What's New in WebObjects 4.5

Contents

Introduction

This document describes changes made to WebObjects between release 4.5 and 4.0. It describes changes made to existing features and describes new features you may want to start using in your applications

Compatibility with WebObjects 4.0

WebObjects 4.5 is compile-compatible with WebObjects 4.0. You should be able to run most WebObjects 4.0 applications by simply rebuilding them using WebObjects 4.5. Note that Direct To Web and Java Client applications require some additional effort to convert to WebObjects 4.5; see the *WebObjects 4.5 Post-Installation Instructions* for more information on converting your existing applications.

Changes in WebObjects 4.5

This section describes the primary changes to WebObjects for the 4.5 release, including Enterprise Objects Framework (EOF), which is considered part of WebObjects 4.5.

Platform and Language Support

WebObjects makes the following improvements to platform and language support in release 4.5:

- WebObjects applications written in Java are now supported on HP-UX.
- WebObjects now supports JDK 1.1.8 on Windows NT and on Solaris (1.1.7 on HP-UX).
- On Solaris and HP-UX, WebObjects no longer needs to be installed into `/opt/Apple`.

Note: HP-UX 10.20 is no longer supported; WebObjects 4.5 requires HP-UX 11.0.

Deploying WebObjects Applications

In addition to the changes listed under Platform and Language Support (above), the following changes have been made to ease the deployment of WebObjects applications:

- Web server adaptors can now get their configuration information from a file on the local machine, a file on another machine running a web server, from `wotaskd` (the replacement for MonitorProxy), or from other machines on the subnet running WebObjects via a new multicast mechanism.
- Many of the responsibilities formerly assumed by Monitor are now handled by `wotaskd` (each host involved in a WebObjects deployment now runs `wotaskd`). Monitor now simply changes settings in your deployment environment.

Profiling and Tuning Applications

WebObjects 4.5 provides many new features to help you profile applications, decrease their memory usage, and increase their speed. This section provides an overview of the features and tells you where to go for more information.

Profiling

WebObjects 4.5 introduces a new feature to help you profile your applications. event logging. A new event logging system records and displays how long certain operations in an application take. The measurements allow you to profile an application and optimize its execution time. For this, the EOF and WebObjects frameworks instrument key portions of their code to measure the elapsed time of functions and methods. You can instrument key portions of your code as well. To learn how to use this feature, see"Event Logging" (page 60) in the chapter What's New in Enterprise Objects Framework.

Tuning

There are many new features to help tune your application. They are:

- Object sharing

  EOF 4.5 introduces a new technique for sharing read-only enterprise objects. The new subclass of EOEditingContext, EOSharedEditingContext, defines a mechanism that allows editing contexts to share enterprise objects for reading. This mechanism can reduce both the number of fetches an application makes and the amount of redundant data it requires. See the section "Object Sharing" (page 67) in the chapter What's New in Enterprise Objects Framework.

- Subclassing EOGenericRecord

  EOF 4.5 adds a new option for creating custom enterprise objects: rather than creating a subclass of EOCustomObject (Java) or NSObject (Objective–C), you can now subclass EOGenericRecord.

  This feature is most significant in applications that use the Java bridge. By default, a subclass of EOGenericRecord stores its properties in a dictionary on the Objective–C side of the bridge instead of in individual instance variables on the Java side. This allows EOF to access enterprise object properties with many fewer trips across the bridge, which reduces memory usage and improves performance.

  See the section "Subclassing EOGenericRecord" (page 77) in the chapter What's New in Enterprise Objects Framework.

- Deferred faulting

  EOF uses faults as stand-ins for objects whose data has not yet been fetched. Although fault creation is much faster than fetching, fault instantiation still takes time. To improve performance, EOF 4.5 has the ability to use **deferred faults** (which are more efficient) for enterprise object classes that enable the feature. See the section "Deferred Faulting" (page 81) in the chapter What's New in Enterprise Objects Framework.

- Snapshot reference counting

  This is a new feature that removes snapshots from an EODatabase when they are no longer used by any enterprise objects in an application. This feature reduces the memory footprint of WebObjects applications. See the section "Snapshot Reference Counting" (page 83) in the chapter What's New in Enterprise Objects Framework.

Tools Improvements

- WebObjects Builder's user interface has been significantly enhanced and should now be even easier to use. See the chapter "WebObjects Tools Changes" (page 43).
- EOModeler supports synchronization of a database schema with the current state of a model. See the section "Schema Synchronization" (page 59) in chapter What's New in Enterprise Objects Framework.

Object Modeling Improvements

EOF 4.5 adds the following features that improve object modeling:

- Better handling of missing faults

  When a fault fires but doesn't have a corresponding row in the database, an exception isn't automatically thrown. An exception is thrown only if the application attempts to make changes to the missing fault. See "Handling Missing Faults" (page 92).
- Handling of ambiguous to-one relationships in Java

  If you enable deferred faulting—a new feature in EOF 4.5—you can have to-one relationships to non-leaf entities in an inheritance hierarchy. This arrangement was impossible without workarounds in earlier versions. See "Deferred Faulting" (page 81).
- Enterprise object classes are more flexible and easier to maintain

  You can now subclass EOGenericRecord, which can benefit the design of your enterprise objects. See "Subclassing EOGenericRecord" (page 77).

Managing Stale Data

EOF 4.5 adds a snapshot timestamping feature to help you keep your application's data fresh. It updates snapshots when fetching and allows an editing context to request that the snapshots used to build enterprise objects are no older than a particular timestamp. This has the effect of refreshing snapshots periodically, keeping the application's view more up-to-date with the database. See "Snapshot Timestamping" (page 86).

Automatic Database Reconnection

In EOF 4.5, a concrete adaptor can now implement methods that cause EOF to automatically attempt to reconnect to a database server when a connection is unexpectedly dropped. This behavior handles the problem of transient communication failures. By default reconnection is attempted by all of the adaptors that ship with EOF 4.5. See the section "Automatic Database Reconnection" (page 93) in the chapter What's New in Enterprise Objects Framework.

Direct to Web

Direct to Web now allows you to create your own visual style and exposes a great deal of new API. For more information, see the section "Direct to Web Changes" (page 53) in the chapter WebObjects Tools Changes.

Java Client

Java Client has been extended considerably, including the following:

- The foundation layer (com.apple.client.foundation) contains a new number formatter based on NSNumberFormatter and adds an NSUndoManager class, which is analogous to the server side class.
- The control layer (com.apple.client.eocontrol) is more complete.
- The distribution layer (com.apple.client.eodistribution on the client side and com.apple.yellow.eodistribution on the server side) now provides support for encrypted client/server communication and for managing user defaults.
- The interface layer (com.apple.client.eointerface) adds support for table cell editing and for displaying images and QuickTime media.

Additionally, Java Client now has a new user interface generation layer, Direct to Java Client, which is comparable to WebObjects' Direct to Web.

For more information on changes to Java Client, see the chapter "What's New in Java Client" (page 111).

LDAP Adaptor

EOF 4.5 comes with a new sample adaptor: the LDAP adaptor. The adaptor provides a simple way to verify a user's password on the Web with an LDAP server. For more information, see the section "LDAP Adaptor Example" (page 102) in the chapter What's New in Enterprise Objects Framework.

C H A P T E R 2

What's New in the WebObjects Framework

This chapter describes changes made to the Web Objects Framework and the Monitor application between release 4.0 and 4.5. It describes changes made to existing features and describes new features you may want to start using in your applications.

This chapter is organized into the following sections:

Between WebObjects 4.0 and 4.5 there have been a number of changes to the WebObjects Framework APIs. These changes are detailed in a separate chapter, "WebObjects Framework API Changes" (page 31). In addition. WebObjects application developers will likely be interested in the new Event Logging feature; see "Event Logging" (page 60).

Executive Summary

The changes to the WebObjects Framework for the 4.5 release of WebObjects can be summed up as follows:

- MonitorProxy has been replaced by `wotaskd`.
- The web server adaptors by default now get their configuration information from `wotaskd` via http requests, and no longer maintain a temporary configuration file in `/tmp`. The web server adaptor can be configured to retrieve its configuration information from a file on the current or a different web server; see "Accessing Configuration Information" (page 19) for details.
- The web server adaptors are now capable of automatically determining which WebObjects application servers are available and which applications are running on those servers.
- The web server adaptor configuration file, if used, now uses an XML format (see the online document *Deploying WebObjects Applications* for details).
- A number of aspects of WebObjects adaptor operation are now configurable at runtime.
- Much of the common functionality between WORequest and WOResponse has been abstracted into a new class, WOMessage, that represents messages with either HTML or—if your WebObjects application is written in Java—XML content. WOMessages allow your WebObjects applications to communicate with other WebObjects applications as well as with applications that "speak " XML.
- Installed as a part of WebObjects is the `com.ibm.xml.dom` package (IBM's alphaWorks), which contains various XML parsers for Java written by IBM. The WebObjects Framework uses the DOM parser to generate document and document fragment objects from XML data (or to manipulate and/or generate XML data from a document object). For more information on the alphaWorks parser, including complete documentation, point your web browser to `http://www.alphaworks.ibm.com` and search for "XML Parser for Java".

- Components can now be made stateless. A single instance of each stateless component is shared between multiple sessions, reducing your application's memory footprint.

To support the above new functionality, there have been various additions to the WebObjects Framework API's. For a complete list of changes, see "WebObjects Framework API Changes" on page 31.

Monitor Changes

Monitor and MonitorProxy have changed substantially in WebObjects 4.5. In particular, MonitorProxy has been replaced with a new process named "`wotaskd`" that's responsible for all things related to deployment on a particular host. Many of the responsibilities formerly assumed by the Monitor application are now handled by `wotaskd` (each host involved in a WebObjects deployment now runs `wotaskd`). Monitor now simply changes settings in your deployment environment.

Note: In theory, you could run multiple `wotaskd`'s on different ports on the same machine and have different web servers talk to the same app servers but see different applications. This has not been tested, however.

Monitor registration with `wotaskd` no longer used DO (Distributed Objects).

Monitor has a number of minor UI enhancements throughout, including:

- the "File Transfer Wizard," which allows Monitor users to move files and directories between hosts being monitored, and
- the "Path Wizard," which should appear everywhere you're required to enter a path in Monitor, allowing you to browse the file system on a particular host.

There are new settings that can be altered from within Monitor that affect the WebObjects adaptors.

Web Server Adaptor Changes

The WebObjects 4.5 API-based web server adaptor is intended to be used as a server plug-in where state information can be maintained from request to request. CGI is supported, but beyond the fact that it's easier to debug, the CGI adaptor provides no real benefit over the API-based adaptor.

Among the features which differentiate it from the 4.0 WebObjects adaptor:

- Many aspects are configurable at runtime, including:
  - Load balancing strategy
  - Application/adaptor communication transport
  - Socket timeouts
  - Application URL version (the adaptor supports WebObjects 3.5, 4.0, and 4.5 application URLs)
  - Error redirect URLs
- It supports a new, additional scheduling technique: "Instance load-based scheduling."
- It uses an XML-based configuration file format (see the online document *Deploying WebObjects Applications* for details).
- Configuration for multiple web servers is simpler and more automatic: configuration files are no longer required.
- The web server adaptor by default automatically discovers WebObjects-enabled systems.
- Performance has been improved.

For information on installing the adaptor, see the installation files for CGI, Apache, Microsoft ISAPI, Netscape NSAPI or Netscape WAI.

Note: The WebObjects 4.5 NSAPI adaptor won't work with Netscape Enterprise 3.5 servers. If you want to use the NSAPI adaptor, you'll need to upgrade to Netscape Enterprise 3.6, or recompile with 3.5.

Configuring the Web Server Adaptor

The web server adaptors by default now get their configuration information from `wotaskd` via HTTP requests, and no longer maintain a temporary configuration file in `/tmp`. This release also includes support for the automatic discovery of systems running WebObjects by web server adaptors. This should remove the necessity to administer the web servers, beyond the initial adaptor installation. This automatic discovery mechanism is the default; to disable it, you'll need to change the web server adaptor's configuration file as outlined in "Accessing Configuration Information" (page 19).

When the web server adaptor starts up, and at intervals determined by the configuration refresh interval setting, the adaptor sends out a multicast request in an effort to discover which WebObjects app servers are available. Each app server's `wotaskd` process replies with its URL (`http://me.myself.com:1085`). The adaptor constructs a list of these URLs and then polls each in turn to get the full site configuration information.

If the configuration refresh interval is 10 seconds, the discovery broadcast happens every 100 seconds (the discovery broadcast occurs a factor of 10 less frequently).

Accessing Configuration Information

Web server adaptor configuration information can now be obtained in a number of ways. In addition to the default "multicast" mode, the web server adaptor can be set to retrieve configuration information from one or more WebObjects application servers running `wotaskd`, or it can be set to obtain configuration information from a configuration file, which can be located either on the local machine or on a machine running a different web server.

By default, the adaptor uses multicast to communicate with all machines on same subnet that are running `wotaskd`. From the responses, the adaptor builds up a list of WebObjects application servers, along with all of the information necessary to allow access to all of the the WebObjects application instances running on those machines.

Although this is the default mode of operation, you can explicitly specify that the Web server adaptor obtain its configuration information this way using an entry similar to the following in your `apache.conf` file:

```
# Retrieve configuration information using multicast
WebObjectsConfig webobjects://239.128.14.2:1085 10
```

Note: For simplicity, only entries in `apache.conf` are shown. However, the corresponding entries can be made for all other supported web servers (with ISAPI you either need to make an entry in the Windows NT registry as shown in "Changing the Web Server Adaptor Multicast Address" (page 21)or you need to rebuild the adaptor). See the installation instructions for your particular web server—in `$NEXT_ROOT/Developer/Examples/WebObjects/Source/Adaptors`—for more information.

In the above, as in all of the configuration file entries shown in this section, the final value—10 in this instance—indicates the configuration refresh interval.

While the multicast mode requires no configuration, in a real deployment scenario you may wish to limit which application servers can be accessed from a given web server. For instance, you may have multiple web servers that each need to access a different set of WebObjects applications. Or, you may want to prevent end-user access to WebObjects applications in development or being tested on the same subnet. To explicitly specify the set of WebObjects application servers that can be accessed from a given web server, use an entry similar to the following in your `apache.conf` file:

```
# Retrieve config file from wotaskd (multiple hosts can be listed)
WebObjectsConfig http://hostA:1085,http://hostB:1085 10
```

Note that `wotaskd` must be running on each WebObjects application server in order for the above to work.

You can also have the web server adaptor obtain its configuration information from a configuration file. Using a configuration file allows you to further limit what a given web server can access: in the configuration file you can specify that the web server can access only specific applications (or even specific instances of a WebObjects application) on individual WebObjects application servers. This configuration file now uses an XML format (previously it was formatted as a property-list) which is fully described in the online document *Deploying WebObjects Applications*. Place an entry similar to *one* of the following in your `apache.conf` file to have the web server adaptor obtain its configuration information from a file:

```
# Retrieve config information from a file (XML-formatted config file)
WebObjectsConfig file:///tmp/WebObjects.xml 10
```

or:

```
# Retrieve config information from a file (old plist-style config file)
WebObjectsConfig file:///tmp/WebObjects.conf 10
```

or:

```
#Retrieve config information from a file on a different web server
WebObjectsConfig http://my.webserver.com/WebObjects.xml 10
```

Changing the Web Server Adaptor Multicast Address

The adaptor sends discovery requests out on a particular multicast "channel" (which is a combination of the IP address and the port). The defaults are:

Default IP Address: `239.128.14.2`

Default port: `1085`

The default multicast address is within the "Adminstratively Scoped Domain." That is, it's within the range of addresses intended for internal use inside organizations.

For Apache, place the following in your `apache.conf` (the final value—10 in this instance—indicates the configuration refresh interval):

```
WebObjectsConfig webobjects://239.128.14.2:1085 10
```

For CGI, either recompile, or set the `WO_CONFIG_URL` environment variable as above.

Note: With Apache, you'll need the `SetEnv` command, which comes with the "env" module. Note that Mac OS X Server doesn't switch this module on by default.

For NSAPI, place something like the following in your `obj.conf`:

Standard:

```
Init fn="WebObjects_init" root="/opt/ns-home/docs"
    config="http://localhost:1085"
```

Multicast:

```
Init fn="WebObjects_init" root="/opt/ns-home/docs"
    config="webobjects://239.128.14.2:1085"
```

For ISAPI, add the following to the registry:

```
\\SOFTWARE\\Apple\\WebObjects\\Configuration\\CONF_URL
    webobjects://239.128.14.2:1085
```

By default `wotaskd` listens for multicast discovery requests on IP address `239.128.14.2`. If you configure the web server adaptor to send such requests to a different IP address, you must also set the `WOConfigMulticastAddress` user default on machines running `wotaskd (`you must to do this as root/administrator). One way to do this is to modify the startup script to set this user default as follows:

```
defaults write wotaskd WOConfigMulticastAddress 239.128.14.2
```

If you don't change the multicast IP address, the above `defaults write` isn't necessary.

Disabling or Protecting Administrator Access

In WebObjects 4.5, sending the URL `http://`*`someHost`*`/cgi-bin/WebObjects/xyzzy` results in the webserver adaptor displaying information about all available application instances. As a convenience to the developer, this functionality is enabled by default. This has certain security implications, however. For deployment this behavior should either be turned off or protected with a username and password.

Note: When `xyzzy` output is password protected, you access the application instance display by supplying a URL of the form: `http://`*`someHost`*`/cgi-bin/WebObjects/xyzzy?`*`username+password`*

You protect the `xyzzy` output by specifying a username and password. To disable it altogether, simply specify a username of "disabled". How you do this depends on which webserver you're using. The following sections detail how you disable or protect this feature for a number of common webservers.

Apache with mod_WebObjects.so

To completely disable `xyzzy` output, add the line

```
WebObjectsAdminUsername disabled
```

to the bottom of the `apache.conf` file. To provide username and password protection, add the following two lines at the bottom of the `apache.conf` file:

```
WebObjectsAdminUsername someName
WebObjectsAdminPassword somePassword
```

NSAPI Adaptors

To completely disable `xyzzy` output, add the following to your `obj.conf` file:

```
Init fn=WebObjects_init root="C:/Netscape/Suitespot/docs"
config="webobjects://239.128.14.2:1085" username="disabled"
```

To provide username and password protection, add something like the following to your `obj.conf` file (providing your own username and password as appropriate):

```
Init fn=WebObjects_init root="C:/Netscape/Suitespot/docs"
config="webobjects://239.128.14.2:1085" username="joe" password="secret"
```

ISAPI Adaptor

To completely disable `xyzzy` output, add the following registry entry:

```
\\HKEY_LOCAL_MACHINE\\SOFTWARE\\Apple\\WebObjects\\Configuration\\WOUSERNAME disabled
```

To provide username and password protection, add registry entries similar to the following (providing your own username and password as appropriate):

```
\\HKEY_LOCAL_MACHINE\\SOFTWARE\\Apple\\WebObjects\\Configuration\\WOUSERNAME joe
\\HKEY_LOCAL_MACHINE\\SOFTWARE\\Apple\\WebObjects\\Configuration\\WOPASSWORD secret
```

WAI

To completely disable `xyzzy` output, add the following to your `obj.conf` file:

```
Init fn="WebObjectsServiceInit" root="C:/Netscape/Suitespot/docs"
config="webobjects://239.128.14.2:1085" username="disabled"
```

To provide username and password protection, add something similar to the following to your `obj.conf` file (provide your own username and password as appropriate):

```
Init fn="WebObjectsServiceInit" root="C:/Netscape/Suitespot/docs"
config="webobjects://239.128.14.2:1085" username="joe" password="secret"
```

CGI

There are two options for disabling or protecting `xyzzy` output when using CGI. First, you can uncomment the relevant code (shown below) in `main.c` and then recompile the CGI adaptor.

```
/*
 * SECURITY ALERT
 *
 * To disable xyzzy, uncomment the next line.
 * st_addStatic(options, WOUSERNAME, "disabled");
 *
 * To specify an xyzzy username and password, uncomment the next two lines.
 * st_addStatic(options, WOUSERNAME, "joe");
 * st_addStatic(options, WOPASSWORD, "secret");
 *
 */
```

Alternatively, if the webserver is configured to pass environment variables, the CGI adaptor will read them. For example, configure Apache to load the module `mod_env.so` by adding or uncommenting the lines in `apache.conf`.

```
LoadModule env_module        /System/Library/Apache/Modules/mod_env.so
AddModule mod_env.c
```

You will also need to add the following System directive in `apache.conf`:

```
PassEnv WO_XYZZY_USERNAME WO_XYZZY_PASSWORD
```

To completely disable `xyzzy` output, you must then create an environment variable `WO_XYZZY_USERNAME` and set its value to "disabled".

