Creating a Web-Based Database Application With WebObjects Builder

WebObjects uses Enterprise Objects Framework to create web-based database applications. This chapter introduces you to the Enterprise Objects Framework by showing you how to create a simple database application with WebObjects Builder. The steps you take in creating this application demonstrate the principles you'll use in every other application you develop with WebObjects Builder and Enterprise Objects Framework.

Before you go on, you should be familiar with WebObjects Builder. If you haven't already, read "Creating a Guest Book Using WebObjects Builder." It introduces many basic concepts and procedures that you should know before doing this tutorial. Also, the application you'll create in this chapter, Movies, is based on the Movies sample database distributed with WebObjects. You must have the sample database installed to do this tutorial. Read the "Enterprise Objects Framework Example Guide" for installation instructions.

The application has three pages: "Browse Movies," "Movie Search," and "Movie Details." The Browse Movies and Movie Search pages of the Movies application let you browse the movies that are in a database and search for movies that match user-specified criteria. For example, you can search for all comedies starting with the letter "T" that have an R rating. You can also use these pages to insert, update, and delete movie records.

The Browse Movies page is the first page of the Movies application. When you start the application, it opens this page. Use the Next Page and Previous Page buttons to page through all the movies in a database.

The Movie Search page provides an interface for searching the database for movies that match user-specified criteria. Type the first part of the strings you want to match and click Match.

Movies also has a Movie Details page that displays the actors who star in a selected movie and the roles that the actors play. From this page, you can add new roles to a movie, change the name of a role, and assign a different actor to a role.

The Movie Details page displays the roles and the actors who play them for the selected movie. You select a movie in either the Browse Movies or Movie Search page. Use the Movie Details page to insert, update, and delete movie roles for the selected movie.

In this tutorial, you'll learn the basic things you must do to create an Enterprise Objects Framework application for the web. You'll discover how to use WebObjects Builder to create web pages that access a database and how to work with models in EOModeler.

Enterprise Objects and the Movies Database

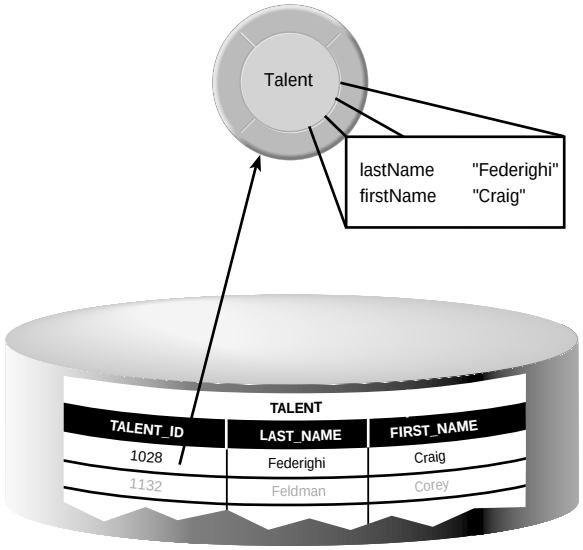
Enterprise Objects Framework manages the interaction between the database and objects in the Movies application. Its primary responsibility is to fetch data from relational databases into *enterprise objects*. The Movies application centers around three enterprise objects: Movie, MovieRole, and Talent. A movie has many roles; and actors, or *talent*, star in those roles.

An enterprise object is like any other object in that it couples data with methods for operating on that data. However, an enterprise object class has certain characteristics that distinguish it from other classes:

- It has properties that map to stored data; an enterprise object instance typically corresponds to a single row or record in a database.
- It knows how to interact with WebObjects Framework and Enterprise Objects Framework to give and receive values for its properties.

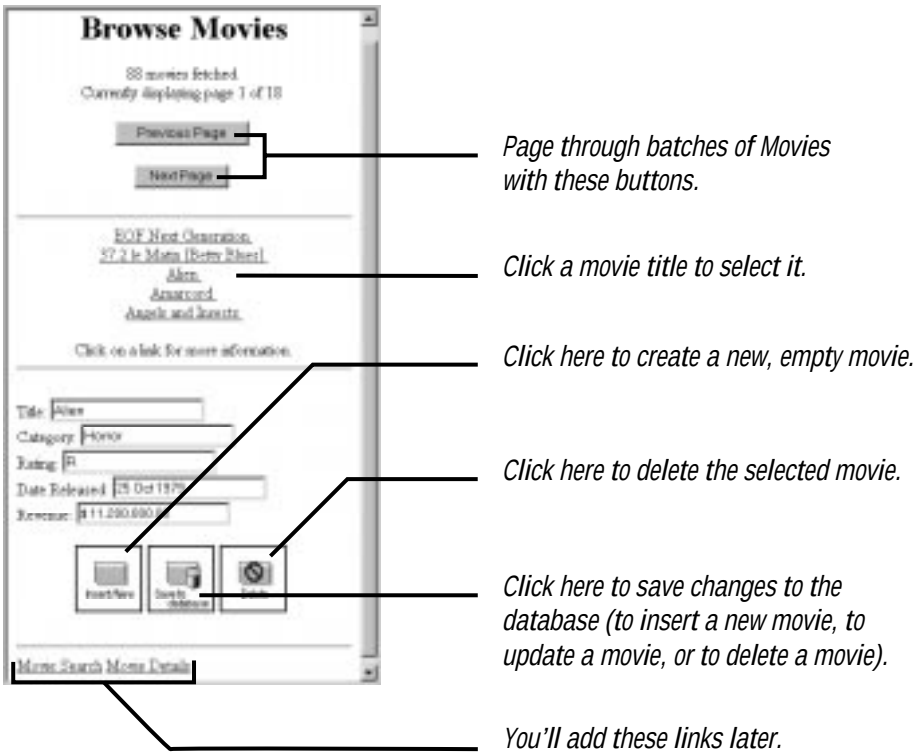
The ingredients that make up an enterprise object are its class definition and the data values from the database row or record with which the object is instantiated.

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, or class properties. (Instance variables based on database data are called class properties.) In an application, Talent objects are instantiated using the data from a corresponding database row, as shown in the following figure:



Designing the Main Page with the Database Wizard

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 Browse Movies page shown below:



In this section, you’ll use the WebObjects Builder Database Wizard to design the Main page. The Database Wizard performs all the setup that’s necessary to fetch database records and display them in a web page. Specifying different wizard options yields completely different pages. For example, both the Browse Movies and Movie Search pages are designed with the Database Wizard.

1. Create a New Application.

   - Use WebObjects Builder to create a new application named “Movies”.

   Recall that you must create the application in a directory that’s under *DocumentRoot*/WebObjects.

2. Activate the Database Wizard.

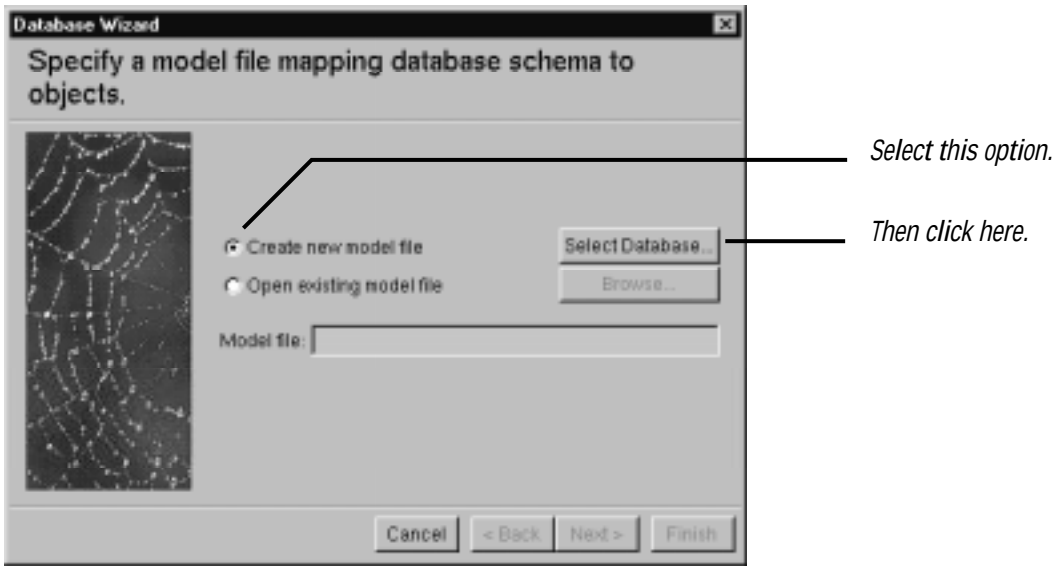
   - Bring the Main component to the front.
   - Choose Tools->Database Wizard.

   The Database Wizard panel opens.

3. Specify a model file.

A *model file* defines a database-to-object mapping that associates database columns with instance variables of objects. It also specifies object relationships based on database join criteria. You typically create model files using the EOModeler application, but the Database Wizard can create a default model as a starting point. Later on you'll use EOModeler to modify the default model created by the wizard.

- In the Database Wizard panel, select Create a new model file.
- Click Select Database.



A Choose Database Adaptor panel opens.

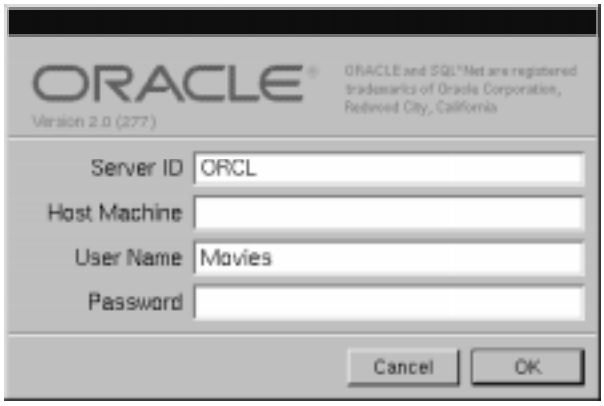
An *adaptor* is a mechanism that connects your application to a particular database server. For each type of server you use, you need a separate adaptor. WebObjects provides adaptors for Informix, Oracle, and Sybase servers, and for any server that is ODBC compliant.

Note: Not all adaptors are available with all versions of WebObjects.

- Choose the adaptor for the database you want to use.

A login panel for the database that corresponds to the selected adaptor opens. The examples in this chapter use the Oracle version of the Movies database included with WebObjects.

- Fill in the login panel.



Different databases require different login information, so each database's login panel looks different. If you don't know how to fill in the login panel, check with your database administrator.

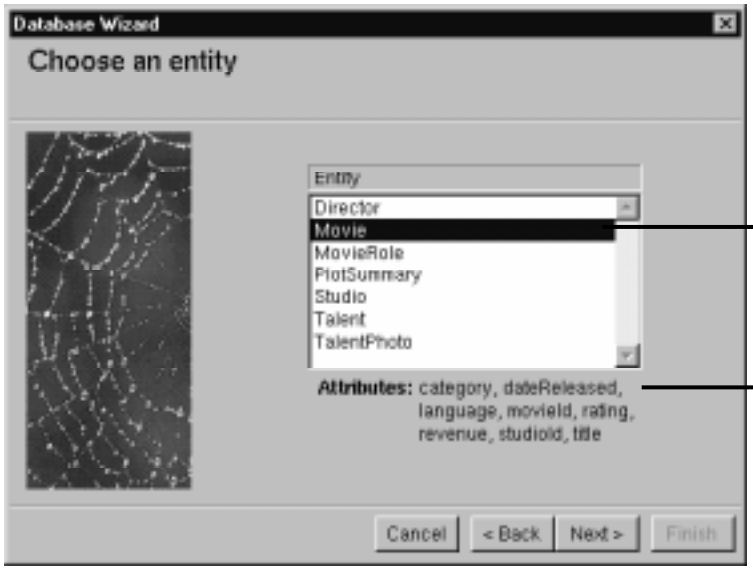
When you use the Database Wizard to create a model file, the wizard uses a database adaptor to read the data dictionary from the database you specified in the login panel and to create a default model.

4. Choose an entity.

An *entity* is a part of the database-to-object mapping that associates a database table with a class (or type) of object. For example, the Movie entity maps rows from the MOVIE table to Movie objects.

