



WebObjects Framework

Java API Reference



Apple Computer, Inc.
© 1999 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except to make a backup copy of any documentation provided on CD-ROM.

The Apple logo is a trademark of Apple Computer, Inc. Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Macintosh, and WebObjects are trademarks of Apple Computer, Inc., registered in the United States and other countries. Enterprise Objects is a trademark of Apple Computer, Inc.

NeXT, the NeXT logo, OPENSTEP, Enterprise Objects Framework, Objective-C, and WEBSOCKET are trademarks of NeXT Software, Inc.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and/or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

ORACLE is a registered trademark of Oracle Corporation, Inc.

SYBASE is a registered trademark of Sybase, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Windows NT is a trademark of Microsoft Corporation.

All other trademarks mentioned belong to their respective owners.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD “AS IS,” AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

WebObjects

Package: com.apple.yellow.webobjects

Introduction

The WebObjects class hierarchy is rooted in the Foundation Framework's NSObject class. The bulk of the WebObjects framework consists of several related groups of classes as well as a few individual classes.

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 persist 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 and WOResponse, along with their parent class WOMessage, store essential data about HTTP requests and responses, such as header information, form values, HTTP version, host and page name, and session, context, and sender IDs. 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 an integral, reusable page (or portion of a page) for display in a web browser. WOElement declares the three request-handling methods: `takeValuesFromRequest`, `invokeActionForRequest`, and `appendToResponse`. 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

Package: com.apple.yellow.webobjects

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

Constructors

WOAdaptor

Obtaining attributes

doesBusyRunOnce

dispatchesRequestsConcurrently

port

Event registering

registerForEvents

unregisterForEvents

Running

runOnce

Constructors

WOAdaptor

```
public WOAdaptor(  
    String aName,  
    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).

CLASS WOAdaptor

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

See Also: `adaptorWithName` (WOApplication class)

Instance Methods

dispatchesRequestsConcurrently

```
public boolean dispatchesRequestsConcurrently()
```

Returns `true` if the adaptor is multi-threaded, `false` otherwise. If the adaptor is multi-threaded, the adaptor may dispatch requests to the application concurrently in separate threads.

See Also: `adaptorsDispatchRequestsConcurrently` (WOApplication class)

doesBusyRunOnce

```
public boolean doesBusyRunOnce()
```

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

port

```
public int port()
```

After the application's constructor has been called, `port` returns the port number on which this adaptor will listen. During execution of the application's constructor, this method returns the value of the `WOPort` user default (or the value of the `-WOPort` command-line option, if one was specified when the application was started).

See Also: `port` (WOApplication class)

CLASS WOAdaptor

registerForEvents

```
public void registerForEvents()
```

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

See Also: `runLoop` (WOApplication class)

runOnce

```
public void runOnce()
```

Invoked by the application's main loop

See Also: `doesBusyRunOnce`

unregisterForEvents

```
public void unregisterForEvents()
```

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

WOApplication

Inherits from: NSObject
Package: com.apple.yellow.webobjects

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.

CLASS WOApplication

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`, 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.

Method Types

Creating

`WOApplication`

Obtaining attributes

`adaptorsDispatchRequestsConcurrently`

`allowsConcurrentRequestHandling`

`isConcurrentRequestHandlingEnabled`

`baseURL`

`name`

`number`

`path`

Locking

`lock`

`unlock`

CLASS WOApplication

lockRequestHandling

unlockRequestHandling

Managing adaptors

adaptorWithName

adaptors

Managing sessions

setSessionStore

sessionStore

saveSessionForContext

restoreSessionWithID

createSessionForRequest

Managing pages

setPageCacheSize

pageCacheSize

permanentPageCacheSize

setPermanentPageCacheSize

setPageRefreshOnBacktrackEnabled

isPageRefreshOnBacktrackEnabled

pageWithName

pageWithName

Creating elements

dynamicElementWithName

Running

runLoop

run

setTimeout

timeout

CLASS WOApplication

defaultRequestHandler

Handling requests

appendToResponse

awake

createContextForRequest

createSessionForRequest

defaultRequestHandler

defaultRequestHandlerClassName

dispatchRequest

handlerForRequest

invokeActionForRequest

registeredRequestHandlerKeys

registerRequestHandler

removeRequestHandlerForKey

requestHandlerForKey

setDefaultRequestHandler

sleep

takeValuesFromRequest

Handling errors

handleSessionCreationErrorInContext

handlePageRestorationErrorInContext

handleSessionRestorationErrorInContext

handleException

Backward compatibility

requiresWOF35RequestHandling

requiresWOF35TemplateParser

CLASS WOApplication

Scripted class support

scriptedClassNameWithPath
scriptedClassNameWithPathEncoding

Script debugging

logString
debugString
printsHTMLParserDiagnostics
setPrintsHTMLParserDiagnostics
trace
traceAssignments
traceObjectiveCMessages
traceScriptedMessages
traceStatements
logTakeValueForDeclarationNamed
logSetValueForDeclarationNamed

Statistics report

setStatisticsStore
statisticsStore
statistics

Monitor support

monitoringEnabled
activeSessionsCount
refuseNewSessions
isRefusingNewSessions
setMinimumActiveSessionsCount
minimumActiveSessionsCount
terminateAfterTimeInterval

CLASS WOApplication

Resource manager support

setResourceManager

resourceManager

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

CLASS WOApplication

setMonitorHost
SMTPHost
setSMTPHost
adaptor
setAdaptor
port
setPort
listenQueueSize
setListenQueueSize
workerThreadCount
setWorkerThreadCount
additionalAdaptors
setAdditionalAdaptors
includeCommentsInResponses
setIncludeCommentsInResponses
componentRequestHandlerKey
setComponentRequestHandlerKey
directActionRequestHandlerKey
setDirectActionRequestHandlerKey
resourceRequestHandlerKey
setResourceRequestHandlerKey
sessionTimeout
setSessionTimeOut
garbageCollectionPeriod
setGarbageCollectionPeriod

Convenience Methods

sharedEditingContext

Constructors

WOApplication

```
public WOApplication()
```

Creates and 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 false).

A exception is thrown if initialization does not succeed.

Note: The global variable “WOApp” is initialized in this method.

Static Methods

adaptor

```
public static String adaptor()
```

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

See Also: `setAdaptor`

CLASS WOApplication

additionalAdaptors

```
public static NSArray additionalAdaptors()
```

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

See Also: setAdditionalAdaptors

application

```
public static WOApplication application()
```

Returns a WOApplication object.

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

applicationBaseURL

```
public static String 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

```
public static boolean autoOpenInBrowser()
```

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

CLASS WOApplication

cgiAdaptorURL

```
public static String 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

```
public static String componentRequestHandlerKey()
```

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

debugString

```
public static void debugString(String aFormatString)
```

Prints a message to the standard error device (`stderr`), if `WODebuggingEnabled` is `true`. The message can include formatted variable data using `String`'s concatenation feature.

You control whether this method displays output with the `WODebuggingEnabled` user default option. If `WODebuggingEnabled` is `true`, then the `debugString` messages display their output. If `WODebuggingEnabled` is `false`, the `debugString` messages don't display their output.

directActionRequestHandlerKey

```
public static String directActionRequestHandlerKey()
```

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

CLASS WOApplication

frameworksBaseURL

```
public static String 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

```
public static boolean 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

```
public static boolean 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

isDebuggingEnabled

```
public static boolean isDebuggingEnabled()
```

Returns whether or not debugging is enabled. If `true`, `debugString` prints out. Most startup-time status message are suppressed if this method returns `false`. By default, debugging is enabled. This is the cover method for the user default WODebuggingEnabled.

See Also: setDebuggingEnabled, debugString

CLASS WOApplication

isDirectConnectEnabled

```
public static boolean isDirectConnectEnabled()
```

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

See Also: `cgiAdaptorURL`

isMonitorEnabled

```
public static boolean isMonitorEnabled()
```

Returns whether or not the application can communicate with a Monitor application. It returns `true` if the application can contact Monitor upon startup and subsequently let Monitor gather statistics. It returns `false` if no communication 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

```
public static Number listenQueueSize()
```

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

See Also: `setListenQueueSize`

loadFrameworks

```
public static NSArray loadFrameworks()
```

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

See Also: `setLoadFrameworks`

CLASS WOApplication

logString

```
public static void logString(String aString)
```

Prints a message to the standard error device (stderr). The message can include formatted variable data using String's concatenation feature, for example:

```
int i = 500;
float f = 2.045;
WOApplication.logString("Amount = " + i + ", Rate = " + f + ", Total = " + i*f);
```

monitorHost

```
public static String monitorHost()
```

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

See Also: `setMonitorHost`, `isMonitorEnabled`

port

```
public static Number 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

```
public static 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`

CLASS WOApplication

recordingPath

```
public static String 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

```
public static String 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

```
public static Number 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`

CLASS WOApplication

setAdaptor

```
public static void setAdaptor(String anAdaptorName)
```

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

See Also: adaptor

setAdditionalAdaptors

```
public static 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

```
public static void setApplicationBaseURL(String 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

```
public static void setAutoOpenInBrowser(boolean isEnabled)
```

Controls whether starting up this application also launches a web browser. If *isEnabled* is *true*, the application launches the web browser. If *false*, 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 your subclass’s constructor.

See Also: autoOpenInBrowser

CLASS WOApplication

setCGIAdaptorURL

```
public static void setCGIAdaptorURL(String 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

```
public static void setCachingEnabled(boolean 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`

setComponentRequestHandlerKey

```
public static void setComponentRequestHandlerKey(String key)
```

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

See Also: `componentRequestHandlerKey`

setDebuggingEnabled

```
public static void setDebuggingEnabled(boolean flag)
```

Sets whether or not debugging is enabled. If `true`, `debugString` prints out. Most startup-time status message are suppressed if this method returns `false`. By default, debugging is enabled.

See Also: `isDebuggingEnabled`, `debugString`

CLASS WOApplication

setDirectActionRequestHandlerKey

```
public static void setDirectActionRequestHandlerKey(String key)
```

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

See Also: `directActionRequestHandlerKey`

setDirectConnectEnabled

```
public static void setDirectConnectEnabled(boolean 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 `true`, 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 `false`, 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`

CLASS WOApplication

setFrameworksBaseURL

```
public static void setFrameworksBaseURL(String 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

```
public static void setIncludeCommentsInResponses(boolean flag)
```

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

See Also: includeCommentsInResponses

setListenQueueSize

```
public static void setListenQueueSize(Number aListenQueueSize)
```

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

See Also: listenQueueSize

setLoadFrameworks

```
public static void setLoadFrameworks(NSArray frameworkList)
```

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

See Also: loadFrameworks

CLASS WOApplication

setMonitorEnabled

```
public static void setMonitorEnabled(boolean flag)
```

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

See Also: isMonitorEnabled

setMonitorHost

```
public static void setMonitorHost(String hostName)
```

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

See Also: monitorHost, isMonitorEnabled

setPort

```
public static void setPort(Number port)
```

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

See Also: port

setProjectSearchPath

```
public static void setProjectSearchPath(NSArray searchPath)
```

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

See Also: projectSearchPath

CLASS WOApplication

setRecordingPath

```
public static void setRecordingPath(String 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

```
public static void setResourceRequestHandlerKey(String key)
```

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

See Also: `resourceRequestHandlerKey`

setSessionTimeOut

```
public void setSessionTimeOut(Number aTimeOut)
```

Accessor to set the default session timeOut.

See Also: `sessionTimeout`

CLASS WOApplication

setSMTPHost

```
public static void setSMTPHost(String hostName)
```

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

See Also: SMTPHost

setWorkerThreadCount

```
public static void setWorkerThreadCount(Number 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

```
public static String 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

```
public static Number 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

```
public 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

```
public WOAdaptor adaptorWithName(  
    String aName,  
    NSDictionary someArguments)
```

Invoked during the constructor 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 created. 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`

CLASS WOApplication

adaptors

```
public 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

```
public boolean adaptorsDispatchRequestsConcurrently()
```

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

allowsConcurrentRequestHandling

```
public boolean allowsConcurrentRequestHandling()
```

Override to return `true` if concurrent request handling is allowed.

See Also: `isConcurrentRequestHandlingEnabled`

appendToResponse

```
public void appendToResponse(  
    WResponse aResponse,  
    WObjectContext aContext)
```

The WOApplication object sends this message to itself to initiate the last phase of request handling. This occurs right after the `invokeActionForRequest` 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`

CLASS WOApplication

awake

```
public 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

```
public String baseUrl()
```

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

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

See Also: `name`, `path`

createContextForRequest

```
public WOContext createContextForRequest(WORequest aRequest)
```

Creates a new context object for a given request. Override this method if you need to provide your own subclass of `WOContext`. If you override it, your implementation need not call `super`.

createSessionForRequest

```
public 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.

CLASS WOApplication

The method then returns an allocated and initialized (using the default WOSession constructor) session instance of the selected class. It throws 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`, `saveSessionForContext`

defaultRequestHandler

```
public 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.

defaultRequestHandlerClassName

```
public String defaultRequestHandlerClassName()
```

The default implementation of this method returns "WOComponentRequestHandler", which is the default request handler. Override this method to return "WODirectActionRequestHandler" to make the direct action request handler the default.

dispatchRequest

```
public WOResponse dispatchRequest(WORequest aRequest)
```

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

CLASS WOApplication

dynamicElementWithName

```
public WODynamicElement dynamicElementWithName(  
    String aName,  
    NSDictionary someAssociations,  
    WOElement anElement  
    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`, 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 a String 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` 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` first searches for a WODynamicElement named `aName`. If a WODynamicElement is found, the method creates an instance and returns it. Otherwise, it searches for a component—either scripted or compiled—to return instead. If neither are found, this method returns `null`.

garbageCollectionPeriod

```
public int garbageCollectionPeriod()
```

Returns the Java garbage collection period in seconds. This value can be set with the `WOGarbageCollectionPeriodKey` user default or with the `setGarbageCollectionPeriod` method.

CLASS WOApplication

handleException

```
public WResponse handleException(  
    Throwable anException,  
    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.

