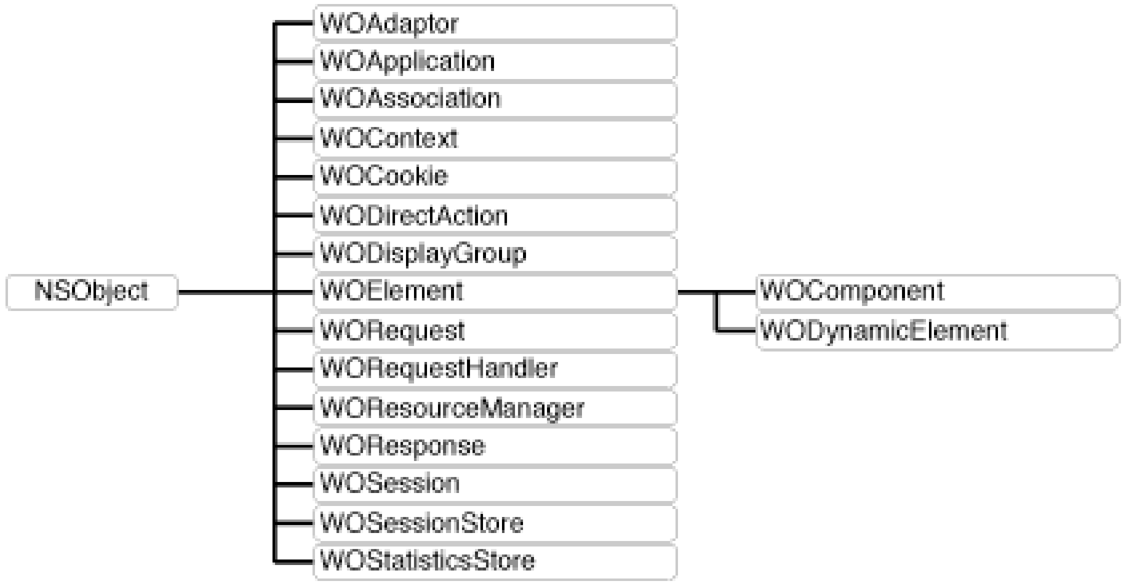
The WebObjects Framework

Framework:	/System/Library/Frameworks/WebObjects.framework
Header File Directories:	/System/Library/Frameworks/WebObjects.framework/Headers

Introduction

The WebObjects class hierarchy is rooted in the Foundation Framework's NSObject class (see Figure 1). The remainder of the WebObjects Framework consists of several related groups of classes as well as a few individual classes.

Figure 1 The WebObjects Framework class hierarchy



The more commonly-used classes within the WebObjects framework can be grouped as follows:

- **Server and Application Level Classes**. WOAdaptor defines the interface for objects mediating the exchange of data between an HTTP server and a WebObjects application. WOApplication receives requests from the adaptor and initiates and coordinates the request-handling process, after which it returns a response to the adaptor.
- **Session Level Classes**. WOSession encapsulates the state of a session; WOSession objects persiste between the cycles of the request-response loop. WOSessionStore provides the strategy or mechanism through which WOSession objects are made persistent.
- **Request Level Classes**. WORequest stores essential data about an HTTP request, such as header information, form values, HTTP version, host and page name, and session, context, and sender IDs. WOResponse stores and allows the modification of HTTP response data, such as header information, status, and HTTP version. WOContext provides access to the objects involved in the current cycle, such as the current request, response, session, and application objects.
- **Page Level Classes**. WOComponent represents and integral, reusable page (or portion of a page) for display in a web browser. WOElement declares the three request-handling methods:

takeValuesFromRequest:inContext:, invokeActionForRequest:inContext:, and appendToResponse: inContext:. WODynamicElement is an abstract class for subclasses that generate particular dynamic elements. WOAssociation knows how to find and set a value by reference to a key.

- **Database Integration Level Classes**. WODisplayGroup performs fetches, queries, creations, and deletions of records from one table in the database.

WOAdaptor

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOAdaptor.h

Class Description

WOAdaptor is an abstract class that represents objects that can receive events from a WebObjects adaptor. A WebObjects adaptor is a process that handles communication between the server and a WebObjects application. The WebObjects application (a WOApplication instance) communicates with the adaptor using messages defined in the WOAdaptor class.

The purpose of the WOAdaptor class is to perform these tasks:

- Register with the application's run loop to begin receiving events.
- Receive incoming events from the run loop and package them as WORequest objects.
- Forward the WORequest to the WOApplication by sending it the message **dispatchRequest:**.
- Receive the WOResponse object from the WOApplication and send it to the client using an RPC mechanism.

Method Types

Creation	
	– initWithName:arguments:
Obtaining attributes	
	– doesBusyRunOnce
	– dispatchesRequestsConcurrently
Event registering	
	– registerForEvents
	– unregisterForEvents
Running	
	– runOnce

Instance Methods

doesBusyRunOnce

– (BOOL)**doesBusyRunOnce**

Returns whether repeatedly invoking **runOnce** would result in busy waiting.

dispatchesRequestsConcurrently

– (BOOL)**dispatchesRequestsConcurrently**

Returns YES if the adaptor is multi-threaded, NO otherwise. If the adaptor is multi-threaded, the adaptor may dispatch requests to the application concurrently in separate threads.

See also: – **adaptorsDispatchRequestsConcurrently** (WOApplication)

initWithName:arguments:

– (id)**initWithName:**(NSString *)*aName* **arguments:**(NSDictionary *)*someArguments*

Initializes a WOAdaptor with the name *aName* and arguments *someArguments*. *aName* is the name of the WOAdaptor subclass. *someArguments* are the default options specified for this adaptor (such as port number and listen queue depth).

The WOApplication method **adaptorWithName:arguments:** invokes this message when it encounters an **WOAdaptor** option on the command line. The WOApplication retains each of its WOAdaptors.

See also: – **adaptorWithName:arguments:** (WOApplication)

registerForEvents

– (void)**registerForEvents**

Performs any actions necessary to have the WOAdaptor start receiving events.

See also: – **runLoop** in WOApplication

runOnce

– (void)**runOnce**

Invoked by the application’s main loop

See also: – **doesBusyRunOnce**

unregisterForEvents

– (void)**unregisterForEvents**

Undoes the actions performed in **registerForEvents** so that the WOAdaptor stops receiving events.

WOApplication

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOApplication.h

Class Description

The primary role of the WOApplication class is to coordinate the handling of HTTP requests. Each application must have exactly one WOApplication object (or, simply, application object). The application object receives client requests from an HTTP server adaptor, manages the processing that generates a response, and returns that response—typically an object representing a web page—to the adaptor. The adaptor, in turn, forwards the response in a suitable form to the HTTP server that originated the request.

In handling requests, an application object creates and manages one or more sessions; a session (represented by a WOSession object) dedicates resources to a period of access by a single user and stores persistent state during that period. Conceptually, each cycle of the request-response loop (or transaction) takes place within a session.

Besides acting as a facilitator between the adaptor and the rest of the application during request handling, WOApplication performs many secondary functions. It returns pages based on component name, caches page instances and component definitions, provides some facilities for error handling and script debugging, coordinates the different levels of multi-threaded execution, and furnishes a variety of data.

Typical deployment schemes balance the processing load by having multiple application instances per server adaptor. A single application, in turn, can interact with multiple adaptors; for example, an application can simultaneously communicate with secure-socket and Distributed Object adaptors as well as HTTP adaptors.

You can instantiate ready-made application objects from the WOApplication class or you can obtain the application object from a custom subclass of WOApplication. Custom WOApplication subclasses are common in WebObjects applications since there is often a need to override the **awake**, **sleep**, **init**, and request-handling methods. Compiled WOApplication subclasses can take any name, but if the name is anything other than “Application” you must implement your own **main** function to instantiate the application object from this class. However, if the class name is “Application,” you don’t need to modify **main**. In scripted applications, the code in the **Application.wos** file becomes the implementation logic of a WOApplication subclass automatically created at run time; the application object is instantiated from this subclass.

Adopted Protocols

NSLocking	
	– lock
	– unlock

Method Types

Creating	
	– init
	+ application
Obtaining attributes	
	– adaptorsDispatchRequestsConcurrently
	– allowsConcurrentRequestHandling
	– isConcurrentRequestHandlingEnabled
	– baseURL
	– name
	– number
	– path
Locking	
	– lock
	– unlock
	– lockRequestHandling
	– unlockRequestHandling
Managing adaptors	
	– adaptorWithName:arguments:
	– adaptors
Managing cache	
	– setCachingEnabled:
	– isCachingEnabled
Managing sessions	
	– setSessionStore:
	– sessionStore
	– saveSessionForContext:
	– restoreSessionWithID:inContext:
	– createSessionForRequest:

Managing pages	
	– setPageCacheSize:
	– pageCacheSize
	– permanentPageCacheSize
	– setPermanentPageCacheSize:
	– setPageRefreshOnBacktrackEnabled:
	– isPageRefreshOnBacktrackEnabled
	– pageWithName:forRequest:
	– pageWithName:inContext:
Creating elements	
	– dynamicElementWithName:associations:template:languages:
Running	
	– runLoop
	– run
	– setTimeOut:
	– timeOut
	– terminate
	– isTerminating
Handling requests	
	– dispatchRequest:
	– awake
	– takeValuesFromRequest:inContext:
	– invokeActionForRequest:inContext:
	– appendToResponse:inContext:
	– sleep
Handling errors	
	– handleSessionCreationErrorInContext:
	– handlePageRestorationErrorInContext:
	– handleSessionRestorationErrorInContext:
	– handleException:inContext:
Backward compatibility	
	– requiresWOF35RequestHandling
	– requiresWOF35TemplateParser
Scripted class support	
	– scriptedClassWithPath:
	– scriptedClassWithPath:encoding:

Script debugging	
	– logWithFormat:
	– debugWithFormat:
	– trace:
	– traceAssignments:
	– traceObjectiveCMessages:
	– traceScriptedMessages:
	– traceStatements:
	– logTakeValueForDeclarationNamed:type:bindingNamed: associationDescription:value:
	– logSetValueForDeclarationNamed:type:bindingNamed: associationDescription:value:
Statistics report	
	– setStatisticsStore:
	– statisticsStore
	– statistics
Monitor support	
	– monitoringEnabled
	– activeSessionsCount
	– refuseNewSessions:
	– isRefusingNewSessions
	– setMinimumActiveSessionsCount:
	– minimumActiveSessionsCount
	– terminateAfterTimeInterval:
	– logToMonitorWithFormat:
Resource manager support	
	– setResourceManager:
	– resourceManager
Request handling	
	– defaultRequestHandler
	– setDefaultRequestHandler:
	– registerRequestHandler:forKey:
	– removeRequestHandlerForKey:
	– registeredRequestHandlerKeys
	– requestHandlerForKey:
	– handlerForRequest:

User defaults	+ loadFrameworks + setLoadFrameworks: + isDebuggingEnabled + setDebuggingEnabled: + autoOpenInBrowser + setAutoOpenInBrowser: + isDirectConnectEnabled + setDirectConnectEnabled: + cgiAdaptorURL + setCGIAdaptorURL: + isCachingEnabled + setCachingEnabled: + applicationBaseURL + setApplicationBaseURL: + frameworksBaseURL + setFrameworksBaseURL: + recordingPath + setRecordingPath: + projectSearchPath + setProjectSearchPath: + isMonitorEnabled + setMonitorEnabled: + monitorHost + setMonitorHost: + SMTPHost + setSMTPHost: + adaptor + setAdaptor: + port + setPort: + listenQueueSize + setListenQueueSize: + workerThreadCount + setWorkerThreadCount: + additionalAdaptors + setAdditionalAdaptors: + includeCommentsInResponses + setIncludeCommentsInResponses: + componentRequestHandlerKey + setComponentRequestHandlerKey: + directActionRequestHandlerKey + setDirectActionRequestHandlerKey:

+ resourceRequestHandlerKey
+ setResourceRequestHandlerKey:
+ sessionTimeout
+ setSessionTimeOut:

Class Methods

adaptor

+ (NSString *)**adaptor**

Returns the class name of the primary adaptor. This is the cover method for the user default WOAdaptor.

See also: + **setAdaptor:**

additionalAdaptors

+ (NSArray *)**additionalAdaptors**

Returns an array of adaptor description dictionaries. This is the cover method for the user default WOAdditionalAdaptors.

See also: + **setAdditionalAdaptors:**

application

+ (WOApplication *)**application**

Initializes and returns a WOApplication object. This initializes application attributes and initializes the adaptor or adaptors specified on the command line. If no adaptor is specified, WODefaultAdaptor is made the default adaptor. Some of the more interesting attribute initializations are:

- Session store is in the server.
- Page cache size is 30 pages.
- Client caching of pages is enabled (**isPageRefreshOnBacktrackEnabled** returns NO).
- Component-definition caching is disabled (**isCachingEnabled** returns NO).

A exception is raised if initialization does not succeed.

You may call this method, but do not override it.

applicationBaseURL

+ (NSString *)**applicationBaseURL**

Returns a path to where the current application may be found under the document root (either the project or the **.woa** wrapper). This is the cover method for the user default WOApplicationBaseURL.

See also: + **setApplicationBaseURL:**

autoOpenInBrowser

+ (BOOL)**autoOpenInBrowser**

Returns whether automatic browser launching is enabled. By default, automatic browser launching is enabled.

cgiAdaptorURL

+ (NSString *)**cgiAdaptorURL**

Returns the URL for the web server including the path to the WebObjects CGI adaptor (for example, **http: //localhost/cgi-bin/WebObjects**). This URL is used by the direct connect feature only. This is the cover for the user default WOCGIAdaptorURL.

See also: + **setCGIAdaptorURL:**

componentRequestHandlerKey

+ (NSString *)**componentRequestHandlerKey**

Returns the key which identifies URLs directed at component-action-based requests. By default, this method returns the string “wo”.

directActionRequestHandlerKey

+ (NSString *)**directActionRequestHandlerKey**

Returns the key which identifies URLs directed at component-based requests. By default, this method returns the string “wa”.

frameworksBaseURL

+ (NSString *)**frameworksBaseURL**

Returns a path to where all frameworks may be found under the document root. This value is used to determine URLs that should be generated to reference Web Server Resources in those frameworks. This is the cover method for the user default WOFrameworksBaseURL.

See also: + **setFrameworksBaseURL:**

includeCommentsInResponses

+ (BOOL)**includeCommentsInResponses**

Returns whether or not HTML comments are appended to the response. This is the cover method for the user default WOIncludeCommentsInResponses.

See also: + **setIncludeCommentsInResponses:**

isCachingEnabled

+ (BOOL)**isCachingEnabled**

Returns whether or not component caching is enabled. If this is enabled, changes to a component will be reparsed after being saved (assuming the project is under the NSProjectSearchPath). Note that this has no effect on page caching. This is the cover method for the user default WOCachingEnabled.

See also: + **setCachingEnabled:**, – **pageCacheSize**, – **isCachingEnabled**

isDebuggingEnabled

+ (BOOL)**isDebuggingEnabled**

Returns whether or not debugging is enabled. If YES, **debugWithFormat:** prints out. Most startup-time status message are supressed if this method returns NO. By default, debugging is enabled. This is the cover method for the user default WODebuggingEnabled.

See also: – **setDebuggingEnabled:**, – **debugWithFormat:**

isDirectConnectEnabled

+ (BOOL)**isDirectConnectEnabled**

Returns whether or not direct connect is enabled. By default it is enabled. For more information, see **setDirectConnectEnabled:**.

See also: + **cgiAdaptorURL**

isMonitorEnabled

+ (BOOL)**isMonitorEnabled**

Returns whether or not the application can communicate with a Monitor application. It returns YES if the application can contact Monitor upon startup and subsequently let Monitor gather statistics. It returns NO if no comunication with Monitor can take place. By default, it can communicate with a Monitor application. 'This is a cover method for the user default WOMonitorEnabled.

See also: + **setMonitorEnabled:**, + **monitorHost**, + **setMonitorHost:**

listenQueueSize

+ (NSNumber *)**listenQueueSize**

Returns the size of the listen queue which will created by the primary adaptor (usually WODefaultAdaptor). This is the cover method for the user default WOListenQueueSize.

See also: + **setListenQueueSize:**

loadFrameworks

+ (NSArray *)**loadFrameworks**

Returns the array of frameworks to be loaded during application initialization.

See also: + **setLoadFrameworks:**

monitorHost

+ (NSString *)**monitorHost**

Returns the host on which Monitor is assumed to be running. This value is used during initialization if **isMonitorEnabled** returns YES. This is a cover for the user default WOMonitorHost.

See also: + **setMonitorHost:**, + **isMonitorEnabled**

port

+ (NSNumber *)**port**

Returns the port number on which the primary adaptor will listen (usually WODefaultAdaptor). This is the cover method for the user default WOPort.

See also: + **setPort:**

projectSearchPath

+ (NSArray *)**projectSearchPath**

Returns an array of file system paths which are searched for projects for rapid turnaround mode. This is the cover method for the user default NSProjectSearchPath.

See also: + **setProjectSearchPath:**

recordingPath

+ (NSString *)**recordingPath**

Returns a file system path which is where the recording information should be saved. By default, this method returns null.

If this method returns a path, all requests and responses are recorded in the HTTP format in numbered files (**0000-request**, **0000-response**, **0001-request**, **0001-response**, and so on), and saved under the recording path specified. This directory is then used by the Playback tool to test the application. You will most likely set this as a command line argument (`-WORecordingPath pathname`), exercise your application to record a scenario you would like to test, and then stop the application. Afterward you can restart the application without the WORecordingPath argument, and point Playback to the recording directory just created to replay your sequence of requests and compare the responses received with the ones recorded.

See also: + **setRecordingPath:**

resourceRequestHandlerKey

+ (NSString *)**resourceRequestHandlerKey**

Returns the key which identifies URLs directed through the resource request handler. Resource requests are only used during development of an application when the application is being run without an HTTP server.

See also: + **setResourceRequestHandlerKey:**

sessionTimeout

+ (NSNumber*)**sessionTimeOut**

Returns the number (of seconds) which will be used as the default timeout for each newly created session. You may either override this method, change the user default WOSessionTimeOut, or set the session timeout in your session's **init** method.

See also: + **setSessionTimeOut:**

setAdaptor:

+ (void)**setAdaptor:**(NSString *)*anAdaptorName*

Sets the the class name of the primary adaptor to *anAdaptorName*.

See also: + **adaptor**

setAdditionalAdaptors:

+ (void)**setAdditionalAdaptors:**(NSArray *)*anAdaptorPlist*

Sets the array of adaptor description dictionaries to *anAdaptorPlist*. Each adaptor description dictionary must have "WOAdaptor" defined, which is the name of the adaptor class. Other attributes such as WOPort may also be specified, but are adaptor specific. For example WOWorkerThreadCount is specific to the WODefaultAdaptor class and may not apply for all adaptors.

See also: + **additionalAdaptors**

setApplicationBaseURL:

+ (void)**setApplicationBaseURL:**(NSString *)*aBaseURL*

Sets to *aBaseURL* the path to which the current application may be found under the document root (either the project or the **.woa** wrapper).

See also: + **applicationBaseURL**

setAutoOpenInBrowser:

+ (void)**setAutoOpenInBrowser:**(BOOL)*isEnabled*

Controls whether starting up this application also launches a web browser. If *isEnabled* is YES, the application launches the web browser. If NO, the application does not launch the browser. Browser launching is enabled by default as long as there is a WOAdaptorURL key in the file **NeXT_ROOT/NextLibrary/WOAdaptors/Configuration/WebServerConfig.plist**.

To disable web browser launching, you must send this message in the **init** method of your application subclass (or application script).

See also: + **autoOpenInBrowser**

setCGIAdaptorURL:

+ (void)**setCGIAdaptorURL:**(NSString *)*aURL*

Sets the URL for the web server to *aURL*. The URL must include the path to the WebObjects CGI adaptor (for example, **http://localhost/cgi-bin/WebObjects**). This URL is used by the direct connect feature only..

See also: + **cgiAdaptorURL**

setCachingEnabled:

+ (void)**setCachingEnabled:**(BOOL)*flag*

Sets whether or not component caching is enabled. If this is enabled, changes to a component will be reparsed after being saved (assuming the project is under the NSProjectSearchPath). Note that this has no effect on page caching.

See also: + **isCachingEnabled**, – **pageCacheSize**, – **isCachingEnabled**

setComponentRequestHandlerKey:

+ (void)**setComponentRequestHandlerKey:**(NSString *)*key*

Sets the component request handler key. This affects all URLs generated during **appendToResponse: inContext:**: of component-based actions.

See also: + **componentRequestHandlerKey**

setDebuggingEnabled:

+ (void)**setDebuggingEnabled:**(BOOL)*flag*

Sets whether or not debugging is enabled. If YES, **debugWithFormat:** prints out. Most startup-time status message are supressed if this method returns NO. By default, debugging is enabled.

See also: + **isDebuggingEnabled**, – **debugWithFormat:**

setDirectActionRequestHandlerKey:

+ (void)**setDirectActionRequestHandlerKey:**(NSString *)*key*

Sets the Direct Action request handler key. This affects all URLs generated during **appendToResponse: inContext:**: of direct actions.

See also: + **directActionRequestHandlerKey**

setDirectConnectEnabled:

+ (void)**setDirectConnectEnabled:**(BOOL)*flag*

Sets whether or not direct connect is enabled. By default it is enabled.

Direct connect actually transforms your application in a simple web server of its own. In particular, it is then able to find and return its images and resources as if it were a web server. It is very useful in development mode: You don’t need a web server. Just point your URL to the port where your application is listening, and the application will handle all urls.

If this flag is YES, the following happens:

- When using **autoOpenInBrowser**, a direct connect URL will be used.
- When using WOMailDelivery to mail pages with dynamic links in them, these links will be generated with a complete direct connect URL format. People receiving these mails will be able to access the application with direct connect.
- *All files on the system are accessible through the resource request handler.* On the other hand, if this flag is NO, the resource request handler can be used to retrieve data objects from memory only, and no more reading in the file system is permitted (secure mode for deployment).

See also: + **isDirectConnectEnabled**, + **cgiAdaptorURL**

setFrameworksBaseURL:

+ (void)**setFrameworksBaseURL:**(NSString *)*aString*

Sets to *aString* the path to where all frameworks may be found under the document root. This value is used to determine URLs that should be generated to reference Web Server Resources in those frameworks.

See also: + **frameworksBaseURL**

setIncludeCommentsInResponses:

+ (void)**setIncludeCommentsInResponses:**(BOOL)*flag*

Sets whether or not HTML comments are appended to the response.

See also: + **includeCommentsInResponses**

setListenQueueSize:

+ (void)**setListenQueueSize:**(NSNumber *)*aListenQueueSize*

Sets the size of the listen queue which will created by the primary adaptor (usually WODefaultAdaptor).

See also: + **listenQueueSize**

setLoadFrameworks:

+ (void)**setLoadFrameworks:**(NSArray *)*frameworkList*

Sets the array of frameworks to be loaded during application initialization.

See also: + **loadFrameworks**

setMonitorEnabled:

+ (void)**setMonitorEnabled:**(BOOL)*flag*

Sets whether or not the application will communicate with a Monitor application. If *flag* is YES, the application can contact Monitor upon startup and subsequently let Monitor gather statistics. If *flag* is NO, no comunication with Monitor can take place. By default, it can communicate with a Monitor application.

See also: + **isMonitorEnabled**

setMonitorHost:

+ (void)**setMonitorHost:**(NSString *)*hostName*

Sets the host on which Monitor is assumed to be running. This value is used during initialization if **isMonitorEnabled** returns YES.

See also: + **monitorHost**, + **isMonitorEnabled**

setPort:

+ (void)**setPort:**(NSNumber *)*port*

Sets the port number on which the primary adaptor will listen (usually WODefaultAdaptor).

See also: + **port**

setProjectSearchPath:

+ (void)**setProjectSearchPath:**(NSArray)*searchPath*

Sets the array of file system paths which are searched for projects for rapid turnaround mode.

See also: + **projectSearchPath**

setRecordingPath:

+ (void)**setRecordingPath:**(NSString *)*path*

Sets the file system path where the recording information should be saved. Use null as the path if you don't want to save recording information. By default, recording information is not saved.

If you save recording information, all requests and responses are recorded in the HTTP format in numbered files (**0000-request**, **0000-response**, **0001-request**, **0001-response**, and so on), and saved under the recording path specified. This directory is then used by the Playback tool to test the application. You will most likely set this as a command line argument (`-WORecordingPath pathname`), exercise your application to record a scenario you would like to test, and then stop the application. Afterward you can restart the application without the WORecordingPath argument, and point Playback to the recording directory just created to replay your sequence of requests and compare the responses received with the ones recorded.

See also: + **recordingPath**

setResourceRequestHandlerKey:

+ (void)**setResourceRequestHandlerKey:**(NSString *)*key*

Sets the resource request handler key. This affects all URLs generated during **appendToResponse: inContext:**: of resources.

See also: + **resourceRequestHandlerKey**

setSessionTimeOut:

public void **setSessionTImeOut**(java.lang.Number *aTimeOut*)
+ (void)**setSessionTimeOut:**(NSNumber*)*aTimeOut*

Accessor to set the default session timeOut.

See also: + **sessionTimeout**

setSMTPHost:

+ (void)**setSMTPHost:**(NSString *)*hostName*

Sets the name of the host that will be used to send e-mail messages created by WOMailDelivery.

See also: + **SMTPHost**

setWorkerThreadCount:

+ (void)**setWorkerThreadCount:**(NSNumber *)*aWorkerThreadCount*

SEts the count of worker threads which will created by the primary adaptor (usually WODefaultAdaptor). A worker thread count of 0 implies single-threaded mode.