- In the Choose an entity panel, select the Movie entity.

When you choose the Movie entity, the wizard displays its attributes: category, dateReleased, language, movieId, rating, revenue, studioId, and title. An *attribute* is a part of the database-to-object mapping that associates a database column with one of a class's instance variables. 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.



5. Choose a primary key.

In a relational database, each table has a column or combination of columns whose values are guaranteed to uniquely identify each row in that table. For example, in the Movies database the MOVIE table has as its primary key the column MOVIE_ID. 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 primary keys.

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 Database Wizard set the Movie entity's primary key to the attribute you choose; however, you'll need to explicitly assign primary keys to the other entities later on.

- Select movieId.

You should choose primary key attributes that are assigned initial values and then are never modified. Consequently applications usually assign primary key values automatically. 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.

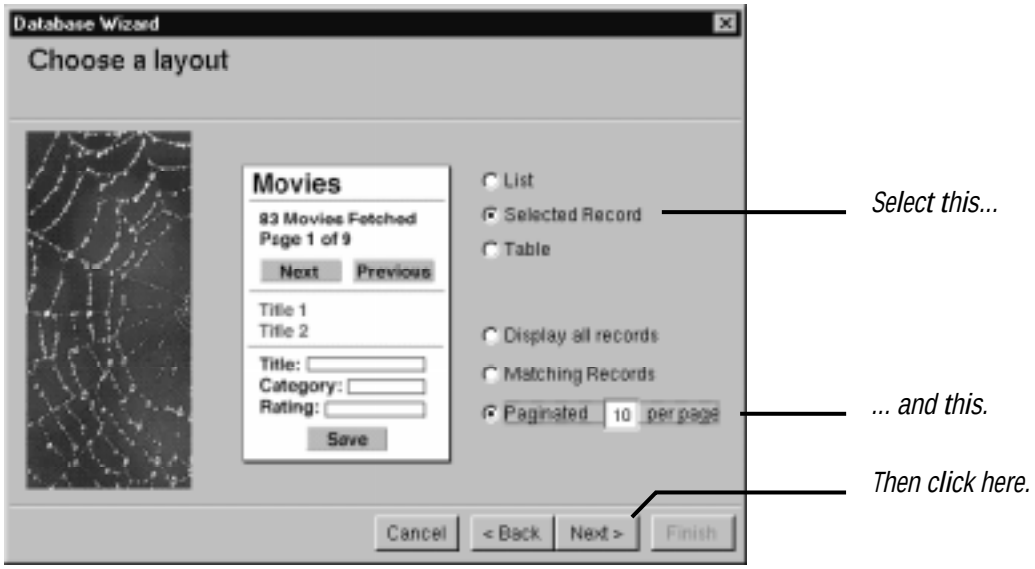
Enterprise Objects Framework provides several mechanisms for generating and assigning unique values to primary key attributes. By default, Enterprise Objects Framework uses a native database mechanism to assign primary key values. The scripts that come with WebObjects for installing the Movies database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

6. Choose a layout.

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

- Choose Selected Record.
- Choose Paginated 10 per page.

Based on your specifications, the wizard shows you a preview of the page it will generate.



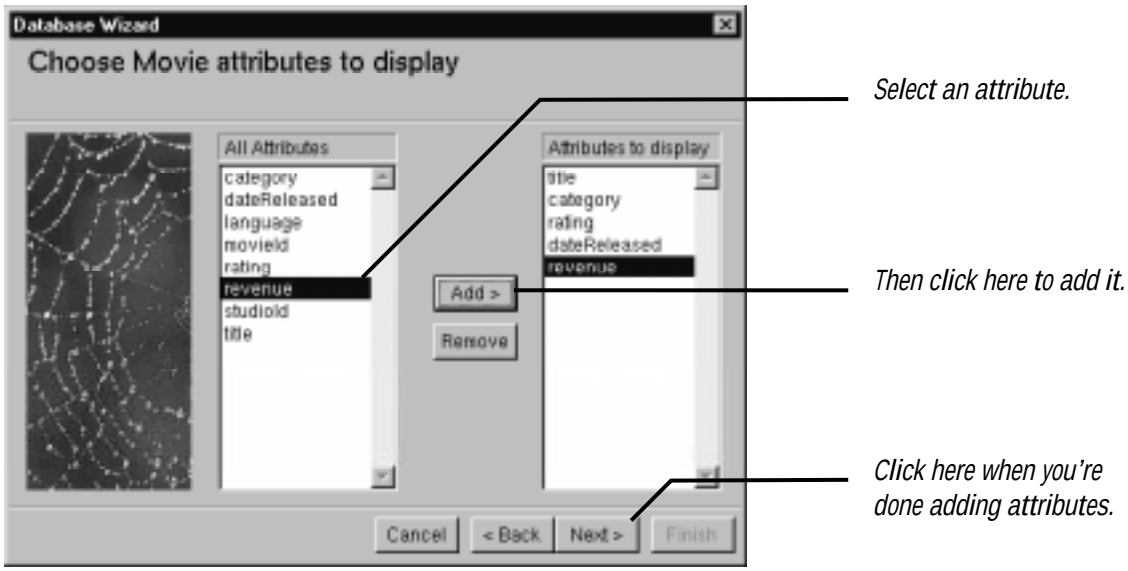
7. Choose attributes to display.

The Database Wizard provides a list of all the Movie attributes.

- Add attributes from the All Attributes column to the Attributes to display column.

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, rating, dateReleased, and revenue.

Don't add any of the remaining attributes. The language, movieId, and studioId attributes don't have meaning to users, and should not be displayed in the page.



8. Choose the attribute to display on the hyperlink.

When you choose the Selected Record layout, the pages generated by the Database Wizard contain a list of hyperlinks—one for each record fetched. Clicking a hyperlink selects the corresponding record in a detail portion of the page so the record can be edited. You choose the attribute that will be shown on the hyperlinks.

- Select the **title** attribute.
- Click Finish.

The Database Wizard designs a page for browsing Movie objects that are fetched from a database. It lays out elements on the page, creates variables, and binds the variables to the page's dynamic elements. Take a minute to examine the bindings the wizard creates.

Later in this tutorial, you'll build a page by hand that contains some of the features as this Main page. At that time, you'll perform by hand many of the procedures that the Database Wizard performs for you automatically.

9. Save the page.

- Bring the Main component to the front.
- Choose File->Save.

> IMPORTANT: At this point, you need to quit and restart WebObjects Builder to avoid display problems that can occur after the use of an NSSecureTextField (such as password fields in database login panels). See the WebObjects Release Notes for more information.

10. Run the application.

- Use a web browser to run the application.

See the "Run the application" section in "Creating a Guest Book Using WebObjects Builder" for assistance.

The Database Wizard produces a functional web-based database application. Try browsing through the batches of movies using the Next Page and Previous Page buttons.

Refining the Main Page

When you run your application, you may notice that the dateReleased values are unformatted. Also, if you try to edit a movie or to create a new one, the application returns an error page. At this point, saving changes to the database doesn't work.

To enable inserting and updating, you need to add a date format for the dateReleased text field element. The dateReleased attribute is stored as an NSCalendarDate object in Movie objects, but it is represented as an NSString object in the text field. Assigning a date format alerts the Movies application that it needs to translate between NSCalendarDates and NSStrings. For more information, see the NSCalendarDate and NSString class specifications in the *Foundation Reference*.

Assigning a date format also allows you to display dates in a more customary format. For example, you can transform a date from its default format (1981-12-27 00:00:00 -0700, for example) to any of the following:

- 12/27/81
- December 21, 1981
- 21 Dec 1981

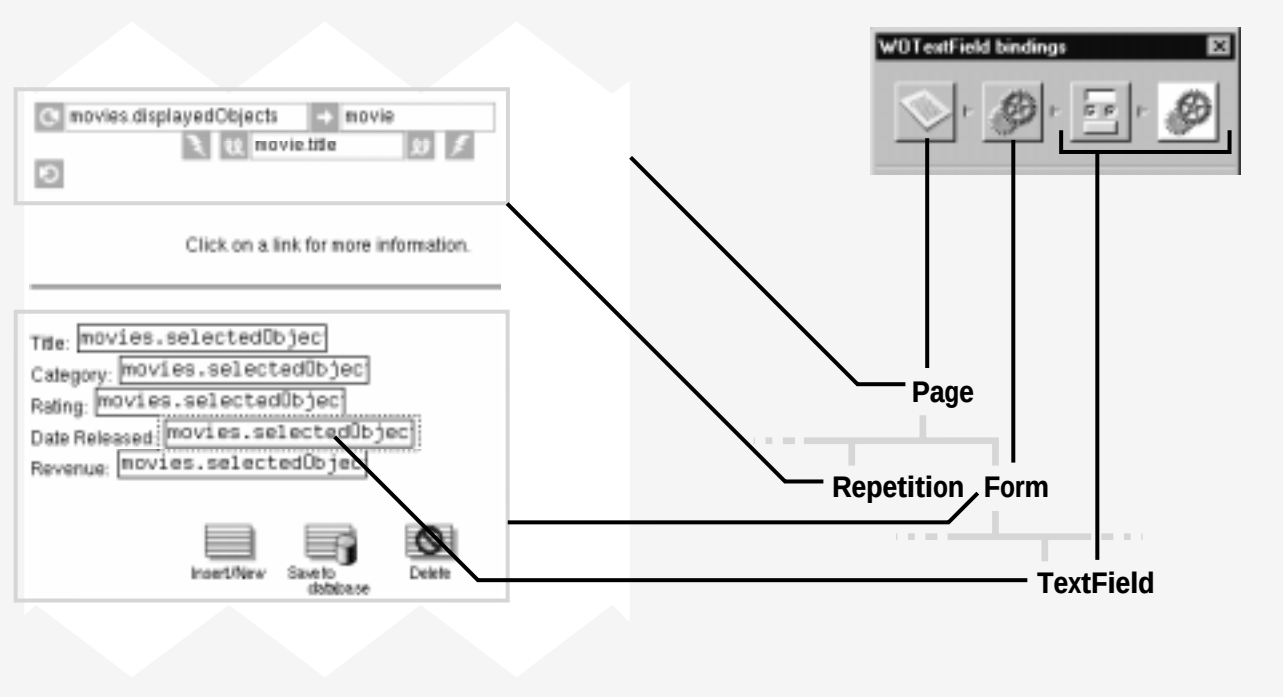
Similarly, you can specify a number format for the revenue text field element. For example, you can add a dollar sign and thousands separators.

In this section, you'll add date and number formats. In addition, you'll configure the Movies application to sort the movies it displays. You should also consider changing the labels of your text fields (for example, change "title" to "Title").

1. Inspect the dateReleased text field's bindings.

    - Select the dateReleased text field element in the Main component window.
    - Click the "i" button.

The inspector that opens displays bindings, attributes, and other settings for elements in a component. Each element occupies a place in the component's HTML hierarchy. For example, a text field element is contained in a form element, which is contained in a page. A page is the highest level in the hierarchy. As shown below, you can use the inspector to traverse the hierarchy from the selected element on up.

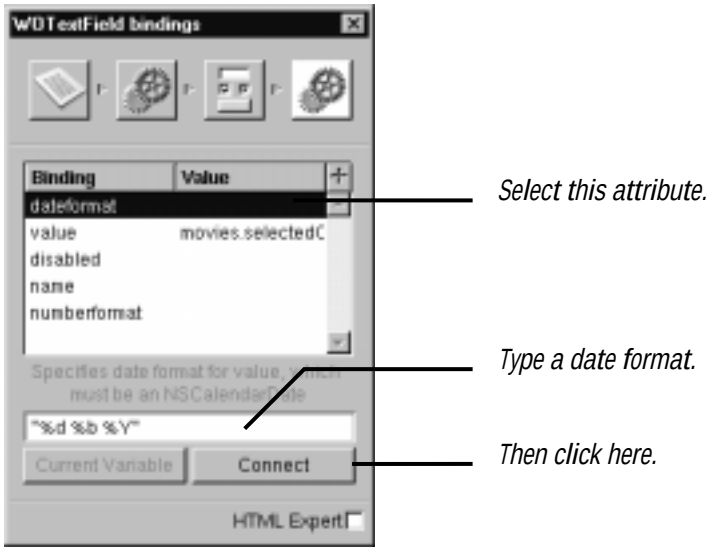


The dateReleased text field element is selected, so the inspector show icons for the text field its parent elements—the surrounding form and the page. Form and abstract elements (and all instances of WODynamicElement subclasses) can have as many as two icons in the inspector's icon path: one for inspecting the HTML attributes of the element and a gears icon for inspecting the element's bindings.