See Also: `handleSessionCreationErrorInContext`, `handleSessionRestorationErrorInContext`

handlePageRestorationErrorInContext

```
public WResponse 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`, `handleSessionCreationErrorInContext`, `handleSessionRestorationErrorInContext`

handleSessionCreationErrorInContext

```
public WResponse 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`, `handlePageRestorationErrorInContext`, `handleSessionRestorationErrorInContext`

CLASS WOApplication

handleSessionRestorationErrorInContext

```
public 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`, `handlePageRestorationErrorInContext`, `handleSessionCreationErrorInContext`

handlerForRequest

```
public WORequestHandler handlerForRequest(WORequest aRequest)
```

Returns the request handler used to handle a given request.

See Also: `registerRequestHandler`, `registeredRequestHandlerKeys`, `requestHandlerForKey`

invokeActionForRequest

```
public WOElement invokeAction(  
    WORequest aRequest,  
    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`

isConcurrentRequestHandlingEnabled

```
public boolean isConcurrentRequestHandlingEnabled()
```

Returns `true` if adaptors dispatch requests concurrently and `allowsConcurrentRequestHandling` has been overridden to allow concurrent request handling.

See Also: `allowsConcurrentRequestHandling`

CLASS WOApplication

isPageRefreshOnBacktrackEnabled

```
public boolean 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 `false` by default.

See Also: `setPageRefreshOnBacktrackEnabled`

isRefusingNewSessions

```
public boolean isRefusingNewSessions()
```

Returns `true` if the application instance is refusing new sessions, and `false` 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

```
public boolean isTerminating()
```

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

See Also: `setTimeout`, `defaultRequestHandler`, `terminateAfterTimeInterval`, `timeout`

lock

```
public void lock()
```

Locks the application object.

lockRequestHandling

```
public void lockRequestHandling()
```

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

CLASS WOApplication

logSetValueForDeclarationNamed

```
public void logSetValueForDeclarationNamed(  
    String aDeclarationName,  
    String aDeclarationType,  
    String aBindingName,  
    String anAssociationDescription,  
    Object aValue)
```

Formats and logs a message anytime a value is set through a WOAssociation, when WODebug is set to `true` 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` for a description of each of the arguments to this method.

logTakeValueForDeclarationNamed

```
public void logTakeValueForDeclarationNamed(  
    String aDeclarationName,  
    String aDeclarationType,  
    String aBindingName,  
    String anAssociationDescription,  
    Object aValue)
```

Formats and logs a message anytime a value is “taken” through a WOAssociation, when WODebug is set to `true` 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.

CLASS WOApplication

minimumActiveSessionsCount

```
public 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 `true`, the application instance terminates. The default is 0.

See Also: `activeSessionsCount`, `refuseNewSessions`, `setMinimumActiveSessionsCount`

monitoringEnabled

```
public boolean monitoringEnabled()
```

Returns `true` if the application is “monitorable” by the Monitor application, and `false` 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 ...
```

name

```
public String name()
```

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

See Also: `baseURL`, `path`

number

```
public String number()
```

Returns “-1”. This is provided for backwards compatibility only.

CLASS WOApplication

pageCacheSize

```
public int pageCacheSize()
```

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

See Also: setPageCacheSize

pageWithName

```
public WOComponent pageWithName(  
    String aName,  
    WORequest aRequest)
```

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

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

See Also: restorePageForContextID (WOSession), savePage (WOSession)

pageWithName

```
public WOComponent pageWithName(  
    String aName,  
    WOContext aContext)
```

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

See Also: pageWithName, restorePageForContextID (WOSession), savePage (WOSession)

CLASS WOApplication

path

```
public String 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

```
public 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)

printsHTMLParserDiagnostics

```
public boolean printsHTMLParserDiagnostics()
```

Returns whether the HTML parser prints diagnostic information to stdout when it encounters unbalanced HTML containers or other syntactically incorrect HTML. This method returns `false` by default.

See Also: `isDebuggingEnabled`, `debugString`

refuseNewSessions

```
public void refuseNewSessions(boolean flag)
```

Controls whether this application instance will create a session when it receives an HTTP request from a new user. If `flag` is `true`, 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 `false`, the application creates new sessions. `false` is the default.

CLASS WOApplication

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 true, the application terminates.

See Also: `activeSessionsCount`, `isRefusingNewSessions`, `minimumActiveSessionsCount`, `setMinimumActiveSessionsCount`

registerRequestHandler

```
public void registerRequestHandler(  
    WORequestHandler aHandler,  
    String 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

```
public NSArray registeredRequestHandlerKeys()
```

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

See Also: `handlerForRequest`, `requestHandlerForKey`

removeRequestHandlerForKey

```
public WORequestHandler removeRequestHandlerForKey(String aRequestHandlerKey)
```

Removes the specified request handler from the application.

See Also: `registerRequestHandler`, `requestHandlerForKey`

CLASS WOApplication

requestHandlerForKey

```
public WORequestHandler requestHandlerForKey(String key)
```

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

See Also: `handlerForRequest`, `registerRequestHandler`, `registeredRequestHandlerKeys`

requiresWOF35RequestHandling

```
public boolean 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 `true`. By default, it returns `false`.

requiresWOF35TemplateParser

```
public boolean 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 `true`. By default, it returns `false`.

resourceManager

```
public WOResourceManager resourceManager()
```

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

See Also: `setResourceManager`

CLASS WOApplication

restoreSessionWithID

```
public WOSession restoreSessionWithID(  
    String aSessionID,  
    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` method, but raises an exception if the `WOSessionStore` object is missing.

See Also: `createSessionForRequest`, `saveSessionForContext`

run

```
public 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`, `defaultRequestHandler`, `terminateAfterTimeInterval`

runLoop

```
public NSRunLoop runLoop()
```

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

saveSessionForContext

```
public 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 throws an exception if the `WOSessionStore` object is missing.

See Also: `restoreSessionWithID`

CLASS WOApplication

scriptedClassNameWithPath

```
public String scriptedClassNameWithPath(String 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.

scriptedClassNameWithPathEncoding

```
public String scriptedClassNameWithPathEncoding(  
    String aPath,  
    int 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

```
public WOSessionStore sessionStore()
```

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

See Also: `setSessionStore`

setDefaultRequestHandler

```
public void setDefaultRequestHandler(WORequestHandler aHandler)
```

Sets the default request handler.

See Also: `defaultRequestHandler`

CLASS WOApplication

setGarbageCollectionPeriod

```
public void setGarbageCollectionPeriod(int seconds)
```

Sets the Java garbage collection period to *seconds*. This period can also be set using the `WOGarbageCollectionPeriod` user default.

setMinimumActiveSessionsCount

```
public 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 `true`, 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

```
public void setPageCacheSize(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.”

See Also: `pageCacheSize`

CLASS WOApplication

setPageRefreshOnBacktrackEnabled

```
public void setPageRefreshOnBacktrackEnabled(boolean flag)
```

When *flag* is *true*, 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 *false*.) 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

```
public void setPermanentPageCacheSize(int aSize)
```

Sets the `permanentPageCacheSize` to `aSize`

See Also: `permanentPageCacheSize`

setPrintsHTMLParserDiagnostics

```
public void setPrintsHTMLParserDiagnostics(boolean flag)
```

Sets whether the HTML parser prints diagnostic information to stdout when it encounters unbalanced HTML containers or other syntactically incorrect HTML. This diagnostic information is not printed by default.

See Also: `setDebuggingEnabled`, `debugString`

CLASS WOApplication

setResourceManager

```
public 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

```
public 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

```
public void setStatisticsStore(WOStatisticsStore aStatisticsStore)
```

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

See Also: `statisticsStore`

setTimeout

```
public void setTimeout(double aTimeInterval)
```

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

CLASS WOApplication

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`

sharedEditingContext

```
public com.apple.yellow.eocontrol.EOSharedEditingContext sharedEditingContext()
```

This is a convenience method that returns the default shared editing context.

See Also: `EOSharedEditingContext` class description in the `EOControl Framework`

sleep

```
public 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

```
public NSDictionary 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

```
public WOStatisticsStore statisticsStore()
```

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

See Also: `setStatisticsStore`

CLASS WOApplication

takeValuesFromRequest

```
public void takeValuesFromRequest(  
    WORequest aRequest,  
    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`, `invokeActionForRequest`

terminate

```
public void terminate()
```

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

See Also: `isTerminating`, `setTimeout`

terminateAfterTimeInterval

```
public void terminateAfterTimeInterval(double 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.

CLASS WOApplication

timeOut

```
public double 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

```
public void trace(boolean flag)
```

If flag is `true`, prints all trace messages (messages for scripted messages, compiled messages, and all statements in the application) to the standard error device. If flag is `false`, stops printing all trace messages.

See Also: traceAssignments, traceObjectiveCMessages, traceScriptedMessages, traceStatements

traceAssignments

```
public void traceAssignments(boolean flag)
```

If flag is `true`, prints a message to the standard error device every time an assignment statement is executed. If flag is `false`, stops printing trace assignment messages.

See Also: trace, traceObjectiveCMessages, traceScriptedMessages, traceStatements

traceObjectiveCMessages

```
public void traceObjectiveCMessages(boolean flag)
```

If flag is `true`, prints a message to the standard error device every time a message is sent to a compiled class from Webscript. If flag is `false`, stops printing these messages.

See Also: trace, traceAssignments, traceScriptedMessages, traceStatements

CLASS WOApplication

traceScriptedMessages

```
public void traceScriptedMessages(boolean flag)
```

If *flag* is `true`, prints a message to the standard error device every time a message is sent to a scripted class from Webscript. If *flag* is `false`, stops printing trace scripted method messages.

See Also: `trace`, `traceAssignments`, `traceObjectiveCMessages`, `traceStatements`

traceStatements

```
public void traceStatements(boolean flag)
```

If *flag* is `true`, prints a message to the standard error device every time a statement in the application is executed from Webscript. If *flag* is `false`, stops printing trace statement messages.

See Also: `trace`, `traceAssignments`, `traceObjectiveCMessages`, `traceScriptedMessages`

unlock

```
public void unlock()
```

Unlocks the application object.

unlockRequestHandling

```
public void unlockRequestHandling()
```

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

WOAssociation

Inherits from: NSObject

Package: com.apple.yellow.webobjects

Class Description

The WOAssociation class 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};
```

CLASS WOAssociation

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.

Method Types

Creation

`associationWithKeyPath`

`associationWithValue`

Obtaining association attributes

`isValueConstant`

`isValueConstantInComponent`

`isValueSettable`

`isValueSettableInComponent`

Setting and retrieving value

`setValue`

`valueInComponent`

Static Methods

associationWithKeyPath

```
public static WOAssociation associationWithKeyPath(String 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 `null`, the value of the WOAssociation is also `null`.

See Also: `associationWithValue`

associationWithValue

```
public static WOAssociation associationWithValue(Object 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

```
public boolean isValueConstant()
```

Returns `true` if the WOAssociation's value is a constant, `false` otherwise.

See Also: `associationWithValue`, `isValueSettable`

isValueConstantInComponent

```
public boolean isValueSettableInComponent(WOComponent aComponent)
```

Returns `false` when the association is "constant." Use this for checking bindings at runtime.

See Also: `associationWithValue`, `isValueSettableInComponent`

isValueSettable

```
public boolean isValueSettable()
```

Returns `false` if the receiver's value is constant, `true` otherwise.

See Also: `associationWithKeyPath`, `isValueConstant`

isValueSettableInComponent

```
public boolean isValueSettableInComponent(WOComponent aComponent)
```

Returns `true` when the association is "constant." Use this for checking bindings at runtime.

See Also: `associationWithKeyPath`, `isValueConstant`

CLASS WOAssociation

setValue

```
public void setValue(  
    Object aValue,  
    WOComponent aComponent)
```

Finds the attribute of *aComponent* pointed to by the left-hand-side of the receiver and sets its value to *aValue*. This method throws an exception if the receiver's value is not settable. For example, sending a `setValue` 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

```
public Object 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.

CLASS WOAssociation

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`

WOComponent

Inherits from:	WOElement : NSObject
Implements:	WOActionResults, EOKeyValueCoding (com.apple.yellow.eocontrol), EOKeyValueCodingAdditions (com.apple.yellow.eocontrol)
Package:	com.apple.yellow.webobjects

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.

CLASS WComponent

WComponent objects also respond to `awake`, `sleep`, and the three request-handling messages: `takeValuesFromRequest`, `invokeActionForRequest`, and `appendToResponse`. 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 WComponent 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`, it forwards that message to its subcomponents

The WComponent 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.

Stateless Components

For extra efficiency, you can create **stateless** components: components that can be shared between sessions. Stateless components aren't replicated each time they're needed; rather, a single shared instance is referenced each time the component is used.

Stateless components cannot have state. They can have instance variables, but the variable's content must be transient. To ensure that when the shared instance of a component is reused by another session there are no side effects, reset your component's instance variables by implementing the `reset` method. In your implementation of `reset`, release and set to `null` each instance variable. Note that a stateless component's instance variables will remain valid for the duration of the phase (`takeValuesFromRequest`, `invokeAction` and `appendToResponse`); this lets you use instance variables in your stateless components to hold things analogous to items in a `WORepetition`.

Stateless components primarily save memory, but they can significantly speed up your application as well depending on how many stateless components you use in your application. To make a component stateless, override the component's `isStateless` method so that it returns `true`.

CLASS W OComponent

If a stateless component is needed simultaneously in separate threads, additional instances of the component are created (and later discarded) as necessary to prevent conflicts. Thus, the number of threads in which a component could be used determines the maximum number of instances of a stateless component that may be allocated at any given time.