See also: + **workerThreadCount**

SMTPHost

+ (NSString *)**SMTPHost**

Returns the name of the host that will be used to send e-mail messages created by WOMailDelivery. This is the cover method for the user default WOSMTPHost.

See also: + **setSMTPHost:**

workerThreadCount

+ (NSNumber *)**workerThreadCount**

Returns the count of worker threads which will created by the primary adaptor (usually WODefaultAdaptor). A worker thread count of 0 implies single-threaded mode. This is the cover method for the user default WOWorkerThreadCount.

See also: + **setWorkerThreadCount:**

Instance Methods

activeSessionsCount

– (int)**activeSessionsCount**

Returns the number of sessions that are currently active. (A session is active if it has not yet timed out.)

The number returned here is only accurate if the application stores state in memory in the server, which is the default. If you use a custom state-storage strategy, there may be no way to tell how many sessions are active for a given application instance.

See also: – **minimumActiveSessionsCount**, – **setMinimumActiveSessionsCount:**

adaptorWithName:arguments:

– (WOAdaptor *)**adaptorWithName:**(NSString *)*aName*
arguments:(NSDictionary *)*someArguments*

Invoked during the **init** method to create an adaptor. If you subclass WOAdaptor, you specify the WOAdaptor subclass you want the application to use with the **-a** option on the application's command line. When WOApplication encounters the **-a** option, it invokes this method. This method looks for a subclass of WOAdaptor with the name *aName* (which was supplied as the **-a** option's argument), and if such a class exists, a new instance is initialized using the WOAdaptor method **initWithName:arguments:**. The *someArguments* array is populated with any adaptor-specific options (such as **-p** or **-q**) that follow the adaptor name on the command line. See the WOAdaptor class for more information.

See also: – **adaptors**

adaptors

– (NSArray *)**adaptors**

Returns the current list of application adaptors. A WOApplication can have multiple adaptors. (To associate the WOApplication with multiple adaptors, you specify each adaptor on the application's command line using the **-a** option.) This allows you to design an application that can not only listen to a socket for incoming HTTP requests (using the WODefaultAdaptor), but can also receive remote request messages using more advanced RPC mechanisms such as DO, CORBA, and DCOM.

adaptorsDispatchRequestsConcurrently

– (BOOL)**adaptorsDispatchRequestsConcurrently**

Returns YES if at least one adaptor contains multiple threads and will attempt to concurrently invoke the request handlers.

allowsConcurrentRequestHandling

– (BOOL)**allowsConcurrentRequestHandling**

Override to return YES if concurrent request handling is allowed.

appendToResponse:inContext:

– (void)**appendToResponse:**(WOResponse *)*aResponse* **inContext:**(WOContext *)*aContext*

The WOApplication object sends this message to itself to initiate the last phase of request handling. This occurs right after the **invokeActionForRequest:inContext:**: method has completed, typically with the return a response page. In the append-to-response phase, the application objects (particularly the response component itself) generate the HTML content of the page. WOApplication's default implementation of this method forwards the message to the session object.

See also: – **invokeActionForRequest:inContext:**

awake

– (void)**awake**

Invoked at the beginning of each cycle of the request-response loop, affording the opportunity to perform initializations with application-wide scope. Since the default implementation does nothing, overridden implementations do not have to call **super**.

See also: – **sleep**

baseURL

– (NSString *)**baseURL**

Returns the application URL relative to the server's document root, for example:

```
WebObjects/Examples/HelloWorld.woa.
```

See also: – **name**, – **path**

createSessionForRequest:

– (WOSession *)**createSessionForRequest:**(WORequest *)*aRequest*

Creates and returns a WOSession object to manage a session for the application. The method goes through several steps to locate the class to use for instantiating this object:

1. First it looks for a compiled class of name "Session" that is a subclass of WOSession.

2. If such a class does not exist, it looks for a "**.wos**" script with the name of "Session" in the application wrapper ("**.woa**" directory).
3. If the **Session.wos** script exists, the method parses the script and dynamically adds a scripted-class subclass of WOSession to the runtime.

The method then returns an allocated and initialized (using the default WOSession initializer) session instance of the selected class. It raises an exception if it is unable to create a new session.

Note: An implication of the foregoing description is that the names of compiled WOSession subclasses should be "Session"; if not, you will have to override this method to use the proper class to create the session object.

See also: – **restoreSessionWithID:inContext:**, – **saveSessionForContext:**

debugWithFormat:

– (void)**debugWithFormat:**(NSString *)*aFormatString,...*

Prints a message to the standard error device (stderr), if **WODebuggingEnabled** is YES. The message can include formatted variable data using printf-style conversion specifiers. Note that in WebScript, all variables are objects, so the only conversion specifier allowed is **%@**. In compiled Objective-C code, all **printf** conversion specifiers are allowed.

You control whether this method displays output with the **WODebuggingEnabled** user default option. If **WODebuggingEnabled** is YES, then the **debugWithStringFormat:** messages display their output. If **WODebuggingEnabled** is NO, the **debugWithStringFormat:** messages don't display their output.

defaultRequestHandler

– (WORequestHandler *)**defaultRequestHandler**

Returns the request handler to be used when no request handler key was found in the URL or WORequest. This method returns the WOComponent request handler by default. When an application is contacted for the first time it is usually via a URL like the following:

```
http://somehost/cgi-bin/WebObjects/AppName.woa
```

The way that URLs of that type are handled is determined by the default request handler.

dispatchRequest:

– (WOResponse *)**dispatchRequest:**(WORequest *)*aRequest*

The main entry point for any given interaction. Invoked by the adaptor.

dynamicElementWithName:associations:template:languages:

– (WODynamicElement *)**dynamicElementWithName:**(NSString *)*aName*
associations:(NSDictionary *)*someAssociations*
template:(WOElement *)*anElement*
languages:(NSArray *)*languages*

Creates and returns a WODynamicElement object based on the element's name, a dictionary of associations, and a template of elements. This method is invoked automatically to provide a WODynamicElement object that represents a `WEBOBJECT` element in the HTML template. You don't ordinarily invoke **dynamicElementWithName:associations:template:languages:**, but you might override it to substitute your own WODynamicElement or reusable component for one of the built-in WODynamicElements.

The arguments *aName* and *someAssociations* are derived from a corresponding line in the declarations file. *aName* is an NSString that identifies the kind of element to create. Generally *aName* specifies a built-in WODynamicElement such as WOString, but it may also identify a reusable component. (For more information, see the chapter "Using Reusable Components" in the *WebObjects Developer's Guide*.) For example, in the **dynamicElementWithName:associations:template:languages:** message for the following declaration:

```
APP_STRING: WOString {value = applicationString;};
```

aName contains the string "WOString".

The *someAssociations* dictionary contains an entry for each attribute specified in the corresponding declaration. For the declaration above, *someAssociations* contains a single entry for WOString's value attribute. The keys of *someAssociations* are the attribute names and the values are WOAssociation objects.

WOApplication's implementation of **dynamicElementWithName:associations:template:languages:** first searches for a WODynamicElement named *aName*. If a WODynamicElement is found, the method creates an instance using the method **initWithName:associations:template:** and returns it. Otherwise, it searches for a component—either scripted or compiled—to return instead. If neither are found, this method returns **nil**.

handleException:inContext:

– (WOResponse *)**handleException:**(NSException *)*anException* **inContext:**(WOContext *)*aContext*

Invoked when an exception occurs within the request-response loop. The default behavior displays a page with debugging information. You can override this method to catch exceptions and display a "friendlier" error page. For example, the following code replaces the standard error page with a component named **ErrorPage.wo**.

```
- (WOResponse *)handleException:(NSException *)anException {
    WOResponse *response = [[WOResponse alloc] init];
    WORequest *request = [[self context] request];
    WOString newURL = [NSString stringWithFormat:@"http:
//%@%@/%@.woa/-/ErrorPage.wo",
        [request applicationHost],
        [request adaptorPrefix],
        [request applicationName]];

    [response setHeader:newURL forKey:@"location"];
    [response setHeader:@"text/html" forKey:@"content-type"];
    [response setHeader:@"0" forKey:@"content-length"];
    [response setStatus:302];
    return response;
}
```

See also: – **handleSessionCreationErrorInContext:**, – **handleSessionRestorationErrorInContext:**

handlePageRestorationErrorInContext:

– (WOResponse *)**handlePageRestorationErrorInContext:**(WOContext *)*aContext*

Invoked when a page (WOComponent) instance cannot be restored, which typically happens when a user backtracks too far. Specifically, this method is invoked when the following occurs: the request is not the first of a session, page restoration by context ID fails, and page re-creation is disabled. The default behavior displays a page with debugging information. You can override this method to display a “friendlier” error page.

See also: – **handleException:inContext:**, – **handleSessionCreationErrorInContext:**, – **handleSessionRestorationErrorInContext:**

handleSessionCreationErrorInContext:

– (WOResponse *)**handleSessionCreationErrorInContext:**(WOContext *)*aContext*

Invoked when a session (WOSession) instance cannot be created. The default behavior displays a page with debugging information. You can override this method to display a “friendlier” error page.

See also: – **handleException:inContext:**, – **handlePageRestorationErrorInContext:**, – **handleSessionRestorationErrorInContext:**

handleSessionRestorationErrorInContext:

– (WOResponse *)**handleSessionRestorationErrorInContext:**(WOContext *)*aContext*

Invoked when a session (WOSession) instance cannot be restored, which typically happens when the session times out. The default behavior displays a page with debugging information. You can override this method to display a “friendlier” error page.

See also: – **handleException:inContext:**, – **handlePageRestorationErrorInContext:**, – **handleSessionCreationErrorInContext:**

handlerForRequest:

– (WORequestHandler *)**handlerForRequest:**(WORequest *)*aRequest*

Returns the request handler used to handle a given request.

See also: – **registerRequestHandler:forKey:**, – **registeredRequestHandlerKeys**, – **requestHandlerForKey:**

init

– (id)**init**

Initializes application attributes and initializes the adaptor or adaptors specified on the command line. If no adaptor is specified, WODefaultAdaptor is made the default adaptor. Some of the more interesting attribute initializations are:

- Session store is in the server.
- Page cache size is 30 pages.
- Client caching of pages is enabled (**isPageRefreshOnBacktrackEnabled** returns NO).
- Component-definition caching is disabled (**isCachingEnabled** returns NO).

A exception is raised if initialization does not succeed.

Note: The global variable “WOApp” is initialized in this method. Your subclasses of WOApplication (including Application.wos) should be sure to call **super**’s **init** method as their first line of code.

invokeActionForRequest:inContext:

– (WOElement *)**invokeActionForRequest:**(WORequest *)*aRequest*
inContext:(WOContext *)*aContext*

The WOApplication object sends this message to itself to initiate the middle phase of request handling. In this phase, the message is propagated through the objects of the application until the dynamic element that has received the user action (for instance, a click on a button) responds to the message by triggering the method in the request component that is bound to the action. The default WOApplication implementation of this method forwards the message to the session object.

See also: – **appendToResponse:inContext:**

isCachingEnabled

– (BOOL)**isCachingEnabled**

Returns YES if starting up the application also launches a web browser, and NO otherwise. Browser launching is enabled by default as long as there is a WOAdaptorURL key in the file **NeXT_ROOT/NextLibrary/WOAdaptors/Configuration/WebServerConfig.plist**.

See also: + **setAutoOpenInBrowser:**

isConcurrentRequestHandlingEnabled

– (BOOL)**isConcurrentRequestHandlingEnabled**

Returns whether component-definition caching is enabled. The default is NO.

See also: – **setCachingEnabled:**

isPageRefreshOnBacktrackEnabled

– (BOOL)**isPageRefreshOnBacktrackEnabled**

Returns whether caching of pages is disabled in the client. If so, the client does not restore request pages from its cache but re-creates them “from scratch” by resending the URL to the server. This flag is set to NO by default.

See also: – **setPageRefreshOnBacktrackEnabled:**

isRefusingNewSessions

– (BOOL)**isRefusingNewSessions**

Returns YES if the application instance is refusing new sessions, and NO otherwise. When the application instance refuses new sessions, the WebObjects adaptor tries to start the session in another instance of the same application. If no other instance is running and accepting new sessions, the user receives an error message.

isTerminating

– (BOOL)**isTerminating**

Returns whether the application will terminate at the end of the current request-response loop.

See also: – **setTimeOut:**, – **terminate**, – **terminateAfterTimeInterval:**, – **timeOut**

lock

– (void)**lock**

Locks the application object.

lockRequestHandling

– (void)**lockRequestHandling**

Serializes request handler access if concurrent request handling isn't enabled.

logSetValueForDeclarationNamed:type:bindingNamed:associationDescription: value:

– (void)**logSetValueForDeclarationNamed:**(NSString*)*aDeclarationName*
type:(NSString*)*aDeclarationType* **bindingNamed:**(NSString*)*aBindingName*
associationDescription:(NSString*)*anAssociationDescription* **value:**(id)*aValue*

Formats and logs a message anytime a value is set through a WOAssociation, when WODebug is set to YES for the declaration in which the association appears. (Setting a value means the child component/element is setting a value in the parent). See **logTakeValueForDeclarationNamed:type:bindingNamed: associationDescription:value:** for a description of each of the arguments to this method.

logTakeValueForDeclarationNamed:type:bindingNamed:associationDescription: value:

– (void)**logTakeValueForDeclarationNamed:**(NSString*)*aDeclarationName*
type:(NSString*)*aDeclarationType* **bindingNamed:**(NSString*)*aBindingName*
associationDescription:(NSString*)*anAssociationDescription* **value:**(id)*aValue*

Formats and logs a message anytime a value is "taken" through a WOAssociation , when WODebug is set to YES for the declaration in which the association appears. (Taking a value means the child component/element is taking a value from the parent). Override this method to alter the format of the log message. The arguments of this method are defined in the following example of a WebObjects declaration.

```
aDeclarationName : aDeclarationType {
    aBindingName = anAssociationDescription;
}
```

Also, *aValue* is the value which is being pushed to or pulled from the child to the parent.

logToMonitorWithFormat:

– (void)**logToMonitorWithFormat:**(NSString *)*aFormat*,...

Same as **logWithFormat:** but prints the string to the Monitor application's standard error. That is, the message is displayed in the command-shell window that was used to launch the Monitor application.

You use this method to log messages about significant events when the application is ready to be deployed and you will use Monitor regularly to monitor the application. Otherwise, use **logWithFormat:**. If the Monitor application is not running or if this application instance is not being monitored, this method does nothing.

logWithFormat:

– (void)**logWithFormat:**(NSString *)*aFormat*,...

Prints a message to the standard error device (stderr). The message can include formatted variable data using printf-style conversion specifiers, for example:

```
id i = 500;
id f = 2.045;
[self logWithFormat:@"Amount = %@, Rate = %@, Total = %@", i, f, i*f];
```

Note that in WebScript, all variables are objects, so the only conversion specifier allowed is **%@** as shown above. In compiled Objective-C code, all **printf** conversion specifiers are allowed. The equivalent method in Java is **logString**.

minimumActiveSessionsCount

– (int)**minimumActiveSessionsCount**

Returns the minimum number of active sessions allowed. If the number of active sessions is less than or equal to this number and **isRefusingNewSessions** is YES, the application instance terminates. The default is 0.

See also: – **activeSessionsCount**, – **refuseNewSessions:**, – **setMinimumActiveSessionsCount:**

monitoringEnabled

– (BOOL)**monitoringEnabled**

Returns YES if the application is "monitorable" by the Monitor application, and NO otherwise. An application is "monitorable" if it was able to find a running Monitor upon startup and it is able to successfully communicate with that Monitor.

By default, all applications are monitorable if the Monitor application is running on the same machine as the application. You can specifically disable monitoring using the `-WOMonitorEnabled NO` option on the application command line. If you want the application to be monitorable and the Monitor is running on another host, you can start up the application through Monitor, or you can specify Monitor's host on the application command line this way:

```
MyApp.exe -WOMonitorEnabled YES -WOMonitorHost monitorHost ...
```

See also: – **logToMonitorWithFormat:**, the online document *ServingWebObjects*

name

– (NSString *)**name**

Returns the name of the application, which is the name of the executable (without the `.exe` extension).

See also: – **baseURL**, – **path**

number

– (NSString *)**number**

Returns `@"-1"`. This is provided for backwards compatibility only.

pageCacheSize

– (unsigned int)**pageCacheSize**

Returns the size of the internal cache for page instances. The default size is 30 instances.

See also: : – **setPageCacheSize:**

pageWithName:forRequest:

– (WOComponent *)**pageWithName:**(NSString *)*aName* **forRequest:**(WORequest *)*aRequest*

Returns a new page instance (a WOComponent object) identified by *aName*. If *aName* is **nil**, the "Main" component is assumed. If the method cannot create a valid page instance, it raises an exception.

As part of its implementation, this method creates a context with *aRequest* and calls **pageWithName: inContext:**.

See also: – **restorePageForContextID:** (WOSession), – **savePage:** (WOSession)

pageWithName:inContext:

– (WOComponent *)**pageWithName:**(NSString *)*aName* **inContext:**(WOContext *)*aContext*

Returns a new page instance (a WOComponent object) identified by *aName*. If *aName* is **nil**, the "Main" component is assumed. If the method cannot create a valid page instance, it raises an exception.

See also: **pageWithName:forRequest:**, – **restorePageForContextID:** (WOSession), – **savePage:** (WOSession)

path

– (NSString *)**path**

Returns the filesystem path of the application, which is an absolute path and includes the "**.woa**" extension; for example "**C:/NETSCAPE/ns-home/docs/WebObjects/Examples/HelloWorld.woa**" is a typical application path.

See also: – **baseURL**, – **name**

permanentPageCacheSize

– (unsigned int)**permanentPageCacheSize**

Returns the permanent page cache size. The default is 30. The permanent page cache holds pages which should not fall out of the regular page cache. For example, a control page in a frameset should exist for the duration of a session.

See also: **savePageInPermanentCache:** (WOApplication)

refuseNewSessions:

– (void)**refuseNewSessions:**(BOOL)*flag*

Controls whether this application instance will create a session when it receives an HTTP request from a new user. If *flag* is YES, the application does not create new sessions; when it receives a request from a new user, it refuses that request, and the adaptor must try to find another application instance that can process the request. If *flag* is NO, the application creates new sessions. NO is the default.

You use this method with **setMinimumActiveSessionsCount:** to gracefully shut down application instances. Use **setMinimumActiveSessionsCount:** to set the active session minimum to a certain number. When number of active sessions reaches the number you set and **isRefusingNewSessions** returns YES, the application terminates.

See also: – **activeSessionsCount**, – **isRefusingNewSessions**, – **minimumActiveSessionsCount**,
– **setMinimumActiveSessionsCount:**

registerRequestHandler:forKey:

– (void)**registerRequestHandler:**(WORequestHandler *)*aHandler* **forKey:**(NSString *)*aKey*

Registers a new request handler. *aKey* must specify a key which can be found in the URLs following the instance number or application name.

See also: – **removeRequestHandlerForKey:**, – **registeredRequestHandlerKeys**,
– **requestHandlerForKey:**

registeredRequestHandlerKeys

– (NSArray *)**registeredRequestHandlerKeys**

Returns an array of strings containing the keys of all of the registered request handlers.

See also: – **handlerForRequest:**, – **requestHandlerForKey:**

removeRequestHandlerForKey:

– (WORequestHandler *)**removeRequestHandlerForKey:**(NSString *)*aRequestHandlerKey*

Removes the specified request handler from the application.

See also: – **registerRequestHandler:forKey:**, – **requestHandlerForKey:**

requestHandlerForKey:

– (WORequestHandler *)**requestHandlerForKey:**(NSString *)*key*

Returns the request handler used to handle requests containing the specified key.

See also: – **handlerForRequest:**, – **registerRequestHandler:forKey:**, – **registeredRequestHandlerKeys**

requiresWOF35RequestHandling

– (BOOL)**requiresWOF35RequestHandling**

For backward compatibility, if your project depends upon features or side effects of the old request handling, you will want to override this method and return YES. By default, it returns NO.

requiresWOF35TemplateParser

– (BOOL)**requiresWOF35TemplateParser**

For backward compatibility, if your project depends upon features or side effects removed from the new, 4.0 template parser, you will want to override this method and return YES. By default, it returns NO.

resourceManager

– (WOResourceManager *)**resourceManager**

Returns the WOResourceManager object that the application uses to manage resources.

See also: – **setResourceManager:**

restoreSessionWithID:inContext:

– (WOSession *)**restoreSessionWithID:**(NSString *)*aSessionID* **inContext:**(WOContext *)*aContext*

Restores the WOSession object representing a session. In normal request handling, this method is invoked at the start of a cycle of the request-response loop. The default implementation simply invokes WOSessionStore’s **checkoutSessionWithID:request:** method, but raises an exception if the WOSessionStore object is missing.

See also: – **createSessionForRequest:**, – **saveSessionForContext:**

run

– (void)**run**

Runs the application in a near-indefinite run loop in the default run-loop mode. Before starting the run loop, the method sends **registerForEvents** to the application’s adaptors so that they can begin receiving run-loop events. Normally, **run** is invoked in the main function.

See also: – **setTimeOut:**, – **terminate**, – **terminateAfterTimeInterval:**

runLoop

– (NSRunLoop *)**runLoop**

Returns the application's run loop. Use this method when you need a run loop for such things as registering timers.

saveSessionForContext:

– (void)**saveSessionForContext:**(WOContext *)*aContext*

Called at the end of the request handling loop, when the current session object needs to be saved. The default implementation simply invokes WOSessionStore's **checkinSessionForContext:** method, but raises an exception if the WOSessionStore object is missing.

See also: – **restoreSessionWithID:inContext:**

scriptedClassWithPath:

– (Class)**scriptedClassWithPath:**(NSString *)*aPath*

Loads a Webscript-based class with the pathname *aPath* into the application. The specified script is parsed assuming the default string encoding, and the class and categories found in the script file are dynamically added to the runtime.

scriptedClassWithPath:encoding:

– (Class)**scriptedClassWithPath:**(NSString *)*aPath* **encoding:**(NSStringEncoding)*anEncoding*

Loads a scripted class with the pathname *aPath* using the encoding *anEncoding*. The class and categories found in the script file are dynamically added to the runtime. The script must use the `@interface/@implementation` syntax.

sessionStore

– (WOSessionStore *)**sessionStore**

Returns the application's current WOSessionStore object (which, by default, stores state in the server).

See also: – **setSessionStore:**

setCachingEnabled:

– (void)**setCachingEnabled:**(BOOL)*flag*

Enables or disables the caching of component definitions. Component definitions contain templates and other information about pages and subcomponents, and are used to generate instances of those components. When this flag is enabled, the application parses the script (or implementation) file, the HTML, and the declaration (".wod") file of a component once and then stores the resulting component definition. By default, this kind of caching is disabled so that you can edit a scripted component without having to relaunch the application every time to check the results. You should always enable component-definition caching when you deploy an application since performance improves significantly.

Do not confuse this type of caching with page-instance caching (see setPageCacheSize:). Caching Strategies in the class description provides further details.

See also: – **isCachingEnabled**

setDefaultRequestHandler:

– (void)**setDefaultRequestHandler:**(WORequestHandler *)*aHandler*

Sets the default request handler. If, for instance, you want the default request handler to use direct actions, write something like the following:

```
aHandler = [self requestHandlerForKey:@”wa”];
[self setDefaultRequestHandler:aHandler];
```

See also: – **defaultRequestHandler**

setMinimumActiveSessionsCount:

– (void)**setMinimumActiveSessionsCount:**(int)*anInt*

Sets the minimum number of active sessions to *anInt*. The default is 0.

You use this method to gracefully shut down application instances. If the active sessions count reaches this number and isRefusingNewSessions returns YES, the application terminates. You might want to terminate application instances periodically for performance reasons; some applications leak a certain amount of memory per transaction, and shutting down and restarting instances of those applications can free up that memory.

See also: – **activeSessionsCount**, – **isRefusingNewSessions**, – **minimumActiveSessionsCount**, – **refuseNewSessions:**

setPageCacheSize:

– (void)**setPageCacheSize**:(unsigned int)*anInt*

Sets whether caching of page instances will occur and the number of pages the cache will hold. When page-instance caching is enabled, the application stores the WOComponent instance corresponding to the response page in the session. When the page is backtracked to, it restores it from the session and makes it the request page. The state of the page is retained. By default, page-instance caching is enabled, with a cache limit of 30 pages.

You turn page-instance caching off by invoking this method with an argument of zero. In this case, when the user backtracks to a page, the page is not stored in the session and so must be re-created "from scratch." Do not confuse this type of caching with component-definition caching (see **setCachingEnabled:**).

See also: – **pageCacheSize**

setPageRefreshOnBacktrackEnabled:

– (void)**setPageRefreshOnBacktrackEnabled:**(BOOL)*flag*

When *flag* is YES, disables caching of pages by the client by setting the page's expiration-time header to the current date and time. (By default, this attribute is set to NO.) Disabling of client caching affects what happens during backtracking. With client caching turned off, the browser resends the URL to the server for the page requested by backtracking. The application must return a new page to the browser (corresponding to a new WOComponent instance). This behavior is desirable when you do not want the user to backtrack to a page that might be obsolete because of changes that have occurred in the session.

When this flag is turned on and a request corresponding to a client backtrack occurs, the retrieved page will only be asked to regenerate its response. The first two phases of a normal request-response loop (value extraction from the request and action invocation) do not occur.

See Caching Strategies in the class description for further details.