To provide username and password protection, create two environment variables, `WO_XYZZY_USERNAME` and `WO_XYZZY_PASSWORD`, and set them to, for example, "joe" and "secret" respectively.

Licensing Changes

From a legal standpoint, the WebObjects 4.5 licensing scheme is not intended to be any different from 4.0. However, WebObjects 4.5 changes both the registration mechanism for instances and the way the configuration is picked up by the webserver adaptors.When the license is correctly enforced, these changes mean certain usage that was possible (though not strictly conforming to the license) with WebObjects 4.0 is no longer possible with 4.5.

Unregistered instances appear with an instance ID of -1. This is all that is allowed with the restricted license. If an app sees a URL with a positive instance ID and the license is restricted, it will generate a "more than one instance detected" error message. This error simply means that there's an instance out there using a positive instance ID when only -1 is allowed. In this instance, it does *not* literally mean there is more than one instance running.

Apps now register via a UDP heartbeat with `wotaskd`. This fixes a large number of problems with the way the old monitoring scheme worked, but, if `wotaskd` knows about it, any app that registers with the correct name on a given port will be given the instance ID assigned for that app name and port combination. This can mean that your app gets a "deployment" instance ID when your license does not actually allow that. This behavior is different from 4.0, since in 4.0 you either built the `.conf` file yourself (and thus could do what you liked), or you allowed the adaptor to pick up the `.conf` file from `/tmp`, in which all instance IDs were set to -1. You can avoid this problem in WebObjects 4.5 by either picking a different port or removing the instance from the configuration on that host using Monitor.

WebObjects 4.5 by default performs instance discovery via multicast. On a large subnet with many machines running WebObjects, instance discovery could produce a lot of replies. WebObjects 4.5 filters out non-deployment instances that aren't running on the same machine as the adaptor, so the only ones you wind up seeing are those you should be able to talk to, and the possibility of name/port collision in the adaptor is reduced. This filtering basically means you can't talk to an app instance that isn't "deployed" if the webserver and adaptor are running on a different machine. To make this work in WebObjects 4.0 you had to create your own configuration file (by copying the configuration file in `/tmp` on the other machine) and use -1 for the instance IDs. In Webobjects 4.5, you should be able to get this old

WebObjects 4.0 behavior by using a file URL to specify the adaptor configuration file. See "Accessing Configuration Information" (page 19) for more information on specifying an adaptor configuration file.

Miscellaneous Changes

In addition to the numerous major changes listed elsewhere in this document, a number of smaller changes are important to note:

- Configuration files have moved from `$(NEXT_ROOT)/Library/WebObjects/Configuration` to `$(NEXT_ROOT)/Local/Library/WebObjects/Configuration`. As with all end-user configurations, `/Local` is reserved for files which you change.
- The `WOMonitorHost` user default (and command-line argument) has been deprecated. WOF always tries to register with the service named `wotaskd` on `localhost`.
- This release supports a `WORequiresWOF40Compatibility` user default that re-enables certain behaviors of the 4.0 release. If you're having problems with an application you've ported from WebObjects 4.0, set the `WORequiresWOF40Compatibility` user default to "YES".
- The WORadioButtonList dynamic element has been deprecated. Use WORadioButtonMatrix (defined in the WebObjectsExtensions Framework) instead.

Supplemental Documentation

The following documentation supplements that found in the *WebObjects Developer's Guide*.

Direct-Connect Mode

For deployment, a web server should be running to receive HTTP requests and to forward them through the WebObjects adaptor. To simplify the development process, though, WebObjects applications are capable of receiving HTTP requests directly. This is the default; invoke `WOApplication`'s `setDirectConnectEnabled` method to disable direct-connect mode.

This 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 web server's document root in order to test them.
- Running without an HTTP server uses less memory on your development machine.
- The WebObjects example applications don't need to be installed under the web server's document root (they are installed under `Developer/Examples/WebObjects`).

The `WOPort` command-line option (also settable from Monitor) allows you to specify the number of the port where the application should listen for requests when operating without a web server. By default, `WOPort` is -1, which assigns an arbitrary high port number to the application. Thus, you aren't required to specify a port number when in direct-connect mode. However, it's generally a good idea to assign a specific port number.

Note that if you do want to use a web server to test WebObjects examples, you can still do so. Before you do, perform a "make install" (be sure to set `INSTALLDIR_WEBSERVER` in the makefile preamble) to install the example's web server resources (such as image files and Java client-side classes) in the web server's 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 the following section for more on developing with and without a web server.

Rapid Turnaround Mode

WebObjects is largely 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 supports rapid turnaround of `.html`, `.wod`, and `.wos` files within application projects, framework projects, and 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.

Pay close attention when you have several projects with the same name in the same directory. 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 with in a WebObjects application, WebObjects checks the path of the project in question against the path to the executable (after resolving symbolic links). and chooses the project whose path matches the initial portion of the executable's path (thus, in the above example it would choose `MyApp`). If multiple projects with the same `PROJECTNAME` attribute reside within a single directory in a framework project, WebObjects chooses the one who's `PROJECTNAME` matches the prefix of the framework's project directory.

Rapid Turnaround and Direct-Connect Mode

Direct-connect mode allows you to test your application without involving a web server. This means that you don't have to install your web server resources 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 web server resources changes to the document root. See "Direct-Connect Mode" (page 27) for more information on Direct-Connect Mode.

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 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:

- Your projects must reside somewhere below your web server's document root.
- `NSProjectSearchPath` should point to all projects of interest.

- 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`".

- 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.

C H A P T E R 3

WebObjects Framework API Changes

This chapter details those changes in the API of the WebObjects Framework, listing new classes and methods and identifying API that has been deprecated since the previous release.

This chapter is organized into the following sections:

New Classes

This release of WebObjects adds two new classes to the WebObjects Framework, WOEvent and WOMessage.

WOEvent

WOEvent is a subclass of EOEvent (defined in the EOControl framework) that serves as the parent class for objects that gather information—such as duration—about various operations in WebObjects. You can see the results of this information gathering in your web browser by accessing a special "event display" page, and you can configure how the results are displayed by accessing a special "event setup" page.

WOEvent adds knowledge of pages and components to the EOEvent class. Events that are subclasses of WOEvent can be grouped or aggregated by page or by component. Although you can subclass WOEvent, in most cases the private subclasses included in the framework will be adequate for analyzing WebObjects applications.

WOMessage

WOMessage is the parent class for both WORequest and WOResponse, and implements much of the behavior that is generic to both. WOMessage represents a message with an HTTP header and either HTML or XML content. HTML content is typically used when interacting with a Web browser, while XML content can be used in messages that originate from or are destined for another application (either an application that "speaks" XML or another WebObjects application).

The methods of the WOMessage class can be divided primarily into two groups, those that deal with a message's content and those that read and set header information. Most of the remaining WOMessage methods control how the content is encoded and allow you to attach arbitrary "user info" to your WOMessage objects in order to pass information about a given message to other objects within your application.

Messages with XML Content

The Java version of the WOMessage class contains three methods that allow you to construct and interpret messages whose content is formatted as XML (these methods aren't available to Objective-C programmers).

The arguments to these methods are XML documents (or, in the case of `appendContentDOMDocumentFragment`, a document fragment) as defined by the Document Object Model (DOM). Installed as a part of WebObjects is the `com.ibm.xml.dom` package (IBM's alphaWorks), which contains various XML parsers for Java written by IBM. The included DOM parser is used to generate document and document fragment objects from XML data (or to manipulate and/or generate XML data from a document object). For more information on the Document Object Model, see the online documentation at `http://www.w3.org/DOM/`.

Changes to WORequest

The following methods are no longer declared in WORequest but are instead inherited from WORequest's new parent class, WOMessage:

`content`

`headerForKey:` (`headerForKey` in Java)

`headerKeys`

`headersForKey:` (`headersForKey` in Java)

`httpVersion`

`userInfo`

Changes to WOResponse

The following methods are no longer declared in WOResponse but are instead inherited from WOResponse's new parent class, WOMessage:

`defaultEncoding` (static/class method)

`setDefaultEncoding:` (`setDefaultEncoding` in Java; static/class method)

`addCookie:` (`addCookie` in Java)

`appendContentCharacter:` (`appendContentCharacter` in Java)

`appendContentData:` (`appendContentData` in Java)

`appendContentString:` (`appendContentString` in Java)

`content`

`contentEncoding`

`cookies`

`headerForKey:` (`headerForKey` in Java)

`headerKeys`

`headersForKey:` (`headersForKey` in Java)

`httpVersion`

`removeCookie:` (`removeCookie` in Java)

`setContentEncoding:` (`setContentEncoding` in Java)

`setContent:` (`setContent` in Java)

`setHTTPVersion:` (`setHTTPVersion` in Java)

`setHeader:ForKey:` (`setHeader` in Java)

WOHTTPConnection

The new WOHTTPConnection class is intended to be used as a client for HTTP communications. It gives you direct access to the HTTP contents and headers. WOHTTPConnection's sendRequest: method allows you to send a WORequest object directly to a server, while readResponse allows you to receive WOResponse objects from that same server.

XML Package

WebObjects 4.5 includes a new XML package, `com.apple.webobjects.xml`, which consists of the following:

WOXMLCoding interface
WOXMLCoder class
WOXMLDecoder class

Use this package to encode and decode objects as XML. The WOXMLCoder and WOXMLDecoder classes can be used to archive and unarchive object data, or to parse and/or generate XML obtained from or destined for an external source (such as the World Wide Web). When working with such "foreign" XML, you describe the XML elements and properties and their mapping to objects in an XML-format "mapping model" that you can create with either a text editor or an XML editor.

For complete details on the XML framework and its use, see the reference documentation in the WOInfoCenter. For examples, see the XML Archiving andRelatedLinks examples (accessible through the WebObjects Info Center under Examples > WebObjects > Java).

New Methods

In addition to the new event classes (see the discussion of `WOEvent`/`EOEvent` elsewhere in this document), the following methods have been added to the classes that make up the WebObjects Framework.

Table 3-1 WOApplication

Method	Description
`createContextForRequest:` (Objective-C) `createContextForRequest` (Java)	Creates a new context object for a given request. Override this method if you need to provide your own subclass of `WOContext`. If you override it, you need not call `super` in your overriding method.
`defaultRequestHandlerClassName`	The default implementation of this method returns "WOComponentRequestHandler", which is the default request handler. If you don't want a session created for your application's home page, override this method to return "WODirectActionRequestHandler", making the direct action request handler the default.
`sharedEditingContext`	This is a convenience method that returns the default shared editing context.
`traceAll:` (Objective-C only; in Java, this method is still named "`trace`")	Old "`trace`" method.

Table 3-2 WOAssociation

Method	Description
`isValueConstantInComponent:` (Objective-C) `isValueConstantInComponent` (Java)	Returns `true` when the association is "constant." Use this for checking bindings at runtime.
`isValueSettableInComponent:` (Objective-C) `isValueSettableInComponent` (Java)	Returns `false` when the association is "constant." Use this for checking bindings at runtime.

Table 3-3 WOComponent

Method	Description
`canGetValueForBinding:` (Objective-C) `canGetValueForBinding` (Java)	Verifies that the binding exists and that `valueForBinding` will return a value.
`canSetValueForBinding:` (Objective-C) `canSetValueForBinding` (Java)	Verifies that the binding exists and that `setValueForBinding` will succeed.

Table 3-3 WOComponent (continued)

Method	Description
`isEventLoggingEnabled`	Called to determine if a component wants event logging. This is not desirable, for example, for components which are associated with event display as they would interfere with the actual event logging. The default implementation of this method returns `true`.
`isStateless`	By default, this method returns `false`, indicating that state will be maintained for instances of the receiver. Overriding this method to return `true` will make the component stateless. A single instance of each stateless component is shared between multiple sessions, reducing your application's memory footprint.
`reset`	This method—which is only invoked if the component is stateless—allows a component instance to reset or delete temporary references to objects that are specific to a given context.

Table 3-4 WODisplayGroup

Method	Description
`globalDefaultForValidatesChangesImmediately`	This static/class method returns the class default controlling whether changes are immediately validated.
`globalDefaultStringMatchFormat`	This static/class method returns the default string match format for the class.
`globalDefaultStringMatchOperator`	This static/class method returns the default string match operator for the class.

Table 3-4 WODisplayGroup (continued)

Method	Description
`insertObjectAtIndex` (New in Java only; the Objective-C version of this method previously existed)	Inserts the supplied object into the receiver's `EODataSource` and displayed objects at the specified index, if possible.
`setGlobalDefaultForValidatesChangesImmediately:` (Objective-C) `setGlobalDefaultForValidatesChangesImmediately` (Java)	This static/class method sets the class default controlling whether changes are immediately validated.
`setGlobalDefaultStringMatchFormat:` (Objective-C) `setGlobalDefaultStringMatchFormat` (Java)	This static/class method sets the default string match format for the class.
`setGlobalDefaultStringMatchOperator:` (Objective-C) `setGlobalDefaultStringMatchOperator` (Java)	This static/class method sets the default string match operator for the class.
`stringQualifierOperators`	Returns an array containing all of the relational operators supported by EOControl's EOQualifier that work exclusively on strings: "`starts with`", "`contains`", "`ends with`", "`is`", and "`like`".

Other WODisplayGroup Changes

In the past, EODisplayGroup and WODisplayGroup replaced all instances of "=" in queries involving string attributes with the `defaultStringMatchOperator`. This is `caseInsensitiveLike`, although it can be changed by a variety of instance or class methods on display group. Additionally, these string based queries had the wildcard character appended to them by means of the `defaultStringMatchFormat` which is initially `@"%@*"`.

These transformations took place even if the `EOQualifierOperatorEqual` is explicitly set in the display group's `queryOperator` dictionary.

As of version 4.5, a display group won't alter "=" if it is explicitly set. It will provide the old behavior only if the query operator is not set. Therefore, `EOQualifierOperatorEqual` will no longer provide any pattern matching functionality and will generate "=" instead of "like" in the SQL.

You can get the old behavior if you set the `WORequiresWOF40Compatibility` default. Alternatively, you can replace the "=" in the appropriate entry of the `queryOperator` dictionary with the empty string or with `"starts with"` (both provide the default behavior).

Additionally, display group provides a new instance method, `stringOperators`, which returns an array of string comparison operators including `"starts with"`, `"ends with"`, `"contains"`, `"is"`, and `"like"`. `"starts with"`, `"ends with"`, and `"contains"` all use the `defaultStringMatchOperator` and the match formats `@"%@*"`, `@"*%@"`, and `@"*%@*"` respectively. The `defaultStringMatchOperator` is still `caseInsensitiveLike`. `"is"` and `"like"` map exactly to the "=" and `caseInsensitiveLike` operators.

The pre-v4.5 behavior was precisely `"starts with"`.

These string operators are only supported by display group, not EOQualifier, nor any other part of EOF in this release.

Deprecated API

Table 3-5 WOApplication

Deprecated API	New API or Workaround
`logToMonitorWithFormat:` (Objective-C) `logToMonitorString` (Java)	New features in Monitor will allow logging of information. The deprecated API does nothing.
`monitorHost` (The Java version of this method has not yet been deprecated)	New monitor features eliminate the need for this method.
`setMonitorHost:` (The Java version of this method has not yet been deprecated)	New monitor features eliminate the need for this method.
`trace:` (Deprecated in Objective-C only; in Java, this method remains as-is)	`traceAll:`

Table 3-6 WOResponse

Deprecated API	New API or Workaround
`appendContentBytes:` (Objective-C only; this method never existed in the Java version of `WOResponse`)	Deprecated. Use `appendContentData:`, which is inherited from `WOMessage`.

WOExtensions Changes

The WebObjects Extensions framework has changed in the following ways:

- Documentation for the components is now provided
- 12 reusable components have been added
- 11 reusable components have been made stateless
- WOTableString has been deprecated

WOExtensions Reference Documentation

A new document titled WebObjects Extensions Component Specifications has been added to cover the reusable components in the WebObjects Extensions framework. You can access this document from the WebObjects Info Center.

New Components

The following reusable components are new:

JSAlertPanel
JSConfirmPanel
JSImageFlyover
JSModalWindow
JSTextFlyover
JSValidatedField
WOCheckboxMatrix
WOEventDisplayPage
WOEventSetupPage
WOKeyValueConditional
WORadioButtonMatrix
WOTabPanel

See the WebObjects Extensions Component Specifications for more information about these components.

Stateless Components

Making a reusable component stateless significantly improves the performance of the component. To this end, the following WebObjects Extension components are now stateless:

WOAnyField
WOBatchNavigationBar
WODictionaryRepetition
WOSimpleArrayDisplay
WOSimpleArrayDisplay2
WOSortOrder
WOSortOrderManyKey
WOTable
WOThresholdColoredNumber
WOToManyRelationship
WOToOneRelationship

The conversion has not changed their function.

Deprecated Elements

WOTableString is now deprecated. This component was used to improve the appearance of table cells containing empty strings. The HTML specification designates that the borders for table cells should only appear when the cell has content; which causes table cells containing empty WOStrings appear without their borders. The WOTableString reusable component was provided as a workaround that rendered as a non-breaking space when its value was empty, preserving the borders.

Instead of using the WOTableString reusable component, use the WOString dynamic element with the valueWhenEmpty attribute bound to "&nbsp;".

C H A P T E R 4

WebObjects Tools Changes

This chapter describes changes to the WebObjects tools between release 4.0.1 and 4.5. It describes changes to existing features and new features that you might want to start using. The WebObjects tools include the Project Builder application, the WebObjects Builder application, and the Direct to Web framework.

Project Builder Changes

Project Builder has received changes in this release in the areas of `NSProjectSearchPath` support, indentation, and the Build, Launch, and Find panels.

Project Builder now includes a Searchpath preferences panel to specify the project's `NSProjectSearchPath` user default. If your project uses framework projects that reside in a directory in the `NSProjectSearchPath`, you can access the source files for the frameworks through the Frameworks suitcase. Using the Build Options panel in the Build panel, you can specify whether to build the frameworks when you build the project. In the Find panel you can specify whether to search the project and frameworks or just the project.

Project Builder's automatic indentation has been upgraded. You can control its behavior with the Indentation preferences panel.

