Writing WebScript in WebObjects Builder

WebScript is WebObject's scripting language, which you can use to write your application's logic. You can write all or a portion of your application in WebScript.

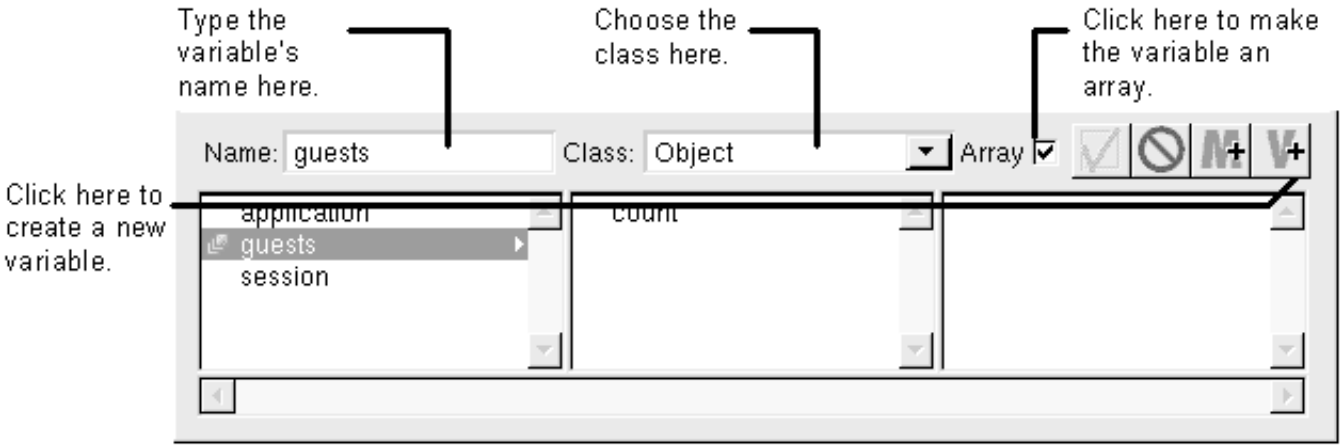
This chapter describes how to create WebScript in WebObjects Builder. To learn how to write a WebObjects application in general, read the WebObjects Developer's Guide. In particular, see the "Using WebScript" chapter, which describes WebScript syntax.

Creating Variables

You can create three different types of variables: application, session, and component. Create application variables in the application window's Application tab, session variables in the application window's Session tab, and component variables in the bottom part of the component window (called the object browser). Application and session variables have a longer lifetime than component variables. For more information, see "Application, Session, and Component Variables."

To create a variable:

1. Click the add variable button.
2. Type the variable's name in the Name field and press Enter.
3. Choose the variable's class in the Class field.
4. If the variable is an array, click the Array check box.



All variables that you create in WebScript are objects. Each object is an instance of a particular class. No matter what you set the class to, the variable is declared in the script this way:

```
id myVar;
```

Setting a variable's class helps when you bind dynamic elements to it because WebObjects Builder can then determine which attributes match the variable's class.

Application, Session, and Component Variables

	Where to create	Lifetime	Access in Component Script
Component variables	The component window's object browser	The component's lifetime	*myvar*
Application variables	Application tab of application window	The application's lifetime	`self.application.`*myvar*
Session variables	Session tab of application window	The session's lifetime	`self.session.`*myvar*

When you create a variable in a component window, you are creating a *component variable* that's visible only inside that component. No other components can access it. A component variable lives only as long as the component that created it. If the component's page is deallocated, so is the component variable. If the page is redrawn after the component has been deallocated, the component variable is reinitialized.

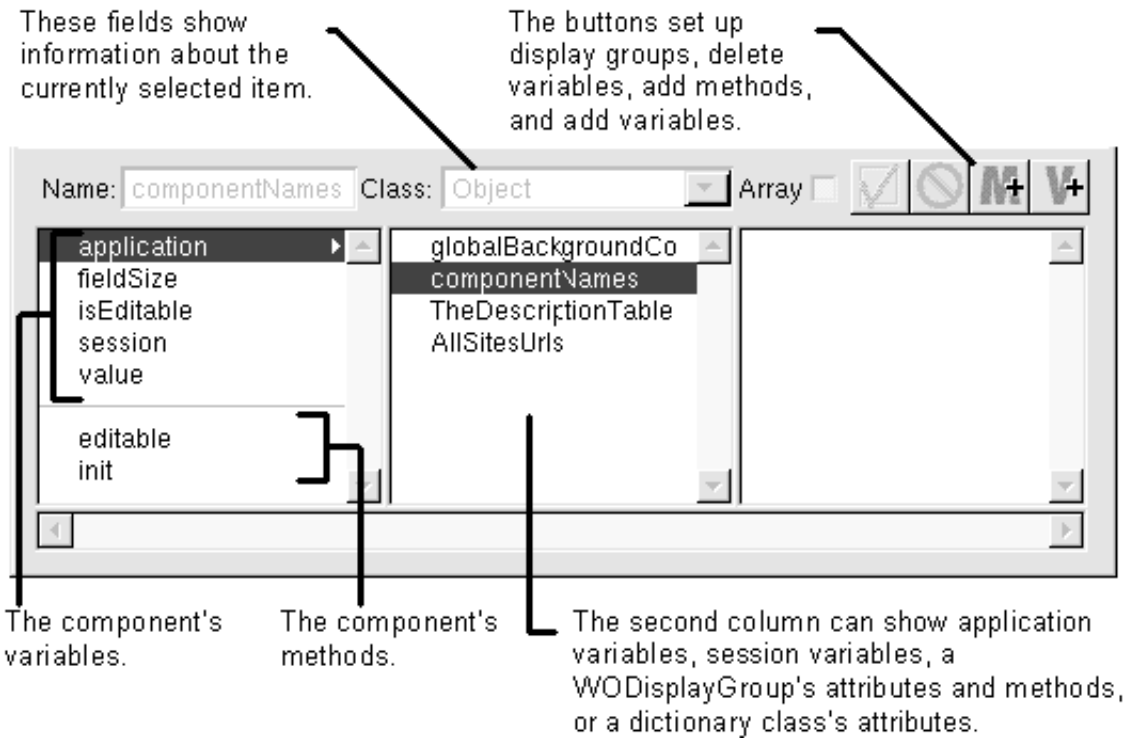
To create a variable with a longer lifetime, create an application or session variable. An *application variable's* lifetime is the lifetime of the application. One instance of an application variable is created per application. That is, if three users are running sessions of the same application, they will share an instance of that application variable.