See also: – **isPageRefreshOnBacktrackEnabled**

setPermanentPageCacheSize:

– (void)**setPermanentPageCacheSize:**(unsigned int)*aSize*;

Sets the permanentPageCacheSize to aSize

See also: – **permanentPageCacheSize**

setResourceManager:

– (void)**setResourceManager:**(WOResourceManager *)*aResourceManager*

Sets the WOResourceManager object to *aResourceManager*. WOResourceManager objects search for and retrieve resources from the application directory and from shared framework directories.

See also: – **resourceManager**

setSessionStore:

– (void)**setSessionStore:**(WOSessionStore *)*aSessionStore*

Set the session-store object for the application. By default, an object that stores session state in process memory (that is, in the server) is used. The session-store object specifies the state storage strategy for the whole application. This object is responsible for making session objects persistent. You should set the session store object when the application starts up, before the first request is handled.

See also: – **sessionStore**

setStatisticsStore:

– (void)**setStatisticsStore:**(WOStatisticsStore *)*aStatisticsStore*

Sets the WOStatisticsStore object to *aStatisticsStore*. WOStatisticsStore objects record application statistics while the application runs.

See also: – **statisticsStore**

setTimeOut:

– (void)**setTimeOut:**(NSTimeInterval)*aTimeInterval*

Sets the number of seconds the application can experience inactivity (no HTTP requests) before it terminates execution.

This method differs from **terminateAfterTimeInterval:** in that with this method, the application must be idle for *aTimeInterval* seconds for the application to terminate.**terminateAfterTimeInterval:** terminates the application whether it is active or not.

See also: – **timeOut**

sleep

– (void)**sleep**

Invoked at the conclusion of a request-handling cycle to give an application the opportunity for deallocating objects created and initialized in its awake method. The default implementation does nothing.

statistics

– (bycopyNSDictionary *)**statistics**

Returns a copy of the dictionary containing the application statistics maintained by WOStatisticsStore. This method is used by the Monitor application to retrieve application statistics. If you need to access the statistics internally, use this message instead:

```
[[[WOApplication application] statisticsStore] statistics]
```

statisticsStore

– (WOStatisticsStore *)**statisticsStore**

Returns the WOStatisticsStore object, which records statistics while the application runs.

See also: – **setStatisticsStore:**

takeValuesFromRequest:inContext:

– (void)**takeValuesFromRequest:**(WORequest *)*aRequest* **inContext:**(WOContext *)*aContext*

The component action request handler sends this message to the WOApplication to start the first phase of request handling. In this phase, the message is propagated to the session and component objects involved in the request as well as the request page's dynamic elements. Each dynamic element acquires any entered data or changed state (such as a check in a check box) associated with an attribute and assigns the value to the variable bound to the attribute. The default WOApplication implementation of this method forwards the message to the session object.

See also: – **appendToResponse:inContext:**, – **invokeActionForRequest:inContext:**

terminate

– (onewayvoid)**terminate**

Terminates the application process. Termination does not take place until the handling of the current request has completed.

See also: – **isTerminating**, – **setTimeOut:**

terminateAfterTimeInterval:

– (void)**terminateAfterTimeInterval:**(NSTimeInterval)*aTimeInterval*

Sets the application to terminate itself after *aTimeInterval* seconds has elapsed. After the specified time interval has elapsed, the application immediately stops all current processing. If any sessions are active, users may lose information.

This method differs from **setTimeOut:** in that it does not set idle time; **terminateAfterTimeInterval:**: shuts down the application regardless of whether it is idle.

timeOut

– (NSTimeInterval)**timeOut**

Returns the application’s time-out interval: a period (in seconds) of inactivity before the application terminates execution. The default application time-out interval is a very large number.

See also: – **setTimeOut:**

trace:

– (void)**trace:**(BOOL)*flag*

If *flag* is YES, prints all trace messages (messages for scripted messages, compiled messages, and all statements in the application) to the standard error device. If *flag* is NO, stops printing all trace messages.

See also: – **traceAssignments:**, – **traceObjectiveCMessages:**:, – **traceScriptedMessages:**, – **traceStatements:**

traceAssignments:

– (void)**traceAssignments:**(BOOL)*flag*

If *flag* is YES, prints a message to the standard error device every time an assignment statement is executed. If *flag* is NO, stops printing trace assignment messages.

See also: – **trace:**, – **traceObjectiveCMessages:**:, – **traceScriptedMessages:**, – **traceStatements:**

traceObjectiveCMessages:

– (void)**traceObjectiveCMessages:**(BOOL)*flag*

If *flag* is YES, prints a message to the standard error device every time a message is sent to a compiled class from Webscript. If *flag* is NO, stops printing these messages.

See also: – **trace:**, – **traceAssignments:**, – **traceScriptedMessages:**, – **traceStatements:**

traceScriptedMessages:

– (void)**traceScriptedMessages:**(BOOL)*flag*

If *flag* is YES, prints a message to the standard error device every time a message is sent to a scripted class from Webscript. If *flag* is NO, stops printing trace scripted method messages.

See also: – **trace:**, – **traceAssignments:**, – **traceObjectiveCMessages:**, – **traceStatements:**

traceStatements:

– (void)**traceStatements:**(BOOL)*flag*

If *flag* is YES, prints a message to the standard error device every time a statement in the application is executed from Webscript. If *flag* is NO, stops printing trace statement messages.

See also: – **trace:**, – **traceAssignments:**, – **traceObjectiveCMessages:**, – **traceScriptedMessages:**

unlock

– (void)**unlock**

Unlocks the application object.

unlockRequestHandling

– (void)**unlockRequestHandling**

Disables serialized request handler access if concurrent request handling isn’t enabled.

Notifications

WOApplicationDidFinishLaunchingNotification

Posted just before the application begins waiting for requests. The notification contains the application instance.

The notification contains the application instance.

WOApplicationWillFinishLaunchingNotification

Posted when an application has finished its **init** method. Register to receive this notification if you have an object that wishes to set various setting in the application. For example, if you have a WORequestHandler implemented in a framework and you want to register it with the WOApplication, you would register to receive this notification and then implement a method that register your WORequestHandler with the application.

The notification contains the application instance.

WOAssociation

Inherits From: NSObject

Conforms To: NSObject (NSObject)

Declared In: WebObjects/WOAssociation.h

Class Description

The WOAssociation class cluster's single public class, WOAssociation, declares the programmatic interface to objects that represent the values of WebObject attributes, as specified in a declarations file. You rarely need to create subclasses of WOAssociation, except in situations where you need to subclass WODynamicElement.

The purpose of a WOAssociation object is to provide a unified interface to values of different types. For example, consider these declarations:

```
TREENAME1:WOString {value = "Ash"};
TREENAME2:WOString {value = treeName};
TREENAME3:WOString {value = selectedTree.name};
```

At runtime, the WebObjects parser scans an HTML template and these declarations and creates three WOString dynamic element objects. In the first case, the WOString's **value** attribute is assigned a constant string. In the second, it's associated with the **treeName** variable of the component in which the dynamic element is declared. In the third, **value** is associated with the **name** attribute of the component's **selectedTree** variable. The search path for the value can be arbitrarily deep, depending on the needs of your application:

```
MAYOR:WOString {value = country.state.city.mayor.name};
```

To resolve a path such as this, WebObjects accesses each part in turn. First, it looks for the component's **country** variable. If the component responds to a **country** message, it sends one to determine the value; otherwise, it directly accesses the component's **country** instance variable to determine the value. Next, it checks the **country** object for a **state** attribute, using the same strategy of looking for an accessor method named **state** and then, if necessary, accessing the **state** variable's value directly. It continues in this way until the ultimate value is determined.

WOAssociation objects present the WebObjects framework with a unified interface to attribute values, whether their values are static or dynamic. The value attribute for TREENAME1 in the example above will never change during the course of program execution, but the other WOStrings have values that are potentially dynamic, and so will have to be determined at runtime. Since the value of any WOAssociation can be determined by sending it a **valueInComponent:** message, objects that use WOAssociation objects don't have to be concerned with how values are resolved. The WODynamicElement class makes extensive use of this feature. See the WODynamicElement class specification for more information.

Adopted Protocols

NSCopying	– copy – copyWithZone:

Method Types

Creation	+ associationWithKeyPath: + associationWithValue:
Obtaining association attributes	– isValueConstant – isValueSettable
Setting and retrieving value	– setValue:inComponent: – valueInComponent:

Class Methods

associationWithKeyPath:

+ (WOAssociation *)**associationWithKeyPath:**(NSString *)*aKeyPath*

Creates and returns a WOAssociation object whose value is determined by evaluating *aKeyPath*. This method is used when a dynamic element's attribute is set to a variable from the component's script. For example, when the WebObjects parser sees a declaration of this sort,

```
TREENAME3:WOString {value = selectedTree.name};
```

it invokes **associationWithKeyPath:** to create a WOAssociation whose key is "selectedTree.name". When the resulting WOAssociation is asked for its value, it searches for the value of the **name** attribute of in the current component's **selectedTree** attribute.

If *aKeyPath* is **nil**, the value of the WOAssociation is also **nil**.

See also: – **associationWithValue:**

associationWithValue:

+ (WOAssociation *)**associationWithValue:**(id)*aValue*

Creates and returns a WOAssociation object whose value is *aValue*, a constant value. This method is used when a dynamic element's attribute is set to a constant. For example, when the WebObjects parser sees a declaration of this sort,

```
TREENAME3:WOString {value = "Time Flies!"};
```

it invokes this method to create a WOAssociation whose value is "Time Flies!".

See also: + **associationWithKeyPath:**

Instance Methods

isValueConstant

– (BOOL)**isValueConstant**

Returns YES if the WOAssociation's value is a constant, NO otherwise.

See also: – **`associationWithValue:`**, – **isValueSettable**

isValueSettable

– (BOOL)**isValueSettable**

Returns NO if the receiver's value is constant, YES otherwise.

See also: + **`associationWithKeyPath:`**, – **isValueConstant**

setValue:inComponent:

– (void)**setValue:**(id)*aValue* **inComponent:**(WOComponent *)*aComponent*

Finds the attribute of *aComponent* pointed to by the left-hand-side of the receiver and set its value to *aValue*. This method raises NSInternalInconsistencyException if the receiver's value is not settable. For example, sending a **setValue:inComponent:** message to a WOAssociation created from this declaration,

```
USER:WOTextField {value = userName};
```

sets the current component's **userName** variable to the value typed into the WOTextField.

One way in which the WebObjects framework uses this method is to synchronize the values of nested components. When attributes in child and parent components are associated with one another and changes occur in one component, this method is invoked to migrate those changes to the other component. See the reusable components chapter in the *WebObjects Developer's Guide* for more information.

See also: – **valueInComponent:**

valueInComponent:

– (id)**valueInComponent:**(WOComponent *)*aComponent*

Returns a value based on the receiver's association and the current component. For example, sending a **value** message to a WOAssociation created from this declaration,

```
DOWNPAYMENT:WOString {value = downpayment};
```

returns the value of the current component's **downpayment** variable.

Sending a **value** message to a WOAssociation created from this declaration,

```
DOWNPAYMENT:WOString {value = "$5000.00"};
```

returns the value "$5000.00" (independent of the current component).

This method raises an exception if it cannot resolve the WOAssociation's value with the current component.

One way in which the WebObjects framework uses this method is to synchronize the values of nested components. When attributes in child and parent components are associated with one another and changes occur in one component, this method is invoked to migrate those changes to the other component. See the reusable components chapter in the *WebObjects Developer's Guide* for more information.

See also: – **setValue:inComponent:**

WOComponent

Inherits From:	WOElement : NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOComponent.h

Class Description

WOComponent objects dynamically render web pages (or sections of pages) at run time. They provide custom navigation and other logic for the page, provide a framework for organizing constituent objects (static and dynamic HTML elements and subcomponents), and enable the attribute bindings of dynamic elements.

The WOComponent class has many methods that have the same names as methods of the WOApplication class. However, the scope of the WOComponent methods is limited to a component rather than being application-wide. For example, you can control component-definition caching on a per-component basis using **setCachingEnabled:**, which has a WOApplication counterpart. When this kind of caching is enabled for a component, the application parses the contents of the component directory the first time the component is requested, creates the component definition, stores this object in memory, and restores it for subsequent requests.

WOComponent objects also respond to **awake**, **sleep**, and the three request-handling messages: **takeValuesFromRequest:inContext:**, **invokeActionForRequest:inContext:**,and **appendToResponse: inContext:**. You can override these methods in your compiled subclasses, and thereby integrate your custom behavior into the request-response loop. (You can also override these methods in component scripts using WebScript.)

Subcomponents

A WOComponent object can represent a dynamic fragment of a Web page as well as an entire page. Such *subcomponents*, or *reusable components*, are nested within a parent component representing the page *or* another subcomponent. Each component keeps track of its parent and subcomponents—when a component receives a request-handling message, such as **takeValuesFromRequest:inContext:**, it forwards that message to its subcomponents

The WOComponent class also provides a child-parent callback mechanism to allow a child component to communicate with its parent. In the parent’s declaration file, bind an arbitrary attribute of the child to an action method of the parent. Then, as the last step in the child’s action method, invoke **performParentAction:** with the argument being the arbitrary attribute, returning the object received back as the response page. See the method description for **performParentAction:** for details.

Adopted Protocols

NSCoding	
	– encodeWithCoder:
	– initWithCoder:
NSCopying	
	– copy
	– copyWithZone:

Method Types

Creation	
	– init
Obtaining attributes	
	– application
	– baseURL
	– context
	– frameworkName
	– hasSession
	– name
	– pageWithName:
	– path
	– session
Caching	
	– isCachingEnabled
	– setCachingEnabled:
Managing resources	
	– templateWithName:
	– pathForResourceNamed:ofType:
Handling requests	
	– appendToResponse:inContext:
	– awake
	– invokeActionForRequest:inContext:
	– sleep
	– takeValuesFromRequest:inContext:
Logging	
	– debugWithFormat:
	– logWithFormat:
	– logWithFormat:arguments:
	– validationFailedWithException:value:keyPath:

Template parsing	
	+ templateWithHTMLString:declarationString:languages:
Components statistics	
	– descriptionForResponse:inContext:
Invoking actions	
	– parent
	– performParentAction:
Synchronizing components	
	– hasBinding:
	– setValue:forBinding:
	– synchronizesVariablesWithBindings
	– valueForBinding:
Other	
	– generateResponse

Class Methods

templateWithHTMLString:declarationString:languages:

+ (WOElement *)**templateWithHTMLString:**(NSString *)*anHTMLString* **declarationString:** (NSString *)*aDeclarationString* **languages:**(NSArray*)*languages*

Programmatically creates the component's template using *anHTMLString* as the HTML template contents and *aDeclarationString* as the declarations file contents. Returns (as a WOElement object) the graph of static and dynamic elements build by parsing the HTML and declaration strings. You can then use the returned WOElement as the component's template.

See also: – **`templateWithName:`**

Instance Methods

appendToResponse:inContext:

– (void)**appendToResponse:**(WOResponse *)*aResponse* **inContext:**(WOContext *)*aContext*

Component objects associated with a response receive this message during the last phase of the request-response loop. In the append-to-response phase, the application objects (particularly the response page instance itself) generate the HTML content of the page. WOComponent's default implementation of this method forwards the message to the root WOElement object of the component template. Compiled or scripted subclasses of WOComponent can override this method to replace or supplement the default behavior with custom logic.

See also: – **`invokeActionForRequest:inContext:`**, – **`takeValuesFromRequest:inContext:`**

application

– (WOApplication *)**application**

Returns the WOApplication object for the current application.

See also: `WOApplication class`, – **`context`**, **– `session`**

awake

– (void)**awake**

Invoked at the beginning of a WOComponent's involvement in a cycle of the request-response loop, giving the WOComponent an opportunity to initialize its instance variables or perform setup operations. The default implementation does nothing.

See also: – **`init`**, **– `sleep`**

baseURL

– (NSString *)**baseURL**

Returns the component URL relative to the server's document root, for example: "/WebObjects/MyApp.woa/Resources/Main.wo"

See also: – **`name`**, – **`path`**

context

– (WOContext *)**context**

Returns the WOContext object for the current transaction.

See also: WOContext `class`, – **`application`**, **– `session`**

debugWithFormat:

– (void)**debugWithFormat:**(NSString *)*aFormatString,...*

Like **logWithFormat:**, prints a message to the standard error device (stderr), but only prints the message if the WODebuggingEnabled user default option is YES. If WODebuggingEnabled is NO, the **debugWithFormat:** messages aren't printed. See **logWithFormat:** for information on the format of *aFormatString*.

See also: – **logWithFormat:arguments:**

descriptionForResponse:inContext:

– (NSString *)**descriptionForResponse:**(WOResponse *)*aResponse*
inContext:(WOContext *)*aContext*

Records information about the component if it is the response component in the current request-response loop transaction. The default implementation records the component's name. You might override this method if you want to record more information about the component. For example, you might want to record the values of some instance variables as well as the component name.

This message is sent only to the top-level response component, that is, the one representing the entire page. Components nested inside of that top-level component do not receive this message.

If a CLFF log file is kept for this application, the string returned by this method is recorded in that log file. Thus, you must ensure that the string you return can be analyzed by a CLFF-analysis tool.

See also: `WOStatisticsStore class`

frameworkName

– (NSString *)**frameworkName**

If the component is stored in a framework, this method returns the name of that framework. For example, if the component is in the framework *NeXT_ROOT*/**System/Library/Frameworks/WOExtensions.framework**, then this method returns the string "WOExtensions".

If the component is not stored in a framework, this method returns **nil**.

See also: `WOResourceManager class`

generateResponse

– (WOResponse *)**generateResponse**

Returns a newly-created WOResponse object. WOComponent's implementation of this method translates the receiving component into a WOResponse object by sending iteself an **appendToResponse:inContext:** message.

See also: – **generateResponse** (WOResponse)

hasBinding:

– (BOOL)**hasBinding:**(NSString *)*aBindingName*

Returns whether the component has a binding named *aBindingName*.

hasSession

– (BOOL)**hasSession**

Returns whether the component is already in a session. For example, in direct actions, sessions are lazily created and you can avoid creating another one unnecessarily by calling **hasSession** before **session**.

See also: **– session**

init

– (id)**init**

Initializes a WOComponent object. If a WebObjects Builder archive file exists in the component directory, it initializes component variables from this archive. An exception is thrown if the method cannot determine the name of the component or if it cannot initialize the object for any other reason. Override **init** in compiled subclasses to perform custom initializations; as always, invoke **super**'s **init** method as the first thing.

See also: **– awake**

invokeActionForRequest:inContext:

– (WOElement *)**invokeActionForRequest:**(WORequest *)*aRequest*
  inContext:(WOContext *)*aContext*

WOComponent objects associated with a request page receive this message during the middle phase of request handling. In this middle phase, the **invokeActionForRequest:inContext:** message is propagated through the WOElement objects of the page; the dynamic element on which the user has acted (by, for example, clicking a button) responds by triggering the method in the request component that is bound to the action. WOComponent's default implementation of this method forwards the message to the root WOElement object of the component template.Compiled or scripted subclasses of WOComponent can override this method to replace or supplement the default behavior with custom logic.

See also: – **appendToResponse:inContext:**, – **takeValuesFromRequest:inContext:**

isCachingEnabled

– (BOOL)**isCachingEnabled**

Returns whether component-definition caching is enabled for this component. NO is the default.

See also: **– setCachingEnabled:**

logWithFormat:

– (void)**logWithFormat:**(NSString *)*aFormat,...*

Prints a message to the standard error device (stderr). The message can include formatted variable data using **printf**-style conversion specifiers, for example:

```
id i = 500;
id f = 2.045;
[self logWithFormat:@"Amount = %@, Rate = %@, Total = %@",
i, f, i*f];
```

Note that in WebScript, all variables are objects, so the only conversion specifier allowed is %@ as shown above. In compiled Objective-C code, all **printf** conversion specifiers are allowed. The equivalent method in Java is **logString**.

logWithFormat:arguments:

– (void)**logWithFormat:**(NSString *)*aFormat* **arguments:**(va_list)*someArguments*

Prints a message to the standard error device (stderr). This method is used by **logWithFormat:**.

name

– (NSString *)**name**

Returns the name of the component minus the ".wo" extension; for example "Main" is a typical component name.

See also: **– baseURL, – path**

pageWithName:

– (WOComponent *)**pageWithName:**(NSString *)*aName*

Returns a new page instance (a WOComponent object) identified by *aName*. If *aName* is **nil**, the “Main” component is assumed. If the method cannot create a valid page instance, it raises an exception.

See also: – **restorePageForContextID:** (WOSession), – **savePage:** (WOSession)

parent

– (WOComponent *)**parent**

Returns the parent component of the receiver.

path

– (NSString *)**path**

Returns the file-system path of the component, which is an absolute path and includes the “.wo” extension; for example “C:\Apple\Library\WOApps\MyApp.woa\Resources\Main.wo” is a typical path.

See also: `– baseURL, – name`

pathForResourceNamed:ofType:

– (NSString *)**pathForResourceNamed:**(NSString *)*aName* **ofType:**(NSString *)*aType*

Returns the absolute path to the component resource having the name of *aName* and an extension of *aType*. The method searches all localized “.lproj” directories of the component before searching directly under the “.wo” component directory for a non-localized resource of the given name and extension.

This method is provided for backwards compatibility only. For WebObjects 3.5 and above, you should use the WOResourceManager API to retrieve resources. WOResourceManager is not able to retrieve resources stored inside component directories.

performParentAction:

– (id)**performParentAction:**(NSString *)*anActionName*

Allows a subcomponent to invoke an action method of its parent component bound to the child component (*attribute*). Parent and child components are “synchronized” when this method returns: the variables that are bound by a declaration of the child component in the parent component’s declaration file have the same value.

An example best illustrates this mechanism. Let’s say you have a Palette subcomponent, and this WOComponent is nested in a parent component with a “displaySelection” action method. When the user selects an item in the palette (perhaps a color), you want to invoke “displaySelection” to show the result of the new selection (perhaps a car in the new color). The declaration in the parent’s “.wod” file would look like this:

```
PALETTE: Palette {
   selection = number;
   callBack = "displaySelection";
};
```

The “callBack” item is an arbitrary attribute of the child component bound in this declaration to the parent component’s “displaySelection” method.The **performParentAction:** method is used to activate this binding. Let’s assume the child component has an action method called “click”; the implementation would look like this:

```
- click {             /* this is the child’s action */
    selection = /* some value */;
    /* now invoke the parent’s action */
    return [self performParentAction:callBack];
}
```

session

– (WOSession *)**session**

Returns the current WOSession object. This method creates a new one if there isn’t one.

See also: WOSession class, – **application**, – **context**, – **hasSession**

setCachingEnabled:

– (void)**setCachingEnabled:**(BOOL)*flag*

Enables or disables the caching of component definitions for the receiving component. WOComponent definitions contain templates and other common information related to components, and are used to generate instances of those components.When this attribute is set to YES, the application parses the HTML template and the declaration (“.wod”) file of a component once and then stores the resulting component definition for future requests. By default, this kind of caching is disabled so that you can edit a *scripted* component without having to relaunch the application every time to check the results.(Note that this does not apply to Java subclasses of WOComponent; in this case, you still have to kill and relaunch the application.)

With WOApplication’s method of the same name, you can turn component-definition caching off globally. You can then control caching of individual component definitions using WOComponent’s version of this method. Selective caching is an especially valuable technique for very large applications where only the most frequently requested components should be cached.

See also: **– isCachingEnabled**

setValue:forBinding:

– (void)**setValue:***aValue* **forBinding:**(NSString *)*aBindingName*

Sets the value of the binding specified by *aBindingName* in the parent component to *aValue*. If the parent doesn't provide *aBindingName* in its declarations file, this method attempts to set the value in the current component using **takeValue:forKey:**. If the current component doesn't define this key, this method silently returns.

See also: – **synchronizesVariablesWithBindings**, – **valueForBinding:**

sleep

– (void)**sleep**

Invoked at the conclusion of a request-handling cycle to give component the opportunity for deallocating objects created and initialized in its **awake** method. The default implementation does nothing.

synchronizesVariablesWithBindings

– (BOOL)**synchronizesVariablesWithBindings**

Returns whether a nested component pulls all values down from its parent and pushes all values to its parent before and after each phase of the request-response loop. By default, this method returns YES. Override this method to create a non-synchronizing component.

See also: – **setValue:forBinding:**, – **valueForBinding:**

takeValuesFromRequest:inContext:

– (void)**takeValuesFromRequest:**(WORequest *)*aRequest* **inContext:**(WOContext *)*aContext*

WOComponent objects associated with a request receive this message during the first phase of the request-response loop. The default WOComponent behavior is to send the message to the root object of the component's template.In this phase, each dynamic element in the template extracts any entered data or changed state (such as a check in a check box) associated with an attribute and assigns the value to the component variable bound to the attribute.Compiled or scripted subclasses of Component can override this method to replace or supplement the default behavior with custom logic.

See also: – **appendToResponse:inContext:**, – **invokeActionForRequest:inContext:**

templateWithName:

– (WOElement *)**templateWithName:**(NSString *)*aName*

Returns the root object of the graph of static and dynamic HTML elements and subcomponents that is used to graphically render the component identified by *aName*. This template is constructed from the ".html" and ".wod" file found in the component directory. You identify the template by specifying the component directory, which consists of the component name plus the "wo" extension: for example, "HelloWorld.wo." If the template is not cached, the application will parse the HTML and declaration files of the specified component to create the template.