Project Builder's Build, Launch (with debugger) and Find panels now have an extra pane where you can edit code. You no longer need to switch between the main Project Builder window and the Build panel when you are editing compile errors. Likewise, you can search for text within the project and edit it without leaving the Find panel.

Project Builder is documented in *WebObjects Tools and Techniques*.

WebObjects Builder Changes

WebObjects Builder's user interface has received major changes for 4.5, specifically in the main window toolbar, the user interface for binding keys, and the table editing user interface. A path view, an API editor, and component validation have been added. WebObjects Builder is documented in *WebObjects Tools and Techniques*.

Main Window Changes

The editing modes from WebObjects 4.0.1 have been renamed. Graphical Editing Mode is now called the layout view. Source Editing Mode is now called the source view. A new view called the preview view has been added.

Layout View Changes

The toolbars have been reorganized. All tools are now available in a single click. You don't have to choose a toolbar. The WOComponentContent dynamic element is now available from the toolbar. In addition, the toolbar also contains an alert icon, which opens a window displaying the validation errors on the page and a puzzle piece icon, which opens the API editor. The API editor is described below.

WebObjects Builder now optionally displays HTML tags with opening and closing tag icons, which makes the HTML structure of the page more transparent. You can choose which tags are marked this way with the Layout preferences panel.

In the center of the window is a path view, which replaces the path view in the inspector and provides extra functionality. The path view displays the elements in the path as HTML tags rather than icons.

The object browser has column headings. Each heading displays the class of the object selected in the previous column. This class also contains the keys in the column.

Double clicking an element's icon no longer collapses the element. Use the preview view instead (described below).

Preview View

A new preview view, similar to the layout view, displays the page so it matches what the user sees in the browser as closely as possible. WebObjects Builder renders elements (for example, WORepetitions) in their collapsed form and hides comments and HTML tags.

Source View

Syntax coloring is new for WebObjects 4.5. By default, WebObjects Builder displays unmatched tags in red, markers in purple, comments in grey, and WebObjects in blue. You can change the colors using the preferences panel.

You can match a tag in the source view by triple-clicking on it. For example, if you want to select a `<FONT FACE=...>` tag, the closing `</FONT>` tag, and everything in between, triple click on `<FONT`, `>`, or `</FONT>`. You can also drag the mouse after you have triple-clicked, which expands the selection by whole containers.

Changes to the Binding Process

The process of binding is more flexible, streamlined, and intuitive in this version of WebObjects Builder.

Inspector Appearance

The inspector now has a horizontal aspect, which is more convenient to tile with the main window than before.

Documentation

Documentation for static dynamic elements is now available from the inspector by clicking the book icon. The documentation appears in the WOInfoCenter.

Binding by Dragging

Binding by dragging from a variable to an element is now different. Instead of opening the inspector when you release the mouse button, WebObjects Builder now opens a menu with the element's attributes. You can click one of these attributes or click “Connect to new binding...” which adds a new binding.

Binding With the Element’s Context Menu

You can bind a variable to an element's attribute by selecting the key in the object browser and Control-clicking (right-click in Windows NT) an element in the upper pane of the main window. A menu appears containing a submenu with the element's attributes. Click one of the attributes to connect it to the selected key. You can also create a new binding by clicking “Connect to a new binding...”

Binding Validation

WebObjects Builder checks for required bindings that are missing and mutually exclusive bindings that are specified. Invalid bindings display in red in the inspector. By clicking the alert icon in the inspector, you can access a description of the missing bindings.

Adding and Deleting Bindings with the Inspector

A pull-down list in the top right corner of the inspector allows you add and delete bindings. You can also add a binding by selecting the inspector and pressing Enter. You can delete a binding by selecting the binding and pressing the delete key.

Binding Aids in the Inspector

Creating bindings for date formats, number formats, booleans, image file names, framework names, page names, direct actions, and direct action classes is now easier. When WebObjects Builder can determine a set of possible values for an attribute, it displays a combo box for the attribute that allows you to choose one of these values. For example, boolean attributes have a combo box which allows you to choose YES or NO.

Binding Name Completion

When you inspect an element, double-click in the binding column, and start typing a key, WebObjects completes the name for you based on the keys in the object browser. For example, to bind to "application.allGuests.count," you simply type "a.a.c" and the inspector fills in the rest. The object browser also selects the key as you type it.

Working with Keys

The keys pull-down list at the bottom left corner of the main window manipulates keys in the component's script file only. It does not affect keys in the application or session files. You can add keys to the application (or session) with the application's context menu by selecting `application` (or `session`) and Control-clicking (right-clicking in Windows NT) in the next column of the object browser.

You can now rename and delete keys in the component's script file using the keys pull-down list. To delete or rename a application or session key, Control-click (right-click in Windows NT) on the key and choose Delete *key* or Rename *key* from the pop-up menu.

Changes to Keyboard Actions

The tables below show the changes to the keys you need to press to perform certain WebObjects Builder functions.

Table 4-1 Layout View Keyboard Actions

	Pre-WebObjects 4.5	**WebObjects 4.5**
Insert Paragraph (`<P>`)	Shift-Enter	Enter
Insert Line Break (`<BR>`)	Enter	Shift-Enter
Delete Text	Backspace, Delete, or Del	unchanged
Delete Structures	Backspace, Delete, or Del	Shift-Backspace, Shift-Delete, or Shift-Del
Add New List Item	Shift-Enter	Enter

Working with Tables

Creating and editing tables is substantially changed and more intuitive in this version of WebObjects Builder.

Creating Tables

To create a table, click the table icon in the toolbar. You can set the dimensions, size, layout, and other parameters for your table in the panel that appears. Press OK to insert the table.

Making Selections

In previous versions of WebObjects Builder, the inspector had a path view. In this version, the path view is in the main window. Thus, selecting the entire table or a single row is done differently in this version. If you select a table cell, you can inspect the row (by clicking <TR> in the path view) or the table itself (by clicking <TABLE> in the path view).

You can select multiple cells by

- clicking in a cell and dragging across the cells
- selecting a cell and shift clicking in another cell
- command (control in NT) clicking each cell.

The first two selection methods ensure that the selected cells form a contiguous region.

Note: selecting all of the cells in a row or table is not the same as selecting the row or table!

Editing Tables

The structure/content table editing modes have been eliminated. Click in a cell to edit its contents. The table data and table row inspectors now have buttons to edit the table's structure.

Working with Fonts

In HTML, the FACE attribute for the FONT tag specifies a comma separated list of font names in order of preference. The browser searches for an installed font with a corresponding name in the list. WebObjects Builder provides a font panel which manipulates these lists. This font panel is available from the font pull-down list in the toolbar.

To delete a font list, select it in the font panel and press the delete key.

Path View Menu

Control-clicking (right-clicking on Windows NT) an element in the path view brings up a menu from which you can

- inspect the element
- make the element a static element (if it is a dynamic element)
- make the element a dynamic element (if it is a static element)
- delete the element and its contents
- delete the element's contents only
- delete the element without deleting its contents (unwrapping the element)
- isolate the selection (wrap the selected content in the parent element separately from the unselected content.)
- make many kinds of selections.

Context Menus

The path view menu is one example of context menus. These menus appear when you control-click (right-click on Windows NT) on elements or keys in the WebObjects Builder window. The following parts of the user interface have context menus:

- elements in the upper pane of the layout and preview views
- elements in the path view
- keys in the object browser

- the empty space in the columns of the object browser
- the empty space in the upper pane of the layout and preview views
- the table views, including the Attribute/Binding table in the inspector, and the Bindings and Messages tables in the API editor.

API Editor

A graphical API editor is now accessible by clicking the puzzle piece icon on the toolbar. It allows you to define the attributes and binding rules for a reusable component. The editor has three panels, one for creating bindings and setting attributes on them, one for creating validation rules, and one for associating an icon and documentation file with the component.

The Bindings tab allows you set up the attributes of your reusable component. WebObjects Builder displays these attributes in the inspector in WebObjects Builder. You can also tell WebObjects Builder which attributes must be bound, which attributes must be bound to a key that can be set, and the kind of values each attribute takes (such as page names, MIME types, or frameworks).

The Validation tab allows you to set up validation rules for the component. These rules specify which attributes must be bound together and which cannot be bound together. The “required” and “will set” checkboxes on the first tab are really shortcuts for setting up validation rules; the generated rules appear on the Validation tab when you check those checkboxes for your attributes. You can also set up more complex rules on the Validation tab. For example, you can specify that a set of attributes all control the same basic property of a component and only one of them may be bound at a time.

The Display tab allows you to choose an image that WebObjects Builder uses to display the reusable component in a document. You can also specify an HTML documentation file for the reusable component with the Display tab. WebObjects Builder displays this file in the WOInfoCenter when the user clicks the book icon in the component’s inspector.

Syntactic and Semantic Constraints

WebObjects Builder optionally enforces constraints defined by the HTML 3.2 specification published by the World Wide Web Consortium (W3C). The W3C maintains the standards governing the World Wide Web. Their website is at http://www.w3.org.

While mainstream browsers tolerate many HTML errors, WebObjects Builder's Layout view does not allow you to introduce semantic errors (although the Source view allows you to create and edit any HTML). If your document already has HTML errors, you can still edit it with the Layout view depending on the settings in your Validation preferences panel. This panel allows you to specify what WebObjects Builder does when it encounters semantic or syntactic errors when it tries to display the document in the Layout view.

Semantic errors occur when an HTML tag cannot be a child of another tag. For example, according to the HTML 3.2 specification, <B> cannot have <H1> as a child because <B> is a text-level tag, while <H1> is a block-level tag. Close tags without corresponding open tags and open tags without corresponding close tags are also semantic errors.

Syntactic errors occur when your HTML is malformed, for example, <//B> or <B.

Depending on the settings in the Validation preferences panel, WebObjects Builder automatically repairs each error it encounters, ignores it, or asks you what to do with it. Note that there are some errors for which WebObjects Builder cannot accommodate the settings you have chosen. For example, a document with two <BODY> tags can neither be repaired nor ignored.

If you choose to repair errors, WebObjects Builder adds missing close tags, and removes extra or malformed tags. When one tag is not allowed as a child of another, WebObjects Builder introduces an intermediate element between the two incompatible elements or adds a close tag for the parent element depending on the context. These repairs can be complicated and intrusive, so you should not routinely allow WebObjects Builder to repair all errors.

If you choose to ignore errors, WebObjects Builder does not modify your document. To see the errors in your document from the Layout view, bring up the validation panel.

If you choose to have the Builder ask about errors, it stops at every error it encounters as it tries to display your document in the Layout view and asks you whether you want to repair the error, ignore it, or stop and display the document in the source view. Note that if you repair some errors and then stop on one, the errors that you repaired revert to their original (incorrect) state; WebObjects Builder does not modify the document unless every error is either repaired or ignored.

One common semantic violation seen in WebObjects HTML templates is improperly nesting tags within a WOConditional. For example:

```
<WEBOBJECT NAME=Conditional1>
    <B>
</WEBOBJECT>
This text is conditionally displayed in boldface.
<WEBOBJECT NAME=Conditional1>
    </B>
</WEBOBJECT>
```

The corresponding entry in the bindings file is:

```
Conditional1: WOConditional {
    condition = myCondition;
}
```

Not only is this a semantic violation, it is also bad coding practice. It is much better to use a WOGenericContainer in such cases:

```
<WEBOBJECT NAME=Generic1>
    This text is conditionally displayed in boldface.
</WEBOBJECT>
```

The corresponding entry in the bindings file is:

```
Generic1 : WOGenericContainer {
    elementName = “B”;
    omitTags = myNegatedCondition;
}
```

Note that you have to negate the original condition.

How WebObjects Builder Handles Bindings Files

WebObjects Builder now respects bindings (`.wod`) files that have been edited by hand. Specifically, if you edit the bindings file of your component with a text editor, open the component in WebObjects Builder, and save it, WebObjects Builder

- preserves the whitespace and comments around entries in the bindings file
- retains the order of the entries (depending on the settings in the `.wod` preferences panel)
- allows a single entry to be used multiple times in the HTML template file

This behavior of WebObjects Builder eases the transition from hand-editing components with a text editor to using WebObjects Builder.

Direct to Web Changes

Direct to Web has received two major changes: exposed API and support for creating your own visual style. There are many additional changes as well. This version of Direct to Web is not completely compatible with the 4.0 release. You need to run conversion scripts on your project's source files, `Main.wod` file, and rule (`.d2wmodel`) files. See "Converting Projects From Earlier Releases" (page 56).

API and Components Exposed

One of the major goals of the 4.5 release is to make more of Direct to Web available to developers. The API and components have been reviewed and streamlined and are now ready for public use. In addition, two books have been added to the documentation about Direct to Web.

Developing WebObjects Applications With Direct to Web discusses the Direct to Web architecture and how to customize your Direct to Web application.

The *Direct to Web Reference* covers the classes and reusable components in the Direct to Web Framework.

You can access these documents from the WOInfoCenter.

Modifying the Visual Style

It is now possible to create your own Direct to Web-style dynamic template. When generating a 'task' page (for example, the List Page for *all* entities) Direct To Web generates a page that is not specific to a particular entity (unlike freezing a page). Instead, the page retains its dynamic capabilities and reacts to the entities it encounters at runtime. You can also access and modify its template, binding, and Java source files. By creating a set of these for all the common tasks (query, list, edit, etc.) you can create your own look.

Note that you can still freeze a Direct to Web page for a particular task and entity as you could in previous releases.

Modifying the D2W Menu

The Navigation tool of Direct to Web is now part of your project when you create it. This allows you to tailor its look and functionality.

Neutral Look

A new look, called the Neutral look, is now available when you create a Direct to Web project. It is ideal for adding your own logo because it contains no Apple or WebObjects logos.

Custom Components

It is now possible to embed regular WebObjects components in a Direct to Web page. Two components, D2WCustomComponent and D2WQueryCustomComponent, have been added for this purpose.

From the Web Assistant, you can configure Direct to Web to use your component for a particular task and property. See *Developing WebObjects Applications with Direct to Web* for more information.

Named Configurations

Once you have configured a given page (for example, List page for Movies) you can save those settings under a name (for example, ListRentedMovies) and use this configuration from your code (either by API or as an embedded component). This lets you have several pages for the same task/entity pair which display different sets of properties. For example, the Rental Store example uses different List Pages for Movies depending whether you are a customer or a clerk, and whether you are renting a movie or just browsing.

Tab Panel Page

A tab panel inspect and edit page has been introduced in the WebObjects and Neutral looks, which lets you group property keys in different tabs. This page is not available in the Basic look.

Better Support for Key Paths in the Web Assistant

The Web Assistant now sports a key-path browser which lets you add a property based on a key path to any D2W page.

Web Assistant Support for EOProject Parser

The Web Assistant now uses EOProject to parse your source code (much like WebObjects builder does) and display custom keys available on your Enterprise Objects in its key-path browser.

Confirmation Page

A new confirmation page has been introduced. The list page uses it before deleting an object. It can be frozen like any other D2W page.

Deployment Performance

The caching mechanism for D2W dynamic components has been rewritten resulting in a twofold to fivefold deployment performance improvement.

Converting Projects From Earlier Releases

Since the D2W components are now public, the Java source files, the `Main.wod` file, and the rule files have changed in this release. Two scripts are provided to ease the conversion process.

The first script, located in `$(NEXT_ROOT)/Developer/Java/Conversion/WebObjects/D2W4_5codechanges.tops`, modifies the code to conform to Direct to Web's API changes (see API Changes). You need to execute it on

- all code generated by releases of Direct to Web earlier than 4.5
- the project's `Main.wod` file

A ReadMe file in the script's directory explains how to execute the script.

The second script, located in `$(NEXT_ROOT)/Developer/Java/Conversion/WebObjects/D2W4_5modelchanges.tops`, modifies the rule files to use renamed property-level components. You need to execute it on all files in your project ending in `.d2wmodel`. See the ReadMe file in the script's directory to see how to execute the script.

API Changes

The NextPageCallback interface has been renamed to NextPageDelegate. This change affects any code that uses this interface including code generated when you freeze a component with WebObjects 4.0. The following methods are changed.

Table 4-2 ConfirmPageInterface

Removed API	Replacement API
setConfirmCallback	setConfirmDelegate
setCancelCallback	setCancelDelegate

Table 4-3 EditPageInterface, EditRelationshipPageInterface, InspecPageInteface, ListPageInterface, and QueryPageInterface

Removed API	Replacement API
setNextPageCallback	setNextPageDelegate

Table 4-4 SelectPageInterface

Removed API	Replacement API
nextPageCallback	nextPageDelegate
setNextPageCallback	setNextPageDelegate

References to the Live Assistant have been changed to Web Assistant. Consequently, the following methods are deprecated.

Table 4-5 D2W

Deprecated API	New API or Workaround
`isLiveAssistantEnabled`	`isWebAssistantEnabled`
`setLiveAssistantEnabled`	`setWebAssistantEnabled`

Table 4-6 D2WComponent

Deprecated API	New API or Workaround
`isLiveAssistantEnabled`	`isWebAssistantEnabled`

C H A P T E R 5

What's New in Enterprise Objects Framework

This chapter describes changes made to the Enterprise Objects Framework (EOF) between release 3.0 and 4.5. It describes changes made to existing features and describes new features you may want to start using in your applications.

Note: To synchronize the version numbers of WebObjects and EOF, the version number of EOF was increased from 3.0 in the last release to 4.5 in this release.

Schema Synchronization

In EOF 4.5, EOModeler supports synchronization of a database schema with the current state of a model.

Note: The Informix and ODBC adaptors do not provide schema synchronization support.

To initiate the process of synchronizing a model and schema, select "Synchronize Schema" from the Model menu. Note that before synchronizing, you need to save your model. Because the operation cannot be undone, the "Synchronize Schema" menu item is only enabled when the model has no unsaved changes.

After starting the synchroniztion process, EOModeler reverse engineers the database's schema and assembles the adaptor operations necessary to synchronize it with the model. These operations are presented to the user for confirmation before execution.

For example, if you add simple attributes (attributes representing database columns) to an entity, schema synchronization adds the columns to the underlying table. Similarly new columns or tables created in the database can be selected for incorporation into the model. External type changes for attributes and the renaming of columns and tables are also supported.

Related API Changes

EOModeler makes use of the schema synchronization API to synchronize your database with your model. You don't need to use the API yourself unless you're implementing the API for a custom adaptor.

If you do need to use or implement this API (very unlikely), see the related documentation in the following class and interface/protocol specifications:

- EOAdaptor
- EOAdaptorChannel
- EOAdaptorChannel Delegate
- EOSQLExpression

Event Logging

WebObjects 4.5 introduces event logging. The goal for this feature is to allow the measurement of how long certain operations in EOF and WebObjects take. Measurements allow you to profile an application and optimize its execution time. For this, the EOF and WebObjects frameworks instrument key portions of their code to measure the elapsed time of functions and methods.

Note: The event logging feature, related classes, and related API are not available in the com.apple.client packages. Therefore, you can't time the client side of a Java Client application.

To support this feature, EOF adds two new classes: EOEvent and EOEventCenter. An EOEvent keeps information (such as duration) about a logged event, and EOEventCenter manages the events. EOEvent is an abstract class whose subclasses are responsible for defining the events they track. For example, there are (private) subclasses for Sybase adaptor events, editing context events, WOApplication events, and so on.

To enable event logging in an application, simply open the WOEventSetup page as described in "WOEventSetup page" (page 61) and enable logging for the event classes you want to see.

