
WebObjects Tutorial

[Tools](#) > [WebObjects](#)



2009-02-04



Apple Inc.
© 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Mac, Mac OS, and WebObjects are trademarks of Apple Inc., registered in the United States and other countries.

Enterprise Objects is a trademark of Apple Inc.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction Introduction 7

Who Should Read This Document? 7
Organization of This Document 7
See Also 7

Chapter 1 Installing WebObjects, Eclipse, and the WOLips Plug-in 9

Installing WebObjects 9
Installing Eclipse 9
Installing the WOLips Plug-in 9

Chapter 2 Creating a WebObjects Database Application 11

The Movies Application 11
Setting Up the Application 13
 Installing the Movies Database 13
 Creating a New WebObjects Application 14
 Linking to the Movies Database and derby.jar 15
Developing an Enterprise Object Model 15
 Creating a New EOModel 16
 Connecting the EOModel to the Movies Database 17
 Creating the Movie Entity 17
 Generating Enterprise Objects from an EOModel 19
Designing the Main Webpage 20
 Designing Main.html 20
 Configuring Main.wod 21
 Configuring Main.java 22
Adding the MovieDetails Webpage 24
 Creating a new Component 25
 Configuring MovieDetails.html and MovieDetails.wod 25
 Configuring MovieDetails.java 26
 Connecting the Main Webpage to the MovieDetails Webpage 27
 Inserting Movies into the Database 28

Chapter 3 Deploying a WebObjects Application 29

Deploying a WebObjects Application 29

Document Revision History 31

Figures and Tables

Chapter 2 **Creating a WebObjects Database Application** 11

Figure 2-1	The MovieSearch webpage	12
Figure 2-2	The MovieDetails webpage	13
Figure 2-3	The WebObjects application template	14
Figure 2-4	The Entity Modeler perspective	16
Figure 2-5	Attributes in the Movies EOModeler window	19
Figure 2-6	The main webpage with search results	24
Table 2-1	Values for entity attributes	18

Introduction

Important: This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple is supplying this information to help you adopt the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be vetted against final documentation. For information about updates to this and other developer documentation, you can check the ADC Reference Library Revision List. To receive notification of documentation updates, you can sign up for a free Apple Developer Connection Online membership and receive the bi-weekly ADC News e-mail newsletter. (See <http://developer.apple.com/products/> for more details about ADC membership.)

WebObjects is an object-oriented environment for developing and deploying web applications. This tutorial helps you learn how to install the tools necessary to develop with WebObjects, how to create a basic database web application, and how to deploy a WebObjects application on a server.

Who Should Read This Document?

WebObjects Tutorial is for anyone who is beginning to develop with WebObjects. It will introduce you to standard WebObjects development tools and processes.

This document assumes a working knowledge of Java, HTML, and databases.

Organization of This Document

This document contains the following chapters:

- [“Installing WebObjects, Eclipse, and the WOLips Plug-in”](#) (page 9) helps you install tools for WebObjects development.
- [“Creating a WebObjects Database Application”](#) (page 11) walks you through the creation of a basic web application in WebObjects.
- (Chapter forthcoming.)

See Also

The following Apple resources provide additional information about WebObjects:

- For a conceptual overview of WebObjects technology, read *WebObjects Overview*.

INTRODUCTION

Introduction

- For a more in-depth treatment of WebObjects development, read *WebObjects Web Applications Programming Guide*.
- For more information on the Enterprise Objects Framework, read *WebObjects Enterprise Objects Programming Guide*.
- For a more in-depth treatment of WebObjects application deployment, read *WebObjects Deployment Guide Using JavaMonitor*.
- For documentation of all WebObjects Java classes, see *WebObjects 5.4.2 Reference*.
- For information on components built into WebObjects, read *WebObjects Dynamic Elements Reference*.

In addition to Apple’s documentation, see the following websites for WebObjects extensions and utilities:

- For more information on the WOLips Eclipse plug-in, visit the [WOLips website](#).
- For a large collection of open-source contributions to WebObjects, visit the [Project WOnDer website](#).

Installing WebObjects, Eclipse, and the WOLips Plug-in

The WOLips plug-in for the Eclipse integrated development environment (IDE) is the recommended environment for WebObjects development. This chapter will guide you through the process of installing Eclipse and the plug-in.

Installing WebObjects

You must have the Apple developer tools installed in order to install WebObjects. Both WebObjects and the Apple developer tools are available from the Downloads page linked from the Apple Developer Connection homepage (<http://connect.apple.com>). If you already have WebObjects installed, ensure that you are using the latest update of WebObjects and read any release notes regarding the update. You do not need to download separate WebObjects updates if you are running Mac OS X Server, because WebObjects updates are included in operating system updates downloaded with Software Update. Apple developer tools updates are not included in Mac OS X Server updates, however, so you should still ensure that you have the latest version of the Apple developer tools.

