has a similar behavior for selecting a MovieRole object in **movieRoleDisplayGroup**.

First you need to add a hyperlink element around the repetition's role name string so that users can select a particular MovieRole. When a user clicks one of the movie role hyperlinks, the application should select the corresponding MovieRole object in the **movieRoleDisplayGroup**.

1. Select the repetition's role name string element.

2. Click the Add WOHyperlink button in the toolbar to add a hyperlink element around the string.

   Now you need to create an action method to invoke when the hyperlink is clicked.

3. Use the Add Action command in the pull-down list to add an action named **selectObject**, returning **null**.

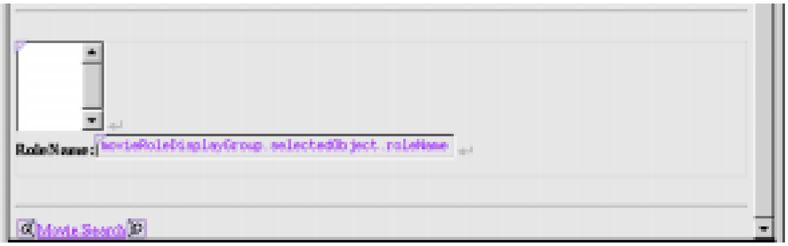
4. Bind the **selectObject** method to the hyperlink's **action** attribute.

5. Now write the code for **selectObject** in **MovieDetail.java**. Modify the **selectObject** action to look like the following:

```
public WOComponent selectObject() {
   movieRoleDisplayGroup.selectObject(movieRole);
   return null;
}
```

Adding a Form

Now lay out the user interface used to view and edit the selected MovieRole. When you're done, it should look like the following:



Role Name: movieRoleDisplayGroup.selectedObject.roleName

1. Add another horizontal rule after the repetition.

2. Use the ▤ button to add a WOForm element between the two horizontal rules.

3. While the Form text is highlighted, click the ▤ button to replace the text with a WOBrowser element.

4. Beneath the browser (within the bounds of the new form), type the bolded text `Role Name:`.

5. Add a text field.

6. Bind the text field's **value** attribute to **movieRoleDisplayGroup.selectedObject.roleName**.

Adding a Talent Display Group

The browser you just created is going to display a list of Talent objects. Like a repetition element, a browser has **list** and **item** attributes. As the browser moves through its **list**, the browser sets **item** to the object at the current index. The Movies application uses a display group to provide the browser with a list of Talent objects, so now you need to create the new display group and a variable to bind to the browser's **item** attribute.

1. Use the Add Key command to create two new instance variables:

   - **talentDisplayGroup**, whose type is WODisplayGroup
   - **talent**, whose type is Talent

You don't need to add set and get methods for the variables.

2. Using the Display Group Options panel, assign the **talentDisplayGroup** object's entity to Talent.

   Remember that to open the Display Group Options panel, simply double-click the **talentDisplayGroup** variable in the object browser. The ⊟ icon initially displayed next to the variable indicates that initialization parameters have not yet been set.

3. Configure **talentDisplayGroup** to sort its objects alphabetically (ascending) by **lastName**.

4. Configure it to fetch on load and click OK.

   After you configure **talentDisplayGroup**, the object browser shows a ✓ icon next to the variable.

The Movies application uses a display group to provide Talent objects, but you could fetch the Talent objects from the database without one. Display groups provide a simple way to fetch, insert, update, and delete enterprise objects without writing much, if any, code. To get finer-grained control over these operations, you can work directly with an EOEditingContext object. An editing context can do everything a display group does and much more, but you have to write more code to use one. For more information, see the EOEditingContext class specification in the *Enterprise Objects Framework Reference*.

Configuring the Browser

Create your browser's bindings. The steps are similar to those for creating bindings for a repetition.

1. Bind **talentDisplayGroup.displayedObjects** to the browser's **list** attribute.

2. Bind **talent** to the browser's **item** attribute.

3. Bind **talent.lastName** to the browser's **value** attribute.

   The **value** attribute tells the browser what string to display. For each **item** in its **list**, the browser evaluates the **item**'s **value**.