- In the inspector, click the gears icon to inspect the text fields bindings.

2. Add a date format.

- Select the dateformat attribute.
- Type (including the quotes): "%d %b %Y".
- Click Connect.



Using this date format ("%d %b %Y"), the Movies application formats the date "1995-12-3 00:00:00 -0700" as "3 Dec 1995". The %d conversion specifier stands for day of the month, %b stands for the 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 NSCalendarDate class specification in the *Foundation Reference*.

3. Add a number format.

- Inspect the revenue text field bindings.
- Bind the text field's numberformat attribute to the string (including the quotes): "$ #,##0.00".

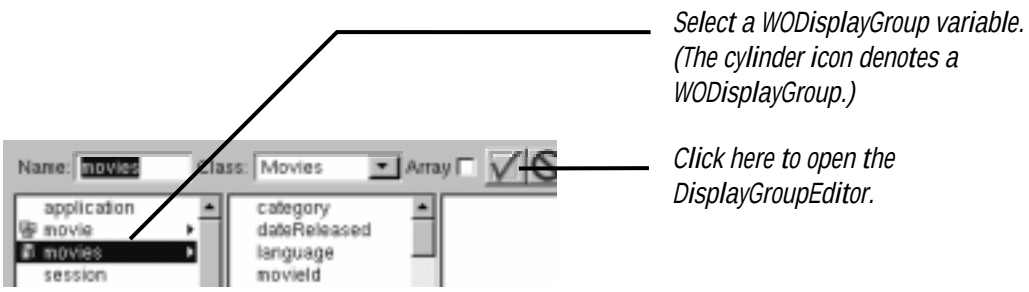
Using this number format ("$ #,##0.00"), the Movies application formats the number 1750000 as "$ 1,750,000". For more information on creating number formats, see the NSNumberFormatter class specification available through NeXTanswers as NeXTanswer 2481.

4. Specify a sort order.

As one of its tasks, the Database Wizard created the component variable movies—a WODisplayGroup object. WODisplayGroups (also referred to as *display groups*) provide a simple interface for interacting with relational databases in terms of objects. Display groups are discussed in more detail in the following sections.

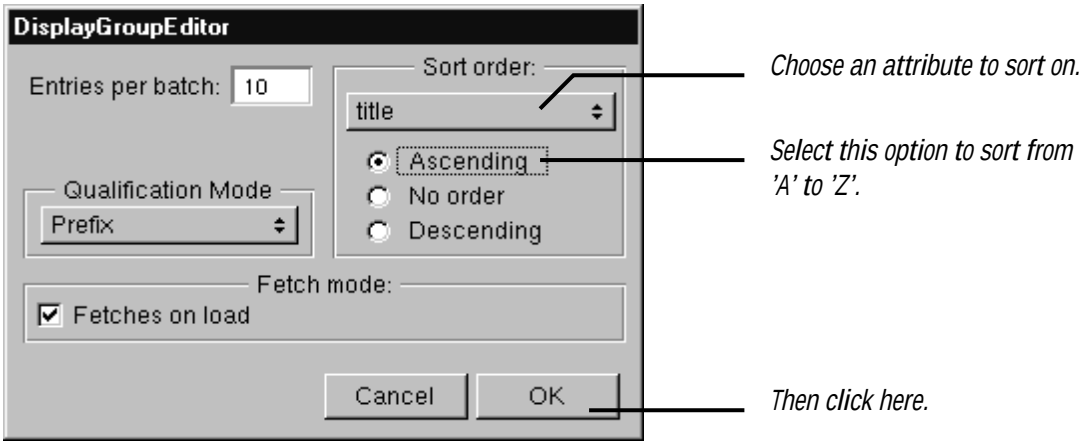
- Select the movies variable in the Main component's object browser.
- Click the checkmark button.

The checkmark button is only enabled when a WODisplayGroup object is selected in the object browser.



In the DisplayGroupEditor panel that opens, you can configure the movies display group to sort the Movie objects it fetches from the database.

- Select the title attribute in the Sort order pop-up list.
- Select Ascending sort order.



5. Save and run your application.

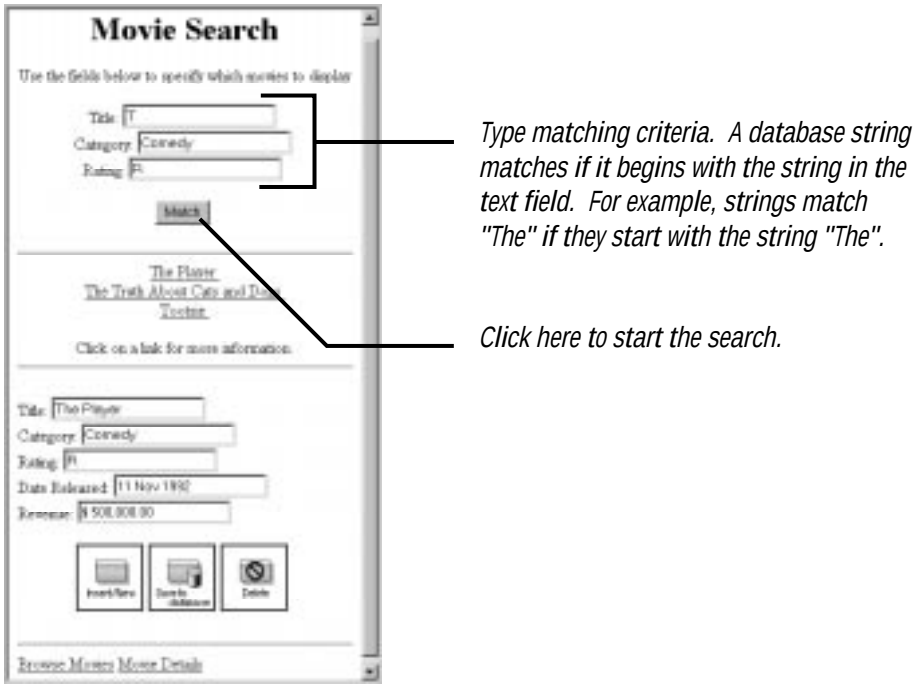
Start a new session with your Movies application by opening the URL you used in the last section. This time, the movies are sorted and the dateReleased and revenue values are formatted.

Try inserting and updating movies using dates and numbers in different formats. In particular, date formatters are very flexible. Try entering the following dateReleased values for a selected movie:

- 12/3/95
- 95-12-3
- December 3, 1995
- today

Adding a MovieSearch Page

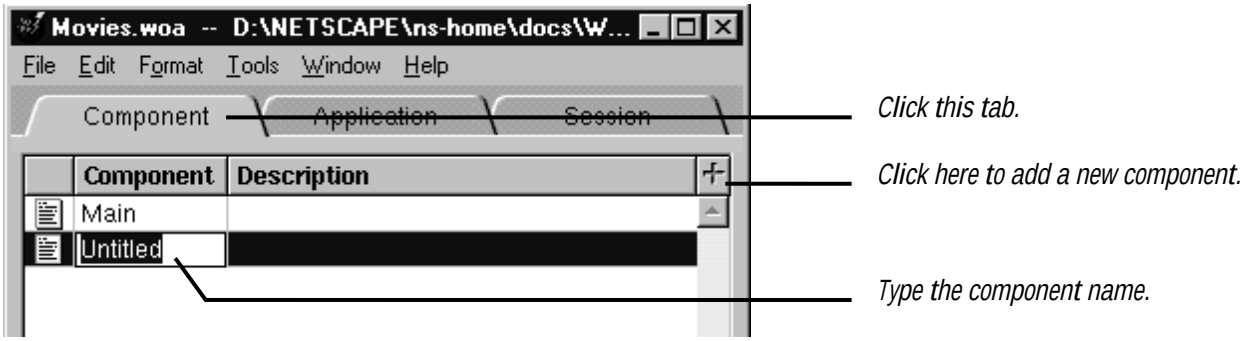
The Database Wizard can also set up pages for searching the database. In this section, you'll create a page that allows you to search for movies that match criteria you specify. For example, you can search for all comedies that start with the letter "T" and are rated R. After specifying matching criteria, click Match. The Movies application lists the movies that match your criteria.



You can insert, update, and delete movies with this page the same way you do in Main. Not surprisingly, the steps for creating the search page are very similar to those for creating Main.

1. Add a component.

   - Bring the Movies application window to the front.
   - Click the Component tab.
   - Click the plus button.
   - Name the component "MovieSearch".



2. Run the Database Wizard.

Activate the wizard from the MovieSearch component window, and perform the same steps you did for the Main component, except this time you'll:

- Use the model you created for the Main component instead of creating a new model.
- Choose the Matching Records layout option instead of Paginated 10 per page.
- Choose attributes on which to match in a window that only opens when you choose the Matching Records layout.

When you use the Database Wizard to create a new model file as you did for the Main page, the wizard saves the model in your application directory. When you run the Database Wizard this time, you can specify the model file in the Movies application directory.

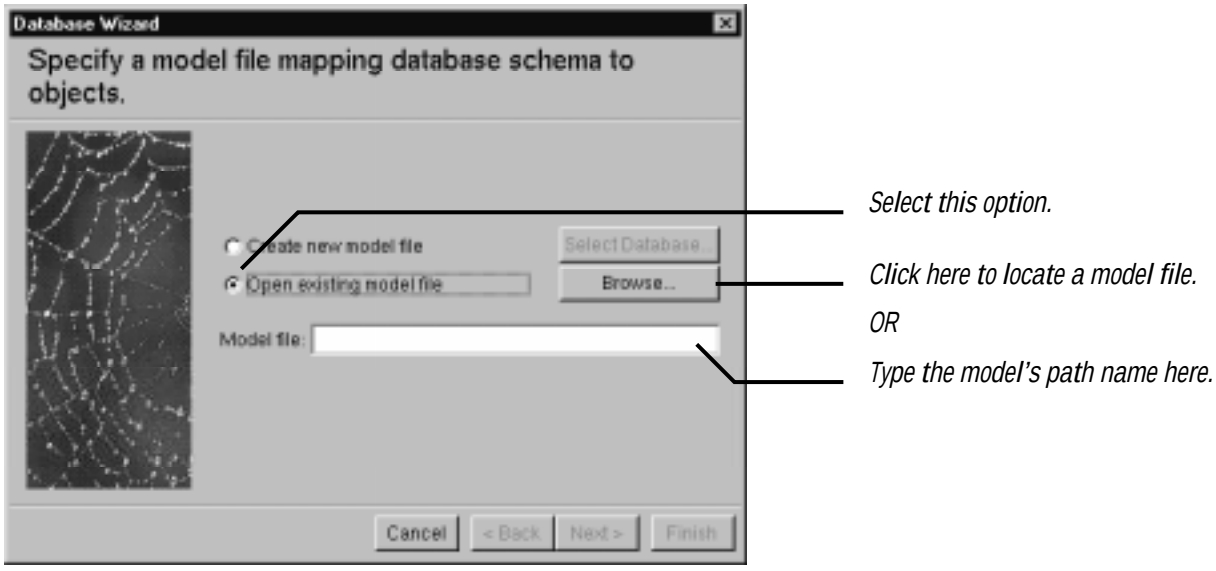
▸ In the Database Wizard's Specify a model file panel, click Browse to locate the model file.

▸ In the Open panel, navigate to the Movies.woa directory.

A model *file* is actually a directory, or *file package*, that contains a collection of files that shouldn't be separated. A model's file package has the extension .eomodeld (for eomodel *directory*).

▸ Select the directory with the extension .eomodeld.

