The Foundation Framework

Package: com.webobjects.foundation

Introduction

The Foundation Framework defines a base layer of classes written in Java. In addition to providing a set of useful primitive object classes, it introduces several paradigms that define functionality not covered by the Java language. The Foundation Framework is designed with these goals in mind:

- Provide a small set of basic utility classes
- Simplifies software development by introducing consistent conventions for things such as notifications, object persistence, key-value coding, and validation.
- Provide a level of OS independence, to enhance portability

This version of the Foundation framework is similar to the WebObjects 4.5 Foundation framework (com.apple.yellow.foundation) but does not rely on the Java Bridge because it is written in pure Java. The API for the pure Java Foundation also follows conventions in Sun's API more closely than the WebObjects 4.5 Foundation.

The pure Java Foundation resembles the WebObjects 4.5 Java Client Foundation (com.apple.client.foundation) but provides a larger set of functions.

Foundation Framework Classes

The Foundation Framework consists of several related groups of classes as well as a few individual classes:

- Data storage. NSData provides object-oriented storage for arrays of bytes. NSArray, NSDictionary, and NSSet provide storage for objects of any class.

- Dates and times. The NSTimestamp and NSTimeZone classes store times and dates. They offer methods for calculating date and time differences, for displaying dates and times in many formats, and for adjusting times and dates based on location in the world. The NSTimestampFormatter class converts dates to user-presentable strings and back.
- Application coordination and timing. NSNotification and NSNotificationCenter provide systems that an object can use to notify all interested observers of changes that occur. NSDelayedCallbackCenter coordinates events.
- Object distribution and persistence. The data that an object contains can be represented in an architecture-independent way using NSCoder and its subclasses, which also stores class information along with the data. The resulting representations are used for archiving and object distribution.
- Object disposal. The NSDisposable interface together with the NSDisposableRegistry ensure that unused objects are collected by Java's garbage collector.
- Key-value coding. The NSKeyValueCoding and NSKeyValueCodingAdditions interfaces along with their support classes provide a consistent way for objects to receive and return values for keys.
- Validation. The NSValidation interface and support classes define and implement a consistent validation mechanism.
- Locking of objects. The NSLock, NSRecursiveLock, and NSMultiReaderLock classes together with the NSLocking interface coordinate the locking of objects or graphs of objects.
- Operating system services. Several classes are designed to insulate you from the idiosyncracies of various operating systems. NSPathUtilities provides a consistent interface for working with file system paths. NSBundle accesses the application's resources.
- Other utility classes. NSRange specifies a range of values. NSComparator defines inequality relationships between objects for sorting. NSUndoManager manages an application's undo function. NSForwardException wraps exceptions into a subclass of Java's RuntimeException. NSPropertyListSerialization converts between property lists and byte arrays.

NSArray

Inherits from:	Object
Implements:	Cloneable java.io.Serializable NSCoding NSKeyValueCoding NSKeyValueCodingAdditions
Package:	com.webobjects.foundation

Class Description

NSArray and its subclass NSMutableArray manage collections of objects called *arrays*. NSArray creates static arrays and NSMutableArray creates dynamic arrays.

Table 0-1 describes the NSArray methods that provide the basis for all NSArray's other methods; that is, all other methods are implemented in terms of these three. If you create a subclass of NSArray, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-1 NSArray's Base API

Method	Description
`count`	Returns the number of elements in the array.
`objectAtIndex`	Provides access to the array elements by index.
`objectsNoCopy`	Returns a natural language array containing the NSArray's objects.

The methods `objectEnumerator` and `reverseObjectEnumerator` grant sequential access to the elements of the array, differing only in the direction of travel through the elements. These methods are provided so that arrays can be traversed in a manner similar to that used for objects of other collection classes in both the Java API and the Foundation Kit, such as java.util.Hashtable or NSDictionary. See the `objectEnumerator` method description for a code excerpt that shows how to use these methods to access the elements of an array.

NSArray provides methods for querying the elements of the array. `indexOfObject` searches the array for the object that matches its argument. To determine whether the search is successful, each element of the array is sent an `equals` message. Another method, `indexOfIdenticalObject`, is provided for the less common case of determining whether a specific object is present in the array. `indexOfIdenticalObject` tests each element in the array to see its the exact same instance as the argument.

To act on the array as a whole, a variety of other methods are defined. You can extract a subset of the array (`subarrayWithRange`) or concatenate the elements of an array of Strings into a single string (`componentsJoinedByString`). In addition, you can compare two arrays using the `isEqualToArray` and `firstObjectCommonWithArray` methods. Finally, you can create new arrays that contain the objects in an existing array and one or more additional objects with `arrayByAddingObject` and `arrayByAddingObjectsFromArray`.

Operators

An NSArray works with NSArray.Operators to perform operations on the array's elements. By default, an array has operators defined for the following keys:

Key	Operator Description
`count`	Returns the number of elements in an array.
`max`	Returns the element in the array with the highest value.
`min`	Returns the element in the array with the lowest value.
`avg`	Returns the average of the array's elements' values.
`sum`	Returns the sum of the array's element's values.

To compute an operation on an array's elements, you use key-value coding methods with a specially formatted key. The character "@" introduces the name of the operator you want to perform. For example, to compute the average salary of an array's elements, you could use the method `valueForKeyPath` with "@avg.salary" as the key path. For more information, see the NSArray.Operator interface specification.

If you write your own operator class, you can make it available for use with NSArrays with the method `setOperatorForKey`. The `operatorNames` method returns the keys for the operators that NSArray knows about, and `operatorForKey` returns the operator for a specified key.

Constants

NSArray defines the following constants:

Constant	Type	Description
AverageOperatorName	String	A key representing the operator (an NSArray.Operator) that computes the average of the elements in an array.
CountOperatorName	String	A key representing the operator (an NSArray.Operator) that computes the number of elements in an array.
NotFound	int	Returned in the place of an index when an object is not found in an array. For example, indexOfObject returns NotFound if none of the receiver's objects are equal to the specified object.
MaximumOperatorName	String	A key representing the operator (an NSArray.Operator) that computes the largest element in an array.
MinimumOperatorName	String	A key representing the operator (an NSArray.Operator) that computes the smallest element in an array.
EmptyArray	NSArray	An empty array, which can be shared to save memory.
SumOperatorName	String	A key representing the operator (an NSArray.Operator) that computes the sum of the elements in an array.

Interfaces Implemented

Cloneable

- clone

java.io.Serializable

NSCoding

- decodeObject
- classForCoder
- encodeWithCoder

NSKeyValueCoding

- takeValueForKey
- valueForKey

NSKeyValueCodingAdditions

- takeValueForKeyPath
- valueForKeyPath

Method Types

Creating arrays

- NSArray
- immutableClone
- mutableClone
- arrayByAddingObject

arrayByAddingObjectsFromArray

sortedArrayUsingComparator

subarrayWithRange

Querying the array

containsObject

count

getObjects

indexOfObject

indexOfIdenticalObject

lastObject

objectAtIndex

objects

objectsNoCopy

objectEnumerator

reverseObjectEnumerator

vector

Comparing arrays

firstObjectCommonWithArray

isEqualToArray

Working with string elements

componentsJoinedByString

componentsSeparatedByString

Operations

operatorForKey

operatorNames

setOperatorForKey

removeOperatorForKey

Methods inherited from Object

equals

hashCode

toString

Sending messages to elements

makeObjectsPerformSelector

Constructors

NSArray

```
public NSArray()
```

Creates an empty, immutable array. After an immutable array has been initialized in this way, it can't be modified. If you need an empty, immutable array, use `EmptyArray` instead. This method is used by mutable subclasses of NSArray.

```
public NSArray(NSArray anArray)
```

Creates an array containing the objects in `anArray`. After an immutable array has been initialized in this way, it can't be modified.

```
public NSArray(Object anObject)
```

Creates an array containing the single element `anObject`. After an immutable array has been initialized in this way, it can't be modified. Throws an IllegalArgumentException if `anObject` is `null`.

```
public NSArray(Object[] objects)
```

Creates an array containing `objects`. Ignores any `null` values it encounters in `objects`. After an immutable array has been initialized in this way, it can't be modified.

```
public NSArray(
    Object[] objects,
    NSRange aRange)
```

Creates an array containing the objects from objects in the range specified by aRange. Ignores any null values it encounters in objects. After an immutable array has been initialized in this way, it can't be modified.

```
public NSArray(
    java.util.Vector aVector,
    NSRange aRange,
    boolean checkForNull)
```

Creates an array containing the objects from aVector in the range specified by aRange. After an immutable array has been initialized in this way, it can't be modified. The checkForNull argument controls the method's behavior when it encounters a null value in the vector: if checkForNull is true, the null value is simply ignored. If checkForNull is false, the method raises an IllegalArgumentException.

Static Methods

componentsSeparatedByString

```
public static NSArray componentsSeparatedByString(
    String string,
    String separator)
```

Returns an array containing substrings from string that have been divided by separator. The substrings in the array appear in the order they did in the receiver. If the string begins or ends with the separator, the first or last substring, respectively, is empty. For example, this code excerpt:

```
String list = "wrenches, hammers, saws";
NSArray listItems = NSArray.componentsSeparatedByString (", ");
```

produces an array with these contents:

Index	Substring
0	wrenches
1	hammers
2	saws

If list begins with a comma and space the array has these contents:

Index	Substring
0	(empty string)
1	wrenches
2	hammers
3	saws

If list has no separators—for example, “wrenches”—the array contains the string itself, in this case “wrenches”.

See Also: componentsJoinedByString

decodeObject

```
public static Object decodeObject(NSCoder coder)
```

Creates and returns an NSArray from the data in coder.

See Also: NSCoding **Interface Description**

operatorForKey

```
public static NSArray.Operator operatorForKey(String operatorName)
```

Returns the operator for the operator named operatorName.

See Also: “Operators” **(page 5)**

operatorNames

```
public static NSArray operatorNames()
```

Returns the names of the operations that can be performed on array elements. By default the operations are `count`, `max`, `min`, `avg`, and `sum`.

See Also: "Operators" **(page 5)**

removeOperatorForKey

```
public static void removeOperatorForKey(String operatorName)
```

Removes the operator identified by `operatorName` from the list of operators that can be performed on array elements.

See Also: "Operators" **(page 5)**

setOperatorForKey

```
public static void setOperatorForKey(
    String key,
    NSArray.Operator operator)
```

Sets the operator for `key` to `operator`. Throws an IllegalArgumentException if either `key` or `operator` are `null`.

See Also: "Operators" **(page 5)**

Instance Methods

arrayByAddingObject

```
public NSArray arrayByAddingObject(Object anObject)
```

Returns a new array that is a copy of the receiver with `anObject` added to the end. If `anObject` is `null`, an IllegalArgumentException is thrown.

See Also: `addObject` **(NSMutableArray)**

arrayByAddingObjectsFromArray

```
public NSArray arrayByAddingObjectsFromArray(NSArray otherArray)
```

Returns a new array that is a copy of the receiver with the objects contained in `otherArray` added to the end.

See Also: `addObjectsFromArray` **(NSMutableArray)**

classForCoder

```
public Class classForCoder()
```

Conformance to NSCoding. NSArray's implementation returns the class NSArray, so subclasses that don't override this method (such as NSMutableArray) are encoded as instances of NSArray.

See Also: `classForCoder` (NSCoding)

clone

```
public Object clone()
```

Simply returns the receiver. Since NSArrays are immutable, there's no need to make an actual clone.

componentsJoinedByString

```
public String componentsJoinedByString(String separator)
```

Constructs and returns a String that is the result of interposing `separator` between the elements of the receiver's array. For example, this code excerpt writes the path `System/Developer` to the console:

```
NSArray pathArray = new NSArray(new Object[] {'System', 'Developer'});
System.out.println('The path is '+ pathArray.componentsJoinedByString('/') + '.');
```

Each element in the receiver's array must handle either `description`, or if it is not implemented, `toString`. If the receiver has no elements, a String representing the empty string is returned.

See Also: `componentsSeparatedByString`

containsObject

```
public boolean containsObject(Object anObject)
```

Returns `true` if the receiver contains an object equal to `anObject`. This method determines whether an object is present in the array by sending an `equals` message to each of the array's objects (and passing `anObject` as the parameter to each `equals` message).

count

```
public int count()
```

Returns the number of objects currently in the array.

encodeWithCoder

```
public void encodeWithCoder(NSCoder coder)
```

Conformance to NSCoding. See the method description for `encodeWithCoder` in the NSCoding interface specification.

equals

```
public boolean equals(Object anObject)
```

Returns `true` if anObject is an NSArray and its contents are equal to the receiver's or `false` otherwise. If you know that `anObject` is an NSArray, use the more efficient method `isEqualToArray` instead.

firstObjectCommonWithArray

```
public Object firstObjectCommonWithArray(NSArray otherArray)
```

Returns the first object contained in the receiver that's equal to an object in `otherArray`, or `null` if no such object is found. This method uses `equals` to check for object equality.

getObjects

```
public void getObjects(Object[] buffer[])
```

Deprecated. Use `public Object[] objects()` instead.

```
public void getObjects(
    Object[] buffer[],
    NSRange aRange)
```

Deprecated. Use `public Object[] objects(NSRange)` instead.

hashCode

```
public int hashCode()
```

See the method description for `hashCode` in the Object class specification.

immutableClone

```
public NSArray immutableClone()
```

Returns an immutable copy of the receiver. Since an NSArray is immutable, NSArray's implementation simply returns the receiver. Subclasses such as NSMutableArray should override this method to create an immutable copy of the reciever.

indexOfIdenticalObject

```
public int indexOfIdenticalObject(Object anObject)
```

Searches all objects in the receiver for `anObject` (testing for equality by comparing object addresses) and returns the lowest index whose corresponding array value is identical to `anObject`. If none of the objects in the receiver are identical to `anObject`, this method returns `NotFound`.

```
public int indexOfIdenticalObject(
    Object anObject,
    NSRange aRange)
```

Searches the specified range within the receiver for anObject (testing for equality by comparing object addresses) and returns the lowest index whose corresponding array value is identical to anObject. If none of the objects in the range are identical to anObject, this method returns NotFound. Throws an IllegalArgumentException if aRange is out of bounds.

indexOfObject

```
public int indexOfObject(Object anObject)
```

Searches all objects in the receiver for anObject and returns the lowest index whose corresponding array value is equal to anObject. Objects are considered equal if equals returns true. If none of the specified objects are equal to anObject, returns NotFound.

```
public int indexOfObject(
    Object anObject,
    NSRange aRange)
```

Searches the specified range within the receiver for anObject and returns the lowest index whose corresponding array value is equal to anObject. Objects are considered equal if equals returns true. If none of the specified objects are equal to anObject, returns NotFound. Throws an IllegalArgumentException if aRange is out of bounds.

isEqualToArray

```
public boolean isEqualToArray(NSArray otherArray)
```

Compares the receiving array to otherArray, returning true if the contents of otherArray are equal to the contents of the receiver, false otherwise. Two arrays have equal contents if they each hold the same number of objects and objects at a given index in each array satisfy the equals test.

lastObject

```
public Object lastObject()
```

Returns the object in the array with the highest index value. If the array is empty, lastObject returns null.

makeObjectsPerformSelector

```
public void makeObjectsPerformSelector(
    NSSelector selector,
    Object[] anObject[])
```

Invokes the method specified by `selector` on each object in the receiver. The method is invoked each time with the values in `anObject` as the method's parameters. The method shouldn't, as a side effect, modify the receiver's collection of objects. The messages are sent using NSSelector's `invoke` method.

mutableClone

```
public NSMutableArray mutableClone()
```

Returns a mutable copy of the receiver. NSArray's implementation creates an NSMutableArray with the receiver's elements, not copies.

objectAtIndex

```
public Object objectAtIndex(int index)
```

Returns the object located at `index`. If the receiver is empty or if `index` is beyond the end of the array (that is, if `index` is greater than or equal to the value returned by `count`), an IllegalArgumentException is thrown.

See Also: `count`

objectEnumerator

```
public java.util.Enumeration objectEnumerator()
```

Returns an enumeration that lets you access each object in the array, in order, starting with the element at index 0. For example, consider the following code excerpt:

```
java.util.Enumeration enumerator = myArray.objectEnumerator();

while (enumerator.hasMoreElements()) {
     Object anObject = enumerator.nextElement();
     /* code to act on each element */
}
```

When this method is used with mutable subclasses of NSArray, your code shouldn't modify the array during enumeration.

See Also: reverseObjectEnumerator

objects

```
public Object[] objects()
```

Returns copies of the receiver's elements in a natural language array.

```
public Object[] objects(NSRange aRange)
```

Returns copies of the receiver's elements that fall within the limits specified by aRange in a natural language array.

objectsNoCopy

```
protected Object[] objectsNoCopy()
```

Returns the receiver's actual elements—not copies—in a natural language array.

reverseObjectEnumerator

```
public java.util.Enumeration reverseObjectEnumerator()
```

Returns an enumeration that lets you access each object in the array, in order, from the element at the highest index down to the element at index 0. For example, consider the following code excerpt:

```
java.util.Enumeration enumerator = myArray.reverseObjectEnumerator();

while (enumerator.hasMoreElements()) {
     Object anObject = enumerator.nextElement();
     /* code to act on each element */
}
```

When this method is used with mutable subclasses of NSArray, your code shouldn't modify the array during enumeration.

See Also: objectEnumerator

sortedArrayUsingComparator

```
public NSArray sortedArrayUsingComparator(NSComparator comparator)
    throws NSComparator.ComparisonException
```

Returns an array that lists the receiver's elements, as determined by comparator. The new array contains the receiver's elements, not copies of them. Throws if the comparator's compare method throws for any reason.

sortedArrayUsingSelector

```
public NSArray sortedArrayUsingSelector(NSSelector selector) throws NSComparator.ComparisonException
```

Deprecated. Use sortedArrayUsingComparator instead.

subarrayWithRange

```
public NSArray subarrayWithRange(NSRange aRange)
```

Returns a new array containing the receiver's elements that fall within the limits specified by aRange. If aRange isn't within the receiver's range of elements, an IndexOutOfBoundsException is thrown.

For example, the following code example creates an array containing the elements found in the first half of wholeArray (assuming wholeArray exists).

```
NSRange theRange = new NSRange(0, wholeArray.count()/2);
NSArray halfArray = wholeArray.subarrayWithRange(theRange);
```

takeValueForKey

```
public void takeValueForKey(
    Object value,
    String key)
```

Conformance to NSKeyValueCoding. For each element in the receiver, NSArray's implementation sets the element's value for key to value. For example, if key is "firstName" and value is "Unknown", this method sets the firstName property of each of the receiver's elements to "Unknown".

takeValueForKeyPath

```
public void takeValueForKeyPath(
    Object value,
    String key)
```

Conformance to NSKeyValueCodingAdditions. For more information, see the `takeValueForKeyPath` method description in the NSKeyValueCodingAdditions interface specification.

toString

```
public String toString()
```

Returns a string representation of the receiver.

valueForKey

```
public Object valueForKey(String key)
```

Conformance to NSKeyValueCoding. NSArray's implementation is more complex than the default:

- If key indicates an operation that doesn't require an argument (such as returning the array's count), `valueForKey` performs the operation and returns the result. key indicates an operation if its first character is "@". For example, if key is "@count", `valueForKey` invokes `compute` on the "count" operator. This has the effect of computing and returning the number of elements in the receiver. Don't use `valueForKey` for operations that take arguments; instead use `valueForKeyPath`.
- For any other key, `valueForKey` creates an array with the same number of elements as the receiver. For each element in the receiver, the corresponding element in the new array is the value for key of the receiver's element. For example, if key is "firstName", this method returns an array containing the `firstName` values for the receiver's elements. The key argument can be a key path of the form relationship.property; for example, "department.name". `valueForKey` replaces null values with an instance of NSKeyValueCoding.Null.

See Also: "Operators" **(page 5)**

valueForKeyPath

```
public Object valueForKeyPath(String keyPath)
```

Conformance to NSKeyValueCodingAdditions. NSArray's implementation is more complex than the default:

- If key indicates an operation takes an argument (such as computing an average), valueForKeyPath performs the operation and returns the result. key indicates an aggregate operation if its first character is "@". For example, if key is "@avg.salary", valueForKey invokes compute on the "avg" operator specifying the receiver and "salary" as arguments. This has the effect of computing and returning the average salary of the receiver's elements.
- Otherwise, valueForKeyPath invokes the default implementation of valueForKeyPath. For more information see the method description for valueForKeyPath in the NSKeyValueCodingAdditions interface specification.

See Also: "Operators" **(page 5)**

vector

```
public java.util.Vector vector()
```

Returns the receiver as a Vector.

NSBundle

Inherits from: Object

Package: com.webobjects.foundation

Class Description

An NSBundle represents a location in the file system that groups code and resources that can be used in a program. NSBundles locate program resources and assist in localization. You build a bundle in Project Builder using a Java WebObjects Application or Java WebObjects Framework project.

An NSBundle is an object that corresponds to a directory where related resources—including executable code—are stored. The directory, in essence, "**bundles** " a set of resources used by an application into convenient chunks, and the NSBundle object makes those resources available to the application. NSBundle can find requested resources in the directory. The term bundle refers both to the object and to the directory it represents.

Bundles are useful in a variety of contexts. Since bundles combine executable code with the resources used by that code, they facilitate installation and localization. NSBundles are also used to locate specific resources and to determine which classes are loaded.

Each resource in a bundle usually resides in its own file. Bundled resources include such things as:

- Images—GIF or JPEG images displayed on web pages
- Localized character strings

Types of Bundles

NSBundle supports two types of bundles: application bundles and framework bundles.

Application Bundles

An application bundle is a bundle that contains the resources needed to launch the application. Its extension is ".woa". To build an application bundle with Project Builder, use the Java WebObjects Application project type.

Every application has a single application bundle called the "main bundle". You obtain an NSBundle object corresponding to the main bundle with the `mainBundle` static method. This is typically the running application itself.

Framework Bundles

A framework bundle is a bundle associated with a framework: a directory containing shared classes along with the resources that go with those classes, such as images and localized strings. A framework directory has a ".framework" extension. To build a framework bundle with Project Builder, use the Java WebObjects Framework project type.

You can get an NSBundle object associated with a framework by invoking the static method `bundleForName` specifying, as the argument, the name of the framework sans the ".framework" extension. Alternatively you can invoke the `bundleForClass` method specifying a class that's defined in the framework. To get all the framework bundles available to your application, you can invoke the `frameworkBundles` static method.

Bundle Availability

When an application starts up, it loads all of the bundles represented by class path components. If the class path contains multiple framework bundles with the same name, only the first one is loaded; the rest are ignored.

If you are unsure which bundles are loaded at startup, you can enable NSBundle debugging by setting the `NSBundleDebugEnabled` user default to `YES`. NSBundle subsequently logs the paths to the bundles as it loads them.

Localized Resources

If an application is to be used in more than one part of the world, its resources may need to be customized, or "localized," for language, country, or cultural region. An application may need, for example, to have separate Japanese, English, French, German, and Spanish versions of the images that label submit buttons.

Resources specific to a particular language are grouped together in a resource directory. This directory has the name of the language (in English) followed by a ".lproj" extension (for "language project"). The application mentioned above, for example, would have Japanese.lproj, English.lproj, French.lproj, German.lproj, and Spanish.lproj directories. The application also has a Nonlocalized.lproj directory, which contains resources shared by all locales.

It is good programming practice to ensure that if a resource appears in one language directory it also appears in all the others. Thus, Icon.gif in French.lproj should be the French counterpart to the Spanish Icon.gif in Spanish.lproj, and so on. However this discipline is not completely necessary. If German.lproj does not contain an Icon.gif resource, the Icon.gif resource in Nonlocalized.lproj will be used instead.

The server's locale determines which set of localized resources will actually be used by the application. NSBundle objects invoke the java.util.Locale.getDefault method to determine the locale and chooses the localized resources accordingly.

How Resources Appear on the File System

A bundle's resources are stored in a directory named Resources within the bundle directory on the file system. Within the Resources directory are all of the language directories except Nonlocalized.lproj. The non-localized resources in the Nonlocalized.lproj directory are mapped into the top level of the Resources directory on the file system.

For example, suppose the NSBundle resources are organized as shown below:

Listing 0-1 Resource organization example

```
English.lproj
     Edit.wo
          Edit.html
          Edit.wod
          Edit.woo
```

```
Nonlocalized.lproj
     Edit.wo
          Edit.html
          Edit.wod
          Edit.woo
     Images
          Icon.gif
          Background.jpeg
     Main.wo
          Main.html
          Main.wod
          Main.woo
```

These resources appear on the file system as:

Listing 0-2 How the resources appear on the file system

```
Resources
     Edit.wo
          Edit.html
          Edit.wod
          Edit.woo
     Images
          Icon.gif
          Background.jpeg
     Main.wo
          Main.html
          Main.wod
          Main.woo
     English.lproj
          Edit.wo
               Edit.html
               Edit.wod
               Edit.woo
```

Determining Available Resources

NSBundle provides two methods to determine the resources it contains: `resourcePathsForResources` and `resourcePathsForLocalizedResources`. These methods return *resource paths*, or paths specified according to NSBundle's resource organization, not the resource organization as it appears on the file system. For example, the resource path to the `Background.jpeg` resource in Listing 0-1 is `Nonlocalized.lproj/Images/Background.jpeg`.

resourcePathsForResources

The `resourcePathsForResources` method takes two arguments: a subdirectory and an extension. The method returns an NSArray containing resource path strings for the resources in the specified subdirectory that have the specified extension. If you specify `null` for the subdirectory, the method returns the resource paths for the resources starting from the top level. If you specify `null` for the extension, the method will not filter resources according to their extension. Table 0-2 shows examples of invoking `resourcePathsForResources` with various parameters for the bundle depicted in Listing 0-1.

Table 0-2 Results from invoking resourcePathsForResources.

extension	subdirectory	Result
null	null	{ "English.lproj/Edit.wo/Edit.html", "English.lproj/Edit.wo/Edit.wod", "English.lproj/Edit.wo/Edit.woo", "English.lproj/Images/Icon.gif", "Nonlocalized.lproj/Edit.wo/Edit.html", "Nonlocalized.lproj/Edit.wo/Edit.wod", "Nonlocalized.lproj/Edit.wo/Edit.woo", Nonlocalized.lproj/Images/Icon.gif", "Nonlocalized.lproj/Images/Background.jpeg", "Nonlocalized.lproj/Main.wo/Main.html", "Nonlocalized.lproj/Main.wo/Main.wod", "Nonlocalized.lproj/Main.wo/Main.woo" }
"gif"	null	{ "English.lproj/Images/Icon.gif", "Nonlocalized.lproj/Images/Icon.gif" }
null	"English.lproj"	{ "English.lproj/Edit.wo/Edit.html", "English.lproj/Edit.wo/Edit.wod", "English.lproj/Edit.wo/Edit.woo", "English.lproj/Images/Icon.gif" }
"gif"	"English.lproj"	{ "English.lproj/Images/Icon.gif" }

resourcePathsForLocalizedResources

The `resourcePathsForLocalizedResources` method returns an NSArray of resource paths to resources that are appropriate for the current locale. If a resource appears in more than one language directory, this method chooses whether to include it in the array based on the following criteria:

- If the resource appears in the language directory for the current locale, the method includes its path in the results array.
- If the resource appears in Nonlocalized.lproj but not in the current locale's language directory, the method includes its path in the results array.
- If the resource doesn't appear in `Nonlocalized.lproj` or the current locale's language directory the method does not include its path in the results array.

The `resourcePathsForLocalizedResources`method also takes the `extension` and `subdirectory` arguments that allow you to filter the result array based on the extension or subdirectory. Table 0-3 shows examples of invoking `resourcePathsForLocalizedResources` with various parameters for the bundle depicted in Listing 0-1.

Table 0-3 Results of invoking resourcePathsForLocalizedResources

extension	subdirectory	Result
null	null	{ "English.lproj/Edit.wo/Edit.html", "English.lproj/Edit.wo/Edit.wod", "English.lproj/Edit.wo/Edit.woo", "Nonlocalized.lproj/Images/Icon.gif", "Nonlocalized.lproj/Images/Background.jpeg", "Nonlocalized.lproj/Main.wo/Main.html", "Nonlocalized.lproj/Main.wo/Main.wod", "Nonlocalized.lproj/Main.wo/Main.woo" }
"html"	null	{ "English.lproj/Edit.wo/Edit.html", "Nonlocalized.lproj/Main.wo/Main.html" }
null	"Edit.wo"	{ "English.lproj/Edit.wo/Edit.html", "English.lproj/Edit.wo/Edit.wod", "English.lproj/Edit.wo/Edit.woo" }
"html"	"Edit.wo"	{ "English.lproj/Edit.wo/Edit.html" }

resourcePathsForDirectories

NSBundle also includes a method called `resourcePathsForDirectories` that returns the directories containing resources. It also takes the `extension` and `subdirectory` parameters. Table 0-4 shows examples of invoking `resourcePathsForDirectories` with various parameters for the bundle depicted in Listing 0-1.

Table 0-4 Results of invoking resourcePathsForDirectories

extension	subdirectory	Result
null	null	{ "English.lproj/Edit.wo", "English.lproj/Images", "Nonlocalized.lproj/Edit.wo", "Nonlocalized.lproj/Images", "Nonlocalized.lproj/Main.wo" }
"wo"	null	{ "Nonlocalized.lproj/Main.wo", "Nonlocalized.lproj/Edit.wo", "English.lproj/Edit.wo" }
null	"English.lproj"	{ "English.lproj/Edit.wo", "English.lproj/Images" }
"wo"	"English.lproj"	{ "English.lproj/Edit.wo" }

Accessing NSBundle Resources

NSBundle provides two methods to access resources: `bytesForResourcePath` and `inputStreamForResourcePath`. Both methods require a single argument: a full resource path as returned by the `resourcePathsForResources` and `resourcePathsForLocalizedResources` methods. The `bytesForResourcePath` method returns a byte array containing data for the resource specified by the path. The `inputStreamForResourcePath` returns an java.io.InputStream for the resource specified by the path.

Sometimes you want to access a localized resource without specifying the full resource path. For example, if you might want to get the `Icon.gif` resource appropriate for the current locale. To do this, you invoke `resourcePathForLocalizedResourceNamed` to determine the full resource path for the localized resource and, in turn, invoke `bytesForResourcePath` or `inputStreamForResourcePath` with the full path.

The `resourcePathForLocalizedResourceNamed` method first searches the current locale's language directory for the resource then the `Nonlocalized.lproj` directory. If it finds the resource, it returns the resource's path. Otherwise it returns `null`. You can specify a subdirectory for the method to search in. For example, if the current locale is English, and the resources are organized as shown

in Listing 0-1, and you invoke `resourcePathForLocalizedResourceNamed` for the "`Edit.html`" resource in the "`Edit.wo`" subdirectory, the method returns "`English.lproj/Edit.wo/Edit.html`". If the current locale is German, the method returns "`Nonlocalized.lproj/Edit.wo/Edit.html`".