In addition to the framework support, the WOExtensions framework provides components for using the feature. WOEventSetup is a page you use to configure event logging, and WOEventDisplay is a page the displays event information. Both pages can be accessed in any WebObjects 4.5 application with a direct action, as described in the following sections.

WOEventSetup page

The page used to set up the logging properties is accessed through a direct action named "WOEventSetup". So for example, you can access the WOEventSetup page for an application named "MyApp" with a URL such as the following:

```
http://myhost:aPort/cgi-bin/WebObjects/MyApp.woa/wa/WOEventSetup
```

On the WOEventSetup page, you can see all families of events that are registered for the application. Since the event classes are registered dynamically as the program executes, it is a good idea to "warm up" an application before accessing WOEventSetup.

The page lists the registered event classes, their subcategories, and a description of the kinds of events that can be logged. For instance, the EOEditingContext event class logs events for the `saveChanges` and `objectsWithFetchSpecification:` methods. Logging for each class can be enabled and disabled with the corresponding check box; it isn't possible to disable individual subcategories of an event class.

WOEventDisplay page

The page that displays collected events, WOEventDisplay, is also accessed through a direct action. For example, you can access the WOEventSetup page for an application named "MyApp" with a URL such as the following:

```
http://myhost:aPort/cgi-bin/WebObjects/MyApp.woa/wa/WOEventDisplay
```

On this page, you can view events in four different ways:

- **Raw root events**. This view, the most verbose, displays all events at the root level (events without an encompassing event). WOEventDisplay shows each event individually, which means that its possible for an event to appear multiple times if the thread of execution crosses its point more than once.
- **Aggregated root events**. This view is similar to the raw root event view, except that multiple identical events are **aggregated** or grouped in a single entry, and their combined time is displayed. In addition, the "Calls" column shows how many times an event was executed (in other words, how many events contributed to the displayed aggregate event).
- **Events grouped by page and component**. In this view, the first level of display shows only page names. By expanding a page, you get a list of components in that page. Expanding a component shows all the events within that component. This means that even events which were collected "deep" within a component are shown immediately below the component name. All identical events are aggregated as in the aggregated root event view for easier reading. It's possible to traverse the component event hierarchy by expanding the hyperlinks within a component.

  Note that since a page is also a component, a page with no dynamic subcomponents seems as if it's nested one level too deep. This is the correct behavior.
- **Events grouped by page only**. This display is similar to the grouped by page and component view, except the events do not have a by-component subgrouping.

In any of these displays, if an event or event group has subevents, it can be expanded by clicking the hyperlink or triangle image.

Each view orders events by duration (in milliseconds) from the longest to the shortest. Aggregation reduces rounding errors, which are a maximum of 1ms per event. In other words, an aggregate event consisting of ten events has at most 1ms deviation from the actual run time; however, manually adding ten individual events as displayed in the table might have up to a 10ms deviation. Therefore, any displayed sum is always more accurate than adding up the durations of individual events. Also note that the sub-events of an event branch doesn't necessarily add up to the duration of the branch event—the branch event's duration might be larger. This because the parent event generally consists of more than just calling the methods causing the sub-events.

Event System User Defaults

The event system provides three defaults for configuring its behavior:

EOEventLoggingEnabled

A boolean value that determines whether event logging is enabled. The default is NO, logging isn't enabled.

You can enable logging with EOEventLoggingEnabled, but the WOEventSetup page gives you more flexibility. The EOEventLoggingEnabled default enables logging for every class, whereas with WOEventSetup you can enable logging on a class by class basis.

EOEventLoggingLimit

An integer value that sets the maximum number of events the event system logs. The default logging limit is 200,000 per thread.

When the logging limit is reached, the event system attempts to purge old events before logging new ones. If the system is unable to purge old events, event logging is aborted.

EOEventLoggingOverflowDisplay

A boolean value that determines whether the event system logs a message when the event logging limit is reached. If enabled, the system logs messages when the event center truncates the log and also when event logging is aborted due to overflow.

Event Logging Questions and Answers

Question

What happens to an EOEvent if an exception is raised before the event is completed?

Answer

As soon as you close another event, the system detects that a previous event was not closed properly and closes it for you. All events logged between an unclosed event (due to an exception or improper coding) and the closing of another event are logged at the wrong place in the event hierarchy, but they are logged.

An improperly closed event can have another negative side effect: the improperly closed event can't be pruned with the automatic memory manager. This is virtually the only thing that can completely abort event logging.

If you have reason to believe that you might raise while an event is in progress, you should cancel the event in an exception handler.

Question

What's the overhead of enabling event logging?

Answer

The logging mechanism is extremely fast and memory efficient. A standard 300 MHz G3 system can log more than 300,000 events per second. Thus, the creation and logging of events is negligible compared to the time required to generate dynamic web pages. The only expensive operations are tree pruning when memory overflows (which takes about as long as logging ten events), and handling exceptions (which is linear to the depth of the tree—rarely more than 5 levels deep under normal circumstances). Therefore, the overhead is not really measurable.

Question

Can I enable event logging for a single-user application in production?

Answer

You shouldn't, because it would use a lot of memory: about 4 MB per thread for the default event log limit (200,000 events). Also, throughout the lifetime of an application, the system must continuously prune the event tree to keep the log size under the limit. As stated above, pruning the tree is a relatively expensive operation.

Question

Is performance impacted by the size of the event log? Does performance degrade if logging isn't reset periodically?

Answer

No. As stated above, the size of the event log is limited. Once you warm up the application and the logging framework, event logging overhead is constant—not per event, but over an average of a series of events.

Custom Event Logging

To define and log custom events, you create an event class, define the event's categories and subcategories, register the event class with the WOEvent center, and instrument the portions of code you want to log. This section describes these steps.

To create a custom event:

1. Create a subclass of EOEvent or an appropriate subclass.

   For example, to log events for a custom adaptor you've written, say MyAdaptor, create an EOEvent subclass named MyAdaptorEvent.

   Your subclass doesn't usually have to override any of the inherited methods, but you can customize the default behavior.

2. Create a description file for your event and add it to your project's Resources folder.

   An event's description file defines the event categories and subcategories used in the WOEventDisplay page. The file's contents is a dictionary in plist format. For the MyAdaptorEvent class, the file's name is `MyAdaptorEvent.description`, and it might look like the following:

   ```
   {
       EOEventGroupName = “MyAdaptor Event”;
       connect = “Connect”;
       openChannel = “Open Channel”;
       evaluateExpression = “Evaluate Expression”;
       fetchRow = “Fetch Row”;
       commitTransaction = “Commit Transaction”;
   }
   ```

   The EOEventGroupName entry is mandatory. It describes the family of events logged by the event class. Any other keys are self defined by the event class. In this example, the other keys (`connect`, `openChannel`, and so on) are the names of the events MyAdaptorEvent logs.

3. Register the event class with the EOEventCenter.

   Typically you register the event class in the `initialize` method of the class whose code you're instrumenting—MyAdaptor in this example.

   ```
   static Class MyAdaptorEventLoggingClass = Nil;
   static NSString *connectEvent = @”connect”;
   ```

```
static NSString *openChannelEvent = @"openChannel";
static NSString *evaluateExpressionEvent = @"evaluateExpression";
static NSString *fetchRowEvent = @"fetchRow";
static NSString *commitTransactionEvent = @"commitTransaction";

+ (void)initialize {
    [EOEventCenter registerEventClass:[MyAdaptorEvent class]
            classPointer:&MyAdaptorEventLoggingClass];
}
```

As in this example, you might want to define string constants for the keys in your event's description dictionary.

4. Instrument the methods.

In any method you want to instrument, add the following code, substituting the appropriate event key. This code instruments the "connect" event of MyAdaptorEvent.

```
MyAdaptorEvent *event=nil;

// Setup and start logging
if(MyAdaptorEventLoggingClass) {
    event = EONewEventOfClass(MyAdaptorEventLoggingClass, connectEvent);
    EOMarkStartOfEvent(event, nil);
}

// Code to be timed goes here.

// Finish logging.
if(event) {
    EOMarkEndOfEvent(event);
```

The second argument to `EONewEventOfClass` is an event key corresponding with an entry in the `.description` file. The corresponding value is used in the Title column of the WOEventDisplay page. If the argument isn't a key in the description dictionary, `EONewEventOfClass` uses the argument instead.

For more information on the methods used in this example, see the class descriptions for EOEvent and EOEventCenter. To see a complete example of timing events, refer to the ODBC Adaptor source code that's distributed with WebObjects.

Related API Changes

Three new classes and one interface have been added to support event logging. They are:

- EOEventCenter (`EOControl/EOEventCenter.h`)
- EOEvent (`EOControl/EOEvent.h`)
- EOAggregateEvent (`EOControl/EOAggregateEvent.h`)
- EOEventCenter.EventRecordingHandler (Java) or EOEventRecordingHandler (Objective–C; `EOControl/EOEventCenter.h`)

For more information, see the corresponding class or interface specifications.

Note: These classes and interfaces aren't available in the com.apple.client.eocontrol package.

Object Sharing

EOF 4.5 introduces a new technique for sharing read-only enterprise objects. The new subclass of EOEditingContext, EOSharedEditingContext, defines a mechanism that allows editing contexts to share enterprise objects for reading. This mechanism can reduce both the number of fetches an application makes and the amount of redundant data it requires.

As an example, consider the FeeType entity in the samle Rentals model that ships with EOF 4.5. A FeeType enterprise objects describes a type of fee that a video store can charge its customers—"Rental" and "Late", are the two FeeTypes in the sample database. It is very uncommon to add or remove FeeTypes, and it's perhaps even more uncommon to modify an existing FeeType (to rename it, for example). For the most part, FeeTypes are read-only.

With 4.5, you can fetch read-only objects such as FeeTypes into a shared editing context once, when an application starts, and all the application's sessions can share those objects. For example, objects in any session can create relationships to the shared FeeType objects even though the FeeTypes are in a different editing context from the source objects. Using previous releases, you would have to make local copies of the read-only FeeTypes in each of the editing contexts that use them.

Note: Support for shared editing contexts is not implemented in the com.apple.client packages. Therefore, shared editing contexts are not available in the client side of a Java Client application.

How It Works

The idea behind shared editing contexts is to load read-only (or read-mostly) objects into a central context that all sessions have transparent access to. It works like this.

1. A model file identifies any objects to be shared.

   Models identify shared objects by defining **shared object fetch specifications**, which define criteria for fetching objects that are to be shared. For information on creating shared object fetch specifications, see "Setting Up Object Sharing" (page 70).

   The first time your application accesses a model's entities, it checks the model for shared object fetch specifications. If any are found, they are evaluated and the corresponding fetched objects are loaded into the default shared editing context. Any existing editing contexts that don't have any registered objects begin using the default shared editing context. Similarly, any editing contexts subsequently created use the default shared editing context.

   Note that if you don't specify any shared fetch specifications, a shared editing context is never created and no object sharing occurs. Conversely, if you do specify shared object fetch specifications, a shared editing context is automatically created and object sharing is enabled for all standard editing contexts.

2. The application's shared editing context is created and populated the first time a model containing shared object fetch specifications is accessed.

   Generally an application's shared editing context is initialized when the first editing context attempts to access the database. At this time, all the application's models are loaded, and any shared object fetch specifications are detected. If any of the application's models have shared object fetch specifications, a shared editing context is created and populated and is set as the shared editing context for all empty editing contexts.

You can disable object sharing on individual editing contexts as described in "Inserting, Updating, and Deleting Shared Objects" (page 72) or you can disable object sharing altogether as described in "Disabling Sharing During Development" (page 73).

3. Standard editing contexts use objects in the shared editing context as if they were local.

   When a standard editing context fetches, any relationships to shared objects are automatically resolved to the shared editing context's objects. Similarly, an object in any standard editing context can create a relationship to a shared editing context's object.

   This all works transparently. EOEditingContext's implementations of `objectForGlobalID` and `faultForGlobalID` look for an object in the shared editing context when a standard editing context doesn't find the object locally. If the methods finds the object in the shared editing context, they return the shared object as they would if the object were local.

To allow object sharing to work, EOF makes the following assumptions:

- Shared objects are read-only (or read-mostly).
- Objects must be unique in their editing contexts and their editing context's shared editing context.

The following sections describe why these assumptions are necessary and how they are enforced.

Shared Objects Are Read-Only

If you could update a shared object, all of an application's users would see the changes immediately since all sessions share the exact same object. This behavior is undesirable. You only want users to see *committed* changes to objects. For example, suppose you make a change to a shared object in a web application, but you haven't yet saved it to the database. The changes are written to the object as soon as the request-response loop begins, and every other web user sees the change you made, even if you undo them later or make further changes before saving.

For this reason EOF enforces the read-only quality of shared objects. Shared objects can't be inserted, updated, or deleted in a shared editing context the same way that normal enterprise objects can be inserted, updated, or deleted when they're in a standard editing context. EOSharedEditingContext overrides EOEditingContext methods that mutate data (`takeValueForKey`, `saveChanges`, `deleteObject`, and `insertObject` for example) to raise exceptions. Correspondingly, methods that report on changes in a shared editing context return either `null`/`nil` or an empty array.

It would also be undesirable if you could delete a shared object. In Objective–C, it would be bad if a shared object were released while objects in standard editing contexts had relationships to it. Objects in a shared editing context are always retained. You are guaranteed that no object in a shared editing context will be destroyed while the application is running (a shared object can become invalid, but it won't go away).

Shared Objects Are Uniqued

EOF uses object uniquing to ensure that a single editing context never has more than one object with the same global ID. Because every standard editing context has access to the objects in a shared editing context as if the objects were local, none of a shared editing context's objects can have the same global ID as any object in any standard editing context.

EOF does most of the work for you. A shared editing context sends out notifications when it initializes new objects. Standard editing contexts listen for these notifications and raise exceptions if they have local objects with the same global ID as the new shared object. However, you have to take some precautions not to fetch into a shared editing context objects that have already been fetched into a standard editing context:

- Don't explicitly fetch into a standard editing context an object that is already shared *unless* you disable sharing, as described in "Inserting, Updating, and Deleting Shared Objects" (page 72).
- If a shared object has relationships to entities that aren't shared, remove the relationships from the model. Otherwise, when the faults fire, the destination objects are fetched into the shared editing context.

Setting Up Object Sharing

EOModeler has a new entity inspector for specifying the objects you want to fetch into a shared editing context when an application starts. To use it, select the entity whose objects you want to share, and open the Shared Objects Inspector.

To share all the entity's objects, select "Share all objects." If the entity has an unqualified fetch specification, that fetch specification is selected. Otherwise, EOModeler creates an unqualified fetch specification and selects it.

To share only some objects, define fetch specifications that select the objects you want to share. In the Shared Objects Inspector, select "Share objects fetched with" and select your fetch specifications.

You can also prefetch relationships into the shared editing context. For example, suppose that Product has a relationship to ProductPlatforms and that both entities should be shared. You can create a Product fetch specification that has a hint to prefetch the Product's ProductPlatforms relationship. Using this fetch specification to fetch Products into the shared editing context also causes ProductPlatforms to be loaded into the shared editing context.

Accessing Shared Objects

Shared editing contexts maintain dictionaries of all the objects they have fetched, both by entity name and by the fetch specification name they were retrieved with. This makes it easy to use shared objects for populating user interface controls. The two methods for accessing these dictionaries are:

```
objectsByEntityName
objectsByEntityNameAndFetchSpecificationName
```

These two methods make it easy to use key-value coding and key paths to bind data to WebObjects elements directly in a WebObjects `.wod` file.

For example: Say you have a model with the entity "Products" and an attribute called "isDiscontinued", which indicates whether an item is unavailable. You could set up a fetch specification in EOModeler named "regularProducts" that fetches only objects that are available (the qualifier would specify that "isDiscontinued = 0"). To use the returned set of "regular products" in a WebObjects page, create a user interface control (such as a WOPopup or WOBrowser), and set its `list` attribute like this:

```
list =
session.defaultSharedEditingContext.objectsByEntityNameAndFetchSpecificatio
nName.Products.regularProducts
```

The user interface control's list is automatically filled with data from shared objects.

Inserting, Updating, and Deleting Shared Objects

Generally shared objects are *read-mostly*. For example, an application could provide an administration mode that allows administrative users to insert, update, and delete otherwise read-only shared objects. You can still use a shared editing context in such an application, but you have to write special code. Since insertions, updates, and deletions of shared objects are typically very infrequent, the performance benefit of sharing objects can still be significant.

To insert new objects, create and insert a new object in a standard editing context. Once you do this, you need to fetch the new object into the shared editing context as described in "Refreshing the Shared Editing Context" (page 72).

Updating and deleting them is a little trickier. To modify shared objects, you need to disable object sharing for a particular editing context, as follows:

```
mySession.defaultEditingContext().setSharedEditingContext(null); // Java
[[mySession defaultEditingContext] setSharedEditingContext:nil]; // ObjC
```

A session that does this doesn't have access to shared objects and can explicitly fetch objects that would otherwise be shared. After fetching the objects, they are local, and you can update or delete them. When you save the changes, the shared editing context receives ObjectsChangedInStoreNotifications. In response to the notifications, the shared editing context refaults updated objects and removes deleted objects from its `objectsByEntityName` and `objectsByEntityNameAndFetchSpecificationName` dictionaries.

Note: Deleted objects remain in the shared editing context so that Objective–C objects having relationships to them won't be left with dangling pointers.

Refreshing the Shared Editing Context

You might want to refresh the shared editing context after an administrative user inserts a new shared object or to periodically synchronize with a database. To refresh a shared editing context, use the method `bindObjectsWithFetchSpecification` (Java) or `bindObjectsWithFetchSpecification:toName:` (Objective–C). For example:

```
EOModelGroup modelGroup = EOModelGroup.defaultGroup();
EOSharedEditingContext sharedEC =
    EOSharedEditingContext.defaultSharedEditingContext();
```

```
EOFetchSpecification fs =
    modelGroup.fetchSpecificationNamed("regularProducts", "Product");
if (fs == null) {
    System.out.println("Couldn't get regularProducts fetch specification.");
} else {
    sharedEC.bindObjectsWithFetchSpecification(fs, "regularProducts");
}
```

This has the effect of refetching all the shared objects and binding them to the `objectsByEntityName` and `objectsByEntityNameAndFetchSpecificationName` dictionaries.

Disabling Sharing During Development

Initializing a shared editing context does increase the amount of time it takes for an application to start up. During development when you starting your application frequently, the additional start up time can become a nuisance. To make debugging passes quicker, you can essentially disable object sharing using the static method `setSharedObjectLoadingEnabled` (or the equivalent class method in Objective–C).

This method specifies whether EOF looks for shared fetch specifications when it loads models. While a shared editing context can still be created and referenced, it won't contain any objects unless you programmatically fetch them into it. The advantage of disabling shared object loading is that the application starts up more quickly.

Performance

Using shared editing contexts can have the following positive performance impact on your application:

- Standard editing contexts don't have their own copies of shared objects, so the memory footprint is smaller.
- Fetching read-only data into an application is a lot simpler, so fewer database trips may be needed.

However, there are potentially negative performance effects as well. They are:

- Every editing context has to hash twice: once to look up objects in their own tables, and again to look up objects in the shared tables. Hashing is *very* fast, but with nested contexts it can add up.
- On Windows NT, locking a thread is significantly slower than it is on other platforms with similar hardware. Because shared editing contexts generate a lot of locking activity, sharing objects can actually degrade the performance of an application on NT. If you plan to deploy on NT, test your server's performance carefully.
- Fetching or firing faults in a shared editing context causes all other threads to block when they try to get shared data. To avoid this problem, try to fetch everything when the application starts up.

