What's New in WebObjects 4.0

This document describes changes made to the WebObjects product between release 3.5 and release 4.0. Release 4.0 is compatible with release 3.5, but you must recompile any existing applications. This document tells you how to convert applications to 4.0, describes changes made to existing features, and then describes some new features you may want to start using in your applications. This document is organized as follows:

WebObjects 4.0 includes Enterprise Objects Framework release 3.0. For a description of the changes to Enterprise Objects Framework, see "What's New in Enterprise Objects Framework 3.0."

Note: This document describes changes in the Java and Objective-C versions of the WebObjects APIs. Where Java and Objective-C method names are dissimilar, both method names are provided. For methods that take zero arguments, both languages use the same name. For single argument methods, the Java name is the Objective-C name minus the trailing colon (:).

Compatibility With Earlier Releases

WebObjects 4.0 is backward compatible to WebObjects 3.5. However, you must keep in mind the following:

- WebObjects 4.0 is the first release of WebObjects that runs on Mac OS X Server and Yellow Box for Windows NT. It does not run on OpenStep 4.2. Because of this change, the locations of WebObjects files have changed. See "File Location Changes" (page 10).
- Because Mac OS X Server and Yellow Box are not upgrades to OpenStep for Mach and OpenStep Enterprise, respectively, their underlying frameworks are not necessarily upwardly-compatible. In particular, the Foundation framework supplied with Mac OS X Server and Yellow Box isn't upwardly-compatible. As one example, NSArray's makeObjectsPerform: method doesn't exist in the versions of Foundation that ship with Mac OS X Server and Yellow Box.
- The file location changes require some changes to your Project Builder projects. See "Converting an Existing WebObjects Application" (page 3).
- Yellow Box uses a different version of the Java-wrapped APIs. The package names, class names, and some method names have changed. There are scripts to help you convert your Java code. See "Converting an Existing WebObjects Application" (page 3).
- WebObjects applications are launched differently in WebObjects 4.0, as described in "Running an Application on WebObjects 4.0" (page 11).
- The WebObjects Framework contains many new optimizations that should greatly improve your application's performance. However, you may find your code relied on some part of the request-response loop or template parsing code that is no longer always performed. If so, there are compatibility flags to disable these optimizations. These flags are described in "Converting an Existing WebObjects Application" (page 3).
- The default adaptor now supports multithreaded request handling. This change should have no effect on your running application other than to make resource loading faster.

Converting an Existing WebObjects Application

To convert a project to WebObjects 4.0, do the following:

1. Convert your project's makefiles as described in the document **$*NEXT_ROOT*/Developer/Makefiles/Conversion/DirectoryLayout/ConvertMakefilesReadMe.rtf**.

   (***$NeXT_ROOT*** is the root installation directory specified when WebObjects was installed.) The project files must change to point to new locations for the build tools. The **ConvertMakefilesReadMe.rtf** document tells you how to make those changes.

2. If your project contains Java code files, convert them as described in "Converting Java Code" (page 4).

Note: The Java APIs have changed considerably. If you've written Java code, you must convert it before you can compile.

3. Open your project in Project Builder, and click "Upgrade Now" when prompted.

   Among other things, upgrading your project in this fashion will:

   - Upgrade your **PB.project** file to the latest version.
   - Upgrade your makefiles to the latest versions.
   - Convert your WebObject project suitcases to the new format.
   - Convert your project so that it conforms to the new localization scheme. As a result of this, script (**.wos**) and **.api** files are moved outside of their component (**.wo**) directories. Script files now appear in the Classes suitcase. For more information on the new localization scheme, see the section "Changes to Localization" (page 51).

4. Build. If errors occur during the build, fix them and re-build the project.

   The WebObjects Framework contains many new optimizations that should greatly improve your application's performance. However, you may find your code relied on some part of the request-response loop or template parsing code that is no longer always performed.

5. Run the project. If your application doesn't run as expected, read the sections "Troubleshooting WebObjects 4.0 Template Parsing" (page 5), "Troubleshooting WebObjects 4.0 Request Handling" (page 7), and "WebScript Changes" (page 10). Among other things, these sections

outline the use of WebObjects 3.5 compatibility flags that allow you to revert to the style of template parsing and request handling that was performed in WebObjects 3.5.

6. At this stage, if you want, you can remove all usage of deprecated API. WebObjects has been rewritten to be thread-safe, which required deprecating some of the existing APIs. You can still use deprecated API as long as you do not enable multithreading in your application. You'll receive warnings about deprecated API at run-time. For a complete list of what's deprecated, see "Support for Multithreaded Applications" (page 17).

Converting Java Code

This section covers the details of converting any Java code you may have in an existing WebObjects application (see step 2 of "Converting an Existing WebObjects Application," above). In WebObjects 4.0, the Java APIs changed considerably. These changes are summarized here:

- A two-letter prefix has been added to each Java class name so that class names are unique without the package names. The Java class name is now identical to its Objective-C counterpart in almost all cases. For example, Component is now WOComponent, and WebApplication is now WOApplication.

- The Java package names have changed to the following:

  ```
  com.apple.yellow.eoaccess
  com.apple.yellow.eocontrol
  com.apple.yellow.foundation
  com.apple.yellow.webobjects
  ```

  Notice that the next.eo package has been split into two packages: eoaccess and eocontrol.

- The basic classes (for arrays, dictionaries, and data) have become more like their Foundation counterparts than their Java counterparts. For example, ImmutableVector is now named NSArray and responds to **count** instead of **size**. MutableHashtable is now named NSMutableDictionary and responds to **setObjectForKey** instead of **put**.

  Note that for numbers and strings, you still use the classes **java.lang.Number** and **java.lang.String**.

Warning: Changing to Foundation-style methods for the dictionary class introduces a subtle change. The Java Hashtable classes take the arguments in the key-value order. For example, the **put** method takes the key and then the value. NSDictionary takes the value and then the key. The conversion scripts change the order of the arguments for you. Unfortunately, these scripts incorrectly convert uses of **get()** and **put()** on java.util.Hashtable objects as well as on objects of other Foundation classes.

- DecimalNumber is no longer available. Use **java.math.BigDecimal** instead.
- CalendarDate is now named NSGregorianDate.
- The root object is now **com.apple.yellow.foundation.NSObject.**
- Delegate interfaces are now declared as inner interfaces of the appropriate class. For example, the DisplayGroupDelegates interface is now **WODisplayGroup.Delegates.**

Scripts are provided with the release to help you convert Java code to the new APIs. They are located in **/System/Developer/Java/Conversion/WebObjects** or, on NT, in **$(NEXT_ROOT)/Developer/Java/Conversion/WebObjects**. Descriptions of these scripts and instructions for their use can be found in the ReadMe file, which is located in the same directory as the script files.

Troubleshooting WebObjects 4.0 Template Parsing

The WebObjects template parser parses the HTML that is to be included in a response. In WebObjects 4.0, the template parser preserves all of the static HTML that you provide in a component's template. Previously, the parser recognized many HTML tags and performed special processing based on the type of tag. The 4.0 template parser ignores all tags besides **<WEBOBJECT>** and HTML comment tags.

The new parser has several advantages:

- It solves the problem many have encountered where WebObjects attempts to "fix" your HTML. For example, it previously was difficult to split a container element, such as a form, across two components because WebObjects would insert a closing tag for you.
- It improves your application's performance because it tends to treat larger parts of a file as a single chunk than the previous parser did.

- It allows you to suppress the copying of comments to the outgoing response. This speeds up response generation and shortens download times.

A WebObjects application may unknowingly depend upon the previous behavior of the template parser. For this reason, a compatibility flag is available on WOApplication to revert to the previous behavior.

Usually when 4.0 template parsing produces an error, it is because you have included a WebObjects dynamic form element inside of a static HTML **FORM** element. Change the **FORM** element to a WOForm, and your component should operate normally again. An error may also arise if your HTML pages contain **BODY** or **IMG** tags with **src** parameters containing relative pathnames (absolute pathnames aren't a problem). Change the affected tags to WOBody and WOImage, respectively.

If you want, you can go back to the previous parser by implementing this method in your application class (shown in Java and WebScript):

```
public boolean requiresWOF35TemplateParser() {
  return true;
}
- requiresWOF35TemplateParser {
  return YES;
}
```

If you use the WebObjects 4.0 template parser, you might want to suppress the inclusion of HTML comments. Use the following methods, which have been added to WOApplication (as an alternative, you can use the option described in the section "Command-Line Options" (page 12)):

WOApplication Template Parsing Methods.

WOApplication

Method	Description
setIncludeCommentsInResponses: (class or static method)	Sets whether the application's HTML parser includes comments from a component's HTML template as part of a response. The default is **YES** or **true**. Use this method only in the application's initializer or constructor.
includeCommentsInResponses (class or static method)	Returns **YES** or **true** if the HTML parser includes comments in the responses. Returns **NO** or **false** if the application doesn't include any comments from a component's HTML template in the response. The default is **YES** or **true**.

Troubleshooting WebObjects 4.0 Request Handling

In previous WebObjects releases, the application, session, request component, and all of the dynamic elements in the request component got a chance to perform the **takeValuesFromRequest:inContext:** and **invokeActionForRequest:inContext:** methods during each cycle of the request-response loop. In WebObjects 4.0, there are some performance enhancements to this request-handling scheme:

- The take values phase is not always performed.

  In release 4.0 if the request has no form values to use as input, the take values phase of the request-response loop (in which the application, the session, the request component, and the component's dynamic elements are sent **takeValuesFromRequest:inContext:**) is not performed.

  If you have overridden **takeValuesFromRequest:inContext:** at the application, session, or component level and your method needs to be invoked even when there are no input values, you must either change your logic or disable 4.0 request handling at the application level. To disable 4.0 request handling, implement the following method in your application class:

```
//WARNING! Put this method in Application class, not component.
//Java implementation
public boolean requiresWOF35RequestHandling() {
  return true;
}
//WebScript implementation
- requiresWOF35RequestHandling {
  return YES;
}
```

It is the application object that makes the decision to perform the take values phase of the request-response loop; therefore, you must disable 4.0 request handling in the application class if you want to ensure that the take values phase always occurs.

- The take values phase does not iterate through WOBrowser and WOPopUpButton lists.

  In previous releases, WebObjects would iterate through the **list** attribute of the WOBrowser and WOPopUpButton looking for the item that the user selected. In release 4.0, this is no longer necessary because WebObjects can directly access the selected item without iterating. WebObjects is able to do this because the use of the **value** attribute has changed so that by default it is set to the index of the item.

  - Use of the **item** attribute as the selection.

    The **item** attribute is intended to point to the current item, and it is updated upon each iteration through the list. Because WebObjects used to iterate through the list until it found a selection, the **item** attribute ended up pointing to the selected item.

    If you need to refer to the selected item, use the **selection** attribute instead of **item**. Make sure **selection** is bound to a variable in your component's code and then use that variable instead of the one bound to **item**.

  - Use of the **value** attribute.

    The **value** attribute was previously used as the string displayed in the browser or pop-up button. It also set the HTML value attribute for the **<OPTION>** tag. In WebObjects 4.0, this attribute still sets the value in HTML, but it no longer specifies the display string. By default, it is set to an index value, which allows WebObjects to find the selection without iterating through the list.

    If you have a binding for the **value** attribute, change it to **displayString**, which is a new attribute that specifies the display string. Change this:

    ```
    value = aCollege.name;
    ```

    to this:

    ```
    displayString = aCollege.name;
    ```

    Use **value** only if you really want to set the HTML value in the **<OPTION>** tag.