Method Types

Accessing resources

`bytesForResourcePath`

`inputStreamForResourcePath`

Finding bundles

`frameworkBundles`

`bundleForClass`

`bundleForName`

`mainBundle`

Getting resource paths

`resourcePathsForDirectories`

`resourcePathForLocalizedResourceNamed`

`resourcePathsForLocalizedResources`

`resourcePathsForResources`

Getting bundle class information

`bundleClassPackageNames`

`bundleClassNames`

Getting bundle attributes

`isFramework`

`name`

`principalClass`

properties

Methods inherited from Object

toString

Deprecated methods

allBundles

allFrameworks

bundlePath

bundleWithPath

infoDictionary

load

pathForResource

pathsForResources

resourcePath

Static Methods

allBundles

```
public synchronized static NSArray allBundles()
```

Deprecated in the Java Foundation framework. Don't use this method. The only non-framework bundle that an application can access without deprecated API is the main bundle. Use `mainBundle` instead.

Returns an array containing all the non-framework bundles available to the application.

allFrameworks

```
public static NSArray allFrameworks()
```

Deprecated in the Java Foundation framework. Don't use this method. Use `frameworkBundles` instead.

Returns an array containing the bundles for all the frameworks included in the application.

bundleForClass

```
public synchronized static NSBundle bundleForClass(Class aClass)
```

Returns the bundle containing the class `aClass`.

bundleForName

```
public synchronized static NSBundle bundleForName(String name)
```

Returns the bundle with the specified name. See `name` for more information about how the name relates to the bundle on the file system.

bundleWithPath

```
public static NSBundle bundleWithPath(String path)
```

Deprecated in the Java Foundation framework. Don't use this method. To access a bundle that was loaded when the application started, use `bundleForName` or `bundleForClass`.

Returns an NSBundle that corresponds to the specified directory `path` or returns `null` if `path` does not identify an accessible bundle directory.

If the bundle object for the specified directory doesn't already exists, this method creates the returned bundle.

frameworkBundles

```
public synchronized static NSArray frameworkBundles()
```

Returns an array containing the bundles for all the frameworks included in the application.

mainBundle

```
public static NSBundle mainBundle()
```

Returns the application's main bundle. In general, the main bundle corresponds to an application file package or application wrapper: a directory that bears the name of the application and is marked by a "`.woa`" extension.

Instance Methods

bundleClassNames

```
public NSArray bundleClassNames()
```

Returns an array containing the names of all the receiver's classes.

bundleClassPackageNames

```
public NSArray bundleClassPackageNames()
```

Returns an array containing the names of all the packages containing the receiver's classes.

bundlePath

```
public String bundlePath()
```

Deprecated in the Java Foundation framework. Don't use this method. You should not need to know the file system path to the bundle directory.

Returns the full file system path name of the receiver's bundle directory.

bytesForResourcePath

```
public byte[] bytesForResourcePath(String resourcePath)
```

Returns a byte array containing the data for the resource specified by `resourcePath`. The resource path must be specified relative to the top level of the resources hierarchy, that is, the directory containing the language subdirectories. Note that the resource path for a resource is not the same as its file system path. See "Determining Available Resources" (page 27) for more information about resource paths.

infoDictionary

```
public NSDictionary infoDictionary()
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns a dictionary that contains information about the receiver. This information is extracted from the property list associated with the bundle. The `CustomInfo.plist` file is a source file for the bundle's property list.

inputStreamForResourcePath

```
public java.io.InputStream inputStreamForResourcePath(String resourcePath)
```

Returns an input stream for the resource specified by `resourcePath`. The resource path must be specified relative to the top level of the resources hierarchy, that is, the directory containing the language subdirectories. Note that the resource path for a resource is not the same as its file system path. See "Determining Available Resources" (page 27) for more information about resource paths.

isFramework

```
public boolean isFramework()
```

Returns whether the receiver represents a framework or not.

load

```
public boolean load()
```

Deprecated in the Java Foundation framework. Don't use this method. Dynamic loading is no longer supported.

Returns `true` if the bundle was loaded at application startup, otherwise returns `false`.

name

```
public String name()
```

Returns the name of the bundle. If the bundle is a Java WebObjects Application, this method returns the name of the directory containing the application without the "`.woa`" extension. If the bundle is a Java WebObjects Framework, this method returns the name of the directory containing the framework without the "`.framework`" extension.

pathForResource

```
public String pathForResource(
    String name,
    String extension)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `resourcePathForLocalizedResourceNamed` instead.

Returns the full file system path name for the resource identified by `name` with the specified file extension. If the `extension` argument is `null` or an empty string (""), the resource sought is identified by `name`, with any (or no) extension. The method first looks for a non-localized resource in the immediate bundle directory; if the resource is not there, it looks for the resource in the language-specific "`.lproj`" directory (the local language is determined by user defaults).

```
public String pathForResource(
    String name,
    String extension,
    String bundlePath)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `resourcePathForLocalizedResourceNamed` instead.

Returns the full file system path name for the resource identified by name, with the specified file name extension, and residing in the directory bundlePath; returns null if no matching resource file exists in the bundle. The argument bundlePath must be a valid bundle directory or null. The argument extension can be an empty string or null; in either case the pathname returned is the first one encountered with name, regardless of the extension. The method searches in this order:

```
<main bundle path>/Resources/bundlePath/name.extension
<main bundle path>/Resources/bundlePath/<language.lproj>/name.extension
<main bundle path>/bundlePath/name.extension
<main bundle path>/bundlePath/<language.lproj>/name.extension
```

The order of language directories searched corresponds to the user's preferences. If bundlePath is null, the same search order as described above is followed, minus bundlePath.

Note: These methods search for resources based on the resource organization on the file system, not the internal NSBundle resource organization as described in the NSBundle class description. Specifically, these methods do not support the "Nonlocalized.lproj" directory. Also, the methods do not "drill-down" into the subdirectories of the specified bundle path (except for the language.lproj subdirectories).

pathsForResources

```
public NSArray pathsForResources(
    String extension,
    String bundlePath)
```

Deprecated in the Java Foundation framework. Don't use this method. Use resourcePathsForResources instead.

Returns an array containing file system path names for all bundle resources having the specified file name extension and residing in the directory bundlePath; returns an empty array if no matching resource files are found. This method provides a means for dynamically discovering bundle resources. The argument bundlePath must be a valid bundle directory or null. The extension argument can be an empty string or null; if you specify either of these for extension, all bundle resources are returned. Although there is no guaranteed search order, all of the following directories will be searched:

```
<main bundle path>/Resources/bundlePath/name.extension
<main bundle path>/Resources/bundlePath/<language.lproj>/name.extension
<main bundle path>/bundlePath/name.extension
```

```
<main bundle path>/bundlePath/<language.lproj>/name.extension
```

The language directories searched corresponds to the current locale. If `bundlePath` is `null`, the same search order as described above is followed, minus `bundlePath`.

Note: This method searches for resources based on the resource organization on the file system, not the internal NSBundle resource organization as described in the NSBundle class description. Specifically, these methods do not support the "`Nonlocalized.lproj`" directory. Also, the methods do not "drill-down" into the subdirectories of the specified bundle path (except for the `language.lproj` subdirectories).

principalClass

```
public Class principalClass()
```

Returns the NSBundle's principal class. The principal class is responsible for ensuring that all classes in the framework are properly initialized. The NSBundle determines its principal class based on the bundle's property list. The property list represents a dictionary; the principle class is the value obtained using the key `NSPrincipalClass`. If the principal class is not specified in the property list, the method returns `null`.

If you create a framework that needs to be initialized using a principal class, you must specify the class name in the `CustomInfo.plist` file, a source file for the bundle's property list. For example, if your principal class is `myPackage.myPrincipalClass`, your `CustomInfo.plist` file should look like:

```
{
    NSPrincipalClass = myPackage.myPrincipalClass;
}
```

properties

```
public java.util.Properties properties()
```

Returns the receiver's properties. These properties are located in the `Properties` file in the `Resources` subdirectory of the directory corresponding to the receiver. See the NSProperties class for more information about the `Properties` file.

resourcePath

```
public String resourcePath()
```

Deprecated in the Java Foundation framework. Don't use this method. Resources are now accessed using the `bytesForResourcePath` and `inputStreamForResourcePath` methods.

Returns the full file system path name of the receiving bundle's subdirectory containing resources.

Note: In the Java Foundation Framework, the term *resource path* refers to the full specification of the location of a resource in an NSBundle. In previous versions of the Foundation Framework, the term *resource path* referred to the file system path to the directory containing a bundle's resources.

resourcePathsForDirectories

```
public NSArray resourcePathsForDirectories(String extension,
    String subdirectory)
```

Returns an array containing the resource paths of all the directories with the specified extension beneath the specified subdirectory. If `extension` is `null`, the method includes directories regardless of extension. If `subdirectory` is `null`, the method returns directories beneath the top level directory (the one containing the language directories). For examples of how this method is used, see "Determining Available Resources" (page 27).

See Also: `resourcePathsForResources`, `resourcePathsForLocalizedResources`

resourcePathForLocalizedResourceNamed

```
public String resourcePathForLocalizedResourceNamed(String name,
    String subdirectory)
```

Returns the resource path for the localized resource with the specified name within the specified subdirectory. This method determines the resource path based on the current locale. See "Accessing NSBundle Resources" (page 29) for more information about how this method chooses the resource path it returns.

If subdirectory is null, the method returns a resource path for a localized resource at the language directory level.

See Also: resourcePathsForLocalizedResources, resourcePathsForResources, resourcePathsForDirectories

resourcePathsForLocalizedResources

```
public NSArray resourcePathsForLocalizedResources(String extension,
    String subdirectory)
```

Returns an array containing the resource paths for all of the receiver's resources that are appropriate for the current locale, have the specified file extension, and lie within the specified subdirectory. See "Determining Available Resources" (page 27) for more information about how this method chooses the resource paths it returns.

If extension is null, the method includes localized resources regardless of extension. If subdirectory is null, the method returns localized resources beneath the top level directory (the one containing the language directories).

See Also: resourcePathsForResources, resourcePathsForDirectories

resourcePathsForResources

```
public NSArray resourcePathsForResources(String extension,
    String subdirectory)
```

Returns an array containing the resource paths of all of the receiver's resources that have the specified file extension and lie within the specified subdirectory. For examples of how this method is used, see "Determining Available Resources" (page 27).

If extension is null, the method includes resources regardless of extension. If subdirectory is null, the method returns resources beneath the top level directory (the one containing the language directories).

See Also: resourcePathsForLocalizedResources, resourcePathsForDirectories

toString

```
public String toString()
```

Returns a string representation of the receiver including its class name (NSBundle or a subclass), its `name`, its path, the names of its packages (as returned by `bundleClassPackageNames`), and the number of classes it contains.

Notifications

BundleDidLoadNotification

```
public static String BundleDidLoadNotification
```

LoadedClassesNotification

```
public static final String LoadedClassesNotification
```

Description forthcoming.

NSCoder

Inherits from: Object

Package: com.webobjects.foundation

Class Description

NSCoder is an abstract class that declares the API used by concrete subclasses to transfer objects and other data items between memory and some other format. This capability provides the basis for archiving (where objects and data items are stored on disk) and distribution (where objects and data items are copied between different processes or threads).

You should never need to subclass NSCoder. Rather, WebObjects provides private concrete subclasses that it uses by default. However, you might interact with a coder object if you create a class that implements the NSCoding interface.

NSCoder operates on scalars (booleans, bytes, and integers, for example), and any other types of object. A coder object stores object type information along with an object's data, so an object decoded from a stream of bytes is normally of the same class as the object that was originally encoded into the stream.

Encoding and Decoding Objects and Data Items

To encode or decode an object or data item, you must first create a coder object, then send it a message defined by NSCoder or by a concrete subclass to actually encode or decode the item. NSCoder itself defines no particular method for creating a coder; this typically varies with the subclass.

To encode an object or data item, use any of the `encode...` methods. To decode an object or data item, simply use the `decode...` method corresponding to the original `encode...` method. Matching these is important, as the method originally used determines the format of the encoded data.

NSCoder's interface is quite general. Concrete subclasses aren't required to properly implement all of NSCoder's methods, and may explicitly restrict themselves to certain types of operations.

Managing Object Graphs

Objects frequently contain references to other objects, which may in turn contain references to other objects. When analyzed, a group of objects may contain circular references or one object may be referred to by several other objects. In these cases, the objects form an object graph and require special handling to preserve the graph structure. NSCoder's `encodeObject` method preserves the graph structure.

Method Types

Encoding data

`encodeBoolean`

`encodeByte`

`encodeBytes`

`encodeChar`

`encodeClass`

`encodeDouble`

`encodeFloat`

`encodeInt`

`encodeLong`

`encodeObject`

`encodeObjects`

encodeShort

Decoding data

decodeBoolean

decodeByte

decodeBytes

decodeChar

decodeClass

decodeDouble

decodeFloat

decodeInt

decodeLong

decodeObject

decodeObjects

decodeShort

All methods

finishCoding

prepareForReading

prepareForWriting

Constructors

NSCoder

```
public NSCoder()
```

The no-arg constructor. Don't use this method; because NSCoder is an abstract class, you can never create an instance of it.

Instance Methods

decodeBoolean

```
public abstract boolean decodeBoolean()
```

Decodes and returns a `boolean` value that was previously encoded with `encodeBoolean`.

decodeByte

```
public abstract byte decodeByte()
```

Decodes and returns a `byte` value that was previously encoded with `encodeByte`.

decodeBytes

```
public abstract byte[] decodeBytes()
```

Decodes and returns an array of `byte` values that were previously encoded with `encodeBytes`.

decodeChar

```
public abstract char decodeChar()
```

Decodes and returns a `char` value that was previously encoded with `encodeChar`.

decodeClass

```
public abstract Class decodeClass()
```

Decodes and returns a class that was previously encoded with `encodeClass`.

decodeDouble

```
public abstract double decodeDouble()
```

Decodes and returns a `double` value that was previously encoded with `encodeDouble`.

decodeFloat

```
public abstract float decodeFloat()
```

Decodes and returns a `float` value that was previously encoded with `encodeFloat`.

decodeInt

```
public abstract int decodeInt()
```

Decodes and returns an `int` value that was previously encoded with `encodeInt`.

decodeLong

```
public abstract long decodeLong()
```

Decodes and returns a `long` value that was previously encoded with `encodeLong`.

decodeObject

```
public abstract Object decodeObject()
```

Decodes and returns an object that was previously encoded with `encodeObject`.

decodeObjects

```
public abstract Object[] decodeObjects()
```

Decodes and returns an array of objects that were previously encoded with `encodeObjects`.

decodeShort

```
public abstract short decodeShort()
```

Decodes and returns a `short` value that was previously encoded with `encodeShort`.

encodeBoolean

```
public abstract void encodeBoolean(boolean aBoolean)
```

Encodes `aBoolean`. To decode a value encoded with this method, use `decodeBoolean`.

encodeByte

```
public abstract void encodeByte(byte aByte)
```

Encodes `aByte`. To decode a value encoded with this method, use `decodeByte`.

encodeBytes

```
public abstract void encodeBytes(byte[] bytes[])
```

Encodes the `bytes` array. To decode a value encoded with this method, use `decodeBytes`.

encodeChar

```
public abstract void encodeChar(char aChar)
```

Encodes `aChar`. To decode a value encoded with this method, use `decodeChar`.

encodeClass

```
public abstract void encodeClass(Class aClass)
```

Encodes `aClass`. To decode a value encoded with this method, use `decodeClass`.

encodeDouble

```
public abstract void encodeDouble(double aDouble)
```

Encodes `aDouble`. To decode a value encoded with this method, use `decodeDouble`.

encodeFloat

```
public abstract void encodeFloat(float aFloat)
```

Encodes `aFloat`. To decode a value encoded with this method, use `decodeFloat`.

encodeInt

```
public abstract void encodeInt(int anInt)
```

Encodes `anInt`. To decode a value encoded with this method, use `decodeInt`.

encodeLong

```
public abstract void encodeLong(long aLong)
```

Encodes `aLong`. To decode a value encoded with this method, use `decodeLong`.

encodeObject

```
public abstract void encodeObject(Object anObject)
```

Encodes `anObject`. To decode a value encoded with this method, use `decodeObject`.

encodeObjects

```
public abstract void encodeObjects(Object[] anObject[])
```

Encodes the `objects` array. To decode a value encoded with this method, use `decodeObjects`.

encodeShort

```
public abstract void encodeShort(short aShort)
```

Encodes aShort. To decode a value encoded with this method, use decodeShort.

finishCoding

```
public void finishCoding()
```

Cleans up the receiver's state after the receiver has finished encoding data. NSCoder's implementation does nothing.

prepareForReading

```
public void prepareForReading(java.io.InputStream inputStream)
```

Prepares the receiver for reading data from inputStream. NSCoder's implementation does nothing.

prepareForWriting

```
public void prepareForWriting(java.io.OutputStream outputStream)
```

Prepares the receiver for writing to outputStream. NSCoder's implementation does nothing.

NSCoding.Support

Inherits from: Object

Package: com.webobjects.foundation

Class Description

NSCoding.Support is an abstract class that defines a mechanism for one class to provide NSCoding behavior on behalf of another class. Subclasses of NSCoding.Support encode and decode objects of a different class. Subclasses of NSCoding.Support are needed to provide coding for classes whose code you don't own and that don't implement NSCoding.

For example, consider Java Client WebObjects applications that use NSCoding to distribute objects between client and server. Not all objects that Java Client distributes implement NSCoding (java.lang.String, for example). To encode and decode non-NSCoding objects, Java Client uses specialized subclasses of NSCoding.Support.

Note: Java Client has private subclasses of NSCoding.Support to encode and decode objects basic Java value classes such as java.lang.String, java.lang.Number, java.math.BigDecimal, and java.util.Date.

A subclass of NSCoding.Support should implement the methods `encodeWithCoder` and `decodeObject` to encode and decode objects of a specific non-NSCoding class. NSCoding.Support's implementations of these methods do nothing.

NSCoding.Support manages a registry of Support classes for classes that don't implement NSCoding. Use the methods `setSupportForClass` and `supportForClass` to register and access the NSCoding.Support classes for performing coding on non-NSCoding objects.

Constructors

NSCoding.Support

```
public NSCoding.Support()
```

The no-arg constructor. Don't use this method; because NSCoding.Support is an abstract class, you can never create an instance of it.

Static Methods

setSupportForClass

```
public static void setSupportForClass(
    NSCoding.Support supportClass,
    Class aClass)
```

Sets `supportClass` as the support class to use for coding instances of `aClass`.

supportForClass

```
public static NSCoding.Support supportForClass(Class aClass)
```

Returns the support class used for coding instances of `aClass`.

Instance Methods

classForCoder

```
public Class classForCoder(Object anObject)
```

Returns the class a coder should record as the class for anObject when anObject is encoded. NSCoding.Support's implementation simply returns anObject's actual class.

See Also: classForCoder (NSCoding)

decodeObject

```
public abstract Object decodeObject(NSCoder aCoder)
```

Implemented by subclasses to decode an object of a specific type from the data in aCoder.

See Also: The NSCoding **class description**

encodeWithCoder

```
public abstract void encodeWithCoder(
    Object anObject,
    NSCoder aCoder)
```

Implemented by subclasses to encode an object of a specific type into aCoder.

See Also: encodeWithCoder (NSCoding)

NSComparator

Inherits from: Object

Package: com.webobjects.foundation

Class Description

NSComparator is an abstract class that defines an API for comparing two objects for the purpose of sorting them. The class defines one method, `compare`, which compares two parameters and returns one of `OrderedAscending`, `OrderedSame`, or `OrderedDescending`.

Instead of invoking `compare` directly on a comparator, you typically use the NSArray method `sortedArrayUsingComparator`, which sorts the elements of the receiving array into a new array, or the NSMutableArray method `sortUsingComparator`, which sorts the elements of an array in place. NSComparator provides default comparators to use with these sorting methods. See the section "Constants" (page 54).

Constants

NSComparator defines the following `int` constants as the possible return values for `compare`:

Constant	Description
`OrderedAscending`	Returned when the object arguments are in ascending order (the value of the first argument is less than the value of the second).
`OrderedSame`	Returned when the values of the object arguments are equal.
`OrderedDescending`	Returned when the object arguments are in descending order (the value of the first argument is less than the value of the second).

Additionally, NSComparator defines the following NSComparator constants to be used for comparing objects of the specified class:

Constant	Compares Objects of Class
`AscendingStringComparator`	String
`DescendingStringComparator`	String
`AscendingCaseInsensitiveStringComparator`	String
`DescendingCaseInsensitiveStringComparator`	String
`AscendingNumberComparator`	Number
`DescendingNumberComparator`	Number
`AscendingTimestampComparator`	NSTimestamp
`DescendingTimestampComparator`	NSTimestamp

Constructors

NSComparator

```
public NSComparator()
```

The no-arg constructor. Don't use this method; because NSComparator is an abstract class, you can never create an instance of it.

Instance Methods

compare

```
public abstract int compare(
    Object first,
    Object second) throws NSComparator.ComparisonException
```

Compares the values of `first` and `second` and returns the result, one of `OrderedAscending`, `OrderedSame`, or `OrderedDescending`. Specifically, for non-null `x`, `y`, and `z`:

- `compare(x, x)` returns `OrderedSame`.
- If `compare(x, y)` returns `OrderedSame`, then `compare(y, x)` returns `OrderedSame`
- If `compare(x, y)` returns `OrderedAscending`, then `compare(y, x)` returns `OrderedDescending`.
- If `compare(x, y)` returns `OrderedDescending`, then `compare(y, x)` returns `OrderedAscending`.
- If `compare(x, y)` returns `OrderedAscending` and `compare(y, z)` returns `OrderedAscending`, then `compare(x, z)` returns `OrderedAscending`.
- Exactly one of the following is true: `compare(x, x) == OrderedSame`, `compare(x, x) == OrderedAscending`, or `compare(x, x) == OrderedDescending`.
- The result of `compare(x, y)` must be the same in all invocations.

Throws an NSComparator.ComparisonException if a comparison between `first` and `second` is impossible or undefined; for example, if either argument is `null`.

NSComparator.ComparisonException

Inherits from: Exception : Throwable : Object

Package: com.webobjects.foundation

Class Description

Instances of the NSComparator.ComparisonException class are created and thrown when an error condition is encountered during the comparison of two objects. For more information, see the NSComparator class specification.

Constructors

NSComparator.ComparisonException

```
public NSComparator.ComparisonException(String message)
```

Creates and returns a new Exception with `message` as the message.

NSData

Inherits from:	Object
Implements:	Cloneable java.io.Serializable NSCoding
Package:	com.webobjects.foundation

Class Description

NSData and its subclass NSMutableData provide data objects, object-oriented wrappers for byte buffers. Data objects let byte arrays take on the behavior of Foundation objects. NSData creates static data objects, and NSMutableData creates dynamic data objects.

Data objects can wrap data of any size. The object contains no information about the data itself (such as its type); the responsibility for deciding how to use the data lies with the client. In particular, it will not handle byte-order swapping when distributed between big-endian and little-endian machines.

Table 0-5 describes the NSData methods that provide the basis for all NSData's other methods; that is, all other methods are implemented in terms of these four. If you create a subclass of NSData, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-5 NSData's Base API

Method	Description
`bytesNoCopy`	Returns the internal byte array that contains the receiver's data. Used by mutable subclasses of NSData.
`immutableBytes`	Returns an immutable byte array that contains the receiver's data.
`immutableRange`	Returns an immutable NSRange object that specifies the receiver's length.
`rangeNoCopy`	Returns the internal NSRange object that specifies the receiver's length. Used by mutable subclasses of NSData.

To extract a data object that contains a subset of the bytes in another data object, use the `subdataWithRange` method. To determine if two data objects are equal, use the `isEqualToData` method, which does a byte-for-byte comparison.

The `writeToStream` method lets you write the contents of a data object to a stream (a java.io.OutputStream object).

Constants

NSData defines the following constant:

Constant	Type	Description
`EmptyData`	NSData	An empty data object, which can be shared to save memory.

Interfaces Implemented

Cloneable

- clone

java.io.Serializable

NSCoding

- classForCoder
- decodeObject
- encodeWithCoder

Method Types

Constructors

- NSData

Accessing data

- bytes
- bytesNoCopy
- immutableBytes
- subdataWithRange

Testing data

- immutableRange
- length
- rangeNoCopy

isEqualToData

Storing data

stream

writeToStream

Methods inherited from Object

equals

hashCode

toString

Deprecated methods

dataWithContentsOfFile

dataWithContentsOfMappedFile

writeToFile

writeToURL

Constructors

NSData

```
public NSData()
```

Creates an empty data object.

```
public NSData(NSData data)
```

Creates a data object containing the contents of another data object, data.

```
public NSData(String string)
```

Deprecated in the Java Foundation Framework. Don't use this constructor. Use `NSData(string.getBytes())` instead.

```
public NSData(byte[] bytes)
```

Creates a data object with all the data in the byte array bytes.

```
public NSData(
    byte[] bytes,
    int offset,
    int count)
```

Creates a data object with the bytes from the language array bytes that fall in the range specified by offset and count.

```
public NSData(
    byte[] bytes,
    NSRange range)
```

Creates a data object with the bytes from the language array bytes that fall in the range specified by range.

```
public NSData(
    byte[] bytes,
    NSRange range,
    boolean noCopy)
```

Creates a data object with the bytes from the language array bytes that are fall in the range specified by range. The noCopy parameter specifies whether or not a copy of bytes is made.

```
public NSData(java.io.File file) throws java.io.IOException
```

Deprecated in the Java Foundation Framework. Don't use this constructor. Use `NSData(new FileInputStream(file),chunkSize)` instead.

```
public NSData(
    java.io.InputStream inputStream,
    int chunkSize) throws java.io.IOException
```

Creates a data object with the data from the stream specified by inputStream. The chunkSize parameter specifies the size, in bytes, of the block that the input stream returns when it reads. For maximum performance, you should set the chunk size to the approximate size of the data. This constructor reads the stream until it detects an end of file or encounters an exception, but it does not close the stream.

```
public NSData(java.net.URL url) throws java.io.IOException
```

Deprecated in the Java Foundation Framework. Don't use this constructor. Use the following code instead:

```
URLConnection connection = url.openConnection();
connection.connect();
NSData myData = new NSData(connection.getInputStream(),chunkSize);
```

Static Methods

dataWithContentsOfFile

```
public static NSData dataWithContentsOfFile(java.io.File file)
    throws java.io.IOException
```

Deprecated in the Java Foundation Framework. Don't use this method. Use the following code instead:

```
myData = new NSData(new FileInputStream(file), chunkSize);
```

```
public static NSData dataWithContentsOfFile(String path)
    throws java.io.IOException
```

Deprecated in the Java Foundation Framework. Don't use this method. Use the following code instead:

```
myData = new NSData(new FileInputStream(path), chunkSize);
```

dataWithContentsOfMappedFile

```
public static NSData dataWithContentsOfMappedFile(java.io.File file) throws java.io.IOException
```

Deprecated in the Java Foundation Framework. Don't use this method. Use the following code instead:

```
myData = new NSData(new FileInputStream(file), chunkSize);
```

decodeObject

```
public static Object decodeObject(NSCoder coder)
```

Creates an NSData from the data in coder.

See Also: NSCoding

Instance Methods

bytes

```
public byte[] bytes(
    int offset,
    int count)
```

Returns a byte array containing the receiver's contents that fall within the range specified by offset and count.

```
public byte[] bytes(NSRange range)
```

Returns a byte array containing the receiver's contents that fall within the range specified by range.

```
public byte[] bytes()
```

Returns a byte array containing all of the receiver's contents.

bytesNoCopy

```
protected byte[] bytesNoCopy()
```

Returns the internal byte array that contains the receiver's data. Due to the internal implementation of NSData, this array may contain bytes that are not actually a part of the receiver's data. The receiver's actual data is composed of the returned array's bytes that lie in the range returned by rangeNoCopy. Used by mutable subclasses of NSData.