Installing Eclipse

Important: The WOLips plug-in is designed to work in Eclipse version 3.3.2, Europa. If you already have Eclipse installed but it is a different version, you must install Europa and use it instead of your other version for the plug-in to function correctly.

Eclipse v3.3.2 is available for download from the Eclipse Foundation website at <http://www.eclipse.org/downloads/packages/release/europa/winter>. Download and install the Mac OS X package of the Eclipse IDE for Java Developers.

Installing the WOLips Plug-in

The simplest way to download the WOLips plug-in is from within Eclipse.

1. Open Eclipse.
2. Choose Help > Software Updates > Find and Install.
3. Select "Search for new features to install" and click Next.
4. Add the WOLips site to the list of sites to search.

- Click the New Remote Site button.
 - In the dialog, enter WOLips in the name field and `http://webobjects.mdimension.com/wolips/stable` in the URL field.
 - Click OK.
5. Select the WOLips site that appears in the site list and click Finish.
 6. Expand the WOLips item that appears in the Search Results dialog. Select Standard Install and click Next.
 7. Read and accept the terms in the license agreements and click Next.
 8. Click Finish. The plug-in will download. A Feature Verification dialog appears, noting that the plug-in is not digitally signed. Click Install All.
 9. Click Yes to restart Eclipse. The plug-in is installed and ready to use.

Note: Installing the WOLips plug-in from Eclipse always installs the latest stable version of the plug-in. This tutorial was written using WOLips plug-in version 3.3.5578, and parts of it may be inconsistent with later versions. You can download a specific version of the WOLips plug-in from <http://webobjects.mdimension.com/wolips/releases/>.

Creating a WebObjects Database Application

This tutorial introduces you to the basic concepts and procedures of developing WebObjects applications by showing you how to create a simple database application. To provide access to a database, WebObjects uses a framework called the Enterprise Objects Framework. The Enterprise Objects Framework is at the heart of every WebObjects database application, managing the interaction between the database and objects in the application. The steps you take in creating this application demonstrate the principles you use in every other application you develop with WebObjects and the Enterprise Objects Framework.

The application you create in this tutorial is called *Movies*. It uses a sample Apache Derby database, the *Movies* database, that comes with WebObjects. If you have not installed Eclipse and the WOLips plug-in, follow the steps in “[Installing WebObjects, Eclipse, and the WOLips Plug-in](#)” (page 9) before you continue with this tutorial.

In this tutorial, you will:

- Use the Entity Modeler included in the WOLips plug-in to maintain a model file
- Create custom enterprise object classes
- Create a Main component that reads from the *Movies* database
- Create a *MovieDetails* component that edits and deletes from the *Movies* database

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

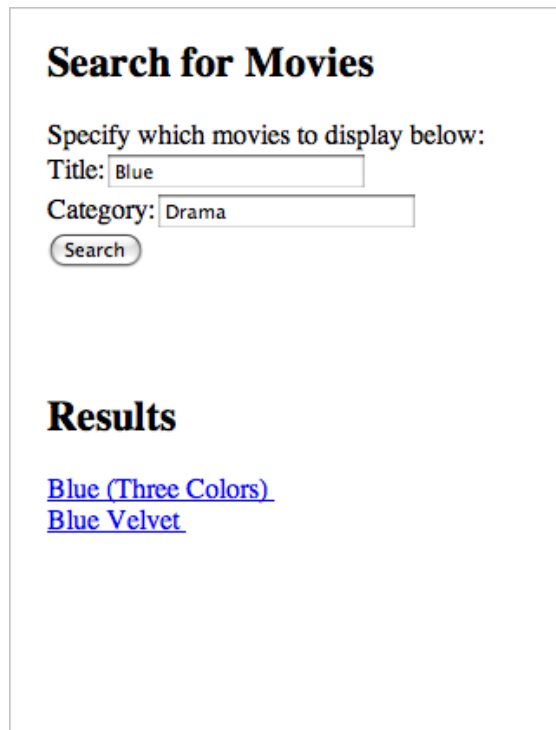
Note: You can also develop database applications using Direct to Web, a high-level technology based on WebObjects. Direct to Web instantly generates a generic database application and allows you to modify its user interface, which makes it a useful starting point for simple projects with flexible user interface requirements. See *WebObjects Direct to Web Guide* for more information.

The Movies Application

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

- *MovieSearch* (the main webpage) lets you search for movies that match user-specified criteria. For example, you can search for all comedies starting with the letter “A”. After you find the movie you are looking for, you can access its details to edit or delete it. Figure 2-1 shows the completed *MovieSearch* webpage.