See also: – **setCachingEnabled:**

validationFailedWithException:value:keyPath:

– (void)**validationFailedWithException:**(NSException *)*exception*
value:(id)*value*
keyPath:(NSString *)*keyPath*

Called when an Enterprise Object or formatter failed validation during an assignment. The default implementation ignores the error. Subclassers can override to record the error and possibly return a different page for the current action.

valueForBinding:

– **valueForBinding:**(NSString *)*aBindingName*

Gets the value for the specified binding from the parent component. If the parent doesn't provide *aBindingName* in its delcarations file, this method attempts to get the value from the current component using **valueForKey:**. If the current component doesn't define this key, this method returns nil. This cascading lookup makes it easy to provide default values for optional bindings.

See also: – **setValue:forBinding:**, – **synchronizesVariablesWithBindings**

WOContext

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOContext.h

Class Description

A WOContext object lets you access objects and information that define the *context* of a transaction. In a typical request-response loop (a transaction), several objects have a hand in what is going on: the WOApplication and WOSession objects, the page involved in the request or response (a WOComponent object), the page's subcomponents (also WOComponents), plus the dynamic elements on the page. The WOContext object passed as an argument in the **takeValuesFromRequest:inContext:**, **invokeActionForRequest:inContext:**, and **appendToResponse:inContext:** messages allows access to these objects. A context is identified by the *context ID*, which appears in the URL after the session ID and page name. Each context ID is an integer that the session increments each time a new context is created.

WOContext objects provide other information and services related to the current transaction. From them you can get the entire URL currently in effect as well as portions of that URL, such as the element ID, the context ID, and the URL up to and including the session ID.

A WOContext object plays a further role behind the scenes. For the benefit of a page's dynamic elements, it keeps track of the *current component*, that is, the WOComponent associated with the current element in the request-handling cycle. The current component can be the WOComponent that represents one of the page's subcomponents or the page itself. By reference to the current component (accessed through WOContext's **component** method), a dynamic element can exchange values associatively between itself and the WOComponent that contains it.

Adopted Protocols

NSCopying

– copy
– copyWithZone:

Method Types

Creating new object instances

+ contextWithRequest:
– init
– initWithRequest:

Obtaining attributes

– component
– contextID
– elementID
– hasSession
– isInForm
– page
– request
– response
– session
– senderID
– setInForm:

Manipulating element ID

– appendElementIDComponent:
– appendZeroElementIDComponent
– deleteAllElementIDComponents
– deleteLastElementIDComponent
– incrementLastElementIDComponent

Generating URLs

– directActionURLForActionNamed:queryDictionary:
– completeURLWithRequestHandlerKey:path:queryString:isSecure:
port:
– componentActionURL
– urlWithRequestHandlerKey:path:queryString:

Class Methods

contextWithRequest:

+ (WOContext *)**contextWithRequest:**(WORequest *)*aRequest*

Creates and returns a WOContext with *aRequest*. This is the preferred way to create a WOContext. All other constructors call this one, so if you subclass WOContext, you need to override only this one.

Instance Methods

appendElementIDComponent:

– (void)**appendElementIDComponent:**(NSString *)*aString*

Appends a string to the current element ID to create an identifier of an HTML element. For example, if the current element ID is "0.1.1" and you send this message with an argument of "NameField," the element ID for that field becomes "0.1.1.NameField".

See also: – **`deleteAllElementIDComponents`**, – **`deleteLastElementIDComponent`**, – **`incrementLastElementIDComponent`**

appendZeroElementIDComponent

– (void)**appendZeroElementIDComponent**

Appends a ".0" to the current element ID to create an identifier of the first "child" HTML element. For example, if the current element ID is "0.1.1", after you send this message the element ID becomes "0.1.1.0".

See also: – **`deleteAllElementIDComponents`**, – **`deleteLastElementIDComponent`**, – **`incrementLastElementIDComponent`**

completeURLWithRequestHandlerKey:path:queryString:isSecure:port:

– (NSString *)**completeURLWithRequestHandlerKey:**(NSString *)*requestHandlerKey*
path:(NSString *)*aRequestHandlerPath*
queryString:(NSString *)*aQueryString*
isSecure:(BOOL)*isSecure*
port:(int)*somePort*

Returns the complete URL for the specified request handler. The *requestHandlerKey* is one of the keys provided by WOApplication. The *requestHandlerPath* is any URL encoded string. The *queryString* is added at the end of the URL behind a “?”. If *isSecure* is YES, this method uses “https” instead of “http.” If *somePort* is 0 (zero), this method uses the default port.

See also: – **urlWithRequestHandlerKey:path:queryString:**

component

– (WOComponent *)**component**

Returns the component that dynamic elements are currently using to push and pull values associatively. This component could represent the current request or response page or a subcomponent of that page.

See also: `WOComponent class`, – **`page`**, – **`request`**, – **`response`**, – **`senderID`**

componentActionURL

– (NSString *)**componentActionURL**

Returns the complete URL for the component action.

contextID

– (NSString *)**contextID**

Returns the context ID of the receiver.

deleteAllElementIDComponents

– (void)**deleteAllElementIDComponents**

Deletes all components of the current element ID.

See also: **`– appendElementIDComponent:`**, **`– appendZeroElementIDComponent`**, **`– incrementLastElementIDComponent`**

deleteLastElementIDComponent

– (void)**deleteLastElementIDComponent**

Deletes the last digit (or name) of the current element ID, along with its dot separator. Thus, after sending this message, "0.0.1.1" becomes "0.0.1".

See also: **`– appendElementIDComponent:`**, **`– appendZeroElementIDComponent`**, **`– incrementLastElementIDComponent`**

directActionURLForActionNamed:queryDictionary:

– (NSString *)**directActionURLForActionNamed:**(NSString *)*anActionName* **queryDictionary:**(NSDictionary *)*aQueryDict*

Returns the complete URL for the specified action. You can specify *aQueryDict*, and *anActionName* can be @"ActionClass/ActionName" or @"ActionName".

See also: WODirectAction class specification

elementID

– (NSString *)**elementID**

Returns the element ID identifying the current WOElement.This method helps you avoid creating a session in direct actions.

hasSession

– (BOOL)**hasSession**

Returns whether a session exists for the receiving context.

See also: – **senderID**

incrementLastElementIDComponent

– (void)**incrementLastElementIDComponent**

Increments the last digit of the current element ID. For example, after this message is sent, "0.0.1.2" becomes "0.0.1.3".

See also: – **appendElementIDComponent:**, – **appendZeroElementIDComponent**, – **deleteAllElementIDComponents**, – **deleteLastElementIDComponent**

init

– (id)**init**

Returns a WOContext instance initialized with a unique context ID. Generally, you should call **initWithRequest:** instead to ensure that the WOContext instance is properly initialized.

initWithRequest:

– (id)**initWithRequest:**(WORequest *)*aRequest*

Returns a WOContext with *aRequest*.

isInForm

– (BOOL)**isInForm**

Returns YES when in the context of a WOForm.

See also: **setInForm:**

page

– (WOComponent *)**page**

Returns the WOComponent object that represents the request or response page.

See also: – **component**, – **request**, – **response**, – **senderID**

request

– (WORequest *)**request**

Returns the transaction's WORequest object.

See also: – **component**, – **page**, – **response**, – **senderID**

response

– (WOResponse *)**response**

Returns the transaction's WOResponse object.

See also: – **component**, – **page**, – **response**, – **senderID**

senderID

– (NSString *)**senderID**

Returns the part of the WORequest's URI that identifies the dynamic element on the page (such as a form or an active image) responsible for submitting the request. The sender ID is the same as the element ID used to identify the dynamic element. A request's sender ID may be **nil**, as it always is on the first request of a session.

See also: **– initWithRequest:,** – **request** , **– uri (WORequest)**

session

– (WOSession *)**session**

Returns the object representing the receiving context's session, if one exists. If the receiver does not have a session, this method creates a new session object and returns it. Note that not all contexts have a session: Direct Actions, for instance, don't always need a session. Use **hasSession** to determine whether a context has a session associated with it.

See also: – **component**, – **page**, – **request**, – **response**, WOSession class

setInForm:

– **setInForm:**(BOOL)*flag*

If you write something that behaves like a WOForm, set this to notify WODynamicElements that they are in a form.

See also: **isInForm**

urlWithRequestHandlerKey:path:queryString:

– (NSString *)**urlWithRequestHandlerKey:**(NSString *)*requestHandlerKey*
path:(NSString *)*aRequestHandlerPath*
queryString:(NSString *)*aQueryString*

Returns a URL relative to `cgi-bin/WebObjects` for the specified request handler. The *requestHandlerKey* is one of the keys provided by WOApplication. The *requestHandlerPath* is any URL encoded string. The *queryString* is added at the end of the URL behind a “?”.

– **completeURLWithRequestHandlerKey:path:queryString:isSecure:port:**

WOCookie

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOCookie.h

Class Description

WOCookie is used for the creation and setting of cookies in your response objects. A cookie allows for the persistent storage of client state. Instead of using a WOSession object (which can potentially have a shorter life span), a cookie allows server-side applications to store state in client browsers for a specific or indeterminate amount of time. An advantage to cookies is that the data will be stored on the client and not on the server, allowing the server to maintain less state information. A specific advantage in WebObjects applications is that cookies allow the server to put state into the browser that is not bound to a session. Hence, the client can “leave” your application and return with its cookie’s state intact.

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

- cookieWithName:value:
- cookieWithName:value:path:domain:expires:isSecure:

To add or remove cookies from the response, use the WOResponse methods **addCookie:** and **removeCookie:**. To retrieve cookie data, use the WORequest methods **cookieValues**, **cookieValueForKey:**, and **cookieValuesForKey:**. WORequest returns the data as name/value pairs and not as WOCookie objects, since browsers don’t return the additional data WOCookies provide, such as path name and expiration date.

For more information about cookies and their implementation details, see Netscape’s preliminary specification at **http://www.netscape.com/newsref/std/cookie_spec.html** and RFC 2109 - HTTP State Management Mechanism at **http://www.cis.ohio-state.edu/htbin/rfc/rfc2109.html**.

If and when new details evolve in the implementation of cookies, you can subclass WOCookie and implement new behaviors. Pay particular attention to how you override **headerString**, which WOResponse uses to fill the HTTP response with a header string.

Method Types

Creation	
	+ cookieWithName:value:
	+ cookieWithName:value:path:domain:expires:isSecure:
	– initWithName:value:path:domain:expires:isSecure:
Obtaining a cookie’s attributes	
	– domain
	– expires
	– headerString
	– isSecure
	– name
	– path
	– value
Setting a cookie’s attributes	
	– setDomain:
	– setExpires:
	– setIsSecure:
	– setName:
	– setPath:
	– setValue:

Class Methods

cookieWithName:value:

+ (WOCookie *)**cookieWithName:**(NSString *)*aName* **value:**(NSString *)*aValue*

Creates and returns a cookie with just a name and its value. It sets the path attribute to your application’s path.

See also: – **cookieWithName:value:path:domain:expires:isSecure:**

cookieWithName:value:path:domain:expires:isSecure:

+ (WOCookie *)**cookieWithName:**(NSString *)*aName*
  value:(NSString *)*aValue*
  path:(NSString *)*aPath*
  domain:(NSString *)*aDomain*
  expires:(NSDate *)*expirationDate*
  isSecure:(BOOL)*flag*

Creates and returns a cookie, specifying all its attributes. For more information, see the descriptions of the methods that return attribute values.

See also: – **cookieWithName:value:**, – **domain**, – **expires**, – **isSecure**, – **name**, – **path**, – **value**

Instance Methods

domain

– (NSString *)**domain**

Returns the value of the cookie’s “domain” attribute. It’s of the form “companyname.com”.

expires

– (NSDate *)**expires**

Returns the value of the cookie’s “expires” attribute as an NSDate. The expiration date tells the browser how long to keep the cookie in its cache. To have the browser remove the cookie from its cache, set the expiration date to a date in the past.

headerString

– (NSString *)**headerString**

Returns the string that will be used in the HTTP header. The returned string has the format:

```
Set-cookie: name=value; expires=date; path=path; domain=domain; secure;
```

The calendar format for the expiration date is:

```
@”%A, %d-%b-%Y %H:%M:%S GMT”
```

where all times are converted relative to Greenwich Mean Time.

This method is called by WOResponse when generating the response.

initWithName:value:path:domain:expires:isSecure:

– **initWithName:**(NSString *)*aName*
value:(NSString *)*aValue*
path:(NSString *)*aPath*
domain:(NSString *)*aDomain*
expires:(NSDate *)*expirationDate*
isSecure:(BOOL)*flag*

Initializes a cookie with all its attributes. For more information, see the descriptions of the methods that return attribute values.

See also: – **domain**, – **expires**, – **isSecure**, – **name**, – **path**, – **value**

isSecure

– (BOOL)**isSecure**

Returns the cookie's "secure" attribute. This attribute specifies whether the cookie should be transmitted only with secure HTTP. The default value is NO.

name

– (NSString *)**name**

Returns the cookie's "name" attribute. The name is similar to the key of a dictionary or hash table. Together, the name and value form the cookie's data.

path

– (NSString *)**path**

Returns the value of the cookie's "path" attribute. Cookies for a specific path are sent only when accessing URLs within that path. For more information on cookies and their paths, see Netscape's preliminary specification at **http://www.netscape.com/newsref/std/cookie_spec.html** and RFC 2109 - HTTP State Management Mechanism at **http://www.cis.ohio-state.edu/htbin/rfc/rfc2109.html**.

setDomain:

– (void)**setDomain:**(NSString *)*aDomain*

Sets the cookie's "domain" attribute to *aDomain*. For more information, see **domain**.

See also: – **cookieWithName:value:path:domain:expires:isSecure:**

setExpires:

– (void)**setExpires:**(NSDate *)*expirationDate*

Sets the cookie's "expires" attribute to *expirationDate*. For more information, see **expires**.

See also: – **cookieWithName:value:path:domain:expires:isSecure:**

setIsSecure:

– (void)**setIsSecure:**(BOOL)*flag*

Sets the cookie's "secure" attribute to *flag*. For more information, see **isSecure**.

See also: – **cookieWithName:value:path:domain:expires:isSecure:**

setName:

– (void)**setName:**(NSString *)*aName*

Sets the cookie's "name" attribute to *aName*. For more information, see ***name***.

See also: – **cookieWithName:value:path:domain:expires:isSecure:**, – **cookieWithName:value:**

setPath:

– (void)**setPath:**(NSString *)*aPath*

Sets the cookie's "path" attribute to *aPath*. For more information, see ***path***.

See also: – **cookieWithName:value:path:domain:expires:isSecure:**

setValue:

– (void)**setValue:**(NSString *)*aValue*

Sets the cookie's "value" attribute to *aValue*. For more information, see ***value***.

See also: – **cookieWithName:value:path:domain:expires:isSecure:**, – **cookieWithName:value:**

value

– (NSString *)**value**

Returns the value of the cookie's value attribute. This attribute is similar to the value of a dictionary or hash table. Together, the name and value form the cookie's data.

WODirectAction

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WODirectAction.h

Class Description

WODirectAction is an abstract class that defines the interface for direct action classes. You subclass WODirectAction to provide an object that is a repository for action methods.

WODirectAction provides the simplest interface for addig logic and custom code to your WebObjects application. WODirectAction objects are instantiated when a URL requested by a client browser is sent to your WebObjects application. The WODirectActionRequestHandler determines the proper class and action to be invoked and then passes control to your WODirectAction subclass.

In contrast to a WOComponent-based action, a direct action is well-defined by the URL that invokes it. For example, the following URL will invoke the method **findEmployeeAction** on the subclass of WODirectAtion called Common:

```
http://localhost/cgi-bin/WebObjects/Myapp.woa/wa/Common/findEmployee
```

A subclass of WODirectAction is a repository for action methods. New WebObjects applications contain a default implementation of the WODirectAction subclass called DirectAction. The DirectAction class is used when no class is specified in the URL.

In summary, here are some URLs and the actions they invoke:

This URL...	Invokes this method...
../MyApp.woa/wa/	**defaultAction** on class DirectAction
../MyApp.woa/wa/ find	**findAction** on classDirectAction , if it exists **defaultAction** on class find , otherwise
../MyApp.woa/wa/Common/find	**findAction** on class Common

WODirectActionRequestHandler invokes methods only on subclasses on WODirectAction. If the specified class or action doesn’t exist, WODirectActionRequestHandler throwsraises an exception.

Method Types

Creation	
	– initWithRequest:
Obtaining attributes	
	– request
Obtaining a session	
	– existingSession
	– session
Obtaining a page	
	– pageWithName:
Performing an action	
	– performActionNamed:
Value assignment	
	– takeFormValueArraysForKeyArray:
	– takeFormValueArraysForKeys:
	– takeFormValuesForKeyArray:
	– takeFormValuesForKeys:
Debugging	
	– debugWithFormat:
	– logWithFormat:

Instance Methods

debugWithFormat:

– (void)**debugWithFormat:**(NSString *)*aFormatString,...*

This method is similar to **logWithFormat:** except that you can control whether it displays output with the **WODebuggingEnabled** user default option. If **WODebuggingEnabled** is YES, then the **debugWithFormat:** messages display their output. If **WODebuggingEnabled** is NO, the **debugWithFormat:** messages don't display their output.

See also: – **debugWithFormat:** (– **WOApplication**)

existingSession

– (WOSession*)**existingSession**

Restores the session based on the request. If the request did not have a session ID or the session ID referred to a non-existent session, then this method returns **nil**. To determine if a session failed to restore, check the request's session ID to see if it non-**nil** and if so, call this method to check its result.

See also: – **session**

initWithRequest:

– **initWithRequest:**(WORequest *)*aRequest*

This is the designated initializer for all subclasses of WODirectAction. Whne you create a subclass, you must override this method to provide any additional initialization.

logWithFormat:

– (void)**logWithFormat:**(NSString *)*aFormatString,...*

Prints a message to the standard error device (stderr). The message can include formatted variable data using printf-style conversion specifiers, for example:

```
id i = 500;
id f = 2.045;
[self logWithFormat:@"Amount = %@, Rate = %@, Total = %@", i, f, i*f];
```

Note that in WebScript, all variables are objects, so the only conversion specifier allowed is **%@** as shown above. In compiled Objective-C code, all **printf** conversion specifiers are allowed. The equivalent method in Java is **logString**.

See also: – **logWithFormat:** (– **WOApplication**)

pageWithName:

– (WOComponent *)**pageWithName:**(NSString *)*aComponentName*

Returns the WOComponent with the specified name.

performActionNamed:

– (id <WOActionResults>)**performActionNamed:**(NSString *)*anActionName*

Performs the action with the specified name and returns the result of that action. The default implementation appends "Action" to *anActionName* and tries to invoke resulting method name. Override this method to change how actions are dispatched.

request

– (WORequest *)**request**

Returns the WORequest object that initiated the action.

session

– (WOSession *)**session**

Returns the current session. If there is no session, this method first tries to restore the session that the request's session ID refers to. If the request has no session ID—which is a possibility if the application is written entirely with direct actions—this method creates a new session and returns it. If the session ID refers to a session that doesn't exist or cannot be restored, this method raises an exception.

See also: – **existingSession**

takeFormValueArraysForKeyArray:

– (void)**takeFormValueArraysForKeyArray:**(NSArray *)*aKeyArray*

Performs **takeValue:forKey:** on each key in *aKeyArray* using values from the receiver's request.

This method uses an NSArray for each form value. This is useful when a user can select multiple items for a form value, such as a WOBrowser. If a form value contains only one item, this method uses an NSArray with one object. To use single objects as form values, use **takeFormValuesForKeyArray:**.

See also: **takeFormValueArraysForKeys:**

takeFormValueArraysForKeys:

– (void)**takeFormValueArraysForKeys:**(NSString *)*aFirstKey,...*

Performs **takeValue:forKey:** on the specified keys using values from the receiver's request. The last key must be nil.

This method uses an NSArray for each form value. This is useful when a user can select multiple items for a form value, such as a WOBrowser. If a form value contains only one item, this method uses an NSArray with one object. To use single objects as form values, use **takeFormValuesForKeys:**.

See also: **takeFormValueArraysForKeyArray:**

takeFormValuesForKeyArray:

– (void)**takeFormValuesForKeyArray:**(NSArray *)*aKeyArray*

Performs **takeValue:forKey:** on the each key in *aKeyArray* using values from the receiver's request.

This method uses an a single object for each form value. If a form value contains more than one item, such as a WOBrowser, this method uses the first item in the array. To use arrays of objects as form values, use **takeFormValueArraysForKeyArray:**.

See also: **takeFormValuesForKeys:**

takeFormValuesForKeys:

– (void)**takeFormValuesForKeys:**(NSString *)*aFirstKey,...*

Performs **takeValue:forKey:** on the specified keys using values from the receiver's request. The last key must be nil.

This method uses an a single object for each form value. If a form value contains more than one item, such as a WOBrowser, this method uses the first item in the array. To use arrays of objects as form values, use **takeFormValueArraysForKeys:**.

See also: **takeFormValuesForKeyArray:**

WODisplayGroup

Inherits From:	NSObject
Conforms To:	NSCoding NSObject (NSObject)
Declared In:	WebObjects/WODisplayGroup.h

Class Description

A WODisplayGroup is the basic user interface manager for a WebObjects application that accesses a database. It collects objects from an EODataSource (defined in EOControl), filters and sorts them, and maintains a selection in the filtered subset. You bind WebObjects dynamic elements to WODisplayGroup attributes and methods to display information from the database on your web page.

A WODisplayGroup manipulates its EODataSource by sending it **fetchObjects**, **insertObject:**, and other messages, and registers itself as an editor and message handler of the EODataSource's EOEditingContext (also defined in EOControl). The EOEditingContext then monitors the WODisplayGroup for changes to objects.

Most of a WODisplayGroup's interactions are with its EODataSource and its EOEditingContext. See the EODataSource, and EOEditingContext class specifications in the *Enterprise Objects Framework Reference* for more information on these interactions.

The Delegate

The WODisplayGroup delegate offers a number of methods, and WODisplayGroup invokes them as appropriate. Besides **displayGroup:displayArrayForObjects:**, there are methods that inform the delegate that the WODisplayGroup has fetched, created an object (or failed to create one), inserted or deleted an object, changed the selection, or set a value for a property. There are also methods that request permission from the delegate to perform most of these same actions. The delegate can return YES to permit the action or NO to deny it. See each method's description in the WODisplayGroup.Delegates protocol specification for more information.

Adopted Protocols

NSCoding	- encodeWithCoder: - initWithCoder:

Method Types

Creating instances

- – init

Configuring behavior

- – setFetchesOnLoad:
- – fetchesOnLoad
- – setSelectsFirstObjectAfterFetch:
- – selectsFirstObjectAfterFetch
- – setValidatesChangesImmediately:
- – validatesChangesImmediately

Setting the data source

- – setDataSource:
- – dataSource

Setting the qualifier and sort ordering

- – setQualifier:
- – qualifier
- – setSortOrderings:
- – sortOrderings

Managing queries

- – qualifierFromQueryValues
- – queryMatch
- – queryMax
- – queryMin
- – queryOperator
- – allQualifierOperators
- – relationalQualifierOperators
- – setDefaultStringMatchFormat:
- – defaultStringMatchFormat
- – setDefaultStringMatchOperator:
- – defaultStringMatchOperator
- – qualifyDisplayGroup
- – qualifyDataSource
- – inQueryMode
- – setInQueryMode:

Fetching objects from the data source

- – fetch

Getting the objects

- – allObjects
- – displayedObjects

Batching the results	
	– setNumberOfObjectsPerBatch:
	– numberOfObjectsPerBatch
	– hasMultipleBatches
	– displayNextBatch
	– displayPreviousBatch
	– batchCount
	– setCurrentBatchIndex:
	– currentBatchIndex
	– indexOfFirstDisplayedObject
	– indexOfLastDisplayedObject
	– displayBatchContainingSelectedObject
Updating display of values	
	– redisplay
	– updateDisplayedObjects
Setting the objects	
	– setObjectArray:
Changing the selection	
	– setSelectionIndexes:
	– selectObjectsIdenticalTo:
	– selectObjectsIdenticalTo:selectFirstOnNoMatch:
	– selectObject:
	– clearSelection
	– selectNext
	– selectPrevious
Examining the selection	
	– selectionIndexes
	– selectedObject
	– selectedObjects
Inserting and deleting objects	
	– insertObject: atIndex:
	– insertObjectAtIndex:
	– insert
	– setInsertedObjectDefaultValues:
	– insertedObjectDefaultValues
	– deleteObjectAtIndex:
	– deleteSelection
	– delete

Setting up a detail display group	
	– hasDetailDataSource
	– setMasterObject:
	– masterObject
	– setDetailKey:
	– detailKey
Working with named fetch specifications	
	– queryBindings
Setting the delegate	
	– setDelegate:
	– delegate

Instance Methods

allObjects

– (NSArray *)**allObjects**

Returns all of the objects collected by the receiver.

See also: – **displayedObjects**, – **fetch**

allQualifierOperators

– (NSArray *)**allQualifierOperators**

Returns an array containing all of the relational operators supported by EOControl's EOQualifier: =, !=, <, <=, >, >=, "**like**" and "**caseInsensitiveLike**".

See also: – **queryOperator**, – **relationalQualifierOperators**

batchCount

– (unsigned)**batchCount**

The number of batches to display. For example, if the displayed objects array contains two hundred records and the batch size is ten, **batchCount** returns twenty (twenty batches of ten records each).

See also: – **currentBatchIndex**, – **displayNextBatch**, – **displayPreviousBatch**, – **hasMultipleBatches**, – **numberOfObjectsPerBatch**

clearSelection

– (BOOL)**clearSelection**

Invokes **setSelectionIndexes:** to clear the selection, returning YES on success and NO on failure.

currentBatchIndex

– (unsigned)**currentBatchIndex**

Returns the index of the batch currently being displayed. The total batch count equals the number of displayed objects divided by the batch size. For example, if the WODisplayGroup has one hundred objects to display and the batch size is twenty, there are five batches. The first batch has a batch index of 1.