- An **item** attribute bound to a method.

  If you bound the **item** attribute to a method, your method used to be invoked several times during the take values phase, and now it is invoked only once (for the selected item). If your component depends upon the previous behavior, you either need to change your logic or use WOApplication's request-handling compatibility flag as described above.

- The invoke action phase does not iterate through a WORepetition's list.

  When a repetition's list is iterated upon, the **item** and **index** attribute values are updated at each iteration. In previous releases, list iteration occurred during the take values phase and during the invoke action phase of the request-response loop. In WebObjects 4.0, WORepetition list iteration occurs during take values only if the request has input values, and it doesn't occur during the invoke action phase. (WebObjects is able to forgo iterating during the invoke action phase because by default it sets the **identifier** attribute to the item's element ID so that it is able to navigate directly to the list item that responds to the requested action. If you already declare a binding for the **identifier** attribute, your binding is used instead of the element ID, and the invoke action phase does iterate through the list.)

  If you've bound the **item** or **index** attribute to a method, your method used to be invoked several times during the take values phase, and then again several more times during the invoke action phase. In WebObjects 4.0, your method will only be invoked during the take values phase if there are input values in the request, and it won't be invoked during the invoke action phase (unless you specify a non-default binding for the **identifier** attribute).

  If your component depends upon the previous behavior, you either need to change your logic or use the component's request-handling compatibility flag. To set the component's request handling compatibility flag, implement this method in the component:

```
// Java implementation
public boolean requiresWOF35RequestHandling() {
  return true;
}
// WebScript implementation
- requiresWOF35RequestHandling {
  return YES;
}
```

When you implement this method at the component level, WebObjects uses the old behavior for invoke action on that component only. All other components use the new behavior for the invoke action phase.

WebScript Changes

In WebObjects 3.5, WebScript would always evaluate both sides of an "&&" or "||" expression. In WebObjects 4.0, these expressions are now short-circuited, so that only the left side is evaluated unless evaluation of the right side is necessary in order to determine the result. For example:

```
(YES || <this will NOT evaluate>)
(NO  || <this will evaluate>)
(YES && <this will evaluate>)
(NO  && <this will NOT evaluate>)
```

To aid in the debugging process, WebObjects 4.0 has a WebScript 3.5 compatibility mode. This mode is controlled by a method in WOApplication named **requiresWOF35Scripting**. By default, this method returns NO; override it to return YES to get backward compatibility.

File Location Changes

WebObjects 4.0 is the first release of WebObjects that runs on Mac OS X Server and Yellow Box for Windows NT instead of OpenStep 4.2. Because of this change, the locations of WebObjects files have changed.

On Mac OS X Server, most WebObjects files are installed in the **System** folder (some can be found in **/Local**). On Windows NT, you still choose a folder in which to install the software, and the **NEXT_ROOT** environment variable points to that folder. The default has changed to **C:\Apple**.

The following table lists new directory names relative to the **System** folder or **NEXT_ROOT** and what each directory contains.

Location	Contains
Developer/Applications	Developer applications such as Interface Builder, EOModeler, Project Builder, and WebObjects Builder
Developer/Examples/EnterpriseObjects	EOModels and database installation scripts needed to run the WebObjects examples
Developer/Examples/WebObjects	WebObjects examples

Location	Contains
Developer/Examples/WebObjects/Java/ConversionScripts	Conversion scripts for Java APIs
Documentation/Developer	Developer documentation for Mac OS X Server, Yellow Box, Enterprise Objects Framework, and WebObjects.
Library/Frameworks	Public frameworks such as **WebObjects.framework**, **EOAccess.framework**, and so on
Library/Executables (Windows NT systems only)	Framework DLLs
Library/Java	Java packages for Yellow Box, WebObjects, and Enterprise Objects Framework
Library/WebObjects/Adaptors	Executables for the various WebObjects adaptors
Library/WebObjects/Applications	The Monitor application and PlaybackManager
Library/WebObjects/Configuration	Configuration files
Library/WebObjects/Executables	WODefaultApp and WOPlayBack

The following are new directories and files installed under the **Local** directory. On Mac OS X Server, the **Local** directory is at the root level. On Windows NT, it is under **NEXT_ROOT**.

Location	Contains
Local/Library/Frameworks	Location where you install custom frameworks
Local/Library/WebObjects/Applications	Location where you install WebObjects applications (**.woa** directories)
Local/Library/Webserver/CGI-Executables	The cgi-bin directory for the Apache web server (Mac OS X Server systems only). The WebObjects executable is installed here. On Windows NT, you specify your web server's cgi-bin directory at install time.
Local/Library/Webserver/Documents	The document root for the Apache web server (Mac OS X Server systems only) On Windows NT, you specify your web server's document root directory at install time.

Running an Application on WebObjects 4.0

There have been several changes to the way you start a WebObjects application:

- The command line options have changed.
- Autostarting is no longer supported.

- A web server is no longer required for development.
- The public configuration file format has changed.
- The WebObjects application URL format has changed.

Command-Line Options

WebObjects 4.0 uses the Foundation NSUserDefaults object to specify application command-line options. As a consequence, all options have been renamed. The following table lists the WebObjects 3.5 options with their new names.

Old Option	New Option	Description
-debug ON|OFF	-WODebuggingEnabled YES|NO	Sets whether the application prints messages to standard error during startup. By default, this option is enabled. WOApplication, WOComponent, and WOSession define a new **debugWithFormat:** method (**debugString** in Java). This method is similar to **logWithFormat:** except that it only prints messages if the **WODebuggingEnabled** option is on
-browser ON|OFF	-WOAutoOpenInBrowser YES|NO	Sets whether the application automatically opens a web browser window to the application's URL (starting up the browser if necessary). By default, this option is enabled.
-m ON|OFF	-WOMonitorEnabled YES|NO	Enables or disables monitoring. By default, this option is disabled. If this option is enabled and you manually start an application, the application tries to find a running Monitor.
-mhost *hostName* | subnet	-WOMonitorHost *hostName* | subnet	If the **WOMonitorEnabled** option is on and you use this option, the application tries to find a running Monitor on the machine named *hostName* instead of on the local machine. If **subnet** is used, the application looks for a running Monitor in its network subnet.
-c	-WOCachingEnabled YES|NO	Requests that the application cache component definitions (templates) instead of reparsing HTML and declaration files upon each new HTTP request. By default, this option is disabled.
-d *documentRoot*	None	You are no longer required to specify the document root.
-a *adaptorClass*	-WOAdaptor *adaptorClass*	The WOAdaptor class name. The default is now WOMultiThreadedAdaptor. See the section "Support for Multithreaded Applications" (page 17) for more information on WOMultiThreadedAdaptor.
-i *instanceNumber*	None	You are no longer required to specify instance numbers when load-balancing applications. The instance number is now private to the configuration file.

Old Option	New Option	Description
-p *portNumber*	-WOPort *portNumber*	The socket port used to connect to an application instance. Unlike previous versions of WebObjects, this option is independent of the adaptor option. A *portNumber* of -1 means use an arbitrary high port number; however, you cannot specify -1 as the value on the command line; to set the value to -1, you must use the **defaults** command.
-q *listenQueueSize*	-WOListenQueueSize *listenQueueSize*	The depth of the listen queue. The default has changed from 4 to 5.
None	-WOWorkerThreadCount *int*	Maximum number of worker threads for a multithreaded application. The default worker thread count is 8. Setting this count to 0 results in single-threaded (WebObjects 3.5-style) request dispatch.
None	-WOOtherAdaptors *plist*	Use this option to attach additional adaptors (other than the one specified by **-WOAdaptor**) to the application. The *plist* option is an array of dictionaries written in property list format.
None	-WOCGIAdaptorURL *path*	The absolute URL that points to the WebObjects CGI adaptor.
None	-WOApplicationBaseURL	The path from the web server's document root to the directory where your application (or project, if in rapid turnaround mode) resides. The default is "/WebObjects", but you may place your application anywhere under the document root. See "Rapid Turnaround Mode" (page 54) for more a complete discussion of this option.
None	-WOFrameworksBaseURL	The location of frameworks under your document root if you're using a web server. The default is **/WebObjects/Frameworks** (as it was in release 3.5). All frameworks that your application uses must be in this directory.
None	-NSProjectSearchPath *pList*	An array of paths in which your project directories are located. (The array is written in property list format.) The default is a single item: ".." If you specify this option, WebObjects looks in the locations you specify for a project that has the same name as the application or framework being loaded. If it finds a project, it uses the images, scripted components, and other resources from the project directory instead of from the application or framework's main bundle. This way, you can modify images and scripted components in your project and test them without having to rebuild the application.
None	-WOIncludeComments InResponses YES|NO	Sets whether the HTML parser includes comments from the components' HTML files in the responses. The default is YES. See "Troubleshooting WebObjects 4.0 Template Parsing" (page 5) for more information.

Old Option	New Option	Description
None	-WOSessionTimeOut *timeout*	Sets the timeout interval for sessions. By default, they now time out after 3600 seconds (in prior releases of WebObjects, sessions never timed out by default).

As with all user defaults, you can set them three ways: on the application's command line, using the **defaults** utility, or programmatically.

Be careful when setting options programmatically. Most options require knowledge of the environment in which the application runs, and the appropriate values change if you move the application to a different machine. For example, you should never set the **WOPort** option programmatically.

Autostarting

In WebObjects 4.0, applications can no longer be autostarted. To autostart an application in WebObjects 3.5, you typed a URL in the client browser, and the WebObjects adaptor would launch the appropriate application. This feature was intended primarily to ease debugging; it was not supported for deployment.

In WebObjects 4.0, you debug an application by launching it in the Project Builder launch panel. By default, the browser is launched automatically and shows the appropriate URL. Thus, autostarting is no longer necessary.

"Serverless" Applications

WebObjects 4.0 applications can receive HTTP requests directly. Previously, a web server had to be running to receive HTTP requests and to forward them through the WebObjects adaptor.

To run a WebObjects application when no HTTP server is present, you simply specify the number of the port where the application should receive requests using the **WOPort** option. By default, **WOPort** is -1, which assigns an arbitrary high port number to the application. Thus, if you specify no port number at all, you can still run your application without a web server.

This new feature has several advantages:

- You can debug applications on a machine that doesn't have a web server present.
- You don't have to install project directories under the document root to test them.

- Running without an HTTP server uses less memory on your development machine.
- The WebObjects example applications no longer need to be installed under the web server's document root. Instead they are installed under **Developer/Examples/WebObjects**.

Note that if you do want to use a web server to test WebObjects examples, you can still do so. Before you do, do a "make install" to install the example's web server resources (such as image files and Java client-side classes) in the document root, just as you do when installing a WebObjects application. If you put your application in a directory other than "WebObjects" under your document root, set the **WOApplicationBaseURL** option to the **.woa** directory's path relative to the document root (**WOApplicationBaseURL** is set to **/WebObjects** by default). If you don't perform these steps, the web server won't be able to find web server resources; when you run the application, you'll see broken images, and client-side classes won't be loaded. (See "Rapid Turnaround Mode" (page 54) for more on developing with and without a web server.)

Changes to Adaptor Configuration Files

The format of the private and public configuration files (**WebObjects.conf**) has changed slightly. Along with this format change come two conceptual changes:

- Entering the path to the WebObjects adaptor in the browser (usually **http://localhost/cgi-bin/WebObjects**) used to provide a list of applications you had installed under the document root. Selecting one of these applications took you to it, autostarting the application if necessary.