```
public byte[] bytesNoCopy(NSMutableRange dataRange)
```

Returns the internal byte array that contains the receiver's data and sets dataRange's offset and length to those of the receiver's internal NSRange object. The receiver's actual data is composed of the returned array's bytes that lie within dataRange.

WARNING NSData assumes the internal byte array is immutable. You should not change the contents of this array.

classForCoder

```
public Class classForCoder()
```

Conformance to NSCoding. See the method description of classForCoder in the interface specification for NSCoding.

clone

```
public Object clone()
```

Simply returns the receiver. Since NSData objects are immutable, there's no need to make an actual clone.

encodeWithCoder

```
public void encodeWithCoder(NSCoder coder)
```

Conformance to NSCoding. See the method description of encodeWithCoder in the interface specification for NSCoding.

equals

```
public boolean equals(Object anObject)
```

Compares the receiving data object to anObject. If anObject is an NSData and the contents of anObject are equal to the contents of the receiver, this method returns true. If not, it returns false. Two data objects are equal if they hold the same number of bytes, and if the bytes at the same position in the objects are the same.

hashCode

```
public int hashCode()
```

Provide an appropriate hash code useful for storing the receiver in a hash-based data structure.

immutableBytes

```
protected byte[] immutableBytes()
```

Returns an immutable byte array that contains the receiver's data.

immutableRange

```
protected NSRange immutableRange()
```

Returns an immutable NSRange object that specifies the receiver's length.

isEqualToData

```
public boolean isEqualToData(NSData otherData)
```

Compares the receiving data object to `otherData`. If the contents of `otherData` are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`. Two data objects are equal if they hold the same number of bytes, and if the bytes at the same position in the objects are the same.

length

```
public int length()
```

Returns the number of bytes contained by the receiver.

rangeNoCopy

```
protected NSRange rangeNoCopy()
```

Returns the internal NSRange object that specifies the offset and length of the receiver's data relative to the internal byte array (as returned by `bytesNoCopy`). Used by mutable subclasses of NSData.

stream

```
public java.io.ByteArrayInputStream stream()
```

Creates and returns a java.io.ByteArrayInputStream containing the receiver's data.

subdataWithRange

```
public NSData subdataWithRange(NSRange range)
```

Returns a data object containing a copy of the receiver's bytes that are fall within the range specified by `range`. If `range` isn't within the receiver's range of bytes, a RangeException is thrown.

toString

```
public String toString()
```

Returns a string representation of the receiver that contains its length, its location, and some of its data.

writeToFile

```
public boolean writeToFile(String path)
```

Deprecated in the Java Foundation Framework. Don't use this method. Use the following code instead:

```
try {
          FileOutputStream fileOutputStream = new FileOutputStream(path);
          myData.writeToStream(fileOutputStream);
          fileOutputStream.close();
     } catch (IOException exception) {
```

```
        /* Do something with the exception */
    }
```

writeToStream

```
public void writeToStream(java.io.OutputStream outputStream)
    throws java.io.IOException
```

Writes the bytes in the receiver contents to the outputStream. If the write fails for any reason, throws a java.io.IOException.

See Also: writeToStream

writeToURL

```
public boolean writeToURL(
    java.net.URL url,
    boolean atomically)
```

Deprecated in the Java Foundation Framework. Don't use this method. Use the following code instead:

```
try {
        FileOutputStream fileOutputStream = new FileOutputStream(url.getFile());
        myData.writeToStream(fileOutputStream);
        fileOutputStream.close();
    } catch (IOException exception) {
        /* Do something with the exception */
    }
```

See Also: writeToStream

NSDelayedCallbackCenter

Inherits from: Object

Package: com.webobjects.foundation

Class Description

An NSDelayedCallbackCenter object (also called a delayed callback center) provides a way to guarantee that particular methods are invoked after an event has ended. You can register selectors with the delayed callback center. The center, in turn, invokes them when the event ends. In WebObjects, this happens at then end of the current WebObjects request-response cycle.

When you register a selector, you also specify a priority, which determines the order in which it is invoked relative to the other selectors. The selectors are invoked in order of ascending priority. To register a selector with the delayed callback center, use `performSelector`. To cancel it before the event ends, use `cancelPerformSelector`.

The event loop invokes `eventEnded` to indicate that the current event has ended. The `eventEnded` method invokes the queued selectors.

Each task has a default delayed callback center that you access with the `defaultCenter` static method.

Method Types

Accessing the default center

`defaultCenter`

Managing selectors

`cancelPerformSelector`

`performSelector`

Indicating the end of an event

`eventEnded`

Static Methods

defaultCenter

```
public static NSDelayedCallbackCenter defaultCenter()
```

Returns the current task's delayed callback center.

Instance Methods

cancelPerformSelector

```
public void cancelPerformSelector(
    NSSelector selector,
    Object target,
    Object argument)
```

Removes the specified selector with the specified target object and argument from the list of registered selectors.

eventEnded

```
public void eventEnded()
```

Invokes the registered selectors in order of ascending priority. The event loop should invoke this method when the current event ends.

performSelector

```
public void performSelector(
    NSSelector selector,
    Object target,
    Object argument,
    int priority)
```

Registers `selector` to be invoked on `target` with the specified argument and priority. When the current event ends, the registered selectors are invoked in order of ascending priority.

NSDictionary

Inherits from:	Object
Implements:	Cloneable java.io.Serializable NSCoding NSKeyValueCoding NSKeyValueCodingAdditions
Package:	com.webobjects.foundation

Class Description

The NSDictionary class declares the programmatic interface to objects that manage immutable associations of keys and values. Use this class, or its subclass NSMutableDictionary when you need a convenient and efficient way to retrieve data associated with an arbitrary key. (For convenience, we use the term **dictionary** to refer to any instance of one of these classes without specifying its exact class membership.)

A key-value pair within a dictionary is called an entry. Each entry consists of one object that represents the key, and a second object which is that key's value. Within a dictionary, the keys are unique. That is, no two keys in a single dictionary are equal (as determined by `equals`).

An instance of NSDictionary is an immutable dictionary: you establish its entries when it's created, and cannot modify them afterwards. An instance of NSMutableDictionary is a mutable dictionary: you can add or delete entries at any time, and the object automatically allocates memory as needed.

Internally, a dictionary uses a hash table to organize its storage and to provide rapid access to a value given the corresponding key. However, the methods defined in this class insulate you from the complexities of working with hash tables, hashing functions, or the hashed value of keys. The methods described below take keys directly, not their hashed form.

Methods that add entries to dictionaries—whether during construction (for all dictionaries) or modification (for mutable dictionaries)—add each value object to the dictionary directly. These methods also add each key object directly to the dictionary, which means that you must ensure that the keys do not change. If you expect your keys to change for any reason, you should make copies of the keys and add the copies to the dictionary.

Table 0-6 describes the NSDictionary methods that provide the basis for all NSDictionary's other methods; that is, all other methods are implemented in terms of these four. If you create a subclass of NSDictionary, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-6 NSDictionary's Base API

Method	Description
`count`	Returns the number of entries in the dictionary.
`objectForKey`	Returns the value associated with a given key.
`keysNoCopy`	Returns a natural language array containing the keys in the dictionary.
`objectsNoCopy`	Returns a natural language array containing the objects in the dictionary.

The other methods declared here operate by invoking one or more of these primitives. The non-primitive methods provide convenient ways of accessing multiple entries at once.

Constants

NSDictionary provides the following constant as a convenience; you can use it when you need an empty dictionary.

Constant	Type	Description
`EmptyDictionary`	NSDictionary	A shared NSDictionary instance containing no entries.

Interfaces Implemented

Cloneable

java.io.Serializable

NSCoding

- classForCoder
- decodeObject
- encodeWithCoder

NSKeyValueCoding

- takeValueForKey
- valueForKey

NSKeyValueCodingAdditions

- takeValueForKeyPath
- valueForKeyPath

Method Types

Constructors

- NSDictionary

Accessing keys and values

- allKeys
- allKeysForObject
- allValues

isEqualToDictionary

keyEnumerator

keysNoCopy

objectEnumerator

objectForKey

objectsForKeys

objectsNoCopy

Counting entries

count

Creating hash tables

hashtable

Copying dictionaries

immutableClone

mutableClone

Methods inherited from Object

clone

equals

hashCode

toString

Constructors

NSDictionary

```
public NSDictionary()
```

Creates an empty dictionary. To improve performance, use the `EmptyDictionary` shared instance instead. See Constants.

```
public NSDictionary(
    NSArray objectArray,
    NSArray keyArray)
```

Creates a NSDictionary with entries from the contents of the keyArray and objectArray NSArrays. This method steps through objectArray and keyArray, creating entries in the new dictionary as it goes. Each key object and its corresponding value object is added directly to the dictionary. An InvalidArgumentException is thrown if the objectArray and keyArray do not have the same number of elements.

Note: NSDictionary assumes that key objects are immutable. If your key objects are mutable, you should make copies of them and add the copies to the dictionary.

```
public NSDictionary(NSDictionary dictionary)
```

Creates a dictionary containing the keys and values found in dictionary.

```
public NSDictionary(
    Object object,
    Object key)
```

Creates a dictionary containing a single object object for a single key key.

Note: NSDictionary assumes that key objects are immutable. If your key objects are mutable, you should make copies of them and add the copies to the dictionary.

```
public NSDictionary(
    Object[] objects[],
    Object[] keys[])
```

Creates a NSDictionary with entries from the contents of the keys and objects arrays. This method steps through objects and keys, creating entries in the new dictionary as it goes. Each key object and its corresponding value object is added directly to the dictionary. An InvalidArgumentException is thrown if the objects and keys do not have the same number of elements.

Note: NSDictionary assumes that key objects are immutable. If your key objects are mutable, you should make copies of them and add the copies to the dictionary.

```
public NSDictionary(
    java.util.Dictionary dictionary,
    boolean ignoreNull)
```

Creates a dictionary containing the keys and values found in `dictionary`. If `ignoreNull` is `false`, throws an InvalidArgumentException if any key or value in `dictionary` is `null`.

Static Methods

decodeObject

```
public static Object decodeObject(NSCoder coder)
```

Creates an NSDictionary from the data in `coder`.

See Also: NSCoding

Instance Methods

allKeys

```
public NSArray allKeys()
```

Returns a new array containing the dictionary's keys or an empty array if the dictionary has no entries. The order of the elements in the array isn't defined.

See Also: `allValues`, `allKeysForObject`

allKeysForObject

```
public NSArray allKeysForObject(Object anObject)
```

Finds all occurrences of the value anObject in the dictionary and returns a new array with the corresponding keys. Each object in the dictionary is sent an equals message to determine if it's equal to anObject. If no object matching anObject is found, this method returns null.

See Also: allKeys, keyEnumerator

allValues

```
public NSArray allValues()
```

Returns a new array containing the dictionary's values, or an empty array if the dictionary has no entries. The order of the values in the array isn't defined.

See Also: allKeys, objectEnumerator

classForCoder

```
public Class classForCoder()
```

Conformance to NSCoding. See the method description of classForCoder in the interface specification for NSCoding.

clone

```
public Object clone()
```

Returns a copy (an NSDictionary object) of the receiver. Since NSDictionaries are immutable, there's no need to make an actual copy.

count

```
public int count()
```

Returns the number of entries in the dictionary.

encodeWithCoder

```
public void encodeWithCoder(NSCoder coder)
```

Conformance to NSCoding. See the method description of `encodeWithCoder` in the interface specification for NSCoding.

equals

```
public boolean equals(Object anObject)
```

Compares the receiving dictionary to `anObject`. If `anObject` is an NSDictionary and the contents of `anObject` are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`.

Two dictionaries have equal contents if they each hold the same number of entries and, for a given key, the corresponding value objects in each dictionary satisfy the `equals` test.

hashCode

```
public int hashCode()
```

Provide an appropriate hash code useful for storing the receiver in a hash-based data structure.

hashtable

```
public java.util.Hashtable hashtable()
```

Returns a java.util.Hashtable containing the receiver's entries.

immutableClone

```
public NSDictionary immutableClone()
```

Returns an immutable copy (an NSDictionary) of the receiver. Since NSDictionaries are immutable, there's no need to make an actual copy.

isEqualToDictionary

```
public boolean isEqualToDictionary(NSDictionary otherDictionary)
```

Compares the receiving dictionary to `otherDictionary`. If the contents of `otherDictionary` are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`.

Two dictionaries have equal contents if they each hold the same number of entries and, for a given key, the corresponding value objects in each dictionary satisfy the `equals` test.

keyEnumerator

```
public java.util.Enumeration keyEnumerator()
```

Returns an Enumeration object that lets you access each key in the dictionary.

```
java.util.Enumeration enumerator = myDict.keyEnumerator();

while (enumerator.hasMoreElements()) {{
     Object anObject = enumerator.nextElement();
     /* code to act on each element */
}
```

When this method is used with mutable subclasses of NSDictionary, your code shouldn't modify the entries during enumeration. If you intend to modify the entries, use the `allKeys` method to create a "snapshot" of the dictionary's keys. Then use this snapshot to traverse the entries, modifying them along the way.

Note that the `objectEnumerator` method provides a convenient way to access each value in the dictionary.

See Also: `allKeys`, `allKeysForObject`, `objectEnumerator`

keysNoCopy

```
protected Object[] keysNoCopy()
```

Returns an array containing the dictionary's values, or an empty array if the dictionary has no entries. The order of the values in the array isn't defined. This method is similar to `allKeys` except the keys are not copied.

See Also: `objectsNoCopy`

mutableClone

```
public NSMutableDictionary mutableClone()
```

Returns a mutable dictionary (an NSMutableDictionary) with the same keys and value objects as the receiver.

objectEnumerator

```
public java.util.Enumeration objectEnumerator()
```

Returns an enumerator object that lets you access each value in the dictionary.

```
java.util.Enumeration enumerator = myDict.objectEnumerator();

while (enumerator.hasMoreElements()) {{
     Object anObject = enumerator.nextElement();
     /* code to act on each element */
}
```

When this method is used with mutable subclasses of NSDictionary, your code shouldn't modify the entries during enumeration. If you intend to modify the entries, use the `allValues` method to create a "snapshot" of the dictionary's values. Work from this snapshot to modify the values.

See Also: `keyEnumerator`

objectForKey

```
public Object objectForKey(Object aKey)
```

Returns an entry's value given its key, or `null` if no value is associated with `aKey`.

See Also: `allKeys`, `allValues`

objectsForKeys

```
public NSArray objectsForKeys(
    NSArray keys,
    Object anObject)
```

Returns the set of objects from the receiver that correspond to the specified keys as an NSArray. The objects in the returned array and the keys array have a one-for-one correspondence, so that the nth object in the returned array corresponds to the nth key in keys. If an object isn't found in the receiver to correspond to a given key, the marker object, specified by anObject, is placed in the corresponding element of the returned array.

objectsNoCopy

```
protected Object[] objectsNoCopy()
```

Returns an array containing the dictionary's values, or an empty array if the dictionary has no entries. The order of the values in the array isn't defined. This method is similar to allValues except the objects are not copied.

See Also: keysNoCopy

takeValueForKey

```
public void takeValueForKey(
    Object object,
    String key)
```

Conformance to NSKeyValueCoding. Since NSDictionaries are immutable, this method simply throws an IllegalStateException.

takeValueForKeyPath

```
public void takeValueForKeyPath(
    Object object,
    String key)
```

Conformance to NSKeyValueCodingAdditions. See the method specification of takeValueForKeyPath in the interface specification for NSKeyValueCodingAdditions.

toString

```
public String toString()
```

Returns a string representation of the receiver containing a string representation of each key-value pair.

valueForKey

```
public Object valueForKey(String key)
```

Conformance to NSKeyValueCoding. Equivalent to `objectForKey`.

valueForKeyPath

```
public Object valueForKeyPath(String key)
```

Conformance to NSKeyValueCodingAdditions. If the key exists in the dictionary, this method returns the corresponding object in the dictionary by invoking `objectForKey`. Otherwise it invokes the default implementation of `valueForKeyPath`. See the method specification of `valueForKeyPath` in the interface specification for NSKeyValueCodingAdditions.

NSDisposableRegistry

Inherits from:	Object
Implements:	NSDisposable Serializable
Package:	com.webobjects.foundation

Class Description

An NSDisposableRegistry object is a registry of NSDisposable objects that should be disposed when the registry is disposed. You can add objects to a registry with `addObject` and `addObjectsFromRegistry`, remove objects with `removeObject`, and dispose of a registries objects with `dispose`.

There are two ways in which you might interact with a disposable registry: adding objects to another object's registry and creating a class whose instances manage their own disposable registries. As an example of the former, consider the EOController class (defined in the eoapplication package and used in Direct to Java Client applications). EOController has a disposable registry, which you can access with the EOController method `disposableRegistry`. In EOController's `dispose` method, it disposes its disposable registry, which in turn disposes all its objects. You can get a controller's registry and add objects to it; they will be disposed along with the EOController. The second way in which you might interact with a disposable registry, then, is to create a class similar to EOController that uses a disposable registry to group objects that should be disposed of along with instances of your class.

Constructors

NSDisposableRegistry

```
public NSDisposableRegistry()
```

Creates an empty disposable registry.

Instance Methods

addObject

```
public void addObject(NSDisposable anObject)
```

Adds `anObject` to the receiver so that `anObject` will be disposed when the receiver is disposed.

addObjectsFromRegistry

```
public void addObjectsFromRegistry(NSDisposableRegistry aDisposableRegistry)
```

Adds the objects in `aDisposableRegistry` to the receiver, so that `aDisposableRegistry`'s objects will be disposed when the receiver is disposed.

dispose

```
public void dispose()
```

Conformance to NSDisposable. NSDisposableRegistry's implementation simply sends `dispose` to all its objects.

See Also: `dispose` (NSDisposable)

removeObject

```
public void removeObject(NSDisposable anObject)
```

Removes anObject from the receiver.

toString

```
public String toString()
```

Returns a string representation of the receiver.

NSForwardException

Inherits from: RuntimeException : Exception : Throwable : Object

Package: com.webobjects.foundation

Class Description

NSForwardException objects (or forward exceptions) are wrappers for Throwable objects that are not RuntimeExceptions. Since NSForwardException is a subclass of RuntimeException, forward exceptions can be omitted from the `throws` clause of a method even if the original exception had to be declared.

NSForwardException is used internally within WebObjects to keep the API congruent with the WebObjects 4.5 API (which uses the Java Bridge). Apple doesn't anticipate the need for you to create NSForwardException objects. You may need to catch them, however. To access the original exception, use the `originalException` method.

Method Types

Constructors

`NSForwardException`

Accessing the wrapped exception

originalException

Methods inherited from Throwable

printStackTrace

stackTrace

toString

Constructors

NSForwardException

```
public NSForwardException(Throwable exception)

public NSForwardException(
    Throwable exception,
    String extraMessage)
```

Creates an NSForwardException from exception. If exception is already an NSForwardException, the constructor wraps the exception's originalException.

The two-argument constructor allows you to specify an extra message; in this case, the new NSForwardException's message is exception's message with extraMessage appended. See the Throwable class specification in Sun's documentation for more information about an exception's messages.

Instance Methods

originalException

```
public Throwable originalException()
```

Returns the exception wrapped by the receiver.

printStackTrace

```
public void printStackTrace()
```

Prints the wrapped exception and its stack trace to the standard error stream. See the class specification for java.lang.Throwable in Sun's Java documentation for more information about the stack trace format.

```
public void printStackTrace(java.io.PrintStream printStream)
```

Prints the wrapped exception and its stack trace to the specified print stream.

```
public void printStackTrace(java.io.PrintWriter printWriter)
```

Prints the wrapped exception and its stack trace to the specified print writer.

stackTrace

```
public String stackTrace()
```

Returns a string containing the wrapped exception and its stack trace.

toString

```
public String toString()
```

Returns a string representation of the receiver indicating the receiver's class, its wrapped exception's class, and the wrapped exception's error message string.

NSKeyValueCoding. DefaultImplementation

Inherits from: Object

Package: com.webobjects.foundation

Class Description

The NSKeyValueCoding. DefaultImplementation class provides default implementations of the NSKeyValueCoding and NSKeyValueCoding.ErrorHandling interfaces. For more information, see the NSKeyValueCoding and NSKeyValueCoding.ErrorHandling interface specifications.

Static Methods

handleQueryWithUnboundKey

```
public static Object handleQueryWithUnboundKey(
    Object anObject,
    String key)
```

Throws an NSKeyValueCoding. UnknownKeyException with anObject as the exception's object and key as the exception's key. Invoked from valueForKey when it finds no property binding for key.

See Also: NSKeyValueCoding.ErrorHandling

handleTakeValueForUnboundKey

```
public static void handleTakeValueForUnboundKey(
    Object anObject,
    Object value,
    String key)
```

Throws an NSKeyValueCoding. UnknownKeyException with anObject as the exception's object and key as the exception's key. Invoked from takeValueForKey when it finds no property binding for key.

See Also: NSKeyValueCoding.ErrorHandling

takeValueForKey

```
public static void takeValueForKey(
    Object anObject,
    Object value,
    String key)
```

Sets anObject's property identified by key to value, or invokes handleTakeValueForUnboundKey.

See Also: takeValueForKey (NSKeyValueCoding)

unableToSetNullForKey

```
public static void unableToSetNullForKey(
    Object anObject,
    String key)
```

Throws an IllegalArgumentException. Invoked from takeValueForKey when it's given a null value for a scalar property (such as an int or a float).

See Also: NSKeyValueCoding.ErrorHandling

valueForKey

```
public static Object valueForKey(
    Object anObject,
    String key)
```

Returns anObject's value for the property identified by key, or invokes handleQueryWithUnboundKey.

See Also: valueForKey (NSKeyValueCoding)

NSKeyValueCoding.Null

Inherits from:	Object
Implements:	Serializable Cloneable NSCoding
Package:	com.webobjects.foundation

Class Description

NSKeyValueCoding.Null is a final class that defines a unique object used to represent `null` values in collection objects, such as NSArrays, which don't allow `null` values.

For instance, Enterprise Objects Framework uses NSKeyValueCoding.Null to represent null values from database rows in its database level snapshots (NSDictionary objects). However, Enterprise Objects Framework automatically translates NSKeyValueCoding.Null to `null` in enterprise objects, so you should rarely need to write code that accounts for this class.

Whenever `null` is represented by NSKeyValueCoding.Null, it should be represented with the instance stored in the NSKeyValueCoding constant, `NullValue`. You can safely use this instance with the == operator to test for the presence of a null value:

```
if (value == NSKeyValueCoding.NullValue) {
    /* ... */
}
```

Interfaces Implemented

NSCoding

`classForCoder`

`encodeWithCoder`

Cloneable

`clone`

Static Methods

decodeObject

```
public static Object decodeObject(NSCoder aNSCoder)
```

Returns the shared instance of NSKeyValueCoding.Null stored in the NSKeyValueCoding constant `NullValue`.

See Also: NSCoding **Interface Description**

Instance Methods

classForCoder

```
public Class classForCoder()
```

Conformance to NSCoding. See the method description for `classForCoder` in the NSCoding interface specification.

clone

```
public Object clone()
```

Simply returns the shared instance of NSKeyValueCoding.Null stored in the constant `NullValue`.

encodeWithCoder

```
public void encodeWithCoder(NSCoder aNSCoder)
```

Conformance to NSCoding. See the method description for `encodeWithCoder` in the NSCoding interface specification.

toString

```
public String toString()
```

Returns a string representation of the receiver.

NSKeyValueCoding. UnknownKeyException

Inherits from: RuntimeException

Package: com.webobjects.foundation

Class Description

Instances of the NSKeyValueCoding. UnknownKeyException class are created and thrown when an unknown key is encountered during key-value coding.

For example, suppose an Employee object receives a `valueForKey` message with "partNumber" as the key. The Employee class doesn't declare a method or instance variable for "partNumber", so `valueForKey` throws an UnknownKeyException. An NSKeyValueCoding. UnknownKeyException has a userInfo dictionary containing entries for the object for which key-value coding failed (`TargetObjectUserInfoKey`) and the unknown key (`UnknownUserInfoKey`). For the Employee / partNumber example, the `TargetObjectUserInfoKey` entry would contain the Employee object and the `UnknownUserInfoKey` would contain the string "partNumber".

For more information on key-value coding and error conditions, see the NSKeyValueCoding and NSKeyValueCoding.ErrorHandling interface specifications.

Constants

NSKeyValueCoding. UnknownKeyException defines the following constants:

Constant	Type	Description
TargetObjectUserInfoKey	String	The key for an entry in the exception's user info dictionary. The entry contains the target object that does not implement the unknown key. This constant is deprecated. You should access this user info dictionary entry using the object method.
UnknownUserInfoKey	String	The key for an entry in the exception's user info dictionary. The entry contains the unknown key. This constant is deprecated. You should access this user info dictionary entry using the key method.

Constructors

NSKeyValueCoding.UnknownKeyException

```
public NSKeyValueCoding.UnknownKeyException(
    String message,
    Object anObject,
    String key)
```

Creates and returns a new UnknownKeyException with message as the message and a userInfo dictionary specifying anObject for the TargetObjectUserInfoKey and key for the UnknownUserInfoKey.

```
public NSKeyValueCoding.UnknownKeyException(
    String message,
    NSDictionary userInfo)
```

Deprecated in the Java Foundation framework. Don't use this method. Use NSKeyValueCoding.UnknownKeyException(String,Object,String) instead. Creates and returns a new UnknownKeyException with the specified message and userInfo dictionary.

Instance Methods

key

```
public String key()
```

Returns the unknown key that caused the exception to be thrown. Equivalent to getting the `UnknownUserInfoKey` entry from the userInfo dictionary.

object

```
public Object object()
```

Returns the object on which key-value coding was operating when an unknown key was encountered. Equivalent to getting the `TargetObjectUserInfoKey` entry from the userInfo dictionary.

userInfo

```
public NSDictionary userInfo()
```

Deprecated in the Java Foundation framework. Don't use this method. Use the `object` and `key` methods to access the exception's object and key instead. Returns the receiver's userInfo dictionary.

NSKeyValueCoding.Utility

Inherits from: Object

Package: com.webobjects.foundation

Class Description

The NSKeyValueCoding.Utility class is a convenience that allows you to access the properties of NSKeyValueCoding objects and non-NSKeyValueCoding objects using the same code. For more information, see the NSKeyValueCoding and interface specification.

Static Methods

handleQueryWithUnboundKey

```
public static Object handleQueryWithUnboundKey(
    Object anObject,
    String key)
```

If `anObject` is an NSKeyValueCoding.ErrorHandling, invokes `handleQueryWithUnboundKey` on `anObject`; otherwise invokes NSKeyValueCoding. DefaultImplementation's `handleQueryWithUnboundKey` method with `anObject` as the object on which to operate.

handleTakeValueForUnboundKey

```
public static void handleTakeValueForUnboundKey(
    Object anObject,
    Object value,
    String key)
```

If anObject is an NSKeyValueCoding.ErrorHandling, invokes handleTakeValueForUnboundKey on anObject; otherwise invokes NSKeyValueCoding. DefaultImplementation's handleTakeValueForUnboundKey method with anObject as the object on which to operate.

takeValueForKey

```
public static void takeValueForKey(
    Object anObject,
    Object value,
    String key)
```

If anObject is an NSKeyValueCoding, invokes takeValueForKey on anObject; otherwise invokes NSKeyValueCoding. DefaultImplementation's takeValueForKey method with anObject as the object on which to operate.

unableToSetNullForKey

```
public static void unableToSetNullForKey(
    Object anObject,
    String key)
```

If anObject is an NSKeyValueCoding.ErrorHandling, invokes unableToSetNullForKey on anObject; otherwise invokes NSKeyValueCoding. DefaultImplementation's unableToSetNullForKey method with anObject as the object on which to operate.

valueForKey

```
public static Object valueForKey(
    Object anObject,
    String key)
```

If anObject is an NSKeyValueCoding, invokes valueForKey on anObject; otherwise invokes NSKeyValueCoding. DefaultImplementation's valueForKey method with anObject as the object on which to operate.

NSKeyValueCoding.ValueAccessor

Inherits from: Object

Package: com.webobjects.foundation

Class Description

NSKeyValueCoding.ValueAccessor is an abstract class that establishes a mechanism by which NSKeyValueCoding can operate on objects' package access instance variables.

By default, Foundation's implementations of NSKeyValueCoding can't access package access instance variables. If you have package access instance variables in your NSKeyValueCoding objects, you can make them available to key-value coding in one of the three ways:

- Implement public `setKey` and `key` accessor methods for those instance variables that set and return the instance variables' values.
- Make the instance variables public.
- Add a subclass of NSKeyValueCoding.ValueAccessor named KeyValueCodingProtectedAccessor to your package. This class provides a mechanism to manipulate package access instance variables.

The best solution is to implement accessor methods or to make the instance variables public. However, if you have a lot of classes with a lot of package access instance variables, you can use the short-term solution that NSKeyValueCoding.ValueAccessor provides until you make the necessary changes to your code.

To use NSKeyValueCoding.ValueAccessor's mechanism, simply create a class in your package as follows:

```
package yourPackage;
import java.lang.reflect.*;
import com.webobjects.foundation.*;

public class KeyValueCodingProtectedAccessor extends NSKeyValueCoding.ValueAccessor {

     public KeyValueCodingProtectedAccessor() {
          super();
     }

     public Object fieldValue(Field field, Object object) throws
     IllegalArgumentException, IllegalAccessException {
          return field.get(object);
     }

     public void setFieldValue(Field field, Object value, Object object) throws
     IllegalArgumentException, IllegalAccessException {
          field.set(object, value);
     }

     public Object methodValue(Method method, Object object) throws
     IllegalArgumentException, IllegalAccessException, InvocationTargetException {
          return method.invoke(object, null);
     }

     public void setMethodValue(Method method, Object value, Object object) throws
     IllegalArgumentException, IllegalAccessException, InvocationTargetException {
          method.invoke(object, new Object[] {value});
     }
}
```

Constructors

NSKeyValueCoding.ValueAccessor

```
public NSKeyValueCoding.ValueAccessor()
```

The no-arg constructor. Don't use this method; because NSKeyValueCoding.ValueAccessor is an abstract class, you can never create an instance of it.

Static Methods

protectedAccessorForPackageNamed

```
public static NSKeyValueCoding.ValueAccessor protectedAccessorForPackageNamed(
    String packageName)
```

Returns the value accessor for the package identified by packageName.

removeProtectedAccessorForPackageNamed

```
public static void removeProtectedAccessorForPackageNamed(
    String packageName)
```

Removes (unregisters) the value accessor for the package identified by packageName.

setProtectedAccessorForPackageWithNamed

```
public static void setProtectedAccessorForPackageNamed(
    NSKeyValueCoding.ValueAccessor accessor,
    String packageName)
```

Sets the value accessor for the package identified by packageName to accessor.

Instance Methods

fieldValue

```
public abstract Object fieldValue(
    Object object,
    reflect.Field field) throws IllegalArgumentException, IllegalAccessException
```

Returns the value of `object`'s `field`.

methodValue

```
public abstract Object methodValue(
    Object object,
    reflect.Method method) throws
    IllegalArgumentException,
    IllegalAccessException,
    reflect.InvocationTargetException
```

Uses `method` to return `object`'s corresponding property value.

setFieldValue

```
public abstract void setFieldValue(
    Object object,
    reflect.Field field,
    Object value) throws IllegalArgumentException, IllegalAccessException
```

Sets `object`'s `field` value to `value`.

setMethodValue

```
public abstract void setMethodValue(
    Object object,
    reflect.Method method,
    Object value) throws
```

```
    IllegalArgumentException,
    IllegalAccessException,
    reflect.InvocationTargetException
```

Uses method to set object's corresponding property to value.

NSKeyValueCodingAdditions. DefaultImplementation

Inherits from: Object

Package: com.webobjects.foundation

Class Description

The NSKeyValueCodingAdditions. DefaultImplementation class provides default implementations of the NSKeyValueCodingAdditions interface. For more information, see the NSKeyValueCodingAdditions interface specification.

Static Methods

takeValueForKeyPath

```
public static void takeValueForKeyPath(
    Object anObject,
    Object value,
    String keyPath)
```

Sets anObject's property identified by keyPath to value. A key path has the form relationship.property (with one or more relationships). This method gets the destination object for each relationship using valueForKey, and sends the final object a takeValueForKeymessage with value and property.

See Also: takeValueForKeyPath (NSKeyValueCodingAdditions)

valueForKeyPath

```
public static Object valueForKeyPath(
    Object anObject,
    String keyPath)
```

Returns anObject's value for the derived property identified by keyPath. A key path has the form relationship.property (with one or more relationships). This method gets the destination object for each relationship using valueForKey, and returns the result of a valueForKey message to the final object.

See Also: valueForKeyPath (NSKeyValueCodingAdditions)

NSKeyValueCodingAdditions.Utility

Inherits from: Object

Package: com.webobjects.foundation

Class Description

The NSKeyValueCodingAdditions.Utility class is a convenience that allows you to access the properties of NSKeyValueCodingAdditions objects and non-`NSKeyValueCodingAdditions` objects using the same code. For more information, see the NSKeyValueCodingAdditions interface specification.

Static Methods

takeValueForKeyPath

```
public static void takeValueForKeyPath(
    Object anObject,
    Object value,
    String keyPath)
```

If `anObject` is an NSKeyValueCodingAdditions, invokes `takeValueForKeyPath` on `anObject`; otherwise invokes NSKeyValueCodingAdditions. DefaultImplementation's `takeValueForKeyPath` method with `anObject` as the object on which to operate.

valueForKeyPath

```
public static Object valueForKeyPath(
    Object anObject,
    String keyPath)
```

If anObject is an NSKeyValueCodingAdditions, invokes valueForKeyPath on anObject; otherwise invokes NSKeyValueCodingAdditions. DefaultImplementation's valueForKeyPath method with anObject as the object on which to operate.

NSLock

Inherits from:	Object
Implements:	NSLocking
Package:	com.webobjects.foundation

Class Description

An NSLock object is used to coordinate the operation of multiple threads of execution within the same application. An NSLock object can be used to mediate access to an application's global data or to protect a critical section of code, allowing it to run atomically.

An NSLock object represents a lock that can be acquired by only a single thread at a time. While one thread holds the lock, any other thread is prevented from doing so until the owner relinquishes the lock. An application can have multiple NSLock objects, each protecting different sections of code. It's safest to create all of the locks before the application becomes multi-threaded, to avoid race conditions. If you want to create additional locks after the application becomes multi-threaded, you should create the new lock inside a critical code section that is itself protected by an existing lock.

The basic interface to NSLock is declared by the NSLocking interface, which defines the `lock` and `unlock` methods. To this base, NSLock adds the `tryLock` methods. Whereas the `lock` method declared in the interface doesn't return until it is successful, the methods declared in this class add more flexible means of acquiring a lock.

An NSLock could be used to coordinate the updating of a visual display shared by a number of threads involved in a single calculation:

```
boolean moreToDo = true;
NSLock myLock = new NSLock();
...
while (moreToDo) {
     /* Do another increment of calculation */
     /* until there's no more to do. */
     if (myLock.tryLock()) {
          /* Update display used by all threads. */
          myLock.unlock();
     }
}
```

The NSLock, NSMultiReaderLock, and NSRecursiveLock classes all adopt the NSLocking protocol and offer various additional features and performance characteristics. See the NSMultiReaderLock and NSRecursiveLock class descriptions for more information.

Method Types

Constructors

NSLock

Instance methods

lock

lockBeforeDate

toString

tryLock

unlock

Constructors

NSLock

```
public NSLock()
```

Creates an NSLock object.

Instance Methods

lock

```
public synchronized void lock()
```

Conformance to NSLocking. See the method description of `lock` in the interface specification for NSLocking.

lockBeforeDate

```
public boolean lockBeforeDate(NSTimestamp timestamp)
```

This method is deprecated. Use `tryLock(NSTimestamp timestamp)` instead.

toString

```
public String toString()
```

Returns a string representation of the receiver indicating whether or not the lock is taken.

tryLock

```
public synchronized boolean tryLock()
```

Attempts to acquire a lock. Returns immediately, with a value of `true` if successful and `false` otherwise.