Interfaces Implemented

WOActionResults

generateResponse

EOKeyValueCoding

handleQueryWithUnboundKey

handleTakeValueForUnboundKey

storedValueForKey

takeStoredValueForKey

takeValueForKey

unableToSetNullForKey

valueForKey

EOKeyValueCoding

takeValueForKeyPath

takeValuesFromDictionary

valueForKeyPath

valuesForKeys

Method Types

Creation

WOComponent

CLASS W OComponent

Obtaining attributes

application
baseURL
context
frameworkName
hasSession
name
pageWithName
path
session

Caching

isCachingEnabled
setCachingEnabled

Handling requests

appendToResponse
awake
ensureAwakeInContext
invokeActionForRequest
sleep
takeValuesFromRequest

Logging

debugString
isEventLoggingEnabled
logString

Template parsing

templateWithHTMLString:declarationString:languages:

CLASS WOComponent

Component statistics

descriptionForResponse

Invoking actions

parent

performParentAction

Synchronizing components

canGetValueForBinding

canSetValueForBinding

hasBinding

setValueForBinding

synchronizesVariablesWithBindings

valueForBinding

Other

isStateless

reset

templateWithName

validateTakeValueForKeyPath

Constructors

WOComponent

```
public WOComponent()
```

WebObjects Builder archive file exists in the component directory, it initializes component variables from this archive. This constructor throws exceptions if it cannot determine the name of the component or if it cannot initialize the object for any other reason. Override `WOComponent()` in compiled subclasses to perform custom initializations; as always, invoke `super`'s default constructor as the first thing.

Static Methods

debugString

```
public static void debugString(String aString)
```

Like `logString`, prints a message to the standard error device (`stderr`), but only prints the message if the `WODebuggingEnabled` user default option is `true`. If `WODebuggingEnabled` is `false`, the `debugString` messages aren't printed. See `logString` for information on the format of `aString`.

logString

```
public static void logString(String aString)
```

Prints a message to the standard error device (`stderr`). The message can include formatted variable data using `String`'s concatenation feature, for example:

```
int i = 500;
```

```
float f = 2.045;
```

```
WOComponent.logString("Amount = " + i + ", Rate = " + f + ", Total = " + i*f);
```

templateWithHTMLString:declarationString:languages:

```
public static WOElement templateWithHTMLString(  
    String anHTMLString,  
    String aDeclarationString,  
    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

```
public void appendToResponse(  
    WResponse aResponse,  
    WContext 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. WComponent's default implementation of this method forwards the message to the root WOEElement object of the component template. Compiled or scripted subclasses of WComponent can override this method to replace or supplement the default behavior with custom logic.

See Also: `invokeActionForRequest`, `takeValuesFromRequest`

application

```
public WApplication application()
```

Returns the WApplication object for the current application.

See Also: WApplication class, `context`, `session`

awake

```
public void awake()
```

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

See Also: `ensureAwakeInContext`, `sleep`

CLASS WOComponent

baseURL

```
public String baseURL()
```

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

See Also: `name`, `path`

canGetValueForBinding

```
public boolean canGetValueForBinding(String aBindingName)
```

Verifies that the binding exists and that `valueForBinding` will return a value.

See Also: `canSetValueForBinding`, `hasBinding`, `valueForBinding`

canSetValueForBinding

```
public boolean canSetValueForBinding(String aBindingName)
```

Verifies that the binding exists and that `setValueForBinding` will succeed.

See Also: `canGetValueForBinding`, `hasBinding`, `setValueForBinding`

context

```
public WOContext context()
```

Returns the `WOContext` object for the current transaction.

See Also: `WOContext` class, `application`, `session`

CLASS WOComponent

descriptionForResponse

```
public String descriptionForResponse(  
    WResponse aResponse,  
    WContext 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: WStatisticsStore class

ensureAwakeInContext

```
public void ensureAwakeInContext(WContext aContext)
```

Ensures that the receiver is awake in the specified context. Invoke this method before using a component which was stored in a variable. You don't need to invoke `ensureAwakeInContext` if the component was just created with `pageWithName`, if it was restored from the WebObjects page cache, or if the page will simply be returned as the result of an action. That is, you only need to invoke this method if you're going to send messages to a component that is otherwise not awakened. If the receiving component is already awake, this method has no effect.

See Also: awake

frameworkName

```
public String 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".

CLASS WOComponent

If the component is not stored in a framework, this method returns `null`.

See Also: `WOResourceManager` class

generateResponse

```
public abstract WOResponse generateResponse()
```

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

See Also: `generateResponse (WOResponse)`

hasBinding

```
public boolean hasBinding(String aBindingName)
```

Returns whether the component has a binding named `aBindingName`. This method traverses the chain of associations to the top-level parent, if necessary.

See Also: `canGetValueForBinding`, `canSetValueForBinding`

hasSession

```
public boolean 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`

CLASS WOComponent

invokeActionForRequest

```
public WOElement invokeAction(  
    WORequest aRequest,  
    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` 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. (Scripted subclasses must use the Objective-C form of this method: `invokeActionForRequest:inContext:`).

See Also: `appendToResponse`, `takeValuesFromRequest`

isCachingEnabled

```
public boolean isCachingEnabled()
```

Returns whether component-definition caching is enabled for this component. `false` is the default.

See Also: `setCachingEnabled`

isEventLoggingEnabled

```
public boolean isEventLoggingEnabled()
```

Called to determine if a component wants event logging. This is not desirable, for example, for components which are associated with event display as they would interfere with the actual event logging. The default implementation of this method returns `true`.

See Also: WOEvent class

CLASS WOComponent

isStateless

```
public boolean isStateless()
```

By default, this method returns `false`, indicating that state will be maintained for instances of the receiver. Overriding this method to return `true` will make the component stateless. A single instance of each stateless component is shared between multiple sessions, reducing your application's memory footprint.

See Also: `reset`

name

```
public String name()
```

Returns the name of the component, which includes a path of all directories under `DOCUMENTROOT/WebObjects` and is minus the `".wo"` extension; for example `"Main"` is a typical component name.

See Also: `baseUrl`, `path`

pageWithName

```
public WOComponent pageWithName(String aName)
```

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

See Also: `restorePageForContextID (WOSession)`, `savePage (WOSession)`

parent

```
public WOComponent parent()
```

Returns the parent component of the receiver.

CLASS WOComponent

path

```
public String 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`

performParentAction

```
public Object performParentAction(String 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:

```
public WOComponent click() {                /* this is the child's action */  
    selection = /* some value */;  
    /* now invoke the parent's action */  
    return performParentAction(callBack);  
}
```

CLASS W OComponent

reset

```
public void reset()
```

This method—which is only invoked if the component is stateless—allows a component instance to reset or delete temporary references to objects that are specific to a given context. To ensure that when the shared instance of a component is reused by another session there are no side effects, implement this method so that it releases and sets to `null` each of the component's instance variables.

See Also: `isStateless`

session

```
public 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

```
public void setCachingEnabled(boolean flag)
```

Enables or disables the caching of component definitions for the receiving component. `WComponent` 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 `true`, 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 `WComponent`; 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 `WComponent`'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`

CLASS WOComponent

setValueForBinding

```
public void setValueForBinding(  
    Object aValue,  
    String aBindingName)
```

Sets the value of the binding specified by *aBindingName* in the parent component to *aValue*. If the binding isn't settable, this method throws an exception.

See Also: `isValueSettableInComponent` (WOAssociation class)

sleep

```
public 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

```
public boolean 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 `true`. Override this method to create a non-synchronizing component.

See Also: `setValueForBinding`, `valueForBinding`

takeValuesFromRequest

```
public void takeValuesFromRequest(  
    WORequest aRequest,  
    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

CLASS WOComponent

scripted subclasses of Component can override this method to replace or supplement the default behavior with custom logic. (Scripted subclasses must use the Objective-C form of this method: `takeValuesFromRequest:inContext:`).

See Also: `appendToResponse`, `invokeActionForRequest`

templateWithName

```
public WOElement templateWithName(String 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 name: for example, “HelloWorld.” 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

```
public void validationFailedWithException(  
    Throwable exception,  
    Object value,  
    String 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.

validateTakeValueForKeyPath

```
public Object validateTakeValueForKeyPath(Object value,  
    String keyPath) throws com.apple.yellow.eocontrol.EOValidation.Exception
```

Validates (and coerces) the given value, assigning it if it is different than the current value. Throws a validation exception if `validateValueForKey` returns an exception. Returns the coerced (assigned) value.

CLASS WOComponent

valueForBinding

```
public Object valueForBinding(String 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 `null`. This cascading lookup makes it easy to provide default values for optional bindings.

See Also: `canGetValueForBinding`, `setValueForBinding`, `synchronizesVariablesWithBindings`

WOContext

Inherits from:	NSObject
Package:	com.apple.yellow.webobjects

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`, `invokeActionForRequest`, and `appendToResponse` 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.

Method Types

Constructors

WOContext

Creating new object instances

contextWithRequest

Obtaining attributes

component

contextID

elementID

hasSession

isInForm

page

request

response

session

session

setInForm

Manipulating element ID

appendElementIDComponent

appendZeroElementIDComponent

deleteAllElementIDComponents

deleteLastElementIDComponent

incrementLastElementIDComponent

CLASS WOContext

Generating URLs

```
directActionURLForActionNamed  
completeURLWithRequestHandlerKey  
componentActionURL  
urlWithRequestHandlerKey
```

Constructors

WOContext

```
public WOContext()
```

Returns a WOContext instance initialized with a unique context ID.

Static Methods

contextWithRequest

```
public static 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

```
public void appendElementIDComponent(String 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

```
public 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

```
public String completeURLWithRequestHandlerKey(  
    String requestHandlerKey,  
    String aRequestHandlerPath,
```

CLASS WOContext

```
String aQueryString,  
boolean isSecure,  
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 true, this method uses "https" instead of "http." If *somePort* is 0 (zero), this method uses the default port.

See Also: `urlWithRequestHandlerKey`

component

```
public 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, session

componentActionURL

```
public String componentActionURL()
```

Returns the complete URL for the component action.

contextID

```
public String contextID()
```

Returns the context ID of the receiver.

deleteAllElementIDComponents

```
public void deleteAllElementIDComponents()
```

Deletes all components of the current element ID.

See Also: `appendElementIDComponent`, `appendZeroElementIDComponent`, `incrementLastElementIDComponent`

CLASS WOContext

deleteLastElementIDComponent

```
public 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

```
public String directActionURLForActionNamed(  
    String anActionName,  
    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

```
public String elementID()
```

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

hasSession

```
public boolean hasSession()
```

Returns whether a session exists for the receiving context.

See Also: `session`

CLASS WOContext

incrementLastElementIDComponent

```
public 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

isInForm

```
public boolean isInForm()
```

Returns true when in the context of a WOForm.

See Also: setInForm

page

```
public WOComponent page()
```

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

See Also: component, request, response, session

request

```
public WORequest request()
```

Returns the transaction's WORequest object.

See Also: component, page, response, session

response

```
public WOResponse response()
```

Returns the transaction's WOResponse object.

See Also: component, page, response, session

CLASS WOContext

senderID

```
public String 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 `null`, as it always is on the first request of a session.

See Also: `request`, `uri` (**WORequest**)

session

```
public 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

```
public void setInForm(boolean flag)
```

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

See Also: `isInForm`

CLASS WOContext

urlWithRequestHandlerKey

```
public String urlWithRequestHandlerKey(  
    String requestHandlerKey,  
    String aRequestHandlerPath,  
    String 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 `"?"`.

See Also: `completeURLWithRequestHandlerKey`

WOCookie

Inherits from:	NSObject
Package:	com.apple.yellow.webobjects

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 the static method `cookieWithName`.

To add or remove cookies from the response, use the WOMessage 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.

CLASS WOCookie

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

Constructors

`WOCookie`

Creation

`cookieWithName`

Obtaining a cookie's attributes

`domain`

`expires`

`headerString`

`isSecure`

`name`

`path`

`value`

Setting a cookie's attributes

`setDomain`

`setExpires`

`setIsSecure`

`setName`

CLASS WOCookie

```
setPath  
setValue
```

Constructors

WOCookie

```
public WOCookie()
```

Creates and returns a new empty cookie. To set its attributes, use the appropriate `set` methods.

See Also: `cookieWithName`, `setDomain`, `setExpires`, `setIsSecure`, `setName`, `setPath`, `setValue`

Static Methods

cookieWithName

```
public static WOCookie cookieWithName(  
    String aName,  
    String aValue)
```

Creates and returns a cookie with just a name and its value. This method sets the path attribute to your application's path.

```
public static WOCookie cookieWithName(  
    String aName,  
    String aValue,  
    String aPath,  
    String aDomain,  
    NSDate expirationDate,  
    boolean 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: `domain`, `expires`, `isSecure`, `name`, `path`, `value`

Instance Methods

domain

```
public String domain()
```

Returns the value of the cookie's "domain" attribute. It's of the form "companyname.com".

expires

```
public 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 recent date in the past (see `setExpires` for more information).

headerString

```
public String 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.

CLASS WOCookie

isSecure

```
public boolean isSecure()
```

Returns the cookie's "secure" attribute. This attribute specifies whether the cookie should be transmitted only with secure HTTP. The default value is `false`.

name

```
public String 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

```
public String 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

```
public void setDomain(String aDomain)
```

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

See Also: `cookieWithName`

setExpires

```
public void setExpires(NSDate expirationDate)
```

Sets the cookie's "expires" attribute to *expirationDate*.

CLASS WOCookie

If you want to set the cookie's expiration date to some date in the distant past—for instance, in order to erase the cookie—don't use `NSDate.distantPast().distantPast` returns a date from the year 1 AD, and some browsers incorrectly interpret this as the year 2001. Instead, set the cookie's expiration date to an actual date in the recent past.

See Also: `cookieWithName`, `expires`

setIsSecure

```
public void setIsSecure(boolean flag)
```

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

See Also: `cookieWithName`

setName

```
public void setName(String aName)
```

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

See Also: `cookieWithName`

setPath

```
public void setPath(String aPath)
```

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

See Also: `cookieWithName`

setValue

```
public void setValue(String aValue)
```

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

See Also: `cookieWithName`

CLASS WOCookie

value

```
public String 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
Package:	com.apple.yellow.webobjects

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 adding 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 WODirectAction 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.