  Because the adaptor can no longer autostart applications, entering the path to the adaptor now takes you to a list of already running applications.
- You no longer specify the application instance number on the command line. The instance number and the host name are now private to the **WebObjects.conf** file.

If a client browser tries to access an application instance that isn't running, the adaptor attempts to perform load-balancing with all existing instances. For example, suppose a user bookmarked a URL that contained 4 as the instance number and tried to use that bookmark two weeks later when instance 4 is no longer running. Instead of failing, WebObjects simply uses one of the instances that is running.

The Application Instance Number

For security reasons, an application instance’s host name and port number cannot be visible either in a page or in a URL. There is a one-to-one mapping between these values and an application’s instance number in the **WebObjects.conf** file; this mapping is resolved by the adaptor. Because it is a one-to-one mapping, the instance number must be unique across a deployment environment for a given application name.

When the adaptor load-balances a request to an application at a given port and host, it assigns an application instance number and places it into the request . (Because of this, you now obtain the application instance number from the WORequest rather than from the WOApplication object.) Application instances take the instance number from the request and send that number back in the response URLs. As far as the application instance is concerned, the instance number could change with every request—the application instance wouldn’t notice.

Changes to WebObjects Application URL

The typical WebObjects application URL now has the following format:

```
http://host[:port]/cgi-bin/WebObjects/App[[.woa][/instance]/key/...
```

where the variables are defined as follows:

Variable	Description
host	The host name of your computer or **localhost**.
port	The port number. This is included if you want to direct connect.
cgi-bin	The cgi-bin directory of your server, usually **cgi-bin** or **Scripts**.
WebObjects	The name of the CGI adaptor, usually **WebObjects** or **WebObjects.exe**.
App	The application name. This field is no longer the path to the application relative to ***DocRoot*/WebObjects**. It is simply the application name. The **WOApplicationBaseURL** option provides the path.
instance	The application instance number.
key	The request handler key. This key specifies which WORequestHandler object should be used to process the request. The WORequestHandler class is new in WebObjects 4.0 and is described in the section “WORequestHandler Class” (page 29).

Variable	Description
...	Information specific to the request handler. Each WORequestHandler uses a different format for the rest of the URL. The two main request handlers are WOComponentRequestHandler and WODirectActionRequestHandler. WOComponentRequestHandler handles requests in exactly the same manner in which they were handled in earlier releases. Its format for the rest of the URL is: *componentName/sessionID/elementID* WODirectActionRequestHandler handles direct actions, a new feature in WebObjects 4.0. (You can read more about this feature in "Direct Actions" (page 23).) Its URLs have this format: [*actionClass* | *actionName* | *actionClass/actionName*][?*key*=*value*&*key*=*value*.....]

Support for Multithreaded Applications

In release 4.0, WebObjects and Enterprise Objects Framework provide thread-safe APIs. This means that you can write a multithreaded WebObjects application where you couldn't before.

To support multithreaded applications, there are two major changes:

- The default adaptor has been rewritten to support multithreaded request handling. By default, this adaptor does multithreaded adaptor I/O and resource handling, but only single-threaded request handling. If you rewrite your application so that it is thread safe, locking any access to shared resources as necessary, you can enable concurrent request handling. To rewrite your application to be thread safe, you'll also need to remove all invocations of some deprecated API; see "Deprecated API," below, for more information. To enable concurrent request handling, override the **allowsConcurrentRequestHandling** method in your application class to return **YES** or **true.**

  The default adaptor can be made to operate in single-threaded mode by setting the **WOWorkerThreadCount** command-line option to 0.

- Certain method names have changed. You can still use the old methods, but their use is deprecated.

For information on multithreading Enterprise Objects Framework operations, see "What's New in Enterprise Objects Framework 3.0." The rest of this section lists the methods that have changed and describes new methods that you can use to ensure that your application is thread-safe.

Deprecated API

The following tables list methods whose use is deprecated in WebObjects 4.0 and list new methods to use in their place. If you don't allow concurrent request handling, you can continue to use the deprecated methods. You'll receive a warning at run-time. If you want to allow concurrent request handling, you must change to the new methods. Use of deprecated methods raises an exception when concurrent request handling is enabled (that is, when you override WOApplication's method **allowsConcurrentRequestHandling** to return **YES** or **true**.)

WOApplication

Old API	New API
session	WOComponent **session**
context	WOComponent **context**
pageWithName:	WOComponent **pageWithName:** WOApplication **pageWithName:inContext:** or **pageWithName:forRequest:** (Objective-C) WOApplication **pageWithName** (Java)
handleSessionCreationError	**handleSessionCreationErrorInContext**:
handleSessionRestorationError	**handleSessionRestorationErrorInContext:**
handlePageRestorationError	**handlePageRestorationErrorInContext:**
handleException:	**handleException:inContext:** (Objective-C) **handleException** (Java)
pathForResourceNamed:ofType:	WOResourceManager **pathForResourceNamed:inFramework:languages:** (Objective-C) WOResourceManager **pathForResourceNamed** (Java)
urlForResourceNamed:ofType:	WOResourceManager **urlForResourceNamed:inFramework:languages:request:** (Objective-C) WOResourceManager **urlForResourceNamed** (Java)
stringForKey:inTableNamed:withDefaultValue:	WOResourceManager **stringForKey:inTableNamed:withDefaultValue:languages:** (Objective-C) WOResourceManager **stringForKey** (Java)
handleRequest:	**dispatchRequest:**
createSession	**createSessionForRequest:**
restoreSession	**restoreSessionWithID:inContext:** (Objective-C) **restoreSessionWithID** (Java)

WOApplication

Old API	New API
restorePageForContextID:	WOSession **restorePageForContextID:**
savePage:	WOSession **savePage:**
saveSession:	**saveSessionForContext:**
dynamicElementWithName:associations: template:	**dynamicElementWithName:associations:template:languages:** (Objective-C) **dynamicElementWithName** (Java)
isBrowserLaunchingEnabled	**autoOpenInBrowser**
setBrowserLaunchingEnabled:	Use the **WOAutoOpenInBrowser** command-line option (see "Command-Line Options" (page 12))
runLoop	**mainThreadRunLoop**

WOAssociation

Old API	New API
value	**valueInComponent:**
setValue:	**setValue:inComponent:** (Objective-C) **setValue** (Java)

WOContext

Old API	New API
setDistributionEnabled:	WOSession **setDistributionEnabled:**
isDistributionEnabled	WOSession **isDistributionEnabled**
application	WOApplication **application** class or static method
urlSessionPrefix	None (deprecated functionality)
url	**componentActionURL**

WORequest

Old API	New API
applicationHost	See NSProcessInfo or NSHost (Objective-C) **java.net.InetAddress** (Java)
contextID	WOContext **contextID**
pageName	WOComponent **name**

WORequest

Old API	New API
senderID	WOContext **senderID**

WOResourceManager

Old API	New API
pathForResourceNamed:inFramework:	**pathForResourceNamed:inFramework:languages:** (Objective-C) **pathForResourceNamed** (Java)
urlForResourceNamed:inFramework:	**urlForResourceNamed:inFramework:languages:request:** (Objective-C) **urlForResourceNamed** (Java)

WOSession

Old API	New API
application	WOApplication **application** class or static method

WOSessionStore

Old API	New API
restoreSession	**restoreSessionWithID:request:** (Objective-C) **restoreSessionWithID** (Java)
saveSession:	**saveSessionForContext:**
cookieSessionStoreWithDistributionDomain:secure:	None. See "Cookie API" (page 41) for new API that allows you to store session IDs in cookies.

WOStatisticsStore

Old API	New API
validateLogin:	**validateLogin:forSession:** (Objective-C) **validateLogin** (Java)
setMovingAverageSampleSize:	**setTransactionMovingAverageSampleSize:** **setSessionMovingAverageSampleSize:**
movingAverageSampleSize	**transactionMovingAverageSampleSize** **sessionMovingAverageSampleSize**

WOComponent

Old API	New API
pathForResourceNamed:ofType:	WOResourceManager **pathForResourceNamed:inFramework:languages:** (Objective-C) WOResourceManager **pathForResourceNamed** (Java)
urlForResourceNamed:ofType:	WOResourceManager **urlForResourceNamed:inFramework:languages:request:** (Objective-C) WOResourceManager **urlForResourceNamed** (Java)
stringForKey:inTableNamed:withDefaultValue:	WOResourceManager **stringForKey:inTableNamed:withDefaultValue:languages:** (Objective-C) WOResourceManager **stringForKey** (Java)
templateWithHTMLString:declarationString:	**templateWithHTMLString:declarationString:languages:** (Objective-C) **templateWithHTMLString** (Java)

WODisplayGroup

Old API	New API
setSortOrdering	**setSortOrderings:**
sortOrdering	**sortOrderings**
endEditing	None (this method had no effect in WebObjects 3.5)
executeQuery	**queryMatch**, **queryMin**, **queryMax**
inputObjectForQualifier	**queryMatch**, **queryMin**, **queryMax**
secondObjectForQualifier	**queryMatch**, **queryMin**, **queryMax**
setBuildsQualifierFromInput:	**queryMatch**, **queryMin**, **queryMax**
buildsQualifierFromInput	**queryMatch**, **queryMin**, **queryMax**
qualifierFromInputValues	**queryMatch**, **queryMin**, **queryMax**
lastQualifierFromInputValues	**queryMatch**, **queryMin**, **queryMax**
localKeys	None (this method was inadvertently carried over from EODisplayGroup)
setLocalKeys:	None (this method was inadvertently carried over from EODisplayGroup)

New Methods to Support Multithreading

To support the writing of multithreaded applications, the following methods have been added to the following classes:

WOApplication

Method	Description
adaptorsDispatchRequestsConcurrently	Returns **YES** or **true** if at least one adaptor contains multiple threads and will attempt to concurrently invoke the request handlers.
allowsConcurrentRequestHandling	Specifies whether the application has been written to be able to safely handle concurrent requests. By default, returns **NO** or **false**. Subclasses should override to return **YES** or **true** if the application is written to be thread-safe.
isConcurrentRequestHandlingEnabled	Returns **YES** or **true** if the application can handle concurrent requests and the adaptor will dispatch requests to the application concurrently.
lockRequestHandling	Serializes request handler access through the use of an internal lock if concurrent request handling is disabled. Request handlers should invoke this method before calling into application code.
unlockRequestHandling	Removes the request handling internal lock to allow access to the request handler again.
lock	Locks access to the WOApplication object.
unlock	Unlocks access to the WOApplication object.

WORequestHandler

Method	Description
lock	Locks access to the WORequestHandler object
unlock	Unlocks access to the WORequestHandler object

WOResourceManager

Method	Description
lock	Locks access to the WOResourceManager object.
unlock	Unlocks access to the WOResourceManager object.

WOSessionStore

Method	Description
checkOutSessionWithSessionID:request: (checkOutSessionWithSessionID in Java)	Checks out a session for exclusive use. When you check a session out, all other access to that session is blocked until the session is checked in again.

WOSessionStore

Method	Description
checkInSessionForContext:	Checks in a session.

WOStatisticsStore

Method	Description
lock	Locks access to the WOStatisticsStore object.
unlock	Unlocks access to the WOStatisticsStore object.

Direct Actions

Previously, all WebObjects applications used the same request-handling scheme: the request to perform an action is passed from the application to the session to the request component. The request component is the component that generated the response for the previous request. Thus, the component that generates the response for one request must be preserved so that it can perform the next requested action. Because components had to be preserved across cycles of the request-response loop, all applications were required to keep some session state.