```
public synchronized boolean tryLock(long msec)
```

Attempts to acquire a lock for `msec` milliseconds. The thread is blocked until the receiver acquires the lock or `msec` milliseconds have passed. Returns `true` if the lock is acquired within this time limit. Returns `false` if the time limit expires before a lock can be acquired.

```
public boolean tryLock(NSTimestamp timestamp)
```

Attempts to acquire a lock until the time specified by `timestamp`. The thread is blocked until the receiver acquires the lock or `timestamp` is reached. Returns `true` if the lock is acquired within this time limit. Returns `false` if the time limit expires before a lock can be acquired.

unlock

```
public synchronized void unlock()
```

Conformance to NSLocking. See the method description of `unlock` in the interface specification for NSLocking.

NSLog

Inherits from: Object

Package: com.webobjects.foundation

Class Description

NSLog is a static class that you use to access the WebObjects Foundation logging system. It allows you to log debugging messages, and provides functionality for controlling the level and focus of debugging output. Logging with NSLog offers greater flexibility and integration with the WebObjects runtime and debugging system than does logging with System.out, System.err, or java.rmi.server.LogStream.

By specifying debug groups and the debug level using NSLog's methods, you can control the scope and granularity of debugging output. Perhaps you are only interested in seeing debugging output that relates to the Enterprise Objects Framework, or more specifically, debugging output that relates to database fetches. NSLog is designed to allow you to specify the output you want to see, based on predefined debug groups and levels. See the section on "Creating Custom Debug Groups" (page 128) to understand how NSLog uses a bit mask to specify sets of debug groups.

Although NSLog provides functionality for advanced debugging, it is also as simple to use as outputting to System.out.*:

```
NSLog.out.appendln("WebObjects");
```

The above code outputs "WebObjects" in the console by default, but output can be directed to custom NSLog.Loggers.

The logging system is made up of three classes: NSLog, NSLog.Logger, and NSLog.PrintStreamLogger. NSLog provides methods for controlling the level (amount of information sent to the logger) and focus (scope of the information sent to the logger) of debugging output. NSLog.Logger is an abstract class that provides the basic functionality for NSLog. NSLog.PrintStreamLogger is a subclass of NSLog.Logger and appends information to the debug logger which is directed to a java.io.PrintStream pointing to System.out, System.err, or to a custom java.io.PrintStream.

The NSLog class cannot be instantiated.

Redirecting the Output of NSLog

The Foundation logging system offers the ability to create custom logging implementations, and to redirect their output to custom java.io.PrintStreams, such as local files. This example illustrates these features of the logging system, and directs output to a custom java.io.PrintStream, the local file "`/Local/Users/log.txt`":

```
// Output defaults to System.out.
NSLog.PrintStreamLogger aLogger = new NSLog.PrintStreamLogger();

// New print stream based on path.
PrintStream aStream = NSLog.printStreamForPath ("/Local/Users/log.txt");

// Direct output to the custom PrintStream.
aLogger.setPrintStream (aStream);

// Assign the custom PrintStreamLogger to the declared instances of NSLogger in NSLog.
NSLog.setOut(aLogger);
NSLog.setErr(aLogger);
NSLog.setDebug(aLogger);

// Enable verbose logging.
aLogger.setIsVerbose (true);
String str = "WebObjects";

// Outputs "[2000-10-18 09:01:00 GMT] <main> WebObjects"
aLogger.appendln (str);

// Disables logging.
aLogger.setIsEnabled (false);
```

```
// Outputs nothing, since logging is disabled.
daLogger.appendln (str);
```

By subclassing NSLog.Logger instead of NSLog.PrintStreamLogger, you can provide custom logging implementations that are not based on java.io.PrintStreams, such as outputting to email, Swing windows, etc. See the documentation for "NSLog.Logger" (page 139) for more details.

Debug Groups

To control the scope of the debug information that is sent to the logger, NSLog declares a number of debug groups, which are listed in the "Constants" (page 130) section. By default, logging is enabled for all the debug groups declared by NSLog. All the WebObjects frameworks, including Foundation and EOF, use NSLog for debugging, and rely on debug groups to control the scope of their debug logging.

As an example, this code uses debug groups to disable logging for debug information related to the Enterprise Objects Framework:

```
NSLog.refuseDebugLoggingForGroups(NSLog.DebugGroupEnterpriseObjects)
```

Before sending debug information to the logger, all classes that log EOF-related debug information check to see if debug logging is allowed for EOF-related issues using one of the `debugLoggingAllowedFor...` methods. Since the above code removed the `DebugGroupEnterpriseObjects` bit from the bit mask, EOF-related issues won't be sent to the logger.

Debug Levels

To control the granularity of debug information that is sent to the logger, NSLog declares a number of debug levels, which are listed in the "Constants" (page 130) section. A debug level specifies the importance of the information displayed in a message. These messages may be the result of exceptions or errors, but may also just be informational, such as printing variable values.

When an exception is encountered, debug information should only be sent to the logger if the debug level is `DebugLevelCritical` or greater. If the debug level is set to `DebugLevelOff`, any error message in the catch block should not be sent to the logger. A simple example:

```
} catch (SomeException e) {
     if (NSLog.debugLoggingAllowedForLevel(DebugLevelCritical) {
```

```
            NSLog.debug.appendln("Exception encountered: " + e.getMessage());
            NSLog.debug.appendln(e);
        }
}
```

When you want to see the value of a variable at particular points in your application, you use `DebugLevelInformational` to control logging. Logging the values of variables can be expensive, so it is prudent to wrap this kind of logging in a debug level conditional. For instance, you should conditionalize the logging of values in an array like this:

```
// Assuming declaration of int[] anArray = new int[10];
if debugLoggingAllowedForLevel(DebugLevelInformational) {
     for (i = 0, i < 9, i++) {
          NSLog.out.appendln(anArray[i]);
     }
}
```

`DebugLevelDetailed` should be used to conditionalize logging for computationally intensive tasks, such as JDBC function calls.

Note that if you specify the allowed debug level using the `setAllowedDebugLevel` method, messsages of that level and lower will be logged. That is, `DebugLevelDetailed` will also log messages of level `DebugLevelInformational` and messages of level `DebugLevelCritical`. Likewise, `DebugLevelInformational` will also log messages of level `DebugLevelCritical`, but not messages of level `DebugLevelDetailed`.

Creating Custom Debug Groups

NSLog uses a bit mask to specify a set of debug groups, which means that you can easily create a custom debug groups mask for more flexible logging. For instance, if you are interested in logging debugging output only for EOF-related issues and EOModel-related issues, you can simply bitwise inclusive OR the groups together using the "|" operator in Java. That creates a debug groups mask which you can use to more flexibly monitor and log errors relating to those two issues. To do this:

```
// First, remove all debug groups from the mask, since they are all enabled by default.
NSLog.refuseDebugLoggingForGroupsMask(~0);
// Create a custom debug groups mask.
NSLog.allowDebugLoggingForGroupsMask
     (DebugGroupEnterpriseObjects | DebugGroupModel);
```

In addition to the debug groups provied for you by WebObject, you can add your own debug groups. The high 32 bits (i.e. bits 32 - 63) are available to developers. The low bits are reserved by WebObjects. An example of declaring a new debug group:

```
public static final long DebugGroupCustomGroup = 1 << 32;
```

NSLog From the Command Line

You can enable debug groups and levels from the command line. For example:

```
% cd MyJavaWOApp.woa
% ./MyJavaWOApp -DNSDebugLevel=2 -DNSDebugGroups=32
```

The above code enables `DebugLevelInformational` and `DebugGroupResources`. The argument for debug level is passed in as an `int`; for debug groups, the argument is passed in as a `long`. Therefore, you must calculate the long value for each debug group you pass on the command line (in order to get the bit position for each group).

To do this, calculate 2 ^ (value for the debug group as listed in the "Constants" (page 130) section). For example, to enable `DebugGroupWebObjects`, calculate 2 ^ 2, and pass the result as the argument on the command line.

You can enable multiple debug groups by summing the long values of each group. For example:

```
% ./MyJavaWOApp -DNSDebugLevel=2 -DNSDebugGroups=16 + 32 + 1024
```

The above code enables `DebugGroupMultithreading`. `DebugGroupResources`, and `DebugGroupFormatting`.

Constants

NSLog defines the following constants:

Constant	Type	Value	Description
DebugGroupApplicationGeneration	long	3	The debug group for logging of general application generation issues.
DebugGroupArchiving	long	6	The debug group for logging of encoding and decoding issues.
DebugGroupAssociations	long	19	The debug group for logging of association exceptions and problems.
DebugGroupComponentBindings	long	9	The debug group for logging of binding exceptions and problems.
DebugGroupControllers	long	20	The debug group for logging of controller exceptions and problems.
DebugGroupComponents	long	26	Description forthcoming.
DebugGroupDatabaseAccess	long	16	The debug group for logging of database access exceptions and problems.
DebugGroupDeployment	long	22	The debug group for logging of Monitor, wotaskd, and deployment related issues.
DebugGroupEnterpriseObjects	long	1	The debug group for enabling logging of general EOF issues.
DebugGroupFormatting	long	10	The debug group for logging of formatting exceptions and problems.
DebugGroupIO	long	13	The debug group for logging of I/O exceptions and problems.
DebugGroupKeyValueCoding	long	8	The debug group for logging of key value coding exceptions and problems

Constant	Type	Value	Description
DebugGroupModel	long	15	The debug group for logging of EOModel exceptions, problems, and inconsistencies.
DebugGroupMultithreading	long	4	The debug group for logging of threading issues.
DebugGroupParsing	long	23	The debug group for logging of HTML parsing issues and other HTML-related issues.
DebugGroupQualifiers	long	11	The debug group for logging of qualifier issues.The debug group for logging of formatting exceptions and problems.
DebugGroupReflection	long	24	The debug group for logging of class introspection issues.
DebugGroupRequestHandling	long	25	The debug group for logging of issues related to the request-response loop.
DebugGroupResources	long	5	The debug group for logging of resource loading / lookup exceptions and problems.
DebugGroupRules	long	21	The debug group for logging of dynamic rule system issues and logging issues.
DebugGroupSQLGeneration	long	17	The debug group for logging of SQL generation issues and logging.
DebugGroupTiming	long	14	The debug group for logging of dynamic rule system issues.
DebugGroupUserInterface	long	18	The debug mask for logging of widget set and view exceptions and problems.
DebugGroupValidation	long	7	The debug group for logging of validation exceptions and problems.
DebugGroupWebObjects	long	2	The debug group for enabling logging of general WebObjects framework issues.
DebugLevelOff	int	0	Logs no messages. The default.

Constant	Type	Value	Description
DebugLevelCritical	int	1	Logs debug messages that should not be sent in non-debug mode, such as stack traces. Logging with this debug level is not likely to affect the performance of your application.
DebugLevelInformational	int	2	Logs debug messages that don't qualify as computationally intensive. Logging with this debug level will slow your application only moderately.
DebugLevelDetailed	int	3	Logs debug messages that qualify as computationally intensive, such as JDBC function calls. Logging with this debug level will slow your application considerably.

Method Types

Setting the debug groups mask

setAllowedDebugGroups

Setting and retrieving the debug level

setAllowedDebugLevel

allowedDebugLevel

Adding and removing debug groups to the mask

allowDebugLoggingForGroups

refuseDebugLoggingForGroups

Determining if debug logging is enabled for a particular debug group or groups mask

debugLoggingAllowedForLevel

debugLoggingAllowedForGroups

debugLoggingAllowedForLevelAndGroups

Redirecting output to custom PrintStreams

setDebug

setErr

setOut

Convenience methods

printStreamForPath

throwableAsString

Static Methods

allowDebugLoggingForGroups

```
public static synchronized void allowDebugLoggingForGroups(
    long debugGroups)
```

Enables logging for the debug group or the debug groups mask specified by debugGroups by adding it (via bitwise inclusive OR) to the debug groups mask. For instance, to allow debug logging for EOF-related issues, add the EOF debug group, DebugGroupEnterpriseObjects to the mask:

```
NSLog.allowDebugLoggingForGroups(DebugGroupEnterpriseObjects);
```

This differs from setAllowedDebugGroups in that allowDebugLoggingForGroups *adds* debug groups and debug groups masks to the bit mask in NSLog. setAllowedDebugGroups, however, *replaces* the bit mask in NSLog with the debug group or debug groups mask it is passed. For example,

```
// This will add the EOF debug group to the mask:
NSLog.allowDebugLoggingForGroups(DebugGroupEnterpriseObjects);

// This will replace all debug groups and masks in the bit mask with the EOF debug group
NSLog.setAllowedDebugGroups(DebugGroupEnterpriseObjects);
```

Use the refuseDebugLoggingForGroups method to disallow specific debug groups.

By default, logging is allowed for all debug groups. If you set the debug level to be low (for example, DEBUG_LEVEL-DETAILED), the debugging output may be overwhelming.

See "Creating Custom Debug Groups" (page 128) for more information.

allowedDebugLevel

```
public static int allowedDebugLevel()
```

Returns the allowed debug level.

debugLoggingAllowedForLevel

```
public static boolean debugLoggingAllowedForLevel(int aDebugLevel)
```

Returns `true` if debug logging is allowed for `aDebugLevel`. Debug logging is allowed if `aDebuglevel` is less than or equal to the debug level set by `setAllowedDebugLevel`. This is because the highest debug level, `DebugLevelDetailed`, implies `DebugLevelInformational` which implies `DebugLevelCritical`.

See the "Debug Levels" (page 127) for more information.

debugLoggingAllowedForGroups

```
public static boolean debugLoggingAllowedForGroups(long debugGroups)
```

Returns `true` if the debug groups specified by `debugGroups` are enabled (that is, the debug groups are part of the debug groups bit mask).

See the "Creating Custom Debug Groups" (page 128) for code examples.

debugLoggingAllowedForLevelAndGroups

```
public static boolean debugLoggingAllowedForLevelAndGroups(
    int aDebugLevel,
    long debugGroups)
```

Returns `true` if the debug groups specified by `debugGroups` are enabled, and if the debug level is less than or equal to the debug level set by `setAllowedDebugLevel`.

printStreamForPath

```
public static java.io.PrintStream printStreamForPath(String aPath)
```

Returns a java.io.PrintStream to a file at the specified path. Returns null if there is any problem with the path (i.e. it doesn't exist), or with the file at that path (i.e. it's a path to a directory, so a PrintStream can't be created for it).

refuseDebugLoggingForGroups

```
public static synchronized void refuseDebugLoggingForGroups(
    long debugGroups)
```

Disables debug logging for the debug groups mask specified by debugGroups.

By default, logging is allowed for all debug groups. If you set the debug level to be low (for example, DEBUG_LEVEL-DETAILED), the debugging output may be overwhelming.

See allowDebugLoggingForGroups for more information.

setAllowedDebugGroups

```
public static void setAllowedDebugGroups(
    long debugGroups)
```

Determines the set (the bit mask) of allowed debug groups. Typically, this method is invoked at the beginning of the application execution, since it affects logging behavior in the entire application, and completely overrides the global debug groups mask. If you want to turn logging on for a smaller scope and for a shorter period of execution, you should use allowDebugLoggingForGroups and refuseDebugLoggingForGroups.

Note: This method completely overrides the global debug groups mask, and affects logging behavior in the entire application. It is suggested that you only use it once at the beginning of application execution.

By default, logging is allowed for all debug groups. If you set the debug level to be low (for example, DEBUG_LEVEL-DETAILED), the debugging output may be overwhelming.

See allowDebugLoggingForGroups for an explanation of when to use that method or setAllowedDebugGroups.

setAllowedDebugLevel

```
public static void setAllowedDebugLevel(int aDebugLevel)
```

Sets the debug level to aDebugLevel. Throws an IllegalArgumentException if aDebugLevel is invalid. Typically, this method is invoked at the beginning of the application execution, since it affects logging behavior in the entire application.

By default, logging is allowed for all debug groups. If you set the debug level to be low (for example, DEBUG_LEVEL-DETAILED), the debugging output may be overwhelming.

See the “Constants” section in this document for a list of predefined debug levels.

setDebug

```
public static void setDebug(NSLog.Logger aLogger)
```

Sets the debugging logger NSLog.debug to aLogger. By default, NSLog.debug is a NSLog.PrintStreamLogger that sends its output to System.err. NSLog.debug is used for status and error messages that will conditionally be shown.

setErr

```
public static void setErr(NSLog.Logger aLogger)
```

Sets the logger NSLog.err to aLogger. By default, NSLog.err is a NSLog.PrintStreamLogger that sends its output to System.err. NSLog.err is used for error messages that will always be shown.

setOut

```
public static void setOut(NSLog.Logger aLogger)
```

Sets the logger NSLog.out to aLogger. By default, NSLog.out is a NSLog.PrintStreamLogger that sends its output to System.out. NSLog.out is used for status messages that will always be shown.

throwableAsString

```
public static String throwableAsString(Throwable aThrowable)
```

Returns the stack trace of aThrowable as a string.

NSLog.Logger

Inherits from: Object

Package: com.webobjects.foundation

Class Description

NSLog.Logger is an abstract class that specifies the core functionality for NSLog.

You can subclass NSLog.Logger to add custom logging implementations based on Email, java.io.PrintWriters, display to a Swing window, etc. To add custom logging implementations based on java.io.PrintStream, subclass NSLog.PrintStreamLogger. If you subclass NSLog.Logger, you need only implement two of the `appendln` methods: `appendln(Object)`, since the other `appendln` methods invoke `appendln(Object)`; and `appendln()`. You must also implement `flush()` if you subclass.

See the class specification on "NSLog" (page 125) and "NSLog.PrintStreamLogger" (page 145) for more information.

Method Types

Appending to output

`appendln`

Maintaining logging options

isEnabled

isVerbose

setIsEnabled

setIsVerbose

Flushing the log

flush

Constructors

NSLog.Logger

```
public NSLog.Logger()
```

Description forthcoming.

Instance Methods

appendln

```
public abstract void appendln(Object anObject)
```

Since this is an abstract method, it does nothing by default. It's up to the subclass to implement the behavior. As implemented in NSLog, this method appends the string representation of `anObject` to the logging output. For example, a generic object passed to this method might output "java.lang.Object@67e5d". The string representation is derived from the `toString()` method of the object.

```
public void appendln(Throwable aThrowable)
```

Calls `appendln(Object anObject)` with `NSLog.throwableAsString(aThrowable)` as an argument.

```
public void appendln(int anInt)
```

Calls `appendln(Object anObject)`, by transforming `anInt` into a Java Integer class object.

```
public void appendln(float aFloat)
```

Calls `appendln(Object anObject)` by transforming `aFloat` into a Java Float class object.

```
public void appendln(short aShort)
```

Calls `appendln(Object anObject)`, by transforming `aShort` into a Java Short class object.

```
public void appendln(long aLong)
```

Calls `appendln(Object anObject)`, by transforming `anInt` into a Java Long class object.

```
public void appendln(byte[] aByteArray)
```

Calls `appendln(Object anObject)`, by transforming `aByte[]` into a Java String class object.

```
public void appendln(char[] aCharArray)
```

Calls `appendln(Object anObject)`, by transforming `aChar[]` into a Java String class object.

```
public void appendln(boolean aBoolean)
```

Calls `appendln(Object anObject)`, passing `true` if `aBoolean` is true, `false` if `aBoolean` is false.

```
public void appendln(double aDouble)
```

Calls `appendln(Object anObject)`, by transforming `aDouble` into a Java Double class object.

```
public void appendln(char aChar)
```

Calls `appendln(Object anObject)`, by transforming `aChar[]` into a Java String class object.

```
public void appendln(byte aByte)
```

Calls `appendln(Object anObject)`, by transforming `aByte` into a Java Byte class object.

```
public void appendln()
```

Since this is an abstract method, it does nothing by default. As implemented in NSLog, this method appends a new line to the logging output.

flush

```
public abstract void flush()
```

Since this is an abstract method, it does nothing by default. As implemented in NSLog, this method allows you to flush the internal buffer.

isEnabled

```
public boolean isEnabled()
```

Returns the value of an internal boolean, which defaults to `true`, and is set by `setIsEnabled`. As implemented in NSLog, the internal boolean regulates whether logging is enabled or disabled. When logging is disabled, the receiver ignores all invocations of `appendln`. By default, logging is enabled.

isVerbose

```
public boolean isVerbose()
```

Returns the value of an internal boolean, which defaults to `true`, and is set by `setIsVerbose`. As implemented in NSLog, the internal boolean regulates whether verbose logging is activated or deactivated. See the method description for `setIsVerbose` for more information. By default, verbose logging is disabled.

setIsEnabled

```
public void setIsEnabled(boolean aBoolean)
```

Sets the value of an internal boolean to `aBoolean`. As implemented in NSLog, the internal boolean disables logging if `aBoolean` is false. When logging is disabled, the receiver ignores all invocations of `appendln`. By default, logging is enabled.

setIsVerbose

```
public void setIsVerbose(boolean aBoolean)
```

Sets the value of an internal boolean to `aBoolean`. As implemented in NSLog, the internal boolean enables verbose logging if `aBoolean` is true. Verbose logging produces output of the format: "*[Current Time] <Current Thread Name> object* ". By default, verbose logging is disabled in NSLog.

NSLog.PrintStreamLogger

Inherits from: NSLog.Logger

Package: com.webobjects.foundation

Class Description

NSLog.PrintStreamLogger is a concrete subclass of NSLog.Logger. It logs output to a java.io.PrintStream which is contained by the logger. This PrintStream can be changed, which causes the receiver to output log messages somewhere else, such as a local file. Methods are provided to enable and disable logging, and to enable and disable verbose logging.

NSLog.out and NSLog.debug are PrintStreamLoggers that point at System.out. NSLog.err is a PrintStreamLogger that points to System.err.

NSLog.PrintStreamLogger looks at the value of the internal variables set by NSLog.Logger.setIsVerbose() and NSLog.Logger.setIsEnabled() to determine whether to produce verbose output and to determine whether to log messages to the logger. See the method descriptions for these methods in the documentation for “NSLog.Logger” (page 139).

Method Types

All methods

NSLog.PrintStreamLogger

appendln

appendln

flush

printStream

setPrintStream

Constructors

NSLog.PrintStreamLogger

public NSLog.PrintStreamLogger(java.io.PrintStream aPrintStream)

Creates a new NSLog.PrintStreamLogger which directs output to aPrintStream. Throws an IllegalArgumentException if aPrintStream is null.

public NSLog.PrintStreamLogger()

Creates a new NSLog.PrintStreamLogger which directs output to System.out.

Instance Methods

appendln

```
public void appendln()
```

Writes a new line to the receiver's PrintStream.

appendln

```
public void appendln(Throwable aThrowable)
```

Writes the stack trace of `aThrowable` to the receiver's PrintStream.

appendln

```
public void appendln(Object anObject)
```

Writes `anObject` to the receiver's PrintStream.

flush

```
public void flush()
```

Flush's the receiver's PrintStream by invoking the PrintStream's `flush` method.

printStream

```
public java.io.PrintStream printStream()
```

Returns the receiver's PrintStream.

setPrintStream

```
public void setPrintStream(java.io.PrintStream aPrintStream)
```

Sets the receiver's print stream to `aPrintStream`. This redirects the receiver's output. For example, to redirect all log messages to the local file `/Local/Users/log.txt`, use the following code:

```
NSLog.PrintStreamLogger aLogger =
     new NSLog.PrintStreamLogger(); // Output defaults to System.out
PrintStream aStream =
     NSLog.printStreamForPath ("/Local/Users/log.txt"); // New print stream based on path.
aLogger.setPrintStream (aStream); // Direct output to the custom PrintStream.
NSLog.setOut(aLogger); // Assign the custom PrintStreamLogger to the declared instances
of NSLogger in NSLog
NSLog.setErr(aLogger);
NSLog.setDebug(aLogger);
aLogger.appendln(anObject);
```

NSMultiReaderLock

Inherits from:	Object
Implements:	NSLocking
Package:	com.webobjects.foundation

Class Description

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

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

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

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

Method Types

Constructors

NSMultiReaderLock

Managing reader locks

lockForReading

retrieveReaderLocks

suspendReaderLocks

tryLockForReading

unlockForReading

Managing writer locks

lock

lockForWriting

tryLockForWriting

unlock

unlockForWriting

Methods inherited from Object

toString

Constructors

NSMultiReaderLock

```
public NSMultiReaderLock()
```

Creates an NSMultiReaderLock object.

Instance Methods

lock

```
public void lock()
```

Conformance to NSLocking. See the method description of `lock` in the interface description for NSLocking. This method is equivalent to `lockForWriting`.

lockForReading

```
public void lockForReading()
```

Acquires a reader lock for the current thread. If the current thread doesn't already have a lock, the method blocks if there are any waiting or active writer locks. If the current thread already has a lock (reader or writer), the lock request count is incremented.

lockForWriting

```
public void lockForWriting()
```

Gets a writer lock for the current thread. If the current thread already has one, the lock request count is incremented, but a new lock is not taken. If the requesting thread has outstanding reader locks, they are temporarily dropped until the writer lock is returned. If other threads have outstanding reader locks, this method blocks until all reader locks have been freed.

retrieveReaderLocks

```
public void retrieveReaderLocks()
```

Reinstates the current thread's reader locks that have been suspended using `suspendReaderLocks`.

suspendReaderLocks

```
public void suspendReaderLocks()
```

Temporarily relinquishes all of the current thread's reader locks, releasing the lock if all reader locks are unlocked. To reinstate the current thread's suspended reader locks, use the `retrieveReaderLocks` method.

toString

```
public String toString()
```

Returns a string representation of the receiver containing the current thread name and a table with the names and reader lock counts of all the receiver's threads.

tryLockForReading

```
public boolean tryLockForReading()
```

Returns `true` if the current thread is able to immediately obtain a reader lock. There are three ways this can happen:

1. There are no outstanding writer locks.
2. The writer lock is held by the current thread.
3. The current thread already has a reader lock.

This method implicitly calls `lockForReading`, so you must call `unlockForReading` if `tryLockForReading` returns `true`.

tryLockForWriting

```
public boolean tryLockForWriting()
```

Returns `true` if the current thread is able to immediately obtain a writer lock. Returns `false` if another thread already has the lock or is queued to receive it. This method implicitly calls `lockForWriting`, so you must call `unlockForWriting` if `tryLockForWriting` returns `true`.

unlock

```
public void unlock()
```

Conformance to NSLocking. See the method description of `unlock` in the interface description for NSLocking. This method is equivalent to `unlockForWriting`.

unlockForReading

```
public void unlockForReading()
```

Releases a reader lock for the current thread. Each `lockForReading` message must be paired with an `unlockForReading` message before the lock is actually released. Invoking this method when the lock count is zero does nothing.

unlockForWriting

```
public void unlockForWriting()
```

Releases a writer lock for the current thread. Each `lockForWriting` message must be paired with an `unlockForWriting` message before the lock is actually released. When the writer lock is released, it checks to see if the thread previously had any reader locks. If so, the reader lock count is restored. Invoking this method when the lock count is zero does nothing.

NSMutableArray

Inherits from: NSArray

Package: com.webobjects.foundation

Class Description

The NSMutableArray defines the programmatic interface for managing collections of objects called *arrays*. It adds insertion and deletion operations to the basic array-handling behavior inherited from its superclass, NSArray.

Table 0-7 describes the NSMutableArray methods that provide the basis for all NSMutableArray's other methods; that is, all other methods are implemented in terms of these. If you create a subclass of NSMutableArray, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-7 NSMutableArray's Base API

Method	Description
`addObject`	Adds an object to the array.
`addObjects`	Adds multiple objects to the array.
`insertObjectAtIndex`	Inserts an object into the array at a specified index.
`removeAllObjects`	Empties the receiver of all its elements.

Table 0-7 NSMutableArray's Base API

Method	Description
`removeObjectAtIndex`	Removes the object at a specified index from the array.
`replaceObjectAtIndex(Object, int)`	Replaces the object at a specified index with another object.
`setArray`	Sets an array's elements to the ones in another array.
`sortUsingComparator`	Sorts the elements of the array.

The other methods provide convenient ways of inserting an object into a specific slot in the array and removing an object based on its identity or position in the array.

Method Types

Creating mutable arrays

`NSMutableArray`

`immutableClone`

`clone`

Adding and replacing objects

`addObject`

`addObjects`

`addObjectsFromArray`

`insertObjectAtIndex`

`replaceObjectAtIndex`

`replaceObjectsInRange`

`setArray`

Removing objects

removeAllObjects

removeIdenticalObject

removeLastObject

removeObject

removeObjectAtIndex

removeObjects

removeObjectsInArray

removeObjectsInRange

Rearranging objects

sortUsingComparator

Constructors

NSMutableArray

```
public NSMutableArray()
```

Creates an empty mutable array.

```
public NSMutableArray(int capacity)
```

Creates an empty mutable array with enough allocated memory to hold the number of objects specified by capacity, a number greater than 0. NSMutableArrays expand as needed, so capacity simply establishes the object's initial capacity.

```
public NSMutableArray(NSArray anArray)
```

Creates a mutable array containing the objects in anArray.

```
public NSMutableArray(Object anObject)
```

Creates a mutable array containing the single element anObject.

```
public NSMutableArray(Object[] objects)
```

Creates a mutable array containing objects.

```
public NSMutableArray(
    Object[] objects,
    NSRange aRange)
```

Creates a mutable array containing the objects from objects in the range specified by aRange. After an immutable array has been initialized in this way, it can't be modified.