Session variables live as long as a session. An instance of a session variable is created for each session. If three users are running sessions of the same application, three instances of a session variable are created. If one user changes the value of that session variable, only that user sees the change.

You can read more about application variables, session variables, and component variables in "Using WebScript" in the *WebObjects Developer's Guide*.

The Object Browser

The object browser provides an abbreviated view of the component's script file. The script window displays the entire script file. The object browser's primary use is to bind dynamic elements to the variables or methods in the script. Thus, you create a variable in the object browser, bind a dynamic element to it, and later you can write a method that accesses or changes the value in that variable.



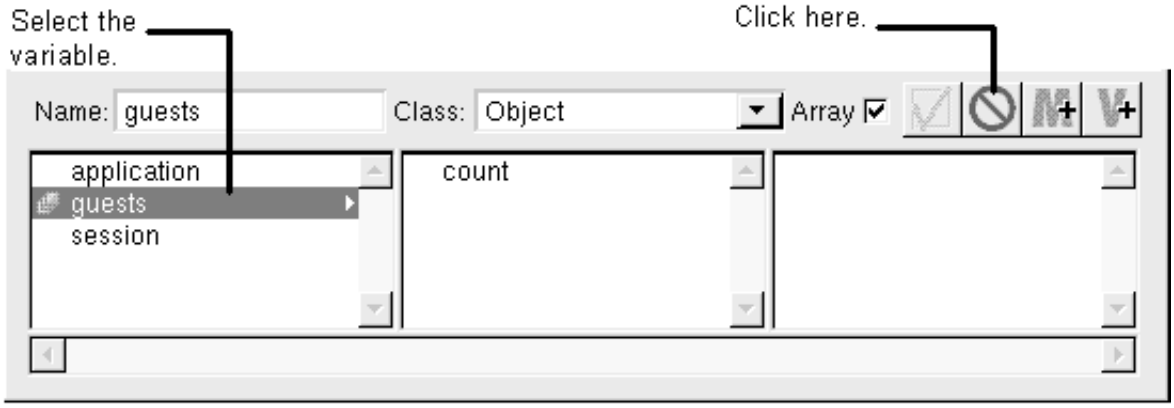
WebObjects Builder has three different object browsers:

Object Browser Location:	Creates:
Bottom half of the component window	Component variables
Application tab of application window	Application variables
Session tab of application window	Session variables

Deleting Variables

1. Select the variable in the object browser.
2. Press the Delete button in the top part of the object browser.

The variable is removed from the script file as well as the object browser.



Always use the object browser to delete variables. For variables, WebObjects Builder always assumes the list in the object browser is correct and updates the script to match. It does not go the other direction: If you delete a variable from the script window, WebObjects Builder tells you there are variables in the object browser that are not declared in the script and asks if it should add them.

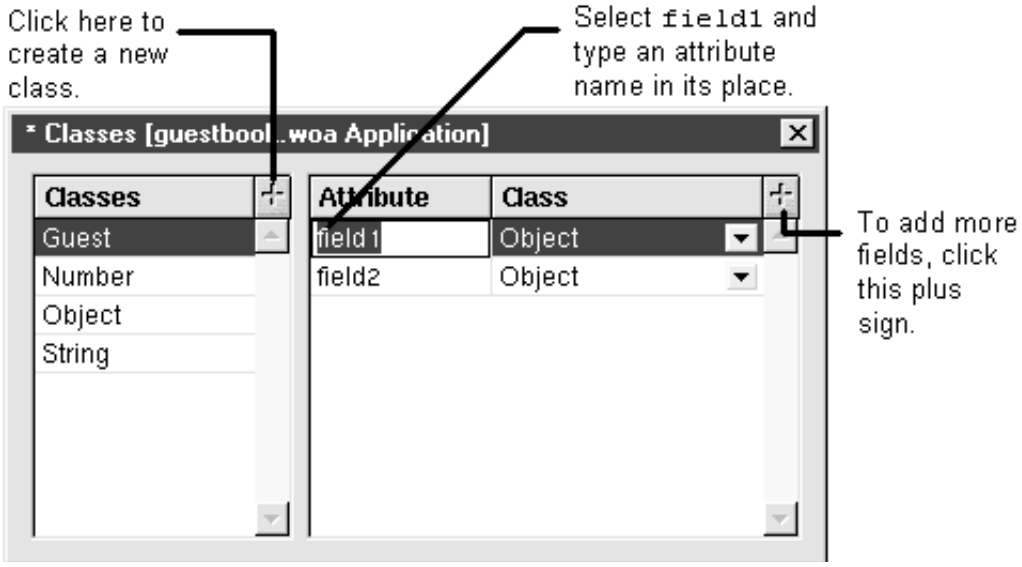
Although you can access application and session variables from a component window's object browser, you can't delete them there. You must delete application and session variables in the application window.

Creating Classes

Components and applications can define classes. If you define a class in the application window, all components can access it. If you define a class starting from the component window, only that component has access to that class.

To create a class:

1. Click the Classes button or choose Tools->Classes.
2. Click the plus sign in the Classes Table.
3. Type the class's name.
4. Click the plus sign in the Attributes table to add an attribute.
5. Choose the attribute's class from the pop-up in the Class field.



The Classes window changes its display as the main window changes. If the main window is a component window, the Classes window shows you that component's classes. If the main window is the application window and the Application tab or Session tab is selected, the Classes window shows you application's classes.

When you're creating a class, be careful that you are creating it where you intend to create it. Check the window's titlebar to see what you are editing.

Classes

A class specifies the type of a variable. There are three types of classes in WebObjects:

- *Base Classes.* The base classes are Object, Number, and String. They represent a single value. That is, a variable of the Number class represents a single number, and a variable of the String class represent a single string.
- *Composite Classes.* The composite classes are Array and Dictionary. Composite classes represent a group of related values. For example, a Dictionary class named Guest might represent all information about one guest, and an Array of Guest might represent a list of all guests.