The browser in the MovieDetails page should display the actors' full names, but there isn't an attribute for full name. In the next section, you'll create a custom Talent class that implements a **fullName** method, but for now just use **talent.lastName** as the **value** attribute.

A browser also has a **selections** attribute that should be bound to an array of objects. A browser's selection can be zero, one, or many objects; but in the Talent browser, the selection should refer to a single object. Consequently, you need to add two methods to manage the browser's selection: one to return an array containing the selected Talent and one to set the selected Talent from an array object.

4. Add the method **talentSelection** to the **MovieDetails.java** class as follows:

```
public NSArray talentSelection() {
    EOEnterpriseObject aTalent;
    EOEnterpriseObject aMovieRole =
       (EOEnterpriseObject)movieRoleDisplayGroup.selectedObject();

    if (aMovieRole == null){
       return null;
    }
    aTalent = (EOEnterpriseObject)aMovieRole.valueForKey("toTalent");
    if (aTalent == null){
       return null;
    } else {
       return new NSArray(aTalent);
    }
}
```

Because the browser expects an array for its **selections** attribute, this method packages the selected MovieRole's **talent** object in an array. If the selected MovieRole object is **null**, **talentSelection** simply returns **null** to indicate that the browser shouldn't set a selection.

5. Add the method **setTalentSelection** as follows:

```
public void setTalentSelection(NSArray talentArray){
    if (talentArray.count() > 0){
       EOEnterpriseObject aMovieRole =
          (EOEnterpriseObject)movieRoleDisplayGroup.selectedObject();
       EOEnterpriseObject selectedTalent =
          (EOEnterpriseObject)talentArray.objectAtIndex(0);

       aMovieRole.addObjectToBothSidesOfRelationshipWithKey(
          selectedTalent, "toTalent");
    }
}
```

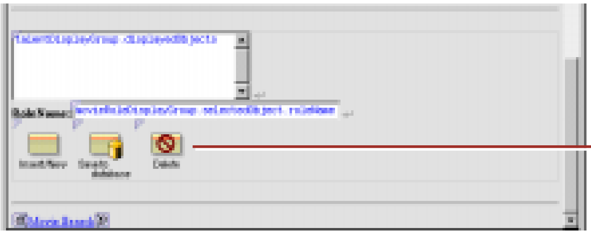
Again because the browser uses an array for its **selections** attribute, the **setTalentSelection** method must take an array as its argument. If the size of **talentArray** is nonzero, then this method sets the selected MovieRole's **talent** to the first object in the array. Note that by default, a user can't select more than one actor in a browser.

With the addition of these methods, WebObjects Builder now displays **talentSelection** in MovieDetail's object browser.

6. Bind **talentSelection** to the browser's **selections** attribute.

Adding Insert, Save, and Delete Buttons

Now add the buttons that let users insert, save, and delete MovieRoles. When you're done, it should look like the following:


talentDisplayGroup.displayedObjects
Role Name:
movieRoleDisplayGroup.selectedObject.roleName
Insert/New
Save to database
Delete
Movie Search

Add the image buttons inside the form element, which is bounded by a light gray box.

1. Inside the form, add three image buttons below the Role Name text field.

2. Inspect the first active image element.

3. Bind the **filename** attribute to the text (including the quotes) `"DBWizardInsert.gif"`.

4. Follow the same procedure to set the second image's **filename** attribute to the text (including the quotes) `"DBWizardUpdate.gif"`.

5. Set the last image's **filename** attribute to the text (including the quotes) `"DBWizardDelete.gif"`.

   The WODisplayGroup class defines the actions **insert** and **delete**. You'll bind to the Insert/New and Delete buttons. It doesn't, however, provide a save method. You'll have to provide that yourself.