For Mach Users Only: If you are doing the tutorial on a Mach system, you need to follow a slightly different procedure. WebObjects application directories such as Movies.woa are also file packages. Ordinarily you can't see the contents of a .woa directory in the Workspace File Viewer, in Open panels, or in Save panels. To select the model file in the Database Wizard's open panel, bring the Workspace File Viewer to the front. Navigate to your Movies.woa application directory, and choose File->Open as Folder. The Workspace opens a new viewer window that's focused on the contents of the Movie.woa directory. Select the model file package, and drag it into the Database Wizard's Open panel.



- In the Choose an entity panel, select the Movie entity.

Model files store database login and primary key information, so when you run the Database Wizard this time it doesn't prompt you with a Choose Adaptor panel, a login panel, or a Pick a Primary Key panel.

- Choose the Selected Record and Matching Records layout options.

As you can see in the Choose a layout panel's page preview, the Matching Records option generates a database search form in which users can specify record-matching criteria.

- Choose the same attributes to display: title, category, rating, dateReleased, and revenue.

- Choose the title attribute to display on the hyperlink.

When you choose the Matching Records layout option, you must tell the Database Wizard what attributes to put in the query form.

- Select title, category, and rating as the attributes on which to match.

As before, the Database Wizard designs a functional page, but this one allows you to search the database for a movie that matches criteria you specify.

3. Add date and number formats.

- Add a date format to the dateReleased text field and a number format to the revenue text field.

In the Main component, you used "%d %b %Y" and "$ #,##0.00".

4. Change the fetch mode.

- Open the DisplayGroupEditor for the MovieSearch component's movies variable.

- Uncheck the Fetches on load box.

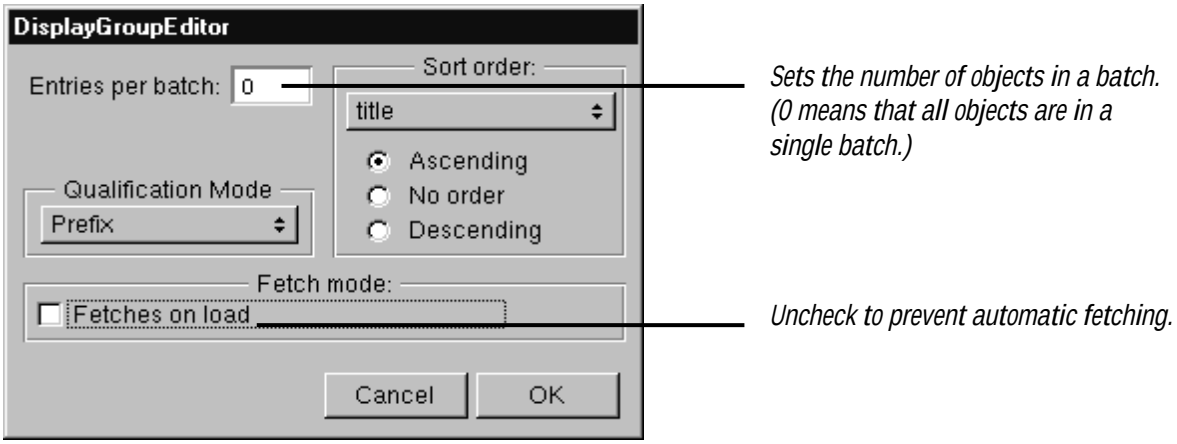
When the Fetches on load box is checked, the application fetches Movie records from the database when the component is loaded into the application. In the Main page, Fetches on load is enabled. As a result, when you run the application, the Main page opens displaying a batch of movie records.

If you disable Fetches on load, you must explicitly tell the application to fetch movie records from the database. This is the behavior you want to for the MovieSearch component. When the page opens, the application shouldn't display any movies. Instead, the user initiates the search by clicking Match.

Notice that Entries per batch is set to 0. This means that the application won't display search results in batches. Instead, all the movies that match a user's search criteria are displayed at once. Contrast this with the Main component. In Main, Entries per batch is set to 10 and you use the Next and Previous buttons to page through the results.

The Database Wizard automatically configures a display group's entries per batch based on the layout options. For the Paginated... option, the wizard sets the entries per batch to the number of records you specify (10 is the default). Whereas for the Display all records and Matching Records options, it sets the entries per batch to 0.

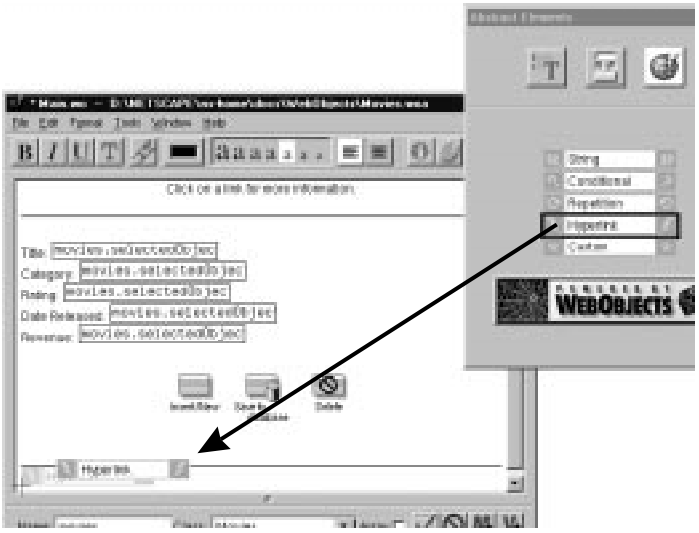
- Set the sort order as you did for the Main component.



5. Add a hyperlink to the Main component.

Now your application needs a way for users to get to the MovieSearch page from the application's Main page. To provide it, add a hyperlink to the Main component that opens the MovieSearch page.

- In the Main component window, place the cursor at the bottom of the page below the horizontal line.
- Drag a Hyperlink from the Abstract Elements palette into the component window.



- Change the “Hyperlink” label to “Movie Search”.
- In the hyperlink bindings inspector, set the pagename attribute to “MovieSearch”.

The pagename binding is a mechanism for navigating to another page. By setting the attribute to “MovieSearch”, you’re telling the application to open the MovieSearch page when the hyperlink is clicked.

6. Add a hyperlink to the MovieSearch component.

A user should also be able to get back to the Main page from the MovieSearch page.

- Create a hyperlink labeled “Browse Movies” in the MovieSearch page that opens the Main page.

7. Save and run your application.

Reload the Browse Movies page in your web browser to get the Movies application to load the new Movie Search link. In the Movie Search page, try searching for movies that match the word “The”. Notice that the Movies application locates only movies that begin with the string “The”.

- Using the DisplayGroupEditor, change the qualification mode for the movies display group.

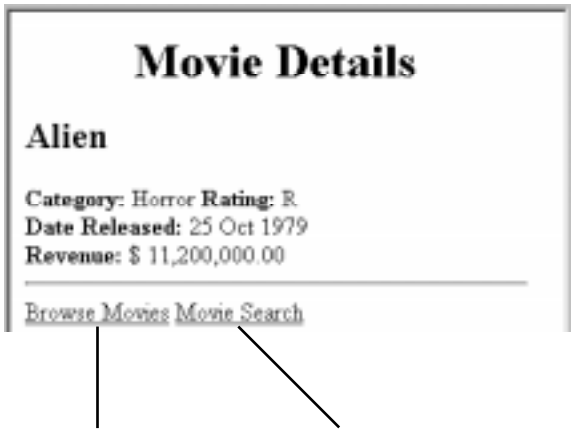
How does the qualification mode affect searching in the Movies application?

Adding a MovieDetails Page

The MovieDetails page shows you the detailed information about a movie you select in either the Main or the MovieSearch page. So, for example, you can choose a movie in the Main page and then click a "Movie Details" hyperlink to display the movie's details in a new page. For this to work, the Main and MovieSearch pages have to tell the MovieDetails page which movie the user selected.

The MovieDetails page uses a WODisplayGroup object to display movie information in the same way that the Main and MovieSearch pages do. In the MovieDetails page, however, the display group manages exactly one record: the selected movie. In the other pages, the display group manages a set of many records.

The Database Wizard doesn't provide a layout option that resembles the user interface of the MovieDetails page, so in this section you'll perform by hand the tasks the Database Wizard performed to create the other two pages. You'll learn how to create a WODisplayGroup variable and how to bind it to dynamic elements. In the next few sections, you'll extend the MovieDetails page to display movie roles and the starring actors.



1. Create a component.

   - Add a new component to the Movies application, and name it "MovieDetails".

2. Launch EOModeler.

   - In the OPENSTEP Enterprise program group, choose the EOModeler entry.

   An empty model opens.

   For Mach Users Only: On Mach systems, the EOModeler application is in /NextDeveloper/Apps.

3. Open your model file.

   - In EOModeler, choose Model->Open.
   - In the Open panel, navigate to your Movies.woa application directory.

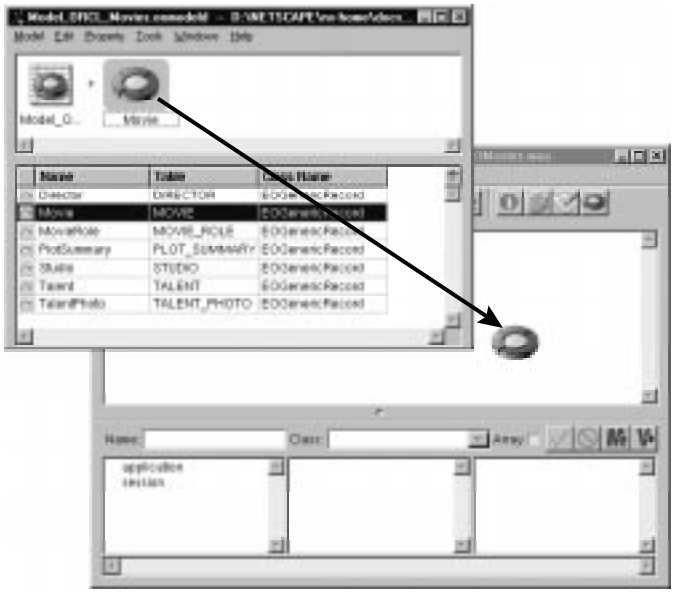
- Select the model file inside the application directory.
- Click Open.

After your model file opens, you can close the empty model.

For Mach Users Only: If you are doing the tutorial on a Mach system, you have to follow a procedure similar to the one you used to choose a model for the Database Wizard for the Movie Search page. you are doing the tutorial on a Mach system, you need to follow a slightly different procedure. To open the model file, bring the Workspace File Viewer to the front. Navigate to your Movies.woa application directory, and choose File->Open as Folder. The Workspace opens a new viewer window that's focused on the contents of the Movie.woa directory. Double-click the model file package to open it in EOModeler.

4. Add a WODisplayGroup to the MovieDetails component.

- Select the Movie entity in the EOModeler's Model Editor.
- Drag the Movie entity into the MovieDetails component window.



WebObjects Builder creates a component variable named movies, a WODisplayGroup object.

5. Add a setMovie: method.

As stated earlier, the MovieDetails display group manages only one object: the movie selected in the previous page. The Main and MovieSearch pages need a way to set this record in the MovieDetails' display group before the MovieDetails page opens. They use the setMovie: method to do this.

- In the MovieDetails component window, click the script button.

The script window is empty except for the following line of code, which declares the component's display group:

```
id movies;
```

- Add the following method to the script:

```
- setMovie:aMovie {
   if (aMovie)
      [movies setObjectArray:[NSArray arrayWithObject:aMovie]];
   else
      [movies setObjectArray:nil];
   [movies selectObject:aMovie];
}
```

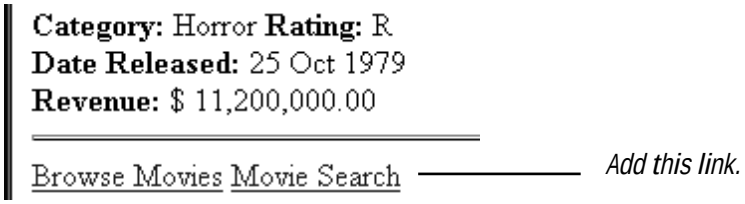
A WODisplayGroup manages a set of objects (called its *objectArray*) fetched from the database. The setMovie: method sets movies' objectArray to an array containing the single object aMovie. Display groups also keep track of a *selection*. Generally, a WODisplayGroup's selection is maintained programmatically. In this case, the setMovie: method sets the movies display group's selection to aMovie.