  Most of the time when you create a new class in WebObjects Builder, you are creating a Dictionary class. When you define a Dictionary class, you must specify its attributes.
- *Enterprise object classes.* Enterprise object classes are used in applications that access a database.

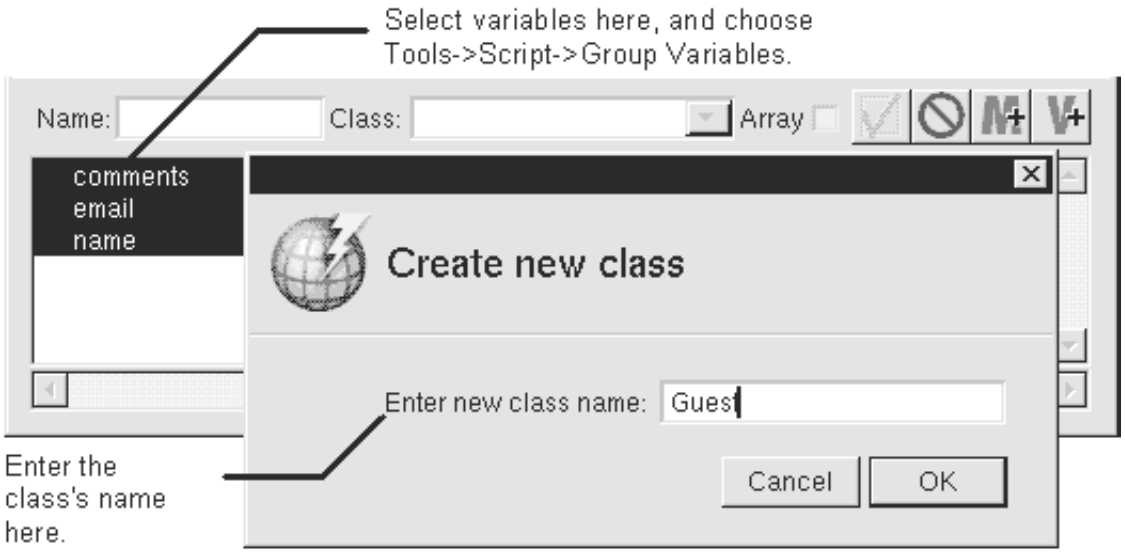
Object, Number, String, and Array are all part of the Foundation Framework. For example, if you click the array checkbox when creating a variable, you are creating an instance of NSMutableArray (mutable meaning you can add or delete values). If you're unfamiliar with the Foundation Framework, see "A Foundation for WebScript Programmers: A Quick Guide to Useful Classes" in the *WebObjects Developer's Guide*.

When you create a dictionary, you're also creating an instance of a Foundation class, namely NSMutableDictionary. The attributes you define for the class are keys to the dictionary.

Enterprise object classes are created when you add a display group to your application. See "Creating a WODisplayGroup."

Creating Classes by Grouping

1. Select variables in the object browser.
2. Choose Tools->Script->Group Variables.
3. Enter the Class's name in the panel that opens.



The Group Variables command takes the selected variables and creates a new dictionary class out of them. The variables become the dictionary's attributes. The class's scope is the same as the variables used to create the class. That is, if you group variables in a component window, the class is visible only in that component. If you group variables in an application window, the class is visible in all components.

The Ungroup Variables command reverses the effects of a Group Variables command. You can use it on any dictionary class, regardless of if it was created using the Group Variables command.

Deleting Classes

1. Select the class in the Classes window.
2. Press the Delete key.

If you have a variable of this class and the variable's attributes are bound to elements in the component, a message box appears telling you so. Click Continue to delete the class. You'll have to bind the elements to other parts of the script.

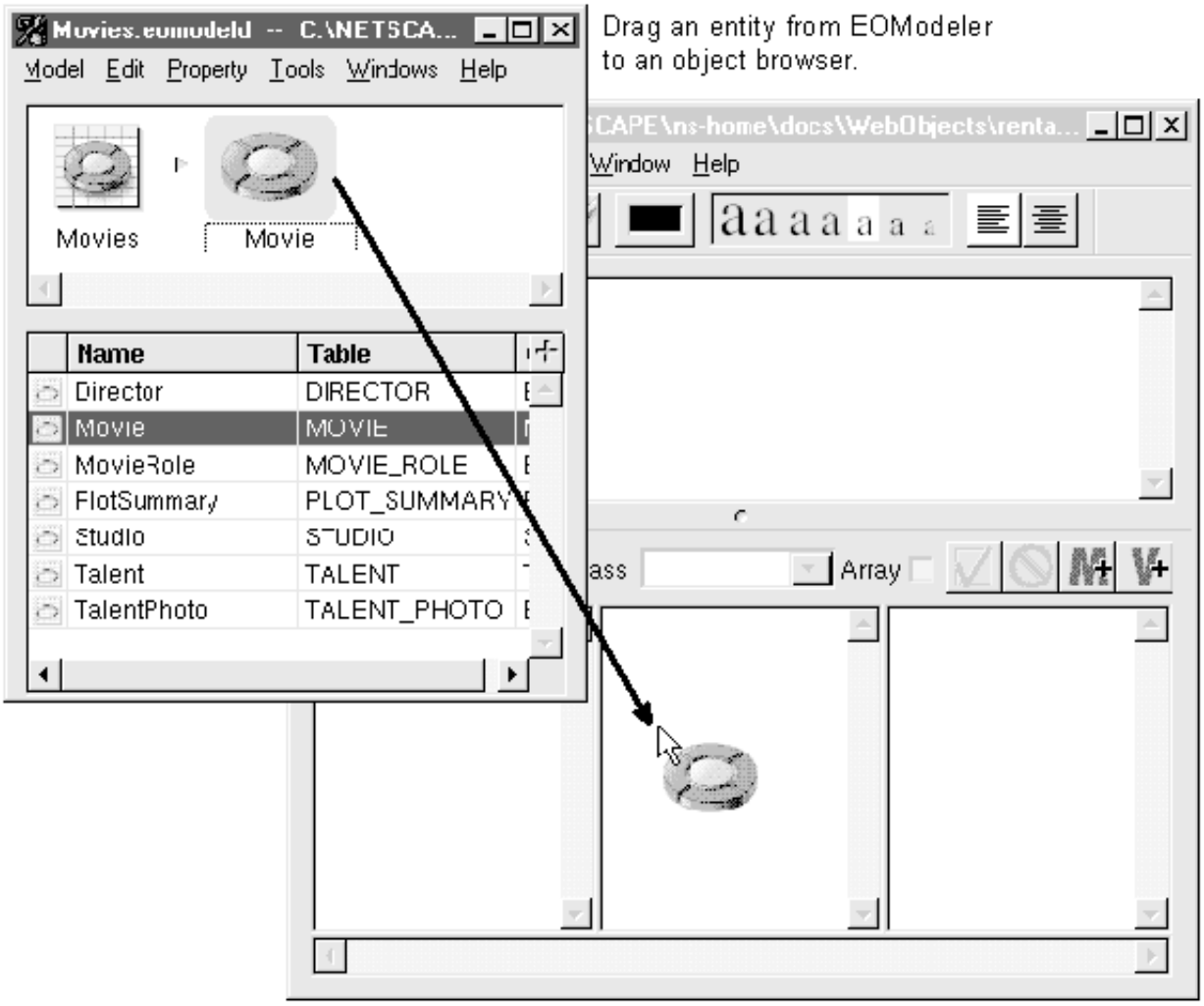
Tip: The Tools->Script->Ungroup Variables command can create variables out of a dictionary's attributes and retain the bindings to those attributes. You might want to use this command before deleting the class. See "Creating Classes by Grouping" for more information.