```
public NSMutableArray(
    java.util.Vector aVector,
    NSRange aRange,
    boolean checkForNull)
```

Creates a mutable array containing the objects from aVector in the range specified by aRange. The checkForNull argument controls the method's behavior when it encounters a null value in the vector: if checkForNull is true, the null value is simply ignored. If checkForNull is false, the method raises an IllegalArgumentException.

Instance Methods

addObject

```
public void addObject(Object anObject)
```

Inserts anObject at the end of the receiver. If anObject is null, an IllegalArgumentException is thrown.

addObjects

```
public void addObjects(Object[] otherArray)
```

Adds the objects contained in otherArray to the end of the receiver's array of objects. If any of the objects in otherArray are null, an IllegalArgumentException is thrown.

addObjectsFromArray

```
public void addObjectsFromArray(NSArray anArray)
```

Adds the objects contained in anArray to the end of the receiver's array of objects.

clone

```
public Object clone()
```

Creates a clone of the receiver. NSMutableArray's implementation simply creates a new NSMutableArray with the objects in the receiver.

immutableClone

```
public NSArray immutableClone()
```

Returns a copy of the receiver as an immutable NSArray.

insertObjectAtIndex

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

Inserts anObject into the receiver at index. If index is already occupied, the objects at index and beyond are shifted down one slot to make room. index cannot be greater than the number of elements in the array. This method throws an IllegalArgumentException if anObject is null or if index is greater than the number of elements in the array.

Note that NSArrays are not like C arrays. That is, even though you might specify a size when you create an array, the specified size is regarded as a hint; the actual size of the array is still 0. Because of this, you can only insert new objects in ascending order—with no gaps. Once you add two objects, the array's size is 2, so you can add objects at indexes 0, 1, or 2. Index 3 is illegal and out of bounds; if you try to add an object at index 3 (when the size of the array is 2), NSMutableArray throws an exception.

mutableClone

```
public NSMutableArray mutableClone()
```

Description forthcoming.

removeAllObjects

```
public void removeAllObjects()
```

Empties the receiver of all its elements.

removeIdenticalObject

```
public void removeIdenticalObject(Object anObject)

public void removeIdenticalObject(
    Object anObject,
    NSRange aRange)
```

Removes all occurrences of `anObject` throughout the array; or if `aRange` provided, removes all occurrences of `anObject` in the specified range. These methods use the `indexOfIdenticalObject` method to locate matches and remove them by using `removeObjectAtIndex`. Throws an IllegalArgumentException if `anObject` is `null` or if `aRange` is out of bounds.

removeLastObject

```
public void removeLastObject()
```

Removes the receiver's element with the highest-valued index. Throws an IllegalArgumentException if there are no objects in the array.

removeObject

```
public void removeObject(Object anObject)

public void removeObject(
    Object anObject,
    NSRange aRange)
```

Removes all occurrences of anObject throughout the array; or if aRange provided, removes all occurrences of anObject in the specified range. These methods use the indexOfObject method to locate matches and remove them by using removeObjectAtIndex. Thus, matches are determined on the basis of an object's response to the equals message. Throws an IllegalArgumentException if anObject is null or if aRange is out of bounds.

removeObjectAtIndex

```
public void removeObjectAtIndex(int index)
```

Removes the object at index and moves all elements beyond index up one slot to fill the gap. This method throws a IllegalArgumentException if the array is empty or if index is beyond the end of the array.

removeObjects

```
public void removeObjects(Object[] objects)
```

This method is similar to removeObject, but allows you to efficiently remove the set of objects in objects with a single operation.

removeObjectsInArray

```
public void removeObjectsInArray(NSArray otherArray)
```

This method is similar to removeObject, but allows you to efficiently remove the set of objects in otherArray with a single operation.

removeObjectsInRange

```
public void removeObjectsInRange(NSRange aRange)
```

Removes each of the objects within the specified range in the receiver using `removeObjectAtIndex`. Throws an IllegalArgumentException if `aRange` is out of bounds.

replaceObjectAtIndex

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

Replaces the object at *index* with *anObject*. This method throws an IllegalArgumentException if *anObject* is `null` or if *index* is beyond the end of the array.

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

This method is deprecated. Use `replaceObjectAtIndex(Object, int)` instead.

replaceObjectsInRange

```
public void replaceObjectsInRange(
    NSRange aRange,
    NSArray otherArray,
    NSRange otherRange)
```

Replaces the objects in the receiver specified by `aRange` with the objects in `otherArray` specified by `otherRange`. `aRange` and `otherRange` don't have to be equal; if `aRange` is greater than `otherRange`, the extra objects in the receiver are removed. If `otherRange` is greater than `aRange`, the extra objects from `otherArray` are inserted into the receiver.

setArray

```
public void setArray(NSArray otherArray)
```

Sets the receiver's elements to those in otherArray. Shortens the receiver, if necessary, so that it contains no more than the number of elements in otherArray. Replaces existing elements in the receiver with the elements in otherArray. If there are more elements in otherArray than there are in the receiver, the additional items are added.

sortUsingComparator

```
public void sortUsingComparator(NSComparator aComparator)
    throws NSComparator.ComparisonException
```

Sorts the receiver's elements, as determined by aComparator. Throws an NSComparator.Exception if aComparator is null.

See Also: sortedArrayUsingComparator (NSArray)

NSMutableData

Inherits from:	NSData : Object
Implements:	Cloneable java.io.Serializable NSCoding
Package:	com.webobjects.foundation

Class Description

The NSMutableData class declares the programmatic interface to an object that contains modifiable data in the form of bytes. The data grows automatically if necessary.

Table 0-8 describes the NSMutableData methods that provide the basis for all NSMutableData's other methods; that is, all other methods are implemented in terms of these nine. If you create a subclass of NSMutableData, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-8 NSMutableData's Base API

Method	Description
`appendByte`	Appends a byte to the receiver.
`appendBytes`	Appends the contents of a byte array to the receiver. The two-argument version is part of the base API.
`bytesNoCopy`	Returns the internal byte array that contains the receiver's data. Inherited from NSData.
`immutableBytes`	Returns an immutable byte array that contains a copy of the receiver's data.
`immutableRange`	Returns an immutable copy of the NSRange object that specifies the receiver's length.
`rangeNoCopy`	Returns the internal NSRange object that specifies the receiver's length. Inherited from NSData.
`resetBytesInRange`	Resets to zero the receiver's bytes that fall within the specified range.
`setData`	Replaces the receiver's contents with the specified NSData object.
`setLength`	Extends or truncates a mutable data object to the specified length.

To modify the data, use the `setData`, `appendByte`, `appendBytes`, and `appendData` methods. If you want to set a range of bytes to zero, use the `resetBytesInRange` method. To change the length of the data, use the `setLength` and `increaseLengthBy` methods.

Interfaces Implemented

Cloneable

clone

java.io.Serializable

NSCoding

classForCoder

decodeObject

encodeWithCoder

Method Types

Constructors

NSMutableData

Modifying the data

appendByte

appendBytes

appendData

resetBytesInRange

setData

Modifying the range

increaseLengthBy

setLength

Accessing internal data directly

immutableBytes

immutableRange

Constructors

NSMutableData

```
public NSMutableData()
```

Creates an empty NSMutableData object.

```
public NSMutableData(NSData data)
```

Creates an NSMutableData object containing the contents of another data object data.

```
public NSMutableData(String string)
```

This constructor is deprecated. Use `NSMutableData(string.getBytes())` instead.

```
public NSMutableData(byte[] bytes)
```

Creates a NSMutableData object with all the data in the byte array bytes.

```
public NSMutableData(
    byte[] bytes,
    NSRange range)
```

Creates an NSMutableData object with the bytes from the language array bytes that fall in the range specified by range.

```
public NSMutableData(
    byte[] bytes,
    NSRange range,
    boolean nocopy)
```

Creates an NSMutableData object with the bytes from the language array bytes that fall in the range specified by range. The noCopy parameter specifies whether or not a copy of bytes is made.

```
public NSMutableData(int capacity)
```

Creates an NSMutableData object prepared to store at least `capacity` bytes. If you know the upper bound on the size of your data, you can use this constructor to improve performance. As long as the data size does not exceed `capacity` bytes, the internal byte array will not be reallocated.

```
public NSMutableData(java.io.File file) throws java.io.IOException
```

This constructor is deprecated. Use `NSMutableData(new FileInputStream(file),myChunkSize)` instead.

```
public NSMutableData(
    java.io.InputStream inputStream,
    int chunkSize) throws java.io.IOException
```

Creates a data object with the data from the stream specified by `inputStream`. The `chunkSize` parameter specifies the size, in bytes, of the block that the input stream returns when it reads. For maximum performance, you should set the chunk size to the approximate size of the data. This constructor does not close the stream.

```
public NSMutableData(java.net.URL url) throws java.io.IOException
```

This constructor is deprecated. Use the following code instead:

```
URLConnection connection = url.openConnection();
connection.connect();
NSMutableData myData = new NSMutableData(connection.getInputStream(),myChunkSize);
```

Instance Methods

appendByte

```
public void appendByte(byte byte)
```

Appends the specified byte to the receiver.

See Also: appendBytes, appendData

appendBytes

```
public void appendBytes(byte[] bytes[])

public void appendBytes(
    byte[] bytes,
    NSRange range)
```

Appends the contents of byte array bytes to the receiver. The two-argument method appends the bytes in bytes that fall within the range specified by range.

See Also: appendByte, appendData

appendData

```
public void appendData(NSData otherData)
```

Appends the contents of a data object otherData to the receiver.

See Also: appendByte, appendBytes

clone

```
public Object clone()
```

Returns a copy (an NSMutableData object) of the receiver.

immutableBytes

```
protected byte[] immutableBytes()
```

Returns an immutable copy of the byte array that contains the receiver's data.

immutableRange

```
protected NSRange immutableRange()
```

Returns an immutable copy of the NSRange object that contains the receiver's length.

increaseLengthBy

```
public void increaseLengthBy(int additionalLength)
```

Increases the length of the receiver by additionalLength. The additional bytes are all set to zero.

See Also: setLength

resetBytesInRange

```
public void resetBytesInRange(NSRange range)
```

Resets to zero the receiver's bytes that fall within the specified range. If the location of range isn't within the receiver's range of bytes, an IllegalArgumentException is thrown. The receiver is resized to accommodate the new bytes, if necessary.

setData

```
public void setData(NSData data)
```

Replaces the entire contents of the receiver with the contents of data.

See Also: setLength

setLength

```
public void setLength(int length)
```

Extends or truncates a mutable data object to the specified length. If the mutable data object is extended, the additional bytes are filled with zero.

See Also: increaseLengthBy, setData

NSMutableDictionary

Inherits from:	NSDictionary : Object
Implements:	Cloneable java.io.Serializable NSCoding NSKeyValueCoding NSKeyValueCodingAdditions
Package:	com.webobjects.foundation

Class Description

The NSMutableDictionary class declares the programmatic interface to objects that manage mutable associations of keys and values. This class adds modification operations to the basic operations it inherits from NSDictionary.

Methods that add entries to NSMutableDictionaries—whether during construction or modification—add each value object to the dictionary directly. These methods also add each key object directly to the dictionary, which means that you must ensure that the keys do not change. If you expect your keys to change for any reason, you should make copies of the keys and add the copies to the dictionary.

Table 0-9 describes the NSMutableDictionary methods that provide the basis for all NSMutableDictionary's other methods; that is, all other methods are implemented in terms of these seven. If you create a subclass of NSMutableDictionary, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-9 NSMutableDictionary's Base API

Method	Description
`count`	Returns the number of entries in the dictionary. Inherited from NSDictionary.
`objectForKey`	Returns the value associated with a given key. Inherited from NSDictionary.
`keysNoCopy`	Returns a natural language array containing the keys in the dictionary. Inherited from NSDictionary.
`objectsNoCopy`	Returns a natural language array containing the objects in the dictionary. Inherited from NSDictionary.
`removeAllObjects`	Empties the dictionary of its entries.
`removeObjectForKey`	Removes the specified key object and its associated value object from the dictionary.
`setObjectForKey`	Adds or replaces an entry to the receiver consisting of the specified key and value objects.

The other methods declared here provide convenient ways of adding or removing multiple entries at a time.

Interfaces Implemented

Cloneable

- clone

java.io.Serializable

NSCoding

- classForCoder
- decodeObject
- encodeWithCoder

NSKeyValueCoding

- takeValueForKey
- valueForKey

NSKeyValueCodingAdditions

- takeValueForKeyPath
- valueForKeyPath

Method Types

Constructors

- NSMutableDictionary

Adding and removing entries

- addEntriesFromDictionary

removeAllObjects

removeObjectForKey

removeObjectsForKeys

setDictionary

setObjectForKey

Copying the dictionary

immutableClone

Methods inherited from Object

clone

Constructors

NSMutableDictionary

```
public NSMutableDictionary()
```

Creates an empty NSMutableDictionary.

```
public NSMutableDictionary(int capacity)
```

Creates an empty NSMutableDictionary prepared to hold at least capacity entries.

```
public NSMutableDictionary(
    NSArray objectArray,
    NSArray keyArray)
```

Creates an NSMutableDictionary with entries from the contents of the keyArray and objectArray NSArrays. This method steps through objectArray and keyArray, creating entries in the new dictionary as it goes. Each key object and its corresponding value object is added directly to the dictionary. An InvalidArgumentException is thrown if the objectArray and keyArray do not have the same number of elements.

Note: NSMutableDictionary assumes that key objects are immutable. If your key objects are mutable, you should make copies of them and add the copies to the dictionary.

```
public NSMutableDictionary(NSDictionary dictionary)
```

Creates an NSMutableDictionary containing the keys and values found in `dictionary`.

```
public NSMutableDictionary(
    Object object,
    Object key)
```

Creates an NSMutableDictionary containing a single object `object` for a single key `key`.

Note: NSMutableDictionary assumes that key objects are immutable. If your key objects are mutable, you should make copies of them and add the copies to the dictionary.

```
public NSMutableDictionary(
    Object[] objects,
    Object[] keys)
```

Creates an NSMutableDictionary with entries from the contents of the `keys` and `objects` arrays. This method steps through `objects` and `keys`, creating entries in the new dictionary as it goes. Each key object and its corresponding value object is added directly to the dictionary. An InvalidArgumentException is thrown if the `objects` and `keys` do not have the same number of elements.

Note: NSMutableDictionary assumes that key objects are immutable. If your key objects are mutable, you should make copies of them and add the copies to the dictionary.

```
public NSMutableDictionary(
    java.util.Dictionary dictionary,
    boolean ignoreNull)
```

Creates an NSMutableDictionary containing the keys and values found in `dictionary`. If `ignoreNull` is `false`, throws an InvalidArgumentException if any key or value in `dictionary` is `null`.

Instance Methods

addEntriesFromDictionary

```
public void addEntriesFromDictionary(NSDictionary otherDictionary)
```

Adds the entries from `otherDictionary` to the receiver.

See Also: `setObjectForKey`

clone

```
public Object clone()
```

Returns a copy (a NSMutableDictionary object) of the receiver.

immutableClone

```
public NSDictionary immutableClone()
```

Returns an immutable copy (an NSDictionary object) of the receiver.

mutableClone

```
public NSMutableArray mutableClone()
```

Description forthcoming.

removeAllObjects

```
public void removeAllObjects()
```

Empties the dictionary of its entries.

See Also: `removeObjectForKey`, `removeObjectsForKeys`

removeObjectForKey

```
public Object removeObjectForKey(Object key)
```

Removes the dictionary entry identified by key and returns the entry's value object. If no entry identified by key exists, this method returns null.

See Also: removeObjectsForKeys, removeAllObjects

removeObjectsForKeys

```
public void removeObjectsForKeys(NSArray keyArray)
```

Removes one or more objects from the receiver. The entries are identified by the keys in keyArray. This method does not raise if the receiver does not contain entries for one or more of the keys.

See Also: removeObjectForKey, removeAllObjects

setDictionary

```
public void setDictionary(NSDictionary otherDictionary)
```

Sets the receiver to entries in otherDictionary. This method removes all entries from the receiver (with removeAllObjects) and adds each entry from otherDictionary into the receiver.

setObjectForKey

```
public void setObjectForKey(
    Object anObject,
    Object aKey)
```

Adds or replaces an entry to the receiver consisting of aKey and its corresponding value object anObject. Throws an InvalidArgumentException if the key or value object is null.

Note: NSMutableDictionary assumes that key objects are immutable. If your key objects are mutable, you should make copies of them and add the copies to the dictionary.

See Also: removeObjectForKey

takeValueForKey

```
public void takeValueForKey(
    Object value,
    String key)
```

Conformance to NSKeyValueCoding. Invokes `setObjectForKey` with the specified parameters if `value` is not `null`. Otherwise invokes `removeObjectForKey` for the specified key.

Note: NSMutableDictionary assumes that key objects are immutable. If your key objects are mutable, you should make copies of them and add the copies to the dictionary.

NSMutableRange

Inherits from:	NSRange : Object
Implements:	Cloneable
Package:	com.webobjects.foundation

Class Description

An NSMutableRange is an object representing a range that can be changed. A range is a measurement of a segment of something linear, such as a byte stream. You can change an NSMutableRange's two primary values, its location and its length. The methods of NSMutableRange also enable you to alter an NSMutableRange based on its union or intersection with another NSRange object.

The main purpose for NSMutableRange is to provide a way for methods to return range values in an "out" parameter. A client creates and passes in one or more NSMutableRanges to a method and gets back changed objects when the method returns. NSMutableRanges are also useful for performance reasons; instead of creating multiple NSRanges in a loop, you can create just one NSMutableRange and reuse it.

Table 0-10 describes the NSMutableRange methods that provide the basis for all NSMutableRange's other methods; that is, all other methods are implemented in terms of these four. If you create a subclass of NSMutableRange, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-10 NSMutableRange's Base API

Method	Description
`location`	Returns the starting location of the receiver. Inherited from NSRange.
`length`	Returns the length of the receiver from its starting location. Inherited from NSRange.
`setLength`	Sets the length of the receiver.
`setLocation`	Sets the starting location of the receiver.

Interfaces Implemented

Cloneable

`clone`

Method Types

Constructors

`NSMutableRange`

Accessing and setting range elements

`setLength`

setLocation

Transforming ranges

intersectRange

unionRange

Methods inherited from Object

clone

Constructors

NSMutableRange

```
public NSMutableRange()
```

Creates and returns an empty NSMutableRange.

```
public NSMutableRange(NSRange aRange)
```

Creates a new NSMutableRange with the location and length values of aRange. This constructor is used in cloning the receiver.

```
public NSMutableRange(
    int location,
    int length)
```

Creates a new NSMutableRange with the range elements of location and length. Throws an IllegalArgumentException if either integer is negative.

Instance Methods

clone

```
public Object clone()
```

Returns a copy (a NSMutableRange object) of the receiver.

intersectRange

```
public void intersectRange(NSRange aRange)
```

Changes the receiver to the range resulting from the intersection of aRange and the receiver before the operation. Sets the receiver to an empty range if they do not intersect.

See Also: unionRange

setLength

```
public void setLength(int newLength)
```

Sets the length of the receiver to newLength. Throws an IllegalArgumentException if newLength is a negative value.

See Also: setLocation

setLocation

```
public void setLocation(int newLocation)
```

Sets the starting location of the receiver to newLocation. Throws an IllegalArgumentException if newLocation is a negative value.

See Also: setLength

unionRange

```
public void unionRange(NSRange aRange)
```

Changes the receiver to the range resulting from the union of aRange and the receiver before the operation. This is the lowest starting location and the highest ending location of the two NSRanges.

See Also: intersectRange

NSMutableSet

Inherits from:	NSSet : Object
Implements:	Cloneable java.io.Serializable NSCoding
Package:	com.webobjects.foundation

Class Description

The NSMutableSet class declares the programmatic interface to an object that manages a mutable set of objects. NSMutableSet provides support for the mathematical concept of a set. A set, both in its mathematical sense, and in the NSMutableSet implementation, is an unordered collection of distinct elements. The NSSet class supports creating and managing immutable sets.

Table 0-11 describes the NSMutableSet methods that provide the basis for all NSMutableSet's other methods; that is, all other methods are implemented in terms of these five. If you create a subclass of NSMutableSet, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-11 NSMutableSet's Base API

Method	Description
`count`	Returns the number of members in the set.
`member`	Returns the object in the set that is equal to the specified object.
`objectsNoCopy`	Returns the actual array of objects in the set.
`removeAllObjects`	Empties the set of all its members.
`removeObject`	Removes the specified object from the set.

Objects are removed from an NSMutableSet using any of the methods `intersectSet`, `removeAllObjects`, `removeObject`, or `subtractSet`.

Objects are added to an NSMutableSet with `addObject`, which adds a single object to the set; `addObjectsFromArray`, which adds all objects from a specified array to the set; or with `unionSet`, which adds all the objects from another set.

Methods that add entries to NSMutableSets—whether during construction or modification—add each member to the set directly. This means that you must ensure that the members do not change. If you expect your members to change for any reason, you should make copies of them and add the copies to the set.

Interfaces Implemented

Cloneable

clone

java.io.Serializable

NSCoding

classForCoder

decodeObject

encodeWithCoder

Method Types

Constructors

NSMutableSet

Adding and removing entries

addObject

addObjectsFromArray

removeAllObjects

removeObject

Combining and recombining sets

intersectSet

setSet

subtractSet

unionSet

Copying the set

immutableClone

Constructors

NSMutableSet

```
public NSMutableSet()
```

Creates an empty NSMutableSet.

```
public NSMutableSet(NSArray anArray)
```

Creates an NSMutableSet containing the objects in anArray.

Note: NSMutableSet assumes that member objects are immutable. If your member objects are mutable, you should make copies of them and add the copies to the set.

```
public NSMutableSet(NSSet aSet)
```

Creates an NSMutableSet containing the objects in aSet.

```
public NSMutableSet(Object anObject)
```

Creates an NSMutableSet containing a single object anObject.

Note: NSMutableSet assumes that member objects are immutable. If your member objects are mutable, you should make copies of them and add the copies to the set.

```
public NSMutableSet(Object[] objects[])
```

Creates an NSMutableSet containing the objects in the objects language array.

Note: NSMutableSet assumes that member objects are immutable. If your member objects are mutable, you should make copies of them and add the copies to the set.

```
public NSMutableSet(int capacity)
```

Creates an NSMutableSet that can hold at least capacity objects.

Instance Methods

addObject

```
public void addObject(Object anObject)
```

Adds the specified object to the receiver if it is not already a member. If anObject is already present in the set, this method has no effect on either the set or on anObject.

Note: NSMutableSet assumes that member objects are immutable. If your member objects are mutable, you should make copies of them and add the copies to the set.

See Also: addObjectsFromArray, unionSet

addObjectsFromArray

```
public void addObjectsFromArray(NSArray anArray)
```

Adds each object contained in anArray to the receiver, if that object is not already a member. If a given element of the array is already present in the set, this method has no effect on either the set or on the array element.

Note: NSMutableSet assumes that member objects are immutable. If your member objects are mutable, you should make copies of them and add the copies to the set.

See Also: addObject, unionSet

clone

```
public Object clone()
```

Creates a clone of the receiver. NSMutableSet's implementation simply creates a new NSMutableSet with the objects in the receiver.

immutableClone

```
public NSSet immutableClone()
```

Creates an immutable copy (a NSSet) of the receiver.

intersectSet

```
public void intersectSet(NSSet otherSet)
```

Removes from the receiver each object that isn't a member of otherSet.

See Also: removeObject, removeAllObjects, subtractSet

mutableClone

```
public NSMutableArray mutableClone()
```

Description forthcoming.

removeAllObjects

```
public void removeAllObjects()
```

Empties the set of all its members.

See Also: removeObject, intersectSet, subtractSet

removeObject

```
public void removeObject(Object anObject)
```

Removes anObject from the set.

See Also: removeAllObjects, intersectSet, subtractSet

setSet

```
public void setSet(NSSet otherSet)
```

Empties the receiver, then adds each object contained in otherSet to the receiver.

subtractSet

```
public void subtractSet(NSSet otherSet)
```

Removes from the receiver each object contained in otherSet that is also present in the receiver. If any member of otherSet isn't present in the receiving set, this method has no effect on either the receiver or on the otherSet member.

See Also: removeObject, removeAllObjects, intersectSet

unionSet

```
public void unionSet(NSSet otherSet)
```

Adds each object contained in otherSet to the receiver, if that object is not already a member. If any member of otherSet is already present in the receiver, this method has no effect on either the receiver or on the otherSet member.

See Also: addObject, addObjectsFromArray

NSNotification

Inherits from:	Object
Implements:	NSCoding java.io.Serializable
Package:	com.webobjects.foundation

Class Description

NSNotification objects encapsulate information so that it can be broadcast to other objects by an NSNotificationCenter object.

Notifications and their Rationale

The standard way to pass information between objects is message passing—one object invokes the method of another object. However, message passing requires that the object sending the message know who the receiver is and what messages it responds to. At times, this tight coupling of two objects is undesirable—most notably because it would join together two otherwise independent subsystems. For these cases, a broadcast model is introduced: An object posts a notification, which is dispatched to the appropriate observers through an NSNotificationCenter object, or simply notification center.

An NSNotification object (referred to as a notification) contains a name, an object, and a dictionary. The name is a tag identifying the notification. The object is any object that the poster of the notification wants to send to observers of that notification (typically, it is the object that posted the notification). The dictionary stores other related objects if any.

Any object may post a notification. Other objects can register themselves as observers to receive notifications when they are posted. The object posting the notification, the object included in the notification, and the observer of the notification may all be different objects or the same object. Objects that post notifications need not know anything about the observers. On the other hand, observers need to know at least the notification name and keys to the dictionary if provided.

NSNotification objects are immutable objects.

Notification Centers

The notification center manages the sending and receiving of notifications. When an object wants to receive a certain notification, it registers itself with the notification center. When an object has a notification to send, it sends it to the notification center. When the notification center receives a notification, it passes that notification along to all objects registered to receive it. (See the NSNotificationCenter class specification for more on posting notifications.)

This notification model frees an object from concern about what objects it should send information to. Any object may simply post a notification without knowing what objects—if any—are receiving the notification. However, objects receiving notifications do need to know at least the notification name if not the type of information the notification contains. The notification center takes care of broadcasting notifications to registered observers. Another benefit of this model is to allow multiple objects to listen for notifications, which would otherwise be cumbersome.

You can create a notification object with the constructor. However, you don't usually create your own notifications directly. The NSNotificationCenter method `postNotification` allows you to conveniently post a notification without creating it first.

Notification and Delegation

Using the notification system is similar to using delegates, but it has these advantages:

- Any number of objects may receive the notification, not just the delegate object. This precludes returning a value.
- An object may receive any message you like from the notification center, not just the predefined delegate methods.
- The object posting the notification does not even have to know the observer exists.

Creating Subclasses

You can subclass NSNotification to contain information in addition to the notification name, object, and dictionary. This extra data must be agreed upon between notifiers and observers.

Interfaces Implemented

NSCoding

`classForCoder`

`encodeWithCoder`

Method Types

Constructors

`NSNotification`

Obtaining information about a notification

`name`

`object`

`userInfo`

Methods inherited from Object

`equals`

`hashCode`

`toString`

Decoding the notification

`decodeObject`

Constructors

NSNotification

```
public NSNotification(
    String aName,
    Object anObject)
```

Creates a notification object that associates the name `aName` with the object `anObject` and contains an empty userInfo dictionary. The `aName` parameter may not be `null`.

```
public NSNotification(
    String aName,
    Object anObject,
    NSDictionary userInfo)
```

Returns a notification object that associates the name `aName` with the object `anObject` and the dictionary of arbitrary data `userInfo`. The dictionary `userInfo` may be `null`; if so, the new notification contains an empty userInfo dictionary. `aName` may not be `null`.

Static Methods

decodeObject

```
public static Object decodeObject(NSCoder coder)
```

Creates an NSNotification from the data in `coder`.

See Also: NSCoding

Instance Methods

classForCoder

```
public Class classForCoder()
```

Conformance with NSCoding. See the method description of `classForCoder` in the interface specification for NSCoding.

encodeWithCoder

```
public void encodeWithCoder(NSCoder aNSCoder)
```

Conformance with NSCoding. See the method description of `encodeWithCoder` in the interface specification for NSCoding.

equals

```
public boolean equals(Object anObject)
```

Compares the receiving NSNotification object to `anObject`. If `anObject` is an NSNotification and the contents of `anObject` are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`. Two notifications are equal if their names, objects, and dictionaries are equal.

See Also: `name`, `object`, `userInfo`

hashCode

```
public int hashCode()
```

Provide an appropriate hash code useful for storing the receiver in a hash-based data structure.

name

```
public String name()
```

Returns the name of the notification. Examples of this might be “PortIsInvalid”. Typically, you invoke this method on the notification object passed to your notification-handler method. (You specify a notification-handler method when you register to receive the notification.)

Notification names can be any string. To avoid name collisions, however, you might want to use a prefix that’s specific to your application.

object

```
public Object object()
```

Returns the object associated with the notification. This is often the object that posted this notification. It may be `null`.

Typically, you invoke this method on the notification object passed in to your notification-handler method. (You specify a notification-handler method when you register to receive the notification.)

toString

```
public String toString()
```

Returns a string representation of the receiver including its name, object, and dictionary.

userInfo

```
public NSDictionary userInfo()
```

Returns the NSDictionary associated with the notification. The NSDictionary stores any additional objects that objects receiving the notification might use. For example a PortIsInvalid notification may provide the port number in the dictionary. The NSDictionary is empty if no userInfo dictionary was specified when the notification was created.

NSNotificationCenter

Inherits from: Object

Package: com.webobjects.foundation

Class Description

An NSNotificationCenter object (or simply, notification center) is essentially a notification dispatch table. It notifies all observers of notifications meeting specific criteria. This information is encapsulated in NSNotification objects, also known as notifications. Client objects register themselves as observers of specific notifications posted by other objects. When an event occurs, an object posts an appropriate notification to the notification center. (See the NSNotification class specification for more on notifications.) The notification center dispatches a message to each registered observer, passing the notification as the sole argument. It is possible for the posting object and the observing object to be the same.

Each task has a default notification center that you access with the `defaultCenter` static method.

NSNotificationCenter is implemented using weak references (see Sun's documentation for java.lang.ref.* for details). Thus, if the default NSNotificationCenter is the last object in your application with a reference to either an object registered to receive notifications or an object being observed, the object will be garbage collected.

Registering to Receive Notifications

There are two ways to register to receive notifications.

If an object wishes to register itself to receive all notifications from all objects, it should send the `addOmniscientObserver` method, specifying the message the notification should send.

Otherwise, an object registers itself to receive a notification by sending the `addObserver` method, specifying the message the notification should send, the name of the notification it wants to receive, and about which object. However, the observer need not specify both the name and the object. If it specifies only the object, it will receive all notifications containing that object. If the object specifies only a notification name, it will receive that notification every time it's posted, regardless of the object associated with it.

It is possible for an observer to register to receive more than one message for the same notification. In such a case, the observer will receive all messages it is registered to receive for the notification, but the order in which it receives them cannot be determined.

Method Types

Constructors

`NSNotificationCenter`

Accessing the default center

`defaultCenter`

Adding and removing observers

`addObserver`

`addOmniscientObserver`

`removeObserver`

`removeOmniscientObserver`

Posting notifications

`postNotification`

Constructors

NSNotificationCenter

```
protected NSNotificationCenter()
```

Standard no-arg constructor. For use by subclasses only.

Static Methods

defaultCenter

```
public static NSNotificationCenter defaultCenter()
```

Returns the current task's notification center, which is used for system notifications.

Instance Methods

addObserver

```
public synchronized void addObserver(
    Object anObserver,
    NSSelector aSelector,
    String notificationName,
    Object anObject)
```

Registers anObserver to receive notifications with the name notificationName and / or containing anObject. When a notification of name notificationName containing the object anObject is posted, anObserver receives an aSelector message with this notification as the argument. The method for the selector specified in aSelector must have one and only one argument. anObject or notificationName can be null, but not both:

anObject	notificationName	Action
null	notificationName	The notification center notifies the observer of all notifications with the name notificationName.
anObject	null	The notification center notifies the observer of all notifications with an object matching anObject.
null	null	Do not invoke this method specifying null for both notificationName and anObject. Instead, use addOmniscientObserver.

addOmniscientObserver

```
public synchronized void addOmniscientObserver(
    Object anObserver,
    NSSelector aSelector)
```

Registers anObserver to receive all notifications from all objects. When a notification is posted, anObserver receives an aSelector message with this notification as the argument. The method for the selector specified in aSelector must have exactly one argument.

Omniscient observers can significantly degrade performance and should be used with care.

postNotification

```
public void postNotification(
    String notificationName,
    Object anObject,
    NSDictionary userInfo)
```

Creates a notification with the name `notificationName`, associates it with the object `anObject` and dictionary `userInfo`, and posts it to the notification center.

This method is the preferred method for posting notifications. `anObject` is typically the object posting the notification. It may be `null`. `userInfo` also may be `null`.