The Main and MovieSearch pages should invoke this method before they open the MovieDetails page, passing MovieDetails their selected movie in the argument aMovie.

6. Invoke the setMovie: method from the Main component.

To get to the MovieDetails page from the Main page, users use a hyperlink. The hyperlink should invoke MovieDetails' setMovie: method and then open the MovieDetails page.

- Next to the "Movie Search" hyperlink in the Main component, add a hyperlink labeled "Movie Details".



- Add the following method to the Main component script:

```
- showDetailsForMovie {
   id detailsPage = [[self application] pageWithName:@"MovieDetails"];
   [detailsPage setMovie:[movies selectedObject]];
   return detailsPage;
}
```

This method creates the MovieDetails page and then invoke's its setMovie: 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.

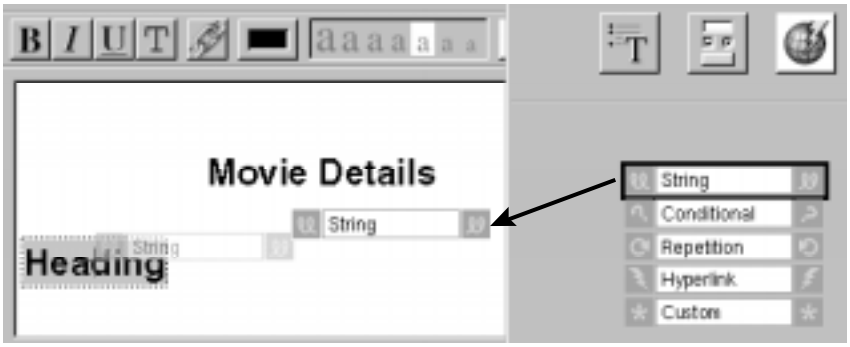
- Connect the showDetailsForMovie method to the action attribute of the Movie Details hyperlink.

7. Invoke the setMovie: method from the MovieSearch component.

- Repeat the same steps to invoke setMovie: from the MovieSearch component.

8. Create the MovieDetails user interface.

- Drag a heading element from the Static Elements palette into the MovieDetails component window.
- Double-click the "Heading" label so it's selected.
- Drag a string element from the Abstract Elements palette into the MovieDetails component window.



The string element is now inside the heading element. As a result, the string's value is formatted as a heading element. In the next task, you'll bind the selected movie's title to this heading string.

- Place the cursor below the heading string.
- Type a "Category: " label.
- Drag another string element into the MovieDetails component window.
- Select the "Category: " label.
- Click the bold button.
- Repeat the steps above adding Rating, Date Released, and Revenue labels and strings.

9. Create bindings.

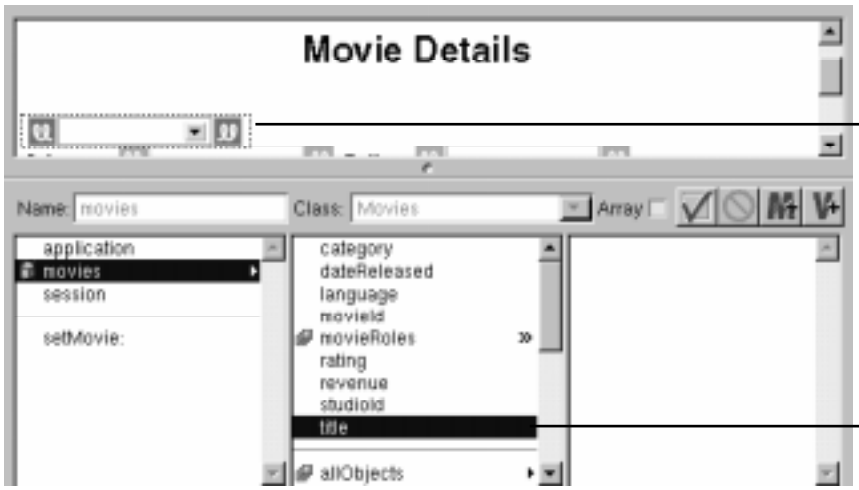
Now you need to bind attributes of the selected movie to the string elements so the selected movie's information will be displayed in the Movie Details page.

- Click inside the heading string.
- Select movies->title in the component window's object browser.

A display group manages objects associated with a single entity. The movies display group operates on Movie objects. The attributes above the line in the middle column of the object browser are actually attributes of Movie objects rather than attributes of the display group itself. These attributes are provided as a convenience; they correspond to the display group's selected object.

- Double-click title.

Double-clicking movies->title binds the title of movies' selected object to the heading string's value attribute. Recall that the setMovie: method sets movies' selected object to the movie selected by a user in the previous page.



Select this heading string. A pull-down button appears in the string element when the element is selected.

Double-click to bind the string element's value binding to the title of the selected movie.

- Repeat the steps above to create bindings for the Category, Rating, Date Released, and Revenue strings.

10. Add date and number formats to the Date Released and Revenue Strings.

    String elements have dateformat and numberformat attributes just like text field elements.

    - Add formats the same way you would for text fields.

    In the Browse Movies page, you used "%d %b %Y" and "$ #,##0.00" for the date and number formats, respectively.

11. Add links back to the Main and MovieSearch pages.

    Now add hyperlink elements to the Movie Details page so a user can navigate to the Main and Movie Search pages from the Movie Details page.

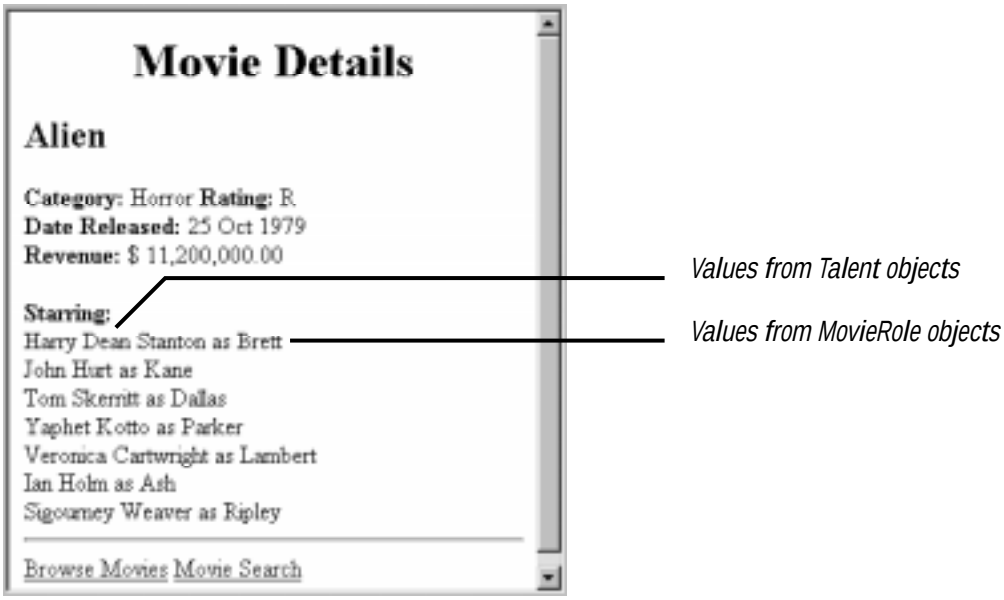
    - Add two hyperlink elements to the bottom of the page.
    - Label one "Browse Movies" and the other one "Movie Search".
    - Set the pagename attributes to "Main" and "MovieSearch".

12. Save and run your application.

    Reload the Browse Movies page in your web browser to get the Movies application to load the new Movie Details link. In the Browse Movies page, select a movie and click the Movie Details link. The Movie Details page should display all the movie's information.

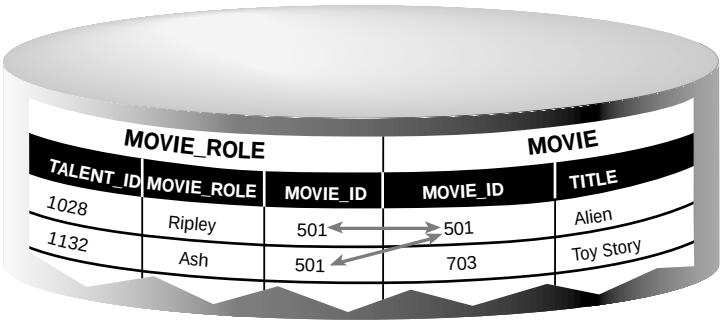
Adding Actors to the MovieDetail Page

So far your Movies application fetches, inserts, updates, and deletes Movie objects. But 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.

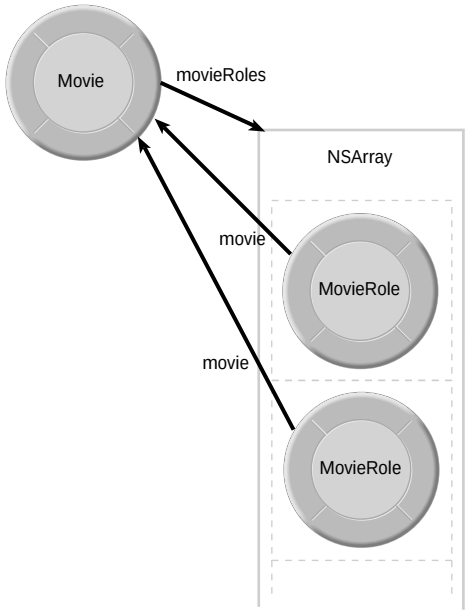


Before you can use MovieRole and Talent objects in your application, you first need to go back to EOModeler and add relationships to the model's entities.

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_ROLE table, MOVIE_ID is a foreign key, while in MOVIE it's a primary key. A foreign key correlates with the primary key of another table to model a relationship a source table (MOVIE) has to a destination table (MOVIE_ROLE). 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".



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 an object graph that connects the Movie object with its MovieRoles. As shown below, a Movie object has an array of its MovieRoles, and the MovieRoles each have a Movie.



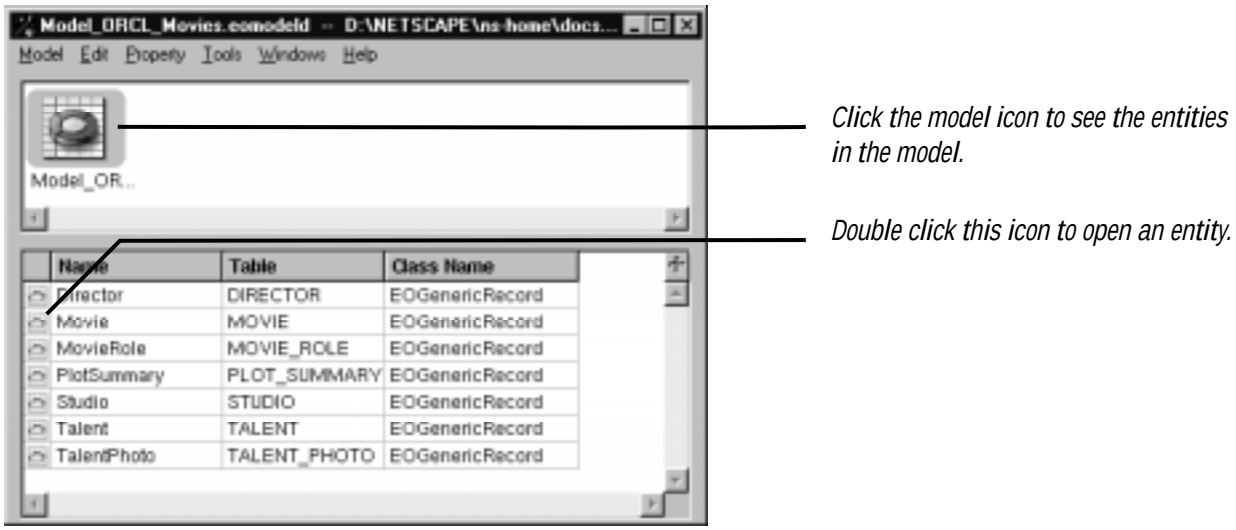
After you add relationships to your model, you can access MovieRole and Talent objects through the Movie objects fetched by the movies display group. Using WebObjects Builder, you'll bind attributes of MovieRole and Talent objects to elements in the Movie Details page.