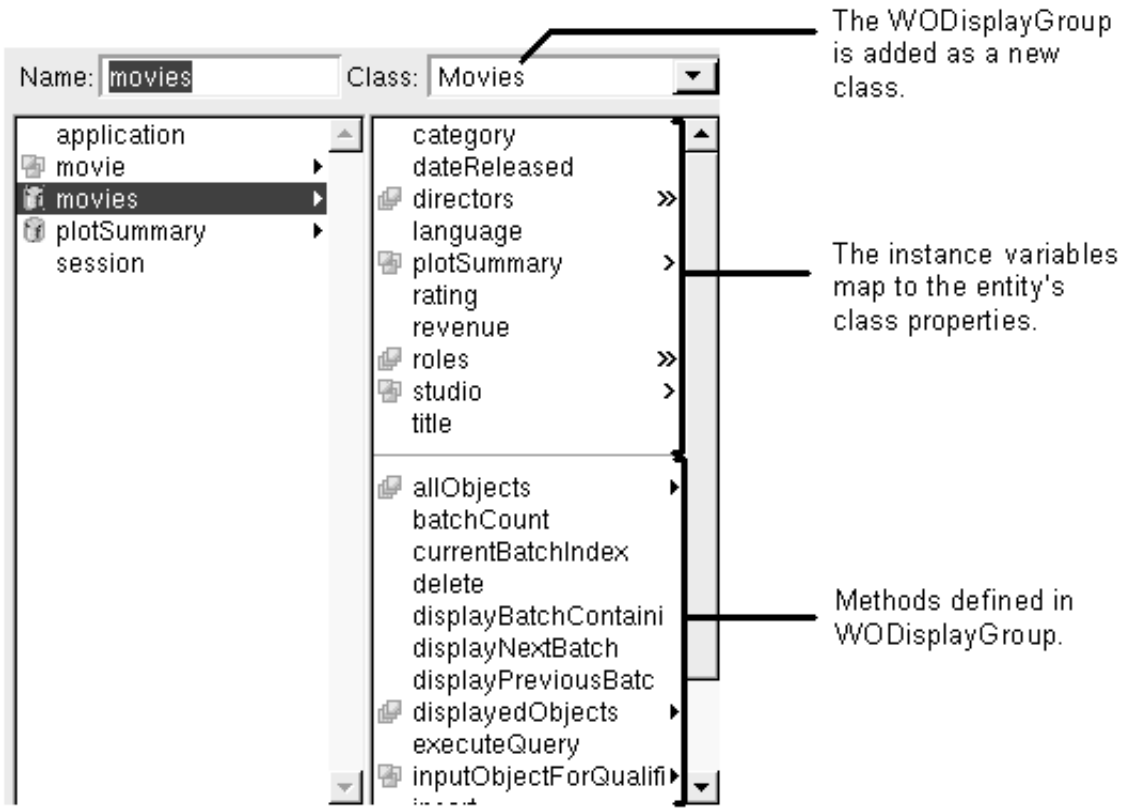
Creating a WODisplayGroup

1. Choose Database Wizard from the Tools menu.
2. Follow the instructions provided in the wizard.

OR:

1. Drag an entity from the icon path in EOModeler to a component window, or drag the entire model file from the file system to a component window.


Movies.eomodeld -- C:\NETSCA...
Model Edit Property Tools Windows Help
Movies
Movie
Name
Table
Director
DIRECTOR
Movie
MOVIE
MovieRole
MOVIE_ROLE
PlotSummary
PLOT_SUMMARY
Studio
STUDIO
Talent
TALENT
TalentPhoto
TALENT_PHOTO
Drag an entity from EOModeler
to an object browser.
CAPE\ns-home\docs\WebObjects\renta...
Window Help
Array



To write a WebObjects application that accesses a database, you need a WODisplayGroup. The WODisplayGroup can fetch, display, update, and search records in a database.

In most cases, you can create a database application using the Database Wizard. The wizard creates the WODisplayGroup, lays out the dynamic elements on the page, and creates bindings between the display group and the dynamic elements. After running the wizard, you have a working WebObjects application that fetches records from a database.

Note: You must run the database wizard from a component that's inside an application. That is, before you run the wizard, you must have created an application and that application must have at least one component.

If you need to do something more complex than the wizard provides, you can start by using the Database Wizard and then modify the results. If you don't want to use one of the wizard's layouts, you can create the application yourself.

If you create the application yourself, create the WODisplayGroup by dragging an entity from the EOModeler application. After you do this, you'll want to set display group options, and you'll have to bind elements to the display group. See "Setting Up a WODisplayGroup" and "Common WODisplayGroup Methods" for help.

Whichever way you add the WODisplayGroup, you get the following items with it:

- The model that contains the WODisplayGroup's entity is copied into the application. If you need to change the model or add another entity to a component, you should use this copy of the model file.
- All of the entities that are defined in the model are added to the component as classes. You can create instances of these classes and bind to them if necessary.