```
public void postNotification(
    String notificationName,
    Object anObject)
```

Creates a notification with the name `notificationName`, associates it with the object `anObject`, and posts it to the notification center. `anObject` is typically the object posting the notification. It may be `null`.

```
public void postNotification(NSNotification notification)
```

Posts `notification` to the notification center. You can create `notification` with the `NSNotification` constructor.

removeObserver

```
public void removeObserver(Object anObserver)
```

Same as `removeObserver(anObserver, null, null)`.

```
public synchronized void removeObserver(
    Object anObserver,
    String notificationName,
    Object anObject)
```

Removes anObserver as the observer of notifications with the name notificationName and object anObject from the notification center. This method interprets null parameters as wildcards:

removeObserver Parameters	Action
null, notificationName, anObject	Removes all observers of notificationName containing anObject.
anObserver, null, anObject	Removes anObserver as an observer of all notifications containing anObject.
anObserver, notificationName, null	Removes anObserver as an observer of notificationName containing any object.
anObserver, null, null	Removes all notifications containing anObserver.
null, notificationName, null	Removes all observers of notificationName.
null, null, anObject	Removes all observers of anObject.

Recall that the object a notification contains is usually the object that posted the notification.

removeOmniscientObserver

```
public synchronized void removeOmniscientObserver(Object anObserver)
```

Unregisters anObserver as an observer of all notifications.

toString

```
public String toString()
```

Returns a String representation of the receiver.

NSNumberFormatter

Inherits from: java.text.Format : Object

Package: com.webobjects.foundation

Class Description

Instances of NSNumberFormatter convert between BigDecimal numbers and textual representations of numeric values. The representation encompasses integers and floating-point numbers; floating-point numbers can be formatted to a specified decimal position. NSNumberFormatters can also impose ranges on the numeric values that can be formatted.

You can associate a number pattern with a WOString or WOTextField dynamic element. WebObjects uses an NSNumberFormatter object to perform the appropriate conversions.

Instances of NSNumberFormatter are mutable.

Creating an Instance of NSNumberFormatter

The most common technique for creating a NSNumberFormatter is to use the one-argument constructor, which takes as its argument a string whose contents can be one of the following:

- "positivePattern"

  For example, "$#,##0.00" (the syntax of format strings is discussed in the following section).
- "positivePattern;negativePattern"

For example, "#,##0.00;(#,##0.00)".

- "positivePattern;zeroPattern;negativePattern"

  For example, "$#,##0.00;0.00;($#,##0.00)". Note that zero patterns are treated as string constants.

As implied in the above list, you're only required to specify a pattern for positive values. If you don't specify a pattern for negative and zero values, a default pattern based on the positive value pattern is used. For example, if your positive value pattern is "#,##0.00", an input value of "0" will be displayed as "0.00".

If you don't specify a pattern for negative values, the pattern specified for positive values is used, preceded by a minus sign (-).

If you specify a separate pattern for negative values, its separators should be parallel to those specified in the positive pattern string. In NSNumberFormatter, separators are either enabled or disabled for all patterns—both your negative and positive patterns should therefore use the same approach.

As an alternative to using the one-argument constructor is to use the no-argument constructor and invoking `setPattern` with the pattern. You can also use the `setPositivePattern` and `setNegativePattern` methods.

Pattern String Syntax

Pattern strings can include the following types of characters:

- Numbers

  Pattern strings can include numeric characters. Wherever you include a number in a pattern string, the number is displayed unless an input character in the same relative position overwrites it. For example, suppose you have the positive pattern string "9,990.00", and the value 53.88 is entered into a cell to which the pattern has been applied. The cell would display the value as 9,953.88.

- Separators

  Pattern strings can include the period character (.) as a decimal separator, and comma character (,) as a thousand separator. If you want to use different characters as separators, you can set them using the `setDecimalSeparator` and `setThousandSeparator` methods.

- Placeholders

You use the pound sign character (#) to represent numeric characters that will be input by the user. For example, suppose you have the positive pattern "$#,##0.00". If the characters 76329 were entered into a cell to which the pattern has been applied, they would be displayed as $76,329.00. Strictly speaking, however, you don't need to use placeholders. The format strings ",0.00", "#,#0.00", and "#,##0.00" are functionally equivalent. In other words, including separator characters in a pattern string signals NSNumberFormatter to use the separators, regardless of whether you use (or where you put) placeholders. The placeholder character's chief virtue lies in its ability to make pattern strings more human-readable, which is especially useful if you're displaying patterns in the user interface.

- Spaces

  To include a space in a pattern string, use the underscore character (_). This character inserts a space if no numeric character has been input to occupy that position.

- Currency

  The dollar sign character ($) is normally treated just like any other character that doesn't play a special role in NSNumberFormatter.

All other characters specified in a pattern string are displayed as typed. The following table shows examples of the how the value 1019.55 is displayed for different positive patterns:

Pattern String	Display
"#,##0.00"	1,019.55
"$#,##0.00"	$1,019.55
"___,__0.00"	1,019.55

Using Separators

NSNumberFormatter supports two different kinds of separators: thousand and decimal. By default these separators are represented by the comma (,) and period (.) characters respectively. The default pattern ("#,##0.##") enables them.

All of the following statements have the effect of enabling thousand separators:

```
// use setPattern:
numberFormatter.setPattern("#,###");

// use setHasThousandSeparators:
numberFormatter.setHasThousandSeparators(true);
```

```
// use setThousandSeparator:
numberFormatter.setThousandSeparator("_");
```

If you use the statement `numberFormatter.setHasThousandSeparators(false)`, it disables thousand separators, even if you've set them through another means.

Both of the following statements have the effect of enabling decimal separators:

```
// use setFormat:
numberFormatter.setFormat("0.00");

// use setDecimalSeparator:
numberFormatter.setDecimalSeparator("-");
```

When you enable or disable separators, it affects both positive and negative patterns. Consequently, both patterns must use the same separator scheme.

You can use the `thousandSeparator` and `decimalSeparator` methods to return a string containing the character the receiver uses to represent each separator. However, this shouldn't be taken as an indication of whether separators are enabled—even when separators are disabled, an NSNumberFormatter still knows the characters it uses to represent separators.

Separators must be single characters. If you specify multiple characters in the arguments to `setThousandSeparator` and `setDecimalSeparator`, only the first character is used.

You can't use the same character to represent thousand and decimal separators.

Localization

NSNumberFormatter provides methods to localize pattern strings. You can change the currency symbol, the decimal separator, and the thousands separator manually, or you can trust NSNumberFormatter to do it for you, based on locales. If you enable localization for an instance of NSNumberFormatter, it will check the current locale and localize pattern strings appropriately for that locale. By default, instances of NSNumberFormatter are not localized. You can enable localization for all new instances of NSNumberFormatter using `setDefaultLocalizesPattern` or for a specific instance of NSNumberFormatter using `setLocalizesPattern`. See the method descriptions for `setDefaultLocalizesPattern` and `setLocalizesPattern` for more information.

Constants

NSNumberFormatter provides the following constants for specifying rounding modes and special numbers.The rounding mode specifiers are used with `roundingBehavior` and `setRoundingBehavior`.

Constant	Type	Description
`RoundDown`	`int`	Rounding mode specifier: round towards negative infinity
`RoundUp`	`int`	Rounding mode specifier: round towards positive infinity
`RoundPlain`	`int`	Rounding mode specifier: round to nearest up
`RoundBankers`	`int`	Rounding mode specifier: round to nearest even
`NSDecimalNotANumber`	`java.math.BigDecimal`	Definition of Not A Number (NaN)
`DefaultPattern`	`String`	Default format for pattern strings: "#,##0.##”.

Method Types

Constructors

`NSNumberFormatter`

Performing formatted conversions

`objectValueForString`

`stringForObjectValue`

Methods inherited from java.text.Format

`format`

parseObject

Accessing patterns

negativePattern

pattern

positivePattern

setNegativePattern

setPattern

setPositivePattern

Accessing floating point settings

allowsFloats

roundingBehavior

setAllowsFloats

setRoundingBehavior

Accessing decimal separator settings

decimalSeparator

setDecimalSeparator

Accessing thousand separator settings

hasThousandSeparators

setHasThousandSeparators

setThousandSeparator

thousandSeparator

Accessing strings for special numbers

setStringForNotANumber

setStringForNull

setStringForZero

stringForNotANumber

stringForNull

stringForZero

Limiting the input number

maximum

minimum

setMaximum

setMinimum

Localization Methods

availableLocales

currencySymbol

defaultLocale

defaultLocalizesPattern

locale

localizesPattern

setCurrencySymbol

setDefaultLocale

setDefaultLocalizesPattern

setLocale

setLocalizesPattern

Deprecated Methods

attributedStringForNil

attributedStringForNotANumber

attributedStringForZero

format

localizesFormat

negativeFormat

positiveFormat

setAttributedStringForNil

setAttributedStringForNotANumber

setAttributedStringForZero

setFormat

setLocalizesFormat

setNegativeFormat

setPositiveFormat

Constructors

NSNumberFormatter

public NSNumberFormatter()

Creates an NSNumberFormatter and sets its pattern to the default pattern: "#,##0.##".

public NSNumberFormatter(String pattern)

Creates an NSFormatter and sets its pattern to pattern. If pattern is illegal, the constructor throws an IllegalArgumentException. See "Pattern String Syntax" (page 208) for an explanation of what makes a pattern legal.

See Also: setPattern

Static Methods

availableLocales

public static java.util.Locale[] availableLocales()

Returns a list of all installed locales.

defaultLocale

```
public static java.util.Locale defaultLocale()
```

Returns the default locale for all instances of NSNumberFormatter.

defaultLocalizesPattern

```
public static boolean defaultLocalizesPattern()
```

Returns `true` to indicate that the receiver's format will be localized for all new instances of NSNumberFormatter in your application. By default, patterns are not localized.

See Also: `setDefaultLocalizesPattern`

setDefaultLocale

```
public static void setDefaultLocale(Locale newLocale)
```

Sets according to `newLocale` the default locale of the receiver. Throws an IllegalArgumentException if `newLocale` is null.

setDefaultLocalizesPattern

```
public static void setDefaultLocalizesPattern(boolean newDefault)
```

Sets according to `newDefault` whether all new NSNumberFormatter objects in your application created after this method is invoked are set to be localized by NSNumberFormatter based on the locale. NSNumberFormatter will choose the appropriate currency symbol, decimal separator, thousands separator, string for zero, and string for not a number based on locale if `newDefault` is `true`. By default, NSNumberFormatters are not localized.

See Also: `defaultLocalizesPattern`

Instance Methods

allowsFloats

```
public boolean allowsFloats()
```

Returns `true` if the receiver allows as input floating point values (that is, values that include the period character (.)), `false` otherwise. When this is set to `false`, only integer values can be provided as input. The default is `true`.

attributedStringForNil

```
public String attributedStringForNil()
```

Deprecated in the Java Foundation framework. Don't use this method. Use `stringForNull` instead.

attributedStringForNotANumber

```
public String attributedStringForNotANumber()
```

Deprecated in the Java Foundation framework. Don't use this method. Use `stringForNotANumber` instead.

attributedStringForZero

```
public String attributedStringForZero()
```

Deprecated in the Java Foundation framework. Don't use this method. Use `stringForZero` instead.

currencySymbol

```
public String currencySymbol()
```

Returns a string representing the symbol the receiver uses to represent currency.

See Also: `setCurrencySymbol`

decimalSeparator

```
public String decimalSeparator()
```

Returns a string containing the character the receiver uses to represent decimal separators. By default this is the period character (.).

format

```
public StringBuffer format(
    Object object,
    StringBuffer toAppendTo,
    java.text.FieldPosition position)
```

Formats `object` to produce a string, appends the string to `toAppendTo`, and returns the resulting StringBuffer. The `position` parameter specifies an alignment field to place the formatted object. When the method returns, this parameter contains the position of the alignment field. See Sun's java.text.Format documentation for more information.

See Also: `stringForObjectValue`

format

```
public String format()
```

Deprecated in the Java Foundation framework. Don't use this method. Use `pattern` instead.

hasThousandSeparators

```
public boolean hasThousandSeparators()
```

Returns `true` to indicate that the receiver's format includes thousand separators, `false` otherwise. The default is `false`.

locale

```
public java.util.Locale locale()
```

Returns the current locale.

localizesFormat

```
public boolean localizesFormat()
```

Deprecated in the Java Foundation framework. Don't use this method. Use `localizesPattern` instead.

localizesPattern

```
public boolean localizesPattern()
```

Returns `true` to indicate that the receiver's format will be localized for a specific instance of NSNumberFormatter. By default, instances of NSNumberFormatter are not localized.

maximum

```
public java.math.BigDecimal maximum()
```

Returns the highest number allowed as input by the receiver.

minimum

```
public java.math.BigDecimal minimum()
```

Returns the lowest number allowed as input by the receiver.

negativeFormat

```
public String negativeFormat()
```

Deprecated in the Java Foundation framework. Don't use this method. Use `negativePattern` instead.

negativePattern

```
public String negativePattern()
```

Returns a string containing the pattern the receiver uses to display negative numbers.

See Also: positivePattern, setPattern

objectValueForString

```
public Object objectValueForString(String inString)
    throws java.text.ParseException
```

Returns a number (a java.math.BigDecimal object) by parsing inString according to the receiver's pattern. Throws a ParseException if the conversion fails for any reason.

See Also: setPattern

parseObject

```
public Object parseObject(
    String source,
    java.text.ParsePosition status)

public Object parseObject(String source) throws java.text.ParseException
```

Parses a string to produce an object. See Sun's java.text.Format documentation for more information.

See Also: objectValueForString

pattern

```
public String pattern()
```

Returns a string containing the pattern the receiver uses to format numbers.

See Also: setPattern

positiveFormat

```
public String positiveFormat()
```

Deprecated in the Java Foundation framework. Don't use this method. Use `positivePattern` instead.

positivePattern

```
public String positivePattern()
```

Returns a string containing the pattern the receiver uses to format positive numbers.

See Also: `setPattern`, `setNegativePattern`

roundingBehavior

```
public int roundingBehavior()
```

Returns an integer indicating the rounding behavior used by the receiver.

setAllowsFloats

```
public void setAllowsFloats(boolean flag)
```

Sets according to `flag` whether the receiver allows as input floating point values (that is, values that include the period character (.)). By default, floating point values are allowed as input.

setAttributedStringForNil

```
public void setAttributedStringForNil(String string)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `setStringForNull` instead.

setAttributedStringForNotANumber

```
public void setAttributedStringForNotANumber(String string)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `setStringForNotANumber` instead.

setAttributedStringForZero

```
public void setAttributedStringForZero(String string)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `setStringForZero` instead.

setCurrencySymbol

```
public void setCurrencySymbol(String newSymbol)
```

Sets the string the receiver uses to represent currency.

See Also: `currencySymbol`

setDecimalSeparator

```
public void setDecimalSeparator(String aString)
```

Sets the character the receiver uses as a decimal separator to `newSeparator`. If `newSeparator` contains multiple characters, only the first one is used. Throws an IllegalArgumentException if `aString` is null or has length other than one character.

setFormat

```
public void setFormat(String format)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `setPattern` instead.

setHasThousandSeparators

```
public void setHasThousandSeparators(boolean flag)
```

Sets according to `flag` whether the receiver uses thousand separators. When `flag` is `false`, thousand separators are disabled for both positive and negative formats (even if you've set them through another means, such as `setPattern`). When `flag` is `true`, thousand separators are used. In addition to using this method to add thousand separators to your format, you can also use it to disable thousand separators if you've set them using another method. The default is `false` (though you in effect change this setting to `true` when you set thousand separators through any means, such as `setPattern`)

setLocale

```
public void setLocale(java.util.Locale newLocale)
```

Sets according to `newLocale` the current locale of the receiver. Throws an IllegalArgumentException if `newLocale` is null.

setLocalizesFormat

```
public void setLocalizesFormat(boolean newDefault)
```

Deprecated for the Foundation framework. Don't use this method. Use `setLocalizesPattern`instead.

setLocalizesPattern

```
public void setLocalizesPattern(boolean newDefault)
```

Sets according to `newDefault` whether the receiver's pattern is set to be localized by NSNumberFormatter based on the locale. NSNumberFormatter will choose the appropriate currency symbol, decimal separator, thousands separator, string for zero, and string for not a number based on locale if `newDefault` is `true`. By default, NSNumberFormatters are not localized.

setMaximum

```
public void setMaximum(java.math.BigDecimal aMaximum)
```

Sets the highest number the receiver allows as input to aMaximum. Throws an IllegalArgumentException if aMaximum is not of type BigDecimal.

setMinimum

```
public void setMinimum(java.math.BigDecimal aMinimum)
```

Sets the lowest number the receiver allows as input to aMinimum. Throws an IllegalArgumentException if aMinimum is not of type BigDecimal.

setNegativeFormat

```
public void setNegativeFormat(String format)
```

Deprecated in the Java Foundation framework. Don't use this method. Use setNegativePattern instead.

setNegativePattern

```
public void setNegativePattern(String aString)
```

Sets the pattern the receiver uses to display positive numbers to pattern. If pattern is illegal, the method throws an IllegalArgumentException. See "Pattern String Syntax" (page 208) for an explanation of what makes a pattern legal. Invokes the private method validatePattern() which throws an IllegalArgumentException if the pattern is null, the pattern string is empty, or the string does not contain one of the characters in ",._#0123456789".

See Also: setPattern

setPattern

```
public void setPattern(String aPattern)
```

Sets the receiver's format to the string aPattern. This pattern can consist of one, two, or three parts separated by ";". The first part of the string represents the positive pattern, the second part of the string represents the zero value, and the last part of the string represents the negative pattern. If

the string just has two parts, the first one becomes the positive pattern, and the second one becomes the negative pattern. If the string just has one part, it becomes the positive pattern, and default formats are provided for zero and negative values based on the positive format. For more discussion of this subject, see the section "Creating an Instance of NSNumberFormatter" (page 207) in the Class Description.

If the positive, negative, or zero pattern is illegal, the method throws an IllegalArgumentException. See "Pattern String Syntax" (page 208) for an explanation of what makes a pattern legal.

The following code excerpt shows the three different approaches for setting an NSNumberFormatter object's format using `setPattern`:

```
NSNumberFormatter numberFormatter = new NSNumberFormatter();

// specify just positive format
numberFormatter.setPattern("$#,##0.00");

// specify positive and negative formats
numberFormatter.setPattern("$#,##0.00;($#,##0.00)");

// specify positive, zero, and negative formats
numberFormatter.setFormat("$#,###.00;0.00;($#,##0.00)");
```

See Also: `pattern`

setPositiveFormat

```
public void setPositiveFormat(String format)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `setPositivePattern` instead.

setPositivePattern

```
public void setPositivePattern(String pattern)
```

Sets the pattern the receiver uses to display positive numbers to pattern. If pattern is illegal, the method throws an IllegalArgumentException. See "Pattern String Syntax" (page 208) for an explanation of what makes a pattern legal. Invokes the private method validatePattern() which throws an IllegalArgumentException if the pattern is null, the pattern string is empty, or the string does not contain one of the characters in ",._#0123456789".

See Also: setPattern

setRoundingBehavior

```
public void setRoundingBehavior(int newRoundingBehavior)
```

Sets the receiver's rounding behavior to newRoundingBehavior. Throws an IllegalArgumentException if newRoundingBehavior is not one of the standard rounding modes: NSRoundDown, NSRoundUp, NSRoundPlain, NSRoundBankers. Consult the Foundation functions and constants documentation for complete information on rounding modes.

setStringForNotANumber

```
public void setStringForNotANumber(String newString)
```

Sets the string the receiver uses to display "not a number" to newString. Throws an IllegalArgumentException if newString is null.

setStringForNull

```
public void setStringForNull(String newString)
```

Sets the string the receiver uses to display null values to newString. Throws an IllegalArgumentException if newString is null.

setStringForZero

```
public void setStringForZero(String aString)
```

Sets the string the receiver uses to display zero values to newString.

setThousandSeparator

```
public void setThousandSeparator(String newSeparator)
```

Sets the character the receiver uses as a thousand separator to newSeparator. If newSeparator contains multiple characters, only the first one is used. If you don't have thousand separators enabled through any other means (such as setPattern), using this method enables them. Throws an IllegalArgumentException if newSeparator is null or has length other than one character.

stringForNotANumber

```
public String stringForNotANumber()
```

Returns the string the receiver uses to display "not a number" values. By default "not a number" values are displayed as the string "NaN".

stringForNull

```
public String stringForNull()
```

Returns the string the receiver uses to display null values. By default, null values are displayed as an empty string.

stringForObjectValue

```
public String stringForObjectValue(Object object) throws IllegalArgumentException
```

Returns a string representing object formatted according to the receiver's pattern. Throws an IllegalArgumentException if object is not an instance of Number.

See Also: setPattern

stringForZero

```
public String stringForZero()
```

Returns the string the receiver uses to display zero values. By default zero values are displayed according to the format specified for positive values; for more discussion of this subject see the section "Creating an Instance of NSNumberFormatter" (page 207) in the Class Description.

thousandSeparator

```
public String thousandSeparator()
```

Returns a string containing the character the receiver uses to represent thousand separators. By default this is the comma character (,). Note that the return value doesn't indicate whether thousand separators are enabled.

NSPathUtilities

Inherits from: Object

Package: com.webobjects.foundation

Class Description

This class provides static methods that are useful when working with paths. Specifically, it includes methods that extract particular path components (`lastPathComponent` and `pathExtension`), modify paths (`stringByAppendingPathComponent`, `stringByAppendingPathExtension`, `stringByDeletingLastPathComponent`, `stringByDeletingPathExtension`, and `stringByStandardizingPath`), and return special paths (`homeDirectory`).

The NSPathUtilities class cannot be instantiated.

Method Types

Extracting path components

`lastPathComponent`

`pathExtension`

Manipulating paths

stringByAppendingPathComponent

stringByAppendingPathExtension

stringByDeletingLastPathComponent

stringByDeletingPathExtension

stringByNormalizingExistingPath

stringByStandardizingPath

Resolving special paths

homeDirectory

Deprecated methods

URLWithPath

fileExistsAtPath

pathIsAbsolute

pathIsEqualToString

Static Methods

fileExistsAtPath

```
public static boolean fileExistsAtPath(String aString)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `(new File(aString)).exists()` instead.

homeDirectory

```
public static String homeDirectory()
```

Returns a string containing the home directory path for the user who executes the application.

lastPathComponent

```
public static String lastPathComponent(String aString)
```

Returns the last path component of aString. The following table illustrates the effect of lastPathComponent on a variety of different paths:

aString's Value	String Returned
"/tmp/scratch.tiff"	"scratch.tiff"
"/tmp/scratch"	"scratch"
"/tmp/"	"tmp"
"scratch"	"scratch"
"/"	"" (an empty string)

pathExtension

```
public static String pathExtension(String aString)
```

Interprets aString as a path, returning the aString's extension, if any (not including the extension divider). The following table illustrates the effect of pathExtension on a variety of different paths:

aString's Value	String Returned
"/tmp/scratch.tiff"	"tiff"
"/tmp/scratch"	"" (an empty string)
"/tmp/"	"" (an empty string)
"/tmp/scratch..tiff"	"tiff"

pathIsAbsolute

```
public static boolean pathIsAbsolute(String aString)
```

Deprecated in the Java Foundation framework. Don't use this method. Use (new File(aString)).isAbsolute() instead.

pathIsEqualToString

```
public static boolean pathIsEqualToString(
    String string1,
    String string2)
```

Deprecated in the Java Foundation framework. Don't use this method. You should never have to invoke it.

stringByAppendingPathComponent

```
public static String stringByAppendingPathComponent(
    String string1,
    String string2)
```

Returns a string made by appending `string1` with `string2`, preceded by if necessary by a path separator. The following table illustrates the effect of this method on a variety of different paths, assuming that `string2` is supplied as "`scratch.tiff`":

`string1`'s Value	Resulting String
"`/tmp`"	"`/tmp/scratch.tiff`"
"`/tmp/`"	"`/tmp/scratch.tiff`"
"`/`"	"`/scratch.tiff`"
"" (an empty string)	"`scratch.tiff`"

See Also: `stringByAppendingPathExtension`, `stringByDeletingLastPathComponent`

stringByAppendingPathExtension

```
public static String stringByAppendingPathExtension(
    String string1,
    String string2)
```

Returns a string made by appending to string1 an extension separator followed by string2. Note that if string1 ends with one or more slashes ("/"), these slashes are deleted. The following table illustrates the effect of this method on a variety of different paths, assuming that string2 is supplied as "tiff":

string1's Value	Resulting String
"/tmp/scratch.old"	"/tmp/scratch.old.tiff"
"/tmp/scratch."	"/tmp/scratch..tiff"
"/tmp/"	"/tmp.tiff"
"scratch"	"scratch.tiff"

stringByDeletingLastPathComponent

```
public static String stringByDeletingLastPathComponent(String aString)
```

Returns a string made by deleting the last path component from aString, along with any final path separator. If aString represents the root path, however, it's returned unaltered. The following table illustrates the effect of this method on a variety of different paths:

aString's Value	Resulting String
"/tmp/scratch.tiff"	"/tmp"
"/tmp/lock/"	"/tmp"
"/tmp/"	"/"
"/tmp"	"/"
"/"	"/"
"scratch.tiff"	"" (an empty string)

See Also: stringByDeletingPathExtension, stringByAppendingPathComponent

stringByDeletingPathExtension

```
public static String stringByDeletingPathExtension(String aString)
```

Returns a string made by deleting the extension (if any, and only the last) from aString. Strips any trailing path separator before checking for an extension. If aString represents the root path, however, it's returned unaltered. The following table illustrates the effect of this method on a variety of different paths:

aString's Value	Resulting String
"/tmp/scratch.tiff"	"/tmp/scratch"
"/tmp/"	"/tmp"
"scratch.bundle/"	"scratch"
"scratch..tiff"	"scratch."
".tiff"	"" (an empty string)
"/"	"/"

See Also: pathExtension, stringByDeletingLastPathComponent

stringByNormalizingExistingPath

```
public static String stringByNormalizingExistingPath(String aString)
```

Returns a string containing the "normalized" path for an existing file with path aString. If the file does not exist, this method returns null. The normalized path is always absolute and corresponds to the canonical path returned by the java.io.File.getCanonicalPath method. See Sun's documentation for this method for more information.

See Also: stringByStandardizingPath

stringByStandardizingPath

```
public static String stringByStandardizingPath(String aString)
```

Returns a string made by resolving various elements of aString. If the path contains tilde (~), inserts the home directory path in its place. If the path contains the parent directory marker (..) removes the previous path component from the path. If the parent directory marker is at the

beginning of aString, throws an IllegalArgumentException. The following table illustrates the effect of this method on a variety of different paths assuming that the home directory is "/Local/Users/guest":

aString's Value	Resulting String
"~/scratch.tiff"	"/Local/Users/guest/scratch.tiff"
"~"	"/Local/Users/guest"
"~/.."	"/Local/Users/"
"guest/../john/scratch.tiff"	"john/scratch.tiff"
"../scratch.tiff"	throws IllegalArgumentException

URLWithPath

```
public static java.net.URL URLWithPath(String aString)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `new URL("file://" + aString)` instead.

NSProperties

Inherits from: Object

Package: com.webobjects.foundation

Class Description

The NSProperties class enhances Java's properties mechanism to merge application properties with the standard system properties available using the `java.lang.System.getProperties()` method. The application properties can come from three sources: the command line, the application's `Properties` file, and the `Properties` files of any frameworks your application includes.

This class has only static methods and cannot be instantiated.

Accessing the Properties

To access the application properties you first need to merge the application and command line properties with the system properties. A WebObjects application automatically performs this step for you. You can then access the property as a string, and convert the string to the property's actual data type.

To obtain the application properties and merge them with the system properties you invoke `setPropertiesFromArgv`. Application properties can come from the application's `Properties` file, the `Properties` files for the frameworks the application includes, and the command line. For more information about `Properties` files, see "The Properties File" (page 238). For more information about specifying properties on the command line, see "Command Line Properties" (page 238).

Every property is a key-value pair. For example, on Unix machines, the property value for the key “file.separator” is “/”. To access a property corresponding to a particular key, use the `java.lang.System.getProperty` method. This method returns the property as a string.

If the property string represents a boolean value, NSArray, or NSDictionary you need to convert it to the appropriate data type using the NSPropertyListSerialization `booleanForString`, `arrayForString`, or `dictionaryForString` method, respectively. NSPropertyListSerialization also provides an `intForString` method to simplify converting a property string to an integer.

The Properties File

The properties must be stored in a file named `Properties` in the application or framework’s `Resources` directory. You can add a `Properties` file to your application or framework by adding it to the Resources suitcase in Project Builder.

The Properties file must be in the format specified by java.io.Properties. See Sun’s documentation for the `load` method in that class for the format specification.

Boolean values, NSArrays, and NSDictionaries must be specified using the property list representation. See the NSPropertyListSerialization class description for more information on property lists.

Command Line Properties

The `setPropertiesFromArgv` method parses the command line arguments are recognizes the property formats listed in the table below.

Format	Example
`-D`*key=value*	`-DWOPort=4321`
`-`*key value*	`-WOAutoOpenInBrowser NO`

Properties specified in these formats will be available as system properties after you invoke `setPropertiesFromArgv`.

Static Methods

arrayForKey

```
public static NSArray arrayForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)` and convert it to an NSArray using NSPropertyListSerialization's `arrayForString` method.

Returns the system property with the specified name as an NSArray or `null` if no system property with that name exists. Throws a NullPointerException if `key` is `null`.

booleanForKey

```
public static boolean booleanForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)` and convert it to a `boolean` using NSPropertyListSerialization's `booleanForString` method.