CLASS WODirectAction

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

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

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

Method Types

Constructors

`WODirectAction`

Obtaining attributes

`request`

Obtaining a session

`existingSession`

`session`

Obtaining a page

`pageWithName`

Performing an action

`performActionNamed`

Value assignment

`takeFormValueArraysForKeyArray`

CLASS WODirectAction

```
takeFormValuesForKeyArray
```

Debugging

```
debugString
```

```
logString
```

Constructors

WODirectAction

```
public WODirectAction(WORequest aWORequest)
```

Subclasses must override to provide any additional initialization.

Static Methods

debugString

```
public static void debugString(String aString)
```

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

logString

```
public static void logString(String aString)
```

Prints a message to the standard error device (stderr). The message can include formatted variable data using String's concatenation feature, for example:

```
int i = 500;
```

CLASS WODirectAction

```
float f = 2.045;
```

```
WOComponent.logString("Amount = " + i + ", Rate = " + f + ", Total = " + i*f);
```

Instance Methods

existingSession

```
public 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 `null`. To determine if a session failed to restore, check the request's session ID to see if it non-`null` and if so, call this method to check its result.

See Also: `session`

pageWithName

```
public WOComponent pageWithName(String aComponentName)
```

Returns the `WOComponent` with the specified name.

performActionNamed

```
public WOActionResults performActionNamed(String 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.

CLASS `WODirectAction`

request

```
public WORequest request()
```

Returns the `WORequest` object that initiated the action.

session

```
public 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 throws an exception.

See Also: `existingSession`

takeFormValueArraysForKeyArray

```
public void takeFormValueArraysForKeyArray(NSArray aKeyArray)
```

Performs `takeValueForKey` 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`

takeFormValuesForKeyArray

```
public void takeFormValuesForKeyArray(NSArray aKeyArray)
```

Performs `takeValueForKey` on the each key in `aKeyArray` using values from the receiver's request.

CLASS WODirectAction

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`:

WODisplayGroup

Inherits from:	NSObject
Implements:	NSCoding
Package:	com.apple.yellow.webobjects

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 `displayGroupDisplayArrayOfObjects`, 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 `true` to permit the action or `false` to deny it. See each method's description in the `WODisplayGroup.Delegates` interface specification for more information.

Method Types

Constructor

`WODisplayGroup`

Configuring behavior

`setFetchesOnLoad`

`fetchesOnLoad`

`setSelectsFirstObjectAfterFetch`

`selectsFirstObjectAfterFetch`

`setGlobalDefaultForValidatesChangesImmediately`

`globalDefaultForValidatesChangesImmediately`

`setValidatesChangesImmediately`

`validatesChangesImmediately`

Setting the data source

`setDataSource`

`dataSource`

Setting the qualifier and sort ordering

`setQualifier`

CLASS WODisplayGroup

qualifier
setSortOrderings
sortOrderings

Managing queries

qualifierFromQueryValues
queryMatch
queryMax
queryMin
queryOperator
allQualifierOperators
relationalQualifierOperators
setGlobalDefaultStringMatchFormat
globalDefaultStringMatchFormat
setDefaultStringMatchFormat
defaultStringMatchFormat
setGlobalDefaultStringMatchOperator
globalDefaultStringMatchOperator
setDefaultStringMatchOperator
defaultStringMatchOperator
qualifyDisplayGroup
qualifyDataSource
inQueryMode
setInQueryMode

Fetching objects from the data source

fetch

Getting the objects

allObjects

CLASS WODisplayGroup

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

clearSelection

selectNext

selectObjectsIdenticalTo

selectObjectsIdenticalToAndSelectFirstOnNoMatch

selectObject

selectPrevious

setSelectedObject

setSelectedObjects

CLASS WODisplayGroup

setSelectionIndexes

Examining the selection

selectionIndexes

selectedObject

selectedObjects

Inserting and deleting objects

insertObjectAtIndex

insertNewObjectAtIndex

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

Constructors

WODisplayGroup

```
public WODisplayGroup()
```

Creates and returns a new WODisplayGroup. The WODisplayGroup then needs to have an EODataSource (defined in EOControl) set with `setDataSource`.

Static Methods

globalDefaultForValidatesChangesImmediately

```
public static boolean globalDefaultForValidatesChangesImmediately()
```

Returns the class default controlling whether changes are immediately validated.

See Also: `validatesChangesImmediately`

globalDefaultStringMatchFormat

```
public static boolean globalDefaultStringMatchFormat()
```

Returns the default string match format for the class.

See Also: `defaultStringMatchFormat`

globalDefaultStringMatchOperator

```
public static boolean globalDefaultStringMatchOperator()
```

Returns the default string match operator for the class.

See Also: `defaultStringMatchOperator`

CLASS WODisplayGroup

setGlobalDefaultForValidatesChangesImmediately

```
public static void setGlobalDefaultForValidatesChangesImmediately(boolean flag)
```

Sets according to *flag* the class default controlling whether changes are immediately validated.

See Also: setValidatesChangesImmediately

setGlobalDefaultStringMatchFormat

```
public static void setGlobalDefaultStringMatchFormat(String format)
```

Sets the default string match format for the class.

See Also: setDefaultStringMatchFormat

setGlobalDefaultStringMatchOperator

```
public static void setGlobalDefaultStringMatchFormat(String operator)
```

Sets the default string match operator for the class.

See Also: setDefaultStringMatchOperator

Instance Methods

allObjects

```
public NSArray allObjects()
```

Returns all of the objects collected by the receiver.

See Also: displayedObjects, fetch

CLASS WODisplayGroup

allQualifierOperators

```
public NSArray allQualifierOperators()
```

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

See Also: queryOperator, relationalQualifierOperators

batchCount

```
public int 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

```
public boolean clearSelection()
```

Invokes setSelectionIndexes to clear the selection, returning true on success and false on failure.

currentBatchIndex

```
public int 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

CLASS `WODisplayGroup`

dataSource

```
public com.apple.yellow.eocontrol.EODataSource dataSource()
```

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

See Also: `hasDetailDataSource`, `setDataSource`

defaultStringMatchFormat

```
public String 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`, `globalDefaultStringMatchFormat`

defaultStringMatchOperator

```
public String 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`, `globalDefaultStringMatchOperator`

delegate

```
public Object delegate()
```

Returns the receiver's delegate.

See Also: `setDelegate`

CLASS `WODisplayGroup`

delete

```
public Object delete()
```

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

See Also: `deleteObjectAtIndex`

deleteObjectAtIndex

```
public boolean deleteObjectAtIndex(int index)
```

Attempts to delete the object at `index`, returning `true` if successful and `false` if not. Checks with the delegate using the method `displayGroupShouldDeleteObject`. If the delegate returns `false`, this method fails and returns `false`. If successful, it sends the delegate a `displayGroupDidDeleteObject` message.

This method performs the delete by sending `deleteObject` to the `EODDataSource` (defined in the `EOControl` framework). If that message raises an exception, this method fails and returns `false`.

See Also: `delete`, `deleteSelection`

deleteSelection

```
public boolean deleteSelection()
```

Attempts to delete the selected objects, returning `true` if successful and `false` if not.

See Also: `delete`, `deleteObjectAtIndex`

detailKey

```
public String 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.

CLASS `WODisplayGroup`

This method returns `null` 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`

```
public Object displayBatchContainingSelectedObject()
```

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

See Also: `displayNextBatch`, `displayPreviousBatch`, `setCurrentBatchIndex`

`displayedObjects`

```
public 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`, `displayGroupDisplayArrayForObjects` (**delegate method**)

`displayNextBatch`

```
public Object 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 `null` to force the page to reload.

See Also: `batchCount`, `currentBatchIndex`, `displayBatchContainingSelectedObject`, `displayPreviousBatch`

CLASS `WODisplayGroup`

`displayPreviousBatch`

```
public Object 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 `null` to force the page to reload.

See Also: `batchCount`, `currentBatchIndex`, `displayBatchContainingSelectedObject`, `displayNextBatch`

`fetch`

```
public Object fetch()
```

Attempts to fetch objects from the `EODDataSource` (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 `EODDataSource` to replace the object array, and if successful sends the delegate a `displayGroupDidFetchObjects` message.

This method returns `null` to force the page to reload.

See Also: `allObjects`, `updateDisplayedObjects`

`fetchesOnLoad`

```
public boolean fetchesOnLoad()
```

Returns `true` if the receiver fetches automatically after the component that contains it is loaded, `false` if it must be told explicitly to fetch. The default is `true`. 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`

CLASS `WODisplayGroup`

hasDetailDataSource

```
public boolean hasDetailDataSource()
```

Returns `true` if the display group's data source is an `EODetailDataSource` (defined in the `EOControl` framework), and `false` 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

```
public boolean hasMultipleBatches()
```

Returns `true` 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

```
public int 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

```
public int 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`

CLASS `WODisplayGroup`

inQueryMode

```
public boolean inQueryMode()
```

Returns `true` to indicate that the receiver is in query mode, `false` 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

```
public Object insert()
```

Invokes `insertNewObjectAtIndex` with an index just past the first index in the selection, or at the end if there’s no selection.

This method returns `null` to force the page to reload.

insertedObjectDefaultValues

```
public 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.

CLASS `WODisplayGroup`

`insertNewObjectAtIndex`

```
public Object insertNewObjectAtIndex(int index)
```

Asks the receiver's `EODDataSource` (defined in the `EOControl` framework) to create a new object by sending it a `createObject` message, then inserts the new object using `insertObjectAtIndex`. If a new object can't be created, this method sends the delegate a `displayGroupCreateObjectFailedForDataSource` message.

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

See Also: `insert`

`insertObjectAtIndex`

```
public void insertObjectAtIndex(  
    Object anObject,  
    int index)
```

Inserts *anObject* into the receiver's `EODDataSource` and displayed objects at the specified index, if possible. This method checks with the delegate before actually inserting, using `displayGroupShouldInsertObject`. If the delegate refuses, *anObject* isn't inserted. After successfully inserting the object, this method informs the delegate with a `displayGroupDidInsertObject` message, and selects the newly inserted object.

Raises an exception if *index* is out of bounds.

See Also: `insertNewObjectAtIndex`, `insert`

`masterObject`

```
public Object 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 `null`.

See Also: `detailKey`, `hasDetailDataSource`, `setMasterObject`

CLASS `WODisplayGroup`

`numberOfObjectsPerBatch`

```
public int numberOfObjectsPerBatch()
```

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

`qualifier`

```
public com.apple.yellow.eocontrol.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`

```
public com.apple.yellow.eocontrol.EOQualifier qualifierFromQueryValues()
```

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

See Also: `qualifyDataSource`, `qualifyDisplayGroup`

`qualifyDataSource`

```
public 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`

CLASS WODisplayGroup

qualifyDisplayGroup

```
public 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

```
public 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

```
public 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 a `String`. 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`

CLASS WODisplayGroup

queryMax

```
public 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

```
public 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

```
public 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

```
public void redisplay()
```

Sends out a contents changed notification.

CLASS `WODisplayGroup`

relationalQualifierOperators

```
public 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

```
public Object selectedObject()
```

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

See Also: `displayedObjects`, `selectionIndexes`, `selectedObjects`

selectedObjects

```
public NSArray selectedObjects()
```

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

See Also: `displayedObjects`, `selectionIndexes`, `selectedObject`

selectionIndexes

```
public NSArray selectionIndexes()
```

Returns the selection as an array of integers. The integers are indexes into the array returned by `displayedObjects`.

See Also: `selectedObject`, `selectedObjects`, `setSelectionIndexes`

selectNext

```
public Object selectNext()
```

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

CLASS `WODisplayGroup`

- 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 `null` to force the page to reload.

See Also: `selectPrevious`, `setSelectionIndexes`

selectObject

```
public boolean selectObject(Object anObject)
```

Attempts to select the object equal to *anObject* in the receiver's displayed objects array, returning `true` if successful and `false` 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

```
public boolean selectObjectsIdenticalTo(NSArray objectSelection)
```

Attempts to select the objects in the receiver's displayed objects array whose addresses are equal to those of objects, returning `true` if successful and `false` otherwise.

See Also: `setSelectionIndexes`, `selectObjectsIdenticalToAndSelectFirstOnNoMatch`

selectObjectsIdenticalToAndSelectFirstOnNoMatch

```
public boolean selectObjectsIdenticalToAndSelectFirstOnNoMatch(  
    NSArray objects,  
    boolean flag)
```

Selects the objects in the receiver's displayed objects array whose addresses are equal to those of *objects*, returning `true` if successful and `false` otherwise. If no objects in the displayed *objects* array match objects and *flag* is `true`, attempts to select the first object in the displayed objects array.

See Also: `setSelectionIndexes`, `selectObjectsIdenticalTo`

CLASS `WODisplayGroup`

`selectPrevious`

```
public Object 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 `null` to force the page to reload.

See Also: `selectNext`, `redisplay`

`selectsFirstObjectAfterFetch`

```
public boolean selectsFirstObjectAfterFetch()
```

Returns `true` if the receiver automatically selects its first displayed object after a fetch if there was no selection, `false` 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`

```
public void setCurrentBatchIndex(int 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).