- An EOEditingContext is added to the session. You generally can't see this in the object browser, but it is accessible in the script.

WODisplayGroup, EOEditingContext, and Enterprise Objects

WebObjects applications that access a database use two things that most other WebObjects applications don't, the Enterprise Objects Framework and a WODisplayGroup instance.

The Enterprise Objects Framework represents items in your database as objects (enterprise objects). To begin creating a database application, you use the EOModeler application that comes with the Enterprise Objects Framework to create an entity-relationship model. (The Database Wizard can help you create a model.) You create an *entity* for each table in the database. For each entity, you create attributes that map to the columns in the table. Each entity in turn maps to an enterprise object, which means you can access that entity as an object. The columns in the database's table become the enterprise object's instance variables.

When you add an entity to a WebObjects application (either by selecting it in the wizard or by dragging it from EOModeler), it creates a WODisplayGroup. The WODisplayGroup manages objects associated with a single entity. For example, if you choose the Movie entity, the resulting WODisplayGroup operates on Movie objects. Note, however, that you can also access other kinds of objects through a Movie object's relationships. For example, if a relationship between a MOVIE table and a STUDIO table can be expressed with a join, you could access Studio objects associated with a particular Movie object.

For more information on creating models, see the chapter "Using EOModeler" in *Enterprise Objects Framework Developer's Guide*.

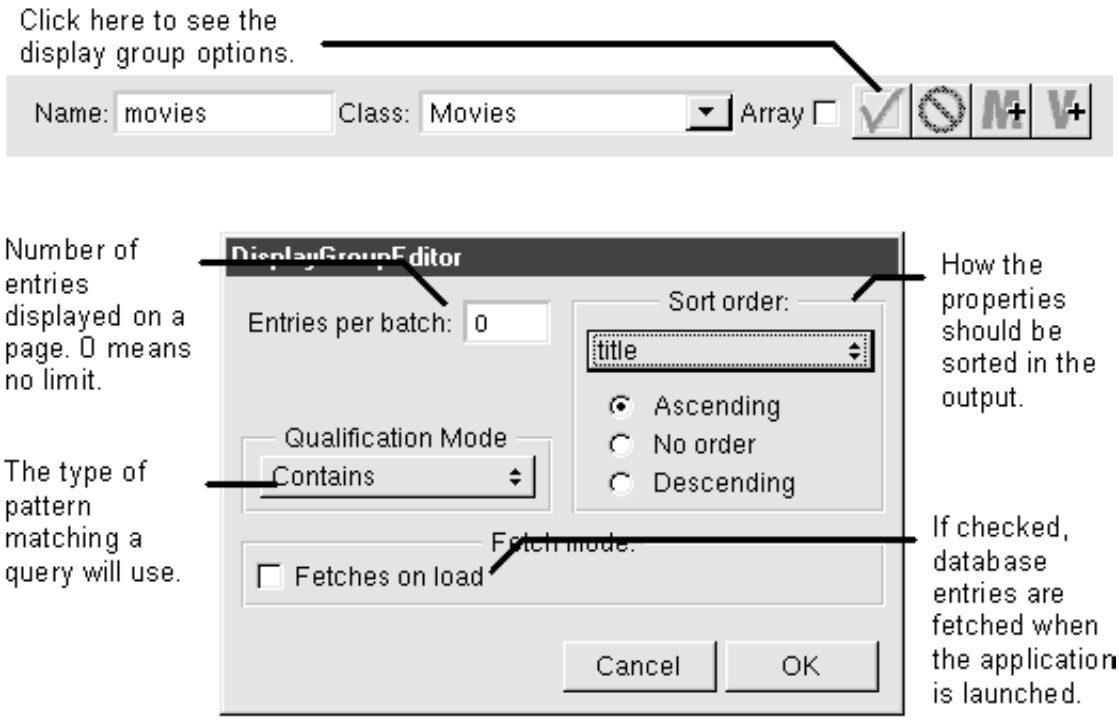
The WODisplayGroup has methods that can perform almost any operation on a database. A WODisplayGroup can fetch, insert, update, delete, and find records in the database as well as manage batches of search results. Thus, WODisplayGroup allows you to create a database application without writing much or any code. All you have to do is bind your dynamic elements to methods in the WODisplayGroup. For a list of commonly used methods and what they do, see "Common WODisplayGroup Methods."

WODisplayGroup uses an EOEditingContext to manage the graph of enterprise objects in your application. The EOEditingContext is stored in the session (WOSession instance) as the variable defaultEditingContext. You generally can't see the EOEditingContext in WebObjects Builder, but some of the scripts created by the Database Wizard reference this object. The EOEditingContext is responsible for ensuring that all parts of your application stay in sync. You need to access the EOEditingContext if your application can update database records—you use the EOEditingContext to save the changes.

To learn more about how to create a WebObjects database application, work through the Movies tutorial in *Getting Started*. The Movies tutorial teaches you how to use Enterprise Objects with WebObjects. For more detailed information on Enterprise Objects, read the *Enterprise Objects Framework Developer's Guide*.

Setting Up a WODisplayGroup

1. Select the WODisplayGroup variable.
2. Click the checkmark button in the object browser.



A WODisplayGroup performs certain database-related operations: it can display records, organize results into batches, fetch records from the database, as well as query the database. You use the options panel to set up how the WODisplayGroup performs these operations.

Displaying

The Entries per batch field specifies the number of records displayed at a time. For example, suppose you are displaying one hundred records. Instead of displaying all of these at once, you can set the batch size so that the page displays a more manageable number (for example, 10). If you set a batch size, the component should have buttons that will return the next batch of records and the previous batch of records. (Bind the buttons to the WODisplayGroup methods displayNextBatch and displayPreviousBatch.) If you don't want records displayed as batches, set the size to 0.

Sorting

The Sort order options specify how the records are sorted. The pop-up list shows all of the class properties for the WODisplayGroup's entity. Select the class property that the results should be sorted upon and then specify the sort order.

Fetching

If you're creating a component that is simply going to list a set of records from the database, you probably want the Fetches on Load check box checked. If checked, the WODisplayGroup performs a fetch when the application starts up. If you're creating a query-based application, you probably want this check box turned off so that the user controls when the fetch is performed.