Returns the system property with the specified name as a `boolean`. Returns `false` if no system property with that name exists. Throws a NullPointerException if `key` is `null`.

dataForKey

```
public static NSData dataForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)`, convert it to a property list using NSPropertyListSerialization's `propertyListFromString` method, and convert the property list to an NSData object using NSPropertyListSerialization's `dataFromPropertyList` method.

Interprets the system property with the specified name as a string representation of a property list, converts it to bytes using the current encoding, and stores the result in an NSData object. Returns `null` if no system property with that name exists. Throws a NullPointerException if `key` is `null`.

dictionaryForKey

```
public static NSDictionary dictionaryForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)` and convert it to an NSDictionary using NSPropertyListSerialization's `dictionaryForString` method.

Returns the system property with the specified name as an NSDictionary. Returns `null` if no system property with that name exists. Throws a NullPointerException if key is `null`.

doubleForKey

```
public static double doubleForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)` and convert it to a `double`.

Returns the system property with the specified name as a `double`. Returns 0 if no system property with that name exists. Throws a NullPointerException if key is `null`.

floatForKey

```
public static float floatForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)` and convert it to a `float`.

Returns the system property with the specified name as a `float`. Returns 0 if no system property with that name exists. Throws a NullPointerException if key is `null`.

integerForKey

```
public static int integerForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)` and convert it to an `int` using NSPropertyListSerialization's `intForString` method.

Returns the system property with the specified name as an `int`. Returns 0 if no system property with that name exists. Throws a NullPointerException if key is `null`.

longForKey

```
public static long longForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)` and convert it to a `long`.

Returns the system property with the specified name as a `long`. Returns 0 if no system property with that name exists. Throws a NullPointerException if `key` is `null`.

setPropertiesFromArgv

```
public static void setPropertiesFromArgv(String[] argv)
```

Loads all of the application's properties and merges them with the Java System properties. This method obtains the properties (by invoking NSBundle's `properties` method) for every bundle in the application including the application and all of the frameworks it includes. It also merges any properties specified by the string array into the system properties.

stringForKey

```
public static String stringForKey(String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Instead, get the system property using `System.getProperty(key)`.

Equivalent to `System.getProperty(key)`.

valuesFromArgv

```
public static NSDictionary valuesFromArgv(String[])
```

Description forthcoming.

NSPropertyListSerialization

Inherits from: Object

Package: com.webobjects.foundation

Class Description

This class provides static methods that convert between property lists and their string representations, which can be either strings or NSData objects. A property list is a structure that represents organized data. It can be built from a combination of NSArrays, NSDictionaries, Strings, and NSData objects.

The string representation can be in XML or the ASCII plist format. To distinguish between the two formats, the parser that converts strings to property lists checks if the string starts with `<?xml`. A discussion of the ASCII plist format, *A Primer on ASCII Property Lists*, is available in the Mac OS X section of the Apple Developer Connection website. A discussion of XML property lists, *Property List Services*, is also available in the same area of the Apple Developer Connection website.

Some methods do not support XML property list representations, specifically `booleanForString` and `intForString`.

The NSPropertyListSerialization class cannot be instantiated.

Method Types

Extracting typed data

arrayForString

booleanForString

dictionaryForString

intForString

Converting to property lists

propertyListFromData

propertyListFromString

Converting from property lists

dataFromPropertyList

stringFromPropertyList

Static Methods

arrayForString

```
public static NSArray arrayForString(String string)
```

Parses the property list representation `string` and returns the resulting property list as an NSArray. If the root object is not an array, this method throws a ClassCastException.

See Also: dictionaryForString.

booleanForString

```
public static boolean booleanForString(String string)
```

Returns a boolean based on string according to the following table:

string's value	Value returned
"YES"	true
"true"	true
Any other value	false

The tests for "YES" and "true" are case insensitive.

dataFromPropertyList

```
public static NSData dataFromPropertyList(
    Object object,
    String encoding)

public static NSData dataFromPropertyList(Object object)
```

Converts the property list object into a string and returns it as a NSData object. The encoding parameter specifies the encoding used to convert the characters in the result string to byte. The one-argument version of the method uses the platform's default character encoding.

See Also: stringFromPropertyList

dictionaryForString

```
public static NSDictionary dictionaryForString(String string)
```

Parses the property list representation string and returns the resulting property list as an NSDictionary. If the root object is not a dictionary, this method throws a ClassCastException.

See Also: arrayForString.

intForString

```
public static int intForString(String string)
```

Parses string and returns the corresponding integer. If string is null, returns zero.

propertyListFromData

```
public static Object propertyListFromData(
    NSData data,
    String encoding)

public static Object propertyListFromData(NSData data)
```

Returns a property list converted from the byte array representation in data. The encoding parameter specifies the encoding used to convert the bytes in the data byte array to characters in a string representation. The one-argument version of the method uses the platform's default character encoding.

See Also: propertyListFromString

propertyListFromString

```
public static Object propertyListFromString(String string)
```

Returns a property list converted from the string representation string.

stringFromPropertyList

```
public static String stringFromPropertyList(Object object)
```

Converts the property list object into a string and returns it.

NSRange

Inherits from: Object

Implements: Cloneable
Serializable

Package: com.webobjects.foundation

Class Description

An NSRange represents a range, a measurement of a segment of something linear, such as a byte stream. An NSRange has two primary values, a location and a length. The methods of NSRange give access to these values, convert between NSRanges and their string representations, test and compare ranges, and create ranges based on operations involving the union, intersection, and subtraction of two ranges.

Table 0-12 describes the NSRange methods that provide the basis for all NSRange's other methods; that is, all other methods are implemented in terms of these two. If you create a subclass of NSRange, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-12 NSRange's Base API

Method	Description
`length`	Returns the length of the receiver from its starting location.
`location`	Returns the starting location of the receiver.

Constants

NSRange provides the following constant as a convenience; you can use it to compare values returned by some NSRange methods:

Constant	Type	Description
ZeroRange	NSRange	An NSRange set to zero in location and length.

Interfaces Implemented

Cloneable

clone

Method Types

Constructors

NSRange

Accessing range elements

length

location

Manipulating ranges

rangeByIntersectingRange

rangeByUnioningRange

subtractRange

Testing ranges

containsLocation

intersectsRange

isEmpty

isEqualToRange

isSubrangeOfRange

locationInRange

maxRange

Methods inherited from Object

equals

hashCode

toString

Converting Strings to NSRanges

fromString

Constructors

NSRange

```
public NSRange()
```

Creates an NSRange with zero location and length. For better performance, use the ZeroRange shared instance. See Constants.

```
public NSRange(NSRange aRange)
```

Creates a new NSRange with the location and length values of aRange.

```
public NSRange(
    int location,
    int length)
```

Creates a new NSRange with the range elements of `location` and `length`. Throws an IllegalArgumentException if either integer is negative.

Static Methods

fromString

```
public static NSRange fromString(String rangeAsString)
```

Creates an NSRange from the string `rangeAsString`. The string must be of the form "{*loc*,*len*}" where *loc* is a number representing the starting location of the range and *len* is the range's length. Throws an IllegalArgumentException if the string is improperly formatted.

See Also: `toString`

Instance Methods

clone

```
public Object clone()
```

Simply returns the receiver. Since NSRange objects are immutable, there's no need to make an actual clone.

containsLocation

```
public boolean containsLocation(int aLocation)
```

Returns whether the location `aLocation` falls within the limits specified by the receiver.

See Also: `intersectsRange`, `location`

equals

```
public boolean equals(Object otherObject)
```

Returns whether otherObject is an NSRange and is equal in location and length to the receiver.

See Also: isEqualToRange, isSubrangeOfRange

hashCode

```
public int hashCode()
```

Provide an appropriate hash code useful for storing the receiver in a hash-based data structure.

intersectsRange

```
public boolean intersectsRange(NSRange aRange)
```

Returns whether the range aRange intersects the receiver.

See Also: rangeByIntersectingRange

isEmpty

```
public boolean isEmpty()
```

Returns whether the length of the receiver is zero.

See Also: maxRange

isEqualToRange

```
public boolean isEqualToRange(NSRange aRange)
```

Returns whether the range aRange is equal in both location and length to the receiver.

See Also: equals, isSubrangeOfRange

isSubrangeOfRange

```
public boolean isSubrangeOfRange(NSRange aRange)
```

Returns whether the receiver's end points match or fall within those of range aRange.

See Also: intersectsRange

length

```
public int length()
```

Returns the length of the receiver from its starting location.

See Also: location

location

```
public int location()
```

Returns the starting location of the receiver.

See Also: length

locationInRange

```
public boolean locationInRange(int aLocation)
```

This method is deprecated. Use containsLocation instead.

maxRange

```
public int maxRange()
```

Returns the extent of the receiver (its starting location plus its length). This number is one greater than the last location in the range.

See Also: isEmpty, length, location

rangeByIntersectingRange

```
public NSRange rangeByIntersectingRange(NSRange aRange)
```

Returns an NSRange that is the intersection of aRange and the receiver. If the ranges do not intersect, returns an empty range (see isEmpty).

See Also: rangeByUnioningRange, subtractRange

rangeByUnioningRange

```
public NSRange rangeByUnioningRange(NSRange aRange)
```

Returns an NSRange that is the union of aRange and the receiver (a range constructed from the lowest starting location and the highest ending location of both NSRanges).

See Also: rangeByIntersectingRange, subtractRange

subtractRange

```
public void subtractRange(
    NSRange otherRange,
    NSMutableRange resultRange1,
    NSMutableRange resultRange2)
```

Returns the ranges resulting from the subtraction of otherRange from the receiver by modifying the mutable ranges resultRange1 and resultRange2 (provided by the caller). Either or both of the the result ranges might be empty when this method returns.

See Also: rangeByIntersectingRange, rangeByUnioningRange

toString

```
public String toString()
```

Returns a string representing the receiver. The string is in the form "{*loc*,*len*}" where *loc* is the starting location of the range and *len* is its length.

See Also: fromString

NSRecursiveLock

Inherits from: Object

Implements: NSLocking

Package: com.webobjects.foundation

Class Description

NSRecursiveLock defines a lock that may be acquired multiple times by the same thread without causing a deadlock, a situation where a thread is permanently blocked waiting for itself to relinquish a lock. While the locking thread has one or more locks, all other threads are prevented from accessing the code protected by the lock. Here's an example where a recursive lock functions properly but other lock types would deadlock:

```
NSRecursiveLock theLock = new NSRecursiveLock();
...
theLock.lock();
/* lengthy operations involving global data */
theLock.lock();   /* possibly invoked in a subroutine */
...
theLock.unlock(); /* relinquishes most recent lock */
...
theLock.unlock(); /* relinquishes the first lock */
```

Unless theLock was an NSRecursiveLock, a deadlock condition would occur at the second `lock` message in the example above.

The NSRecursiveLock object keeps track of the recursion count: the number of lock requests that the owning thread has made and not unlocked. This is also the number of times unlock must be invoked to return the lock. To access the recursion count, use the `recursionCount` method.

The NSLock, NSMultiReaderLock, and NSRecursiveLock classes all adopt the NSLocking protocol and offer various additional features and performance characteristics. See the NSLock and NSMultiReaderLock class descriptions for more information.

Method Types

Constructors

- `NSRecursiveLock`

Instance methods

- `lock`
- `tryLock`
- `lockBeforeDate`
- `recursionCount`
- `toString`
- `unlock`

Constructors

NSRecursiveLock

```
public NSRecursiveLock()
```

Creates an NSRecursiveLock.

Instance Methods

lock

```
public void lock()
```

Conformance to NSLocking. See the method description of `lock` in the interface description for NSLocking. If the current thread already owns the lock, this method increments the recursion count.

lockBeforeDate

```
public boolean lockBeforeDate(NSTimestamp timestamp)
```

This method is deprecated. Use `tryLock(NSTimestamp timestamp)` instead.

recursionCount

```
public synchronized long recursionCount()
```

Returns the receiver's recursion count (the number of unlocks needed to return the lock) if the current thread owns the lock. If the current thread is not the owner of the lock, returns zero.

toString

```
public String toString()
```

Returns a string representation of the receiver that includes the thread that owns it and its recursion count.

tryLock

```
public boolean tryLock()
```

Attempts to acquire a lock. If the lock is not already taken by another thread, acquires the lock, sets the recursion count to 1 and returns `true`. If the current thread owns the lock, increments the recursion count and returns with a value of `true`. If the another thread owns the lock, returns `false` immediately.

```
public boolean tryLock(long msec)
```

Attempts to acquire a lock for `msec` milliseconds. If the current thread owns the lock, increments the recursion count and returns `true`. Otherwise, the thread is blocked until the receiver acquires the lock or `msec` milliseconds have passed. Returns `true` if the lock is acquired within this time limit. Returns `false` if the time limit expires before a lock can be acquired.

```
public boolean tryLock(NSTimestamp timestamp)
```

Attempts to acquire a lock until the time specified by `timestamp`. If the current thread owns the lock, increments the recursion count and returns `true`. Otherwise, the thread is blocked until the receiver acquires the lock or `timestamp` is reached. Returns `true` if the lock is acquired within this time limit. Returns `false` if the time limit expires before a lock can be acquired.

unlock

```
public synchronized void unlock()
```

```
public synchronized void unlock(long levels)
```

Decrements the recursion count by `levels` (decrements the recursion count by one in the no-argument version). If the resulting recursion level is zero, returns the lock. This method throws an Error if the thread that invokes it is not the lock's owner. Invoking this method when the lock count is zero does nothing.

NSSelector

Inherits from: Object

Implements: Serializable

Package: com.webobjects.foundation

Class Description

An NSSelector object (also called a selector) specifies a method signature, which is a method's name and parameter list. You can later apply a selector on any object, and it performs the method that matches the selector, if there is one.

To create a selector, use NSSelector's single constructor, which takes the method's name and an array of the parameter types. Note that to obtain a Class object for a type, append `.class` to the type's name. For example, the Class object for Object is `Object.class` and the Class object for boolean is `boolean.class`

This code sample creates a selector for the `doIt` method:

```
void doIt(String str, int i) { . . . }
NSSelector sel =
     new NSSelector("doIt", new Class[] {String.class, int.class} );
```

To apply a selector on an object, use the overloaded instance method `invoke`. It performs the method that matches the selector and returns the result. If the target object doesn't have a method matching the selector, it throws NoSuchMethodException. The most basic form of `invoke` takes the target object and an Object array of the arguments. Other forms are convenience

methods for selectors with no, one, or two arguments. Note that to pass an argument of a primitive type to `invoke`, use an object of the corresponding wrapper class. `invoke` converts the object back to the primitive type when it invokes the method. For example, to pass the float `f`, use `new Float(f)`; and to pass the boolean value `true`, use `new Boolean(true)`.

This code sample gives you two ways to apply the selector `sel` (defined above) to an object:

```
MyClass obj1 = new MyClass(), obj2 = new MyClass();
int i = 5;
sel.invoke(obj1, new Object[] { "hi", new Integer(i) });
sel.invoke(obj2, "bye", new Integer(10));
```

To create and apply a selector in one step, use the overloaded static method `invoke`. The basic form takes four arguments: the method name, an array of the parameter types, the target object, and an array of the arguments. Other forms are convenience methods for selectors with one or two arguments. This code sample shows two ways to create and apply a selector for the `doIt` method:

```
void doIt(String str, int i) { . . . }
MyClass obj1 = new MyClass(), obj2 = new MyClass();
int i = 5;

NSSelector.invoke("doIt", new Class[] {String.class, int.class},
     obj1, new Object[] {"hi", new Integer(i)});
NSSelector.invoke("doIt", String.class, int.class,
     obj1, "bye", new Integer(10));
```

Other methods return whether an object or class has a method that matches a selector (`implementedByObject` and `implementedByClass`) and returns the method name and parameter types for a selector (`name` and `parameterTypes`).

NSSelector is similar to java.lang.reflect.Method, which fully specifies a particular class's implementation of a method, and you can apply it only to objects of that class. NSSelector doesn't specify the method's class, so you can apply it to an object of any class. To find the java.lang.reflect.Method object for a method that matches a selector and that's in a particular object or class, use `methodOnObject` or `methodOnClass`.

Method Types

Constructors

NSSelector

Static methods

invoke

Invoking selectors

invoke

Testing selectors

implementedByClass

implementedByObject

Converting selectors to java.lang.reflect.Methods

methodOnClass

methodOnObject

Accessing selector elements

name

parameterTypes

Methods inherited from Object

equals

hashCode

toString

Constructors

NSSelector

```
public NSSelector(String methodName)
```

Creates a selector for the method that's named methodName and takes no parameters.

```
public NSSelector(
    String methodName,
    Class[] parameterTypes)
```

Creates a selector for the method that's named methodName and takes parameters parameterTypes. To create a selector for a method that takes no arguments, use null for parameterTypes. For an example, see the class description for this class.

Static Methods

invoke

```
public static Object invoke(
    String methodName,
    Class[] parameterTypes,
    Object target,
    Object[] arguments) throws IllegalAccessException,
    IllegalArgumentException,
    java.lang.reflect.InvocationTargetException,
    NoSuchMethodException
```

Creates and applies a selector that has any number of arguments. This method creates a selector with methodName and the parameter types in the array parameterTypes, applies that selector to target with the arguments in the array arguments, and returns the result. To apply a method that takes no arguments, use null for the arrays parameterTypes and arguments. As part of its implementation, this method uses the NSSelector constructor and the instance method invoke. For more information, see those method descriptions.

```
public static Object invoke(
    String methodName,
    Class parameterType,
    Object target,
    Object argument) throws IllegalAccessException,
    IllegalArgumentException,
    java.lang.reflect.InvocationTargetException,
    NoSuchMethodException
```

Creates and applies a selector that has one argument. This method creates a selector with methodName and parameterType, applies that selector to target with argument, and returns the result. As part of its implementation, this method uses the NSSelector constructor and the instance method invoke. For more information, see those method descriptions.

```
public static Object invoke(
    String methodName,
    Class parameterType1,
    Class parameterType2,
    Object target,
    Object argument1,
    Object argument2) throws IllegalAccessException,
    IllegalArgumentException,
    java.lang.reflect.InvocationTargetException,
    NoSuchMethodException
```

Creates and applies a selector that has two arguments. This method creates a selector with methodName and the parameter types parameterType1 and parameterType2, applies that selector to target with the arguments argument1 and argument2, and returns the result. As part of its implementation, this method uses the NSSelector constructor and the instance method invoke. For more information, see those method descriptions.

Instance Methods

equals

```
public boolean equals(Object anObject)
```

Compares the receiving NSSelector object to anObject. If anObject is an NSSelector and the contents of anObject are equal to the contents of the receiver, this method returns true. If not, it returns false. Two selectors are equal if their names and parameter types are equal.

hashCode

```
public int hashCode()
```

Provide an appropriate hash code useful for storing the receiver in a hash-based data structure.

implementedByClass

```
public boolean implementedByClass(Class targetClass)
```

Returns whether the class targetClass implements a method that matches the selector.

implementedByObject

```
public boolean implementedByObject(Object target)
```

Returns whether the object target implements a method that matches the selector. As part of its implementation, this method uses implementedByClass.

invoke

```
public Object invoke(
    Object target,
    Object[] arguments) throws IllegalAccessException,
    IllegalArgumentException,
    java.lang.reflect.InvocationTargetException,
    NoSuchMethodException
```

Invokes the method specified by the selector on target with arguments, and returns the result. If that method is void, it returns null. Note that the method may be a static or instance method.

invoke can't handle arguments or return values of primitive types (such as boolean, int, or float). If the method matching the selector returns a value of a primitive type, invoke returns the value in an object of the corresponding wrapper type (such as Boolean, Integer, or Float). To pass an argument of a primitive type to invoke, use an object of the corresponding wrapper class. invoke converts the object back to the primitive type when it invokes the method.

invoke throws an exception in the following cases:

- If target has no method that matches the selector, it throws NoSuchMethodException.
- If a method matches the selector but is inaccessible to target, it throws IllegalAccessException.

- If it can't convert an argument to the type specified in the selector, it throws IllegalArgumentException.
- If the invoked method throws an exception, it wraps that exception in a java.lang.reflect.InvocationTargetException and throws the new exception without completing.

As part of its implementation, this method uses `methodOnClass`.

For an example, see the class description for this class.

```
public Object invoke(
    Object target) throws IllegalAccessException,
    IllegalArgumentException,
    java.lang.reflect.InvocationTargetException,
    NoSuchMethodException
```

Invokes the method specified by the selector on `target` with no arguments, and returns the result. If that method is `void`, it returns `null`. Note that the method may be a static or instance method.

As part of its implementation, this method calls the `invoke` instance method that takes an array of arguments. For more information, see that method's description.

```
public Object invoke(
    Object target,
    Object argument) throws IllegalAccessException,
    IllegalArgumentException,
    java.lang.reflect.InvocationTargetException,
    NoSuchMethodException
```

Invokes the method specified by the selector on `target` with one argument (`argument`), and returns the result. If that method is `void`, it returns `null`. Note that the method may be a static or instance method.

As part of its implementation, this method calls the `invoke` instance method that takes an array of arguments. For more information, see that method's description.

```
public Object invoke(
    Object target,
    Object argument1,
    Object argument2) throws IllegalAccessException,
```

```
    IllegalArgumentException,
    java.lang.reflect.InvocationTargetException,
    NoSuchMethodException
```

Invokes the method specified by the selector on target with two arguments (argument1 and argument2), and returns the result. If that method is void, it returns null. Note that the method may be a static or instance method.

As part of its implementation, this method calls the invoke instance method that takes an array of arguments. For more information, see that method's description.

methodOnClass

```
public java.lang.reflect.Method methodOnClass(Class targetClass)
    throws NoSuchMethodException
```

Returns the method on the class targetClass that matches the selector. If targetClass has no method that matches the selector, this method throws NoSuchMethodException.

methodOnObject

```
public java.lang.reflect.Method methodOnObject(Object target)
    throws NoSuchMethodException
```

Returns the method on the object target that matches the selector. If target has no method that matches the selector, this method throws NoSuchMethodException.

name

```
public String name()
```

Returns the name of the method specified by the selector.

parameterTypes

```
public Class[] parameterTypes()
```

Copies and returns the array of parameter types specified by the selector.

toString

```
public String toString()
```

Returns a string representation of the receiver indicating its class and method name.

NSSet

Inherits from:	Object
Implements:	Cloneable java.io.Serializable NSCoding
Package:	com.webobjects.foundation

Class Description

The NSSet and NSMutableSet classes declare the programmatic interface to an object that manages a set of objects. NSSet provides support for the mathematical concept of a set. A set, both in its mathematical sense and in the implementation of NSSet, is an unordered collection of distinct elements. The NSMutableSet class is provided for sets whose contents may be altered.

NSSet declares the programmatic interface for static sets of objects. You establish a static set's entries when it's created, and thereafter the entries can't be modified. NSMutableSet, on the other hand, declares a programmatic interface for dynamic sets of objects. A dynamic—or mutable—set allows the addition and deletion of entries at any time, automatically allocating memory as needed.

Use sets as an alternative to arrays when the order of elements isn't important and performance in testing whether an object is contained in the set is a consideration—while arrays are ordered, testing for membership is slower than with sets. When testing for set membership, the `equals` method is invoked only once.

Methods that add entries to sets—whether during construction (for all sets) or modification (for mutable sets)—add each member to the set directly. This means that you must ensure that the members do not change. If you expect your members to change for any reason, you should make copies of them and add the copies to the set.

Table 0-13 describes the NSSet methods that provide the basis for all NSSet's other methods; that is, all other methods are implemented in terms of these three. If you create a subclass of NSSet, you need only ensure that these base methods work properly. Having done so, you can be sure that all your subclass's inherited methods operate properly.

Table 0-13 NSSet's Base API

Method	Description
`count`	Returns the number of members in the set.
`member`	Returns the object in the set that is equal to the specified object.
`objectsNoCopy`	Returns the actual array of objects in the set.

NSSet provides methods for querying the elements of the set. The `allObjects` method returns an array containing the objects in a set. The `anyObject` method returns some object in the set. Additionally, `intersectsSet` tests for set intersection, `isEqualToSet` tests for set equality, and `isSubsetOfSet` tests for one set being a subset of another.

The `objectEnumerator` method provides for traversing elements of the set one by one.

Constants

NSSet provides the following constant as a convenience; you can use it when you need an empty set.

Constant	Type	Description
`EmptySet`	NSSet	A shared NSSet instance containing no members.

Interfaces Implemented

Cloneable

clone

java.io.Serializable

NSCoding

classForCoder

decodeObject

encodeWithCoder

Method Types

Constructors

NSSet

Counting entries

count

Accessing the members

allObjects

anyObject

containsObject

member

objectEnumerator

objectsNoCopy

Comparing sets

intersectsSet

isEqualToSet

isSubsetOfSet

Joining sets

setByIntersectingSet

setBySubtractingSet

setByUnioningSet

Methods inherited from Object

equals

hashCode

toString

Copying sets

immutableClone

mutableClone

Constructors

NSSet

```
public NSSet()
```

Creates an empty NSSet. To improve performance, use the `EmptySet` shared instance. See Constants.

```
public NSSet(NSArray anArray)
```

Creates an NSSet containing the objects in anArray.

Note: NSSet assumes that the member objects are immutable. If your member objects are mutable, you should make copies of them and add the copies to the set.

```
public NSSet(NSSet aSet)
```

Creates an NSSet containing the objects in aSet.

```
public NSSet(Object object)
```

Creates an NSSet containing the single object object.

Note: NSSet assumes that member objects are immutable. If your member objects are mutable, you should make copies of them and add the copies to the set.

```
public NSSet(Object[] objects[])
```

Creates an NSSet containing the objects in the objects language array.

Note: NSSet assumes that member objects are immutable. If your member objects are mutable, you should make copies of them and add the copies to the set.

Static Methods

decodeObject

```
public static Object decodeObject(NSCoder coder)
```

Creates an NSSet from the data in coder.

See Also: NSCoding

Instance Methods

allObjects

```
public NSArray allObjects()
```

Returns an array containing the receiver's members, or an empty array if the receiver has no members. The order of the objects in the array isn't defined.

See Also: anyObject, objectEnumerator

anyObject

```
public Object anyObject()
```

Returns one of the objects in the set (essentially chosen at random), or `null` if the set contains no objects.

See Also: allObjects, objectEnumerator

classForCoder

```
public Class classForCoder()
```

Conformance with NSCoding. Please see the method description of `classForCoder` in the interface specification for NSCoding.

clone

```
public Object clone()
```

Returns a copy (a NSSet object) of the receiver. Since NSSets are immutable, there's no need to make an actual copy.

containsObject

```
public boolean containsObject(Object anObject)
```

Returns `true` if `anObject` is present in the set, `false` otherwise.

See Also: `member`

count

```
public int count()
```

Returns the number of members in the set.

encodeWithCoder

```
public void encodeWithCoder(NSCoder aNSCoder)
```

Conformance with NSCoding. Please see the method description of `encodeWithCoder` in the interface specification for NSCoding.

equals

```
public boolean equals(Object anObject)
```

Compares the receiving set to `anObject`. If `anObject` is an NSSet and the contents of `anObject` are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`.

hashCode

```
public int hashCode()
```

Provide an appropriate hash code useful for storing the receiver in a hash-based data structure. This value is the number of objects in the set.

immutableClone

```
public NSSet immutableClone()
```

Returns an immutable copy (an NSSet) of the receiver. Since the NSSets are immutable, there's no need to make an actual copy.

intersectsSet

```
public boolean intersectsSet(NSSet otherSet)
```

Returns `true` if at least one object in the receiver is also present in `otherSet`, `false` otherwise. The result of this method corresponds to the mathematical concept of disjoint sets: if the sets are not disjoint, `intersectsSet` returns `true`, otherwise it returns `false`.

See Also: `isEqualToSet`, `isSubsetOfSet`

isEqualToSet

```
public boolean isEqualToSet(NSSet otherSet)
```

Compares the receiving set to `otherSet`. If the contents of `otherSet` are equal to the contents of the receiver, this method returns `true`. If not, it returns `false`.

Two sets have equal contents if they each have the same number of members and if each member matches a member in the other set (as determined by `equals`).

See Also: `intersectsSet`, `isSubsetOfSet`

isSubsetOfSet

```
public boolean isSubsetOfSet(NSSet otherSet)
```

Returns `true` if every object in the receiver is also present in `otherSet`, `false` otherwise.

See Also: `intersectsSet`, `isEqualToSet`

member

```
public Object member(Object anObject)
```

If anObject is present in the set (as determined by equals), the object in the set is returned. Otherwise returns null.

mutableClone

```
public NSMutableSet mutableClone()
```

Returns a mutable set (an NSMutableSet) with the same members as the receiver.

objectEnumerator

```
public java.util.Enumeration objectEnumerator()
```

Returns an enumerator object that lets you access each object in the set

```
java.util.Enumeration enumerator = mySet.objectEnumerator();

while (enumerator.hasMoreElements()) {{
     Object anObject = enumerator.nextElement();
     /* code to act on each element */
}
```

When this method is used with mutable subclasses of NSSet, your code shouldn't modify the set during enumeration. If you intend to modify the set, use the allObjects method to create a "snapshot" of the set's members. Enumerate the snapshot, but make your modifications to the original set.

objectsNoCopy

```
protected Object[] objectsNoCopy()
```

Returns the actual array of objects contained in the set.

setByIntersectingSet

```
public NSSet setByIntersectingSet(NSSet otherSet)
```

Returns a set of objects that are in both the receiver and otherSet.

See Also: intersectsSet, isSubsetOfSet, isEqualToSet, setBySubtractingSet, setByUnioningSet

setBySubtractingSet

```
public NSSet setBySubtractingSet(NSSet otherSet)
```

Returns a set of objects that are in the receiver but not in otherSet.

See Also: intersectsSet, isSubsetOfSet, isEqualToSet, setByIntersectingSet, setByUnioningSet

setByUnioningSet

```
public NSSet setByUnioningSet(NSSet otherSet)
```

Returns a set of objects that are either in the receiver or in otherSet or both. If an object is in both, the resulting set contains it only once.

See Also: intersectsSet, isSubsetOfSet, isEqualToSet, setByIntersectingSet, setBySubtractingSet

toString

```
public String toString()
```

Returns a string representation of the receiver. The string has the form "(*object1*, *object2*, ...)".

NSSocketUtilities

Inherits from: Object

Package: com.webobjects.foundation

Class Description

This class provides an easy way to get a TCP socket with a connection timeout. The static methods in this class correspond to all of the java.net.Socket constructors. See Sun's documentation for the java.net.Socket class for more information.

Calling `getSocketWithTimeout` will either return a socket, or will throw an IOException if it times out (because a socket cannot be created). When a new socket is requested with `getSocketWithTimeout`, the polling interval regulates how often that socket is requested. By default, the polling interval is 100 milliseconds, and can be changed using `setPollingInterval`. A timeout argument is passed to `getSocketWithTimeout`, and the socket request times out when a timer that keeps track of the total polling time exceeds the timeout value.

This class only contains static methods. It is never instantiated.

Method Types

All methods

getSocketWithTimeout

pollingInterval

setPollingInterval

Static Methods

getSocketWithTimeout

```
public static java.net.Socket getSocketWithTimeout(
    String remoteHost,
    int remotePort,
    java.net.InetAddress localInetAddress,
    int localPort,
    int timeOut) throws java.net.UnknownHostException, java.io.IOException
```

Creates a socket and connects it to the port specified by remotePort at the host specified by remoteHost. Binds the socket to the local port specified by localPort at the local host specified by localInetAddress with a timeout specified by timeout. Throws an UnknownHostException if remoteHost cannot be resolved. Throws an IOException if the socket can't be created.