In WebObjects 4.0, you can set up all or part of your application to handle direct actions. With direct actions, the action is sent directly to an object that can handle it. Direct actions have several advantages over component actions:

- Direct actions have simpler, static URLs. Your users can bookmark a direct action URL and return to it at any time.
- Direct actions have simpler request handling.
- By default, direct actions don't use session objects and thus don't store state. If you are writing a stateless application, you may find it easier to frame your application logic using direct actions instead of component actions.

How Direct Action Requests Are Sent

Dynamic elements that have an **action** attribute are bound to component actions. Dynamic elements that have a **directActionName** attribute are bound to direct actions. The list of dynamic elements bound to direct actions includes

WOActiveImage, WOForm, WOFrame, WOHyperlink, WOImageButton, and WOSubmitButton.

When you create a WebObjectsApplication project in release 4.0, a subclass of WODirectAction (a new class in WebObjects 4.0 that is a container for action methods) named "DirectAction" is created for you (along with the WOApplication subclass named "Application" and the WOSession subclass named "Session"). "DirectAction" is the default name for a WODirectAction subclass, and can be renamed if you prefer. You can create several WODirectAction subclasses each performing a single action or a set of actions, or you can have a single WODirectAction subclass perform all of the actions.

For example, the declaration for a WOHyperlink that triggers a direct action might look like this:

```
myLink: WOHyperlink {
  actionClass = "MyActions";
  directActionName = "logout";
}
```

The **actionClass** parameter specifies a subclass of WODirectAction (it defaults to "DirectAction" if omitted). The **directActionName** should refer to an action name; if omitted, WebObjects invokes the method **defaultAction** within the specified class. Method names are derived from action names by appending "Action" to the action name; thus, a **directActionName** of "logout" corresponds to the **logoutAction** method.

Putting Values into a WORequest

WebObjects 4.0 allows you to set arguments for an action as follows:

```
myLink : WOHyperlink {
  directActionName = "display";
  queryDictionary = arguments;
  ?sku = currentProduct.sku;
}
```

The **queryDictionary** attribute is set to an NSDictionary that contains arguments for the **displayAction** method. The keys in this dictionary are variables in the action method. The **sku** argument is an additional argument for the **displayAction** method and is an alternate way of setting arguments for the action.

Note: Although the above example uses a direct action, use of the **queryDictionary** and the "?" binding aren't limited to direct actions: you can use them any time you need to put a value into a WORequest.

Suppressing Session IDs in a Direct Action URL

When you construct an HTML template that has some of its components bound to direct actions and some bound to component actions, depending on the placement of your direct action components their URLs may include session IDs. You can prevent the inclusion of a session ID in a direct action URL as shown in the following example:

```
MyLink:WOHyperlink {
  directActionName = "something";
  ?wosid = NO;
}
```

How Direct Action Requests Are Received

Clicking the WOHyperlink from the previous section generates a URL that looks something like this:

```
http://localhost/cgi-bin/WebObjects/AppName.woa/wa/
display?sku=value&aKey=aValue...
```

The **wa** after the application name is a request handler key. It specifies which WORequestHandler should handle the request. WORequestHandler is a new class in WebObjects 4.0. You can read more about it under "WORequestHandler Class" (page 29). The **wa** string is the key for the WODirectActionRequestHandler, a private subclass of WORequestHandler.

In WebObjects 4.0, when the WOApplication receives a request from the WOAdaptor, it looks at the request handler key to determine which WORequestHandler should handle the request. It then sends that WORequestHandler a **handleRequest:** message.

If the URL doesn't have a request handler key (as is the case with the initial URL used to begin a session with a WebObjects application), WOApplication uses whatever its default request handler is set to be. By default, the default request handler is WOComponentRequestHandler, which performs the request handling scheme that you're used to. If you want to write an application entirely using direct actions, set the default request handler in your WOApplication's **init** method or constructor in this way:

```
// Java implementation
public WOApplication() {
  super();
  ...
  setDefaultRequestHandler(requestHandlerForKey(
      WOApplication.directActionRequestHandlerKey()));
  ...
}
```

```
//WebScript implementation
- init {
  self = [super init];
  ...
  [self setDefaultRequestHandler:[self requestHandlerForKey:
      [WOApplication directActionRequestHandlerKey]]];
  ...
  return self;
}
```

If WODirectActionRequestHandler is the default request handler, the first request triggers the **defaultAction** method, which is declared for you in your DirectAction class.

In its implementation of **handleRequest:**, WODirectActionRequestHandler extracts the direct action class and the action from the URL. (If your WODirectAction subclass isn't named DirectAction, the class name appears in the URL immediately before the action.) WODirectActionRequestHandler then sends the message **performActionNamed:** to your WODirectAction object.

Each action method in your WODirectAction class should end with the string "Action" and should return either a WOComponent or a WOResponse object. For example:

```
- (WOComponent *)displayAction
```

There's a new protocol and interface named WOActionResults conformed to by WOResponse and WOComponent. Your action may actually return any object that conforms to WOActionResults.

When the action method returns, WODirectActionRequestHandler sends the message **generateResponse** to the object returned by the action method. This is the method defined in the WOActionResults protocol. **generateResponse** returns a WOResponse object. WOResponse's implementation is simply to return itself. WOComponent's implementation translates the component into a WOResponse by sending itself **appendToResponse:inContext:**.

Note: WOComponent's **generateResponse** method is also useful for the **handleException...** methods defined in WOApplication.

Upon receiving the WOResponse, WODirectActionRequestHandler returns the response to the WOApplication, and the WOApplication passes it to the WOAdaptor.

Comparison of Request Processing

The following table shows the sequence of events in processing a traditional, component action request and compares it to the sequence of events for processing a new direct action. Note that in both component actions and direct actions, the bulk of the time is spent in the generate response phase, in which the component performs **appendToResponse:inContext:** and sends each of its dynamic elements **appendToResponse:inContext:**. This step is the same in component actions and direct actions.

Component Action	Direct Action
The adaptor creates a WORequest object and passes it to the application.	The adaptor creates a WORequest object and passes it to the application.
The application determines that the WOComponentRequestHandler should handle the request.	The application determines that the WODirectActionRequestHandler should handle the request.
The application, session, and the request component are created, if necessary, and sent the **awake** message.	Application **awake** is called.
The **takeValuesFromRequest:inContext:** message is propagated from the application to the session to the request component to each dynamic element in the request component (if the request has input values).	WODirectActionRequestHandler parses the URL and instantiates the WODirectAction class.
The **invokeActionForRequest:inContext:** message is propagated from the application to the session to the request component to each dynamic element in the request component, resulting in the appropriate action method in the component being invoked.	WODirectActionRequestHandler sends the message **performActionNamed:** to the WODirectAction, resulting in the appropriate action being invoked. If there are any input values, WODirectAction uses **takeFormValues...** methods to extract them from the WORequest.
The **action** method creates and returns a response component or response.	The **action** method creates and returns a response component or response.
The application awakens the response component. The **appendToResponse:inContext:** message is propagated from the application to the session to the response component to each dynamic element in the response component.	The object returned by the action method is sent a **generateResponse** method to guarantee that the object returned is a WOResponse. If the action returns a WOComponent, WOComponent's **generateResponse** invokes **appendToResponse:inContext:**, which sends each dynamic element in the component an **appendToResponse:inContext:** message as well.
The application forwards the WOResponse to the adaptor.	The application forwards the WOResponse to the adaptor.

Component Action	Direct Action
The application, session, and all of the components are sent the **sleep** message.	The WODirectAction is release or marked for garbage collection. Application **sleep** is called.
The component is saved in the session so it can handle any subsequent requests.	If the returned component contained any component actions, the component is saved in the session so it can handle any subsequent requests.

API for Direct Actions

This section describes the new classes, methods, protocols, and interfaces added to support direct actions.

WODirectAction Class

The main purpose of WODirectAction is to act as a repository for action methods. WODirectAction also defines these methods, which you can use in your actions:

WODirectAction

Method	Description
initWithRequest: (Objective-C) WODirectAction (Java)	Subclasses must override to provide any additional initialization.
request	Returns the WORequest object that initiated the action.
session	Returns the current session. If there is no session, which is a possibility if the application is written entirely with direct actions, this method creates a new session before returning it.
existingSession	Attempts to restore and then return the session based upon the request. If the request didn't have a session ID or the session ID referred to a non-existent session, this method returns **nil** (**null** in Java).
pageWithName:	Creates and returns an instance of WOComponent with the specified name.
takeFormValuesforKeyArray:	Extracts input values from the request URL and assigns them to the WODirectAction instance using **takeValue:forKey:**. The argument is an NSArray of keys.
takeFormValuesForKeys: (Objective-C only)	Extracts input values from the request URL. The argument is a comma-separated list of NSStrings.
takeFormValueArraysForKeyArray:	Extracts input values from the request URL where the values are arrays. The argument is an NSArray of keys.
takeFormValueArraysForKeys: (Objective-C only)	Extracts input values from the request URL where the values are arrays. The argument is a comma-separated list of NSStrings.
performActionNamed:	Performs the action with the specified name and returns the result of that action.

WOActionResults Protocol and Interface

WOActionResults is an Objective-C protocol and Java interface that is now adopted by WOResponse and WOComponent. It defines one method:

WOActionResults

Method	Description
generateResponse	Returns a WOResponse object. WOResponse's implementation simply returns itself. WOComponent creates a WOResponse object by sending itself the **appendToResponse:inContext:** message.

WOActiveImage, WOForm, WOFrame, WOHyperlink, WOImageButton, WOSubmitButton

All elements that support direct actions have the following new attributes:

Attribute	Description
actionClass	Specifies the WODirectAction subclass that contains the action named in the **directActionName** attribute. The **actionClass** attribute defaults to "DirectAction" if omitted.
directActionName	Specifies the action to invoke when this element is activated. The name of the corresponding method that is invoked is determined by appending "Action" to the **directActionName** (for example, the "display" direct action corresponds to the "displayAction" method). The **directActionName** attribute defaults to "defaultAction" if omitted.

WORequestHandler Class

A WORequestHandler is an object that can handle requests received by the WebObjects application server. The WORequestHandler class defines three methods.

WORequestHandler

Method	Description
handleRequest:	Request handlers must implement this method and perform all request-specific handling. By default, a request is an HTTP request. You must supply your own server-side adaptor to accept anything other than HTTP.
lock	Locks access to the WORequestHandler object.
unlock	Unlocks access to the WORequestHandler object.

A WORequestHandler class must be registered with the WOApplication object before it can be used. When you register a WORequestHandler, you specify a key for that handler, which is used in the URL. This key can be any alphanumeric string, but must contain at least one letter.

WOApplication Methods

The following methods have been added to WOApplication to support the use of WORequestHandler objects:

WOApplication

Method	Description
registerRequestHandler:forKey: (Objective-C) registerRequestHandler (Java)	Adds a new WORequestHandler to the list of request handlers. The key is a string that will be used in the URL to indicate which request handler should process the request. The key can be any string of letters and numbers, but it must include at least one letter.
removeRequestHandlerForKey:	Removes a WORequestHandler from the list of request handlers.
defaultRequestHandler	Returns the request handler that is used when the request URL doesn't contain a request handler key. This typically happens only on the first request.
setDefaultRequestHandler:	Sets the request handler that should be used when the request URL doesn't contain a request handler key. If you don't use this method, the default request handler is WOComponentRequestHandler, which handles requests routed through a component.
handlerForRequest:	Returns the WORequestHandler that can handle the current request, determined by the request handler key in the URL. That handler is returned and is subsequently sent the message **handleRequest:**, where all request-specific processing is done.
registeredRequestHandlerKeys	Returns an array of request handler keys that have been registered with the application.
setComponentRequestHandlerKey: (class or static method)	Sets the key used to indicate the WOComponentRequestHandler, which handles the traditional WebObjects URL containing a component name, session ID, and context ID. The default is "wo".
setDirectActionRequestHandlerKey: (class or static method)	Sets the key used to indicate the WODirectActionRequestHandler, which handles the WODirectAction-style URLs containing an action name. The default is "wa".
setResourceRequestHandlerKey: (class or static method)	Sets the key used to indicate the WOResourceRequestHandler, which handles resource requests, such as requests for images. The default is "wr".
dispatchRequest:	Determines which request handler should handle the request and then sends that request handler a **handleRequest:** message. This method determines which WORequestHandler should handle the request by looking up the request handler key in the URL.