Querying

Qualification Mode sets how the pattern matching works. When a user enters a string in a text field to search the database for that string, the qualification mode sets which part of the string must be matched.

- Prefix means that the text starts with the string the user enters in the search field. (For example, suppose you are searching a database of movies and you enter "the" in the search for title text field. Prefix mode returns all of the movies that begin with "the.")
- Contains means the string can be anywhere in the entry. (For example, it returns all movies that contain the string "the" anywhere in the title.)
- Suffix means the entry ends with the string the user typed in the field. (For example, it returns all movies that end with the string "the").

Common WODisplayGroup Methods

The WODisplayGroup performs all database accesses for your application. The more commonly used methods are listed below, sorted by the type of operation they perform.

Displaying Results

These methods give you access to database objects and allow you to display them.

allObjects
: All of the records to be displayed. For example, if your application has a query that results in one hundred records being fetched but you have set up the display group to only display ten at a time, allObjects contains all one hundred records.

displayedObjects
: The records actually being displayed. For example, if your application has a query that results in one hundred records being fetched but you have set up the display group to only display ten at a time, displayedObjects contains those ten records.

selectedObjects
: The records in the current selection.

selectedObject
: A single selected record. Usually, this returns the first record in the selectedObjects array.

Managing Batches

These methods control the grouping of records into manageable batches to display. (You set the batch size on the WODisplayGroup options panel.)

displayPreviousBatch
: Select the previously-displayed batch of records and then reloads the page.

displayNextBatch
: Selects the next batch of records and then reloads the page.

batchCount
: The number of batches to display. For example, if you're fetching two hundred records and the batch size is ten, batchCount returns twenty (twenty batches of ten records each).

batchIndex
: The number of the batch currently displayed, where 1 is the first batch displayed. For example, if the batch size is ten records and displayedObjects is showing records 11 through 20, the batchIndex is 2.

Querying

These methods are used in applications that perform queries.

executeQuery
: Builds a qualifier using **inputObjectForQualifier** and the pattern matching you set on the display group options panel, and then fetches the records that match that qualifier. The qualifier is not preserved.

inputObjectForQualifier
: Returns an entity object that is used to create the qualifier. For example, if you wanted the application to search on movie titles, you would bind a text field to **inputObjectForQualifier.title**. (**title** comes from the entity that created the WODisplayGroup.)

secondObjectForQualifier
: Used for from-to queries to specify the "to" value. This method must use the same property as **inputObjectForQualifier**. For example, if you wanted to query all movies released between two dates, you would bind the first field **inputObjectForQualifier.dateReleased** and the second field to **secondObject-ForQualifier.dateReleased**. The resulting qualifier would fetch all movies release after the date specified in **inputObjectForQualifier** but before the date specified in **secondObjectForQualifier**.

Modifying the Database

These methods modify the database.

insert
: Adds a new empty record. You should provide a dictionary containing default values for the record and use the **setInsertedObjectDefaultValues:** to specify that the WODisplayGroup should use that dictionary to create all new values. For example, these statements ensure that all new records contain a value for movie title:

```
[dictionary setObject:@"New title"] forKey:@"title"];
[movies setInsertedObjectDefaultValues:dictionary];
```

delete
: Deletes the selected records.

If you're modifying a database, you must explicitly save the changes. Write a method that uses the session's EOEditingContext to save the changes. For example:

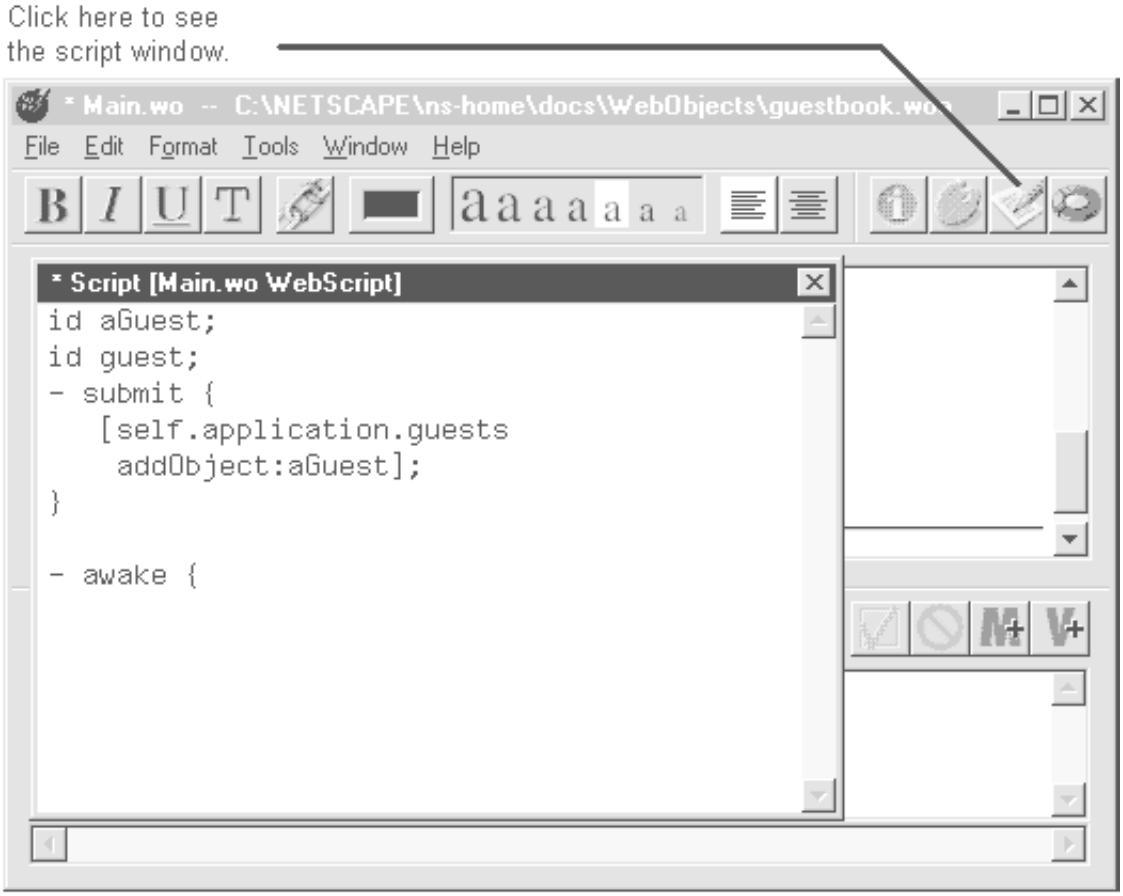
```
[self.session.defaultEditingContext saveChanges];
```

See the EOEditingContext class description in the *Enterprise Objects Framework Reference*.