Multithreaded Access and Locking

WebObjects applications take care of most multithreading issues for you. This is also the case with applications that use shared editing contexts. Here are the basics of how EOF locks shared editing contexts:

- Shared editing contexts use a new lock, EOMultiReaderLock, which allows multiple contexts to read data from the shared editing context concurrently. When the shared editing context needs to update objects (because of fetching or firing faults), the context waits for all reader locks to be surrendered and then issues a writer lock. There can only be one writer at a time.
- Whenever an EOEditingContext `lock` method is called, it also obtains a reader lock for its shared editing context.
- When an EOEditingContext is about to call its parent object store, it surrenders its shared reader lock until the call is complete (this prevents deadlocks with the object store lock).

From a WebObjects perspective, everything works automatically as long as you interact with a shared editing context from within the context of a session. This is because when a web session awakes, it locks its editing context, which reader-locks the shared editing context. If you need to interact with the shared editing context outside the context of a session, you should access it through a locked EOEditingContext.

If you invoke methods on a shared editing context directly, you must obey a strict lock ordering protocol to avoid deadlocks and unsafe multithreaded access. If you directly invoke any method besides `objectsWithFetchSpecification`, `objectForGlobalID`, or `faultForGlobalID` on a shared editing context, you need to take the a reader or writer lock yourself before calling the method.

Related API Changes

A new subclass of EOEditingContext, EOSharedEditingContext, has been added to support object sharing. For more information on this class, see the corresponding class specification.

Note: With this release of EOF, there is a subclass of EOEditingContext for the first time. You might have to adjust your code if you ever assume than an editing context is always an instance of EOEditingContext.

Note: Support for shared editing contexts is not implemented in the com.apple.client packages.

Additionally, new API has been added to existing classes as summarized in the following tables.

Table 5-1 EODatabaseContext (EOAccess/EODatabaseContext.h)

New or Changed API	Description
`setSharedObjectLoadingEnabled` (Java) `setSharedObjectLoadingEnabled:` (Objective–C)	A static/class method that sets according to the specified flag whether or not to automatically load enterprise objects into the default shared editing context when a database context loads a model. The default is `true`/`YES` (the database automatically loads shared objects).
`isSharedObjectLoadingEnabled`	A static/class method that returns `true`/`YES` if database contexts automatically load enterprise objects into the default shared editing context when database contexts load models, `false`/`NO` otherwise.

Table 5-2 EOEntity (EOAccess/EOEntity.h)

New or Changed API	Description
`setSharedObject FetchSpecificationsByName` (Java) `setSharedObject FetchSpecificationsByName:` (Objective–C)	Sets the fetch specifications used to load objects into a shared editing context to the fetch specifications identified by name in the specified array.
`sharedObject FetchSpecificationNames`	Returns an array of strings, which are the names of the fetch specifications used to load objects into a shared editing context.
`addSharedObjectFetchSpecification ByName` (Java) `addSharedObjectFetchSpecification ByName:` (Objective–C)	Adds the fetch specification identified by the specified name to the set of fetch specifications used to load objects into a shared editing context.
`removeSharedObject FetchSpecificationByName` (Java) `removeSharedObject FetchSpecificationByName:` (Objective–C)	Removes the fetch specification identified by the specified name from the set of fetch specifications used to load objects into a shared editing context.

Table 5-3 EOModel (EOAccess/EOModel.h)

New or Changed API	Description
`entitiesWithSharedObjects`	Returns an array of entities that have objects to load into a shared editing context.

Table 5-4 EOModelGroup (EOAccess/EOModelGroup.h)

New or Changed API	Description
`entitiesWithSharedObjects`	Returns an array of entities that have objects to load into a shared editing context.

Table 5-5 EOEditingContext (EOControl/EOEditingContext.h)

New or Changed API	Description
`setSharedEditingContext` (Java) `setSharedEditingContext:` (Objective–C)	Sets the receiver's shared editing context. If the receiver is listening for (`EO)DefaultSharedEditingContextWasInitializedNotification`, it removes itself as an observer. By default, the shared editing context is `null`/`nil` but is set when an `(EO)DefaultSharedEditingContextWasInitializedNotification` is posted. Changing the shared editing context to `null`/`nil` allows the receiver to obtain private, editable copies of objects that would otherwise be shared. If both the receiver and the specified shared editing context have registered objects, the objects of both contexts are compared to verify that the objects are unique. If the objects are not unique, an exception is raised by the editing context.
`sharedEditingContext`	Returns the shared editing context used by the receiver.

Subclassing EOGenericRecord

EOF 4.5 adds a new option for creating custom enterprise objects: rather than creating a subclass of EOCustomObject (Java) or NSObject (Objective–C), you can now subclass EOGenericRecord.

This feature is most significant in applications that use the Java bridge. By default, a subclass of EOGenericRecord stores its properties in a dictionary on the Objective–C side of the bridge instead of in individual instance variables on the Java side. This allows EOF to access enterprise object properties with many fewer trips across the bridge, which reduces memory usage and improves performance.

More specifically, subclassing EOGenericRecord provides the following advantages over subclassing EOCustomObject (Java):

- Java enterprise objects don't incur a performance penalty relative to Objective–C. Previously, each call to an EOKeyValueCoding method (such as `valueForKey`) resulted in a bridged method invocation and possibly the creation of a new object. Because EOGenericRecord provides the storage for an enterprise object's values, the values don't need to cross the bridge during a fetch. This results in a dramatic performance improvement in EOKeyValueCoding-bound operations such as fetching, snapshotting, and validation.
- Java enterprise objects don't use more memory than Objective–C enterprise objects. Since an enterprise object's values were previously stored in its instance variables, each value object had to be morphed between Java and ObjC when interacting with the EOF frameworks. This meant that the values couldn't be shared between an enterprise object and its snapshots in the EOEditingContext and EODatabase. Consequently, many more objects were resident in memory than would be the case in the non-bridged case.
- Enterprise object classes are more flexible and easier to maintain. As you edit your model by adding or removing keys, you only have to update your enterprise object class if you want to add or remove custom accessors or validation methods. You don't have to maintain instance variables since property values are stored in a dictionary.

  Also, you can use a specific subclass of EOGenericRecord for multiple entities as you can when you use EOGenericRecord. In previous releases, a single enterprise object class (other than EOGenericRecord) could correspond to only one entity. Subclasses of EOGenericRecord can now correspond to multiple entities the same way EOGenericRecord can. This might be useful for providing common behavior such as validation logic. Note that a "generic" subclass of EOGenericRecord (that is, a subclass which is shared among entities) *must* be instantiated using EOClassDescription's `createInstanceWithEditingContext` method in the same way that EOGenericRecord must. If it isn't, the resulting instance won't be properly initialized.
- Subclasses of EOGenericRecord automatically use deferred faulting, which increases performance and decreases memory usage even further. For more information on deferred faulting, see "Deferred Faulting" (page 81).

Given all the advantages of subclassing EOGenericRecord, you might wonder if there's ever a reason to subclass EOCustomObject. Subclassing EOGenericRecord usually yields better performance. Even if it doesn't yield *better* performance, you can design it to be at least as fast and to have the same level of functionality as a subclass of EOCustomObject by storing properties in instance variables. Therefore, you really don't ever need to subclass EOCustomObject.

Property Storage: Dictionary or Instance Variables

A subclass of EOGenericRecord can take three approaches to storing property values:

1. Store them in the EOGenericRecord dictionary.
2. Store them in instance variables.
3. Store them in a combination of the above.

Generally the first approach is the best because it reduces the number of trips across the Java bridge, improving performance and reducing memory usage. Enterprise objects are accessed from Objective–C much more frequently than from Java. A typical enterprise object is populated from Objective–C when it's fetched from the database, snapshotted from Objective–C each time the object is changed, and accessed from Objective–C to provide values for bindings in WebObjects components.

On the other hand, access from Java is typically limited to custom validation (implemented in the enterprise object class) and to accesses by other enterprise objects to obtain property values explicitly (for example, to check the department budget before validating a new salary).

The second approach might be appropriate if your application accesses and manipulates enterprise objects from Java quite frequently, however this case is probably rare.

Note: When you create an instance variable for an EOGenericRecord subclass, ensure that the accessor methods read and write the instance variable instead of invoking `valueForKey` and `takeValueForKey`, which is what the implementations do if the code is generated by EOModeler.

The third approach is to create instance variables for some of your enterprise object's properties and leave the rest in the EOGenericRecord dictionary. This approach is the most difficult to maintain, because the implementation of accessor methods depends on the way the property is stored.

Creating a Subclass

EOModeler has been updated with templates that take advantage of this feature, so new enterprise object classes are automatically subclasses of EOGenericRecord. To re-parent existing business objects, perform the following steps in your source files:

1. Set the enterprise object class's superclass to EOGenericRecord.
2. Delete all EOKeyValueCoding-related instance variables (that is, instance variables that store values for attribute and relationship keys).
3. (Optional) For maximum performance, delete the three-argument constructor if you don't use the arguments. If you need custom logic in your constructor but don't need the three arguments, implement a custom default constructor. If you don't need a custom constructor, don't implement one at all; the compiler inserts the default constructor for you. If the three-argument constructor is present, EOF uses it; otherwise the default constructor is invoked.
4. (Optional) For maximum performance, delete all accessors that don't contain custom logic. If you want strong typing to simplify business logic, implement accessors using `storedValueForKey`, such as:

   ```
   public NSGregorianDate dateReleased() {
       return (NSGregorianDate)storedValueForKey("dateReleased");
   }
   ```

In lieu of (or in addition to) adding custom accessors, it might be convenient to add and use String constants for an enterprise object class's keys, as shown in the following:

```
public class Movie extends EOGenericRecord {
    public static final String Title = "title";
    public static final String DateReleased = "dateReleased";
    public static final String Studio = "studio";

    public NSGregorianDate dateReleased() {
        return (NSGregorianDate)storedValueForKey(DateReleased);
    }
}
```

Deferred Faulting

EOF uses faults as stand-ins for objects whose data has not yet been fetched. Although fault creation is much faster than fetching, fault instantiation still takes time. To improve performance, EOF 4.5 has the ability to use **deferred faults** (which are more efficient) for enterprise object classes that enable the feature.

In an object whose class enables deferred faulting, the object's relationships are initially set to deferred faults. For a particular relationship, a single deferred fault is shared between all instances of an enterprise object class. This sharing of deferred faults can significantly reduce the number of faults that need to be created, and usually reduces the overhead of fault creation during a fetch.

For example, consider a Movie class with a `studio` relationship. Assuming the worst case in which each movie has a different studio, without deferred faulting, during a fetch of twenty Movie objects, twenty faults are created for the `studio` relationship—one fault for each movie. With deferred faulting, only one fault is created—a deferred fault that is shared by all the movies.

Deferred faults have a special fault handler, which knows how to replace the deferred fault with a standard fault. Once the deferred fault is replaced with a normal fault, the normal faulting behavior applies.

In 4.5, EOGenericRecord enables deferred faulting; you get the behavior without making any changes to existing code. Non-EOGenericRecord enterprise object classes don't enable deferred faulting by default. To enable it on a custom class, implement the class method `useDeferredFaultCreation` to return `true`/`YES` (the default is `false`/`NO`). Additionally, invoke the method `willReadRelationship` before accessing a relationship that might be a deferred fault. The `willReadRelationship` method allows the special fault handler to instantiate a normal fault before it is accessed.

Deferred Faulting and Inheritance

In Java programming, there's an additional benefit of deferred faulting: Deferred faulting allows you to have to-one relationships into an inheritance hierarchy. Without deferred faulting, a to-one relationship to a non-leaf entity is impossible without implementing workarounds (because of strong typing in the Java language). The workarounds are not necessary if you use deferred faulting. For more information on the inheritance limitation in Java and the workarounds, refer to the *Enterprise Objects Framework Developer's Guide*.

Related API Changes

New API added to support the deferred faulting feature is summarized in this section.

Table 5-6 EODeferredFaulting interface/informal protocol (EOControl/EOFault.h)

New or Changed API	Description
`useDeferredFaultCreation`	A static/class method that defaults to `false`/`NO`. Override to return `true`/`YES` on your enterprise object class to use deferred fault creation. EOGenericRecord's implementation returns `true`/`YES`, indicating that it uses the more efficient deferred fault technique. Note that in Java, this static method is not a formal part of the EODeferredFaulting interface because the Java language doesn't support the inclusion of static methods in an interface. It is, however, an informal part of the interface. A static method implemented in a custom enterprise object class will be invoked automatically to determine whether or not to use deferred fault creation.
`willReadRelationship` (Java) `willReadRelationship:` (Objective–C)	Invoke this before using any relationship in an enterprise object that uses deferred fault creation. This method sets the corresponding instance variable of the receiver using `takeStoredValueForKey`/`takeStoredValue:forKey:`. Returns a newly instantiated fault if the object is a fault and has a deferred fault handler. Returns the object otherwise.

Table 5-7 EOFaultHandler (EOControl/EOFaultHandler.h)

New or Changed API	Description
`createFaultForDeferredFault` (Java) `createFaultForDeferredFault:sourceObject:` (Objective–C)	Invoked by `willReadRelationship` to ensure that a fault is valid. EOFaultHandler's implementation simply returns its fault.

Snapshot Reference Counting

Snapshot reference counting is a new feature that removes snapshots from an EODatabase when they are no longer used by any enterprise objects in an application. This feature reduces the memory footprint of WebObjects applications.

Note: Snapshot reference counting is not available in the client side of Java Client applications. The corresponding API is in the Java Client packages, but the snapshots are not released.

The reference count on a snapshot is implicitly incremented when you create an enterprise object, either by fetching it in a batch or by faulting it in. An EOEditingContext decrements the reference count automatically when it forgets or refaults an enterprise object.

To support this feature, the method `editingContextDidForgetObjectWithGlobalID`/ `editingContext:didForgetObjectWithGlobalID:` has been added to EOObjectStore. This method is not intended to be called directly by your application.

As a developer you should not worry about this feature; it should just work without any additional code on your part. There are only a few cases where it could be incompatible with pre-EOF 4.5 applications:

- When the snapshots are removed because their reference counts fall to zero, the globalID associated with these snapshots are also released. If your application is written in Objective–C and you cache globalIDs in your application, make sure you retain them. If you don't, your application will crash with "Message sent to freed object" errors when you try to access globalIDs that have been freed.

Previously GlobalIDs were never released; they were kept with the snapshots. This is not a problem if you use Java as your development language, since Java automatically retains all referenced objects.

- If your application assumes that snapshots for fetched objects are always present, you could have a problem. This is no longer the case, and asking for a snapshot might now return `null`/`nil`.

Sometimes you want a snapshot to stay around for the lifetime of your application. If you need this functionality there are three ways to force selected snapshots to stay around:

- Use the method `incrementSnapshotCountForGlobalID` on EODatabase to force the snapshot to stay around. If you use this approach, make sure to call `decrementSnapshotCountForGlobalID` when you don't need the snapshot anymore.
- Fetch the objects whose snapshots you need. (Note that if your application is written in pure Objective–C and it's not multithreaded, the EOEditingContext doesn't retain its enterprise objects—you have to retain them yourself.)
- Turn on entity caching for the entity. This automatically increments the reference count for all the snapshots your entity caches.

You can turn the whole feature off and revert to the previous behavior of retaining snapshots forever by invoking the class method `disableSnapshotRefCounting` on EODatabase. Make sure to call this method early in your application initialization code, before the creation of any EODatabase instances.

Related API Changes

The following tables summarize new or changed API to support the snapshot reference counting feature.

Table 5-8 EODatabase (EOAccess/EODatabase.h)

New or Changed API	Description
`incrementSnapshotCountForGlobalID` (Java) `incrementSnapshotCountForGlobalID:` (Objective–C)	If the receiver releases unreferenced snapshots, increments the reference count for the shared snapshot associated with the specified globalID.
`decrementSnapshotCountForGlobalID` (Java) `decrementSnapshotCountForGlobalID:` (Objective–C)	If the receiver releases unreferenced snapshots, decrements the reference count for shared snapshot associated with the specified globalID; if no more objects refer to the snapshot, removes it from the snapshot table.
`disableSnapshotRefCounting`	A static/class method that disables the snapshot reference counting feature so that instances don't release snapshots.

Table 5-9 EOObjectStore (EOControl/EOObjectStore.h)

New or Changed API	Description
`editingContextDidForgetObject WithGlobalID` (Java) `editingContext:didForgetObjectWithGlobalID:` (Objective–C)	Invoked to inform the object store that it can stop keeping data about an object it passed to a child. Don't invoke this method; it is invoked automatically by the Framework.

Snapshot Timestamping

EOF caches database snapshots and uses the cached values to initialize objects. This significantly improves performance, since using the cached values is much faster than making round-trips to the database for fetches. However, this behavior can lead to staleness of the cached data; and it sometimes produces confusing results, especially when new values available from a fetch are ignored in favor of the stale snapshots.

A new snapshot timestamping feature updates snapshots appropriately when fetching and allows an editing context to request that the snapshots used to build enterprise objects are no older than the editing context's `fetchTimestamp`. The default value for the `fetchTimestamp` of a new editing context is one hour earlier than the creation time of the editing context. So any snapshots that are less than an hour old are acceptable to the editing context; older cached values are ignored. The default "lag" can be adjusted using the static method `setDefaultFetchTimestampLag` (or the corresponding Objective–C class method), and the `fetchTimestamp` for a specific editing context can be set directly with `setFetchTimestamp`.

Note that the `fetchTimestamp` is significant only when fetching data (typically by sending `objectsWithFetchSpecification`). An existing enterprise object is unaffected by changes to the editing context's `fetchTimestamp`.

When an EODatabase records a snapshot, it now also records a timestamp for that snapshot. The timestamp is an NSTimeInterval (relative to the reference date as documented for the NSDate class). Methods that return a snapshot from EODatabase or EODatabaseContext now have variants that take an extra argument to specify a minimal timestamp for the snapshot. If the recorded snapshot's timestamp is earlier than the requested time, `null`/`nil` is returned. If the snapshot is recent enough for the request, the snapshot is returned as usual.

Related API Changes

The following tables summarize new API added to support the snapshot timestamp feature.

Note: Support for snapshot timestamping is not available in the com.apple.client packages. However, since snapshots are maintained only on the server, no timestamping takes place on the client side of a Java Client application. In other words, this feature can be used to its fullest potential in a Java Client application.

Table 5-10 EODatabase (EOAccess/EODatabase.h)

New or Changed API	Description
`DistantPastTimeInterval`(Java) `EODistantPastTimeInterval` (Objective–C) (constant)	The NSTimeInterval used as a lower bound on timestamps.
`setTimestampToNow`	Sets the internal timestamp to value returned by NSDate's `timeIntervalSinceReferenceDate` method. Used for recording subsequent snapshots.

`snapshotForGlobalID(EOGlobalID)`
`snapshotForGlobalID(`
`EOGlobalID,`
`double)`
(Java)
	The new, overloaded version of this method takes a double argument to use as a timestamp. It returns the snapshot associated with the specified globalID. Returns `null` if there isn't a snapshot or if its timestamp is less than the specified timestamp.
	The old version that only takes a globalID has been reimplemented to invoke the new version using `DistantPastTimeInterval` as the timestamp.