WORequest Methods

The following methods have been added to WORequest to support the use of WORequestHandler objects:

WORequest

Method	Description
requestHandlerPathArray	Returns an array containing the portion of the URL following the request handler key, up to the “?”, if present. Each part of the string separated by a “/” is stored in a separate element of the returned array.
requestHandlerPath	Returns the portion of the URL following the request handler key, up to the “?”, if present.
requestHandlerKey	Returns the part of the request’s URL that identifies the request handler key. The returned key identifies a request handler for the receiving request.

New Notifications

WOApplication now declares two notifications:

- WOApplicationWillFinishLaunchingNotification
- WOApplicationDidFinishLaunchingNotification

Objects can observe one or both of these notifications to add WORequestHandlers to WOApplication’s list of request handlers. Alternatively, you can register handlers in your application subclass’s **init** method or constructor. However, if you define a request handler inside of a framework, your framework’s class should observe this notification and register itself when the notification is received.

WOSession now declares the following notifications:

- WOSessionDidTimeOutNotification
- WOSessionDidRestoreNotification
- WOSessionDidCreateNotification

WOContext Changes

The following methods have been added to WOContext to return information from the request URL:

WOContext

Method	Description
directActionURLForActionNamed:queryDictionary: (Objective-C) directActionURLforActionNamed (Java)	Returns the complete URL for the specified action.
componentActionURL	Returns the complete URL for the component action.
urlWithRequestHandlerKey:path:queryString: (Objective-C) urlWithRequestHandlerKey (Java)	Returns a URL relative to **cgi-bin/WebObjects**.
completeURLWithRequestHandlerKey:path:queryString:isSecure:port: (Objective-C) completeURLWithRequestHandlerKey (Java)	Returns the complete URL for the specified request handler.

Improved Nested Component Support

In WebObjects 4.0, support for nested, reusable components has been improved in these ways:

- Template parsing improvements make HTML generation for components faster. Thus, you'll see only a small performance loss by using a component instead of a dynamic element.
- You can now create a nested component that serves as an HTML container element, one that wraps other HTML and text inside of it (similar to the way a WORepetition wraps other HTML elements).
- You can turn off component synchronization, in which values are pulled from the parent component and pushed to the parent component before and after each phase of the request-response loop, and perform synchronization manually. When you perform synchronization manually, components are more predictable and behave more like dynamic elements.
- It's now easier to use components to mimic and customize the behavior of dynamic elements. Because of the performance improvements and the ability to define non-synchronized components, you shouldn't find it necessary to have to write a subclass of WODynamicElement.

- WOComponent now has a **parent** method that returns the receiver's parent WOComponent.

"Container" Components (WOComponentContent)

WOComponentContent is a dynamic element that allows you to write nested components as HTML container elements. A container element is an element that can include text and other elements between its opening and closing tags. For example, the HTML FORM element is a container element. As well, WORepetition is a container element.

Using WOComponentContent you can, for example, write a component that defines the header and footer for all of your application's pages. To do so, you'd define a component with HTML similar to the following:

```
<HTML>
  <HEAD>
      <TITLE>Cool WebObjects App</TITLE>
  </HEAD>
  <BODY>
      <!-- A banner common to all pages here -->
      <!-- Start of content defined by the parent element -->
      <WEBOBJECT name=ParentContent></WEBOBJECT>
      <!-- End of content defined by the parent element -->
      <!-- Put a footer common to all pages here. -->
  </BODY>
</HTML>
```

The **<WEBOBJECT>** element on this page is a WOComponentContent element declared like this:

```
ParentContent : WOComponentContent {};
```

WOComponentContent is simply a marker that specifies where the contents wrapped by this component's **WEBOBJECT** tag should go. You can have only one WOComponentContent element in a given component.

To use the component shown above, you'd wrap the contents of all of the other components in the application with a **<WEBOBJECT>** tag that specifies the component defined above. For example, suppose you named the above component **HeaderFooterPage.wo**. You could use it in another component like this:

```
<!-- HTML for a simple component wrapped with HeaderFooterPage -->
<WEBOBJECT name = templateWrapperElement>
  <P>Hello, world!</P>
</WEBOBJECT>
```

Where **templateWrapperElement** is declared in the **.wod** file like this:

```
templateWrapperElement : HeaderFooterPage {};
```

At run-time, the contents wrapped by **templateWrapperElement** are substituted for the WOComponentContent definition. As a result, the HTML generated for this component would be:

```
<!-- HTML for a simple component wrapped with HeaderFooterPage -->
<HTML>
  <HEAD>
      <TITLE>Cool WebObjects App</TITLE>
  </HEAD>
  <BODY>
      <!-- A banner common to all pages here -->
      <!-- Start of content defined by the parent element -->
      <P>Hello, world!</P>
      <!-- End of content defined by the parent element -->
      <!-- Put a footer common to all pages here. -->
  </BODY>
</HTML>
```

Non-Synchronizing Components

By default, a nested component pulls all values from its parent and pushes all values to its parent before and after each phase of the request-response loop. This can lead to problems where values are being set when you don't want them set. In addition, the reusability of components is diminished if you must pre-compute everything a nested component needs before using it inside of another component.

The solution to both of these problems is non-synchronizing components. When components are not synchronized, they behave more like dynamic elements in that values are not pushed or pulled until they are needed.

To create a non-synchronizing nested component, do the following:

- Override the **synchronizesVariablesWithBindings** method to return **NO** or **false**.
- Use these two methods to push and pull values:

WOComponent

Method	Description
valueForBinding:	Gets (pulls) the value that the parent component bound to the specified attribute.
setValue:forBinding: (Objective-C) setValueForBinding (Java)	Sets (pushes) the value of the variable that the parent component bound to the specified attribute to the specified value.

For example, consider a nested component named NonSyncComponent that you declare in a parent component in this way:

```
//parent component's .wod file
MySubcomponent : NonSyncComponent {
  stringValue = @"I'm a string!";
}
```

Suppose NonSyncComponent contains a WOString element that it declares in this way:

```
// NonSyncComponent.wod
MyString : WOString {
  value = someStringValue;
}
```

If NonSyncComponent's script file looks like the following, the value that the parent bound to the **stringValue** attribute is pushed and pulled to WOString's **value** attribute whenever WOString requests it. Thus, the WOString in this NonSyncComponent displays "I'm a string!"

```
// NonSyncComponent.wos
- synchronizesVariablesWithBindings {
  return NO;
}

- someStringValue {
  return [self valueForBinding:@"stringValue"];
}

- setSomeStringValue:aValue {
  [self setValue:aValue ForBinding:@"stringValue"];
}
```

If NonSyncComponent has no other need for **someStringValue** than to resolve the **value** attribute for its WOString, then NonSyncComponent can instead use this shorthand notation in its declarations file:

```
// Alternate NonSyncComponent.wod
MyString : WOString {
  value = ^stringValue;
}
```

The carat (^) syntax means "use the value that my parent bound to my **stringValue** attribute to resolve **value**." This syntax is a convenience that saves you from having to always write cover methods for **valueForBinding:** and **setValue:forBinding:**. In addition to being more convenient, this syntax is often more efficient because none of your code is invoked to do either the pushing or the pulling.

Components That Mimic Dynamic Elements

It's common to want to be able to subclass a particular dynamic element to provide behavior specific to your application. Creating a true subclass of a particular dynamic element can be a difficult task. In WebObjects 4.0, however, you can do this much more easily. Instead of subclassing the dynamic element class, you write a reusable component that mimics the behavior of the dynamic element and provides your own custom behavior.

To learn how to write components that mimic dynamic elements, see the MinimalPrimitivesTest example application. You can use this example application as a starting point for writing your own components.

The MinimalPrimitivesTest example uses some new WOGenericElement and WOGenericContainer attributes that make it easy to use these two elements to define other dynamic elements. The new attributes on WOGenericElement and WOGenericContainer are listed below:

WOGenericElement and WOGenericContainer Attributes

Attribute	Description
elementName	The name of the HTML element you want to create. This attribute isn't new, but it has some changes. It is now optional. You can now bind this attribute to a variable instead of a constant string. You can also set the value of this attribute to **nil** or **null**, which effectively shuts this element off (that is, WebObjects doesn't generate HTML tags for this element).
formValue	Bind this attribute to a variable that should contain the component's input value. If this attribute is specified, WebObjects extracts the form value with the key matching this component's element ID from the request and assigns it to the variable bound to this attribute. If the request has no form value for this element, the variable isn't set.
formValues	Same as **formValue**, but should be bound to an array variable. Use this attribute if the element should receive more than one input value. (For example, a WOBrowser with multiple selections enabled could receive more than one input value.)
invokeAction	The action to be invoked. Use this attribute to simulate elements like WOHyperlink, WOSubmitButton, and so on, that define an action.
omitTags	If **YES**, WebObjects doesn't generate HTML tags for this element. This attribute allows you to define an element that conditionally wraps HTML in a container tag. For example, if the **elementName** attribute is bound to the string "B" and the **omitTags** attribute is bound to a boolean variable, you can use that boolean variable to set whether this element generates the bold tag or not. (The body contained in the element is generated regardless of this setting.)
elementID	Bind this attribute to a variable if you want to obtain programmatic access to the element ID for this element.

Improved Image Loading

To make it easier to display images from a database, the following attributes have been added to WOActiveImage, WOImage, WOImageButton, WOFrame, WOBody, and WOEmbeddedObject.

Attribute	Description
data	An NSData object containing the image or embedded object.
mimeType	The MIME type of the image to be put in the content-type header field.
key	The key under which the data is stored in an application-wide cache. If the key is already in the image cache table, then the value isn't computed again. This attribute is optional; the default is a random key, which means the data will be removed from the cache after access.

Images are cached by the WOResourceManager object. The following are new methods on WOResourceManager that access the image cache:

WOResourceManager

Method	Description
flushDataCache	Removes all data from the image data cache.
setData:forKey:mimeType:session: (Objective-C) setData (Java)	Adds image data of the specified type to the data cache with the specified key.
removeDataForKey:session: (Objective-C) removeDataForKey (Java)	Removes the data for the specified key from the image data cache. The session argument is ignored.

New Methods

This section details various new methods added to the classes that make up the WebObjects Framework.

WOAdaptor

Method	Description
runOnce	The main application run loop invokes this method for each iteration through the request-response loop. This method is where adaptors should do the bulk of their work.

WOAdaptor

Method	Description
doesBusyRunOnce	An adaptor should implement this method to return whether repeatedly invoking **runOnce** would result in busy waiting.
dispatchesRequestsConcurrently	An adaptor should implement this method to return **YES** or **true** if the adaptor contains multiple threads and concurrently invokes request handlers.