Creating Methods

1. Click the script button to open the script window.

2. Type the method's implementation into the script window.



Methods perform the action of a component. Components usually define several methods. The script window shows the implementation of all of the component's methods.

After you add a method to the script file, the component window displays that method in the object browser so that it's possible to bind a dynamic element on the page to the method.

If you're unsure of the method syntax, you can use the object browser's add method button (the button with an M+) to create the method. This creates a method named action, lists it in the object browser, and creates its declaration in the script file. You can then open the script window, change the name, and type its implementation.

WebScript supports two different syntax styles. Set your style preference in the Preferences panel. See "Setting the Style Preference."

Standard Methods You May Want to Implement

Applications, sessions, and components define some common methods that you may want to implement. These methods are invoked during initialization, deallocation, and the request-response loop. (The *request-response loop* is a WebObjects application's main loop. It begins when the user generates a request by typing an URL or clicking a button on the page. It ends when the application has processed the request and generated a response page. One request-response loop cycle is called a *transaction*.) The list below provides a brief description of when you want to implement these methods.

Important: The chapter "Integrating Your Code Into the Request-Response Loop" in the *WebObjects Developer's Guide* provides full details on how to implement these methods. Read it before you write any code.

init
: Initializes application variables, session variables, and any component variables that should persist as long as the component persists.

dealloc
: Deallocates all variables initialized in the init method.

awake
: Initializes variables that need to be reinitialized at the top of the request-response loop.

sleep
: Sets to nil all variables that were initialized in the awake method.

takeValuesFromRequest:inContext:
: Invoked at the beginning of the request-response loop, right after the awake method. Use this method to perform any tasks that need to be performed before the action of the page but after the bindings have been synchronized. (Bindings are synchronized by calling the superclass, WOComponent, implementation of takeValuesFromRequest:inContext:.)

invokeActionForRequest:inContext:
: Invoked when the component is about to perform an action based on user input. Use this method if you want to generate a different object for the response page than would normally be created.

appendToResponse:inContext:
: Invoked right after the request page has generated the response page. Use this method if you want to append text to the response page.

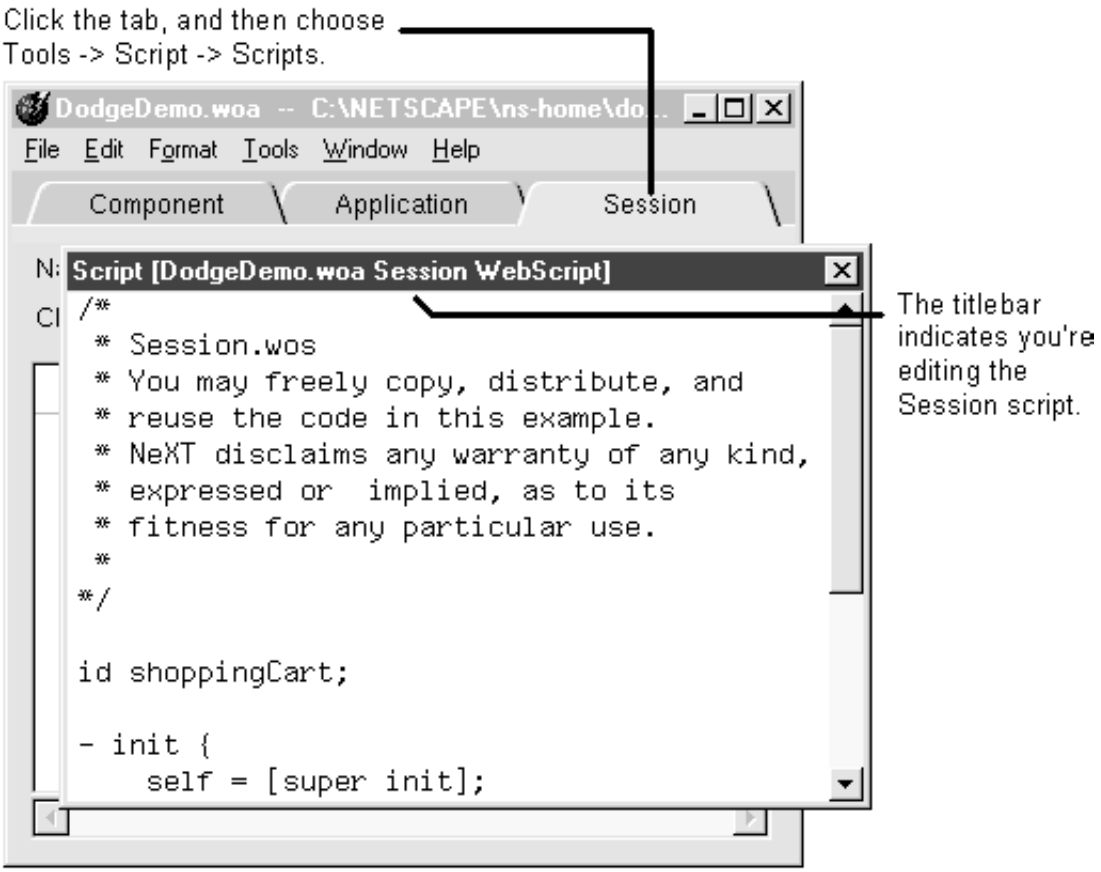
The Script Window

The script window shows you the current component's script. All of the variables and methods shown in the object browser are declared in the script.

To open the script window, choose Tools->Script->Scripts or press the script button on the component window.

The script window changes its display as the main window changes. If the main window is a component window, the script window shows you that component's script. If the main window is the application window and the application tab is selected, the script window shows you application script. Similarly, if the session tab is selected, the script window shows you the session script.

When you're editing a script, be careful that you are editing the correct script. Check the window's titlebar to see which script you are editing.