Table 5-10 EODatabase (EOAccess/EODatabase.h) (continued)

New or Changed API	Description
`snapshotForSourceGlobalID( EOGlobalID, String)` `snapshotForSourceGlobalID( EOGlobalID, String, double)` (Java)	The new, overloaded version of this method takes a double argument to use as a timestamp. It returns the to-many snapshot for the specified globalID and relationship name. Returns `null` if there isn't a to-many snapshot or if the timestamp is less than the specified timestamp. The old version that only takes a globalID and a String has been reimplemented to invoke the new version using `DistantPastTimeInterval` as the timestamp.
`snapshotForGlobalID:after:` (Objective–C)	Returns the snapshot associated with the specified globalID. Returns `nil` if there isn't a snapshot or if its timestamp is less than the specified timestamp. Note that `snapshotForGlobalID:` has been reimplemented to invoke `snapshotForGlobalID:after:` with `EODistantPastTimeInterval` as the timestamp.
`snapshotForSourceGlobalID:relationshipName:after:` (Objective–C)	Returns the to-many snapshot for the specified globalID and relationship name. Returns `nil` if there isn't a to-many snapshot or if the timestamp is less than the specified timestamp. Note that `snapshotForSourceGlobalID:relationshipName:` has been reimplemented to invoke `snapshotForSourceGlobalID:relationshipName:after:` with `EODistantPastTimeInterval` as the timestamp.
`timestampForGlobalID` (Java) `timestampForGlobalID:` (Objective–C)	Returns the timestamp of the snapshot for the specified globalID. Returns `(EO)DistantPastTimeInterval` if there isn't a snapshot.
`timestampForSourceGlobalID` (Java) `timestampForSourceGlobalID:relationshipName:` (Objective–C)	Returns the timestamp of the to-many snapshot for the specified globalID. Returns `(EO)DistantPastTimeInterval` if there isn't a snapshot.

Table 5-11 EODatabaseContext (EOAccess/EODatabaseContext.h)

New or Changed API	Description

`snapshotForGlobalID(EOGlobalID)`
`snapshotForGlobalID(`
`    EOGlobalID,`
`    double)`
(Java)
	The new, overloaded version of this method takes a double argument to use as a timestamp. It returns the snapshot associated with the specified globalID. Returns `null` if there isn't a snapshot or if its timestamp is less than the specified timestamp. Searches first locally (in the transaction scope) and then in the receiver's EODatabase.
	The old version that only takes a globalID has been reimplemented to invoke the new version using `DistantPastTimeInterval` as the timestamp.

Table 5-11 EODatabaseContext (EOAccess/EODatabaseContext.h)

New or Changed API	Description
`snapshotForSourceGlobalID( EOGlobalID, String)` `snapshotForSourceGlobalID( EOGlobalID, String, double)` (Java)	The new, overloaded version of this method takes a double argument to use as a timestamp. It returns the to-many snapshot for the specified globalID and relationship name. Returns `null` if there isn't a to-many snapshot or if the timestamp is less than the specified timestamp. The old version that only takes a globalID and a String has been reimplemented to invoke the new version using `DistantPastTimeInterval` as the timestamp.
`snapshotForGlobalID:after:` (Objective–C)	Returns the snapshot associated with the specified globalID. Returns `nil` if there isn't a snapshot or if its timestamp is less than the specified timestamp. Searches first locally (in the transaction scope) and then in the receiver's EODatabase. Note that `snapshotForGlobalID:` has been reimplemented to invoke `snapshotForGlobalID:after:` with `EODistantPastTimeInterval` as the timestamp.
`snapshotForSourceGlobalID:relationshipName:after:` (Objective–C)	Returns the to-many snapshot for the specified globalID and relationship name. Returns `nil` if there isn't a to-many snapshot or if the timestamp is less than the specified timestamp. Note that `snapshotForSourceGlobalID:relationshipName:` has been reimplemented to invoke `snapshotForSourceGlobalID:relationshipName:after:` with `EODistantPastTimeInterval` as the timestamp.

Table 5-12 EOEditingContext (EOControl/EOEditingContext.h)

New or Changed API	Description
`setDefaultFetchTimestampLag` (Java) `setDefaultFetchTimestampLag:` (Objective–C)	A static/class method that assigns the default timestamp lag for new editing contexts. The default value is 3600.0 seconds (one hour). When a new editing context is initialized, it is assigned a fetch timestamp equal to the current time less the default timestamp lag. Setting the lag too high might cause every new editing context to accept very old cached data. Setting the lag too low might degrade performance due to excessive fetching. A negative lag value is treated as 0.0.
`defaultFetchTimestampLag` (Java) `defaultFetchTimestampLag` (Objective–C)	A static/class method that returns the default timestamp lag.
`setFetchTimestamp` (Java) `setFetchTimestamp:` (Objective–C)	Sets the receiver's fetch timestamp. When an editing context fetches objects from its parent object store, the parent object store can use the timestamp to determine whether to use cached data or to refetch the most current values. An editing context prefers that fetched values are at least as recent as its fetch timestamp. Note that the parent object store is free to ignore the timestamp; so this value should be considered a hint or request and not a guarantee. Changing the fetch timestamp has no effect on existing objects in the editing context; it can affect only subsequent fetches. To refresh existing objects, invoke `refaultObjects` before you invoke `setFetchTimestamp`. The initial value for the fetch timestamp of a new non-nested editing context is the current time less the `defaultFetchTimestampLag`. A nested editing context always uses its parent's fetch timestamp. `setFetchTimestamp` raises if it's invoked on a nested editing context.
`fetchTimestamp`	Returns the receiver's fetch timestamp.

Handling Missing Faults

In previous versions of EOF, if a fault fired but no corresponding database row could be found (for example because of a referential integrity problem or because the row was deleted without EOF's knowledge), the delegate method `databaseContextFailedToFetchObject`/ `databaseContext:failedToFetchObject:globalID:` was called. If the delegate didn't fix the problem, an exception was raised immediately. In 4.5, The delegate is invoked, but the exception is delayed or avoided, and an empty enterprise object is returned. If the application later tries to save an object graph that requires the missing fault, the exception is raised during `saveChanges`. If the object is never needed, no exception is raised.

Related API Changes

The following tables summarize new API added to support the better handling of missing faults.

Table 5-13 EODatabaseContext (EOAccess/EODatabaseContext.h)

New or Changed API	Description
`missingObjectGlobalIDs`	Returns the globalIDs of any "missing" enterprise objects. Returns an empty array if no missing objects are known to the receiver. An object is "missing" when a fault fires and the corresponding row for the fault isn't found in the database. To be notified when a missing object is discovered, implement the delegate method `databaseContextFailedToFetchObject`/ `databaseContext:failedToFetchObject:globalID:`. If an application tries to save a missing object, an exception is raised.

Table 5-14 EODatabaseContext Delegate (EOAccess/EODatabaseContext.h)

New or Changed API	Description
`databaseContextFailedToFetchObject` (Java) `databaseContext: failedToFetchObject: globalID:` (Objective–C) (changed behavior)	Invoked when a to-one fault can't find its data in the database. Now if the method returns `false`/`NO`, the specified database context doesn't immediately raise as before. Instead, it simply tracks the globalID of the offending object. If the tracked globalID is in the list of updated objects when `prepareForSaveWithCoordinator`/ `prepareForSaveWithCoordinator:editingContext:` is invoked (`saveChanges` invokes this method), an exception is raised. To get a list of the objects that failed to fetch, see the method `missingObjectGlobalIDs`.

Automatic Database Reconnection

In EOF 4.5, a concrete adaptor can now implement methods that cause EOF to automatically attempt to reconnect to a database server when a connection is unexpectedly dropped. This behavior handles the problem of transient communication failures. By default reconnection is attempted by all of the adaptors that ship with EOF 4.5.

EOF now sends `isDroppedConnectionException` to the adaptor if an exception is raised during fetching or saving. If the adaptor returns `true`/`YES`, then it attempts to reconnect to the database and retry the operation. (The adaptor context must also implement `handleDroppedConnetion` to clean up the state of the context and its channels before the reconnection is attempted.) If the reconnection attempt fails, the exception from the failure is raised as usual.

The delegate method `reconnectionDictionaryForAdaptor` can be used to provide a new connection dictionary for the reconnection attempt. If the delegate is not implemented, the adaptor uses its existing connection dictionary when reconnecting to the server.

You can completely override the database reconnection behavior with the delegate method `databaseContextShouldHandleDatabaseException` (Java) or `databaseContext: shouldHandleDatabaseException` (Objective–C). For more information, see Table 5-20 (page 97).

Related API Changes

The following tables summarize new API added to support the database reconnection feature.

Table 5-15 EOAdaptor (EOAccess/EOAdaptor.h)

New or Changed API	Description
`handleDroppedConnection`	The adaptor cleans up after a dropped connection by sending `handleDroppedConnection` to all of its adaptor contexts and then clearing its array of contexts. If the receiver's delegate implements `reconnectionDictionaryForAdaptor`, that method is invoked and the result is used as the new connection dictionary for the adaptor. Otherwise, the adaptor attempts new connections using the original connection dictionary. You should never invoke this method; it is invoked automatically by the Framework. Subclasses don't normally need to override the superclass implementation.
`isDroppedConnectionException` (Java) `isDroppedConnectionException:` (Objective–C)	Returns `true`/`YES` if the exception is one that the adaptor can attempt to recover from by reconnecting to the database, `false`/`NO` otherwise. The default implementation returns `false`/`NO`. Subclasses should implement it to allow for automatic database reconnection.

Table 5-16 EOAdaptor Delegate (EOAccess/EOAdaptor.h)

New or Changed API	Description
`reconnectionDictionaryForAdaptor` (Java) `reconnectionDictionaryForAdaptor:` (Objective–C)	Invoked from `handleDroppedConnection`. If this method returns a non-`null`/`nil` value, the value is used as the adaptor's new connection dictionary. The delegate is responsible for guaranteeing that the new connection dictionary is compatible with any EODatabase that is using the adaptor. If reconnection succeeds, the EODatabase continues to use its database snapshots as if nothing had happened. Therefore, the new database server should have the same data as the original.

Table 5-17 EOAdaptorContext (EOAccess/EOAdaptorContext.h)

New or Changed API	Description
`handleDroppedConnection`	Implemented by subclasses to provide automatic database reconnection support. If database reconnection is not supported, subclasses don't have to implement it. Subclass implementations should clean up the state of the adaptor context and its associated adaptor channels so that they can be safely released and deallocated without any errors. Don't invoke this method; it's invoked automatically by the Framework.

Table 5-18 EODatabase (EOAccess/EODatabase.h)

`handleDroppedConnection`	Received when a dropped connection is detected to initiate cleanup. It cleans up by sending `handleDroppedConnection` to its adaptor, and then sending `handleDroppedConnection` to all of its registered database contexts. When the cleanup procedure is complete, the Framework can automatically reconnect to the database. Don't invoke this method; it's invoked automatically by the Framework.

Table 5-19 EODatabaseContext (EOAccess/EODatabaseContext.h)

`handleDroppedConnection`	Cleans up after a dropped connection by effectively releasing adaptor context and database channels, and then creating a new adaptor context. Don't invoke this method; it's invoked automatically by the Framework.

Table 5-20 EODatabaseContext Delegate (EOAccess/EODatabaseContext.h)

`databaseContextShouldHandleDatabaseException` (Java) `databaseContext:shouldHandleDatabaseException` (Objective–C)	Invoked when an exception is thrown in response to a lost database connection. Implement this method only if you want to override the default reconnection behavior. If the delegate method is not implemented, the reconnection decision is made according to the adaptor's response to `isDroppedConnectionException` (`isDroppedConnectionException:` in Objective–C). The implementation of this method should inspect the exception to determine if the exception is a response to a dropped connection. If not, the delegate should simply raise the specified exception. If the exception is in response to a dropped connection, the method can return `true`/`YES` to allow the database context to handle the exception by automatically reconnecting to the database or can return `false`/`NO` to customize connection behavior. If the delegate returns `false`/`NO`, then the delegate is responsible for handling the exception and implementing an appropriate reconnection strategy. The database context retries the operation that generated the original exception without doing any additional clean up and without attempting to reconnect to the database.

Setting Access Layer Delegates

EOAdaptor, EOAdaptorContext, and EODatabaseContext now implement the static method `setDefaultDelegate` to simplify the setting of delegates for new instances of those classes (there are corresponding class methods in Objective–C). By default, the default delegate for those classes is `null`/`nil`. However, `setDefaultDelegate` can be used to establish a default delegate for a particular class that will be assigned to any new instance of that class when the instance is initialized.

Related API Changes

The following tables summarize new API added to support the database reconnection feature.

Table 5-21 EOAdaptor (EOAccess/EOAdaptor.h)

New or Changed API	Description
`setDefaultDelegate` (Java) `setDefaultDelegate:` (Objective–C)	A static/class method that sets the default delegate for all newly created EOAdaptor instances. That is, specifies the object that is assigned as delegate to new adaptor objects.
`defaultDelegate`	A static/class method that returns the default delegate.

Table 5-22 EOAdaptorContext (EOAccess/EOAdaptorContext.h)

New or Changed API	Description
`setDefaultDelegate` (Java) `setDefaultDelegate:` (Objective–C)	A static/class method that sets the default delegate for all newly created EOAdaptorContext instances (and their EOAdaptorChannels). That is, specifies the object that is assigned as delegate to new adaptor context objects and their channels.
`defaultDelegate`	A static/class method that returns the default delegate.

Table 5-23 EODatabaseContext (EOAccess/EODatabaseContext.h)

New or Changed API	Description
`setDefaultDelegate` (Java) `setDefaultDelegate:` (Objective–C)	A static/class method that sets the default delegate for all newly created EODatabaseContext instances. That is, specifies the object that is assigned as delegate to new database context objects.
`defaultDelegate`	A static/class method that returns the default delegate.

Key Value Coding Changes

EOF 4.5 introduces two improvements to key-value coding: The addition of key binding objects and the enforcement of lowercase key names. The follow sections describe each.

Key Bindings

New key-value coding primitives have been introduced. The new primitives are based on **bindings**, which associate a class/key pair to a mechanism for accessing the key. There are two types of bindings, **get bindings** and **set bindings**. They are represented by the new class EOKeyBinding.

To access an object's data with key-value coding, clients typically use the EOKeyValueCoding and EOKeyValueCodingAdditions methods `valueForKey`, `takeValueForKey`, and so on. However, in EOF 4.5, clients can optimize access by obtaining and caching bindings for particular class/key combinations and applying those bindings directly. Classes can override the EOKeyValueCoding methods (such as `valueForKey)` directly, but overriding `keyValueBindingForKey` to return an appropriately initialized binder object provides the maximum performance benefit.

Note: The need to override the default key-value coding methods, particularly the new binding primitives, is extremely rare. The default behavior is very efficient and should be well suited to almost any application.

Clients caching bindings are responsible for checking that the class of the object to which a binding is applied matches the target class of the binding. Using a binding obtained from one class on an object of another results in undefined behavior.

Enforcing Lowercase Key Names

EOF expects keys to begin with a lowercase letter. It now logs a warning if that restriction is violated. For backwards compatibility with previous releases which did not strictly check capitalization, you can use the EOKeyValueCoding.KeyBinding static method `suppressCapitalizedKeyWarning` to suppress the warning for capitalized keys (EOKeyBinding class method in Objective–C). However, note that this method is deprecated and will be removed in a future release.

Related API Changes

The following new classes have been added to support key value bindings:

- EOKeyValueCoding.KeyBinding (Java) and EOKeyBinding (Objective–C)
- EOKeyValueCoding.KeyGetBinding (Java) and EOKeyGetBinding (Objective–C)
- EOKeyValueCoding.KeySetBinding (Java) and EOKeySetBinding (Objective–C)

For more information, see the class specification for EOKeyValueCoding.KeyBinding/EOKeyBinding.

In addition, the EOKeyValueCoding interface/informal protocol has been enhanced to create and return key bindings. In Java, the new methods are defined in a new interface called EOKeyValueCoding.KeyBindingCreation. In Objective–C, the new methods are defined in `EOKeyValueCoding.h`. For more information, see the EOKeyValueCoding.KeyBindingCreation interface specification (Java) or the EOKeyBindingCreation informal protocol specification (Objective–C).

Recursive Reader and Writer Locks

A new class, EOMultiReaderLock, provides EOF with recursive reader and writer locks. The locks are recursive; a single thread can request a lock many times, but a lock is actually taken only on the first request. Likewise, when a thread indicates it's finished with a lock, it takes an equal number of unlock calls to return the lock.

There is no limit on the number of reader locks that can be taken by a process. However, there can only be one writer lock at a time, and a writer lock is not issued until all reader locks are returned. (Reader locks aren't issued to new threads when there is a thread waiting for a writer lock, but threads that already have a reader lock can increment their lock count.)

Thread safety is maintained by using mutex locks (binary semaphores), which ensures that no more than one critical section of the class can be processed at a time. The queueing order of requests for writer locks is not managed by the class; the underlying implementation of mutex signaling manages the queue order.

EOMultiReaderLock correctly handles promotion of a read lock to a write lock, and the extension of a reader lock to the current writer. This prevents a thread from deadlocking on itself when requesting a combination of lock types.

EOMultiReaderLocks are slightly more time-expensive than NSRecursiveLocks because the recursion count has to be stored per-thread, causing each request for a reader lock to incur a hash. Writer locks are even more expensive because EOMultiReaderLock must poll the hashtable until all reader locks have been returned before the writer lock can be taken.

Related API Changes

A new class, EOMultiReaderLock, has been added to the Framework to support the new multi reader and writer lock feature. For more information, see the corresponding class specification.

Note: This class doesn't exist in com.apple.client.eocontrol. Multithreaded clients aren't yet supported in Java Client applications. All the client-side locks in Java Client application's are no-ops.

LDAP Adaptor Example

EOF 4.5 comes with a new sample adaptor: the LDAP adaptor. It provides read, modify, and delete access and limited support for inserting. Additionally it provides a simple way to verify a user's password on the Web with an LDAP server without requiring a model.

LDAP Client Libraries

The LDAP adaptor is ready to use when you install WebObjects. The source code is provided so that you can use a different client library or enhance the adaptor yourself.

The LDAP adaptor consists of two pieces: The adaptor framework and an LDAP client library. Apple ships a client library based on the public University of Michigan LDAP Client. You can build and install this library as a framework, or you can substitute your own client library, such as the Netscape LDAP library. (The latter has not been tested, but both libraries conform to RFC 1823, the LDAP client API).

Creating Models

You create a model for an LDAP server with EOModeler the way you create models for other adaptors. Simply choose New from the Model menu, and choose LDAP as the adaptor for the new model.

Note: If all you are trying to do is authenticate users using an LDAP server's usernames and encrypted passwords, you don't need to create a model. See "Performing Authentication" (page 105).

Logging In

The LDAP adaptor defines the following connection keys, which are represented in the login panel:

Server Name
: The server's address. Either an IP address (e.g. 17.205.15.205) or a name (e.g. ldap.bigfoot.com). If you are inside a corporate firewall, it might not allow LDAP traffic to pass through. Check with your administrator if you have trouble accessing LDAP servers on the open internet.

User Name
: If your LDAP server requires you to login, enter a username. This version of the LDAP adaptor supports simple authentication only.

Password
: If your LDAP server requires a password, enter it here.

Search Base
: Entries on an LDAP server (roughly equivalent to rows in a database) are organized in trees. The search base indicates what branch of the tree to begin searching in. Common designs are to have the top level of the tree have country or organization branches, so good search bases have the form "c=US" or "o=Apple Computer". An approximate database equivalent is to say the search base is ANDed with any qualifiers you pass in.