CLASS `WODisplayGroup`

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

See Also: `batchCount`, `currentBatchIndex`, `displayBatchContainingSelectedObject`, `displayNextBatch`, `displayPreviousBatch`, `numberOfObjectsPerBatch`

setDataSource

```
public void setDataSource(com.apple.yellow.eocontrol.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

```
public void setDefaultStringMatchFormat(String format)
```

Sets how pattern matching will be performed on `String` values in the `queryMatch` dictionary. This format is used for properties listed in the `queryMatch` dictionary that have `String` 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 (this default can be overridden on a class basis using `setGlobalDefaultStringMatchFormat`). 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`, `setGlobalDefaultStringMatchFormat`

CLASS WODisplayGroup

setDefaultStringMatchOperator

```
public void setDefaultStringMatchOperator(String operator)
```

Sets the operator used to perform pattern matching for String values in the `queryMatch` dictionary. This operator is used for properties listed in the `queryMatch` dictionary that have String 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 (this default can be overridden on a class basis using `setGlobalDefaultStringMatchOperator`). The other possible value for this operator is `like`, which matches the case of the letters exactly.

See Also: `allQualifierOperators`, `defaultStringMatchOperator`, `relationalQualifierOperators`, `setDefaultStringMatchFormat`, `setGlobalDefaultStringMatchOperator`

setDelegate

```
public void setDelegate(Object anObject)
```

Sets the receiver's delegate to `anObject`.

See Also: `delegate`, `WODisplayGroup.Delegate`

setDetailKey

```
public void setDetailKey(String 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`

CLASS `WODisplayGroup`

setFetchesOnLoad

```
public void setFetchesOnLoad(boolean flag)
```

Controls whether the receiver automatically fetches its objects after being loaded. If *flag* is `true` it does; if *flag* is `false` the receiver must be told explicitly to fetch. The default is `false`. You can also set this behavior in WebObjects Builder in the Display Group Options panel.

See Also: `fetch`, `fetchesOnLoad`

setInQueryMode

```
public void setInQueryMode(boolean 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

```
public 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.

See Also: `insertedObjectDefaultValues`

CLASS `WODisplayGroup`

setMasterObject

```
public void setMasterObject(Object 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

```
public void setNumberOfObjectsPerBatch(int 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

```
public void setObjectArray(NSArray objects)
```

Sets the receiver's objects to *objects*, regardless of what its `EODDataSource` (defined in the EOControl framework) provides. This method doesn't affect the `EODDataSource`'s objects at all; specifically, it results in neither inserts nor deletes of objects in the `EODDataSource`. *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.

CLASS `WODisplayGroup`

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

```
public void setQualifier(com.apple.yellow.eocontrol.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 `displayGroupDisplayArrayForObjects`, that method is used instead of the qualifier to filter the objects.

See Also: `displayedObjects`, `qualifier`

setSelectedObject

```
public void setSelectedObject(Object anObject)
```

Sets the first selected object in the displayed objects array to *anObject*.

See Also: `displayedObjects`, `selectionIndexes`, `selectedObjects`

setSelectedObjects

```
public void setSelectedObjects(NSArray objects)
```

Sets the objects selected in the receiver's displayed objects array to *objects*.

See Also: `displayedObjects`, `selectionIndexes`, `selectedObject`

CLASS `WODisplayGroup`

setSelectionIndexes

```
public boolean setSelectionIndexes(NSArray selection)
```

Selects the objects at selection in the receiver's array if possible, returning `true` if successful and `false` if not (in which case the selection remains unaltered). *selection* is an array of Integers. 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 `displayGroupShouldChangeSelectionToIndexes` message. If the delegate returns `false`, this method also fails and returns `false`. If the receiver successfully changes the selection, its observers each receive a `subjectChanged` 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

```
public void setSelectsFirstObjectAfterFetch(boolean 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 `true` it does; if *flag* is `false` 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

```
public 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.

CLASS `WODisplayGroup`

If the receiver's delegate responds to `displayGroupDisplayArrayOfObjects`, that method is used instead of the sort orderings to order the objects.

See Also: `displayedObjects`, `sortOrderings`, `updateDisplayedObjects`

`setValidatesChangesImmediately`

```
public void setValidatesChangesImmediately(boolean 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 (although this default can be overridden on a class basis; see `setGlobalDefaultForValidatesChangesImmediately`).

See Also: – `saveChanges` (in `EOControl's EOEditingContext`), – `tryToSaveChanges` (`EOEditingContext Additions`), `validatesChangesImmediately`, `setGlobalDefaultForValidatesChangesImmediately`

`sortOrderings`

```
public 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`

```
public void updateDisplayedObjects()
```

Recalculates the receiver's displayed objects arrays and redisplay. If the delegate responds to `displayGroupDisplayArrayOfObjects`, 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`

CLASS `WODisplayGroup`

`validatesChangesImmediately`

```
public boolean validatesChangesImmediately()
```

Returns `true` 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`, `globalDefaultForValidatesChangesImmediately`

CLASS WODisplayGroup

WODynamicElement

Inherits from: WOElement : NSObject
Package: com.apple.yellow.webobjects

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 NSObject have associative capabilities through NSObject's implementation of the Key-Value Coding interface.

WODynamicElements must implement the default constructor 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:

CLASS `WODynamicElement`

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

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

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

```
public void appendToResponse(WOResponse r, WOContext c){
    if(!c.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.

Constructors

WODynamicElement

```
public WODynamicElement(  
    String aName,  
    NSDictionary associations,  
    WOElement anElement)
```

Returns a dynamic element identified by class *aName* and initialized with the objects in dictionary *associations*. 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 `null`, is the root object of a graph of WOElements associated with the dynamic element.

Typically, a key in the *associations* 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.

CLASS WODynamicElement

WOElement

Inherits from: NSObject

Package: com.apple.yellow.webobjects

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
- invokeActionForRequest
- appendToResponse

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.

CLASS WOElement

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

Constructors

WOElement

```
public WOElement()
```

Returns an initialized WOElement.

Instance Methods

appendToResponse

```
public void appendToResponse(  
    WResponse aResponse,  
    WContext 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 WResponse 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: WResponse class

CLASS WOElement

invokeActionForRequest

```
public WOElement invokeAction(  
    WRequest aRequest,  
    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 `null`.

See Also: WOContext class for a description of element IDs

takeValuesFromRequest

```
public void takeValuesFromRequest(  
    WRequest aRequest,  
    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: WRequest class for methods used to extract form data

WOEvent

Inherits from: EOEvent (EOControl framework)
Package: com.apple.yellow.webobjects

Class Description

WOEvent is a subclass of EOEvent (defined in the EOControl framework) that serves as the parent class for objects that gather information—such as duration—about various operations in WebObjects. You can see the results of this information gathering in your web browser by accessing a special “event display” page, and you can configure how the results are displayed by accessing a special “event setup” page. Both of these are accessed through special direct actions (WOEventDisplay and WOEventSetup, respectively). For example, if you’ve been running the ComponentElementsTour, the following URL will access the event display page:

<http://localhost/cgi-bin/WebObjects/ComponentElementsTour.woa/wa/WOEventDisplay>

CLASS WOEvent

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

Event Group	Logged Events
WOApplication Event	pageWithName
WOAssociation Event	valueForKeyPath, takeValueForKeyPath
WOComponent Event	takeValuesFromRequest, invokeAction, appendToResponse, awake, sleep
WOComponentReference Event	pushComponent

Instance Methods

comment

```
public String comment()
```

In the default implementation, this method returns the description of the “info” instance variable which is passed at log time. This method can be overridden by subclasses to provide information for the event display.

setComponentName

```
public void setComponentName(String componentName)
```

Sets the event’s component name to *componentName*. Event data can be grouped or aggregated according to the component name.

CLASS WOEvent

setPageName

```
public void setPageName(String pageName)
```

Sets the event's page name to *pageName*. Event data can be grouped or aggregated according to the page name.

signatureOfType

```
public Object signatureOfType(int type)
```

Returns a "signature" for the receiver of the specified type. These signatures are used to group or aggregate data on the WOEventDisplay page. WOEvent is able to generate signatures for the following types:

Type	Signature
EOBasicEventSignature	A combination of the event's type and the component name
WOComponentSignature	The component name
WOPageSignature	The page name
WOAssociationSignature	varies based upon the context

Override this method if you are creating a custom subclass of WOEvent and need to provide signatures for additional event types.

title

```
public String title()
```

The default implementation of this method returns the "title" value from the EventTypeDescriptions dictionary. This method is required for proper functioning of the event logging display.

CLASS WOEvent

WOHTTPConnection

Inherits from: NSObject

Package: com.apple.yellow.webobjects

Class Description

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

Constructors

WOHTTPConnection

```
public WOHTTPConnection(  
    String hostName,  
    int portNumber)
```

Returns a WOHTTPConnection instance initialized with the specified host name and port number.

Instance Methods

keepAliveEnabled

```
public boolean keepAliveEnabled()
```

Returns whether the socket will be left open after requests are sent.

See Also: `setKeepAliveEnabled`

readResponse

```
public WResponse readResponse()
```

Reads a response from the server and returns it as a `WResponse` object. This method blocks until the contents of the response have been fully received. Returns `null` if an error is detected while reading or interpreting the response.

`readResponse` sets the keep-alive enabled flag to `false` unless the response indicates that the connection should be held open.

See Also: `sendRequest`

sendRequest

```
public boolean sendRequest(WORequest aRequest)
```

Opens a socket connection to the server indicated by the receiver's host name and port number and writes `aRequest` to that socket. Returns `true` if the socket is being held open for a subsequent invocation of `sendRequest`, or `false` if it has been closed. Use the `setKeepAliveEnabled` method to control whether the socket is to be held open.

Throws an exception if the socket connection cannot be established.

See Also: `readResponse`

CLASS `WOHTTPConnection`

setKeepAliveEnabled

```
public void setKeepAliveEnabled(boolean flag)
```

Specifies according to *flag* whether the socket is to be left open after each request has been sent so that subsequent requests don't require the socket to be re-opened.

See Also: `keepAliveEnabled`, `readResponse`, `sendRequest`

CLASS WOHTTPConnection

WOLongResponsePage

Inherits from: WOComponent : WOElement : NSObject

Package: com.apple.yellow.webobjects

Class Description

WOLongReponsePage is an abstract subclass of WOComponent that spawns a separate thread in which to perform the action and returns a status page indicating that the request is being processed. Use WOLongReponsePage when a requested action will take a long time to complete (more than 5 seconds, say).

To use WOLongResponsePage, your long-running action should use WOComponent's `pageWithName` method to instantiate and return a component that is a subclass of WOLongResponsePage. This subclass should override the `performAction` method and perform the actual computation.

Note: If you access WebObjects framework objects from within your implementation of `performAction`, you must check out the session (using WOSessionStore's `checkoutSessionWithID` method) just before invoking the WebObjects framework method, and check it back in (using WOSessionStore's `checkInSessionForContext` method) just after.

Method Types

Performing the computation

`performAction`

CLASS WO LongResponsePage

Returning pages

```
cancelPageForStatus  
pageForException  
pageForResult  
refreshPageForStatus
```

Locking

```
lock  
unlock
```

Managing refresh

```
refresh  
refreshInterval  
setRefreshInterval  
setStatus
```

Managing cancellation

```
cancel  
isCancelled
```

Instance Methods

cancel

```
public WOComponent cancel()
```

Cancels the request. You should bind a cancel button on the refresh page to this method.

See Also: `isCancelled`

CLASS `WO LongResponsePage`

cancelPageForStatus

```
public WComponent cancelPageForStatus(Object status)
```

Returns the cancel page, which is displayed when the request is cancelled.

isCancelled

```
public boolean isCancelled()
```

Returns `true` if the request has been cancelled. The long running computation should check this value to see if it should abort.

See Also: `cancel`

lock

```
public void lock()
```

Locks the page.

See Also: `unlock`

pageForException

```
public WComponent pageForException(Throwable anException)
```

Returns the exception page, which is displayed when an exception occurs in `performAction`.

pageForResult

```
public WComponent pageForResult(Object aResult)
```

Returns the result page that is displayed when `performAction` completes.

CLASS WO LongResponsePage

performAction

```
public Object performAction()
```

Override this method to perform the requested long computation. Returns the result of that computation as an object.

refresh

```
public WComponent refresh()
```

Called by the WOMetaRefresh invokeAction callback (can also be called manually if the page is not self refreshing). This method either invokes `pageForException`, `pageForResult`, `refreshPageForStatus`, or `cancelPageForStatus` depending on the state of the long response.

refreshInterval

```
public double refreshInterval()
```

Returns the interval after which the refresh page is refreshed.

refreshPageForStatus

```
public WComponent refreshPageForStatus(Object status)
```

Returns the page that is displayed while the long-running computation is running. This page displays the current status of the computation.

setRefreshInterval

```
public void setRefreshInterval(double interval)
```

Sets the refresh interval.

See Also: `refreshInterval`

CLASS `WO LongResponsePage`

setStatus

```
public void setStatus(Object status)
```

Sets the status of the computation to *status*. The long computation should send this message periodically so that the refresh page reflects the status of the computation.

unlock

```
public void unlock()
```

Unlocks the page.

See Also: `lock`

WOMailDelivery

Inherits from: NSObject
Package: com.apple.yellow.webobjects

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` static. You cannot create one of your own.

Method Types

Obtaining an instance

```
sharedInstance
```

CLASS WOMailDelivery

Composing mail

```
composeComponentEmail  
composePlainTextEmail
```

Sending mail

```
sendEmail
```

Static Methods

sharedInstance

```
public static WOMailDelivery sharedInstance()
```

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

Instance Methods

composeComponentEmail

```
public String composeComponentEmail(  
    String sender,  
    NSArray destination,  
    NSArray ccAddresses,  
    String subject,  
    WOComponent aComponent,  
    boolean 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 `true`, the message is sent immediately.