WOApplication

Method	Description
handleInitialTimer	Initial timer callback method.
cancelInitialTimer	Cancels the initial timer.

WOComponent

Method	Description
validationFailedWithException:value:keyPath: (Objective-C) validationFailedWithException (Java)	Called when an enterprise object or a formatter failed validation during an assignment. The default implementation of this method ignores the error. Subclassers can override it to record the error and possibly return a different page for the current action.
hasSession	A convenience method that returns **YES** or **true** if there is a current session.
pageWithName:	A convenience method that returns the WOComponent with the specified name.

WOContext

Method	Description
contextWithRequest:	Returns a new WOContext for the specified WORequest object.
initWithRequest: (Objective-C only)	Initializes a newly-created WOContext object with the specified WORequest.
hasSession	Returns whether the session exists.
isInForm	Returns whether WebObjects is executing code within a WOForm. Invoking **setInForm:** with **true** or **YES** also causes this method to return **true** or **YES**. Use this method when writing your own WOForm dynamic element, or an input type dynamic element.
setInForm:	Forces the value returned by **isInForm**. Invoking this method impacts all elements processed thereafter. Use this method when writing your own WOForm dynamic element, or an input type dynamic element.
directActionURLForActionNamed:queryDictionary: (Objective-C) directActionURLForActionNamed (Java)	Returns the full URL for the named action.

WOContext

Method	Description
componentActionURL	Returns the full URL for the element you are currently at.
urlWithRequestHandlerKey:path:queryString: (Objective-C) urlWithRequestHandlerKey (Java)	Returns a URL relative to cgi-bin/WebObjects for the specified request handler.
completeURLWithRequestHandlerKey:path: queryString:isSecure:port: (Objective-C) completeURLWithRequestHandlerKey (Java)	Returns the complete URL for the specified request handler, including **http://** or **https://**.

WODisplayGroup

Method	Description
setSelectedObjects:	Sets the selected objects.
setSelectedObject:	Sets the selected object.
indexOfFirstDisplayedObject	Returns the index of the first object in the current batch.
indexOfLastDisplayedObject	Returns the index of the last object in the current batch.

WORequest

Method	Description
browserLanguages	Returns the language preference list from the user's browser.
formValues	Returns an NSDictionary containing all of the form data name/value pairs.

WOResponse

Method	Description
disableClientCaching	Disables caching of this response in the user's browser.
defaultEncoding (Objective-C)	Returns the default character encoding used to construct WOResponses. By default, this encoding is NSISOLatin1.
setDefaultEncoding: (Objective-C)	Allows you to specify the character encoding that is used by default when constructing WOResponses.
stringByEscapingHTMLString:	Returns the supplied string with certain characters escaped out. Use this method to escape strings which will appear in the visible part of an html file (that is, not inside a tag).
stringByEscapingHTMLAttributeValue:	Returns the supplied string with certain characters escaped out. Use this method to escape strings which will appear as attribute values of a tag.

WOSession

Method	Description
removeObjectForKey: (Objective-C only)	Removes the object from the session dictionary that corresponds to the specified key.
setDefaultEditingContext:	Sets the editing context to be returned by **defaultEditingContext**. This can be used to set an editing context initialized with a parent object store other than the default (useful, for instance, when each session needs its own login to the database).

WOMailDelivery Class

WOMailDelivery uses the WOSendMail tool to construct an email from a file and send it using SMTP. It requires an SMTP server to be set (the default value for the SMTP hostname is "smtp"; you can change this value with **defaults write NSGlobalDomain WOSMTPHost** *hostName* or by supplying the hostname as a WOApplication command-line argument).

WOMailDelivery defines the following methods:

WOMailDelivery

Method	Description
sharedInstance	Returns the shared WOMailDelivery object to which you should send the **composeEmailFrom...** and **sendEmail:** messages.
composeEmailFrom:to:cc:subject: plainText:send:	Composes an email message with a textual body and optionally sends it. The content type is set to **Content-type: TEXT/PLAIN; CHARSET=US-ASCII**.
composeEmailFrom:to:cc:subject: component:send:	Composes an email message (and optionally sends it) where the body is the HTML that results when **generateResponse** is sent to the specified component. Note that the HTML generated is different from what would be generated in a request-response loop: all URLs in the page are complete (from **http://**) so that the mail reader can follow the links on the mailed page.
sendEmail:	Sends an email message created with one of the WOMailDelivery **composeEmailFrom...** methods.

Cookie API

The WebObjects Framework contains new classes and methods that allow you to use cookies more easily:

- The WORequest class has new methods that allow you to extract cookie data from the request.
- The WOResponse class has new methods that allow you to add a cookie to the response.
- The WOSession class has new methods that enable and disable the cookie mechanism, and control various aspects of that mechanism.
- A new class, WOCookie, defines the cookies that you add to the response.

WORequest Cookie Methods

Method	Description
cookieValuesForKey:	Returns an array of values for a cookie key. Use this method to retrieve information stored in a cookie in an HTTP header. Valid keys are specified in the cookie specification.
cookieValueForKey:	Returns a string value for a cookie key.
cookieValues	Returns a dictionary of cookie values and cookie keys.

WOResponse Cookie Methods

Method	Description
addCookie:	Adds a WOCookie object to the response.
removeCookie:	Removes a WOCookie object from the response.
cookies	Returns an array of WOCookie objects to be included in the response.

WOSession Cookie Methods

Method	Description
setStoresIDsInCookies:	Enables and disables the storing of session and instance IDs in cookies.
storesIDsInCookies	Returns whether session and instance IDs are stored in cookies.
expirationDateForIDCookies (Objective-C only)	Override to return an expiration date for cookies created for the purpose of storing session and instance IDs (by default, no expiration is set).
domainForIDCookies (Objective-C only)	Returns the path passed when creating a session or instance ID cookie.

A WOCookie object defines a cookie that can be added to the HTTP header for your response. You create a cookie using one of two methods:

WOCookie Creation Methods

Method
cookieWithName:value: (Objective-C) cookieWithName(String, String) (Java)
cookieWithName:value:path:domain:expires:isSecure: (Objective-C) cookieWithName(String, String, String, String, NSDate, boolean) (Java)

Storing Session and Instance IDs

The WOSession class provides methods for storing session and instance IDs in both cookies and in URLs. See "Cookie API" (page 41) for a listing of the methods used to store IDs in cookies. The following table lists those WOSession methods you use to store IDs in URLs:

WOSession

Method	Description
setStoresIDsInURLs:	Controls whether session and instance IDs are stored in URLs.
storesIDsInURLs	Returns whether session and instance IDs are stored in URLs.

You should store IDs either in URLs or in cookies, but not both. Enabling both ID storage mechanisms (cookies and URLs) can have unpredictable results.

WOExtensions Changes

WebObjects 4.0 includes the source code for the WOExtensions framework, in **/System/Developer/Examples/WebObjects/Source/WOExtensions** (on NT, **$*NEXT_ROOT*/Developer/Examples/WebObjects/Source/WOExtensions).**

New Components

A number of components have been added to the WOExtensions framework: WOAppleScript, WOBatchNavigationBar, WOCompletionBar, WOIFrame, and WOMetaRefresh.

WOAppleScript

The WOAppleScript component provides the ability to include client-side AppleScript in web pages, allowing WebObjects to control Macintosh computers that have the appropriate browser plug-in.

WOAppleScript has the following attributes:

WOAppleScript

Attribute	Description
scripttext	A string identifying the AppleScript to be executed on the client. This attribute is required.
controller	An optional string containing either “True” or “False” that determines whether or not the controller panel should appear.
height	The height of the AppleScript component in client browser. Optional.
width	The width of the AppleScript component in the client browser. Optional.
scriptcomment	An optional comment for the AppleScript plug-in.
scripttitle	An optional title for the AppleScript.

WOAppleScript is a non-synchronizing component. See “Non-Synchronizing Components” on page 34 for more information.

WOBatchNavigationBar

The WOBatchNavigationBar component provides the ability to navigate through display batches of a WODisplayGroup. The component provides buttons that allow you to navigate to the next batch and the previous batch. It also tells you how many batches there are to display, the number of the batch you are currently viewing, how many objects are in each batch.

WOBatchNavigationBar has the following attributes:

WOBatchNavigationBar

Attribute	Description
displayGroup	The display group to be navigated through.
width	The width of the navigation bar.

WOCompletionBar Component

The WOCompletionBar component displays a progress bar on your page. You might use WOCompletionBar in the status page of your long-running action (see "WOLongResponsePage Class" (page 46)).

WOCompletionBar has the following attributes:

WOCompletionBar

Attribute	Description
valueMin	Minimum value for the bar or value at which the bar begins.
valueMax	Maximum value for the bar, or value at which the bar ends.
value	The current amount completed. The bar sizes to this number and displays the number inside itself.
numberformat	Format in which to display the value number.
progressColor	Color for showing the value. That is, this color shows the amount completed.
backgroundColor	Color for the uncompleted portion.
width	Table width used to make the bar.
border	Table border used to make the bar.
align	Alignment of the number to be displayed.

WOIFrame Component

The WOIFrame component inserts an IFRAME tag into your page. This tag is a container to create an in-line or floating frame: a frame in which the contents of another HTML document can be seen. The difference between an IFRAME and a normal frame is that the floating frame can be seen inside a document and is treated as a part of the document. This means that when you scroll through the page the frame will scroll with it. IFRAME tags are supported by Microsoft's Internet Explorer browser.

WOIFrame has three special *mutually-exclusive* attributes, all of which are identical to WOFrame:

WOIFrame

Attribute	Description
src	External source that will supply the content for this frame.
pageName	Name of a WebObjects component that will supply the content for this frame.
value	Method that will supply the content for this frame.

The WOIFrame component's remaining attributes are simply passed through to the IFRAME:

WOIFrame

Attribute	Description
frameborder	Specifies whether or not a border should be displayed for the frame.
height	Specify the height of the frame to the browser, either as a number of pixels or as a percentage of the current screen height.
marginheight	An optional attribute that controls the vertical margins for the frame (margins are specified in pixels).
marginwidth	An optional attribute that controls the horizontal margins for the frame (margins are specified in pixels).
name	An optional argument that assigns a name to a frame so it can be targeted by links in other documents or, more commonly, from other frames in the same document.
scrolling	Specifies whether or not the frame should have a scrollbar. Optional.
width	Specifies the height of the frame to the browser, either as a number of pixels or as a percentage of the current screen height.

WOMetaRefresh Component

The WOMetaRefresh component inserts a meta-refresh tag into your page. You can set the number of seconds before the page is refreshed and either a page to transition to or an action to perform after the delay.

WOMetaRefresh has the following attributes:

WOMetaRefresh

Attribute	Description
seconds	Number of seconds before the page is refreshed.
pageName	Component to navigate to after the page is refreshed.
action	Action method to invoke after the page is refreshed.

WOMetaRefresh is a non-synchronizing component. See "Non-Synchronizing Components" on page 34 for more information.

WOLongResponsePage Class

WOLongResponsePage, defined in the WOExtensions Framework, is an abstract subclass of WOComponent. You use WOLongResponsePage when a requested action will take a long time to complete, say more than 5 seconds.

To use WOLongResponsePage, your long-running action should instantiate (with **pageWithName:**) and return a component that is a subclass of WOLongResponsePage. The subclass of WOLongResponsePage should override the method **performAction**, which is where the actual computation takes place. WOLongResponsePage performs the computation in a separate thread and returns a status page that indicates that the request is being processed.