Schema Base
: This value is used only during reverse engineering the LDAP server. The Adaptor looks at this location for the subschema entry. Note that during reverse engineering, the scope is temporarily overridden to Base because the subschema does not have any children.

Value Separator
: The character used to separate multiple values for a property.

Port
: LDAP servers are almost always on port 389, but secure servers might use a different port.

Timeout
: Maximum time to wait during LDAP operations before raising an exception. Timeouts come from the client library and are raised as EOGeneralAdaptorExceptions.

Scope
: LDAP entries belong to an object class. The Adaptor maps each class to an entity. Like classes in object-oriented programming, LDAP classes have an inheritance hierarchy. Depending on the scope you set, results include results from the that object class only (Base), its immediate children (One Level), or the entire subtree (Subtree). The more you include, the larger the result set is for a given search. Object class hierarchies and entry hierarchies work together to give order to the LDAP server's data, and the Adaptor provides control over how ordered or chaotic the data appears.

If Reverse Engineering Fails

The LDAP specification does not require LDAP servers to provide a means of learning the server's layout; providing a schema to clients is optional. For this reason, the LDAP adaptor might not be able to reverse-engineer your LDAP server to generate an EOModel automatically. If this is the case, you might need to create your model by hand.

Generally speaking, Netscape Directory Servers automatically create and provide schemas to clients. So, if you have a Netscape Directory server, you should be fine.

However, if the adaptor can't find subschema information, it creates a default model with two entities: Person and NoSchemaDataAvailable. If the server is `ldap.bigfoot.com` (a well known server) and the scope is set to Subtree, this model works for finding names. The NoSchemaDataAvailable entity is designed to be an error message and should be deleted from the model. (EOAdaptors aren't allowed to return `null`/`nil` when reverse engineering a database, so the best alternative is to return a minimal model.)

Adding Entries to the Server

The LDAP adaptor provides limited support for adding entries to the server. If you plan to use the adaptor to insert entries, keep the following points in mind:

- The LDAP adaptor doesn't provide automatic primary key generation. so set the primary key (the distinguished name, or "dn") as a class property and set an appropriate value yourself before saving a new entry.
- Consider adding an objectclass attribute to the entity so your can add the entry to multiple classes at once.
- You must enforce object rules in custom code. The adaptor detects object class violations once you attempt to save changes, but the violations cause the adaptor to raise an exception.

Performing Authentication

One of the common applications for LDAP is to verify a user's password on the Web. The authentication is generally done by the LDAP server, not by retrieving the user's password from an LDAP entry. So, in essence, all that is needed is a mutable connection dictionary from a model and a request to validate the connection. The LDAP Adaptor provides a simple way to do this without a model:

```
java.lang.Throwable athenticateUser()
```

or in Objective–C,

```
+ (NSException *)authenticateUserString:(NSString *)userString
    password:(NSString *)password
    withServer:(NSString *)server
    matchOnAttribute:(NSString *)searchAtt
    searchBase:(NSString *)base
    searchScope:(NSString *)scope;
```

The method is pretty easy to use and the header file contains specifics on its use. Here is a sample code fragment from a WebObjects application that has one page, "Main" with three instance variables: `userName1`, `password1`, and `isLoggedIn`. This method is tied to the web page's Submit button.

```
- authenticateUser {
    NSException *exception;
    exception = [LDAPAdaptor authenticateUserString:userName1
            password:password1
            withServer:@”bigbird.apple.com”
            matchOnAttribute:@”cn”
            searchBase:@”o=apple computer” searchScope:@”Subtree”];ì
    if (exception) {
        NSLog(@”Auth failed.”);
        isLoggedIn = NO;
    } else {
        NSLog(@”Auth successful.”);
        isLoggedIn = YES;
    }
    return self;
}
```

Or, in Java (`isLoggedIn` is a string instead of a BOOL in this example):

```
public WOComponent authenticateUser() {
    java.lang.Throwable ex = null;
    ex = LDAPAdaptor.authenticateUser (
        userName,
        pswField,
        “bigbird.apple.com”,
        “cn”,
        “o=Apple Computer”,
        “Subtree”);
    if (ex == null) {
          isLoggedIn = “You are now logged in.”;
    } else {
        isLoggedIn = ex.getMessage();
    }
    return this;
}
```

Miscellaneous API Enhancements

This section summarizes other miscellaneous methods added in EOF 4.5 not covered in the other sections.

Table 5-24 EOAdaptorContext (EOAccess/EOAdaptorContext.h)

New or Changed API	Description
`hasOpenTransaction`	Returns `true`/`YES` if a transaction is open (begun but not yet committed or rolled back). For more information on this addition, see the section "Deprecated API" (page 111).

Table 5-25 EODatabaseContext Delegate (EOAccess/EODatabaseContext.h)

New or Changed API	Description
`databaseContextWillFire ObjectFaultForGlobalID` (Java) `databaseContext: willFireObjectFaultForGlobalID: withFetchSpecification: editingContext:` (Objective–C)	Invoked just before the Framework-generated fetch specification (provided as an argument) is used to clear the fault for the specified globalID. Note that it is very dangerous to modify the fetch specification.
`databaseContextWillFire ArrayFaultForGlobalID` (Java) `databaseContext: willFireArrayFaultForGlobalID: relationship: withFetchSpecification: editingContext:` (Objective–C)	Invoked just before the Framework-generated fetch specification (provided as an argument) is used to clear the fault for the specified globalID and relationship. Note that it is very dangerous to modify the fetch specification.

Table 5-26 EOEditingContext Additions (Objective–C only; EOAccess/EOUtilities.h)

New or Changed API	Description
`createAndInsertInstanceOfEntityNamed:`	Creates a new enterprise object of the specified entity, inserts it into the receiving editing context, and returns the new object.

Table 5-27 EOUtilities (Java)

New or Changed API	Description
`createAndInsertInstance`	Creates a new enterprise object of the specified entity, inserts it into the specified editing context, and returns the new object.

Table 5-28 EOClassDescription (EOControl/EOClassDescription.h)

New or Changed API	Description
`fetchSpecificationNamed` (Java) `fetchSpecificationNamed:`(Objective–C)	The receiver returns the fetch specification it associates with the specified name. EOClassDescription's implementation returns `null`/`nil`; subclasses can override it to return a fetch specification.

Table 5-29 EOFetchSpecification (EOControl/EOFetchSpecification.h)

New or Changed API	Description
`fetchSpecificationNamed` (Java) `fetchSpecificationNamed:entityNamed:` (Objective–C)	A static/class method that returns the fetch specification that the specified entity associates with the specified fetch specification name.

Table 5-30 EOQualifier (EOControl/EOQualifier.h)

New or Changed API	Description
`evaluateWithObject` (Java) `evaluateWithObject:` (Objective–C)	Implemented by subclasses to return `true`/`YES` if the provided object matches the criteria specified in the receiver, `false`/`NO` otherwise. The argument should be an enterprise object, a snapshot dictionary, or something that implements key-value coding.
`allQualifierKeys`	Returns an NSSet of strings, which are the left-hand sides of all the qualifiers in the receiver. For example, if you have a qualifier salary > 10000 AND manager.lastName = 'smith' `allQualifierKeys` returns an array containing the strings "salary" and "manager.lastName". Subclasses should not override this method, instead they should override `addQualifierKeysToSet`.
`addQualifierKeysToSet` (Java) `addQualifierKeysToSet:` (Objective–C)	Adds the receiver's qualifier keys to the specified NSMutableSet. The subclasses in the EOControl framework do this by traversing the tree of qualifiers. Node qualifiers (such as EOAndQualifier) recursively invoke this method until they reach a leaf qualifier (such as EOKeyValueQualifier) which adds its key to the set. Subclasses of EOQualifier must implement this method.

Table 5-31 EOTemporaryGlobalID

New or Changed API	Description
`assignGloballyUniqueBytes` (Java only)	This method, which was not wrapped in 4.0, is the equivalent of the Objective–C method, `assignGloballyUniqueBytes:`, which assigns a network-wide unique ID

Table 5-32 EOAssociation (EOInterface/EOAssociation.h)

New or Changed API	Description
`isExplicitlyDisabled` (Java Client only) `setExplicitlyDisabled` (Java Client only)	Returns or sets whether or not the association is explicitly disabled. These methods are used by the new user interface generation layer, which is described in "Direct To Java Client" (page 138). An association is "explicitly disabled" when the display object shouldn't be editable, such as in the case where the display object simply displays the results of a search.

Table 5-33 EODisplayGroup (EOInterface/EODisplayGroup.h)

New or Changed API	Description
`globalDefaultStringMatchOperator`	A static/class method that returns the default operator used for matching strings, one of `caseInsensitiveLike` or `like`
`setGlobalDefaultStringMatchOperator` (Java) `setGlobal DefaultStringMatchOperator:` (Objective–C)	A static/class method that sets the default operator for instances to use for string matching.
`globalDefaultStringMatchFormat`	A static/class method that returns the default format used for matching strings ("`%@*`", for example).

Table 5-33 EODisplayGroup (EOInterface/EODisplayGroup.h) (continued)

New or Changed API	Description
`setGlobalDefaultStringMatchFormat` (Java) `setGlobalDefaultStringMatchFormat:` (Objective–C)	A static/class method that sets the default format for instances to use for string matching.
`globalDefaultForValidatesChangesImmediately`	A static/class method that returns `true`/`YES` if instances validate immediately by default.
`setGlobalDefaultForValidatesChangesImmediately` (Java) `setGlobalDefaultForValidatesChangesImmediately:` (Objective–C)	A static/class method that sets the default validation behavior for instances.

Deprecated API

Nested transactions are no longer supported. EOF never actually used nested transactions. Furthermore, the concrete adaptors were not guaranteed to support them, especially since the SQL/92 standard doesn't allow nested transactions. New features in EOF 4.5 make nested transactions impossible to support.

Consequently, the following methods are deprecated.

Table 5-34 EOAdaptorContext (EOAccess/EODeprecated.h)

Deprecated API	New API or Workaround
`canNestTransactions:`	None. No adaptor can nest transactions.
`transactionNestingLevel`	`hasOpenTransaction` Returns `true`/`YES` if a transaction is open (begun but not yet committed or rolled back).

For backwards compatibility, the Sybase Adaptor still allows you to attempt to begin a nested transaction, but the implementation ignores the nesting.

What's New in Java Client

This chapter describes changes made to Java Client between WebObjects release 4.0 and release 4.5. Java Client has been extended considerably, including the following:

- The foundation layer (com.apple.client.foundation) contains a new number formatter based on NSNumberFormatter and adds an NSUndoManager class, which is analogous to the server side class.
- The control layer (com.apple.client.eocontrol) is more complete.
- The distribution layer (com.apple.client.eodistribution on the client side and com.apple.yellow.eodistribution on the server side) provides support for encrypted client/server communication and for managing user defaults.
- The interface layer (com.apple.client.eointerface) adds support for table cell editing and for displaying images and QuickTime media.

Additionally, Java Client now has a new user interface generation layer, Direct to Java Client, which is comparable to WebObjects' Direct to Web.

The following sections describe how Java Client has been synchronized with the rest of EOF, the changes to the procedures for running Java Client applications, and the new Direct to Java Client technology.

Note: Java Client applications require some conversion to run on WebObjects 4.5; see the *WebObjects 4.5 Post-Installation Instructions* for more information on converting your existing applications.

Foundation Layer Changes

This section lists some of the new features in the Java Client Foundation framework (com.apple.client.foundation).

Number Formatter

The Java Client number formatter has been rewritten; it is now based on the NSNumberFormatter in the Foundation framework. It preserves the previous AWT-based API, so it isn't necessary to convert your code.

The new number formatter supports a subset of the Objective–C number formatter. It doesn't support attributed strings, but it does support customized text strings for zero, null, and NaN (Not a Number).

New Foundation Layer Classes and Interfaces

The following table lists the classes and interfaces that have been added in this release.

Class or Interface	Description
NSDisposable	An interface that defines a method (`dispose`) for performing any necessary housecleaning in preparation for garbage collection.
NSInlineObservable	An interface that observable objects implement. NSNotificationCenters and EOObserverCenters use the interface's two methods (`observerData` and `setObserverData`) to avoid creating uncollectable object references. Note that this interface is only needed in the absence of weak references; it might be removed in the future when support for weak references is available.
NSUndoManager	Analogous to the server side class.

Control Layer Changes

The Java Client control layer API is more complete in this release. Many features that were missing in 4.0 have been ported, and many of the server-side features added in EOF 4.5 are available on the client side as well. This section lists the new classes, interfaces, and methods added to Java Client's control layer (com.apple.client.eocontrol) as well as classes and methods that have been removed. The purpose of most of the API changes is to synchronize Java Client's feature set and API with EOF's. However, some of the new API is exclusive to Java Client.

New Control Layer Classes and Interfaces

The following table lists the classes and interfaces that have been added in this release.

Class or Interface	Description
EOFoundationExtras	Exclusive to Java Client. A utility class containing convenience methods for working with classes in the com.apple.client.foundation package.
EOKeyValueCoding.KeyBinding	Analogous to the server side class. Corresponds to a new feature in EOF 4.5. See "Key Value Coding Changes" (page 99).
EOKeyValueCoding. KeyBindingCreation (interface)	Analogous to the server side interface. Corresponds to a new feature in EOF 4.5. See "Key Value Coding Changes" (page 99).
EOKeyValueCoding. UnknownKeyException	The kind of exception raised by key value coding methods when they encounter an unknown key.
EOQualifier. QualifierVariableSubstitutionException	The kind of exception raised when an EOQualifierVariable object requires bindings for all its variables and one or more variable is missing from the bindings.
EOQualifierVariable	Analogous to the server side class.

New API

The following tables summarize the methods and constants that have been added to the client side control layer in this release. Most methods are analogous to methods in the server side control layer and have been added to synchronize the two layers.

Table 6-1 EOClassDescription

API	Description
`invalidateClassDescriptionCache`	Analogous to the server side method.
`defaultFormatterForKey`	Analogous to the server side method.
`defaultFormatterForKeyPath`	Analogous to the server side method.
`displayNameForKey`	Analogous to the server side method.
`fetchSpecificationNamed`	Analogous to the server side method.
`userPresentableDescriptionForObject`	Analogous to the server side method.

Table 6-2 EOCustomObject

API	Description
`createKeyValueBindingForKey`	Analogous to the server side method. Conformance to EOKeyValueCoding.KeyBindingCreation, a new feature in EOF 4.5. See "Key Value Coding Changes" (page 99).
`keyValueBindingForKey`	Analogous to the server side method. Conformance to EOKeyValueCoding.KeyBindingCreation, a new feature in EOF 4.5. See "Key Value Coding Changes" (page 99).
`observerData` `setObserverData`	Exclusive to Java Client. Conformance to NSInlineObservable.

Table 6-3 EODelayedObserver

API	Description
`observerData` `setObserverData`	Exclusive to Java Client. Conformance to NSInlineObservable.

Table 6-4 EOEditingContext

API	Description
`dispose`	Exclusive to Java Client. Conformance to NSDisposable.
`invalidatesObjectsWhenFinalized` `setInvalidatesObjectsWhenFinalized`	Analogous to the server side method.
`lock` `unlock`	Analogous to the server side method. Note that multithreaded clients aren't yet supported. All the client-side locks in Java Client application's are no-ops.
`observerData` `setObserverData`	Exclusive to Java Client. Conformance to NSInlineObservable.
`redo`	Analogous to the server side method.
`refault`	Analogous to the server side method.
`refetch`	Analogous to the server side method.
`reset`	Analogous to the server side method.
`revert`	Analogous to the server side method.
`undoManager` `setUndoManager`	Analogous to the server side method.
`undo`	Analogous to the server side method.

Table 6-5 EOEditingContext.Delegate

API	Description
`editingContextDidMergeChanges`	Analogous to the server side method.
`editingContextShouldInvalidateObject`	Analogous to the server side method.
`editingContext ShouldMergeChangesForObject`	Analogous to the server side method.
`editingContext ShouldUndoUserActionsAfterFailure`	Analogous to the server side method.

Table 6-6 EOEditingContext.MessageHandler

API	Description
`editingContextPresentErrorMessage`	Analogous to the server side method.

Table 6-7 EOEnterpriseObject

API	Description
`changesFromSnapshot`	Analogous to the server side method.
`reapplyChangesFromDictionary`	Analogous to the server side method.
`userPresentableDescription`	Analogous to the server side method.

Table 6-8 EOFaultHandler

API	Description
`eoShallowDescription`	Returns a string that describes the object that receiver's fault represents. Used to prevent faulting an object upon receipt of an `eoShallowDescription` message.
`descriptionForObject`	Analogous to the server side method.

Table 6-9 EOFaulting

API	Description
`faultHandler`	If the receiver is a fault, returns its EOFaultHandler; otherwise returns `null`.

Table 6-10 EOFetchSpecification

API	Description
`fetchSpecificationNamed`	Analogous to the server side method.
`fetchSpecification WithQualifierBindings`	Analogous to the server side method.

Table 6-11 EOKeyValueCoding

API	Description
`TargetObjectUserInfoKey` `UnknownUserInfoKey`	String constants defining the keys in the userInfo dictionary of an EOKeyValueCoding.UnknownKeyException. An UnknownKeyException is raised by key value coding methods when they are invoked with a key that does not correspond to a method or instance variable in the receiving object. The userInfo of the exception contains the target object (`TargetObjectUserInfoKey`) and the key (`UnknownUserInfoKey`).
`SetKeyBindingMask`	Analogous to the server side constant.
`StoredKeyBindingMask`	Analogous to the server side constant.

Table 6-12 EOKeyValueCoding.Support (Exclusive to Java Client)

API	Description
`createKeyValueBindingForKey`	A static method that provides a default implementation of the corresponding EOKeyValueCoding method.
`keyValueBindingForKey`	A static method that provides a default implementation of the corresponding EOKeyValueCoding method.

Table 6-13 EOKeyValueCodingAdditions.Support (Exclusive to Java Client)

API	Description
`takeStoredValuesFromDictionary`	A static method that provides a default implementation of the corresponding EOKeyValueCodingAdditions method.
`valueForKeyPath`	A static method that provides a default implementation of the corresponding EOKeyValueCodingAdditions method.

Table 6-14 EOObjectStore

API	Description
`editingContext DidForgetObjectWithGlobalID`	Analogous to the server side method. Corresponds to a new feature in EOF 4.5. See "Snapshot Reference Counting" (page 83). However, note that Java Client doesn't implement snapshot reference counting yet.
`faultForRawRow`	Analogous to the server side method.

Table 6-15 EOQualifier

API	Description
`qualifierToMatchAllValues`	Analogous to the server side method.
`qualifierToMatchAnyValue`	Analogous to the server side method.
`addQualifierKeysToSet`	Analogous to the server side method, which is new in EOF 4.5. See "Miscellaneous API Enhancements" (page 107).
`allQualifierKeys`	Analogous to the server side method, which is new in EOF 4.5. See "Miscellaneous API Enhancements" (page 107).
`bindingKeys`	Analogous to the server side method.

Table 6-15 EOQualifier (continued)

API	Description
`keyPathForBindingKey`	Analogous to the server side method.
`qualifierWithBindings`	Analogous to the server side method.
`validateKeysWithRootClassDescription`	Analogous to the server side method.

Table 6-16 EOValidation

API	Description
`validateTakeValueForKeyPath`	Analogous to the server side method.