Figure 2-1 The MovieSearch webpage



Search for Movies

Specify which movies to display below:

Title:

Category:

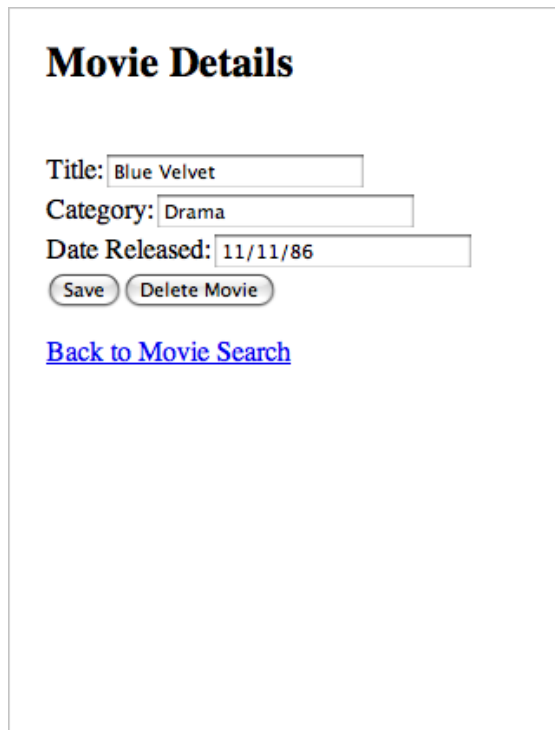
Results

[Blue \(Three Colors\)](#)

[Blue Velvet](#)

- MovieDetails displays the title, category, and release date of a movie. You can edit these properties or delete the movie from the database. Figure 2-2 shows the completed MovieDetails webpage.

Figure 2-2 The MovieDetails webpage



Movie Details

Title:

Category:

Date Released:

[Back to Movie Search](#)

Setting Up the Application

Properly setting up your WebObjects application before beginning development is critical to avoiding complications later on. This section leads you through installing the Movies database, creating a new WebObjects project, and configuring the project's connection to the database.

Installing the Movies Database

This tutorial uses a sample Apache Derby database called Movies. The Apple developer tools include a script for installing the database at `/Developer/Examples/JavaWebObjects/installDatabases.sh`. The script must be run as root and requires your user name as an argument. After you run the script, the Movies database, along with several other sample databases, is installed in `~/Library/Databases/derby-10.3.2.1/data`. For reference, the SQL commands that create and populate the databases can be found in `/Developer/Examples/JavaWebObjects/Databases`.

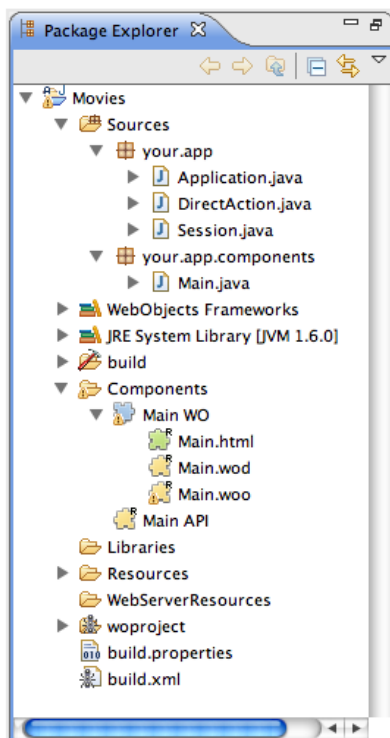
Note: If you provide a value other than your user name to the database installation script, the script still creates the full directory structure rooted at the value specified.

Creating a New WebObjects Application

1. In Eclipse, choose File > New > Project.
2. Choose WebObjects Application from the WOLips folder in the New Project dialog. Click Next.
3. Enter `Movies` as the project name in the New WebObjects Project dialog. Click Finish.

The new project appears in the Package Explorer window. Figure 2-3 shows the structure of the WebObjects application template.

Figure 2-3 The WebObjects application template



The following files are of particular importance:

- `Application.java` defines application variables that persist as long as the application does.
- `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.
- `Session.java` defines session variables that persist for the lifetime of one user's session.

- `Main.java` is a file that allows you to specify behavior associated with the Main component. A corresponding file is generated for each subsequent component.
- `Main WO` is the Main component, which comprises three files:
 - `Main.html` is the HTML template for your webpage. It can include tags for dynamic WebObjects elements as well as regular HTML.
 - `Main.wod` is the declarations file that specifies bindings between the dynamic elements and variables or methods in your Java code.
 - `Main.woo` contains information that describes `WODisplayGroup` objects and other information for build tools, such as text encodings.
- `Properties` defines Java system properties for the application.