CLASS `WMailDelivery`

composePlainTextEmail

```
public String composePlainTextEmail(  
    String sender,  
    NSArray destination,  
    NSArray ccAddresses,  
    String subject,  
    String message,  
    boolean 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

```
public void sendEmail(String mailString)
```

Sends *anEmail*, with *anEmail* being a String following the SMTP format. The `compose...Email` methods return such Strings and this method lets you modify those strings before sending them.

WOMessage

Inherits from: NSObject

Package: com.apple.yellow.webobjects

Class Description

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

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

Note: Headers are case-insensitive. WebObjects enforces the HTTP specification, but avoid mixing the case of header keys. See the HTTP specification or HTTP documentation for more information on the HTTP headers and 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 `WebObjects` string-constant names.

int Value	WebObjects Name	Notes
1	<code>NSASCIIStringEncoding</code>	0 through 127
2	<code>NSNEXTSTEPStringEncoding</code>	
3	<code>NSJapaneseEUCStringEncoding</code>	
4	<code>NSUTF8StringEncoding</code>	
5	<code>NSISOLatin1StringEncoding</code>	default
6	<code>NSSymbolStringEncoding</code>	
7	<code>NSNonLossyASCIIStringEncoding</code>	7-bit verbose ASCII to represent all unichars
8	<code>NSShiftJISStringEncoding</code>	
9	<code>NSISOLatin2StringEncoding</code>	
10	<code>NSUnicodeStringEncoding</code>	
11	<code>NSWindowsCP1251StringEncoding</code>	Cyrillic; same as <code>AdobeStandardCyrillic</code>
12	<code>NSWindowsCP1252StringEncoding</code>	Windows Latin1
13	<code>NSWindowsCP1253StringEncoding</code>	Windows Greek
14	<code>NSWindowsCP1254StringEncoding</code>	Windows Turkish
15	<code>NSWindowsCP1250StringEncoding</code>	Windows Latin2
21	<code>NSISO2022JPStringEncoding</code>	ISO 2022 Japanese encoding for electronic mail

Messages with XML Content

The `WOMessage` class contains three methods that allow you to construct and interpret messages whose content is formatted as XML. `appendContentDOMDocumentFragment` allows you to build up an XML message piece by piece. `setContentDOMDocument`, on the other hand, allows you to specify the message's content all at once. To obtain the content of a message that is formatted as XML, use `contentAsDOMDocument`.

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

Note that the XML parser is a Java package, and that `WOMessage` doesn't provide corresponding Objective-C versions of `appendContentDOMDocumentFragment`, `contentAsDOMDocument`, and `setContentDOMDocument`.

Method Types

Creation

`WOMessage`

Working with message headers

`headerForKey`

`headerKeys`

`headers`

`headersForKey`

`httpVersion`

`setHeader`

`setHeaders`

CLASS `WOMessage`

`setHTTPVersion`

Working with message content

`addCookie`

`appendContentCharacter`

`appendContentData`

`appendContentString`

`appendContentDOMDocumentFragment`

`content`

`contentAsDOMDocument`

`cookies`

`removeCookie`

`setContent`

`setContentDOMDocument`

Controlling content encoding

`defaultEncoding`

`setDefaultEncoding`

`contentEncoding`

`setContentEncoding`

Working with user info

`setUserInfo`

`userInfo`

Constructors

WOMessage

```
public WOMessage()
```

Returns an initialized `WOMessage` instance. The default string encoding is set to ISO Latin 1.

Static Methods

defaultEncoding

```
public static int defaultEncoding()
```

Returns the default character encoding used to construct a new `WOMessage` which initially is `NSISOLatin1`. For more information, see “Content Encodings”.

setDefaultEncoding

```
public static void setDefaultEncoding(int aStringEncoding)
```

Lets you specify the character encoding to be used by default when constructing a new `WOMessage`. For more information, see “Content Encodings”.

Instance Methods

addCookie

```
public void addCookie(WOCookie aCookie)
```

A convenience method that adds the specified `WOCookie` object to the message content.

See Also: `cookies`, `removeCookie`, `WOCookie` **class specification**

appendContentCharacter

```
public void appendContentCharacter(byte aChar)
```

Appends a single ASCII character (*aChar*) to the message's contents.

appendContentData

```
public void appendContentData(NSData dataObject)
```

Appends a data-encapsulating object (*dataObject*) to the message's contents.

appendContentString

```
public void appendContentString(String aString)
```

Appends a string to the content of the message's contents. 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 a browser can interpret them as HTML.

CLASS `WOMessage`

`appendContentDOMDocumentFragment`

```
public void  
    appendContentDOMDocumentFragment(org.w3c.dom.DocumentFragment aDocumentFragment)
```

Converts the supplied DOM document fragment to an XML string and appends it to the message's contents.

See Also: `contentAsDOMDocument`, `setContentDOMDocument`, [Messages with XML Content](#)

`content`

```
public 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 `WOMessage`'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` method, for example.

See Also: `setContent`, `setContentEncoding`

`contentAsDOMDocument`

```
public org.w3c.dom.Document contentAsDOMDocument()
```

Returns the content of the receiver as a DOM document object. Throws a `DOMParserException` if the DOM parser throws an exception.

See Also: `appendContentDOMDocumentFragment`, `setContentDOMDocument`

CLASS `WOMessage`

contentEncoding

```
public int contentEncoding()
```

Returns an integer representing the encoding used for the message's content. See "Content Encodings" in the class description for a mapped list of supported encodings and their `WOObjects` names. For responses, 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`.

The default string encoding is ISO Latin1.

See Also: `setContent`, `setContentEncoding`

cookies

```
public NSArray cookies()
```

A convenience method that returns an array of `WOCookie` objects to be included in the message (which is usually a `WOResponse`).

See Also: `addCookie`, `removeCookie`, `WOCookie` **class specification**

headerForKey

```
public String headerForKey(String 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 `null` if the message has no headers for the key.

See Also: `setHeader`

headerKeys

```
public NSArray headerKeys()
```

Returns an array of string keys associated with the receiver's HTTP headers. Returns `null` if there are no headers. You could easily test to see if a header is included by doing something similar to this:

CLASS `WOMessage`

```
ImmutableVector hKeys = aMessage.headerKeys();
    if (hKeys.contains("expires")) {
        // do something
    }
```

See Also: `setHeaders`

headers

```
public NSDictionary headers()
```

Returns the header dictionary with which the message was initialized.

headersForKey

```
public NSArray headersForKey(String aKey)
```

Returns *all* HTTP headers identified by *aKey*.

See Also: `setHeaders`

httpVersion

```
public String httpVersion()
```

Returns the version of HTTP used for the message (for example, "HTTP/1.0").

See Also: `setHTTPVersion`

removeCookie

```
public void removeCookie(WOCookie aCookie)
```

A convenience method that removes the specified `WOCookie` object from the message.

See Also: `cookies`, `removeCookie`, `WOCookie class specification`

CLASS `WOMessage`

setContent

```
public void setContent(NSData someData)
```

Sets the message contents to *someData*.

See Also: `content`

setContentDOMDocument

```
public void setContentDOMDocument(org.w3c.dom.Document aDocument)
```

Sets the XML content of the response to the DOM document *aDocument*.

See Also: `appendContentDOMDocumentFragment`, `contentAsDOMDocument`

setContentEncoding

```
public void setContentEncoding(int anEncoding)
```

Sets the encoding used for the message contents. See “Content Encodings” in the class description for a mapped list of supported encodings and their WebObjects names. The default string encoding is ISO Latin1.

See Also: `contentEncoding`

setHTTPVersion

```
public void setHTTPVersion(String aVersion)
```

Sets the version of HTTP used for the message (for example, "HTTP/1.0").

See Also: `httpVersion`

CLASS `WOMessage`

setHeader

```
public void setHeader(  
    String aHeader,  
    String 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", "content-type");
```

See Also: `headerForKey`

setHeaders

```
public void setHeaders(  
    NSArray headerList,  
    String aKey)
```

Appends *headerList* to the list of HTTP header in the receiver and associates, for retrieval, the HTTP key *aKey* with the list of header elements. If a header doesn't already exist for the receiver, one is created before the list of headers is appended.

See Also: `headerKeys`, `headersForKey`

setUserInfo

```
public void setUserInfo(NSDictionary aDictionary)
```

Sets a dictionary in the `WOMessage` object that, as a convenience, can contain any kind of information related to the current response. Objects further down the `appendToResponse` message "chain" can retrieve this information using `userInfo`.

CLASS `WOMessage`

`userInfo`

```
public 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` message "chain" can set this dictionary in the `WOMessage` object as a way to pass information to other objects.

See Also: `setUserInfo`

WOREdirect

Inherits from: WComponent : WOElement : NSObject

Package: com.apple.yellow.webobjects

Class Description

WOREdirect is a subclass of WComponent that may be used to force the user's browser to redirect to another URL. You should only return this component as a response to an action method and never use it in an declarations file directly. This component can be useful, for example, if you have an image map with both static and dynamic actions.

Instance Methods

setURL

```
public void setURL(String aURL)
```

Sets the URL to which the user's browser should be redirected to *aURL*.

url

```
public String url()
```

Returns the URL to which the user's browser will be redirected when this component is returned.

WORequest

Inherits from: WOMessage : NSObject
Package: com.apple.yellow.webobjects

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 an HTTP server in a request. Requests usually 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. As well, 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

CLASS WORequest

the request-response loop. The application supplies WORequest objects as arguments in the `takeValuesFromRequest` and `invokeActionForRequest` 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.

Note that WORequest inherits from WOMessage. Of particular interest are those WOMessage methods that allow you to access the request headers (`headerForKey`, `headerKeys`, `headers`, and `headersForKey`) and `content` and `contentAsDOMDocument`, which return the contents of the request.

Programmatically Creating WORequest Objects

As stated above, in most WebObjects applications WORequest objects are created for you; your application is more concerned with interpreting and responding to WORequest objects. However, it is possible to place two WebObjects applications in a peer-to-peer configuration and have them communicate using WORequest and WOResponse objects. In situations like these, your application will need to create the WORequest objects itself and send them to the peer application using WOHTTPConnection.

The methods declared directly on WORequest allow you to extract information from a WORequest object. WORequest inherits a number of methods from WOMessage, however, that allow you to programmatically specify the contents of a request. In particular, the `appendContent...` and `setContent` methods in the WOMessage class are designed to do this. For more information, see the WOMessage class specification.

Method Types

Constructors

WORequest

CLASS WORequest

Working with cookies

cookieValueForKey

cookieValues

cookieValuesForKey

Form values

defaultFormValueEncoding

formValueEncoding

formValueForKey

formValueKeys

formValues

formValuesForKey

isFormValueEncodingDetectionEnabled

Request handling

requestHandlerKey

requestHandlerPath

requestHandlerPathArray

Form Values

setDefaultFormValueEncoding

setFormValueEncodingDetectionEnabled

Obtaining attributes

adaptorPrefix

applicationName

applicationNumber

browserLanguages

isFromClientComponent

method

sessionID

uri

Constructors

WORequest

```
public WORequest(  
    String aMethod,  
    String aURL,  
    String anHTTPVersion,  
    NSDictionary someHeaders,  
    NSData aContent,  
    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 thrown.
- *aURL* must be a valid URL; if the URL is invalid, an exception is thrown.

If either argument is omitted, the constructor throws an exception.

The remaining arguments are optional; if you specify `null` for these, the constructor substitutes default values or initializes them to `null`. The *someHeaders* argument (if not `null`) should be a dictionary whose `String` 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.

See Also: `method`, `httpVersion` (`WOMessage` class), `content` (`WOMessage` class), `userInfo` (`WOMessage` class)

Instance Methods

adaptorPrefix

```
public String 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

```
public String 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

```
public 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`

CLASS WORequest

browserLanguages

```
public NSArray browserLanguages()
```

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

cookieValueForKey

```
public String cookieValueForKey(String aKey)
```

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

See Also: [cookieValues](#), [cookieValuesForKey](#), [WOCookie class specification](#)

cookieValues

```
public NSDictionary cookieValues()
```

Returns a dictionary of cookie values and cookie keys.

See Also: [cookieValueForKey](#), [cookieValuesForKey](#), [WOCookie class specification](#)

cookieValuesForKey

```
public NSArray cookieValuesForKey(String 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](#)

CLASS WORequest

defaultFormValueEncoding

```
public int 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

```
public int 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

```
public Object formValueForKey(String 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

```
public 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.

CLASS WORequest

formValues

```
public NSDictionary formValues()
```

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

formValuesForKey

```
public NSArray formValuesForKey(String aKey)
```

Returns an array of all values (as Strings) 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.

isFormValueEncodingDetectionEnabled

```
public boolean isFormValueEncodingDetectionEnabled()
```

Returns whether detection of form-value encoding is allowed to take place when form values are obtained.

See Also: `setFormValueEncodingDetectionEnabled`

isFromClientComponent

```
public boolean 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 HTML code in the response, you should check that the request is not from a client-side component before writing into the response.

CLASS WORequest

method

```
public String 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` (WOMessage class), `httpVersion` (WOMessage class)

requestHandlerKey

```
public String requestHandlerKey()
```

Returns the part of the request's URI which identifies the request handler. This identifies the request handle which will process the request and cannot be `null`

requestHandlerPath

```
public String 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

```
public NSArray requestHandlerPathArray()
```

Returns the request handler path decomposed into elements.

sessionID

```
public String sessionID()
```

Returns the session ID, or `null` 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.

CLASS WORequest

setDefaultFormValueEncoding

```
public void setDefaultFormValueEncoding(int 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

```
public void setFormValueEncodingDetectionEnabled(boolean 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

```
public String 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

WORequestHandler

Inherits from: NSObject

Package: com.apple.yellow.webobjects

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:

- WComponentRequestHandler, which handles requests for actions implemented in a component.
- WODirectActionRequestHandler, which handles requests for actions implemented in a WODirectAction class.
- WResourceRequestHandler, 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.

Instance Methods

handleRequest

```
public WResponse 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.

lock

```
public void lock()
```

Locks the WORequestHandler object.

unlock

```
public void unlock()
```

Unlocks the WORequestHandler object.