Deleted API

The two interfaces EOKeyValueCoding.KeyValueGetter and EOKeyValueCoding.KeyValueSetter have been removed. Additionally, the following methods, organized by class, have been deleted and replaced where appropriate, for compatibility with the Java wrappers (com.apple.yellow packages).

Table 6-17 EOClassDescription

Deleted API	New API or Workaround
`registerForName`	`registerClassDescription`
`registerForClass`	`registerClassDescription`

Table 6-18 EOEditingContext

Deleted API	New API or Workaround
`globalIDsForObjects`	`globalIDForObject`
`objectsForGlobalIDs`	`objectForGlobalID`
`registeredGlobalIDs`	`registeredObjects` and `globalIDForObject`

Table 6-19 EOEditingContext.MessageHandler

Deleted API	New API or Workaround
`editingContextPresentException`	`editingContextPresentErrorMessage`

Server-Side Features Not in Java Client

Features added to server-side in the 4.5 release that are *not* yet available in Java Client are:

- Shared editing contexts
- Event logging
- Snapshot timestamps
- Snapshot reference counting
- Recursive reader/writer locks

Distribution Layer Changes

This section describes the changes made to the client and server sides of the distribution layer for Java Client applications. They are:

- EODistributionContexts are associated with WOSessions.

An EODistributionContext is now associated with a WOSession, and by default, a distribution context's editing context is the session's default editing context.

- Client server communication supports encryption.

  Additional delegate methods for EODistributionContext and an EODistributionChannel delegate provide hooks in which you can encrypt and decrypt data being sent between client and server.

- The EODistributedObjectStore method `invokeRemoteMethodWithKeyPath` now invokes the method on the server side EODistributionContext if the provided key path is `null`.

- User defaults support

  EODistributionContext now sends out notifications to load and save user defaults, which can be used to manage defaults on a per-user basis.

New Distribution Layer Classes and Interfaces

The following table lists the classes and interfaces that have been added in this release.

Class or Interface	Description
EODistributionChannel.Delegate	An interface defining methods that allow you to encrypt and decrypt data being sent between client and server

Related API Changes

The following tables summarize the updated and new API in the distribution layer.

Table 6-20 EODistributionContext (Server side; EOJavaClient/EODistributionContext.h)

New or Changed API	Description

`public EODistributionContext( WOSession session, EOEditingContext context)`
`public EODistributionContext( WOSession session)`
(Java)
	Creates a new EODistributionContext for use within the specified session and with the specified editing context, if provided. If an editing context isn't provided, the new distribution context is associated with the session's default editing context.

`initWithSession:editingContext:` (Objective–C)	Initializes a new EODistributionContext for use in the specified session and with the specified editing context.
`initWithSession:` (Objective–C)	Initializes a new EODistributionContext for use in the specified session and with that session's default editing context.
`editingContext`	Returns the EOEditingContext with which the distribution context is associated.

Table 6-20 EODistributionContext (Server side; EOJavaClient/EODistributionContext.h) (continued)

New or Changed API	Description
`session`	Returns the WOSession with which the distribution context is associated.
`LoadUserDefaultsNotification` (Java) `EOLoadUserDefaultsNotification` (Objective–C)	A string constant defining the name of a notification that's posted whenever a distribution context receives a request for user default values from a client application. Receivers can load default values (from a database, for example) and add them to the mutable dictionary provided in the notification's userInfo.
`SaveUserDefaultsNotification` (Java) `EOSaveUserDefaultsNotification` (Objective–C)	A string constant defining the name of a notification that's posted whenever the distribution context receives user default values from a client application. Receivers can use this notification to store the default values (in a database, for example). The default values are in the notification's userInfo dictionary.

Table 6-21 EODistributionContext.Delegate (Server side; EOJavaClient/EODistributionContext.h)

New or Changed API	Description
`distributionContextDidReceiveData` (Java) `distributionContext:didReceiveData:` (Objective–C)	Invoked after a distribution context has received data. You can use this method and its counterpart, `distributionContextWillSendData`, to implement encryption in client server communication, encrypting in `distributionContextWillSendData` and decrypting in `distributionContextDidReceiveData`.
`distributionContextWillSendData` (Java) `distributionContext:willSendData:` (Objective–C)	Invoked before a distribution context sends data to the client.

Table 6-22 WOJavaClientApplet (Server side; EOJavaClient/WOJavaClientApplet.h)

New or Changed API	Description
`EOAllParameterNamesKey` (Objective–C)	A string constant defining a dictionary key used internally to collect the names of all HTML parameters passed to the client (the names of all bindings of the WOJavaClientApplet), including any additional bindings that you add to the applet.
`EOSessionIDKey` (Objective–C)	A string constant defining a dictionary key used internally to identify the session with which the server side EODistributionContext is associated.
`EOComponentURLKey` (Objective–C)	A string constant defining a dictionary key used internally to identify the WOJavaClientApplet component on the server side which corresponds to the EOApplet on the client side.

Table 6-23 EODistributionChannel (Client side)

New or Changed API	Description
`observerData`, `setObserverData`	Conformance to NSInlineObservable.
`delegate`, `setDelegate`	Accessing the distribution channel's delegate.

Table 6-24 EODistributedObjectStore (Client side)

New or Changed API	Description
`observerData`, `setObserverData`	Conformance to NSInlineObservable.
`invokeRemoteMethodWithKeyPath` (Changed behavior)	If the specified key path is `null`, this method now invokes the method on the server side EODistributionContext rather than on the EODistributionContext's invocation target as it did in earlier releases.

Deleted API

The following methods have been deleted and replaced where appropriate, for backwards compatibility with the Java wrappers (com.apple.yellow packages).

Table 6-25 EODistributionContext (Server side; EODistributionContext.h)

Deleted API	New API or Workaround
`public EODistributionContext( EOEditingContext context)` (Java)	`public EODistributionContext( WOSession session, EOEditingContext context)`
`initWithEditingContext:` (Objective–C)	`initWithSession:editingContext:`

Interface Layer Changes

This section describes changes in the interface layer of Java Client applications, including the following new features:

- Support for table cell editing
- Support for displaying images and QuickTime media
- URL Aspect for Associations
- Changes to string matching behavior in EODisplayGroup

The changes to EODisplayGroup have also been made to WODisplayGroup. For more information on the changes, see the section "Other WODisplayGroup Changes" (page 38) in the chapter WebObjects Framework API Changes.

Support for Table Cell Editing

Swing implements JTable editing using javax.swing.table.TableCellEditor, a single method interface returning a java.awt.Component to act as editor. The new Java Client class EOColumnEditor implements this interface to mediate between the Component it returns and the EOTableColumnAssociation bound to the edited column. Abstract hooks for component instantiation and protected methods for editing event communication allow concrete subclasses such as EOTextColumnEditor to focus purely upon their Component's specifics.

EOTableColumnAssociation acquires the editors for associated TableColumns from its TableCellCustomizer, a new static object serving as the source of both EOColumnEditors and TableCellRenderers. EOTableColumnAssociation's implementation of `establishConnection` now sets the editor and renderer of its TableColumn to the objects returned by this object. Consumers may customize columns by installing their own TableCellCustomizer.

QuickTime Association

If you use the QuickTime view and association classes, you should note the following:

- While you don't have to have the QuickTime software or the QuickTime for Java packages installed during development, clients must have both installed to view the QuickTime media. If one or the other aren't installed, the QuickTime view is simply blank.
- Apple only provides QuickTime for Java on MacOS and Windows, *not* on MacOS X Server.
- EOQuickTimeAssociation doesn't support the `ValueAspect`, only `URLAspect`.
- HTTP urls are only supported in QuickTime v4.0.
- The QuickTime association and view aren't supported by the Java Client palette in Interface Builder.

URLAspect for Associations

EOTextAssociation, EOImageAssociation, and EOQuickTimeAssociation support a new aspect, `URLAspect`. As opposed to the `ValueAspect`, which is the raw data for the text, image, or movie, the `URLAspect` is a url that references data on disk or over the Web. You can bind the `URLAspect` in Interface Builder. The corresponding values are read only.

Package Reorganization and Changes

The following classes have been moved from the eointerface package to the eoapplication package, a new package to support Direct to Java Client:

- EOApplet
- EOApplication
- EOArchive
- EOInterfaceController

Additionally, the classes EOApplication and EOInterfaceController have new superclasses, and their APIs have been modified considerably. EOArchive has also changed, resulting in the requirement that you open and explicitly save every interface file in your projects.

New Interface Layer Classes and Interfaces

The following table lists the classes and interfaces that have been added to the client side interface layer in this release.

Class or Interface	Description
EOColumnEditor	An abstract class that implements generalized cell editing management for javax.swing.JTables. See "Support for Table Cell Editing" (page 128).
EOImageAssociation	A class whose instances associate the contents of their display groups with EOImageViews.
EOImageView	A class whose instances display images (java.awt.Image objects) in Java Client applications.
EOQuickTimeAssociation	A class whose instances associate the contents of their display groups with EOQuickTimeViews.
EOQuickTimeView	A class whose instances display QuickTime media in Java Client applications. See "QuickTime Association" (page 128).
EOTableColumnAssociation. TableColumnCustomizer (interface)	An interface that defines the API an object uses to specify custom editors and renderers for an EOTableColumnAssociation. See "Support for Table Cell Editing" (page 128).
EOTextColumnEditor	EOTextColumnEditor is a concrete subclass of EOColumnEditor whose instances mediate between EOTextColumnAssociations and EOTextFields. See "Support for Table Cell Editing" (page 128).

For more information on these classes and interface, see the corresponding class and interface specifications.

Added Methods

The following tables summarize the methods and constants that have been added to the Java Client interface layer in this release. The majority of the additions are in EODisplayGroup. Most of the additions are analogous to API in the server side WODisplayGroup and have been added to synchronize the classes.

Table 6-26 EOAssociation

New or Changed API	Description
`URLAspect` (Constant)	A string constant that defines the name of a new aspect. See "URLAspect for Associations" (page 129).
`dispose`	Conformance to NSDisposable.
`isEnabled`	Returns `false` if the receiver has explicitly disabled its display object or if the receiver's `EnabledAspect` (if bound) resolves to `false`; `true` otherwise.
`isEnabledAtIndex`	Returns `false` if the receiver has explicitly disabled its display object or if the receiver's `EnabledAspect` (if bound) resolves to `false` for the specified index; `true` otherwise.
`isExplicitlyDisabled`	Returns `true` if the receiver has explicitly disabled its display object, `false` otherwise.
`setExplicitlyDisabled`	Sets whether or not the receiver is explicitly disabled. This method and its counterpart `isExplicitlyDisabled` are used by Direct to Java Client. An association is "explicitly disabled" when the display object shouldn't be editable, such as in the case where the display object simply displays the results of a search.

Table 6-27 EOControlActionAdapter

New or Changed API	Description
`dispose`	Conformance to NSDisposable.

Table 6-28 EODisplayGroup

New or Changed API	Description
`globalDefaultForValidatesChangesImmediately` `setGlobalDefaultForValidatesChangesImmediately`	Static methods that return or set the default validation behavior for new display group instances: `true` if they immediately handle validation errors, or `false` if they leave errors for the EOEditingContext to handle when saving changes.
`globalDefaultStringMatchFormat` `setGlobalDefaultStringMatchFormat`	Static methods that return or set the default string match format string used by display group instances.
`globalDefaultStringMatchOperator` `setGlobalDefaultStringMatchOperator`	Static methods that return or set the default string match operator used by display group instances.
`awakeFromNib`	Invoked when the receiver is unarchived from a nib file to prepare it for use in an application.
`defaultStringMatchFormat` `setDefaultStringMatchFormat`	Returns or sets the default string match format string used by the receiver.
`defaultStringMatchOperator` `setDefaultStringMatchOperator`	Returns or sets the default string match operator used by the receiver.
`delete`	Analogous to WODisplayGroup's method.
`dispose`	Conformance to NSDisposable.
`editingContextPresentErrorMessage`	Invoked as part of the EOEditingContext.MessageHandlers, to present an attention panel a message to display.
`enterQueryMode`	Puts the receiver in query mode.

Table 6-28 EODisplayGroup (continued)

New or Changed API	Description

`equalToQueryValues`
 `setEqualToQueryValues`
	Returns or sets the receiver's dictionary of equalTo query values. Similar to the WODisplayGroup `queryMatch` method.

`fetch`	Analogous to the WODisplayGroup method.
`finishInitialization`	Invoked from the EODisplayGroup constructor and from `awakeFromNib` to finish initializing a newly created display group.

`greaterThanQueryValues`
 `setGreaterThanQueryValues`
	Returns or sets the receiver's dictionary of greaterThan query values. Similar to the WODisplayGroup `queryMin` method.

`inQueryMode`
 `setInQueryMode`
	Analogous to WODisplayGroup's methods.

`insert`	Analogous to the WODisplayGroup method.
`insertObjectAtIndex`	Analogous to the WODisplayGroup method `insertNewObjectAtIndex`.

`insertedObjectDefaultValues`
 `setInsertedObjectDefaultValues`
	Analogous to WODisplayGroup's methods.

`lessThanQueryValues`
 `setLessThanQueryValues`
	Returns or sets the receiver's dictionary of lessThan query values. Similar to the WODisplayGroup `queryMax` method.

`observerData`
 `setObserverData`
	Conformance to NSInlineObservable.

`qualifierFromQueryValues`	Analogous to the WODisplayGroup method.
`qualifyDataSource`	Analogous to the WODisplayGroup method.
`qualifyDisplayGroup`	Analogous to the WODisplayGroup method.

`queryBindingValues`
 `setQueryBindingValues`
	Returns or sets a dictionary containing the actual values that the user wants to query upon.

Table 6-28 EODisplayGroup (continued)

New or Changed API	Description
queryOperatorValues setQueryOperatorValues	Returns or sets a dictionary of operators to use on items in the query dictionaries (equalToQueryValues, greaterThanQueryValues, and lessThanQueryValues).
selectNext	Analogous to the WODisplayGroup method.
selectObjectsIdenticalToSelectFirstOnNoMatch	Analogous to the WODisplayGroup method.
selectPrevious	Analogous to the WODisplayGroup method.
setValueForObjectAtIndex (Changed arguments)	The position of the integer index argument has been swapped with that of the String value argument.
setValueForObject (Changed arguments)	The position of the last two arguments has been swapped and the EOKeyValueCodingAdditions argument has been retyped as an Object.
sortOrderings	Analogous to the WODisplayGroup method.
undoManager	Returns the receiver's NSUndoManager.
valueForObjectAtIndex (Changed arguments)	The position of the arguments has been swapped.
valueForObjectKey	Returns the value in the specified object for the property identified by the specified key.
willChange	Notifies observers that the receiver will change.

Table 6-29 EOFormCell

New or Changed API	Description
dispose	Conformance to NSDisposable.

Table 6-30 EOMasterDetailAssociation

New or Changed API	Description
`dispose`	Conformance to NSDisposable.

Table 6-31 EOTable

New or Changed API	Description
`dispose`	Conformance to NSDisposable.

Table 6-32 EOTableAssociation

New or Changed API	Description
`removeColumnAssociation`	Removes the specified column association from the receiver's set of EOTableColumnAssociations.

Table 6-33 EOTableColumnAssociation

New or Changed API	Description
`tableColumnCustomizer` `setTableColumnCustomizer`	Returns and sets the association's table column customizer.
`dispose`	Conformance to NSDisposable.
`table` `setTable` (Changed)	Returns and sets the association's EOTable object. Note that the pre-4.5 `setTable` method had a java.awt.Component as an argument instead of an EOTable (from which you can get Swing table).

Table 6-34 EOView

New or Changed API	Description
`dispose`	Conformance to NSDisposable.

Deleted API

The following methods, organized by class, have been deleted and replaced, for backwards compatibility with the Java wrappers (com.apple.yellow packages):

Table 6-35 EOActionAssociation

Deleted API	New API or Workaround
`enabled`	`isEnabled` (inherited from EOAssociation)

Table 6-36 EODisplayGroup

Deleted API	New API or Workaround
`EODisplayGroup(EODataSource dataSource)`	Invoke the default constructor followed by `setDataSource` with the data source as an argument.
`editingContextPresentException(`	`editingContextPresentErrorMessage`
`insertNewObjectAtIndex(int index);`	`insertObjectAtIndex(int index)`
`selectObjectsIdenticalTo`	`selectObjectsIdenticalToSelectFirstOnNoMatch`
`sortOrdering`	`sortOrderings` Note the addition of the "s" in "Orderings"
`valueForObject( String value, EOKeyValueCodingAdditions object)`	`valueForObjectKey( EOKeyValueCodingAdditions object, String value)` Note that in addition to the method name change, the position of the arguments was swapped.

Table 6-37 EOViewLayout

Deleted API	New API or Workaround
`Fixed` (Constant)	For use in `setAutosizingMask`, simply use 0 to mean fixed size.

Running Java Client Applications

There are two changes to running Java Client Applications: the syntax for starting applications has changed and the classpath requirements have changed for all platforms except Mac OS X Server.

In WebObjects 4.5, there are three ways to run Java Client applications:

1. As a java application with the command:

   ```
   java [-debug] -classpath <classpath>
       com.apple.client.eoapplication.EOApplication
       -applicationURL <url> [-page <page>]
   ```

   Note that you might have to vary the parameters if you use special distribution channels.

2. As an applet in Applet Viewer with the commands:

   ```
   export CLASSPATH=<classpath>
   appletviewer [-debug] <url>
   ```

3. As an applet in a browser

For Non-Mac OS X Server Users

If you run Java Client applications on non-Mac OS X Server platforms, you might need to change the classpath you ordinarily use. If you previously put the `awt.jar` file (`$(NEXT_ROOT)/Library/Frameworks/JavaVM.framework/Classes/awt.jar`) in your classpath, you should remove it. Except on Mac OS X Server, you can't use the `awt.jar`.

Direct To Java Client

Direct to Java Client is an addition to Enterprise Objects Framework and Java Client. It dynamically generates complete Java Client applications (or parts of them) from information in a model. User interface configuration information is stored as a set of rules. You can use the user interface specified by the default set of rules, or you can customize the user interface by changing the rules.

The version of Direct to Java Client shipping with WebObjects 4.5 is a technology preview. It is very stable and robust, but its programming interfaces and generated user interfaces are not guaranteed to be the same in future releases.

Two new packages have been added to the Java Client technology to support Direct to Java Client, eoapplication and eogeneration. The eoapplication package provides application level logic such as document management and classes for creating advanced user interfaces. The eogeneration package is for defining applications that are completely dynamic.

The eoapplication and eogeneration packages consists mainly of **controllers**. A new class, EOController, defines the basic controller behavior.

Note: The EOInterfaceController class, which was in the eointerface package in the previous release, has been redefined as a subclass of EOController. In greater detail, EOController is a subclass of EOEntityController, which is subclass of EOComponentController, which is subclass of EOController.

There are different types of controllers:

- **User interface controllers** (for controlling tabbed panes and windows, for example)
- **Entity level controllers** (for controlling user interfaces that operate at an entity or object level; query and editor interfaces, for example)
- **Property level controllers** (for controlling user interface widgets such as text fields, combo boxes, action buttons that operate on properties of objects)
- **Application objects**

Controllers are organized in a hierarchy. The root controller is the shared application object, typically an EOApplication. Children of the EOApplication object, or the EOApplication's **subcontrollers**, are usually window or applet controllers which themselves have one or multiple subcontrollers.

Note: The eoapplication and eogeneration APIs are preliminary, and might change in future releases.

To learn more about Direct to Java Client, including how to create dynamic applications, see the tutorial "Getting Started with Direct to Java Client".