This template builds and runs without any modifications. Control-click the Movies project folder in Package Explorer and choose Run As > WOApplication. The first time you run the application, Eclipse prompts you for the main class. Select “Application - your.app”. Your default web browser opens and displays a white page with Hello World in the upper-left corner. Click the red Terminate button located above the Eclipse console to terminate the application.

Linking to the Movies Database and derby.jar

In order for your application to find the Movies database, you must provide it with the path to the Derby home directory. Double-click the Properties file in the Resources folder in Package Explorer to open it and add the following, replacing the indicated portion with your user name:

```
derby.system.home=/Users/<REPLACE WITH YOUR  
USERNAME>/Library/Databases/derby-10.3.2.1/data
```

Java applications require a driver class to connect to a database. WebObjects provides a Derby JDBC driver in the `derby.jar` library. To reference the library, control-click your project folder in Package Explorer and choose Build Path > Add External Archives. Navigate to `/Library/WebObjects/Extensions` in the file chooser and select `derby.jar`. The library appears in the Package Explorer under Referenced Libraries. You provide the name of the driver when you configure your EOModel.

Developing an Enterprise Object Model

The Enterprise Objects Framework manages the interaction between the database and objects in a WebObjects application. Its primary responsibility is to fetch data from relational databases and represent them as 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. An EOModel represents the entity-relationship model that maps tables and rows to enterprise objects. Enterprise object classes typically correspond to database tables. An enterprise object instance corresponds to a single row or record in a database table.

- The Outline window, located in the upper left, shows the entities, attributes, and database configurations in the EOModel. A new EOModel contains only a single database configuration, named Default.
- The Properties window, located in the lower left, shows the properties for whatever element is selected in the Outline window.
- The Entity Modeler window, located on the right, shows an EOModel's list of entities or an entity's list of attributes, depending on what is selected in the Outline window.

Note: Whenever you create, open, or switch to an EOModel file, Eclipse automatically enters the Entity Modeler perspective. This perspective is necessary for EOModel development, but it is poorly suited for other tasks. Whenever you switch from an EOModel to another file type, you should change to the WOLips perspective by choosing Window > Open Perspective > Other and selecting WOLips from the dialog.

Connecting the EOModel to the Movies Database

Immediately after creating an EOModel, you should provide it with information about the database it communicates with—in this case, the Movies database:

1. Select the Default database configuration in the Outline window. The properties for the configuration will appear in the Properties window.
2. Change the value of the Name field to `Movies`.
3. Ensure that JDBC is specified as the adaptor type.
4. Leave the Username and Password fields blank—the Movies database requires neither.
5. In the URL field, enter `jdbc:derby:movies`.
6. In the Driver field, enter `org.apache.derby.jdbc.AutoLoadedDriver`. This is the Derby JDBC driver included in `derby.jar`.

The EOModel is now configured to communicate with the Movies database.

Creating the Movie Entity

Each entity in an EOModel corresponds to a table in the database. Each attribute in an entity corresponds to a column in the table. You can use entities and attributes to represent whatever portion of your database you want to interact with. For instance, the `MOVIE` table in the Movies database has nine columns, but this tutorial only uses four of them. If a column can have a null value, you do not need to create an attribute for it.

To create and configure the entity for the `MOVIE` table, do the following:

1. Control-click the Movies EOModel in the Outline window and choose New Entity. The entity's properties appear in the Properties window.
2. Change the value of the Name field to `Movie` in the Properties window.
3. Change the value of the Table Name field to `MOVIE`. This is the name of the table in the database.

4. Change the value of the Class Name field to `your.app.eo.Movie`. This is the name of the Java class that will be generated by EOGenerator.

This tutorial uses four columns in the `MOVIE` table: `MOVIE_ID` (the primary key for the table), `TITLE`, `CATEGORY`, and `DATE_RELEASED`. A separate attribute must be created for each of these columns.

To create and configure the attribute for the `MOVIE_ID` column, do the following:

1. Control-click the Movie entity in the Outline window and choose New Attribute. The attribute's properties appear in the Properties window.
2. Change the value of the Name field to `movieid`.
3. Ensure that the menu for the second field is set to `Column`. Enter `MOVIE_ID` as the value.
4. Change the value of the External Type field to `INTEGER`. This is the data type stored in the column.
5. Ensure that the Allows Null checkbox is not selected.
6. Select `Integer - Integer i` from the menu for the Data Type field. This is the Java class that corresponds to the data type in the column.

Repeat this process for the remaining three attributes. Table 2-1 indicates the values you should provide for each attribute in the Properties window.

Table 2-1 Values for entity attributes

Name	Column	External Type	Allows Null	Data Type	External Width
<code>title</code>	<code>TITLE</code>	<code>CHAR</code>	Not selected	<code>String - String S</code>	100
<code>category</code>	<code>CATEGORY</code>	<code>CHAR</code>	Selected	<code>String - String S</code>	20
<code>datereleased</code>	<code>DATE_RELEASED</code>	<code>TIMESTAMP</code>	Selected	<code>Timestamp - NSTimestamp T</code>	N/A

Note: When populating the `datereleased` attribute, leave the Server Timezone field blank. It is not used in this application.