WOResourceManager

Inherits from: NSObject

Package: com.apple.yellow.webobjects

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

Method Types

Retrieving resources

`pathForResourceNamed`

`urlForResourceNamed`

Retrieving localized strings

`stringForKey`

Managing the application-wide data cache

`flushDataCache`

CLASS WOResourceManager

`removeDataForKey`

`setData`

Controlling access

`lock`

`unlock`

Instance Methods

flushDataCache

```
public 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`, `setData`

lock

```
public 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`

CLASS WOResourceManager

pathForResourceNamed

```
public String pathForResourceNamed(  
    String aResourceFile,  
    String aFrameworkName,  
    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 `null` 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`

removeDataForKey

```
public void removeDataForKey(  
    String key,  
    WOSession aSession)
```

Removes the data stored in the data cache under the key *key*. The session argument is currently ignored; specify `null` 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`, sent to the dynamic element, and then removed from the cache using `removeDataForKey`. If the *key* attribute is set, `removeDataForKey` 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`

CLASS WOResourceManager

setData

```
public void setData(  
    NSData someData,  
    String key,  
    String mimeType,  
    WOSession aSession)
```

Adds the image or embedded 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 `null` 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`, sends it to the dynamic element, and then removes it from the cache using `removeDataForKey`. 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

```
public String stringForKey(  
    String aKey,  
    String aTableName,  
    String aDefaultValue,  
    String aFrameworkName,  
    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).

CLASS WOResourceManager

unlock

```
public 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

```
public String urlForResourceNamed(  
    String aResourceFile,  
    String aFrameworkName,  
    NSArray languagesList,  
    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 `null` 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.

CLASS WOResourceManager

Access to the WOResourceManager object is locked at the beginning of this method and unlocked at the end.

See Also: pathForResourceNamed

WOResponse

Inherits from:	WOMessage : NSObject
Implements:	WOActionResults
Package:	com.apple.yellow.webobjects

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` 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 WComponent 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 used when 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.

The WOMessage class—from which WOResponse inherits—declares most of the methods you use when constructing a response. These methods can be divided into two groups, those that add to a response's HTML content and those that read and set HTTP information. To the

CLASS WOResponse

methods provided by WOMessage, the WOResponse class adds two methods that escape HTML (`appendContentHTMLAttributeValue` and `appendContentHTMLString`). For images and other binary data, use `appendContentData` (declared in the WOMessage class). You can obtain and set the entire content of the response with WOMessage's `content` and `setContent` methods. The following example shows a sequence of `appendContent...` messages that compose an HTTP "POST" message:

```
aResponse.appendContentString("<form method=\"POST\" action=\"");  
aResponse.appendContentHTMLAttributeValue(aContext.url());  
aResponse.appendContentCharacter('');  
aResponse.appendContentString(">");
```

The remaining WOResponse instance methods set and read the the HTTP status code. WOResponse also provides two class methods that allow you to escape string objects.

Interfaces Implemented

WOActionResults

- `generateResponse`

Method Types

Creation

- `WOResponse`

Working with HTTP status

- `setStatus`

- `status`

Working with HTML content

- `appendContentHTMLAttributeValue`

CLASS `WResponse`

```
appendContentHTMLString  
generateResponse  
stringByEscapingHTMLString  
stringByEscapingHTMLAttributeValue
```

Controlling Client Caching

```
disableClientCaching
```

Constructors

WResponse

```
public WResponse()
```

Returns an initialized `WResponse` instance. HTTP status is set to 200 (OK), client caching is enabled, and the default string encoding is made ISO Latin 1.

Static Methods

stringByEscapingHTMLAttributeValue

```
public static String stringByEscapingHTMLAttributeValue(String 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 as attribute values of a tag. The escaped characters are:

Character	Escaped character
&	&
"	"
\t		
\n	

\r	
<	<
>	>

stringByEscapingHTMLString

```
public static String stringByEscapingHTMLString(String 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:

Character	Escaped character
&	&
"	"
<	<
>	>

Instance Methods

appendContentHTMLAttributeValue

```
public void appendContentHTMLAttributeValue(String 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 "****".

See Also: `setContentEncoding` (WOMessage class)

CLASS WOResponse

appendContentHTMLString

```
public void appendContentHTMLString(String 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 "`<P>`".

See Also: `setContentEncoding` (WOMessage class)

disableClientCaching

```
public void disableClientCaching()
```

Attempts to disable caching in the client browser by appending a “no-cache” Cache-Control response directive to the HTTP response and by appending Expires and Date values that equal (they are both set to the current date and time).

This method shouldn't be invoked more than once for a given response.

generateResponse

```
public WOResponse generateResponse()
```

Returns a WOResponse object. WOResponse's implementation simply returns itself.

See Also: `generateResponse` (WOComponent)

setStatus

```
public void setStatus(int anInt)
```

Sets the HTTP status to *anInt*. Consult the HTTP specification or HTTP documentation for the significance of status integers.

See Also: `status`

CLASS `WOResponse`

status

```
public 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`

WOSession

Inherits from:	NSObject
Package:	com.apple.yellow.webobjects

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.

CLASS `WOSession`

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 convertible 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`, `invokeAction`, and `appendToResponse`) 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 `setObjectForKey` 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.

CLASS `WOSession`

- `WOSession` objects also allow you to affect load balancing with the `setDistributionEnabled` method; if the flag set by this method is `false` (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.

Method Types

Constructor

`WOSession`

Obtaining attributes

`domainForIDCookies`

`expirationDateForIDCookies`

`isDistributionEnabled`

`sessionID`

`storesIDsInCookies`

`storesIDsInURLs`

Setting attributes

`setDistributionEnabled`

`setStoresIDsInCookies`

`setStoresIDsInURLs`

Terminating

`terminate`

`isTerminating`

`timeOut`

`setTimeOut`

Localization

`languages`

CLASS WOSession

setLanguages

Managing component state

setObjectForKey

objectForKey

removeObjectForKey

Managing enterprise objects

defaultEditingContext

setDefaultEditingContext

Handling requests

appendToResponse

awake

context

invokeAction

sleep

takeValuesFromRequest

Statistics

statistics

Debugging

debugString

logString

Page Management

savePage

restorePageForContextID

Constructors

WOSession

```
public WOSession()
```

Returns an initialized WOSession object. Session time-out is set by default to a very long period. This method throws exceptions if no session ID has been assigned or if it cannot initialize the object for any other reason. The `isDistributionEnabled` flag is set to `false`, meaning that each transaction will be assigned to an application instance specified in a configuration file for load balancing

Static Methods

debugString

```
public static void debugString(String aFormatString)
```

Prints a message to the standard error device (`stderr`), if `WODebuggingEnabled` is `true`. The message can include formatted variable data using String's concatenation feature.

You control whether this method displays output with the `WODebuggingEnabled` user default option. If `WODebuggingEnabled` is `true`, then the `debugString` messages display their output. If `WODebuggingEnabled` is `false`, the `debugString` messages don't display their output.

logString

```
public static void logString(String aString)
```

Prints a message to the standard error device (`stderr`). The message can include formatted variable data using String's concatenation feature, for example:

```
int i = 500;  
float f = 2.045;
```

```
WOApplication.logString("Amount = " + i + ", Rate = " + f + ", Total = " + i*f);
```

Instance Methods

appendToResponse

```
public void appendToResponse(  
    WResponse aResponse,  
    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` and then `descriptionForResponse` 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: `invokeAction`, `takeValuesFromRequest`

awake

```
public 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`

CLASS WOSession

context

```
public WOContext context()
```

Returns the WOContext object for the current transaction.

See Also: WOContext class

defaultEditingContext

```
public com.apple.yellow.eocontrol.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's parent object store is initialized to the default parent object store.

domainForIDCookies

```
public String 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

```
public NSDate expirationDateForIDCookies()
```

Returns when session and instance ID cookies expire. By default, no expiration date is set and this method returns null. 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.

CLASS `WOSession`

- 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.

`invokeAction`

```
public WOElement invokeAction(  
    WORequest aRequest,  
    WOContext aContext)
```

`WOSession` objects receive this message during the middle phase of the request-response loop. During this phase, the `invokeAction` 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`, `takeValuesFromRequest`

`isDistributionEnabled`

```
public boolean 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`

CLASS WOSession

isTerminating

```
public boolean isTerminating()
```

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

See Also: terminate

languages

```
public 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' browser preferences unless explicitly set with `setLanguages`. For details, see "Localization" in the WebObjects programming topics.

See Also: setLanguages

objectForKey

```
public Object objectForKey(String key)
```

Returns an object stored in the session under a specific key.

See Also: setObjectForKey

removeObjectForKey

```
public void removeObjectForKey(String key)
```

Removes the object stored in the session under the specified key.

CLASS WOSession

restorePageForContextID

```
public WOComponent restorePageForContextID(String 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 `null` if restoration is impossible.

See Also: `savePage`

savePage

```
public 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

```
public 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 when 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

```
public String 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

```
public void  
    setDefaultEditingContext(com.apple.yellow.eocontrol.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

```
public void setDistributionEnabled(boolean 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

```
public 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`

setObjectForKey

```
public void setObjectForKey(Object anObject,  
    String key)
```

Stores an object within the session under a given key. 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

```
public void setStoresIDsInCookies(boolean 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

```
public void setStoresIDsInURLs(boolean flag)
```

Enables or disables the storing of session and instance IDs in URLs. By default, IDs are stored in URLs.

setTimeout

```
public void setTimeout(double 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`

CLASS WOSession

sleep

```
public 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

```
public 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` 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`

storesIDsInCookies

```
public boolean storesIDsInCookies()
```

Returns whether the cookie mechanism for storing session and instance IDs is enabled. The cookie mechanism is disabled by default.

storesIDsInURLs

```
public boolean storesIDsInURLs()
```

Returns whether the URL mechanism for storing session IDs and instance IDs is enabled. The URL mechanism is enabled by default.

CLASS `WOSession`

takeValuesFromRequest

```
public void takeValuesFromRequest(  
    WORequest aRequest,  
    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`, `invokeAction`

terminate

```
public void terminate()
```

Causes the session to terminate after the conclusion of the current request-response loop.

See Also: `isTerminating`

timeOut

```
public double timeOut()
```

Returns the timeout interval in seconds.

See Also: `setTimeout`

Notifications

WOSessionDidCreateNotification

```
public static final String WOSessionDidCreateNotification
```

Sent at the the end of the session initiation (including awake). The object of the notification is the session instance

WOSessionDidRestoreNotification

```
public static final String WOSessionDidRestoreNotification
```

Sent after the sesion is fully restored (including awake). The object of the notification is the session instance.

WOSessionDidTimeOutNotification

```
public static final String WOSessionDidTimeOutNotification
```

Sent when a session times out but before it is released. The session ID is the object of the notification.

WOSessionStore

Inherits from: NSObject

Package: com.apple.yellow.webobjects

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`, 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.

CLASS `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` 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
```

Saving and restoring a context

```
restoreSessionWithID
```

```
saveSessionForContext
```

Static Methods

serverSessionStore

```
public static 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 the coding constructor for the session (`WOSession(NSCoder)`) to restore session state from the archived data.

Instance Methods

checkInSessionForContext

```
public 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 proceed. This method is called by `WOApplication` to save the session even if the session was not previously checked out via `checkoutSessionWithID` (that is, the session is a new session which was just created and, therefore, not restored).

CLASS `WOSessionStore`

checkoutSessionWithID

```
public WOSession checkoutSessionWithID(  
    String aSessionID,  
    WORequest aRequest)
```

This method returns a session for *aSessionID* if one is stored. This method calls `restoreSessionWithID` (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

```
public WOSession restoreSessionWithID(  
    String aSessionID,  
    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

```
public 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 `NSCoding` interface method `encodeWithCoder` to save session state to archived data.

WOStatisticsStore

Inherits from: NSObject

Package: com.apple.yellow.webobjects

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` method, the WOSession sends the `recordStatisticsForResponse` message to the WOStatisticsStore. This message tells the WOStatisticsStore to begin recording statistics. Then, WOSession sends it a `descriptionForResponse` message. This method sends the response component a `descriptionForResponse` message. The default implementation of `descriptionForResponse` in WComponent returns the component's name.

You can override `descriptionForResponse` 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` method.

Maintaining a Log File

You can maintain an application log file by sending the message `setLogFile` 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` 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

Constructor

`WOStatisticsStore`

Recording information

`recordStatisticsForResponse`

`descriptionForResponse`

`setSessionMovingAverageSampleSize`

`transactionMovingAverageSampleSize`

Retrieving information

`statistics`

Maintaining a CLFF log file

`setLogFile`

`logFileRotationFrequencyInDays`

`logFile`

Recording information in the CLFF log file

`formatDescription`

CLASS WOStatisticsStore

logString

Securing access to the WOStats page

setPassword

validateLogin

Constructors

WOStatisticsStore

```
public WOStatisticsStore()
```

Returns an initialized WOStatisticsStore.

Instance Methods

descriptionForResponse

```
public String descriptionForResponse(  
    WOResponse aResponse,  
    WOContext aContext)
```

Records information about the current response by sending the `descriptionForResponse` 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` method, after the `recordStatisticsForResponse` method.