Note: If you access WebObjects framework objects within **performAction**, you must check out the session (using the WOSessionStore's **checkOutSessionWithSessionID:request:** method) just before the WebObjects call and check it back in (with the **checkInSessionForContext:** method) just after the call.

WOLongResponsePage defines the following methods:

WOLongResponsePage

Method	Description
performAction	Override this method to perform the requested long computation. Returns the result of that computation as an object.
pageForResult:	Returns the result page that is displayed when **performAction** completes.
setStatus:	Sets the status of the computation. The long computation should send this message periodically so that the refresh page reflects the status of the computation.
refreshPageForStatus:	Returns the page that is displayed while the computation is running. This page displays the current status of the computation.
refreshInterval	The interval after which the refresh page is refreshed.
setRefreshInterval:	Sets the refresh interval.
refresh	Called by the WOMetaRefresh **invokeAction** callback (can also be called manually if the page is not self refreshing). This method calls either **pageForException:**, **pageForResult:**, **refreshPageForStatus:** or **cancelPageForStatus:** depending of the state of the long response.
isCancelled	Returns **YES** or **true** if the request has been cancelled. The long running computation should check this value to see if it should abort.
cancel	Cancels the request. You should bind a cancel button on the refresh page to this method.
cancelPageForStatus:	Returns the cancel page, displayed when the request is cancelled.
pageForException:	Returns the exception page, displayed when an exception occurs in **performAction**.
lock	Locks the page.

WOLongResponsePage

Method	Description
unlock	Unlocks the page.

Dynamic Elements Changes

- All dynamic elements now define an **otherTagString** attribute. Use this attribute to include a string directly in the element's HTML tag. Some HTML elements contain parameters that are not key-value pairs. If you wish to include one of these parameters in your element, you can send it using this attribute.

- Every element which supports the **displayString** binding now has an **escapeHTML** attribute. For the following elements, **escapeHTML** defaults to YES:

  WOBrowser
  WOPopUpButton

  For these following elements, **escapeHTML** defaults to YES if you use the **displayString** binding, and NO if you use the (now deprecated) **value** binding:

  WOCheckBoxList
  WORadioButtonList
  WONestedList

- Two new dynamic elements have been added to the WebObjects framework to better support JavaScript. They are called WOActionURL, which is similar to WOHyperlink; and WOResourceURL, which is similar to WOImage.

- WOPopUpButton and WOBrowser have a new **selectedValues** attribute which passes the selected objects to a popup or browser via a list of selected values rather than a list of selected objects. The selected values come directly from the form values of the request.

- The first item in a WOPopUpButton can now be an empty selection. Bind the **noSelectionString** attribute to a string that, if chosen, represents an empty selection. If the user leaves the WOPopUpButton at this item, then the **selection** attribute is set to **null** or **nil**.

- WOTextField and WOString have a new attribute, **formatter**, which should be bound to an NSFormatter instance. In the event a user enters a value that cannot be formatted, these elements will pass the invalid value through, allowing you to send back an error page that shows the invalid value. Note that the prior behavior in this case was to pass back a blank value for the field.

- The behavior of WORadioButton and WOCheckBox changed slightly so that they now push NSNumber objects with a value of 1 or 0 rather than @"1" or **nil** (**null** in Java) to indicate the state of the button or check box. The old behavior still applies if WebObjects 3.5 request handling is enabled (see "Troubleshooting WebObjects 4.0 Request Handling" (page 7)).

New Dynamic Element: WOFileUpload

A WOFileUpload element displays a form element in which a client browser can specify a file to be uploaded to the server. It corresponds to the HTML: **<INPUT type=file>**.

Note: WOFileUpload elements inside of a WOForm require that the WOForm have the attribute's encoding type set as follows:

```
enctype = "multipart/form-data"
```

For further information on the file upload specification, see RFC1867: **http://www.w3.org/RT/REC-html32.html#rfc1867**.

WOFileUpload has the following attributes:

WOFileUpload

Attribute	Description
filePath	The full file path and name of the file uploaded is sent by the browser and returned as a string to the variable or method bound to this attribute.
data	The file that is uploaded will be returned as an NSData object to the variable or method bound to this attribute.

If you want to process a file upload in a direct action, use WORequest's **formValueForKey:** method to get the contents of the file that has been uploaded. This method previously returned an NSString. It is now declared as follows:

```
- (id)formValueForKey:(NSString *)aKey
```

or, in Java,

```
public java.lang.Object formValueForKey(java.lang.String aKey)
```

New Component: WOQuickTime

WOQuickTime is the component for incorporating QuickTime objects (movie, sound, VR, ...) into your WebObjects applications. The WOQuickTime API is essentially based on the QuickTime plug-ins API.

The WOQuickTime component supports QuickTime VR with hotspots. If you specify a list of hotspots and the user clicks inside the QuickTime VR object, the method specified by the **action** attribute is performed and the parameter selection is set to the value of the selected hotspot.

WOQuickTime components should be used outside of an HTML form.

WOQuickTime has the following attributes. Those attributes relevant only to VR movies are indicated with “[VR]” in the description column.

WOQuickTime

Attribute	Description
filename	Path to the QuickTime object relative to the WebServerResources directory.
src	URL locating the QuickTime object. Use this attribute for complete URLs; for relative URLs use **filename** instead.
framework	The framework that contains the QuickTime object. This attribute is only necessary if the QuickTime object is in a different location from the component. That is, if the component and the QuickTime object are both in the application or if the component and the QuickTime object are both in the same framework, this attribute isn’t necessary. If the QuickTime object is in a framework and the component is in the application, specify the framework’s name here (minus the **.framework** extension). If the QuickTime object should be in the application but the component is in a framework, specify the **app** keyword in place of the framework name.
width	QuickTime object width in pixels. The width parameter is required. Never specify a width of less than 2 as this can cause problems with some browsers. If you are trying to hide the movie, use the **hidden** tag instead. If you don’t know the width of the movie, open your movie with MoviePlayer (it comes with QuickTime) and select Get Info from the Movie menu. If you don’t use the **scale** tag and you supply a width that is smaller than the actual width of the movie, the movie will be cropped to fit. If you supply a width that is greater than the width of the movie, the movie will be centered inside this width.
height	Quicktime object height in pixels. If you want to display the movie’s controller, you’ll need to add 16 pixels to the height. **height** is required unless you use the **hidden** attribute. Never specify a height of less than 2 as this can cause problems with some browsers. If you are trying to hide the movie, use the **hidden** tag instead. If you don't know the height of the movie, open your movie with MoviePlayer and select Get Info from the Movie menu. If you do not use the **scale** tag and you supply a height that is smaller than the actual height of the movie (plus 16 if you are showing the controller), the movie will be cropped to fit. If you supply a height that is greater than the height of the movie, the movie will be centered inside this height.

WOQuickTime

Attribute	Description
pluginsPage	This optional attribute allows you to specify a URL from which the user can fetch the necessary plug-in if it is not installed. This attribute is handled by your browser. If your browser cannot find the plug-in when loading your page, it will warn the user and allow them to bring up the specified URL. Generally this parameter should be set to “http://www.apple.com/quicktime”. This attribute is appropriate for both QuickTime movies and QuickTime VR Objects and Panoramas.
hotspotList	[VR] The hotspot list is an array of strings, each of which should be mapped to a hotspot ID as defined when the hotspots are created with the QuickTime VR authoring tools.
selection	[VR] A string corresponding to the ID of the user-selected hotspot.
action	Method to invoke when the QuickTime object is clicked. The selection parameter then contains the ID of the selected hotspot if a hotspot list has been specified, or **nil** otherwise.
href	URL to direct the browser to when the QuickTime object is clicked and no hotspots are hit.
pageName	Name of the WebObjects page to display when the QuickTime object is clicked and no hotspots are hit.
bgcolor	Background color for the QuickTime object. This is an optional attribute. Use **bgcolor** to specify the background color for any space that is not taken by the movie—as, for example, if you embed a 160x120 movie in a 200x120 space. Specify the color as a hex value.
target	(optional) When set, the **target** attribute is the name of a valid frame (including **_self**, **_top**, **_parent**, **_blank** or an explicit frame name) that will be the target of a link specified by the **hotspot** or **href** attribute.
volume	(optional) Possible values are 0 through 100. A setting of 0 effectively mutes the audio; a setting of 100 is maximum volume.
pan	[VR] This optional attribute allows you to specify the initial pan angle for a QuickTime VR movie.The range of values for a typical movie would be 0.0 to 360.0 degrees. If no value for **pan** is specified, the value stored in the movie is used.
tilt	[VR] This optional attribute allows you to specify the initial tilt angle for a QuickTime VR movie. The range of values for a typical movie would be -42.5 to 42.5 degrees. If no value for **tilt** is specified, the value stored in the movie is used.
fov	[VR] This optional attribute allows you to specify the initial field of view angle for a QuickTime VR movie.The range of values for a typical movie would be 5.0 to 85.0 degrees. If no value is specified for **fov**, the value stored in the panoramic movie is used.
node	[VR] This optional attribute allows you to specify the initial node for a multi-node QuickTime VR movie. If no value is specified for **node**, the default node and view (specified at creation time of the movie) is used.
correction	[VR] (optional) Possible values are “NONE”, “PARTIAL”, or “FULL” (the default). This attribute is only appropriate for QuickTime VR objects and panoramas.
cache	(optional) If set to YES, the browser will cache movies when possible just like other documents.

WOQuickTime

Attribute	Description
autoplay	(optional) When set to YES, causes the movie to start playing as soon as the QuickTime Plug-In estimates that it'll be able to play the entire movie without waiting for additional data. This attribute's default is specified by a user setting in the QuickTime Plug-in Preferences.
hidden	This optional attribute controls the visibility of the movie. By default the value is YES; if you set it to NO the movie won't be visible on the page. This option is not appropriate for QuickTime VR Objects or Panoramas. You can use the hidden setting to hide a sound-only movie.
playEveryFrame	When this optional attribute is set to YES the QuickTime plug-in plays every frame, even if it is necessary to play at a slower rate to do so. This parameter is particularly useful to play simple animations, and is appropriate for QuickTime movies. Note that setting it to YES will turn off any audio tracks your movie may have.
controller	This optional attribute sets the visibility of the movie controller (with QTVR 2.1, you can have a controller on VR Panarama or Object Movies). If you don't specify **controller**, the default is YES for QuickTime movies. For compatibility with existing web pages, the default is NO for QuickTime VR movies.
prefixHost	This attribute should be used to fix a bug with the QuickTime 2.x plug-in on Windows platforms. Setting **prefixHost** to YES (the default is NO) will automatically add the http host name at the beginning of each dynamic URL, allowing old plug-ins to correctly handle WOQuickTime component.

Changes to Localization

In WebObjects 3.5 (and earlier releases), localized versions of a component's HTML templates are located in each component's **.wo** folder, in subdirectories called ***language*.lproj** (**French.lproj**, for example). This mechanism is functional, but isn't supported by developer tools such as Project Builder and WebObjects Builder.

To improve support for developing multi-language web applications, WebObjects 4.0 adopts a localization scheme that's similar to the one for Yellow Box applications. Now components (**.wo**'s) and other resources (such as **.gif** images) are localizable from Project Builder.

The new scheme changes the locations of localized files as follows:

- Localized files go in ***language*.lproj** folders in the Web Components, Resources, and Web Server Resources directories. Any component, application resource, or web server resource can have a version in one or more **.lproj** folders.