See also: – **batchCount**, – **numberOfObjectsPerBatch**, – **setCurrentBatchIndex:**

dataSource

– (EODataSource *)**dataSource**

Returns the receiver's EODataSource (defined in the EOControl framework).

See also: – **hasDetailDataSource**, – **setDataSource:**

defaultStringMatchFormat

– (NSString *)**defaultStringMatchFormat**

Returns the format string that specifies how pattern matching will be performed on string values in the **queryMatch** dictionary. If a key in the **queryMatch** dictionary does not have an associated operator in the **queryOperator** dictionary, then its value is matched using pattern matching, and the format string returned by this method specifies how it will be matched.

See also: – **defaultStringMatchOperator**, – **setDefaultStringMatchFormat:**

defaultStringMatchOperator

– (NSString *)**defaultStringMatchOperator**

Returns the operator used to perform pattern matching for string values in the **queryMatch** dictionary. If a key in the **queryMatch** dictionary does not have an associated operator in the **queryOperator** dictionary, then the operator returned by this method is used to perform pattern matching. Unless the default is changed, this method returns caseInsensitiveLike.

See also: – **defaultStringMatchFormat**, – **setDefaultStringMatchOperator:**

delegate

– (id)**delegate**

Returns the receiver's delegate.

See also: – **setDelegate:**

delete

– (id)**delete**

Uses **deleteSelection** to attempt to delete the selected objects and then causes the page to reload. Returns **nil** to force reloading of the web page.

See also: – **deleteObjectAtIndex:**

deleteObjectAtIndex:

– (BOOL)**deleteObjectAtIndex:**(unsigned)*index*

Attempts to delete the object at *index*, returning YES if successful and NO if not. Checks with the delegate using the method **displayGroup:shouldDeleteObject:**. If the delegate returns NO, this method fails and returns NO. If successful, it sends the delegate a **displayGroup:didDeleteObject:** message.

This method performs the delete by sending **deleteObject** to the EODataSource (defined in the EOControl framework). If that message raises an exception, this method fails and returns NO.

See also: – **delete**, – **deleteSelection**

deleteSelection

– (BOOL)**deleteSelection**

Attempts to delete the selected objects, returning YES if successful and NO if not.

See also: – **delete**, – **deleteObjectAtIndex:**

detailKey

– (NSString *)**detailKey**

For detail display groups, returns the key to the master object that specifies what this detail display group represents. That is, if you send the object returned by the **masterObject** method a **valueForKey:** message with this key, you obtain the objects controlled by this display group.

This method returns **nil** if the receiver is not a detail display group or if the detail key has not yet been set. You typically create a detail display group by dragging a to-many relationship from EOModeler to an open component in WebObjects Builder.

See also: – **hasDetailDataSource**, – **masterObject**, – **setDetailKey:**

displayBatchContainingSelectedObject

– (id)**displayBatchContainingSelectedObject**

Displays the batch containing the selection and sets the current batch index to that batch's index. Returns **nil** to force the page to reload.

See also: : – **displayNextBatch**, – **displayPreviousBatch**, – **setCurrentBatchIndex:**

displayedObjects

– (NSArray *)**displayedObjects**

Returns the objects that should be displayed or otherwise made available to the user, as filtered by the receiver's delegate, by the receiver's qualifier and sort ordering.

If batching is in effect, **displayedObjects** returns the current batch of objects.

See also: – **allObjects**, – **updateDisplayedObjects**, – **qualifier**, – **setSortOrderings:**, – **displayGroup: displayArrayForObjects:** (delegate method)

displayNextBatch

– (id)**displayNextBatch**

Increments the current batch index, displays that batch of objects, and clears the selection. If the batch currently being displayed is the last batch, this method displays the first batch of objects. Returns **nil** to force the page to reload.

See also: – **batchCount**, – **currentBatchIndex**, – **displayBatchContainingSelectedObject**, – **displayPreviousBatch**

displayPreviousBatch

– (id)**displayPreviousBatch**

Decrements the current batch index, displays that batch of objects, and clears the selection. If the batch currently being displayed is the first batch, this method displays the last batch of objects. Returns **nil** to force the page to reload.

See also: – **batchCount**, – **currentBatchIndex**, – **displayBatchContainingSelectedObject**, – **displayNextBatch**

fetch

– (id)**fetch**

Attempts to fetch objects from the EODataSource (defined in the EOControl framework).

Before fetching, this method sends **displayGroupShouldFetch:** to the delegate. If this method was successful, it then sends a **fetchObjects** message to the receiver's EODataSource to replace the object array, and if successful sends the delegate a **displayGroup:didFetchObjects:** message.

This method returns **nil** to force the page to reload.

See also: – **allObjects**, – **updateDisplayedObjects**

fetchesOnLoad

– (BOOL)**fetchesOnLoad**

Returns YES if the receiver fetches automatically after the component that contains it is loaded, NO if it must be told explicitly to fetch. The default is YES. You can set this behavior in WebObjects Builder using the Display Group Options panel. Note that if the display group fetches on load, it performs the fetch each time the component is loaded into the web browser.

See also: – **fetch**, – **setFetchesOnLoad:**

hasDetailDataSource

– (BOOL)**hasDetailDataSource**

Returns YES if the display group's data source is an EODetailDataSource (defined in the EOControl framework), and NO otherwise. If you drag a to-many relationship from EOModeler to an open component in WebObjects Builder, you create a display group that has an EODetailDataSource. You can also set this up using the Display Group Options panel in WebObjects Builder.

See also: – **detailKey**, – **masterObject**

hasMultipleBatches

– (BOOL)**hasMultipleBatches**

Returns YES if the batch count is greater than 1. A display group displays its objects in batches if the **numberOfObjectsPerBatch** method returns a number that is less than the number of objects in the **displayedObjects** array.

See also: – **batchCount**, – **setNumberOfObjectsPerBatch:**

indexOfFirstDisplayedObject

– (unsigned)**indexOfFirstDisplayedObject**

Returns the index of the first object displayed by the current batch. For example, if the current batch is displaying items 11 through 20, this method returns 11.

See also: – **indexOfLastDisplayedObject**

indexOfLastDisplayedObject

– (unsigned)**indexOfLastDisplayedObject**

Returns the index of the last object display by the current batch. For example, if the current batch is displaying items 11 through 20, this method returns 20.

See also: – **indexOfFirstDisplayedObject**

init

– **init**

Initializes the WODisplayGroup. The WODisplayGroup then needs to have an EODataSource set with **setDataSource:**.

inQueryMode

– (BOOL)**inQueryMode**

Returns YES to indicate that the receiver is in query mode, NO otherwise. In query mode, controls in the user interface that normally display values become empty, allowing users to type queries directly into them (this is also known as a “Query by Example” interface). In effect, the receiver’s “displayedObjects” are replaced with an empty **queryMatch** dictionary. When **qualifyDisplayGroup** or **qualifyDataSource** is subsequently invoked, the query is performed and the display reverts to displaying values—this time, the objects returned by the query.

See also: – **setInQueryMode:**

insert

– (id)**insert**

Invokes **insertObjectAtIndex:** with an index just past the first index in the selection, or at the end if there's no selection.

This method returns **nil** to force the page to reload.

insertedObjectDefaultValues

– (NSDictionary *)**insertedObjectDefaultValues**

Returns the default values to be used for newly inserted objects. The keys into the dictionary are the properties of the entity that the display group manages. If the dictionary returned by this method is empty, the **insert** method adds an object that is initially empty. Because the object is empty, the display group has no value to display on the HTML page for that object, meaning that there is nothing for the user to select and modify. Use the **setInsertedObjectDefaultValues:** method to set up a default value so that there is something to display on the page.

insertObjectAtIndex:

– (id)**insertObjectAtIndex:**(unsigned)*index*

Asks the receiver's EODataSource (defined in the EOControl framework) to create a new object by sending it a **createObject** message, then inserts the new object using **insertObject: atIndex:**. If a new object can't be created, this method sends the delegate a **displayGroup:createObjectFailedForDataSource:** message.

If the object is successfully created, this method then sets the default values specified by **insertedObjectDefaultValues**.

See also: – **insert**

insertObject: atIndex:

– (void)**insertObject:***anObject* **atIndex:**(unsigned)*index*

Inserts *anObject* into the receiver's EODataSource and displayed objects at *index*, if possible. This method checks with the delegate before actually inserting, using **displayGroup:shouldInsertObject:atIndex:**. If the delegate refuses, *anObject* isn't inserted. After successfully inserting the object, this method informs the delegate with a **displayGroup:didInsertObject:** message, and selects the newly inserted object.

Raises an NSRangeException if index is out of bounds.

See also: – **insertObjectAtIndex:**, – **insert**

masterObject

– (id)**masterObject**

Returns the master object for a detail display group (a display group that represents a detail in a master-detail relationship). A detail display group is one that uses an EODetailDataSource (defined in the EOControl framework). You create a detail display group by dragging a to-many relationship from EOModeler to an open component in WebObjects Builder. If the display group is not a detail display group or does not have a master object set, this method returns **nil**.

See also: – **detailKey**, – **hasDetailDataSource**, – **setMasterObject:**

numberOfObjectsPerBatch

– (unsigned)**numberOfObjectsPerBatch**

Returns the batch size. You can set the batch size using **setNumberOfObjectsPerBatch:** or using WebObjects Builder's Display Group Options panel.

qualifier

– (EOQualifier *)**qualifier**

Returns the receiver's qualifier, which it uses to filter its array of objects for display when the delegate doesn't do so itself.

See also: – **displayedObjects**, – **setQualifier:**:,– **updateDisplayedObjects**

qualifierFromQueryValues

– (EOQualifier *)**qualifierFromQueryValues**

Builds a qualifier constructed from entries in these query dictionaries: **queryMatch**, **queryMax**, **queryMin**, and **queryOperator**.

See also: – **qualifyDataSource**, – **qualifyDisplayGroup**

qualifyDataSource

– (void)**qualifyDataSource**

Takes the result of **qualifierFromQueryValues** and applies to the receiver's data source. The receiver then sends itself a **fetch** message. If the receiver is in query mode, query mode is exited. This method differs from **qualifyDisplayGroup** as follows: whereas **qualifyDisplayGroup** performs in-memory filtering of already fetched objects, **qualifyDataSource** triggers a new qualified fetch against the database.

See also: – **queryMatch**, – **queryMax**,, – **queryMin**,– **queryOperator**

qualifyDisplayGroup

– (void)**qualifyDisplayGroup**

Takes the result of the **qualifierFromQueryValues** and applies to the receiver using **setQualifier:**. The method **updateDisplayedObjects** is invoked to refresh the display. If the receiver is in query mode, query mode is exited.

See also: – **qualifyDataSource**, – **queryMatch**, – **queryMax**, -– **queryMin**, – **queryOperator**

queryBindings

– (NSMutableDictionary *)**queryBindings**

Returns a dictionary containing the actual values that the user wants to query upon. You use this method to perform a query stored in the model file. Bind keys in this dictionary to elements on your component that specify query values, then pass this dictionary to the fetch specification that performs the fetch.

queryMatch

– (NSMutableDictionary *)**queryMatch**

Returns a dictionary of query values to match. The **qualifierFromQueryValues** method uses this dictionary along with the **queryMax** and **queryMin** dictionaries to construct qualifiers.

Use the **queryOperator** dictionary to specify the type of matching (=, <, >, **like**, and so on) for each key in the **queryMatch** dictionary.

If the **queryOperator** dictionary does not contain a key contained in the **queryMatch** dictionary, the default is to match the value exactly (=) if the value is a number or a date and to perform pattern matching if the value is an NSString. In the case of string values, the **defaultStringMatchFormat** and **defaultStringMatchOperator** specify exactly how the pattern matching will be performed.

See also: – **allQualifierOperators**, – **qualifyDataSource**, – **qualifyDisplayGroup**,
– **relationalQualifierOperators**

queryMax

– (NSMutableDictionary *)**queryMax**

Returns a dictionary of “less than” query values. The **qualifierFromQueryValues** method uses this dictionary along with the **queryMatch** and **queryMin** dictionaries to construct qualifiers.

See also: – **qualifyDataSource**, – **qualifyDisplayGroup**, – **queryOperator**

queryMin

– (NSMutableDictionary *)**queryMin**

Returns a dictionary of “greater than” query values. The **qualifierFromQueryValues** method uses this dictionary along with the **queryMatch** and **queryMin** dictionaries to construct qualifiers.

See also: – **qualifyDataSource**, – **qualifyDisplayGroup**, – **queryOperator**

queryOperator

– (NSMutableDictionary *)**queryOperator**

Returns a dictionary of operators to use on items in the **queryMatch** dictionary. If a key in the **queryMatch** dictionary also exists in **queryOperator**, that operator for that key is used. The **allQualifierOperators** method returns the operator strings you can use as values in this dictionary.

See also: – **qualifierFromQueryValues**, – **queryMax**, – **queryMin**, – **relationalQualifierOperators**

redisplay

– (void)**redisplay**

Sends out a contents changed notification.

relationalQualifierOperators

– (NSArray *)**relationalQualifierOperators**

Returns an array containing all of the relational operators supported by EOControl’s EOQualifier: =, !=, <, <=, >, and >=. In other words, returns all of the EOQualifier operators except for the ones that work exclusively on strings: “**like**” and “**caseInsensitiveLike**”.

See also: – **allQualifierOperators**, – **queryOperator**

selectedObject

– (id)**selectedObject**

Returns the first selected object in the displayed objects array, or **nil** if there's no such object.

See also: – **displayedObjects**, – **selectionIndexes**, – **selectedObjects**

selectedObjects

– (NSArray *)**selectedObjects**

Returns the objects selected in the receiver's displayed objects array.

See also: – **displayedObjects**, – **selectionIndexes**, – **selectedObject**

selectionIndexes

– (NSArray *)**selectionIndexes**

Returns the selection as an array of NSNumbers. The NSNumbers are indexes into the array returned by **displayedObjects**.

See also: – **selectedObject**, – **selectedObjects**, – **setSelectionIndexes:**

selectNext

– (id)**selectNext**

Attempts to select the object just after the currently selected one. The selection is altered in this way:

- If there are no objects, does nothing.
- If there's no selection, selects the object at index zero.
- If the first selected object is the last object in the displayed objects array, selects the first object.
- Otherwise selects the object after the first selected object.

This method returns **nil** to force the page to reload.

See also: – **selectPrevious**, – **setSelectionIndexes:**

selectObject:

– (BOOL)**selectObject:**(id)*anObject*

Attempts to select the object equal to *anObject* in the receiver’s displayed objects array, returning YES if successful and NO otherwise. *anObject* is equal to an object in the displayed objects array if its address is the same as the object in the array.

See also: – **selectNext**, – **selectPrevious**

selectObjectsIdenticalTo:

– (BOOL)**selectObjectsIdenticalTo:**(NSArray *)*objectSelection*

Attempts to select the objects in the receiver’s displayed objects array whose **id**s are equal to those of objects, returning YES if successful and NO otherwise.

See also: – **setSelectionIndexes:**, – **selectObjectsIdenticalTo:selectFirstOnNoMatch:**

selectObjectsIdenticalTo:selectFirstOnNoMatch:

– (BOOL)**selectObjectsIdenticalTo:**(NSArray *)*objects* **selectFirstOnNoMatch:**(BOOL)*flag*

Selects the objects in the receiver’s displayed objects array whose **id**s are equal to those of *objects*, returning YES if successful and NO otherwise. If no objects in the displayed objects array match objects and *flag* is YES, attempts to select the first object in the displayed objects array.

See also: – **setSelectionIndexes:**, – **selectObjectsIdenticalTo:**

selectPrevious

– (id)**selectPrevious**

Attempts to select the object just before the presently selected one. The selection is altered in this way:

- If there are no objects, does nothing.
- If there’s no selection, selects the object at index zero.
- If the first selected object is at index zero, selects the last object.
- Otherwise selects the object before the first selected object.

This method returns **nil** to force the page to reload.

See also: – **selectNext**, – **redisplay**

selectsFirstObjectAfterFetch

– (BOOL)**selectsFirstObjectAfterFetch**

Returns YES YES if the receiver automatically selects its first displayed object after a fetch if there was no selection, NO if it leaves an empty selection as-is.

WODisplayGroups by default do select the first object after a fetch when there was no previous selection.

See also: – **displayedObjects**, – **fetch**, – **setSelectsFirstObjectAfterFetch:**

setCurrentBatchIndex:

– (void)**setCurrentBatchIndex:**(unsigned)*anInt*

Displays the *anInt* batch of objects. The total batch count equals the number of displayed objects divided by the batch size. For example, if the WODisplayGroup has one hundred objects to display and the batch size is twenty, there are five batches. The first batch has a batch index of 1. **setCurrentBatchIndex:3** would display the third batch of objects (objects 41 to 60 in this example).

If *anInt* is greater than the number of batches, this method displays the first batch.

See also: – **batchCount**, – **currentBatchIndex**,
– **displayBatchContainingSelectedObject**,– **displayNextBatch**,
– **displayPreviousBatch**,– **numberOfObjectsPerBatch**

setDataSource:

– (void)**setDataSource:**(EODataSource *)*aDataSource*

Sets the receiver's EODataSource (defined in the EOControl framework) to *aDataSource*. In the process, it performs these actions:

Unregisters itself as an editor and message handler for the previous EODataSource's EOEditingContext (also defined in EOControl), if necessary, and registers itself with *aDataSource*'s EOEditingContext. If the new EOEditingContext already has a message handler, however, the receiver doesn't assume that role.

Clears the receiver's array of objects.

Sends **displayGroupDidChangeDataSource:** to the delegate if there is one.

See also: – **dataSource**

setDefaultStringMatchFormat:

– (void)**setDefaultStringMatchFormat:**(NSString *)*format*

Sets how pattern matching will be performed on NSString values in the **queryMatch** dictionary. This format is used for properties listed in the **queryMatch** dictionary that have NSString values and that do not have an associated entry in the **queryOperator** dictionary. In these cases, the value is matched using pattern matching and *format* specifies how it will be matched.

The default format string for pattern matching is "**%@***" which means that the string value in the **queryMatch** dictionary is used as a prefix. For example, if the **queryMatch** dictionary contains a value "Jo" for the key "Name", the query returns all records whose name values begin with "Jo".

See also: – **defaultStringMatchFormat**, – **setDefaultStringMatchOperator:**

setDefaultStringMatchOperator:

– (void)**setDefaultStringMatchOperator:**(NSString *)*operator*

Sets the operator used to perform pattern matching for NSString values in the **queryMatch** dictionary. This operator is used for properties listed in the **queryMatch** dictionary that have NSString values and that do not have an associated entry in the **queryOperator** dictionary. In these cases, the operator *operator* is used to perform pattern matching.

The default value for the query match operator is **caseInsensitiveLike**, which means that the query does not consider case when matching letters. The other possible value for this operator is **like**, which matches the case of the letters exactly.

See also: – **allQualifierOperators**, – **defaultStringMatchOperator**, – **relationalQualifierOperators**, – **setDefaultStringMatchFormat:**

setDelegate:

– (void)**setDelegate:**(id)*anObject*

Sets the receiver's delegate to *anObject*, without retaining it.

See also: – **delegate**, WODisplayGroupDelegate

setDetailKey:

– (void)**setDetailKey:**(NSString *)*detailKey*

Sets the detail key to *detailKey* for a detail display group. The detail key is the key that retrieves from the master object the objects that this display group manages. You must set a detail key before you set a master object.

If the receiver is not a detail display group, this method has no effect. A display group is a detail display group if its data source is an EODetailDataSource (defined in the EOControl framework). You typically create a detail display group by dragging a to-many relationship from EOModeler to an open component in WebObjects Builder. Doing so sets the detail key and master object, so you rarely need to use this method.

See also: – **hasDetailDataSource**, – **detailKey**, – **setMasterObject:**

setFetchesOnLoad:

– (void)**setFetchesOnLoad:**(BOOL)*flag*

Controls whether the receiver automatically fetches its objects after being loaded. If *flag* is YES it does; if *flag* is NO the receiver must be told explicitly to fetch. The default is NO. You can also set this behavior in WebObjects Builder in the Display Group Options panel.

See also: – **fetch**, – **fetchesOnLoad**

setInQueryMode:

– (void)**setInQueryMode:**(BOOL)*flag*

Sets according to *flag* whether the receiver is in query mode. In query mode, controls in the user interface that normally display values become empty, allowing users to type queries directly into them (this is also known as a “Query by Example” interface). In effect, the receiver’s “displayedObjects” are replaced with an empty **queryMatch** dictionary. When **qualifyDisplayGroup** or **qualifyDataSource** is subsequently invoked, the query is performed and the display reverts to displaying values—this time, the objects returned by the query.

See also: – **inQueryMode**

setInsertedObjectDefaultValues:

– (void)**setInsertedObjectDefaultValues:**(NSDictionary *)*defaultValues*

Sets default values to be used for newly inserted objects. When you use the **insert** method to add an object, that object is initially empty. Because the object is empty, there is no value to be displayed on the HTML page, meaning there is nothing for the user to select and modify. You use this method to provide at least one field that can be displayed for the newly inserted object. The possible keys into the dictionary are the properties of the entity managed by this display group. For example, a component that displays a list of movie titles and allows the user to insert new movie titles might contain these statements to ensure that all new objects have something to display as a movie title:

```
[defaultValues setObject:@"New title" forKey:@"title"];
```

```
[movies setInsertedObjectDefaultValues:defaultValues];
```

See also: – **insertedObjectDefaultValues**

setMasterObject:

– (void)**setMasterObject:**(id)*masterObject*

Sets the master object to *masterObject* for detail display groups and then performs a fetch if the display group is set to fetch on load. The master object owns the objects controlled by this display group.

Before you use this method, you should use the **setDetailKey:** to set the key to this relationship. You typically create a detail display group by dragging a to-Many relationship from EOModeler to an open component in WebObjects Builder. Doing so sets the master object and detail key, so you typically do not have to use this method.

If the receiver is not a detail display group, this method has no effect.

See also: – **hasDetailDataSource**, – **masterObject**

setNumberOfObjectsPerBatch:

– (void)**setNumberOfObjectsPerBatch:**(unsigned)*count*

Sets the number of objects the receiver displays at a time. For example, suppose you are displaying one hundred records. Instead of displaying all of these at once, you can set the batch size so that the page displays a more manageable number (for example, 10). WebObjects Builder allows you to set the number of objects per batch on the Display Group Options panel.

See also: – **batchCount**, – **displayNextBatch**, – **displayPreviousBatch**, – **numberOfObjectsPerBatch**

setObjectArray:

– (void)**setObjectArray:**(NSArray *)*objects*

Sets the receiver's objects to *objects*, regardless of what its EODataSource (defined in the EOControl framework) provides. This method doesn't affect the EODataSource's objects at all; specifically, it results in neither inserts nor deletes of objects in the EODataSource. *objects* should contain objects with the same property names or methods as those accessed by the receiver. This method is used by **fetch** to set the array of fetched objects; you should rarely need to invoke it directly.

After setting the object array, this method restores as much of the original selection as possible. If there's no match and the receiver selects after fetching, then the first object is selected.

See also: – **allObjects**, – **displayedObjects**, – **fetch**, – **selectsFirstObjectAfterFetch**

setQualifier:

– (void)**setQualifier:**(EOQualifier *)*aQualifier*

Sets the receiver's qualifier to *aQualifier*. This qualifier is used to filter the receiver's array of objects for display. Use **updateDisplayedObjects** to apply the qualifier.

If the receiver's delegate responds to **displayGroup:displayArrayForObjects:**, that method is used instead of the qualifier to filter the objects.

See also: – **displayedObjects**, – **qualifier**

setSelectionIndexes:

– (BOOL)**setSelectionIndexes:**(NSArray *)*selection*

Selects the objects at *selection* in the receiver's array if possible, returning YES if successful and NO if not (in which case the selection remains unaltered). This method is the primitive method for altering the selection; all other such methods invoke this one to make the change.

This method checks the delegate with a **displayGroup:shouldChangeSelectionToIndexes:** message. If the delegate returns NO, this method also fails and returns NO. If the receiver successfully changes the selection, its observers each receive a **displayGroupDidChangeSelection:** message and, if necessary, a **displayGroupDidChangeSelectedObjects:** message.

Note: The selection set here is only a programmatic selection; the objects on the screen are not highlighted in any way.

See also: – **allObjects**

setSelectsFirstObjectAfterFetch:

– (void)**setSelectsFirstObjectAfterFetch:**(BOOL)*flag*

Controls whether the receiver automatically selects its first displayed object after a fetch when there were no selected objects before the fetch. If *flag* is YES it does; if *flag* is NO then no objects are selected.

WODisplayGroups by default do select the first object after a fetch when there was no previous selection.

See also: – **displayedObjects**, – **fetch**, – **selectsFirstObjectAfterFetch**

setSortOrderings:

– (void)**setSortOrderings:**(NSArray *)*keySortOrderArray*

Sets the EOSortOrdering objects (defined in the EOControl framework) that **updateDisplayedObjects** uses to sort the displayed objects to orderings. Use **updateDisplayedObjects** to apply the sort orderings.You can also set this value using the WebObjects Builder Display Group Options panel.

If the receiver's delegate responds to **displayGroup:displayArrayForObjects:**, that method is used instead of the sort orderings to order the objects.

See also: – **displayedObjects**, – **sortOrderings**, – **updateDisplayedObjects**

setValidatesChangesImmediately:

– (void)**setValidatesChangesImmediately:**(BOOL)*flag*

Controls the receiver's behavior on encountering a validation error. In the Web context, this method has no effect.