CLASS WOStatisticsStore

formatDescription

```
public String formatDescription(  
    String responseDescription,  
    WOResponse aResponse,  
    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` 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` method.

This method is used by `WOSession` to record information about the current transaction when log file recording is enabled.

See Also: `logFile`, `logString`

lock

```
public void lock()
```

Locks access to the `WOStatisticsStore` object.

CLASS WOStatisticsStore

logFile

```
public String logFile()
```

Returns the full path to the CLFF log file. This log file does not exist unless you send `setLogFile` to the `WOStatisticsStore`.

See Also: `formatDescription`, `logFileRotationFrequencyInDays`, `logString`

logFileRotationFrequencyInDays

```
public 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`

logString

```
public void logString(String 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`

recordStatisticsForResponse

```
public void recordStatisticsForResponse(  
    WResponse aResponse,  
    WOContext aContext)
```

Records statistics for the current cycle of the request-response loop. This method is invoked at the end of `WOSession`'s `appendToResponse` method, immediately before the `descriptionForResponse` method. By default, this method records the name of the response page

CLASS WOStatisticsStore

for later use by `descriptionForResponse`. 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.

sessionMovingAverageSampleSize

```
public 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`

setLogFile

```
public void setLogFile(  
    String filePath,  
    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`

setPassword

```
public void setPassword(String 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`

CLASS WOStatisticsStore

setSessionMovingAverageSampleSize

```
public 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

setTransactionMovingAverageSampleSize

```
public 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

```
public 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.

CLASS WOStatisticsStore

transactionMovingAverageSampleSize

```
public 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`

unlock

```
public void unlock()
```

Unlocks access to the WOStatisticsStore object.

validateLogin

```
public boolean validateLogin(  
    String string,  
    WOSession aSession)
```

Returns `true` if `string` is the password set by `setPassword`, and `false` otherwise. The password controls if the user can see the WOStats page.

WOActionResults

Implemented by: WComponent, WResponse

Package: com.apple.yellow.webobjects

Interface Description

The WOActionResults interface is the return type for direct actions. As a convenience, direct actions can return either WComponent objects or WResponse objects; both of which implement the WOActionResults protocol. This interface implements only one method: `generateResponse`.

If you want to return any other class from a direct action, then that class must implement this protocol.

Instance Methods

generateResponse

```
public abstract WResponse generateResponse()
```

Returns a response object. WResponse's implementation of this method returns the receiver. WComponent's implementation of this method calls `appendToResponse` on itself and all children components in its template and returns the result as a WResponse object. If you want to return any other class from a direct action, then that class must implement this method.

WODisplayGroup.Delegate

Implemented by: WODisplayGroup delegate objects

Package: com.apple.yellow.webobjects

Interface 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 `true` to permit the action or `false` to deny it. See each method's description for more information.

Instance Methods

displayGroupCreateObjectFailedForDataSource

```
public abstract void displayGroupCreateObjectFailedForDataSource(  
    WODisplayGroup aDisplayGroup,  
    Object aDataSource)
```

Invoked from `insertNewObjectAtIndex` 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.

displayGroupDidChangeDataSource

```
public abstract void displayGroupDidChangeDataSource(WODisplayGroup aDisplayGroup)
```

Informs the delegate that *aDisplayGroup*'s `EODDataSource` (defined in the `EOControl` framework) has changed.

displayGroupDidChangeSelectedObjects

```
public abstract void displayGroupDidChangeSelectedObjects(WODisplayGroup aDisplayGroup)
```

Informs the delegate that *aDisplayGroup*'s selected objects have changed, regardless of whether the selection indexes have changed.

displayGroupDidChangeSelection

```
public abstract void displayGroupDidChangeSelection(WODisplayGroup aDisplayGroup)
```

Informs the delegate that *aDisplayGroup*'s selection has changed.

INTERFACE `WODisplayGroup.Delegate`

`displayGroupDidDeleteObject`

```
public abstract void displayGroupDidDeleteObject(  
    WODisplayGroup aDisplayGroup,  
    Object anObject)
```

Informs the delegate that *aDisplayGroup* has deleted *anObject*.

`displayGroupDidFetchObjects`

```
public abstract void displayGroupDidFetchObjects(  
    WODisplayGroup aDisplayGroup,  
    NSArray objects)
```

Informs the delegate that *aDisplayGroup* has fetched *objects*.

`displayGroupDidInsertObject`

```
public abstract void displayGroupDidInsertObject(  
    WODisplayGroup aDisplayGroup,  
    Object anObject)
```

Informs the delegate that *aDisplayGroup* has inserted *anObject*.

`displayGroupDidSetValue`

```
public abstract void displayGroupDidSetValue(  
    WODisplayGroup aDisplayGroup,  
    Object value,  
    Object anObject,  
    String key)
```

Informs the delegate that *aDisplayGroup* has altered a property value of *anObject*. *key* identifies the property, and *value* is its new value.

INTERFACE `WODisplayGroup.Delegate`

`displayGroupDisplayArrayForObjects`

```
public abstract NSArray displayGroupDisplayArrayForObjects(  
    WODisplayGroup aDisplayGroup,  
    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`

`displayGroupShouldChangeSelectionToIndexes`

```
public abstract boolean displayGroupShouldChangeSelectionToIndexes(  
    WODisplayGroup aDisplayGroup,  
    NSArray newIndexes)
```

Allows the delegate to prevent a change in selection by `aDisplayGroup`. `newIndexes` is the proposed new selection. If the delegate returns `true`, the selection changes; if the delegate returns `false`, the selection remains as it is.

`displayGroupShouldDeleteObject`

```
public abstract boolean displayGroupShouldDeleteObject(  
    WODisplayGroup aDisplayGroup,  
    Object anObject)
```

Allows the delegate to prevent `aDisplayGroup` from deleting `anObject`. If the delegate returns `true`, `anObject` is deleted; if the delegate returns `false`, the deletion is abandoned.

INTERFACE `WODisplayGroup.Delegate`

`displayGroupShouldFetch`

```
public abstract boolean displayGroupShouldFetch(WODisplayGroup aDisplayGroup)
```

Allows the delegate to prevent *aDisplayGroup* from fetching. If the delegate returns `true`, *aDisplayGroup* performs the fetch; if the delegate returns `false`, *aDisplayGroup* abandons the fetch.

`displayGroupShouldInsertObject`

```
public abstract boolean displayGroupShouldInsertObject(  
    WODisplayGroup aDisplayGroup,  
    Object anObject,  
    int anIndex)
```

Allows the delegate to prevent `redisplay` from inserting *anObject* at *anIndex*. If the delegate returns `true`, *anObject* is inserted; if the delegate returns `false`, the insertion is abandoned.

`displayGroupShouldRedisplayForChangesInEditingContext`

```
public abstract boolean displayGroupShouldRedisplayForChangesInEditingContext(  
    WODisplayGroup aDisplayGroup,  
    NSNotification aNotification)
```

Invoked whenever *aDisplayGroup* receives an `EObjectsChangedInEditingContextNotification`, this method allows the delegate to suppress redisplay based on the nature of the change that has occurred. If the delegate returns `true`, *aDisplayGroup* redisplay; if it returns `false`, *aDisplayGroup* doesn't.

See Also: `redisplay`

INTERFACE WODisplayGroup.Delegate

displayGroupShouldRefetchForInvalidatedAllObjectsNotification

```
public abstract boolean displayGroupShouldRefetchForInvalidatedAllObjects(  
    WODisplayGroup aDisplayGroup,  
    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 *true*, *aDisplayGroup* immediately fetches its objects. If the delegate returns *false*, *aDisplayGroup* doesn't immediately fetch, instead delaying until absolutely necessary.

See Also: `redisplay`

Deprecated API

This file enumerates those WebObjects Framework classes and methods that have been deprecated and should no longer be used. Wherever possible, notes have been included to indicate what API should be used in place of the deprecated class or method.

WOApplication

context

```
public WOContext context()
```

Deprecated in WebObjects 4.0. Use `WOComponent`'s `context` method instead.

createSession

```
public WOSession createSession()
```

Deprecated in WebObjects 4.0 Use `createSessionForRequest` instead.

handleException

```
public WOResponse handleException(Throwable anException)
```

Deprecated in WebObjects 4.0. Use `handleException` instead.

OTHER REFERENCE

handlePageRestorationError

```
public WResponse handlePageRestorationError()
```

Deprecated in WebObjects 4.0. Use `handlePageRestorationErrorInContext` instead.

handleRequest

```
public WResponse handleRequest(WORequest aRequest)
```

Deprecated in WebObjects 4.0. Use `dispatchRequest` instead.

handleSessionCreationError

```
public WResponse handleSessionCreationError()
```

Deprecated in WebObjects 4.0. Use `handleSessionCreationErrorInContext` instead.

handleSessionRestorationError

```
public WResponse handleSessionRestorationError()
```

Deprecated in WebObjects 4.0. Use `handleSessionRestorationErrorInContext` instead.

logToMonitorString

```
public static void logToMonitorString(String aFormat)
```

Deprecated in WebObjects 4.5. New features in the Monitor application allow logging of information. The deprecated API does nothing.

pageWithName

```
public WComponent pageWithName(String aName)
```

Deprecated in WebObjects 4.0. Use `pageWithName` instead.

OTHER REFERENCE

pathForResourceNamed

```
public String pathForResourceNamed(  
    String aName,  
    String anExtension)
```

Deprecated in WebObjects 4.0. Use `WOResourceManager`'s implementation of `pathForResourceNamed` instead.

requiresWOF35RequestHandling

```
public boolean requiresWOF35RequestHandling()
```

Deprecated in WebObjects 4.5. Apps should be rewritten so that they don't require WebObjects 3.5 behavior.

requiresWOF35TemplateParser

```
public boolean requiresWOF35TemplateParser()
```

Deprecated in WebObjects 4.5. Apps should be rewritten so that they don't require WebObjects 3.5 behavior.

restorePageForContextID

```
public WOComponent restorePageForContextID(String contextID)
```

Deprecated in WebObjects 4.0. Use `WOSession`'s `restorePageForContextID` instead.

restoreSession

```
public WOSession restoreSession()
```

Deprecated in WebObjects 4.0. Use `restoreSessionWithID` instead.

OTHER REFERENCE

savePage

```
public void savePage(WOComponent aPage)
```

Deprecated in WebObjects 4.0. Use WOSession's implementation of `savePage` instead.

saveSession

```
public void saveSession(WOSession aSession)
```

Deprecated in WebObjects 4.0. Use `saveSessionForContext` instead.

session

```
public WOSession session()
```

Deprecated in WebObjects 4.0 Use WOComponent's `session` method instead.

stringForKeyInTable

```
public String stringForKeyInTable(  
    String aKey,  
    String aTableName,  
    String aDefaultValue)
```

Deprecated in WebObjects 4.0. Use WOREsourceManager's implementation of `stringForKey` instead.

urlForResourceNamed

```
public String urlForResourceNamed(  
    String aName,  
    String anExtension)
```

Deprecated in WebObjects 4.0. Use WOREsourceManager's implementation of `urlForResourceNamed` instead.

WOAssociation

setValue

```
public void setValue(Object aValue)
```

Deprecated in WebObjects 4.0. Use `setValue` instead.

value

```
public Object value()
```

Deprecated in WebObjects 4.0. Use `valueInComponent` instead.

WOComponent

pathForResourceNamed

```
public String pathForResourceNamed(  
    String aName,  
    String anExtension)
```

Deprecated in WebObjects 4.0. Use `WOResourceManager`'s implementation of `pathForResourceNamed` instead.

OTHER REFERENCE

stringForKeyInTable

```
public String stringForKeyInTable(  
    String aKey,  
    String aTableName,  
    String aDefaultValue)
```

Deprecated in WebObjects 4.0. Use WOResourceManager's implementation of `stringForKey` instead.

urlForResourceNamed

```
public String urlForResourceNamed(  
    String aResourceName,  
    String anExtension)
```

Deprecated in WebObjects 4.0. Use WOResourceManager's implementation of `urlForResourceNamed` instead.

WOContext

application

```
public WOApplication application()
```

Deprecated in WebObjects 4.0. Use WOApplication's `application` method instead.

isDistributionEnabled

```
public boolean isDistributionEnabled()
```

Deprecated in WebObjects 4.0. Use WOSession's implementation of `isDistributionEnabled` instead.

OTHER REFERENCE

setDistributionEnabled

```
public void setDistributionEnabled(boolean flag)
```

Deprecated in WebObjects 4.0. Use `WOSession`'s implementation of `setDistributionEnabled` instead.

url

```
public String url()
```

Deprecated in WebObjects 4.0. Use `componentActionURL` instead.

urlSessionPrefix

```
public String urlSessionPrefix()
```

Deprecated in WebObjects 4.0. There is no alternative since the URL session prefix doesn't mean anything in the current URL format.

WODisplayGroup

setSortOrdering

```
public void setSortOrdering(NSArray keySortOrderArray)
```

Deprecated in WebObjects 4.0. Renamed to `setSortOrderings`.

sortOrdering

```
public NSArray sortOrdering()
```

Deprecated in WebObjects 4.0. Renamed to `sortOrderings`.

WORequest

applicationHost

```
public String applicationHost()
```

Deprecated in WebObjects 4.0. See `java.net.InetAddress`.

contextID

```
public String contextID()
```

Deprecated in WebObjects 4.0. Use `WOContext`'s implementation of `contextID` instead.

pageName

```
public String pageName()
```

Deprecated in WebObjects 4.0. Use `WOComponent`'s `name` method instead.

senderID

```
public String senderID()
```

Deprecated in WebObjects 4.0. Use `WOContext`'s implementation of `senderID` instead.

WOResourceManager

pathForResourceNamedInFramework

```
public String pathForResourceNamedInFramework(  
    String aResourceName,  
    String aFrameworkName)
```

Deprecated in WebObjects 4.0. Use `pathForResourceNamed` instead.

urlForResourceNamedInFramework

```
public String urlForResourceNamedInFramework(  
    String aResourceName,  
    String aFrameworkName)
```

Deprecated in WebObjects 4.0. Use `urlForResourceNamed` instead.

WOSession

application

```
public WOApplication application()
```

Deprecated in WebObjects 4.0. Use `WOApplication`'s `application` method instead.

WOSessionStore

restoreSession

```
public WOSession restoreSession()
```

Deprecated in WebObjects 4.0. Use `restoreSessionWithID` instead.

saveSession

```
public void saveSession(WOSession aSession)
```

Deprecated in WebObjects 4.0. Use `saveSessionForContext` instead.

WOStatisticsStore

validateLogin

```
public boolean validateLogin(String aString)
```

Deprecated in WebObjects 4.0. Use `validateLogin` instead.

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software.

Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER

Greg Wilson

PRODUCTION EDITOR

Gerri Gray