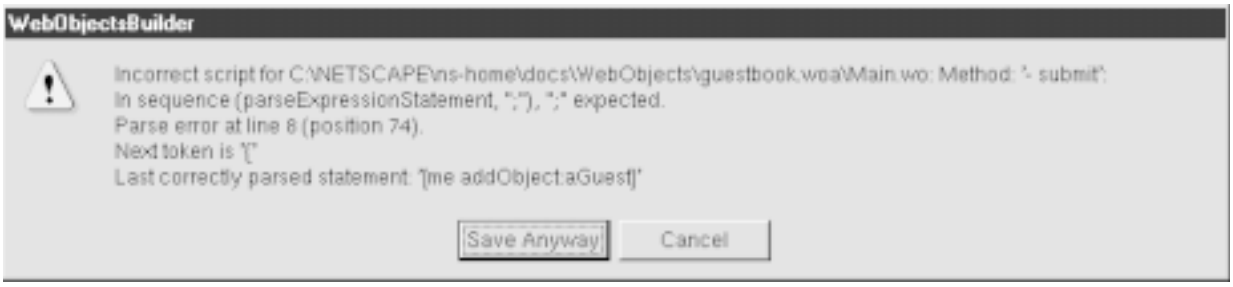


Deleting Methods

To delete a method, simply delete it from the script window. When the component window updates, the method is no longer listed in the object browser.

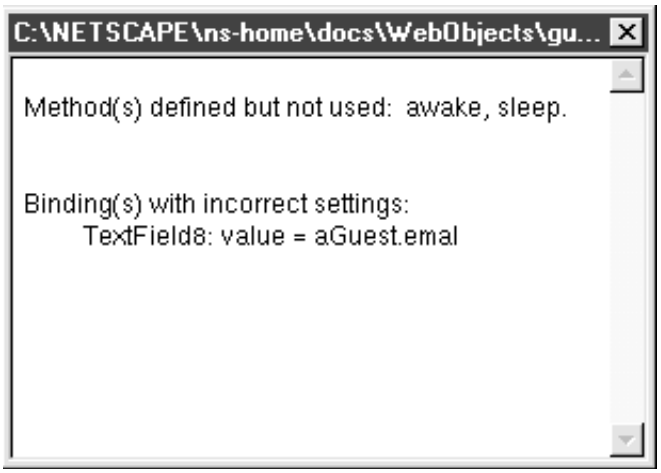
Error Checking

If you make a syntax error when writing a method, WebObjects Builder displays a message box describing the error when you try to close the script window or to save the component.



As further error checking, you can use the Check Consistency command to verify that the component's elements are bound to variables and methods that actually exist in the script file.

1. Select the component you want to check.
2. Choose Tools->Check Consistency.

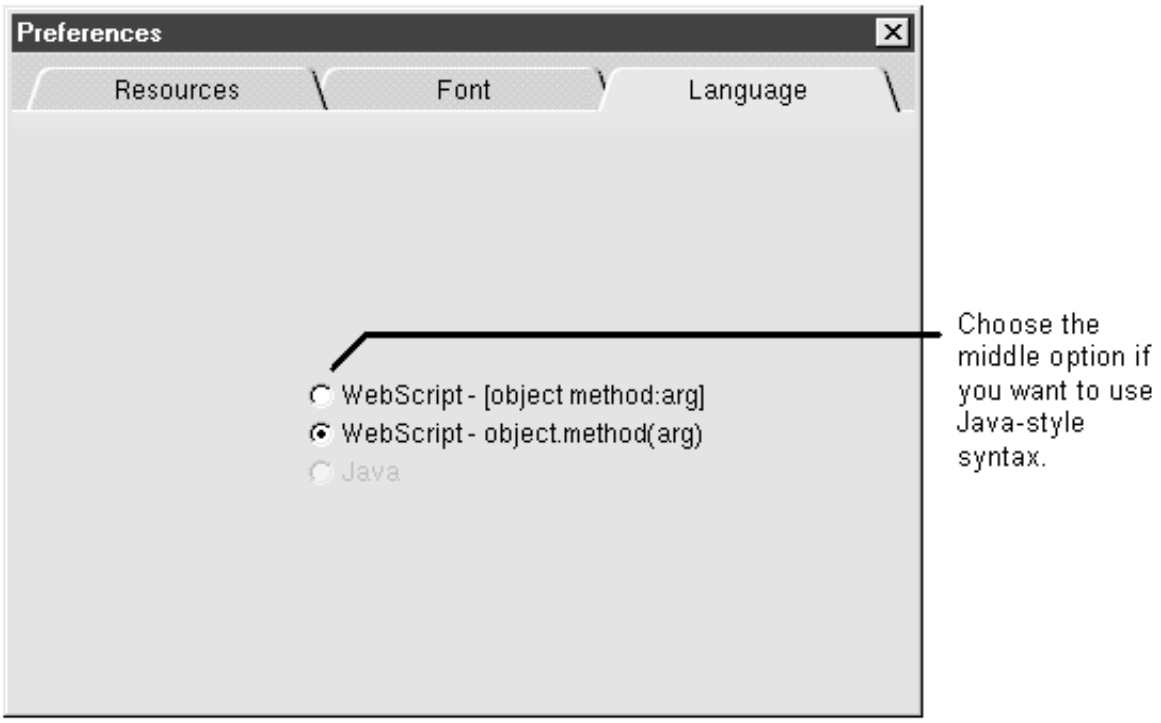


The Check Consistency command also tells you if it detects a method that is declared but is not bound to anything. This is not necessarily an error—for example, you would never bind awake or sleep to any element in a component, but these two methods are used. They are invoked at the beginning and end of each request-response loop.

Setting the Style Preference

WebScript has two different accepted syntax styles. One style is similar to Objective-C. The other style is similar to Java. The default is the Objective-C style syntax, and that is the style WebObjects Builder uses when you create a new method.

To have WebObjects Builder use Java-style syntax, bring up the Preferences panel, go to the Language tab and choose WebScript - object.method(arg).



After you do this, any method you create look like those shown below. (See "Using WebScript" in the *WebObjects Developer's Guide* for a description of both syntaxes.)

Script [Main.wo WebScript]

```
id guest;
id aGuest;
function submit() {
    self.application.guests.addObject(aGuest);
}

function awake() {
    aGuest = NSMutableDictionary.dictionary();
}

function sleep() {
    aGuest = nil;
}
```