WODisplayGroups by default don't validate changes immediately.

See also: – **saveChanges** (in EOControl's EOEditingContext), **-** tryToSaveChanges (EOEditingContext Additions), – **validatesChangesImmediately**

sortOrderings

– (NSArray *)**sortOrderings**

Returns an array of EOSortOrdering objects (defined in the EOControl framework) that **updateDisplayedObjects** uses to sort the displayed objects, as returned by the **displayedObjects** method.

See also: – **setSortOrderings:**

updateDisplayedObjects

– (void)**updateDisplayedObjects**

Recalculates the receiver's displayed objects arrays and redisplays. If the delegate responds to **displayGroup:displayArrayForObjects:**, it's sent this message and the returned array is set as the WODisplayGroup's displayed objects. Otherwise, the receiver applies its qualifier and sort ordering to its array of objects. In either case, any objects that were selected before remain selected in the new displayed object's array.

See also: – **redisplay**, – **allObjects**, – **displayedObjects**, – **qualifier**, – **selectedObjects**, – **sortOrderings**

validatesChangesImmediately

– (BOOL)**validatesChangesImmediately**

Returns YES if the receiver immediately handles validation errors, or leaves them for the EOEditingContext (defined in the EOControl framework) to handle when saving changes.

By default, WODisplayGroups don't validate changes immediately.

See also: – **setValidatesChangesImmediately:**

WODynamicElement

Inherits From: WOElement : NSObject

Declared In: WebObjects/WODynamicElement.h

Class Description

WODynamicElement is an abstract superclass for classes that generate dynamic elements: objects representing HTML or PDF elements whose values can programmatically change at run time. Dynamic elements have a name and one or more *properties*, instance variables holding such things as user-entered data or user-triggerable actions. The properties of a dynamic element are associated with, or "bound" to, the properties of the WOComponent object that represents the page (or portion of a page) in which the dynamic element appears.

At runtime, a dynamic element can extract values from the request, feed those values across the bindings to the owning component, receive back new data, and include that data in the next representation of the page. A dynamic element can also detect if the user has manipulated it (for instance, clicking a button) to signal some intention and then trigger the appropriate action method in the owning WOComponent. The bindings between properties of a dynamic element and properties of a WOComponent are made possible by *associations*, objects that know how to "push" and "pull" values to and from another object using keys. All objects that inherit from NextObject have associative capabilities through NextObjects's implementation of the KeyValueCoding protocol.

WODynamicElements must implement the default initializer to initialize their instance variables with the appropriate association objects (passed in). As WOElement objects, they must also implement one or more of the three request-handling methods. In the context of request handling, a dynamic element can use its associations to:

- Push request values into the associated properties of their WOComponent (**takeValuesFromRequest: inContext:**)
- Invoke action methods of the WOComponent (**invokeActionForRequest:inContext:**)
- Extract values from the WOComponent when composing a dynamic HTML response (**appendToResponse:inContext:**)

All dynamic elements must implement **appendToResponse:inContext:**. If they accept user input or respond to user actions (such as mouse clicks), they should implement **takeValuesFromRequest: inContext:** and **invokeActionForRequest:inContext:**, respectively.

Note: If you write a dynamic element that appends content to the response (this is typically done by overriding **appendToResponse:inContext:**), be sure to verify that the request is not client-side:

```
- (void)appendToResponse:(WOResponse *)response inContext:(WOContext *)context {
        if(![[context request]] isFromClientComponent]){
            // append content here
        }
    }
```

Dynamic elements do not know about their WOComponent object until run time. During request-handling, the application stores components (representing a page and subcomponents on the page) on a stack maintained by the WOContext object, with the currently referenced WOComponent on top of the stack. A dynamic element's WOAssociation retrieves the current WOComponent (through an invocation of WOContext's **component** method) and reads and writes values from and to the WOComponent using KeyValueCoding methods.

A dynamic element can represent a single HTML or PDF element (such as an editable text field) or a compound element, such as the LoginPanel whose implementation is described below. WebObjects includes a suite of ready-made dynamic elements and the WebObjects Builder application makes these objects available on its palettes. The *Dynamic Elements Reference* describes WebObjects' dynamic elements and provides examples showing how to use them.

Method Types

Creation

– **initWithName:associations:template:**

WODynamicElement

Instance Methods

initWithName:associations:template:

– (id)**initWithName:**(NSString *)*aName* **associations:**(NSDictionary *)*someAssociations* **template:** (WOElement *)*anElement*

Returns a dynamic element identified by class *aName* and initialized with the objects in dictionary *someAssociations*. The dictionary contains WOAssociation objects, which know how to take values from, and set values in, an "owning" WOComponent. To properly initialize a dynamic element, you should use the published keys of the dynamic element to get the associations that belong to the dynamic element; then assign these objects to instance variables. The *anElement* argument, if not **nil**, is the root object of a graph of WOElements associated with the dynamic element.

Typically, a key in the *someAssociations* dictionary is identified with a property of the element, and the value of this key is the name of a property of the associated Component. For example, the value of key "userName" might be bound to "employee.name" in the WOComponent; this designation means that WOComponent has a property called "employee" (possibly referring to an "Employee" object) which in turn has a property called "name". In this case, the binding is two-way; changes in the dynamic element are reflected in the WOComponent property, and changes in the WOComponent property are communicated to the dynamic element. The value of an association can also be a constant, in which case the binding is one-way: WOComponent to dynamic element.

WOElement

Inherits From:	NSObject
Declared In:	WebObjects/WOElement.h

Class Description

The WOElement class is the abstract superclass of all objects that represent static and dynamic UI elements on a World Wide Web page (currently, HTML and PDF elements). You cannot directly instantiate objects from WOElement; you must create a concrete subclass of WOElement and generate objects from it.

Note: For custom dynamic elements, you need to create a subclass of WODynamicElement.

WOElement declares the three methods corresponding to the phases of the request-response loop (invoked in the following order), but WOElement's implementations do nothing:

- **takeValuesFromRequest:inContext:**
- **invokeActionForRequest:inContext:**
- **appendToResponse:inContext:**

The first argument of these messages is an object that represents the HTTP request or response (WORequest or WOResponse). The second argument is a WOContext object that represents the context of the transaction.

Concrete subclasses of WOElement (or WODynamicElement) must, at minimum, implement **appendToResponse:inContext:**. Subclasses of WODynamicElement must implement one or both of the remaining methods.

Method Types

Handling requests	– **appendToResponse:inContext:**
	– **invokeActionForRequest:inContext:**
	– **takeValuesFromRequest:inContext:**

Instance Methods

appendToResponse:inContext:

– (void)**appendToResponse:**(WOResponse *)*aResponse* **inContext:**(WOContext *)*aContext*

This method is invoked in WOElement objects in the request-handling phase when objects involved in the current transaction append their HTML content to the transaction's WOResponse object. If the WOElement has child WOElements, it should forward the message to them. WOElement's default implementation of this method does nothing.

See also: WOResponse class for methods used to append HTML content

invokeActionForRequest:inContext:

– (WOElement *)**invokeActionForRequest:**(WORequest *)*aRequest* **inContext:**(WOContext *)*aContext*

This method is invoked in WOElements in the phase of request handling that results in the triggering of an action method and the return of a response WOComponent. In this phase, the message is propagated through the objects of the application until the dynamic element for the activated HTML control (for instance, a custom button) responds to the message by invoking the method in the request component that is bound to the action. To see if it has been activated, the dynamic element should check its element ID (obtained from its WOContext) against the sender ID in the request and context. To invoke the action method, the dynamic element should return the value of the action. The default WOElement implementation of this method returns nil

See also: WOContext class for a description of element IDs

takeValuesFromRequest:inContext:

– (void)**takeValuesFromRequest:**(WORequest *)*aRequest* **inContext:**(WOContext *)*aContext*

This method is invoked in (dynamic) WOElement objects during the phase of request handling that extracts user-entered data. Each dynamic element acquires any entered data (such as HTML form data) or changed state (such as a check in a check box) associated with an attribute and assigns the value to the WOComponent variable bound to the attribute. In this way, even back-end business objects are updated. The default WOElement implementation of this method does nothing.

See also: WORequest class for methods used to extract form data

WOMailDelivery

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOMailDelivery.h

Class Description

WOMailDelivery uses a tool compiled on all platforms: `/System/Library/WebObjects/Executables/WOSendMail[.exe]`. This tool constructs an email message from a file and uses SMTP to send it. It requires an SMTP server to be set. There is a default value for this SMTP hostname: "smtp". To change this value, use the following command:

```
defaults write NSGlobalDomain WOSMTPHost "aHostName"
```

Note that this default can be handled by WOApplication as a command-line argument.

There is only one instance of WOMailDelivery, which you access with the **sharedInstance** class method. You cannot create one of your own.

Method Types

Obtaining an instance	
	– sharedInstance
Composing mail	
	– composeEmailFrom:to:cc:subject:component:send:
	– composeEmailFrom:to:cc:subject:plainText:send:
Sending mail	
	– sendEmail:

Class Methods

sharedInstance

+ (WOMailDelivery *)**sharedInstance**

Instance Methods

Returns the current application's WOMailDelivery instance. Use this method instead of creating an instance of your own.

composeEmailFrom:to:cc:subject:component:send:

– (NSString *)**composeEmailFrom:**(NSString *)*sender*
to:(NSArray *)*destination*
cc:(NSArray *)*ccAddresses*
subject:(NSString *)*subject*
component:(WOComponent *)*aComponent*
send:(BOOL)*flag*

Composes an email message to *destination* with "from," "cc," and "subject" lines. The body of the message is the HTML generated when this method invokes **generateResponse** on *aComponent*. WOMailDelivery uses the WOCGIAdaptorURL default to complete all URLs in the message to be mailed, so the email's reader can click on the URLs to visit them.

If *flag* is YES, the message is sent immediately.

composeEmailFrom:to:cc:subject:plainText:send:

– (NSString *)**composeEmailFrom:**(NSString *)*sender*
to:(NSArray *)*destination*
cc:(NSArray *)*ccAddresses*
subject:(NSString *)*subject*
plainText:(NSString *)*message*
send:(BOOL)*flag*

Composes an email message to *destination* with "from," "cc," and "subject" lines, setting the content type of the email as (Content-type: TEXT/PLAIN; CHARSET=US-ASCII). If *flag* is YES, the message is sent immediately.

sendEmail:

– (void)**sendEmail:**(NSString *)*mailString*

Sends *anEmail*, with *anEmail* being an NSString following the SMTP format.The **composeEmailFrom...** methods return such NSStrings and this method lets you modify those strings before sending them.

WORequest

Inherits From:	NSObject
Declared In:	WebObjects/WORequest.h

Class Description

A WORequest object typically represents an HTTP request and thus constitutes an event that requires a reaction from a WebObjects application. WORequest objects encapsulate the data transmitted to a HTTP server in a request. Requests originate from user actions in a browser, such as the submission of a URL or a mouse click on a hyperlink, button, or active image in a page; from the perspective of WebObjects, the URL identifies a WebObjects application and the click on a control usually results in the display of a page of a WebObjects application. Such actions cause the browser to send an HTTP request to an HTTP server, which forwards the request to a WebObjects adaptor, which converts it to a WORequest object and sends that object to the appropriate request handler.

WORequest objects can also be created from HTTP requests sent by client-side components (Java applets specially modified to interact with the server side of a WebObjects application), and from HTTP requests submitted by custom client-side programs that don't use the Java client-side components. More rarely, WORequest objects can originate from custom adaptors that handle HTTP requests *or* non-HTTP events. (All the adaptors shipped with WebObjects handle HTTP events only).

Since adaptors usually create WORequest objects, and since you can usually use WebObjects' adaptors without modifications, you probably won't have to create your own instances of WORequest in your code (although you can if you need to). More typically, your code will obtain information from WORequest objects as they become available during certain points in the request-response loop. The application supplies WORequest objects as arguments in the **takeValuesFromRequest:inContext:** and **invokeActionForRequest:inContext:** methods, which are implementable by WOApplication, WOSession, WOComponent, and WOElement objects. You can also obtain the current WORequest object at any time during request handling through WOContext's **request** method.

Note: Because WORequest objects usually correspond to HTTP requests, the data they encapsulate is almost the same as what you would find in an HTTP request. Thus an understanding of HTTP requests is important for understanding the data vended by WORequest objects. A recommended prerequisite therefore is to review the current HTTP specification or HTTP documentation.

Adopted Protocols

NSCopying

- – copy
- – copyWithZone:

Method Types

Working with cookies

- – cookieValueForKey:
- – cookieValues
- – cookieValuesForKey:

Form values

- – defaultFormValueEncoding
- – formValueEncoding
- – formValueForKey:
- – formValueKeys
- – formValues
- – formValuesForKey:
- – isFormValueEncodingDetectionEnabled

Headers

- – headerForKey:
- – headerKeys
- – headersForKey:

Request handling

- – requestHandlerKey
- – requestHandlerPath
- – requestHandlerPathArray

Form Values

- – setDefaultFormValueEncoding:
- – setFormValueEncodingDetectionEnabled:

Obtaining attributes

– adaptorPrefix
– applicationName
– applicationNumber
– browserLanguages
– content
– httpVersion
– isFromClientComponent
– initWithMethod:uri:httpVersion:headers:content:userInfo:
– method
– uri
– userInfo

Instance Methods

adaptorPrefix

– (NSString *)**adaptorPrefix**

Returns the part of the request's URI that is specific to a particular adaptor. This is typically a URL ending in "/WebObjects", "/WebObjects.exe", "/WebObjects.dll", or uppercase versions of these strings. WebObjects uses a request's adaptor prefix to set the adaptor prefix in the generated response's URL. A WORequest must always have an adaptor prefix.

See also: – **applicationName**, – **applicationNumber**, – **uri**

applicationName

– (NSString *)**applicationName**

Returns the part of the request's URI that identifies the application the request is intended for. This name does not include the ".woa" extension of an application directory. A WORequest must always have an application name specified.

See also: – **adaptorPrefix**, – **applicationNumber**, **– uri**

applicationNumber

– (int)**applicationNumber**

Returns the part of the request's URI that identifies the particular application instance the request is intended for. This attribute is -1 if the request can be handled by any instance of the application, which is always the case for the first request in a session.

See also: – **applicationName**, – **uri**

browserLanguages

– (NSArray *)**browserLanguages**

Returns the language preference list from the user’s browser.

content

– (NSData *)**content**

Returns the content the WORequest was initialized with (which defaults to **nil**). The format of the data is undefined, but you can usually identify it by the value of the "content-type" header.

See also: – **httpVersion**, – **method**

cookieValueForKey:

– (NSString *)**cookieValueForKey:**(NSString *)*aKey*

Returns a string value for the cookie key specified by *aKey*.

See also: – **cookieValues**, – **cookieValuesForKey:**, WOCookie class specification

cookieValues

– (NSDictionary *)**cookieValues**

Returns a dictionary of cookie values and cookie keys.

See also: – **cookieValueForKey:**, – **cookieValuesForKey:**, WOCookie class specification

cookieValuesForKey:

– (NSArray *)**cookieValuesForKey:**(NSString *)*aKey*

Returns an array of values for the cookie key specified by *aKey*. Use this method to retrieve information stored in a cookie in an HTTP header. Valid keys are specified in the cookie specification.

See also: – **cookieValueForKey:**, – **cookieValues**, WOCookie class specification

defaultFormValueEncoding

– (NSStringEncoding)**defaultFormValueEncoding**

Returns the *default* string encoding the WORequest object uses for converting form values from ASCII to Unicode. It uses the default encoding only when it can detect no encoding from the ASCII form values or if encoding detection is disabled. If no default form-value encoding is set, NSISOLatin1StringEncoding is used.

See also: **– setDefaultFormValueEncoding:**

formValueEncoding

– (NSStringEncoding)**formValueEncoding**

Returns the encoding last used to convert form values from ASCII to Unicode. This encoding is either the result of an earlier detection of form-value encoding or the default form value encoding.

See also: – **defaultFormValueEncoding**, **– isFormValueEncodingDetectionEnabled**

formValueForKey:

– id **formValueForKey:**(NSString *)*aKey*

Returns a form value identified by the name *aKey.* If there are multiple form values identified by the same name, only one of the values is returned, and which of these values is not defined. You should use this method for names that you know occur only once in the name/value pairs of form data.

formValueKeys

– (NSArray *)**formValueKeys**

Returns an array of NSStrings corresponding to the names (or keys) used to access values of a form. The array is not sorted in any particular order, and is not necessarily sorted in the same order on successive invocations of this method.

formValues

– (NSDictionary *)**formValues**

Returns an NSDictionary containing all of the form data name/value pairs.

formValuesForKey:

– (NSArray *)**formValuesForKey:**(NSString *)*aKey*

Returns an array of all values (as NSStrings) of the form identified by the name *aKey.* This array is not sorted in any particular order, and is not necessarily sorted in the same order on successive invocations of this method. You should use this method when you know that a name (key) used for accessing form data can be matched with more than one value.

headerForKey:

– (NSString *)**headerForKey:**(NSString *)*aKey*

Returns one value of a particular header in the header dictionary the request was initialized with. This will be a string corresponding to one of the values of the header whose name is passed in as the key argument. If the specified header has multiple values, only one of these values is returned, and which one of them this is is not defined. However, on successive invocations of this method, the same value will always be returned. This method is intended to be used for headers that are known to have only one value.

headerKeys

– (NSArray *)**headerKeys**

Returns an array of the keys of the header dictionary the request was initialized with (which default to an empty dictionary). This will be an array of strings corresponding to the headers' names. The array is not sorted in any particular order, and not necessarily sorted in the same order on successive invocations of this method.

headersForKey:

– (NSArray *)**headersForKey:**(NSString *)*aKey*

Returns the values of a particular header that is identified by *aKey.* The returned object contains NSStrings sorted in no particular order, but which will always be sorted in the same order on successive invocations of this method. Use this method for headers that you know have (or can have) multiple values.

httpVersion

– (NSString *)**httpVersion**

Returns the HTTP version the request was initialized with. An application uses the WORequest's HTTP version to initialize the HTTP version of the response that is generated by request handling. The WORequest's HTTP version typically derives from the HTTP version of the client (for example, the browser) that initiated the request.

initWithMethod:uri:httpVersion:headers:content:userInfo:

– (id)**initWithMethod:**(NSString *)*aMethod* **uri:**(NSString *)*aURL* **httpVersion:**(NSString *)*anHTTPVersion* **headers:**(NSDictionary *)*someHeaders* **content:**(NSData *)*aContent* **userInfo:**(NSDictionary *)*userInfo*

Returns a WORequest object initialized with the specified parameters. The first two arguments are required:

- *aMethod* must be either "GET" or "POST"; anything else causes an exception to be raised.
- *aURL* must be a valid URL; if the URL is invalid, an exception is raised.

If either argument is omitted, an exception is raised.

The remaining arguments are optional; if you specify **nil** for these, the designated initializer substitutes default values or initializes them to **nil**. The *someHeaders* argument (if not **nil**) should be a dictionary whose NSString keys correspond to header names and whose values are arrays of one or more strings corresponding to the values of each header. The *userInfo* dictionary can contain any information that the WORequest object wants to pass along to other objects involved in handling the request.

For more information on each argument, see the description of the corresponding accessor method.

isFormValueEncodingDetectionEnabled

– (BOOL)**isFormValueEncodingDetectionEnabled**

Returns whether detection of form-value encoding is allowed to take place when form values are obtained.

See also: **– setFormValueEncodingDetectionEnabled:**

isFromClientComponent

– (BOOL)**isFromClientComponent**

Returns whether the request originated from an event in a client-side component (that is, a Java applet that can interact with the server side of a WebObjects application).

If you use dynamic elements and write write HTML code in the response, you should check that the request is not from a client-side component before writing into the response.

method

– (NSString *)**method**

Returns the method the WORequest object was initialized with. A WORequest's method defines where it will look for form values. The only currently supported methods are "GET" and "PUT", which have the same meaning as the HTTP request method tokens of the same name.

See also: – **content**, **– httpVersion**

requestHandlerKey

– (NSString *)**requestHandlerKey**

Returns the part of the request’s URI which identifies the request handler. This identifies the request handle which will process the reuquest and cannot be **null**.

requestHandlerPath

– (NSString *)**requestHandlerPath**

Returns the part of the URL which identifies, for a given request handler, which information is requested. Different request handlers use this part of the URL in different ways.

requestHandlerPathArray

– (NSArray *)**requestHandlerPathArray**

Returns the request handler path decomposed into elements.

sessionID

– (NSString *)**sessionID**

Returns the session ID, or **nil** if no session ID is found. This method first looks for the session ID in the URL, then checks the form values, and finally checks to see if the session ID is stored in a cookie.

setDefaultFormValueEncoding:

– (void)**setDefaultFormValueEncoding:**(NSStringEncoding)*anEncoding*

Sets the default string encoding for the receiver to use when converting its form values from ASCII to Unicode. The default string encoding is called into play if the WORequest cannot detect an encoding from the ASCII form values or if encoding detection is disabled. If no default form value encoding is explicitly set, the WORequest uses NSISOLatin1StringEncoding.

See also: – **defaultFormValueEncoding**, **– setFormValueEncodingDetectionEnabled:**

setFormValueEncodingDetectionEnabled:

– (void)**setFormValueEncodingDetectionEnabled:**(BOOL)*flag*

Enables or disables automatic detection of the best encoding for the receiver to use when it converts form values from ASCII to Unicode. When detection is enabled, a WORequest object scans the ASCII form values and applies heuristics to decide which is the best encoding to use. If no specific encoding is discernible, or if detection is disabled, the WORequest uses the default form value encoding for the conversion.

See also: **– isFormValueEncodingDetectionEnabled**,**– setDefaultFormValueEncoding:**

uri

– (NSString *)**uri**

Returns the Uniform Resource Identifier (URI) the WORequest was initialized with. For a session's first request, the URI indicates the resource that the request is seeking (such as a WebObjects application); for subsequent requests in the session, the URI indicates which page of the application should handle the request. If the request was caused (as is usually the case) by a web browser submitting a URL to an HTTP server, the URI is that part of the URL that follows the port number. Because the format of WebObjects URLs and the corresponding request URI might change between different versions of WebObjects, you should not attempt to parse the URI returned by this method. Instead, use WORequest's accessor methods to access particular URI/URL components.

See also: – **adaptorPrefix**, – **applicationName**, **– applicationNumber**

userInfo

– (NSDictionary *)**userInfo**

Returns the value of the user information the receiver was initialized with (**nil** by default). WebObjects imposes no restrictions on the format or content of the user information dictionary. In fact, WebObjects classes do not themselves use the dictionary, but just pass it around as the request is handled. Custom adaptors, for example, could initialize the dictionary with special information for other objects of an application.

WORequestHandler

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOResourceManager.h

Class Description

WORequestHandler is an abstract class that defines request handlers. A request handler is an object that can handle requests received by the WebObjects adaptor. All WebObjects applications have multiple request handlers that can handle certain types of requests. Three private request handlers are defined in the WebObjects framework:

- WOComponentRequestHandler, which handles requests for actions implemented in a component.
- WODirectActionRequestHandler, which handles requests for actions implemented in a WODirectAction class.
- WOResourceRequestHandler, which handles requests for resources.

These three request handlers handle most styles of requests that an application can typically receive. If you want to create your own style of request, then you should write your own WORequestHandler. Unless you write your own request handler, your code typically won't have to directly interact with WORequestHandler objects at all.

Adopted Protocols

NSLocking

– lock
– unlock

Method Types

Handling Requests

– handleRequest:

Instance Methods

handleRequest:

– (WOResponse *)**handleRequest:**(WORequest *)*aRequest*

Request handlers must implement this method and perform all request-specific handling. By default, a request is an HTTP request. You must supply your own server-side adaptor to accept anything other than HTTP.

WOResourceManager

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOResourceManager.h

Class Description

WOResourceManager manages an application's resources. It defines methods that retrieve resources from standard directories. Each WebObjects application contains a resource manager object, which you can access by sending **resourceManager** to the WOApplication class

Adopted Protocols

NSLocking

– lock
– unlock

Method Types

Retrieving resources

– pathForResourceNamed:inFramework:languages:
– urlForResourceNamed:inFramework:languages:request:

Retrieving localized strings

– stringForKey:inTableNamed:withDefaultValue:inFramework:
languages:

Managing the application-wide data cache

– flushDataCache
– removeDataForKey:session:
– setData:forKey:mimeType:session:

Instance Methods

flushDataCache

– (void)**flushDataCache**

Removes all data from the image data cache. Use this method if you are storing data in the application-wide cache that you no longer need.

Access to the WOResourceManager object is locked at the beginning of this method and unlocked at the end.

See also: – **removeDataForKey:session:**, – **setData:forKey:mimeType:session:**

lock

– (void)**lock**

Locks access to the WOResourceManager object. When the WOResourceManager is locked, no other threads may access it.

Usually, you don’t need to invoke this method in your own code. All messages that you send to a WOResourceManager object lock access to the object at the beginning of the method and unlock access at the end. You only need to use this method if you’re subclassing WOResourceManager. In that case, you should lock access to the WOResourceManager object in methods that load resources.

See also: – **unlock**

pathForResourceNamed:inFramework:languages:

– (NSString *)**pathForResourceNamed:**(NSString *)*aResourceFile*
    inFramework:(NSString *)*aFrameworkName*
    languages:(NSArray *)*languagesList*

Returns the absolute path for the resource *aResourceFile*. Include the file’s extension when specifying *aResourceFile*. If the file is in the application, specify **lnil** for the framework argument.

This method always returns a path like **/Local/Library/WebObjects/Applications/MyApp.woa/WebServerResources/MyResource**. It does not return the path relative to the HTTP server’s document root unless the entire application is located in the document root.