6. Copy the **saveChanges** method from the **Main.java** class and paste it into the **MovieDetails.java** class:

```
public void saveChanges() throws Exception {
    try {
       this.session().defaultEditingContext().saveChanges();
    }
    catch (Exception exception) {
       System.err.println("Cannot save changes ");
       throw exception;
    }
}
```

7. Bind **movieRoleDisplayGroup.insert** to the Insert/New image's **action** attribute.

8. Bind the **saveChanges** method to the "Save to database" image's **action** attribute.

9. Bind **movieRoleDisplayGroup.delete** to the Delete image's **action** attribute.

Adding Behavior to Your Enterprise Objects

Right now, the Movies application maps all its entities to the EOGenericRecord class. As the preceding sections illustrate, you can go quite far in an application using just this default enterprise object class, but now you need to add some custom classes to the Movies application.

In this section, you'll learn how to:

- Generate source code for a custom enterprise object class.
- Provide default values in a custom enterprise object class.

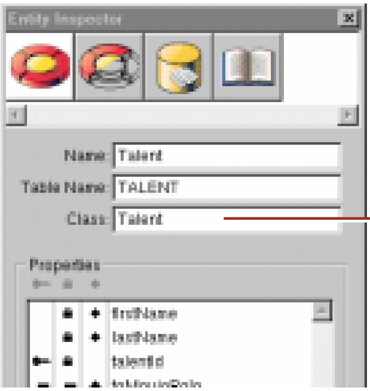
You'll create custom classes for the Talent and MovieRole entities. In the Talent class, you'll write a **fullName** method that concatenates a Talent's first and last names. You'll use the method to populate MovieDetail's browser element. In the MovieRole class, you'll provide default values for newly inserted MovieRoles so they don't show up in the list of movie roles as a blank line.

Specifying Custom Enterprise Object Classes

Unless you specify otherwise, EOModeler maps entities to the EOGenericRecord class. When you want to use a custom class instead, you need to specify that custom class in the model.

1. In EOModeler, inspect the Talent entity.

2. In the Entity Inspector for Talent, type `Talent` in the Class field.



Name: Talent
Table Name: TALENT
Class: Talent

Type the name of your custom class here.

3. Set the MovieRole entity's class to MovieRole.

Now you can generate the source files for your Talent and MovieRole classes.

Generating Custom Enterprise Object Classes

You can easily create a custom class to hold your business logic: EOModeler provides a command to generate enterprise object classes.

1. In EOModeler, select the Talent entity.
2. Choose Property ► Generate Java Files.

   A Choose Class Name panel opens. If you opened the model file from Project Builder, the Choose Class Name panel displays the project as the destination directory and **Talent.java** as the default filename.

3. Ensure that the Movies project directory is selected.
4. Click Save.

   A panel opens, asking if you want to insert the file in your project.

5. Click Yes.

   EOModeler creates the source file **Talent.java** and adds it to your project.

6. Follow the same procedure for MovieRole.

Adding Custom Behavior to Talent

Now add the **fullName** method to Talent and bind it to the browser.

1. Open **Talent.java** in Project Builder.

   The class file declares instance variables for all of Talent's class properties (**firstName** and **lastName**) and implements set and get methods for those instance variables.

2. Add the method, **fullName**, as follows.

```
public String fullName(){
   return firstName() + " " + lastName();
}
```

   After you save, **fullName** appears in the object browser of WebObjects Builder as a property of Talent.

3. Bind **talent.fullName** to the browser's **value** attribute.

Providing Default Values in MovieRole

As discussed in "Specifying Default Values for New Enterprise Objects" (page 93), there are two main ways to specify default values for new enterprise objects without making explicit assignments:

- Assign default values in the enterprise object class.
- Specify default values using a display group.

For the Movie class, you specified default values using a display group. This approach is also the more appropriate choice for the MovieRole class, but you'll use the other approach for MovieRole just to see how its done.

1. Open **MovieRole.java** in Project Builder.

2. Add the method, **awakeFromInsertion**, as follows

```
public void awakeFromInsertion(EOEditingContext context){
   super.awakeFromInsertion(context);
   roleName = "New Role";
}
```

   This method is automatically invoked right after your enterprise object class creates a new MovieRole and inserts it into an editing context, which happens when you use a display group to insert.

Running Movies

Be sure that all your project's files are saved (including your model file), and build and run your application. Now when a user clicks the Insert/New button on the MovieDetails page, a new MovieRole is inserted, with "New Role" already displayed as the role name.