Figure 2-5 shows the state of the EOModeler window after the remaining steps in this section are performed.

Figure 2-5 Attributes in the Movies EOModeler window

			0	Prototype	Name	Column Name	External Width	Precision	Scale
◆	◆	◆	✓		category	CATEGORY	20		
◆	◆	◆	✓		datereleased	DATE_RELEASED			
◆	◆	◆			movieid	MOVIE_ID			
◆	◆	◆			title	TITLE	100		

Most of the columns in the Entity Modeler window correspond to fields in the Properties window and are self-explanatory. The four leftmost columns are more cryptic in appearance; their purposes are the following:

- The key column is used to indicate that an attribute represents a primary key in the table.
- The diamond column is used to indicate that accessors for an attribute should be added to the Java class generated for the attribute's entity.
- The lock column is used to indicate that an attribute is well suited for detecting failures in optimistic locking. Data types of constant size are best suited for this.
- The zero column is used to indicate that an attribute can be null.

The primary key for the `MOVIE` table is `MOVIE_ID`. Click the key cell in the `movieid` row to indicate this fact. A key symbol appears in the cell. WOLips also automatically removes the diamond symbol from the row, as it is usually the case that the primary key for a table should not be exposed to the front end of an application.

Generating Enterprise Objects from an EOModel

After you have finished configuring your EOModel, WOLips can automatically generate the enterprise object classes.

1. Choose **Window > Open Perspective > Other**, choose WOLips from the dialog, and click OK to return to the WOLips perspective. Double-click the `Movies.eogen` file in the `Resources` folder in the Package Explorer window. The EOGenerator Editor appears, providing options for customizing the behavior of generating classes. No customization is necessary in this tutorial, so close the file.
2. Control-click `Movies.eogen` in the Package Explorer and choose **EOGenerate**. `Movie.java` appears in the `Sources` folder in Package Explorer. An additional file, `_Movie.java`, is created in the same folder but does not appear in the Package Explorer.

`_Movie.java` is a fully functional enterprise object that enables you to create instances representing new or existing rows in the `MOVIE` table. `Movie.java` is a subclass of `_Movie.java` that you can customize and enhance to suit your needs. You should only alter the subclass of an enterprise object class. If you ever need to modify an enterprise object class, you can see it in the Package Explorer by switching to the standard Java perspective.

Designing the Main Webpage

Every WebObjects application has at least one component—usually named `Main`—that represents the first webpage the application displays. Each component has four parts: an HTML file that defines the visual layout, a Java file that queries the database and maintains instance variables for dynamic portions of the webpage, a WOD file that binds the elements of the HTML to the logic of the Java, and a WOO file that describes `WODisplayGroup` objects and other information for build tools. The next three sections walk you through creating the HTML, Java, and WOD files for your `Main` component, which represents the `MovieSearch` webpage. The WOO file does not need to be modified in this tutorial.

Designing Main.html

A webpage consists of elements. In addition to the standard static HTML elements found in all webpages, WebObjects allows you to create dynamic elements, whose look and behavior are determined at runtime. In this step, you will create a basic interface for searching for information contained in the `Movies` database.

Double-click `Components > Main WO` in the Package Explorer to open both the HTML and WOD files for the component. The WOD file, displayed in the lower half of the main window, is blank, and the HTML file, displayed in the upper half of the main window, consists of the following template:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head>
    <title>Untitled</title>
  </head>
  <body>
Hello World
  </body>
</html>
```

Change the text within the `title` element from `Untitled` to something more descriptive, such as `Movie Search`. Replace the “Hello World” text in the `body` element with the following HTML:

```
<h2>Search for Movies</h2>
Specify which movies to display below:<br>
<webobject name="SearchForm">
  Title:<webobject name="TitleField"></webobject><br>
  Category:<webobject name="CategoryField"></webobject><br>
  <webobject name="Submit"></webobject>
</webobject>
<br><br>
<h2>Results</h2>
<webobject name = "Results">
  <webobject name = "ResultTitle"></webobject><br>
```

```
</webobject>
```

The six `webobject` elements in this block of HTML compose the search form and the portion of the webpage that will display search results. Every `webobject` element requires a `name` attribute to map it to a definition that you will add to the WOD file.

Choose **File > Save** to save the Main component. When you save, Eclipse notes six errors in the application, one for each `webobject` HTML element. The Problems window explains that the elements have not been defined in `Main.wod`.

Configuring Main.wod

A WOD file is a list of definitions that bind the `webobject` elements of a component's HTML file to variables and methods in the component's Java class. To match the `webobject` elements defined in `Main.html`, add the following to `Main.wod`:

```
SearchForm : WOForm {
    enctype = "multipart/form-data";
}

TitleField : WOTextField {
    value = title;
}

CategoryField : WOTextField {
    value = category;
}

Submit : WOSubmitButton {
    action = searchMovies;
    value = "Search";
}

Results : WORepetition {
    list = resultsList;
    item = movie;
}

ResultTitle : WOString {
    value = movie.title;
}
```

Every dynamic element definition follows the same format: the name and type of the element, separated by a colon, followed by a bracketed list of attribute assignments. Each attribute is assigned either a constant value or a value that refers to a variable or method contained in the component's Java file. For example, in the above code, `Submit` is of type `WOSubmitButton`, its attribute `value` is assigned the constant value `Search`, and its attribute `action` is assigned the value `searchMovies`, which will correspond to a method in `Main.java`.

Note: An element's type can be any of the dynamic elements included with WebObjects (see *WebObjects Dynamic Elements Reference* for descriptions of these elements) or any other component you include in your project. This nesting of components makes it easy to create modular components that can be used in several places in a single application or across multiple applications.

Save the Main component. Eclipse notes six new errors, this time because `Main.java` does not yet have corresponding variables or methods for `title`, `category`, `searchMovies`, `resultsList`, or `movie`.

Configuring Main.java

A component's Java file must define any instance variables and methods that are mentioned in the component's WOD file. In the Movies application, instance variables are defined to keep track of the user's title and category search terms, and to store the results from a search. Methods are defined to provide access to these variables and to perform the search.

The Main class uses several classes to communicate with the Movie enterprise object. Add the following import statements to `Main.java`:

```
import com.webobjects.appserver.WOComponent;
import com.webobjects.appserver.WOContext;
import com.webobjects.eoaccess.EOUtilities;
import com.webobjects.eocontrol.EOQualifier;
import com.webobjects.eocontrol.EOSortOrdering;
import com.webobjects.foundation.NSArray;
import your.app.eo.Movie;
```

Add the following instance variables to the body of the Main class:

```
private String title;           // Stores the value of the Title search
                                field.
private String category;       // Stores the value of the Category search
                                field.
private Movie movie;           // Stores the current movie when looping
                                through search results.
private NSArray<Movie> resultsList; // Stores the results of a search.
```

Each instance variable in a component's Java file that contributes to HTML content requires standard setter and getter methods in order for your application to be able to access it. Each setter method should have a name of the form `set<Variablename>`, and each getter method should be the name of the instance variable. All of the instance variables in `Main.java` contribute to HTML content, so create setter and getter methods for all of them. All of the setter and getter methods in `Main.java` should consist of simple assignment statements and return statements, respectively.

Finally, `Main.java` needs to define the `searchMovies` method that is called when the Search button is clicked. Add the following method to the Main class:

```
public WOComponent searchMovies() {
    if (title == null && category == null) {
        return context().page();
    }
    if (title == null) {
        title = "";
    }
}
```

```

    if (category == null) {
        category = "";
    }
    NSArray<EOSortOrdering> sortOrdering = new NSArray<EOSortOrdering>(new
EOSortOrdering("title", EOSortOrdering.CompareAscending));

    String[] argArray = new String[2];
    argArray[0] = title + "*";
    argArray[1] = category + "*";

    NSArray<String> args = new NSArray<String>(argArray);
    EOQualifier qualifier = EOQualifier.qualifierWithQualifierFormat("title LIKE
%@ AND category LIKE %@", args);
    resultsList = Movie.fetchMovies(session().defaultEditingContext(), qualifier,
sortOrdering);
    return context().page();
}