1. Add relationships to your model.

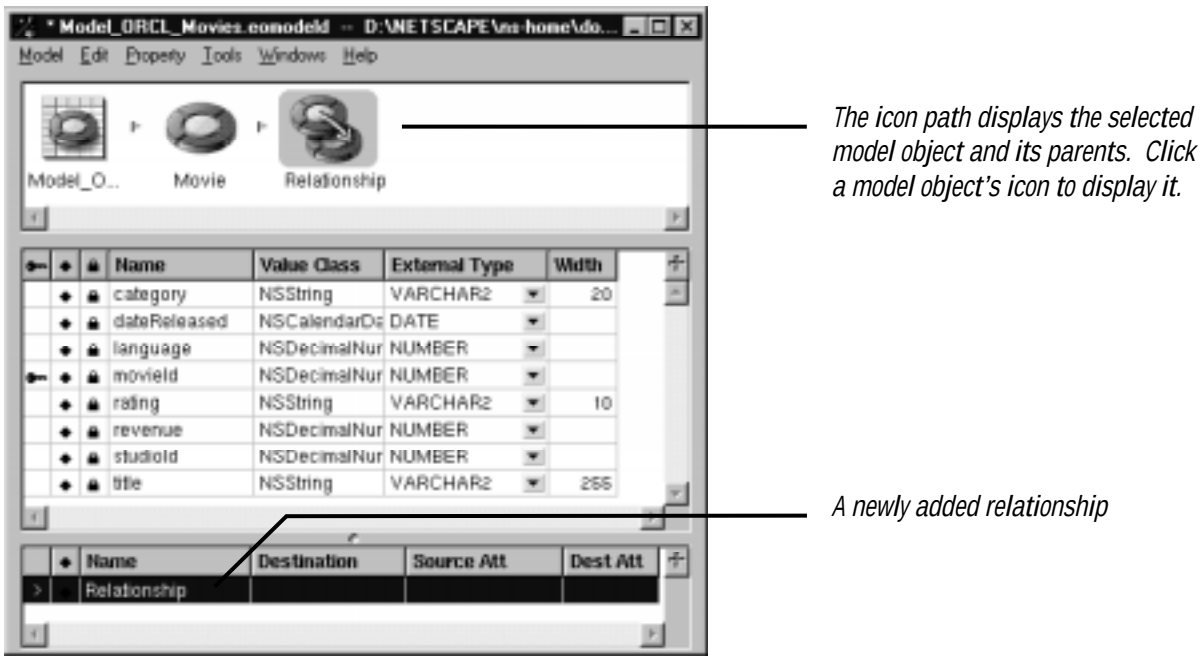
The Movies application uses two pairs of inverse relationships. The first pair defines the relationship between the Movie and MovieRole entities, while the second pair defines the relationship between the MovieRole and Talent entities. An Enterprise Objects Framework relationship is *directed*; that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

To create a relationship:

- In EOModeler, double-click the open entity icon for the entity to which you want to add a relationship.



- Choose Property->Add Relationship.



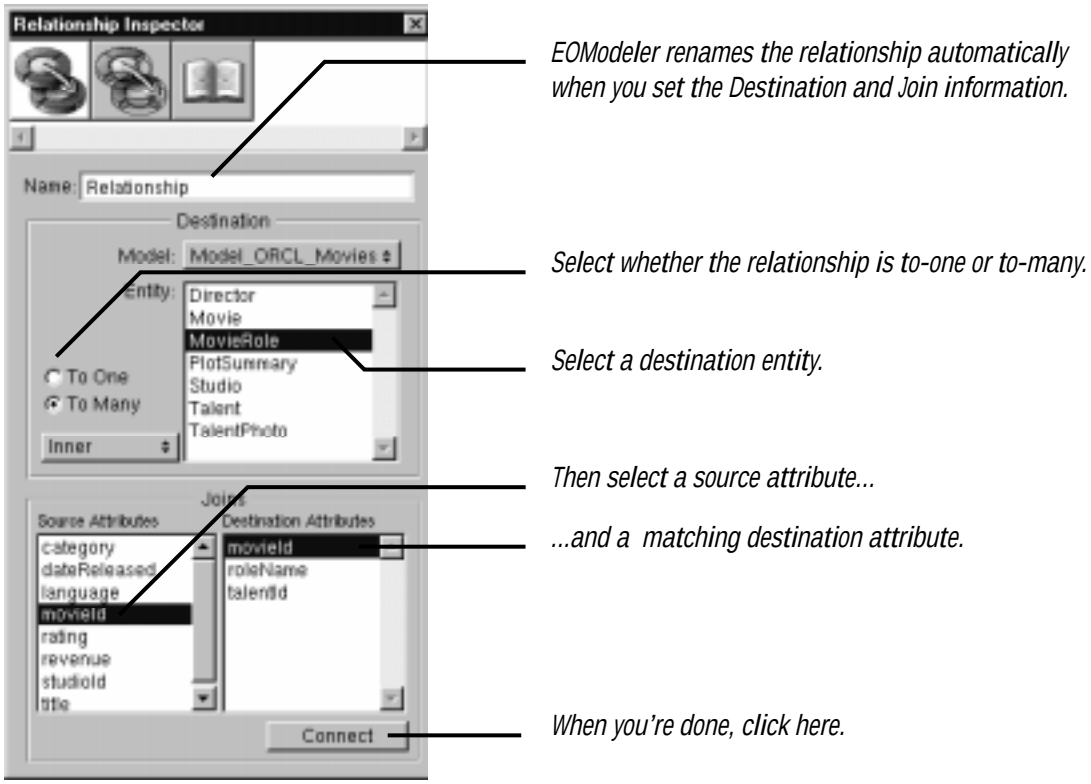
A new relationship named "Relationship" is added in the table view at the bottom of the model file window.

- Choose Tools->Inspector.

The Relationship Inspector opens.

- Set whether the relationship is To One or To Many.
- Select a destination entity.

- Select a source attribute.
- Select a destination attribute.
- 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 "movieRoles". 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) "movie".

- Create the following relationships using the steps outlined above:
  - A to-many relationship in the Movie entity where:

    The destination entity is MovieRole.
    The source attribute is movieId.
    The destination attribute is movieId.

  - A to-one relationship in the MovieRole entity where:

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

- A to-one relationship in the MovieRole entity where:

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

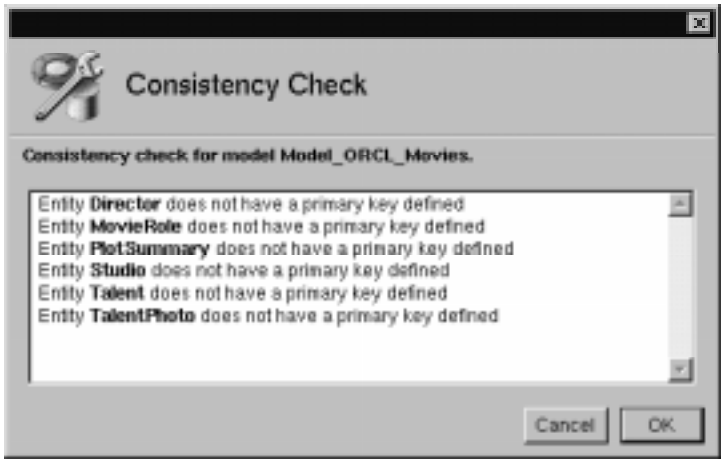
- A to-many relationship in the Talent entity where:

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

2. Save the model file.

▸ Choose Model->Save.

A Consistency Check panel runs alerting you that several of the model's entities don't have primary keys.



The Database Wizard assigned a primary key to the Movie entity. None of the other entities have been assigned a primary key.

▸ Click OK to save your model anyway.

You'll add primary key values in the next step.

3. Assign primary key attributes.

The Database Wizard assigned a primary key attribute to the Movie entity when it created the model. You need to assign primary keys to the rest of the entities using EOModeler.

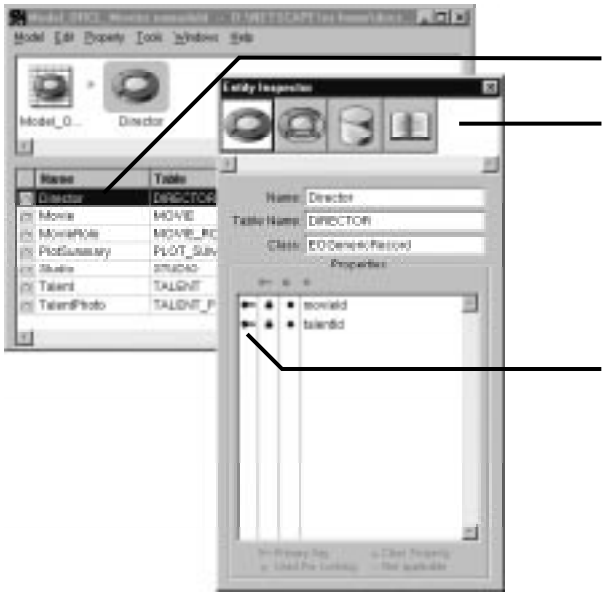
To assign a primary key:

▸ Click the model icon in the icon path to display the model's entities.

▸ Select the entity icon to which you want to assign a primary key.

▸ Open the Inspector.

▸ Click in the primary key column for the primary key attributes.



Select the entity to which you want to assign a primary key.

There are several entity inspectors. Be sure that the inspector title says "Entity Inspector".

Click in the primary key column for the attribute that is a primary key attribute.

The Movies application only uses the Movie, MovieRole, and Talent entities, but you should assign primary keys to all the entities in case you extend the application later.

- Specify the following primary keys for the remaining entities:
  - Director's primary keys are movieId and talentId.
  - MovieRole's primary keys are movieId and roleName.
  - PlotSummary's primary key is movieId.
  - Studio's primary key is studioId.
  - Talent's primary key is talentId.
  - TalentPhoto's primary key is talentId.

Note that some of the entities have *compound primary keys*; that is, a primary key that is composed of more than one attribute.

Now when you save EOModeler should not display the Consistency Check panel.

4. Add a repetition to the MovieDetails component.

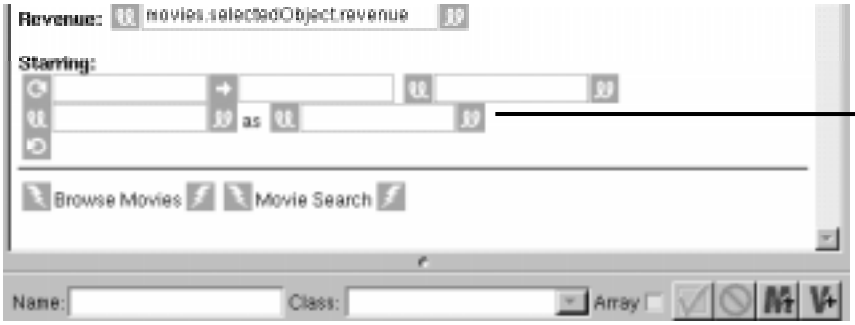
Now you'll extend the user interface of the MovieDetails component to display the actors in the selected movie. Since different movies have different numbers of roles, you need the dynamism of a repetition element.

- In the MovieDetails component window, add the bolded text **"Starring:"** beneath the Revenue line.
- Place the cursor below the "Starring:" label.
- Drag a repetition element into the component window from the Abstract Elements palette.
- Delete the "Repetition" text inside the element.

- Add three string elements inside the repetition.

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

- Type the word "as" between the last two string elements.