- As with Java and Objective-C source code, script files now go at the top level of the project (or subproject), outside the **.wo**, and they are visible in Project Builder's Classes suitcase. Note that Project Builder still keeps track of the relationship between your script files and their components. For example, if you select **Main.wo/Main.html** and then select the Classes suitcase, Project Builder automatically displays **Main.wos**.

- Similarly **.api** files go at the top level of the project (or subproject), outside the **.wo**, as they (like the script and source files) apply to all localized versions of a component. In Project Builder, they are visible in the Resources suitcase.

- All language versions of a localized component (**.wo**) must contain both the **.html** and **.wod** files. If a **.woo** file exists for the component, it must be in included in each version of the component as well.

Project Builder Support

To localize a component in Project Builder:

1. Select the component to localize.

2. Open the File Attributes Inspector.

3. Click Localized, then check the languages to support.

4. Click Apply.

Project Builder creates copies of the component in the appropriate Web Components **.lproj** folders and offers to remove the global (non-localized) version from the disk. You localize other kinds of resources the same way. To view a localized resource, simply select it in its **.lproj** folder.

Building a localized application or framework creates a **.woa** or **.framework** that contains Resources and WebServerResources directories, each containing the **.lproj** folders for your project. The localized versions of each resource are installed their corresponding **.lproj** folders following a successful build.

WebObjects Builder Support

WebObjects Builder can open both localized and non-localized components. Since only one script or code file exists for a given component, WebObjects Builder is always able to locate the correct file for adding actions and variables.

Tool Changes

This section describes changes made to the tools that you use to create and test WebObjects applications. Be sure to check the document titled "What's New in Enterprise Objects Framework 3.0" for changes to EOModeler, and the Project Builder release notes for changes to Project Builder.

WOPlayback Changes

The WOPlayback tool has been rewritten in Java. You can use it as a command-line tool like you did in WebObjects 3.5, or you can use it in the Java applet viewer as a GUI tool.

There is also a new WebObjects application named PlaybackManager. PlaybackManager helps you play back a recorded session through several different clients so that you can simulate different loads that your application might have to handle.

PlaybackManager is installed in **Library/WebObjects/Applications.** Before using it, read the file **Library/WebObjects/Applications/PlaybackManager.woa/Resources/ReadMe.rtf.**

WebObjects Builder Changes

WebObjects Builder has two new major features for 4.0: Undo and frame editing support.

- You can now undo changes using Control-z on Windows NT, Command-z on Mac OS X Server, or by choosing Undo from the Edit menu.
- To create a frameset, execute the New Frameset command from the Component menu. WebObjects Builder creates an HTML file that defines a frameset. Within the frameset, you can define either static frame elements or dynamic WOFrame elements.

In addition, the following changes have been made to WebObjects Builder:

- WebObjects Builder's handling of incorrect HTML has been improved.
- WebObjects Builder now properly preserves whitespace in your HTML.
- WebObjects Builder now has an "HTML reformat" option in raw mode.
- The Palettes menu now has a SaveAs menu item, allowing you to save a palette to a new location.
- The WebObjects Builder toolbar is now active in raw mode.

- WebObjects Builder now understands fully-qualified Java classes.
- Escape completion is now supported in combo boxes.
- Performance has been improved when rendering large or complex pages.
- WebObjects Builder now supports a graphical alternative display for shared/reusable components (**WXY.wo** contains **WXY.tiff** / **WXY.bmp**, for example). It has no built-in support for generating them, however; you have to insert the image file manually, without help from WebObjects Builder.
- WebObjects Builder now works on Japanese-language systems (it works with the platform-provided language-input system, such as that provided with the Japanese version of Windows NT).

Direct to Web Changes

Direct to Web in the 4.0 release has these improvements:

- Performance improvements
- Safe editing of relationships
- A new error page component

Currently, there is no component generation support for the new components.

Monitor Changes

Monitor's user interface has been completely redesigned for this release (See the document "Serving WebObjects" for more information). In addition, the following additional features have been added:

- Monitor now allows you to set the Application Base URL under both Application Configuration and Instance Configuration. See the default value that appears in Monitor for an example of how you must format the Application Base URL variable.

Rapid Turnaround Mode

For the most part, WebObjects is an interpreted environment. The HTML templates, declarations files, and WebScript files each represent interpreted languages. One of the main benefits of an interpreted environment is that you needn't recompile every time you make a change to the project. The ability to test your changes without rebuilding the project is called "rapid turnaround" and, when using rapid turnaround capability, you're said to be in "rapid turnaround mode."

WebObjects has always supported rapid turnaround of **.html**, **.wod**, and **.wos** files within the application project. WebObjects 4.0 adds support for rapid turnaround of these files within framework projects and within subprojects of either application or framework projects.

To support rapid turnaround, WebObjects must be able to locate the resources of your application and its associated frameworks within your system's projects rather than the built products (the **.woa** or **.framework** wrappers). To tell WebObjects where to look for your system's projects you must define the NSProjectSearchPath user default. Set this default to an array of paths where your projects may be found. (Relative paths are taken relative to the executable of your project.) The order of the entries in the array defines the order in which projects will be located. The default NSProjectSearchPath is ("../.."), which causes WebObjects to look for any other applicable projects in the directory where your application's project resides. For example, if your application's executable resides within:

```
c:\web\docroot\WebObjects\Projects\MyProject\MyProject.woa
```

then from the executable's directory, "../.." would point to:

```
c:\web\docroot\WebObjects\Projects
```

If you've set your project's "Build In" directory to something other than the default, "../.." isn't likely to be appropriate; you should set your NSProjectSearchPath to point to the directories where you keep your projects while you work on them.

When your application is starting up, pay close attention to those log messages which indicate that a given project is found and will be used instead of the built product. Many problems can be solved by understanding how to interpret this output. If no such log message is seen for a given project, it won't be possible to use rapid turnaround for that project. As well, if you have several projects with the same name in the same directory, a conflict will be reported. This often happens when you have several copies of the same project as backups in your project directory. For example, you might have:

```
c:\web\docroot\WebObjects\Projects\MyApp
c:\web\docroot\WebObjects\Projects\Copy of MyApp
c:\web\docroot\WebObjects\Projects\MyAppOld
```

Even though the folders containing the projects have different names, the **PB.project** files within them might be identical. WebObjects uses the

PROJECTNAME attribute inside your project's **PB.project** file to determine the name of the project, not the name of the directory for the project. If this happens, you'll need to move the backups to another directory to avoid the conflict.

Rapid Turnaround and Direct Connect Mode

Direct connect mode is a new feature in WebObjects 4.0 which allows you to test your application without involving a web server. This means that you don't have to install your WebServerResources under the document root of your web server. The result is that rapid turnaround is even more convenient when in direct connect mode because you needn't rebuild to install WebServerResources changes to the document root.

Testing With a Web Server

When you're working in direct connect mode, few issues arise with respect to rapid turnaround. If your application has features which require a web server be used even for testing, however, there are a couple of things to know to make rapid turnaround work for you. Specifically, since you are relying on the web server to locate files within WebServerResources, you must follow these guidelines:

1. Your projects must reside somewhere below your web server's document root.

2. NSProjectSearchPath should point to all projects of interest.

3. For application projects, the WOApplicationBaseURL user default should specify the directory containing the application project. For example, if your application's project folder is:

   ```
   c:\web\docroot\WebObjects\MyApp
   ```

   then the WOApplicationBaseURL user default must be "/WebObjects".

4. For framework projects, the WOFrameworksBaseURL user default should specify the directory containing all framework projects used by the application. For example, if your application uses MyFramework.framework and that project resides in:

   ```
   c:\web\docroot\WebObjects\Frameworks\MyFramework
   ```

   then the WOFrameworksBaseURL user default must be "/WebObjects/Frameworks".

Conveniently, the two examples above use the default locations for WOApplicationBaseURL and WOFrameworksBaseURL; if your projects reside in these default locations, you need only set NSProjectSearchPath.

Also, while it is possible to point WOApplicationBaseURL and WOFrameworksBaseURL to other locations, it is not suggested that WOFrameworksBaseURL be moved since all WebObjects applications use WOExtensions.framework, which resides in the default location. If you set WOFrameworksBaseURL to point elsewhere, one side effect will be that the images in the "Raised Exception" panel will not render.

Debugging

In WebObjects 4.0, you debug an application by launching it in the Project Builder launch panel. By default, the browser is launched automatically and shows the appropriate URL.

A new feature of WebObjects 4.0 allows you to debug applications on a machine that doesn't have a web server present. See ""Serverless" Applications" (page 14) for more information.

WOApplication, WOComponent, and WODirectAction define a method named **debugWithFormat:** (**debugString** in Java). This method is similar to **logWithFormat:/logString** except that you can control whether it displays output with the **WODebuggingEnabled** user default option. If **WODebuggingEnabled** is YES, then the **debugWithFormat:** messages display their output. If **WODebuggingEnabled** is NO, the **debugWithFormat:** messages don't display their output. WODirectAction also defines **logWithFormat:**.

All dynamic elements and components now have a **WODebug** attribute that can be helpful when you are trying to locate unwanted behavior (and can also help you understand how non-synchronized components work). When **WODebug** is set to YES, it turns on a "verbose mode" for all dynamic associations for the element. This results in logs like the following being generated:

```
[NestedList:WXNestedList] (item: {label = Alpha.2.1; value = A.2.1; })
==> currentItem
[NestedList:WXNestedList] (index: 0) ==> currentIndex
[NestedList:WXNestedList] sublist <== (currentItem.sublist: *nil*)
[NestedList:WXNestedList] (item: {isNew = 1; label = Alpha.2.2; value =
A.2.2; }) ==> currentItem
[NestedList:WXNestedList] (index: 1) ==> currentIndex
[NestedList:WXNestedList] sublist <== (currentItem.sublist: *nil*)
```

The format of these logs is controlled by two new methods on WOApplication that can be overridden to customize the log messages: **logTakeValueForDeclarationNamed:type:bindingNamed:associationDescription:value:** (**logTakeValueForDeclarationNamed** in Java), and **logSetValueForDeclarationNamed:type:bindingNamed:associationDescription:value:** (**logSetValueForDeclarationNamed** in Java).

Other Changes

- The WebObjects Framework now allows you to save pages in a separate, permanent page cache. This allows you to have pages that remain in the cache; they won't drop out as the page stack is filled by other pages; you no longer have to write custom code to keep toolbars and other "static" pages around. Use WOSession's **savePageInPermanentCache:** method to save pages in this new cache. Use WOApplication's **setPermanentPageCacheSize:** and **permanentPageCacheSize** methods to set and get the size of the permanent page cache (the default size is 30 pages).
- WebScript now supports Objective-C style exception handling. See the WebObjects Developer's Guide for a complete discussion.
- WOStats is now a direct action; it no longer creates a session when you access it, causing the statistics to be thrown off. Access it now with:

  ```
  ..../App.woa/wa/WOStats
  ```
- WebObjects now includes a native adaptor for the Apache web server, as well as a WAI adaptor.
- WOContext's **session** method will create a new session of one doesn't already exist.
- WOAdaptor's **init** method now takes an NSDictionary instead of an NSArray. WOApplication's **adaptorWithName:arguments:** method also takes a dictionary instead of an array.
- WebObjects documentation and examples are now accessible through the WebObjects Info Center, instead of the "WOHomePage." The Info Center is a full-fledged WebObjects application that, in addition to acting as a single point-of-entry to developer documentation and examples, allows you to search the documentation and examples that are on your disk.

- You can now use the debug and profile targets in Project Builder when building your WebObjects applications. Parts of WebObjects—including the WebObjects Framework, the WOExtensions Framework, and MultiScript—are provided in profiled form.