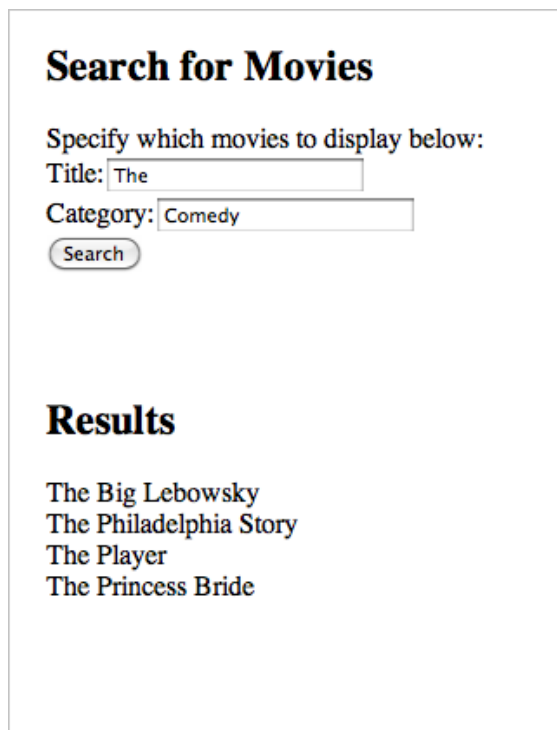
```

The first `if` statement in the method ensures that a value has been supplied for at least one search field, and reloads the current webpage without querying the database if this is not the case. The two subsequent `if` statements provide the empty string for any field the user chooses to omit. The remaining lines perform the following actions:

1. A sort ordering is created to indicate that results retrieved from the database should be sorted in ascending order by title.
2. An array of arguments for a formatting string is created. An asterisk is added to the end of both the `title` and `category` strings to indicate a wildcard suffix in a SQL LIKE clause.
3. A qualifier is created to specify the movies to be returned from the database query. The string passed into `EOQualifier.qualifierWithQualifierFormat` is the equivalent of a WHERE clause in a SQL statement that begins with `SELECT * FROM MOVIE`. The `%@` substring indicates a placeholder to be filled by an object in `args`.
4. The static method `Movie.fetchMovies` populates `resultsList` with the movies that meet the requirements set by `qualifier`, in the order specified by `sortOrdering`.
5. The response webpage is returned, in this case the same webpage. The Results section of the page is now populated with the titles of movies in `resultsList`.

Save your project and run it. Performing a search with `The` in the Title field and `Comedy` in the Category field produces the webpage shown in Figure 2-6.

Figure 2-6 The main webpage with search results



Search for Movies

Specify which movies to display below:

Title:

Category:

Results

The Big Lebowski
The Philadelphia Story
The Player
The Princess Bride

Note: The application in this tutorial performs very few error checks in order to maintain focus on WebObjects concepts. Make sure to handle all possible exceptions when writing your future WebObjects applications.

Adding the MovieDetails Webpage

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

- Create a new component.
- Assign a movie selected on the Main webpage to a variable in the MovieDetails webpage.
- Create an interface for editing or deleting a movie's information.
- Create a way to navigate from Main to MovieDetails and back.

Following this section, you will possess all of the fundamental skills necessary to create a dynamic multipage WebObjects database application.

Creating a new Component

To create a the MovieDetails component, control-click the Components folder in the Package Explorer and choose **New > WOComponent**. In the **New WebObjects Component** dialog, change the name of the component to `MovieDetails` and click **Finish**. The new component appears in the Components folder below `Main WO`.

Configuring MovieDetails.html and MovieDetails.wod

The `MovieDetails` webpage is very similar to the `Main` webpage in structure and appearance, consisting of an input form with two submit buttons: a `Save` button and a `Delete Movie` button. Consequently, the HTML and WOD files for the `MovieDetails` webpage are nearly identical to their `Main` webpage counterparts.

Add the following HTML to `MovieDetails.html`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head>
    <title>Movie Details</title>
  </head>
  <body>
    <h2>Movie Details</h2>
    <webobject name="Message"></webobject><br>
    <webobject name="EditForm">
      Title:<webobject name="TitleField"></webobject><br>
      Category:<webobject name="CategoryField"></webobject><br>
      Date Released:<webobject name="DateField"></webobject><br>
      <webobject name="Save"></webobject><webobject
name="Delete"></webobject>
    </webobject>
    <webobject name="LinkToMain">Back to Movie Search</webobject>
  </body>
</html>
```

Add the following dynamic element definitions to `MovieDetails.wod`:

```
Message : WOString {
  value = message;
}

EditForm : WOForm {
  enctype = "multipart/form-data";
  multipleSubmit = true;
}

TitleField : WOTextField {
  value = title;
}

CategoryField : WOTextField {
  value = category;
}

DateField : WOTextField {
  value = datereleased;
}
```

```

Save : WOSubmitButton {
    action = saveMovie;
    value = "Save";
}

Delete : WOSubmitButton {
    action = deleteMovie;
    value = "Delete Movie";
}

LinkToMain : WOLink {
    pageName = "Main";
}

```

Setting the `multipleSubmit` attribute of a `WOForm` element to `true` allows more than one submit button to be associated with that form. In this case, the Save and Delete Movie buttons are associated with the same form.

Configuring MovieDetails.java

Whereas the Java logic in the Main component deals with reading from the Movies database, the `MovieDetails` component deals with modifying the database. Users can change the title, category, and release date of a movie, or delete the movie from the database entirely.

Open `MovieDetails.java`. If Eclipse displays an error in the file, make sure that the `MovieDetails` class extends from the correct class, namely `WOComponent`.