Access to the WOResourceManager object is locked at the beginning of this method and unlocked at the end.

See also: – **urlForResourceNamed:inFramework:languages:request:**

removeDataForKey:session:

– (void)**removeDataForKey:**(NSString *)*key* **session:**(WOSession *)*aSession*

Removes the data stored in the data cache under the key *key*. The session argument is currently ignored; specify **nil** to have WOResourceManager use the application-wide cache.

This method is used by default when a dynamic element requests an image or embedded object from a database and the **key** attribute is not set for that dynamic element. If the **key** attribute isn't set, the data retrieved from the database is stored in the cache using **setData:forKey:mimeType:session:**, sent to the dynamic element, and then removed from the cache using **removeDataForKey:session:**. If the **key** attribute is set, **removeDataForKey:session:** is not invoked.

You rarely need to invoke this method yourself. Use it only if you need to limit the amount of memory your application uses, if your application has data stored in the cache, and you know that the data is no longer needed.

Access to the WOResourceManager object is locked at the beginning of this method and unlocked at the end.

See also: – **flushDataCache**

setData:forKey:mimeType:session:

– (void)**setData:**(NSData *)*someData*
  forKey:(NSString *)*key*
  mimeType:(NSString *)*type*
  session:(WOSession *)*aSession*

Adds the image or embbedded object *someData* of MIME type *type* to the data cache for the session specify by *aSession*. The data is stored under the key *key*. The session argument is currently ignored; specify **nil** to have WOResourceManager use the application-wide cache.

This method is invoked any time a dynamic element requests an image or embedded object from a database. You rarely need to invoke it yourself.

By default when a dynamic element requests an image from the database, WOResourceManager fetches the image, stores it in the cache using **setData:forKey:mimeType:session:**, sends it to the dynamic element, and then removes it from the cache using **removeDataForKey:session:**. However, if the dynamic element has a **key** attribute defined, then the image is stored in the database under that key, and it is not removed from the database.

Access to the WOResourceManager object is locked at the beginning of this method and unlocked at the end.

See also: – **flushDataCache**

stringForKey:inTableNamed:withDefaultValue:inFramework:languages:

– (NSString *)**stringForKey:**(NSString *)*aKey*
  inTableNamed:(NSString *)*aTableName*
  withDefaultValue:(NSString *)*aDefaultValue*
  inFramework:(NSString *)*aFrameworkName*
  languages:(NSArray *)*languagesList*

Returns a localized string from string table *aTable***.strings** using *aKey* to look it up. If no string value for the key is found in the table, *aDefaultValue* (optional) is returned. The method first searches the *aTable***.strings** file, if it exists, in each of the localized (**.lproj**) subdirectories of the application wrapper; searching proceeds in the order specified by the array *languagesList*. If no string value matching the key is found, the search then continues to the *aTable***.strings** file (if it exists) directly under the Resources directory (inside the directory with the **.woa** extension).

unlock

– (void)**unlock**

Removes the lock on the WOResourceManager object, allowing other threads to access it.

Usually, you don't need to invoke this method in your own code. All messages that you send to a WOResourceManager object lock access to the object at the beginning of the method and unlock access at the end. You only need to use this method if you're subclassing WOResourceManager. In that case, you should lock access to the WOResourceManager object in methods that load resources and unlock when the method is finished accessing the WOResourceManager object.

See also: – **lock**

urlForResourceNamed:inFramework:languages:request:

– (NSString *)**urlForResourceNamed:**(NSString *)*aResourceFile*
  inFramework:(NSString *)*aFrameworkName*
  languages:(NSArray *)*languagesList*
  request:(WORequest *)*aRequest*

Returns the URL associated with a resource named *aResourceFile*. The URL returned is of the following form:

WebObjects/MyApp.woa/WebServerResources/English.lproj/*aResourceFile*

Include the file's extension when specifying *aResourceFile*. If the file is in the application, specify**l nil** for the framework argument.

This method locates resources under the application or framework. The URL returned is computed by concatenating the application's base URL (returned by WOApplication's **baseURL** method and settable using the WOApplicationBaseURL user default) and the relative path of the resource. This method does not check to see if the file is actually under the document root. For example, if your application is installed in **/Local/Library/WebObjects/Applications**, and the method finds *aResourceFile* in the **Resources** directory, it returns:

/WebObjects/MyApp.woa/Resources/*aResourceFile*

even though the **Resources** directory is not under the document root.

Access to the WOResourceManager object is locked at the beginning of this method and unlocked at the end.

See also: – **pathForResourceNamed:inFramework:languages:**

WOResponse

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOResponse.h

Class Description

A WOResponse object represents an HTTP response that an application returns to a Web server to complete a cycle of the request-response loop. The composition of a response occurs during the third and final phase of this loop, a phase marked by the propagation of the **appendToResponse:inContext:** message through the objects of the application. The WOApplication object first sends this message, passing in a newly-created WOResponse object as an argument. WOElement objects, which represent the dynamic and static HTML elements on a page, respond to the message by appending their HTML representation to the content of the WOResponse object. WOApplication, WOSession, and WOComponent objects can also respond to the message by adding information to the WOResponse object.

A WOResponse has two major parts: HTML content and HTTP information. The content is what is displayed in a Web browser; it can include *escaped* HTML, which is HTML code shown "as is," uninterpreted. The other information encapsulated by a WOResponse object is in the handling the response. This HTTP data includes headers, status codes, and version string. See the HTTP specification or HTTP documentation for descriptions of these items.

As you might expect, the methods of the WOResponse class can be divided into two groups, those that add to a response's HTML content and those that read and set HTTP information. The former group consists of methods that escape HTML (**appendContentHTMLAttributeValue:** and **appendContentHTMLString:**) and those that don't. For images and other binary data, you can use the **appendContentData:**. You can obtain and set the entire content of the response with **content** and **setContent:**. The following example shows a sequence of "appendContent" messages that compose an HTTP "POST" message:

```
[aResponse appendContentString:@"&ltform method=\"POST\" action=\""];
[aResponse appendContentHTMLAttributeValue:[aContext url]];
[aResponse appendContentCharacter:'"'];
[aResponse.appendContentString:@"&gt"];
```

Most of the remaining WOResponse methods set and read the response's HTTP headers, the HTTP status code, and the HTTP version.

Content Encodings

You can set the string encoding used for the response content with **setContentEncoding:** and you find out what the current encoding is with **contentEncoding**. An integer represents the type of encoding. The following table lists these integer values along with their OPENSTEP string-constant names.

int Value	OPENSTEP Name	Notes
1	NSASCIIStringEncoding	0 through 127
2	NSNEXTSTEPStringEncoding	
3	NSJapaneseEUCStringEncoding	
4	NSUTF8StringEncoding	
5	NSISOLatin1StringEncoding	default
6	NSSymbolStringEncoding	
7	NSNonLossyASCIIStringEncoding	7-bit verbose ASCII to represent all unichars
8	NSShiftJISStringEncoding	
9	NSISOLatin2StringEncoding	
10	NSUnicodeStringEncoding	
11	NSWindowsCP1251StringEncoding	Cyrillic; same as AdobeStandardCyrillic
12	NSWindowsCP1252StringEncoding	Windows Latin1
13	NSWindowsCP1253StringEncoding	Windows Greek
14	NSWindowsCP1254StringEncoding	Windows Turkish
15	NSWindowsCP1250StringEncoding	Windows Latin2
21	NSISO2022JPStringEncoding	ISO 2022 Japanese encoding for electronic mail

Adopted Protocols

NSCopying	
	– copy
	– copyWithZone:

WOActionResults	– generateResponse

Method Types

Creation	– init
Obtaining attributes	+ defaultEncoding – content – headerForKey: – headerKeys – headersForKey: – httpVersion – status – userInfo
Setting attributes	+ setDefaultEncoding: – setContent: – setHeader:forKey: – setHeaders:forKey: – setHTTPVersion: – setStatus: – setUserInfo:
Appending response content	– appendContentBytes:length: – appendContentCharacter: – appendContentData: – appendContentString: – setContentEncoding: – contentEncoding
Working with HTML content	– appendContentHTMLAttributeValue: – appendContentHTMLString: + stringByEscapingHTMLString: + stringByEscapingHTMLAttributeValue:
Working with cookies	– addCookie: – cookies – removeCookie:

Class Methods

defaultEncoding

+ (NSStringEncoding)**defaultEncoding**

Returns the default character encoding used to construct a new WOResponse. By default, this encoding is NSISOLatin1. For more information, see “Content Encodings”.

setDefaultEncoding:

+ (void)**setDefaultEncoding:**(NSStringEncoding)*aStringEncoding*

Lets you specify the character encoding used by default when construcing a new WOResponse. For more information, see “Content Encodings”.

stringByEscapingHTMLString:

+ (NSString *)**stringByEscapingHTMLString:**(NSString *)*aString*

This method takes a string and, if escaping is required, returns a new string with certain characters escaped out. If escaping is not required, no conversion is performed and *aString* is returned. Use this method to escape strings which will appear in the visible part of an HTML file (that is, not inside a tag). The escaped characters are:

&	&amp;
"	&quot;
<	&lt;
>	&gt;

stringByEscapingHTMLAttributeValue:

+ (NSString *)**stringByEscapingHTMLAttributeValue:**(NSString *)*aString*

This method takes astring and, if escaping is required, returns a new string with certain characters escaped out. If escaping is not required, no conversion is performed and *aString* is returned. Use this method to escape strings which will appear as attribute values of a tag. The escaped characters are:

&	&amp;

"	&quot;
\t	&#9;
\n	&#10;
\r	&#13;
<	&lt;
>	&gt;

Instance Methods

addCookie:

– (void)**addCookie:**(WOCookie *)*aCookie*

Adds the specified WOCookie object to the response.

See also: – **cookies**, – **removeCookie:**, WOCookie class specification

appendContentBytes:length:

– (void)**appendContentBytes:**(const void *)*someBytes* **length:**(unsigned)*length*

Appends *length* number of bytes pointed to by *someBytes* to the HTTP response.

appendContentCharacter:

– (void)**appendContentCharacter:**(char)*aChar*

Appends a single ASCII character (*aChar*) to the HTTP response.

Example:

```
// ...
if (aFlag)
    [aResponse appendContentCharacter:'Y'];
else
    [aResponse appendContentCharacter:'N'];
```

appendContentData:

– (void)**appendContentData:**(NSData *)*dataObject*

Appends a data-encapsulating object (*dataObject*) to the HTTP response.

appendContentHTMLAttributeValue:

– (void)**appendContentHTMLAttributeValue:**(NSString *)*aValue*

Appends an HTML attribute value to the HTTP content after transforming the string *aValue* into an NSData object using the receiver's content encoding. Special HTML characters ("<", ">", "&", and double quote) are *escaped* so that the browser does not interpret them. In other words, the message

```
[aResponse appendContentHTMLAttributeValue:@"<B>"];
```

would transform the argument to "&lt;B&gt;".

See also: **– setContentEncoding:**

appendContentHTMLString:

– (void)**appendContentHTMLString:**(NSString *)*aString*

Appends an HTML string to the HTTP response after transforming the string *aString* into an NSData object using the receiver's content encoding. Special HTML characters ("<", ">", "&", and double quote) are *escaped* so that the browser does not interpret them. For example, "<P>" becomes "&ltP&gt".

See also: **– setContentEncoding:**

appendContentString:

– (void)**appendContentString:**(NSString *)*aString*

Appends a string to the content of the HTTP response. The string is transformed into an NSData object using the receiver's content encoding. The special HTML characters "<", ">", "&", and double-quote are not escaped so the browser can interpret them as HTML.

content

– (NSData *)**content**

Returns the HTML content of the receiver as an NSData object.

An exception is raised if you attempt to get the content when all elements of the page have not had their chance to append HTML to the response. Thus, you should invoke this method in the application object’s

handleRequest: method, after super’s **handleRequest:** has been invoked. (For scripted applications, **handleRequest:** is implemented in Application.wos). Note that at this point in the request-handling process, the components, pages, and session have already been put to sleep, so you won’t have access to any context, session, or page information. If you need such information for your response, store it somewhere--such as in WOResponse’s “user info” dictionary—at a point when you do have access to it. You may want to do this in your application’s **appendToResponse:inContext:** method, for example.

See also: – **setContent:**, **– setContentEncoding:**

contentEncoding

– (NSStringEncoding)**contentEncoding**

Returns an integer representing the encoding used for the response content. See “Content Encodings” in the class description for a mapped list of supported encodings and their Objective-C names. Usually, you will want the response encoding to be the same as that used by the submitting form on the client browser. In this case it is preferable to use WORequest's **formValueEncoding**.

```
NSStringEncoding theEncoding = [[aContext request] formValueEncoding];
```

The default string encoding is ISO Latin1.

See also: – **setContent:**, **– setContentEncoding:**

cookies

– (NSArray *)**cookies**

Returns an array of WOCookie objects to be included in the response.

See also: – **addCookie:**, – **removeCookie:**, WOCookie class specification

generateResponse

– (WOResponse *)**generateResponse**

Returns a WOResponse object. WOResponse’s implementation simply returns itself.

See also: – **generateResponse** (WOComponent)

headerForKey:

– (NSString *)**headerForKey:**(NSString *)*aKey*

Returns the HTTP header information identified by *aKey*. If there are multiple headers associated with the one key, only the first one is returned. Returns **nil** if there are no headers for the key.

See also: **– setHeader:forKey:**

headerKeys

– (NSArray *)**headerKeys**

Returns an array of string keys associated with the receiver's HTTP headers. Returns **nil** if there are no headers. You could easily test to see if a header is included by doing something similar to this:

```
NSArray *hKeys =  [aResponse headerKeys];
if ([hKeys containsObject:@"expires"]) {
    // do something
}
```

See also: **– setHeaders:forKey:**

headersForKey:

– (NSArray *)**headersForKey:**(NSString *)*aKey*

Returns *all* HTTP headers identified by *aKey*.

See also: **– setHeaders:forKey:**

httpVersion

– (NSString *)**httpVersion**

Returns the version of HTTP used for the response (for example, "HTTP/1.0").

See also: **– setHTTPVersion:**

init

– (id)**init**

Initializes a WOResponse instance. HTTP status is set to 200 (OK), client caching is enabled, and the default string encoding is made ISO Latin 1.

removeCookie:

– (void)**removeCookie:**(WOCookie *)*aCookie*

Removes the specified WOCookie object from the response.

See also: – **cookies**, – **removeCookie:**, WOCookie class specification

setContent:

– (void)**setContent:**(NSData *)*someData*

Sets the HTML content of the HTTP response to *someData*.

See also: – **content**

setContentEncoding:

– (void)**setContentEncoding:**(NSStringEncoding)*anEncoding*

Sets the encoding used for the content of the HTTP response. See “Content Encodings” in the class description for a mapped list of supported encodings and their Objective-C names. The default string encoding is ISO Latin1.

See also: – **contentEncoding**

setHTTPVersion:

– (void)**setHTTPVersion:**(NSString *)*aVersion*

Sets the version of HTTP used for the response (for example, "HTTP/1.0").

See also: – **httpVersion**

setHeader:forKey:

– (void)**setHeader:**(NSString *)*aHeader* **forKey:**(NSString *)*aKey*

Sets the HTTP header *aHeader* in the receiver and associates, for retrieval, the HTTP key *aKey* with the header. This method is commonly used to set the type of content in a response, for example:

```
[aResponse setHeader:@"text/html" forKey:@"content-type"];
```

See also: – **headerForKey:**

setHeaders:forKey:

– (void)**setHeaders:**(NSArray *)*headerList* **forKey:**(NSString *)*aKey*

Sets the list of HTTP headers *headerList* in the receiver and associates, for retrieval, the HTTP key *aKey* with the list of header elements.

See also: – **headerKeys**, **– headersForKey:**

setStatus:

– (void)**setStatus:**(unsigned int)*anInt*

Sets the HTTP status to *anInt*. Consult the HTTP specification or HTTP documentation for the significance of status integers.

See also: **– status**

setUserInfo:

– (void)**setUserInfo:**(NSDictionary *)*aDictionary*

Sets a dictionary in the WOResponse object that, as a convenience, can contain any kind of information related to the current response. Objects further down the **appendToResponse:inContext:** message "chain" can retrieve this information using **userInfo**.

status

– (unsigned int)**status**

Returns an integer code representing the HTTP status. Consult the HTTP specification or HTTP documentation for the significance of these status codes.

By default, the status is 200 ("OK" status).

See also: **– setStatus:**

userInfo

– (NSDictionary *)**userInfo**

Returns a dictionary that, as a convenience, can contain any kind of information related to the current response. An object further "upstream" in the **appendToResponse:inContext:** message "chain" can set this dictionary in the WOResponse object as a way to pass information to other objects.

See also: **– setUserInfo:**

WOSession

Inherits From: NSObject

Declared In: WebObjects/WOSession.h

Class Description

WOSession objects represent *sessions*, periods during which access to a WebObjects application and its resources is granted to a particular client (typically a browser). An application can have many concurrent sessions, each with its own special "view" of the application and its own set of data values. For instance, one client could be accessing a "catalog" application, where that client is going from page to page, filling a virtual shopping cart with items for purchase. Another client might be accessing the same application at the same time, but that person might have different items in his or her shopping cart.

Perhaps the most important thing a WOSession object does is encapsulate state for a session. After the application handles a request, it stores the WOSession until the next request of the session occurs. All the information that is important for maintaining continuity throughout the session is preserved. And the integrity of session data is maintained as well; the data of a session not only persists between requests but is kept separate from that of all other sessions.

When you develop an application, you identify data with session-wide scope by declaring instance variables in your subclass of WOSession (or, for scripted applications, in **Session.wos**). Then, before the end of a cycle of the request-response loop, ensure that the instance variables hold current session values.

The application uses a *session ID* to identify a session object. Upon receiving the first request of a session, the application assigns a session ID (a unique, randomly generated string) to the session. The session ID appears in the URL between the application name and the page name.

At the end of each cycle of the request-response loop, the application stores the WOSession object according to the storage strategy implemented by the chosen WOSessionStore. When the application receives the next request of the session, it restores the WOSession, using the session ID as key.

To be stored and restored according to any WOSessionStore strategy, a WOSession must be convertable to an object archive. WOSessions are therefore asked to serialize and deserialize themselves prior to being archived and unarchived (in either binary or ASCII format). To accomplish this, the WOSession should implement the **encodeWithCoder:** and **initWithCoder:** methods of the NSCoding protocol.

Because storage of sessions in application memory can consume large amounts of memory over time, WOSession includes methods for controlling the lifespan of session objects. The **setTimeOut:** method sets a period of inactivity after which the session is terminated. The **terminate** method explicitly ends a session.

The WOSession class provides several other methods useful for tasks ranging from localization to database access:

- WOSession objects can interject custom session behavior into the request-response loop by implementing the request-handling methods (**takeValuesFromRequest:inContext:**, **invokeActionForRequest:inContext:**, and **appendToResponse:inContext:**) as well as **awake** and **sleep**.
- For database access, the **defaultEditingContext** method gives each WOSession object in an application its own Enterprise Objects editing context.
- An object in an application doesn't have to know which instance variables its WOSession holds in order to store session values. With the **setObject:forKey:** and **objectForKey:** methods it can store and retrieve values as needed. This mechanism is especially useful for reusable components.
- An application's WOSession objects also play a role in localization. Through the **setLanguages:** method you can store a list of the languages supported by the application. The sequence of language strings in the list indicates the order of language preference for a particular session. Several resource-access methods in WOResourceManager, WOApplication, and WOComponent refer to the **languages** array when they locate such things as localized strings, images, and sounds.
- WOSession objects also allow you to affect load balancing with the **setDistributionEnabled:** method; if the flag set by this method is NO (the default), transactions of the session are restricted to a single application instance. If this the case, the application instance number as well as the application host name are appended to the URL.

Adopted Protocols

NSCoding	
	– encodeWithCoder:
	– initWithCoder:
NSCopying	
	– copy
	– copyWithZone:

Method Types

Creating	
	– init

Obtaining attributes	
	– domainForIDCookies
	– expirationDateForIDCookies
	– isDistributionEnabled
	– sessionID
	– storesIDsInCookies
	– storesIDsInURLs
Setting attributes	
	– setDistributionEnabled:
	– setStoresIDsInCookies:
	– setStoresIDsInURLs:
Terminating	
	– terminate
	– isTerminating
	– timeOut
	– setTimeOut:
Localization	
	– languages
	– setLanguages:
Managing component state	
	– setObject:forKey:
	– objectForKey:
	– removeObjectForKey:
Managing enterprise objects	
	– defaultEditingContext
	– setDefaultEditingContext:
Handling requests	
	– appendToResponse:inContext:
	– awake
	– context
	– invokeActionForRequest:inContext:
	– sleep
	– takeValuesFromRequest:inContext:
Statistics	
	– statistics
Debugging	
	– debugWithFormat:
Page Management	
	– savePage:
	– restorePageForContextID:

Instance Methods

appendToResponse:inContext:

– (void)**appendToResponse:**(WOResponse *)*aResponse* **inContext:**(WOContext *)*aContext*

This method is invoked during the phase of the request-response loop during which the objects associated with a response page append their HTML content to the response. WOSession's default implementation of this method forwards the message to the WOComponent that represents the response page. Then, it records information about the current transaction by sending **recordStatisticsForResponse:inContext:** and then **descriptionForResponse:inContext:** to the WOStatisticsStore object.

Compiled or scripted subclasses of WOSession can override this method to replace or supplement the default behavior with custom logic.

See also: – **invokeActionForRequest:inContext:**, – **takeValuesFromRequest:inContext:**

awake

– (void)**awake**

Invoked at the beginning of a WOSession's involvement in a cycle of the request-response loop, giving the WOSession an opportunity to initialize its instance variables or perform setup operations. The default implementation does nothing.

See also: – **sleep**

context

– (WOContext *)**context**

Returns the WOContext object for the current transaction.

See also: WOContext class

debugWithFormat:

– (void)**debugWithFormat:**(NSString *)*aFormatString,...*

Prints a message to the standard error device (stderr), if **WODebuggingEnabled** is YES. The message can include formatted variable data using printf-style conversion specifiers, Note that in WebScript, all variables are objects, so the only conversion specifier allowed is **%@**. In compiled Objective-C code, all **printf** conversion specifiers are allowed.

You control whether this method displays output with the **WODebuggingEnabled** user default option. If **WODebuggingEnabled** is YES, then the **debugWithFormat:** messages display their output. If **WODebuggingEnabled** is NO, the **debugWithFormat:** messages don't display their output.

defaultEditingContext

– (EOEditingContext *)**defaultEditingContext**

Returns the default EOEditingContext object for the session. The method creates the editing context the first time that it is invoked and caches it for subsequent invocations. There is only one unique EOEditingContext instance per session. The instance is initialized with the default object store coordinator as the parent object store.

domainForIDCookies

– (NSString *)**domainForIDCookies**

Returns the path that is passed when creating a rendezvous cookie for the application. This path is lazily created the first time it is used from the current request's **adaptorPrefix** and the application name (including the ".woa" extension).

expirationDateForIDCookies

– (NSDate *)**expirationDateForIDCookies**

Returns when session and instance ID cookies expire. By default, no expiration date is set and this method returns nil. Override this method if you want to return some other time, such as the session expiration date.

Different applications can override this method to enforce different behavior:

- A typical online banking application might use cookies and set the timeout to a very short amount of time (two minutes, for example), so that if the client doesn't interact with the browser and no request is made of the server, the client's session is timed out. This could be easily enforced on both the client—by setting the cookie timeout—and on the server from within the session object.
- A site wishing to personalize its pages based upon a user ID might set the timeout far into the distant future so that even when a client shuts down his browser, the cookie will still be there when he comes back with a bookmarked URL.
- Sites that want you to log in each time you visit could store the user ID in a cookie and then set the expiration date on the cookie to nil so that the cookie will go away whenever the client quits their browser.

init

– (id)**init**

Initializes a WOSession object. Session timeout is set from the WOApplication method **sessionTimeout**. This method throws exceptions if no session ID has been assigned or if it cannot initialize the object for any other reason. Override **init** in compiled subclasses to perform custom initializations; as always, invoke the superclass method as the first thing.

invokeActionForRequest:inContext:

– (WOElement *)**invokeActionForRequest:**(WORequest *)*aRequest*
  inContext:(WOContext *)*aContext*

WOSession objects receive this message during the middle phase of the request-response loop. During this phase, the **invokeActionForRequest:inContext:** message is propagated through the objects of an application, most importantly, the WOElement objects of the request page. The dynamic element on which the user has acted (by, for example, clicking a button) responds by triggering the method in the request WOComponent that is bound to the action. The default behavior of WOSession is to send the message to the WOComponent object that represents the request. Compiled or scripted subclasses of WOSession can override this method to replace or supplement the default behavior with custom logic.

See also: – **appendToResponse:inContext:**, – **takeValuesFromRequest:inContext:**

isDistributionEnabled

– (BOOL)**isDistributionEnabled**

Returns whether state distribution among multiple application instances is enabled. Returns **false** by default since the default WOSessionStore (state in the server) does not allow distribution. If this flag is disabled, a specific application instance (whose identifying number is embedded in the URL) is assigned to the session.

See also: **setDistributionEnabled:**

isTerminating

– (BOOL)**isTerminating**

Returns whether the WOSession object will terminate at the end of the current request-response loop.

See also: – **terminate**

languages

– (NSArray *)**languages**

Returns the list of languages supported by the session. The order of language strings (for example, "French") indicates the preferred order of languages. This is initialized from the users's browser preferences unless explicitly set with **setLanguages:**. For details, see "Localization" in the WebObjects programming topics.