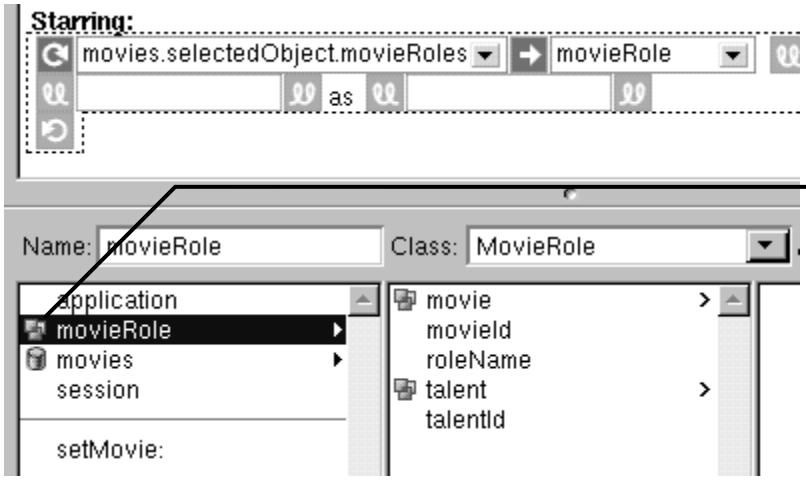


The only carriage return in the repetition goes here.

5. Create bindings

- Click in the first field of the repetition element.
- Navigate to movies->movieRoles in the object browser.
- Double-click movieRoles.

Remember that a Movie object has a movieRoles instance variable that's an array of MovieRole objects. The steps above bind the movieRoles array of the selected Movie object to the repetition's list attribute. WebObjects Builder automatically adds a new variable to the MovieDetails component named movieRole and binds it to the repetition's item attribute.



This icon denotes a variable whose class is defined in the class dictionary. When the Database Wizard creates a display group, it adds a class definition to the class dictionary for each entity in the model.

The movieRole variable, a MovieRole object, has an instance variable—talent—that's a Talent object. (The talent instance variable is the result of the to-one relationship from the MovieRole entity to the Talent entity.) Through the movieRole instance variable, you can access the name of the actor playing a particular role.

- Click in the first string element in the repetition.
- Navigate to movieRole->talent->firstName in the object browser.
- Double-click firstName.
- Similarly, bind movieRole->talent->lastName to the second string element.
- Bind movieRole->roleName to the last string element.

6. Save and run your application.

A WebObjects application only reads its model file once--when it loads the corresponding WODisplayGroup. As a result, you'll need to restart your the Movies application so it can see the relationships you added.

You may need the assistance of your system administrator to end the running Movies process, but generally you use the Windows NT Task Manager to end processes.

- Right-click on the toolbar.
- Choose Task Manager

The Task Manager window opens.

- Select the WODefaultApp process that corresponds to your Movies application.
- Click End Process.

For Unix Users Only: On Unix systems you can use the kill command to end the WODefaultApp process.

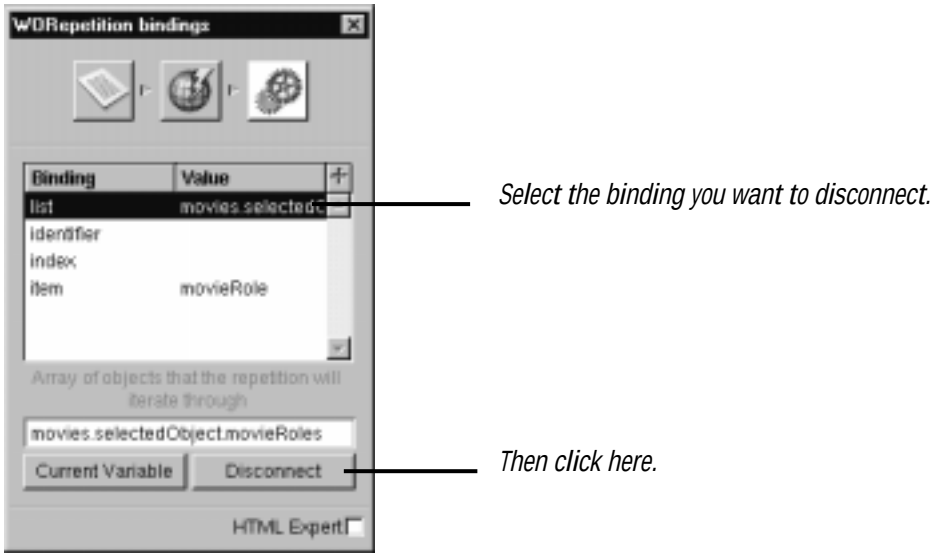
Adding a Detail Display Group to MovieDetails

In the previous section, you navigated relationships to access attributes in other entities. Using the movieRoles relationship in the Movies entity, you were able to display attributes of a movie's roles. Similarly, you displayed the names of the actors who played a role by navigating two relationships: movieRoles to get from a movie to one of its roles and talent to get from the role to the actor who played it. Traversing the object graph this way works very well in read-only scenarios where you only want to display information in a related object. However, t doesn't work completely in scenarios where you want to be able to make changes to the related objects.

In the next section, you'll give the application the ability to insert, update, and delete movie roles for the selected movie. This read-write scenario requires a slight modification to the way you access movie role information. To be able to insert new movie roles for a movie, you need a WODisplayGroup for managing MovieRole objects.

1. Disconnect the bindings for the "Starring:" repetition.

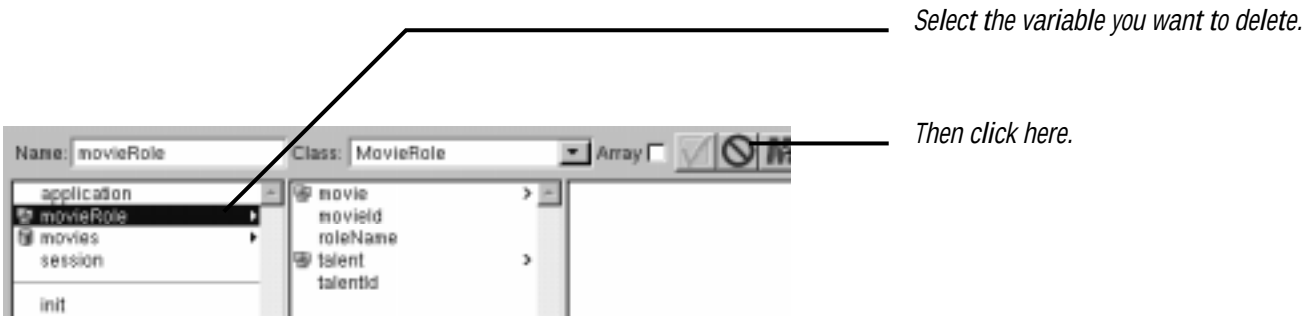
   - Inspect the repetition.
   - Disconnect the list binding.
   - Disconnect the item binding.



   - Similarly, disconnect the value bindings for each of the three string elements.

2. Delete the movieRole variable.

   - Select the movieRole variable in the object browser.
   - Click the delete button.



Recall that the movieRole variable was created automatically when you bound movies->movieRoles to the repetition's list attribute. Since you disconnected the list and item bindings, you don't need the movieRole variable anymore.

3. Create a WODisplayGroup to manage MovieRole objects.

- Drag the MovieRole entity from your model file into the MovieDetails component window.

WebObjects Builder creates a variable named movieroles in the MovieDetails component. movieroles is a WODisplayGroup that manages MovieRole objects.

4. Add an init method to the MovieDetails component.

As is, the movieroles display group manages *all* the movie roles in the database, but it should manage only the MovieRole objects that are related to the selected movie. To restrict the objects that movieroles manages, you need to set up a *master-detail* relationship between the movieroles and movies display groups. The right place to establish such a relationship is in the init method.

- Add the following init method to MovieDetails' script:

```
- init {
   [super init];
   [movieroles setDataSource:[[movies dataSource]
      dataSourceQualifiedByKey:@"movieRoles"]];
   return self;
}
```

The first line of this init method invokes any initialization performed by the object's superclass. Whenever you implement an init method, include a call to the super class's init method. If you omit the call to super, your objects won't be fully initialized.

The second line of this method assigns a new data source to the movieroles display group. A data source—an instance of an EODataSource subclass—is an object that defines a basic interface for providing enterprise objects. It exists primarily as a simple means for a WODisplayGroup or other higher-level class to access a store of objects. For example, when you tell a display group to fetch, it does so by telling its data source to fetch.

To restrict the objects that movieroles displays, you need to replace movieroles' data source with a *detail data source*. A detail data source is a data source that qualifies (restricts) its set of enterprise objects to an object that's selected in a *master data source*. A detail data source is set up to provide objects for the destination entity of a particular relationship.

The following expression:

```
[[movies dataSource] dataSourceQualifiedByKey:@"movieRoles"]
```

gets the data source from the movies display group and asks it to provide a detail data source. The data source returned by dataSourceQualifiedByKey: is set up to provide MovieRole objects that are related to one of movies' enterprise objects through the movieRoles relationship.

5. Qualify movieroles' data source.

In a master-detail setup, changes to the detail apply to the objects in the master; for example, adding an object to the detail also adds it to the relationship of the *master object*. Once you have a detail data source, you can set the master object by sending the detail a qualifyWithRelationshipKey:ofObject: message. The detail then uses the master object in evaluating the relationship, and applies inserts and deletes to that master object. In the MovieDetails page, this plays out as follows: If you insert a new MovieRole object in the movieroles display group, it is also added to the movieRoles array of the selected Movie object.

- Modify the setMovie: method to look like this:

```
- setMovie:aMovie {
   if (aMovie)
      [movies setObjectArray:[NSArray arrayWithObject:aMovie]];
   else
      [movies setObjectArray:nil];
   [movies selectObject:aMovie];

   // Add the following lines.
   [[movieroles dataSource]
         qualifyWithRelationshipKey:@"movieRoles"
         ofObject:aMovie];
   [movieroles fetch];
}
```

6. Create the repetition bindings.

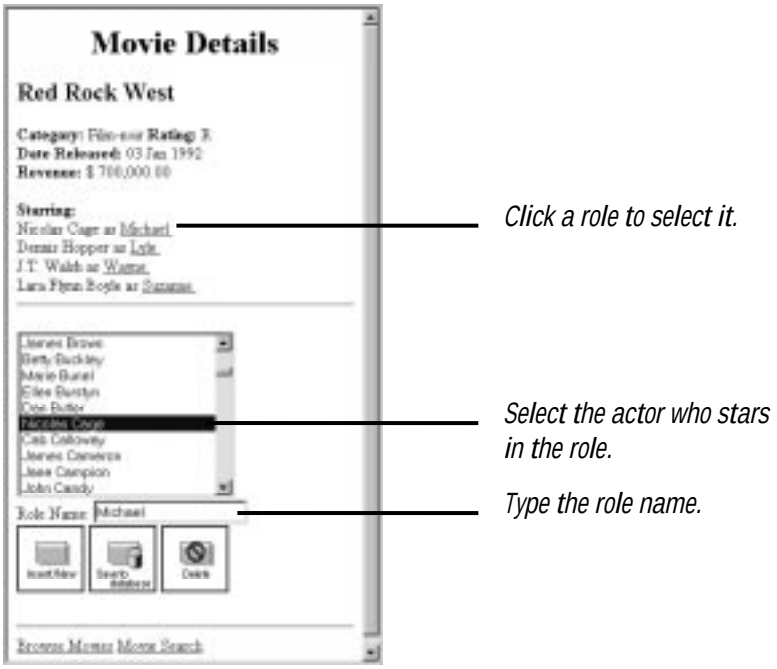
- Bind the movieroles displayedObjects method to the repetition's list attribute.

WebObjects Builder adds the movieRole variable back and automatically binds it to the repetition's item attribute.

- Bind movieRole->talent->firstName to the first string element.
- Bind movieRole->talent->lastName to the second string element.
- Bind movieRole->roleName to the last string element.