Add the following import statements to `MovieDetails.java`:

```

import com.webobjects.foundation.NSTimestamp;
import java.text.SimpleDateFormat;
import java.text.ParseException;
import java.util.Date;
import your.app.eo.Movie;

```

The `MovieDetails` class requires the following instance variables:

```

private Movie movie;           // The movie for which the details are
displayed.
private SimpleDateFormat formatter; // A utility class for converting between
date formats.
private String category;      // The value of the Category text field.
private String datereleased;  // The value of the Date Released text
field.
private String message;      // The value of a message that indicates
operation success or errors.
private String title;        // The value of the Title text field.

```

Create setter and getter accessor methods for each variable as you did in `Main.java`. The `formatter` instance variable does not directly contribute to HTML content, so it does not require setter or getter methods. The `MovieDetails` class populates the `category`, `datereleased`, and `title` instance variables with values taken from `movie`. As a result, `setMovie` requires slightly more logic than a simple assignment statement.

```

public void setMovie (Movie m) {

```

```

        movie = m;
        setTitle(movie.title().trim());
        setCategory(movie.category().trim());
        setDateReleased(formatter.format(movie.dateReleased()));
    }

```

Two instance variables, `formatter` and `message`, need to be instantiated in the constructor of `MovieDetails`. Add the following two lines to the constructor:

```

formatter = new SimpleDateFormat("M/d/y");
message = "";

```

The `MovieDetails` class requires two additional methods, one for each submit button. The first, `saveMovie`, alters the current movie's row in the database to reflect changes made by the user:

```

public WComponent saveMovie() {
    if (movie.title() == null) {
        setMessage("Error: Title is required.");
        return context().page();
    }

    Date newDate = null;
    try {
        newDate = formatter.parse(dateReleased);
    } catch (ParseException e) {
        setMessage("Error: Date must be in mm/dd/yy format.");
        return context().page();
    }

    movie.setTitle(title);
    movie.setCategory(category);
    movie.setDateReleased(new NSTimestamp(newDate));
    session().defaultEditingContext().saveChanges();
    setMessage("Movie Saved.");
    return context().page();
}

```

Calling `saveChanges` on the default editing context will save any modifications of enterprise objects to the database. This includes alterations to a row, inserting a row, and deleting a row.

The second method, `deleteMovie`, deletes the movie from the database:

```

public WComponent deleteMovie() {
    session().defaultEditingContext().deleteObject(movie);
    session().defaultEditingContext().saveChanges();
    return pageWithName("Main");
}

```

Because deleting a movie should prevent the user from subsequently editing it, `deleteMovie` returns the user to the Main webpage instead of reloading the `MovieDetails` webpage.

Connecting the Main Webpage to the MovieDetails Webpage

So far, there is no way to reach the `MovieDetails` webpage from the Main webpage, because the search results produced by the Main webpage are in plaintext. Each result should instead be a hyperlink to a corresponding `MovieDetails` webpage. This requires small changes to the HTML and WOD files of the Main component.

In `Main.html`, change the line that reads:

```
<webobject name = "ResultTitle"></webobject><br>
```

To the following:

```
<webobject name = "MovieLink"><webobject name =
"ResultTitle"></webobject></webobject><br>
```

Add the following dynamic element definition to `Main.wod`:

```
MovieLink : WOHyperlink {
    action = showDetails;
}
```

Finally, add the `showDetails` method to `Main.java`:

```
public WComponent showDetails() {
    MovieDetails detailsPage = (MovieDetails)pageWithName("MovieDetails");
    detailsPage.setMovie(movie);
    return detailsPage;
}
```

Inserting Movies into the Database

Database applications that support deleting from a database most likely support inserting into the database as well. The steps for creating this functionality are essentially identical to the steps you took to create the search functionality.

1. Add necessary `webobject` elements to `Main.html` to create a form that allows the user to specify a title, category, and release date.
2. Add corresponding element definitions to `Main.wod`.
3. Add necessary instance variables to `Main.java`, along with a method for adding a new movie. The main logic of the method looks like the following:

```
public WComponent insertMovie() {
    Movie newMovie = Movie.createMovie(session().defaultEditingContext(),
newtitle);
    newMovie.setCategory(newcategory);
    newMovie.setDateReleased(formatter.format(newdatereleased));
    session().defaultEditingContext().saveChanges();
    return context().page();
}
```

For improved user experience, add error checking and message updates to the method.

Deploying a WebObjects Application

This chapter will guide you through the basics of deploying the Movies application on a Mac running Mac OS X Server. For further information on WebObjects application deployment, read *WebObjects Deployment Guide Using JavaMonitor*.

Deploying a WebObjects Application

Chapter is forthcoming.

Document Revision History

This table describes the changes to *WebObjects Tutorial*.

Date	Notes
2009-02-04	New document that describes the process for building a simple WebObjects application using Eclipse and the WOLips plug-in.

REVISION HISTORY

Document Revision History