See also: – **setLanguages:**

objectForKey:

– (id)**objectForKey:**(NSString *)*aKey*

Returns an object stored in the session under a specific key.

See also: **– setObject:forKey:**

removeObjectForKey:

– (void)**removeObjectForKey:**(NSString *)*key*

Removes the object stored in the session under the specified key.

restorePageForContextID:

– (WOComponent *)**restorePageForContextID:**(NSString *)*contextID*

Returns a page instance stored in the session page cache. The key to the stored instance is its context ID, which derives from the transaction's WOContext or WORequest objects. This method returns **nil** if restoration is impossible.

See also: – **savePage:**

savePage:

– (void)**savePage:**(WOComponent *)*aPage*

Saves the page instance *aPage* in the session page cache. The context ID for the current transaction is made the key for obtaining this instance in the cache using **restorePageForContextID:**.

savePageInPermanentCache:

– (void)**savePageInPermanentCache:**(WOComponent*)*aPage*

Puts *aPage* into a separate page cache. This cache is searched first when attempting to restore the page the next time its requested. This effectively makes *aPage* live for the duration of the application regardless of the size of your page cache. This is useful whe you are using frames and its possible for a page of controls to be bumped from the page cache.

See also: – **permanentPageCacheSize** (WOApplication), – **setPermanentPageCacheSize:** (WOApplication)

sessionID

– (NSString *)**sessionID**

Returns the unique, randomly generated string that identifies the session object. The session ID occurs in the URL after the request handler key.

setDefaultEditingContext:

– (void)**setDefaultEditingContext:**(EOEditingContext *)*editingContext*

Sets the editing context to be returned by **defaultEditingContext**. This can be used to set an editing context initialized with a different parent object store than the default. This is useful when, for instance, each session needs its own login to the database. Once a default editing context has been established, you may not call **setDefaultEditingContext:** again. Therefore, to provide your own default editing context, you must call **setDefaultEditingContext:** before ever calling **defaultEditingContext** since that will lazily establish an editing context.

See also: – **defaultEditingContext**

setDistributionEnabled:

– (void)**setDistributionEnabled:**(BOOL)*aFlag*

Enables or disables the distribution mechanism that effects load balancing among multiple application instances. When disabled (the default), generated URLs include the application instance number; the adaptor uses this number to route the request to the specific application instance based on information in the configuration file. When this flag is enabled, generated URLs do not contain the application instance number, and thus transactions of a session are handled by whatever application instance is available.

See also: – **isDistributionEnabled**

setLanguages:

– (void)**setLanguages:**(NSArray *)*languages*

Sets the languages for which the session is localized. The ordering of language strings in the array determines the order in which the application will search *languages*.lproj directories for localized strings, images, and component definitions.

See also: – **languages**

setObject:forKey:

– (void)**setObject:**(id)*anObject* **forKey:**(NSString *)*aKey*

Stores an object within the session under a given key (*aKey*). This method allows a reusable component to add state dynamically to any WOSession object. This method eliminates the need for prior knowledge of the WOSession's instance variables. A suggested mechanism for generating a unique key prefix for a given subcomponent is to concatenate the component's name and its element ID. For a specific component instance, such a prefix should remain unique and invariant within a session.

See also: **– objectForKey:**

setStoresIDsInCookies:

– (void)**setStoresIDsInCookies:**(BOOL)*flag*

Enables or disables the cookie mechanism. Two cookies are created for you when enabled: a session ID cookie with the name “wosid,” and an instance ID cookie with the name “woinst.” By default, the cookie mechanism is disabled.

setStoresIDsInURLs:

– (void)**setStoresIDsInCookies:**(BOOL)*flag*

Enables or disables the storing of session and instance IDs in URLs. By default, IDs are stored in URLs.

setTimeOut:

– (void)**setTimeOut:**(NSTimeInterval)*seconds*

Set the session timeout in seconds. When a session remains inactive—that is, the application receives no request for this session—for a period longer than the time-out setting, the session will terminate, resulting in the deallocation of the WOSession object. By default, the session time-out is set from the WOApplication method **sessionTimeout**.

See also: – **timeOut**

sleep

– (void)**sleep**

Invoked at the conclusion of each request-response loop in which the session is involved, giving the WOSession the opportunity to deallocate objects initialized in the **awake** method. The default WOSession implementation does nothing.

statistics

– (NSArray *)**statistics**

Returns a list of the pages accessed by this session, ordered from first accessed to last. For each page, the string stored is obtained by sending **descriptionForResponse:inContext:** to the WOComponent object. By default, this returns the component's name. If the application keeps a CLFF log file, this list is recorded in the log file when the session terminates.

See also: – **appendToResponse:inContext:**

storesIDsInCookies

– (BOOL)**storesIDsInCookies**

Returns whether the cookie mechanism for storing session and instance IDs is enabled. The cookie mechanism is disabled by default.

storesIDsInURLs

– (BOOL)**storesIDsInURLs**

Returns whether the URL mechanism for storing session IDs and instance IDs is enabled. The URL mechanism is enabled by default.

takeValuesFromRequest:inContext:

– (void)**takeValuesFromRequest:**(WORequest *)*aRequest* **inContext:**(WOContext *)*aContext*

WOSession objects receive this message during the first phase of the request-response loop. During this phase, the dynamic elements associated with the request page extract any user input and assign the values to the appropriate component variables. The default behavior of WOSession is to send the message to the WOComponent object that represents the request. Compiled or scripted subclasses of WOSession can override this method to replace or supplement the default behavior with custom logic.

See also: – **appendToResponse:inContext:**, – **invokeActionForRequest:inContext:**

terminate

– (void)**terminate**

Causes the session to terminate after the conclusion of the current request-response loop.

See also: – **isTerminating**

timeOut

– (NSTimeInterval)**timeOut**

Returns the timeout interval in seconds.

See also: – **setTimeOut:**

WOSessionStore

Inherits From:	NSObject
Declared In:	WebObjects/WOSessionStore.h

Class Description

WOSessionStore, an abstract superclass, offers an object abstraction for storing client state per session. The application object (WOApplication) uses an instance of a concrete WOSessionStore subclass to implement a strategy for storing and retrieving session state. You typically set the WOSessionStore during application initialization through WOApplication's **setSessionStore:** method.

An application first creates a session (WOSession) when it receives a request without a session ID. When this first request has been handled, the application stores the WOSession object under a randomly generated session ID by invoking its own **saveSessionForContext:** method. This method by default forwards the message to the chosen WOSessionStore and that WOSessionStore takes care of the details of saving session state. When the next request comes in for that session, the application restores the session by sending itself **restoreSessionWithID:request:**, which by default is forwarded to the application's WOSessionStore. The WOSessionStore then asks the WOContext of the transaction for the session ID of the session. Based on the implementation of the WOSessionStore, the session object is located and returned.

There is one subclass of WOSessionStore implemented for the developer's convenience. A *server* WOSessionStore (the default) stores session state in the server, in application memory. The **serverSessionStore** method returns this WOSessionStore.

See the chapter "Managing State" in the *WebObjects Developers Guide* for the purposes, mechanisms, and limitations of session store in the server, page, and cookies.

You can create a custom session store by making a subclass of WOSessionStore. The subclass should properly implement the **saveSessionForContext:** and **restoreSessionWithID:request:** methods (using the session ID as the key for storage) and should have a public method that the application object can use to obtain an instance. Some interesting session stores could be:

- A database session store that stores session data in a database as blobs, with the session ID as the primary key. This kind of WOSessionStore can be shared by many instances of the same WebObjects application, thus distributing the load (requests) among the instances.
- An adaptive session store that stores session state either in cookies or on the server, depending on what the client supports.

If you create your own WOSessionStore class that generates persistent objects, you should implement an algorithm that cleans up session state after the session is inactive for a long time. The server WOSessionStore provided by WebObjects performs this clean-up properly, but the API is not yet public.

Method Types

Obtaining a session store	
	+ serverSessionStore
Checking a session in and out	
	– checkinSessionForContext:
	– checkoutSessionWithID:request:
Saving and restoring a context	
	– restoreSessionWithID:request:
	– saveSessionForContext:

Class Methods

serverSessionStore

+ (WOSessionStore *)**serverSessionStore**

Returns a WOSessionStore object that stores session state in application memory. Since this is the default storage strategy, you do not need to explicitly set the session store during application initialization if this is the strategy you want.

State storage in the server is the most secure and is the easiest to implement. You can also easily manage the amount of storage consumed by setting session timeouts, limiting the size of the page-instance cache, and page uniquing. (See "Managing State" in the *WebObjects Developers Guide* for details on these techniques.)

You may use WOSession's **initWithCoder:** method to restore session state from the archived data.

Instance Methods

checkinSessionForContext:

– (void)**checkinSessionForContext:**(WOContext *)*aContext*

This method calls **saveSessionForContext:** (implemented in the concrete subclass) to save the session referred to by *aContext* using whatever storage technique is supported by the receiver. This method also “checks in” the session so that pending (and future) requests for the same session may procede. This method is called by WOApplication to save the session even if the session was not previously checked out via **checkoutSessionWithID:request:** (that is, the session is a new session which was just created and, therefore, not restored).

checkoutSessionWithID:request:

– (WOSession*)**checkoutSessionWithID:**(NSString *)*aSessionID* **request:**(WORequest *)*aRequest*

This method returns a session for *aSessionID* if one is stored. This method calls **restoreSessionWithID: request:** (implemented in the concrete subclass) to do the actual session restoration using whatever storage technique is supported by the receiver. If the session is located and restored, this method also “checks out” *aSessionID* so that simultaneous access to the same session is precluded. If the session is not restored, the *aSessionID* is not checked out.

restoreSessionWithID:request:

– (WOSession *)**restoreSessionWithID:**(NSString *)*aSessionID* **request:**(WORequest *)*aRequest*

Implemented by a private concrete subclass to restore the current session object from a particular type of storage.

The default implementation of this method does nothing

saveSessionForContext:

– (void)**saveSessionForContext:**(WOContext *)*aContext*

Implemented by a private concrete subclass to save the current session object using a particular strategy for state storage. The default implementation of this method does nothing.

You may use the method **encodeWithCoder:** to save session state to archived data.

WOStatisticsStore

Inherits From:	NSObject
Conforms To:	NSObject (NSObject)
Declared In:	WebObjects/WOStatisticsStore.h

Class Description

The WOStatisticsStore object records statistics about a WebObjects application while that application runs. All WebObjects applications have a WOStatisticsStore object, which you can access by sending **statisticsStore** to the WOApplication object.

Recording Information

The WOStatisticsStore object records the bulk of its statistics at the end of each cycle of the request-response loop. Specifically, at the end of WOSession's **appendToResponse:inContext:** method, the WOSession sends the **recordStatisticsForResponse:inContext:** message to the WOStatisticsStore. This message tells the WOStatisticsStore to begin recording statistics. Then, WOSession sends it a **descriptionForResponse:inContext:** message. This method sends the response component a **descriptionForResponse:inContext:** message. The default implementation of **descriptionForResponse: inContext:** in WOComponent returns the component's name.

You can override **descriptionForResponse:inContext:** in each of your components if you want to record more information. For example, you might want to record the values of all of the component's variables or perhaps just one or two key variables.

If you want to record extra information about the session, you can override WOStatisticsStore's **recordStatisticsForResponse:inContext:** method.

Maintaining a Log File

You can maintain an application log file by sending the message **setLogFile:rotationFrequencyInDays:** to the WOStatisticsStore object. When a log file has been specified, each session records information in the log file about the pages it accessed.

The log is maintained in Common Log File Format (CLFF) so that it can be analyzed by any standard CLFF-analysis tool. (For more information about the statistics recorded in the log file, see the **formatDescription:forResponse:inContext:** method description.) If a log file has been specified, the WOSession object keeps its own statistics about which pages it has accessed. When the session terminates, it writes this information to the log file.

Method Types

Recording information	
	– recordStatisticsForResponse:inContext:
	– descriptionForResponse:inContext:
	– setSessionMovingAverageSampleSize:
	– transactionMovingAverageSampleSize
Retrieving information	
	– statistics
Maintaining a CLFF log file	
	– setLogFile:rotationFrequencyInDays:
	– logFileRotationFrequencyInDays
	– logFile
Recording information in the CLFF log file	
	– formatDescription:forResponse:inContext:
	– logString:
Securing access to the WOStats page	
	– setPassword:
	– validateLogin:forSession:

Instance Methods

descriptionForResponse:inContext:

– (NSString *)**descriptionForResponse:**(WOResponse *)*aResponse*
inContext:(WOContext *)*aContext*

Records information about the current response by sending the **descriptionForResponse:inContext:** message to the response page and returning the result. This method is invoked at the end of the request-response loop in WOSession’s **appendToResponse:inContext:** method, after the **recordStatisticsForResponse:inContext:** method.

formatDescription:forResponse:inContext:

– (NSString *)**formatDescription:**(NSString *)*responseDescription*
forResponse:(WOResponse *)*aResponse*
inContext:(WOContext *)*aContext*

If log file recording is enabled, this method formats the string *responseDescription* in using Common Log File Format (CLFF). The resulting string contains:

- The host from which the HTTP request was received

- The name of the user that performed the request
- The current date
- The request’s HTTP method (GET or PUT)
- The WebObjects application name
- The result of the **descriptionForResponse:inContext:** method (by default, this method returns the response component’s name)
- The request’s HTTP version
- The HTTP status of the response
- The size of the response

You enable log file recording by setting a log file using the **setLogFile:rotationFrequencyInDays:** method.

This method is used by WOSession to record information about the current transaction when log file recording is enabled.

See also: – **logFile**, – **logString:**

logFile

– (NSString *)**logFile**

Returns the full path to the CLFF log file. This log file does not exist unless you send **setLogFile: rotationFrequencyInDays:** to the WOStatisticsStore.

See also: – **formatDescription:forResponse:inContext:**, – **logFileRotationFrequencyInDays**, – **logString:**

logFileRotationFrequencyInDays

– (double)**logFileRotationFrequencyInDays**

The number of days a log file lasts. That is, a log file’s contents are flushed after a certain time interval to ensure that it does not grow too large and a new log file is started. This method returns that time interval.

Before a new log file is started, the contents of the current log file are saved to a backup file. You can then inspect this log file and/or remove it when its data has grown stale.

See also: – **setLogFile:rotationFrequencyInDays:**

logString:

– (void)**logString:**(NSString *)*aString*

Writes the string *aString* to the CLFF log file specified by **logFile**. The method is used to record a session's statistics when that session ends. You can also use it to record any string to the log file that might be helpful to you.

See also: – **formatDescription:forResponse:inContext:**

sessionMovingAverageSampleSize

– (int)**sessionMovingAverageSampleSize**

Returns the sample size used to compute moving average statistics for each session. The WOStatisticsStore object uses this sample size to compute the response time for the last *n* transactions and the idle time between the last *n* transactions, where *n* is the number returned by this method. The default sample size is 10.

See also: – **setSessionMovingAverageSampleSize:**

recordStatisticsForResponse:inContext:

– (void)**recordStatisticsForResponse:**(WOResponse *)*aResponse* **inContext:**
(WOContext *)*aContext*

Records statistics for the current cycle of the request-response loop. This method is invoked at the end of WOSession's **appendToResponse:inContext:** method, immediately before the **descriptionForResponse: inContext:** method. By default, this method records the name of the response page for later use by **descriptionForResponse:inContext:**. You can override it if you want to record more information about the session before the current request and response are deallocated. You must begin your implementation by invoking the superclass method.

setLogFile:rotationFrequencyInDays:

– (void)**setLogFile:**(NSString *)*filePath* **rotationFrequencyInDays:**(double)*logRotation*

Sets the full path of the log file to which CLFF statistics will be recorded to *filePath*. The *logRotation* argument specifies the number of days statistics will be recorded to this log file. Every *logRotation* days, the contents of the current log file are saved to a backup file and a new log file is started.

The default is not to record information to a log file.

See also: – **logFile**, – **logFileRotationFrequencyInDays**

setSessionMovingAverageSampleSize:

– (void)**setSessionMovingAverageSampleSize:**(int)*aSize*

Sets the moving average sample size for each session to *aSize*. The WOStatisticsStore object uses this sample size to compute the response time for the last *aSize* transactions and the idle time between the last *aSize* transactions.

The default moving average session sample size is 10 transactions.

See also: – **sessionMovingAverageSampleSize**

setPassword:

– (void)**setPassword:**(NSString *)*aPassword*

Implements security for the WOStats page by setting its password to *aPassword*. By default, there is no password, so any user can access the WOStats page (provided they know the URL). If you implement this method, when you enter the WOStats URL, a login panel appears. You can leave the User name field blank; as long as you type the appropriate password in the password field, the WOStats page will appear.

See also: – **validateLogin:forSession:**

setTransactionMovingAverageSampleSize:

– (void)**setTransactionMovingAverageSampleSize:**(int)*aSize*

Sets the moving average sample size for each transaction to *aSize*. The WOStatisticsStore object uses this sample size to compute the response time for the last *aSize* transactions and the idle time between the last *aSize* transactions.

The default moving average transaction sample size is 100 transactions.

See also: – **transactionMovingAverageSampleSize**

statistics

– (NSDictionary *)**statistics**

Returns a dictionary containing the statistics that the WOStatisticsStore records.

The averages that are displayed by this method are not computed until this method is invoked. Therefore, invoking this method is costly and should not be done at every request.

transactionMovingAverageSampleSize

– (int)**transactionMovingAverageSampleSize**

Returns the sample size used to compute moving average statistics for each transaction. The WOStatisticsStore object uses this sample size to compute the response time for the last *n* transactions and the idle time between the last *n* transactions, where *n* is the number returned by this method. The default sample size is 100.

See also: – **setTransactionMovingAverageSampleSize:**

validateLogin:forSession:

– (BOOL)**validateLogin:**(NSString *)*string* **forSession:**(WOSession *)*aSession*

Returns YES if *string* is the password set by **setPassword:**, and NO otherwise. The password controls if the user can see the WOStats page.

EOEditingContext Additions

Inherits From:	NSObject
Declared In:	WebObjects/WODisplayGroup.h

Class Description

The WebObjects Framework adds one method to the Enterprise Objects Framework's EOEditingContext class that atempts to commit changes made in the receiver to its parent EOObjectStore.

Instance Methods

tryToSaveChanges

– (NSException *)**tryToSaveChanges**

Attempts to commit changes made in the receiver to its parent EOObjectStore by sending it the message **saveChangesInEditingContext:**. If the parent is an EOObjectStoreCoordinator, it guides its EOCooperatingObjectStores, typically EODatabaseContexts, through a multi-pass save operation (see the EOObjectStoreCoordinator class specification for more information). If no message handler or delegate is available and a database error occurs, an exception is raised that can be caught in WebScript; the error message indicates the nature of the problem.

WOActionResults

Adopted By:	WOComponent, WOResponse
Declared In:	WebObjects/WOResponse.h

Protocol Description

The WOActionResults protocol is the return type for direct actions. As a convenience, direct actions can return either WOComponent objects or WOResponse objects; both of which implement the WOActionResults protocol. This protocol implement only one method **generateResponse**.

If you want to return any other class from a direct action, then that class must implement this protocol.

generateResponse

– (WOResponse *)**generateResponse**

Returns a response object. WOResponse’s implementation of this method returns the receiver. WOComponent’s implementation of this method calls **appendToResponse:inContext:** on itself and all children components in its template and returns the result as a WOResponse object. If you want to return any other class from a direct action, then that class must implement this method.

WODisplayGroupDelegate

Adopted By: WODisplayGroup delegate objects

Declared In: WebObjects/WODisplayGroup.h

Protocol Description

WODisplayGroup offers a number of methods for its delegate to implement; if the delegate does implement them, the WODisplayGroup instances invoke them as appropriate. There are methods that inform the delegate that the EODisplayGroup has fetched, created an object (or failed to create one), inserted or deleted an object, changed the selection, or set a value for a property. There are also methods that request permission from the delegate to perform most of these same actions. The delegate can return YES to permit the action or NO to deny it. See each method's description for more information.

Instance Methods

displayGroup:createObjectFailedForDataSource:

– (void)**displayGroup:**(WODisplayGroup *)*aDisplayGroup*
  createObjectFailedForDataSource:(id)*aDataSource*

Invoked from **insertObject: atIndex:** to inform the delegate that *aDisplayGroup* has failed to create a new object for *aDataSource*. If the delegate doesn't implement this method, the WODisplayGroup fails silently.

displayGroup:didDeleteObject:

– (void)**displayGroup:**(WODisplayGroup *)*aDisplayGroup* **didDeleteObject:**(id)*anObject*

Informs the delegate that *aDisplayGroup* has deleted *anObject*.

displayGroup:didFetchObjects:

– (void)**displayGroup:**(WODisplayGroup *)*aDisplayGroup* **didFetchObjects:**(NSArray *)*objects*

Informs the delegate that *aDisplayGroup* has fetched *objects*.

displayGroup:didInsertObject:

– (void)**displayGroup:**(WODisplayGroup *)*aDisplayGroup* **didInsertObject:**(id)*anObject*

Informs the delegate that *aDisplayGroup* has inserted *anObject*.

displayGroup:didSetValue:forObject:key:

– (void)**displayGroup:**(WODisplayGroup *)*aDisplayGroup*
  didSetValue:(id)*value*
  forObject:(id)*anObject*
  key:(NSString *)*key*

Informs the delegate that *aDisplayGroup* has altered a property value of *anObject*. *key* identifies the property, and *value* is its new value.

displayGroup:displayArrayForObjects:

– (NSArray *)**displayGroup:**(WODisplayGroup *)*aDisplayGroup*
  displayArrayForObjects:(NSArray *)*objects*

Invoked from **updateDisplayedObjects**, this method allows the delegate to filter and sort *aDisplayGroup*'s array of objects to limit which ones get displayed. *objects* contains all of *aDisplayGroup*'s objects. The delegate should filter any objects that shouldn't be shown and sort the remainder, returning a new array containing this group of objects. You can use the NSArray methods **filteredArrayUsingQualifier:** and **sortedArrayUsingKeyOrderingArray:** to create the new array.

If the delegate doesn't implement this method, the WODisplayGroup uses its own qualifier and sort ordering to update the displayed objects array.

See also: – **displayedObjects**, – **qualifier**, – **sortOrderings**

displayGroupDidChangeDataSource:

– (void)**displayGroupDidChangeDataSource:**(WODisplayGroup *)*aDisplayGroup*

Informs the delegate that *aDisplayGroup*'s EODataSource (defined in the EOControl framework) has changed.

displayGroupDidChangeSelectedObjects:

– (void)**displayGroupDidChangeSelectedObjects:**(WODisplayGroup *)*aDisplayGroup*

Informs the delegate that *aDisplayGroup*’s selected objects have changed, regardless of whether the selection indexes have changed.

displayGroupDidChangeSelection:

– (void)**displayGroupDidChangeSelection:**(WODisplayGroup *)*aDisplayGroup*

Informs the delegate that *aDisplayGroup*’s selection has changed.

displayGroupShouldFetch:

– (BOOL)**displayGroupShouldFetch:**(WODisplayGroup *)*aDisplayGroup*

Allows the delegate to prevent *aDisplayGroup* from fetching. If the delegate returns YES, *aDisplayGroup* performs the fetch; if the delegate returns NO, *aDisplayGroup* abandons the fetch.

displayGroup:shouldChangeSelectionToIndexes:

– (BOOL)**displayGroup:**(WODisplayGroup *)*aDisplayGroup*
shouldChangeSelectionToIndexes:(NSArray *)*newIndexes*

Allows the delegate to prevent a change in selection by *aDisplayGroup*. *newIndexes* is the proposed new selection. If the delegate returns YES, the selection changes; if the delegate returns NO, the selection remains as it is.

displayGroup:shouldDeleteObject:

– (BOOL)**displayGroup:**(WODisplayGroup *)*aDisplayGroup* **shouldDeleteObject:***anObject*

Allows the delegate to prevent *aDisplayGroup* from deleting *anObject*. If the delegate returns YES, *anObject* is deleted; if the delegate returns NO, the deletion is abandoned.

displayGroup:shouldInsertObject:atIndex:

– (BOOL)**displayGroup:**(WODisplayGroup *)*aDisplayGroup*
shouldInsertObject:*anObject*
atIndex:(unsigned int)*anIndex*

Allows the delegate to prevent **redisplay** from inserting *anObject* at *anIndex*. If the delegate returns YES, *anObject* is inserted; if the delegate returns NO, the insertion is abandoned.

displayGroup:shouldRedisplayForChangesInEditingContext:

– (BOOL)**displayGroup:**(WODisplayGroup *)*aDisplayGroup*
shouldRedisplayForEditingContextChangeNotification:(NSNotification *)*aNotification*

Invoked whenever *aDisplayGroup* receives an EOObjectsChangedInEditingContextNotification, this method allows the delegate to suppress redisplay based on the nature of the change that has occurred. If the delegate returns YES, *aDisplayGroup* redisplays; if it returns NO, *aDisplayGroup* doesn’t.

See also: – **redisplay**

displayGroup:shouldRefetchForInvalidatedAllObjectsNotification:

– (BOOL)**displayGroup:**(WODisplayGroup *)*aDisplayGroup*
shouldRefetchForInvalidatedAllObjectsNotification:(NSNotification *)*aNotification*

Invoked whenever *aDisplayGroup* receives an EOInvalidatedAllObjectsInStoreNotification, this method allows the delegate to suppress the refetching of the invalidated objects. If the delegate returns YES, *aDisplayGroup* immediately fetches its objects. If the delegate returns NO, *aDisplayGroup* doesn’t immediately fetch, instead delaying until absolutely necessary.

See also: – **redisplay**