Inserting, Updating, and Deleting MovieRoles

In this final section, you'll add the ability to insert, update, and delete movie roles:

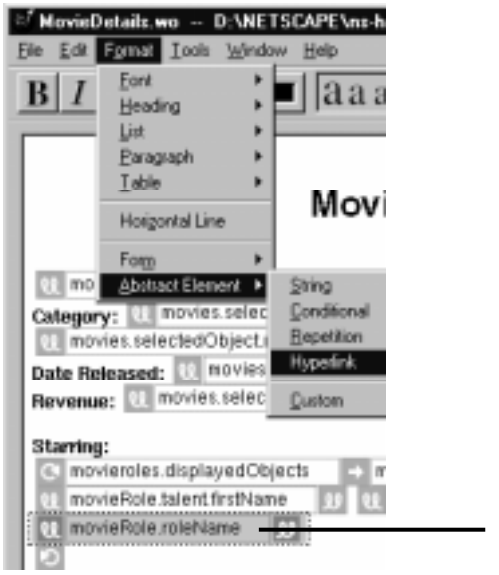


Most of the set up is done already, but you have to add a few more methods. In this section, you'll add methods to the MovieDetails component that:

- Select a movie role when a user clicks a link.
- Manage the selected actor for the page's browser element.
- Return the full name of an actor for display in the browser.
- Save changes to movie roles to the database.

1. Add a hyperlink around the roleName string.

    - Select the string element that displays role names.
    - Choose Format->Abstract Element->Hyperlink.



The whole string element is gray when it's selected.

The Hyperlink command adds a hyperlink element as the selected element's parent. Now the string element is nested inside the hyperlink.

2. Add an action method to MovieRoles.

When you click one of the movie role hyperlinks, the application should select the corresponding MovieRole object in the movieroles display group. You need to write a method to do this.

- Add the following method to MovieDetails' script:

```
- selectObject {
   [movieroles selectObject:movieRole];
}
```

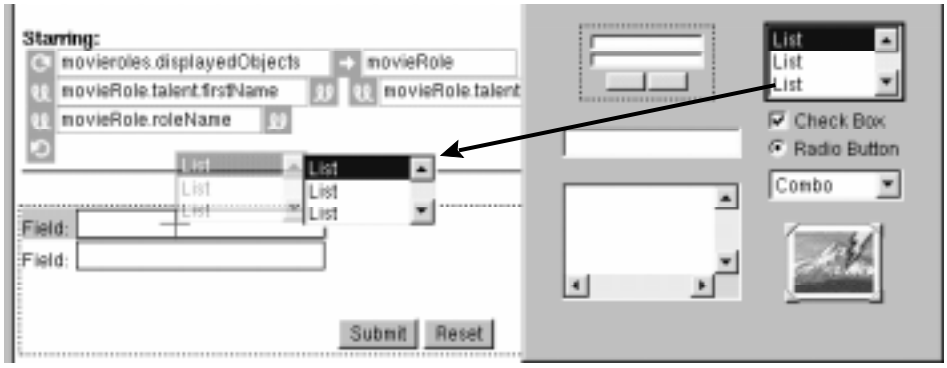
The movieRole variable is bound to the repetition element's item attribute. In the selectObject method above, movieRole represents the role a user clicked on.

3. Create the hyperlink bindings.

- Bind the selectObject method to the hyperlink action attribute.

4. Add a form element to the MovieDetails page.

- Drag a form element from the Form Elements palette into the MovieDetails page.
- Replace the form's first label and text field with a browser element.



Drag a browser element to replace the first field and its label.

- Label the remaining text field “Role Name:”.

5. Add a Talent display group.

You need a WODisplayGroup to provide a list of all the Talent objects for the browser.

- Drag the Talent entity from EOModeler into the MovieDetails page.
- Configure the newly created **talents** display group to sort its objects alphabetically (ascending) by **lastName**.
- Configure it to fetch on load.
- Set its Entries per batch to 0.

6. Create list and item bindings for the browser element.

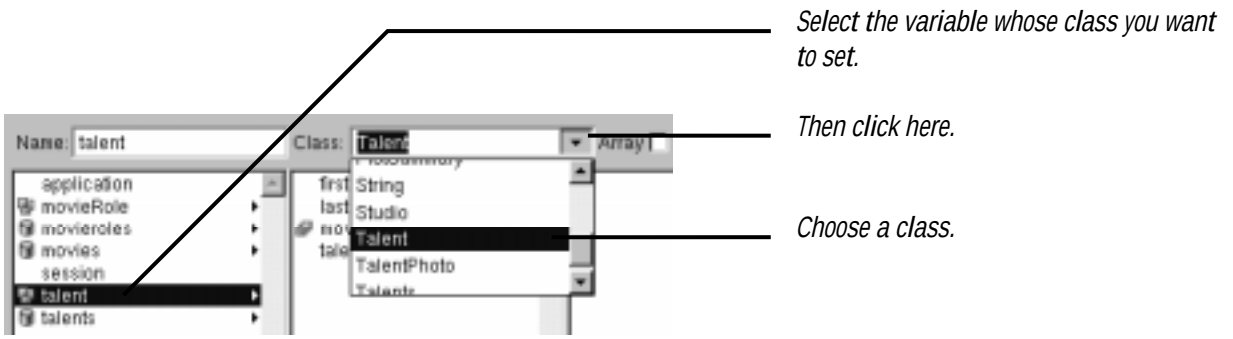
- Bind **talents->displayedObjects** to the browser element’s **list** attribute.

Like a repetition element, a browser has an **item** attribute. As the browser moves through its **list**, it sets **item** to the object at the current index.

- In the browser inspector, select the **item** attribute.
- Type **talent** in the text field.
- Click Connect.

WebObjects Builder creates a new variable named **talent** in the MovieDetails component and binds it to the browser’s **item** attribute.

- Set the class of the **talent** variable to be the Talent class.



Strictly speaking, this step is unnecessary, but it makes it easier to see what's going on. Now when you select talent variable in the object browser, you can see that it has lastName and firstName attributes.

7. Create the value binding for the browser element.

The value method tells the browser how to get the strings to display in the browser. For each item in its list, the browser evaluates its value. Typically you bind an attribute of the item variable to the value attribute. For example, you could bind talent->lastName to the value attribute. However, the browser in the MovieDetails page should display the full name for each of the actors. Since the Talent class doesn't provide an attribute for a full name, you need to write a method to create and return a string containing the full name for the current actor.

- Add the following method to the MovieDetails script.

```
- fullName {
   return [NSString stringWithFormat:@"%@ %@",
         [talent valueForKey:@"firstName"],
         [talent valueForKey:@"lastName"]];
}
```

- Bind the fullName method to the browser's value attribute.

As the browser iterates through its list, it sets its item (in this case, talent) to the current Talent object and then invokes the fullName method. When fullName is invoked, it returns a string containing the full name of the current actor.

8. Create the selections binding for the browser element.

Browser elements have a selections attribute that should be set to an array of objects. A browser's selection can be zero, one, or many objects; but in this talent browser element, 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 actor and one to set the selected actor from an array object.

- Add the following method to the MovieDetails script:

```
- talentSelection {
   id selectedTalent = [[movieroles selectedObject] valueForKey:@"talent"];
   if (selectedTalent)
      return [NSArray arrayWithObject:selectedTalent];
   else
      return nil;
}
```

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

- Add the following method:

```
- setTalentSelection:array {
   if ([array count])
      [[movieroles selectedObject]
            takeValue:[array objectAtIndex:0]
            forKey:@"talent"];
}
```

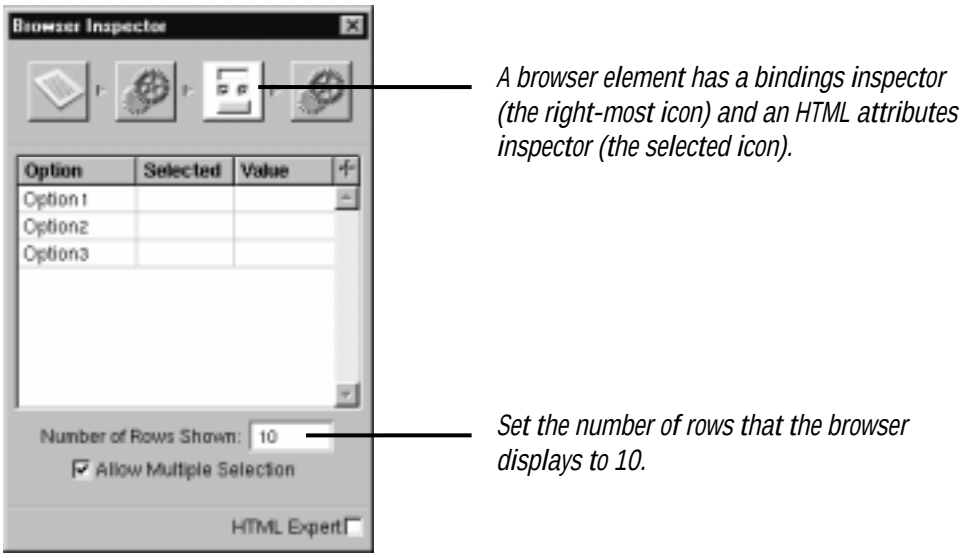
Again because the browser uses an array for its selections attribute, the setTalentSelection: method must take an array as its argument. If array's count is non-zero, then this method sets the selected MovieRole's talent object to the first object in the array. (Note that a user can select more than one actor. A production-quality application would probably alert the user that a MovieRole can only be played by one actor, but this simple tutorial application won't take that extra step.)

- Bind the talentSelection method to the browser's selections attribute.

Given this binding, the browser invokes talentSelection to determine what the selection is. It invokes setTalentSelection when a user changes the selection. As a result, the talent instance variable of the selected MovieRole object is set to the newly selected actor.

9. Configure the browser's appearance.

- Open the Browser Inspector.
- Set the number of rows shown to 10.



10. Create the text field binding.

- Bind movieroles->roleName to the Role Name text field.

11. Add Insert, Update, and Delete buttons.

- Delete the Submit and Reset buttons.
- In the MovieDetails script, delete the submit method associated with the Submit button.

Note: If you save your application after deleting the buttons, WebObjects Builder automatically deletes the submit method.

- Drag three active image elements from the Form Elements palette into the form element.

12. Assign images to the active image elements.

- Select the first active image element.
- Open the Active Image Inspector.
- Click Browse.

- In the Open panel navigate to the Main.wo directory in the Movies.woa application directory.
- Select the DBWizardInsert.gif file.
- Click Open.
- Follow the same procedure to set the second image's source to DBWizardUpdate.gif.
- Set the last image's source to DBWizardDelete.gif.

13. Create bindings for the active image elements.

- In the MovieDetails script, delete the three submit... methods that are bound to the active images.

The WODisplayGroup class defines the methods insert and delete that you'll bind to the insert and delete active image elements, respectively. It doesn't, however, provide a save method. You'll have to provide that.

When you use the Database Wizard and choose the Selected Record layout, the wizard sets up three active image elements just as you're doing in this page. As a part of that set up, it adds a method called saveChanges to the component script. You can copy the saveChanges method generated by the Database Wizard for the Main component and paste it into the MovieDetails component script.

- Copy the saveChanges method from the Main script and paste it into the MovieDetails script.

```
function saveChanges() {
   id exception;

   exception = self.session.defaultEditingContext.tryToSaveChanges();
   if (exception != nil) {
      [exception raise];
   }
}
```

self.session (the same as [self session]) refers to a WOSession object that represents a connection to the application by a single user. WOSession objects provides access to an EOEditingContext object. The expression

```
self.session.defaultEditingContext.tryToSaveChanges()
```

which is the same as the expression:

```
[[[self session] defaultEditingContext] tryToSaveChanges]
```

sends a tryToSaveChanges message to the WOSession'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*.

EOEditingContext's tryToSaveChanges method uses other Enterprise Objects Framework objects to analyze its graph of enterprise objects (Movie, MovieRole, and Talent 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, tryToSaveChanges returns an NSException object. (See the NSException class specification in the *Foundation Reference* for more information on exceptions.) By default, the 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.

- Bind the saveChanges method to the update image's action attribute.
- Bind movieroles->insert to the insert image's action attribute.
- Bind movieroles->delete to the delete image's action attribute.

Conclusion

Although it's a simple example, the Movies application introduced you to many of the concepts, tools, and skills you'll need to create Enterprise Objects Framework applications for the web. You've learned about:

- Using WebObjects Builder's Database Wizard
- Adding formatting to your user interface
- Using EOModeler to edit a model file
- Creating WODisplayGroups and binding them to dynamic elements
- Creating master-detail configurations
- Inserting, updating, and deleting enterprise objects