```
public static java.net.Socket getSocketWithTimeout(
    String remoteHost,
    int remotePort,
    int timeout) throws java.net.UnknownHostException, java.io.IOException
```

Creates a socket and connects it to the port specified by remotePort at the host specified by remoteHost with a timeout specified by timeout. Throws an UnknownHostException if remoteHost cannot be resolved. Throws an IOException if the socket can't be created.

```
public static java.net.Socket getSocketWithTimeout(
    InetAddress remoteAddress,
    int remotePort,
    int timeout) throws java.io.IOException
```

Creates a socket and connects it to the port specified by `remotePort` at the host specified by `remoteAddress` with a timeout specified by `timeout`. Throws an IOException if the socket can't be created.

```
public static Socket getSocketWithTimeout(
    InetAddress remoteAddress,
    int remotePort,
    InetAddress localAddress,
    int localPort,
    int timeout) throws java.io.IOException
```

Creates a socket and connects it to the port specified by `remotePort` at the host specified by `remoteAddress`. Binds the socket to the local port specified by `localPort` at the local host specified by `localAddress` with a timeout specified by `timeout`. Throws an IOException if the socket can't be created.

pollingInterval

```
public static int pollingInterval()
```

Returns the polling interval in milleseconds. The default polling interval is 100 milliseconds. It can be changed with `setPollingInterval`. See the "Class Description" section for more information.

setPollingInterval

```
public static void setPollingInterval(int interval)
```

Sets the polling interval to `interval` provided `interval` is greater than 0. See the "Class Description" section for more information.

NSTimestamp

Inherits from: java.sql.Timestamp : java.util.Date : Object

Implements: NSCoding

Package: com.webobjects.foundation

Class Description

NSTimestamp objects represent a particular instance in time. NSTimestamp is a subclass of java.sql.Timestamp, itself a subclass of java.util.Date. Unlike the NSGregorianDate class in previous versions of the Foundation framework, NSTimestamp does not support calendar functions. Refer to the table below to determine which classes to use to perform date operations in WebObjects.

Class	Description
NSTimestamp	Represents an instance in time
java.util.GregorianCalendar	Represents a calendar date
NSTimeZone	Represents a time zone
NSTimestampFormatter	Converts NSTimestamps to strings and vice versa.

See the Sun's documentation for the java.util.GregorianCalendar class for more information about using calendars in Java.

Common Date Operations

For the following code segments, you need to import `java.util.*` to access Java's date API.

To break up a NSTimestamp into its component year, month, day, hour, etc., you can convert it into a java.util.GregorianCalendar and invoke its `get` method on the individual fields:

```
NSTimestamp myNSTimestamp;
GregorianCalendar myCalendar = new GregorianCalendar();
myCalendar.setTime(myNSTimestamp);
int year = myCalendar.get(GregorianCalendar.YEAR);
int dayOfMonth = myCalendar.get(GregorianCalendar.DAY_OF_MONTH);
```

To create an NSTimestamp based on its components, use java.util.GregorianCalendar:

```
NSTimeZone myTimeZone = new NSTimeZone("EST");
GregorianCalendar myCalendar = GregorianCalendar.getInstance(myTimeZone);
myCalendar.set(year,month,day,hours,minutes,seconds);
NSTimestamp myTimestamp = new NSTimestamp(myCalendar.getTime());
```

To add an offset in Gregorian units to an NSTimestamp, use the following code:

```
<<Please show me the best code to do this.>>
```

To create an NSTimestamp representing the current time, use the no-argument constructor:

```
NSTimestamp currentTime = new NSTimestamp();
```

The Enterprise Objects Framework expects dates to be represented as NSTimestamp objects. To convert a java.util.Date to an NSTimestamp use:

```
NSTimestamp myNSTimestamp = new NSTimestamp(myJavaUtilDate);
```

Since NSTimestamp is a subclass of java.util.Date, you don't need to convert an NSTimestamp into a java.util.Date.

Constants

NSTimestamp provides the following constants.

Constant	Type	Description
DistantFuture	NSTimestamp	An NSTimestamp that represents a date in the distant future (in terms of centuries).
DistantPast	NSTimestamp	An NSTimestamp that represents a date in the distant future (in terms of centuries).

Interfaces Implemented

NSCoding

classForCoder

decodeObject

encodeWithCoder

Method Types

Constructors

NSTimestamp

Instance methods

compare

toString

Deprecated methods

currentTimeIntervalSinceReferenceDate

dayOfCommonEra

dayOfMonth

dayOfWeek

dayOfYear

distantFuture

distantPast

earlierTimestamp

gregorianUnitsSinceTimestamp

hourOfDay

laterTimestamp

microsecondOfSecond

millisecondsToTimeInterval

minuteOfHour

monthOfYear

secondOfMinute

timeIntervalSinceNow

timeIntervalSinceReferenceDate

timeIntervalSinceTimestamp

timeIntervalToMilliseconds

setDate

setHours

setMinutes

setMonth

setNanos

setSeconds

setTime

setYear

timestampByAddingGregorianUnits

timestampByAddingTimeInterval

timeZone

yearOfCommonEra

Constructors

NSTimestamp

```
public NSTimestamp()
```

Creates an NSTimestamp representing the current time, accurate to the millisecond.

```
public NSTimestamp(java.sql.Timestamp date)
```

Creates an NSTimestamp object representing the same instant in time as date.

```
public NSTimestamp(java.util.Date date)
```

Creates an NSTimestamp object representing the same instant in time as date.

```
public NSTimestamp(long milliseconds)
```

Creates an NSTimestamp object representing the specified number of milliseconds since January 1, 1970, 00:00:00 GMT.

```
public NSTimestamp(
    long milliseconds,
    java.util.TimeZone timezone)
```

Creates an NSTimestamp object representing the specified number of milliseconds since January 1, 1970, 00:00:00 in the specified time zone.

```
public NSTimestamp(
    long milliseconds,
    int nanoseconds)
```

Creates an NSTimestamp object representing the specified number of milliseconds since January 1, 1970, 00:00:00 GMT and sets the number of nanoseconds to nanoseconds.

```
public NSTimestamp(
    long milliseconds,
    int nanoseconds,
    java.util.TimeZone timezone)
```

Creates an NSTimestamp object representing the specified number of milliseconds and nanoseconds since January 1, 1970, 00:00:00 in the specified time zone.

```
public NSTimestamp(
    long milliseconds,
    NSTimestamp timestamp)
```

Creates an NSTimestamp object representing the specified number of milliseconds after the time specified by timestamp.

```
public NSTimestamp(
    int year,
    int month,
    int day,
    int hours,
    int minutes,
    int seconds,
    java.util.TimeZone timezone)
```

Deprecated in the Java Foundation framework. Don't use this method. See the "Common Date Operations" (page 284) for information on how to create a NSTimestamp based on its components.

Creates an NSTimestamp object representing the specified year, month, day, hours, minutes, and seconds in the specified time zone.

Static Methods

currentTimeIntervalSinceReferenceDate

```
public static long currentTimeIntervalSinceReferenceDate()
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the number of milliseconds between the system's absolute reference date (the first instant of 1 January 1970, GMT) and the current date and time.

decodeObject

```
public static Object decodeObject(NSCoder coder)
```

Creates a NSTimestamp from data in coder.

See Also: NSCoding

distantFuture

```
public static NSTimestamp distantFuture()
```

Deprecated in the Java Foundation framework. Don't use this method. Use the constant `DistantFuture` instead.

Creates and returns an NSTimestamp that represents a date many centuries in the future. You can pass this value where an NSTimestamp is required to have the date argument essentially ignored. For example, the NSLock method `tryLock(NSTimeStamp)` returns `false` if the receiver fails to acquire the lock before the specified date. You can use the object returned by `distantFuture` as the date argument to wait indefinitely to acquire the lock.

distantPast

```
public static NSTimestamp distantPast()
```

Deprecated in the Java Foundation framework. Don't use this method. Use the constant `DistantPast` instead.

Creates and returns an NSTimestamp that represents a date many centuries in the past. You can use this object in your code as a control date, a guaranteed temporal boundary.

millisecondsToTimeInterval

```
public static long millisecondsToTimeInterval(long milliseconds)
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the time interval in seconds corresponding to the specified time represented in milliseconds. Any fractional part of a second is truncated.

timeIntervalToMilliseconds

```
public static long timeIntervalToMilliseconds(long timeInterval)
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns a time interval in milliseconds corresponding to the specified time interval represented in seconds.

Instance Methods

classForCoder

```
public Class classForCoder()
```

Conformance to NSCoding. See the method description of `classForCoder` in the interface specification for NSCoding.

compare

```
public int compare(NSTimestamp timestamp)
```

Returns an integer indicating whether the receiver is before, after, or the same as `timestamp`. If the receiver is before `timestamp`, this method returns NSComparator.OrderedAscending. If the receiver is after `timestamp`, this method returns NSComparator.OrderedDescending. If the dates match, this method returns NSComparator.OrderedSame.

dayOfCommonEra

```
public int dayOfCommonEra()
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the number of days since the beginning of the Common Era. The base year of the Common Era is 1 C.E. (which is the same as 1 A.D.).

dayOfMonth

```
public int dayOfMonth()
```

Deprecated in the Java Foundation framework. Don't use this method. See the"Class Description" (page 283) for `NSTimestamp` for an alternative.

Returns a number that indicates the day of the month (1 through 31) of the receiver.

dayOfWeek

```
public int dayOfWeek()
```

Deprecated in the Java Foundation framework. Don't use this method. See the"Class Description" (page 283) for `NSTimestamp` for an alternative.

Returns a number that indicates the day of the week (0 through 6) of the receiver; 0 indicates Sunday.

dayOfYear

```
public int dayOfYear()
```

Deprecated in the Java Foundation framework. Don't use this method. See the"Class Description" (page 283) for NSTimestamp for an alternative.

Returns a number that indicates the day of the year (1 through 366) of the receiver.

earlierTimestamp

```
public NSTimestamp earlierTimestamp(NSTimestamp timestamp)
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the receiver or timestamp, whichever is earlier.

encodeWithCoder

```
public void encodeWithCoder(NSCoder coder)
```

Conformance to NSCoding. See the method description of encodeWithCoder in the interface specification for NSCoding.

gregorianUnitsSinceTimestamp

```
public void gregorianUnitsSinceTimestamp(
    NSTimestamp.IntRef years,
    NSTimestamp.IntRef months,
    NSTimestamp.IntRef days,
    NSTimestamp.IntRef hours,
    NSTimestamp.IntRef minutes,
    NSTimestamp.IntRef seconds,
    NSTimestamp timestamp)
```

Deprecated in the Java Foundation framework. Don't use this method. See the"Class Description" (page 283) for NSTimestamp for an alternative.

Computes the time difference in calendar units between the receiver and timestamp and returns it in years, months, days, hours, minutes, and seconds.

NSTimestamp.IntRef is a local class that contains a single element: the integer value.

You can choose any representation you wish for the time difference by passing `null` for the arguments you want to ignore. For example, the following code fragment computes the difference in months, days, and years between two dates:

```
NSTimestamp momsBDay =
     new NSTimestamp(1936, 1, 8, 7, 30, 0, java.util.TimeZone.getTimeZone("EST"));
NSTimestamp dateOfBirth =
     new NSTimestamp(1965, 12, 7, 17, 25, 0, new NSTimeZone("EST"));

NSTimestamp.IntRef years = new NSTimestamp.IntRef();
NSTimestamp.IntRef months = new NSTimestamp.IntRef();
NSTimestamp.IntRef days = new NSTimestamp.IntRef();

dateOfBirth.gregorianUnitsSinceTimestamp(momsBDay, years, months, days,
     null, null, null)
```

This message returns 29 years, 10 months, and 29 days. If you want to express the years in terms of months, you pass `null` for the years argument:

```
dateOfBirth.gregorianUnitsSinceTimestamp(momsBDay, null, months, days,
     null, null, null);
```

This message returns 358 months and 29 days.

hourOfDay

```
public int hourOfDay()
```

Deprecated in the Java Foundation framework. Don't use this method. See the"Class Description" (page 283) for `NSTimestamp` for an alternative.

Returns the hour value (0 through 23) of the receiver. On Daylight Savings "fall back" days, a value of 1 is returned for two consecutive hours, but with a different time zone (the first in daylight savings time and the second in standard time).

laterTimestamp

```
public NSTimestamp laterTimestamp(NSTimestamp timestamp)
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the receiver or `timestamp`, whichever is later.

microsecondOfSecond

```
public int microsecondOfSecond()
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the microseconds value (0 through 999,999) of the receiver.

minuteOfHour

```
public int minuteOfHour()
```

Deprecated in the Java Foundation framework. Don't use this method. See the"Class Description" (page 283) for `NSTimestamp` for an alternative.

Returns the minutes value (0 through 59) of the receiver.

monthOfYear

```
public int monthOfYear()
```

Deprecated in the Java Foundation framework. Don't use this method. See the"Class Description" (page 283) for `NSTimestamp` for an alternative. Note that java.util.Calendar represents months as integers from 0 to 11.

Returns a number that indicates the month of the year (1 through 12) of the receiver.

secondOfMinute

```
public int secondOfMinute()
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the seconds value (0 through 59) of the receiver. See the"Class Description" (page 283) for `NSTimestamp` for an alternative.

setDate

```
public void setDate(int date)
```

Deprecated in the Java Foundation framework. Don't use this method. NSTimestamp objects are immutable.

setHours

```
public void setHours(int hours)
```

Deprecated in the Java Foundation framework. Don't use this method. NSTimestamp objects are immutable.

setMinutes

```
public void setMinutes(int minutes)
```

Deprecated in the Java Foundation framework. Don't use this method. NSTimestamp objects are immutable.

setMonth

```
public void setMonth(int month)
```

Deprecated in the Java Foundation framework. Don't use this method. NSTimestamp objects are immutable.

setNanos

```
public void setNanos(int nanoseconds)
```

Deprecated in the Java Foundation framework. Don't use this method. NSTimestamp objects are immutable.

setSeconds

```
public void setSeconds(int seconds)
```

Deprecated in the Java Foundation framework. Don't use this method. NSTimestamp objects are immutable.

setTime

```
public void setTime(long milliseconds)
```

Deprecated in the Java Foundation framework. Don't use this method. NSTimestamp objects are immutable.

setYear

```
public void setYear(int year)
```

Deprecated in the Java Foundation framework. Don't use this method. NSTimestamp objects are immutable.

timeIntervalSinceNow

```
public long timeIntervalSinceNow()
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the number of milliseconds between the receiver's time and the current system time. This value is negative if the receiver's time is earlier than the current system time.

timeIntervalSinceReferenceDate

```
public long timeIntervalSinceReferenceDate()
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the number of milliseconds between the receiver's time and the reference date (January 1, 1970, 00:00 GMT). This value is negative if the receiver's time is earlier than the reference date.

timeIntervalSinceTimestamp

```
public long timeIntervalSinceTimestamp(NSTimestamp time)
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the number of milliseconds between the receiver's time and time. This value is negative if the receiver's time is earlier than time.

timestampByAddingGregorianUnits

```
public NSTimestamp timestampByAddingGregorianUnits(
    int year,
    int month,
    int day,
    int hour,
    int minute,
    int second)
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns an NSTimestamp that is updated with the year, month, day, hour, minute, and second offsets specified as arguments. The offsets can be positive (future) or negative (past). This method preserves "clock time" across changes in Daylight Savings Time zones and leap years. For example, adding one month to an NSTimestamp with a time of 12 noon correctly maintains time at 12 noon.

The following code fragment shows an NSTimestamp created with a date a week later than an existing NSTimestamp.

```
NSTimestamp now = new NSTimestamp();
NSTimestamp nextWeek =
     now.timestampByAddingGregorianUnits(0, 0, 7, 0, 0, 0);
```

timestampByAddingTimeInterval

```
public NSTimestamp timestampByAddingTimeInterval(long timeInterval)
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns an NSTimestamp with the specified time interval in milliseconds added to the receiver's time.

timeZone

```
public java.util.TimeZone timeZone()
```

Deprecated in the Java Foundation framework. Don't use this method.

Returns the time zone object associated with the receiver. You can explicitly set the time zone to an java.util.TimeZone object using a constructor that takes an java.util.TimeZone object as an argument. If you do not specify a time zone for an object at initialization time, NSTimestamp uses the default time zone for the locale.

toString

```
public String toString()
```

Returns a string representation of the receiver in the form:
"YYYY/MM/DD HH:MM:SS TimeZone".

yearOfCommonEra

```
public int yearOfCommonEra()
```

Deprecated in the Java Foundation framework. Don't use this method. See the"Class Description" (page 283) for `NSTimestamp` for an alternative.

Returns a number that indicates the year, including the century, of the receiver (for example, 1995). The base year of the Common Era is 1 C.E. (which is the same as 1 A.D).

NSTimestampFormatter

Inherits from: java.text.Format : Object

Package: com.webobjects.foundation

Class Description

Instances of NSTimestampFormatter format NSTimestamps into their textual representations and convert textual representations of dates and times into NSTimestamps. You can express the representation of dates and times very flexibly: “Thu 22 Dec 1994” is just as acceptable as “12/22/94”.

You can associate an date pattern with a WOString or WOTextField dynamic element. WebObjects uses an NSTimestampFormatter object to perform the appropriate conversions.

You can also create an NSTimestampFormatter with the constructor, provide a date pattern string, and use java.text.Format’s `format` and `parseObject` methods to convert between NSTimestamps and their textual representations:

```
NSTimestampFormatter formatter = new NSTimestampFormatter(“%m/%d/%y”);
String description = formatter.format(myNSTimestamp);

NSTimestampFormatter formatter = new NSTimestampFormatter(“%m/%d/%y”);
NSTimestamp myNSTimestamp = formatter.parseObject(myTimestampString);
```

Instances of NSTimestampFormatter are immutable.

The Calendar Pattern

You must specify a pattern whenever you create a NSTimestampFormatter. This pattern is a string that contains specifiers that are very similar to those used in the standard C library function `strftime()`. When NSTimestampFormatter converts a date to a string, it uses this pattern.

The date conversion specifiers cover a range of date conventions:

Specifier	Description
`%%`	a '`%`' character
`%a`	abbreviated weekday name
`%A`	full weekday name
`%b`	abbreviated month name
`%B`	full month name
`%c`	shorthand for "`%X %x`", the locale format for date and time
`%d`	day of the month as a decimal number (01-31)
`%e`	same as `%d` but does not print the leading 0 for days 1 through 9
`%F`	milliseconds as a decimal number (000-999)
`%H`	hour based on a 24-hour clock as a decimal number (00-23)
`%I`	hour based on a 12-hour clock as a decimal number (01-12)
`%j`	day of the year as a decimal number (001-366)
`%m`	month as a decimal number (01-12)
`%M`	minute as a decimal number (00-59)
`%p`	AM / PM designation for the locale
`%S`	second as a decimal number (00-59)
`%w`	weekday as a decimal number (0-6), where Sunday is 0
`%x`	date using the date representation for the locale

Specifier	Description
%X	time using the time representation for the locale
%y	year without century (00-99)
%Y	year with century (such as 1990)
%Z	time zone name (such as Pacific Daylight Time)
%z	time zone offset in hours and minutes from GMT (HHMM)

Alternatively, you can specify the pattern using Sun's date pattern specifiers. See Sun's documentation for the java.text.SimpleDateFormat class for more information.

Converting Date Strings Without Time Zones

When you convert a string without a time zone specification to an NSTimestamp, the formatter assumes the time zone is the default parse time zone (see the `defaultParseTimeZone` and `setDefaultParseTimeZone` methods). An analogous time zone, called the default format time zone, is used when converting an NSTimestamp without a time zone to a string.

Sometimes you need to give the user a choice of time zones. For example, you might put the time zones in a pull-down list. In such cases, you can use the `parseObjectInUTC` method to parse a date string for the UTC time zone. The following code shows how you can compute the offset from UTC for the particular time zone the user chooses and add it to the parsed timestamp:

```
NSTimestamp date = (NSTimestamp)myFormatter.parseInUTC(myString);
NSTimeZone tz = NSTimeZone.timeZoneWithName(myTimeZoneName);
int offset = tz.secondsFromGMTForTimestamp(date);
long milliseconds = date.getTime() - offset * 1000;
NSTimestamp dateWithTimeZone = new NSTimestamp(milliseconds);
```

Constructors

NSTimestampFormatter

```
public NSTimestampFormatter()
```

Creates a NSTimestampFormatter with the default pattern (`%m/%d/%y`).

```
public NSTimestampFormatter(String pattern)
```

Creates an NSTimestampFormatter with the pattern string `pattern`. If `pattern` is `null`, this constructor uses the default pattern (`%m/%d/%y`). See "The Calendar Pattern" (page 300) for more information on specifying the pattern string.

```
public NSTimestampFormatter(String pattern,
    java.text.DateFormatSymbols formatSymbols)
```

Creates an NSTimestampFormatter with the specified pattern using the specified date format symbols. If `pattern` is `null`, this constructor uses the default pattern (`%m/%d/%y`) with the slashes replaced by the appropriate date symbol. See "The Calendar Pattern" (page 300) for more information on specifying the pattern string.

```
public NSTimestampFormatter(String pattern,
    java.util.Locale locale)
```

Creates an NSTimestampFormatter with the specified pattern using the date symbols for the specified locale. If `pattern` is `null`, this constructor uses the default pattern (`%m/%d/%y`) with the slashes ("/") replaced by the appropriate date symbol. See "The Calendar Pattern" (page 300) for more information on specifying the pattern string.

Instance Methods

defaultFormatTimeZone

```
public NSTimeZone defaultFormatTimeZone()
```

Returns the default time zone the receiver uses for formatting (converting an NSTimestamp into a string). If the default format time zone is not `null`, the receiver uses the default format time zone when it performs the conversion. Otherwise the receiver uses the time zone of the NSTimestamp that it is converting. The default format time zone itself defaults to `null`.

See Also: `defaultParseTimeZone`

defaultParseTimeZone

```
public synchronized NSTimeZone defaultParseTimeZone()
```

Returns the default time zone the receiver uses for parsing (converting a string to an NSTimestamp). During the conversion, if the NSTimestamp has a time zone (its `timeZone` method returns something other than `null`), the receiver uses the NSTimestamp's time zone. Otherwise the receiver uses the default parse time zone. The default parse time zone itself defaults to the time zone specified by the `user.timezone` system property.

See Also: `defaultFormatTimeZone`

format

```
public StringBuffer format(
    Object object,
    StringBuffer toAppendTo,
    java.text.FieldPosition position)
```

Formats `object` to produce a string, appends the string to `toAppendTo`, and returns the resulting StringBuffer. The `position` parameter specifies an alignment field to place the formatted object. When the method returns, this parameter contains the position of the alignment field. See Sun's java.text.Format documentation for more information.

parseObjectInUTC

```
public Object parseObjectInUTC(
    String source,
    java.text.ParsePosition status)
```

Parses a string to produce an object using UTC as the time zone. This method ignores the time zone specified by the string and the value of the parse time zone. For parameter definitions, see Sun's java.text.Format documentation for the `parseObject` method.

See Also: `parseObject`

parseObject

```
public Object parseObject(
    String source,
    java.text.ParsePosition status)
```

Parses a string to produce an object. If the string does not specify a time zone, uses the default parse time zone. See Sun's java.text.Format documentation for more information.

See Also: `setDefaultParseTimeZone`

pattern

```
public String pattern()
```

Returns the receiver's pattern. See "The Calendar Pattern" (page 300) for more information about the pattern.

setDefaultFormatTimeZone

```
public synchronized void setDefaultFormatTimeZone(NSTimeZone timeZone)
```

Sets the default time zone the receiver uses for formatting (converting an NSTimestamp into a string) to `timeZone`.

See Also: `setDefaultParseTimeZone`

setDefaultParseTimeZone

```
public synchronized void setDefaultParseTimeZone(NSTimeZone timeZone)
```

Sets the default time zone the receiver uses for parsing (converting a string to an NSTimestamp) to timeZone.

See Also: setDefaultFormatTimeZone

setPattern

```
public synchronized void setPattern(String pattern)
```

Sets the receiver's pattern to pattern. See "The Calendar Pattern" (page 300) for more information about the pattern.

toString

```
public String toString()
```

Returns a string representation of the receiver that includes the default format time zone, the default parse time zone, and the pattern.

See Also: defaultFormatTimeZone, defaultParseTimeZone, pattern

NSTimestamp.IntRef

Inherits from: Object

Package: com.webobjects.foundation

Class Description

Deprecated in the Java Foundation framework. NSTimestamp.IntRef objects act as a containers to hold the values returned by NSTimestamp's `gregorianUnitsSinceTimestamp` method and consist solely of an instance variable, `value`. See the `gregorianUnitsSinceTimestamp` deprecated method description in the NSTimestamp class specification for more information about using this class.

Constructors

NSTimestamp.IntRef

```
public NSTimestamp.IntRef()
```

Description forthcoming.

Instance Methods

toString

```
public String toString()
```

Returns a String representation of the receiver.

NSTimeZone

Inherits from:	java.util.TimeZone : Object
Implements:	Cloneable Serializable NSCoding
Package:	com.webobjects.foundation

Class Description

NSTimeZone defines the behavior of time zone objects. Time zone objects represent geopolitical regions. Consequently, these objects have names for these regions. Time zone objects also represent a temporal offset, either plus or minus, from Greenwich Mean Time (GMT) and an abbreviation (such as “PST”).

NSTimeZone provides several constructors to get time zone objects. The class also permits you to set the default time zone within your application (`setDefaultTimeZone`). You can access this default time zone at any time with the `defaultTimeZone` static method, and with the `localTimeZone` static method, you can get a relative time zone object that decodes itself to become the default time zone for any locale in which it finds itself.

Because NSTimeZone is a subclass of java.util.TimeZone, you can also use the java.util.TimeZone API with NSTimeZones.

WARNING NSTimeZone is only intended to be used with NSTimestamp and NSTimestampFormatter. It produces incorrect results when used with Java's date-related classes.

Some NSTimestamp methods return date objects that are automatically bound to time zone objects. These date objects use the functionality of NSTimeZone to adjust dates for the proper locale. Unless you specify otherwise, objects returned from NSTimestamp are bound to the default time zone for the current locale.

Interfaces Implemented

Cloneable

- `clone`

java.io.Serializable

NSCoding

- `decodeObject`
- `classForCoder`
- `encodeWithCoder`

Method Types

Constructors

- `NSTimeZone`

Getting the default time zone

- `localTimeZone`
- `defaultTimeZone`
- `setDefaultTimeZone`

resetSystemTimeZone

Getting time zone information

abbreviationDictionary

knownTimeZoneNames

Getting information about a specific time zone

abbreviation

abbreviationForTimestamp

name

secondsFromGMT

secondsFromGMTForTimestamp

isDaylightSavingTime

isDaylightSavingTimeForTimestamp

data

Comparing time zones

equals

isEqualToTimeZone

Instance methods inherited from java.util.TimeZone

getAvailableIDs

getDefault

getDisplayName

getID

getOffset

getRawOffset

hasSameRules

inDaylightTime

setDefault

setID

setRawOffset

useDaylightTime

Constructors

NSTimeZone

```
public NSTimeZone()
```

This constructor is used internally to implement the java.io.Serializable and java.io.Externalizable interfaces and should be considered private. Use the static factory methods to create time zones.

```
protected NSTimeZone(
    String aTimeZoneName,
    NSData data)
```

This constructor is used internally by NSTimeZone and should be considered private. Use the static factory methods to create time zones.

Static Methods

abbreviationDictionary

```
public static NSDictionary abbreviationDictionary()
```

Returns a dictionary holding the mappings of time zone abbreviations to time zone names.

More than one time zone may have the same abbreviation. For example, US/Pacific and Canada/Pacific both use the abbreviation “PST.” In these cases `abbreviationDictionary` chooses a single name to map the abbreviation to.

decodeObject

```
public static Object decodeObject(NSCoder coder)
```

Creates and returns an NSTimeZone from the data in `coder`.

See Also: NSCoding **Interface Description**

defaultTimeZone

```
public static synchronized NSTimeZone defaultTimeZone()
```

Returns the default time zone set for your application. If no default time zone has been set, this method invokes `systemTimeZone` and returns the system time zone.

See Also: `localTimeZone`, `setDefaultTimeZone`, `systemTimeZone`

getAvailableIDs

```
public static String[] getAvailableIDs()
```

See the method description for `getAvailableIDs` in the java.util.TimeZone class specification.

getDefault

```
public static java.util.TimeZone getDefault()
```

See the method description for `getDefault` in the java.util.TimeZone class specification.

knownTimeZoneNames

```
public static NSArray knownTimeZoneNames()
```

Returns an array of strings listing the names of all the time zones known to the system.

localTimeZone

```
public static NSTimeZone localTimeZone()
```

Returns an object that forwards all messages to the default time zone for your application. This behavior is particularly useful for NSTimestamp objects that are archived or sent as Distributed Objects and may be interpreted in different locales.

See Also: defaultTimeZone, setDefaultTimeZone

resetSystemTimeZone

```
public static synchronized void resetSystemTimeZone()
```

Clears the previously determined system time zone, if any. Subsequent calls to systemTimeZone will attempt to redetermine the system time zone.

setDefault

```
public static synchronized void setDefault(java.util.TimeZone zone)
```

See the method description for setDefault in the java.util.TimeZone class specification.

setDefaultTimeZone

```
public static synchronized void setDefaultTimeZone(NSTimeZone aTimeZone)
```

Sets the time zone appropriate for your application. There can be only one default time zone, so by setting a new default time zone, you lose the previous one.

See Also: defaultTimeZone, localTimeZone

systemTimeZone

```
public static synchronized NSTimeZone systemTimeZone()
```

Returns the time zone currently used by the system. If it can't figure out the current time zone, returns the GMT time zone.

timeZoneForSecondsFromGMT

```
public static synchronized NSTimeZone timeZoneForSecondsFromGMT(int seconds)
```

Returns a time zone object with seconds offset from Greenwich Mean Time. The name of the new time zone is GMT + / - the offset, in hours and minutes. Time zones created with this never have daylight savings and the offset is constant no matter the date; the name and abbreviation do NOT follow the POSIX convention of minutes-west.

See Also: timeZoneWithName

timeZoneWithName

```
public static synchronized NSTimeZone timeZoneWithName(
    String aTimeZoneName,
    boolean trybbreviation)
```

Returns the time zone object identified by the name aTimeZoneName. If tryAbbreviation is false, this method searches the time zone information directory for matching names. If tryAbbreviation is true, this method attempts to resolve the abbreviation to a name using the abbreviation dictionary. Returns null if there is no match on the name.

See Also: timeZoneForSecondsFromGMT, knownTimeZoneNames

timeZoneWithNameAndData

```
public static synchronized NSTimeZone timeZoneWithNameAndData(
    String aTimeZoneName,
    NSData data)
```

Returns the time zone with the name aTimeZoneName whose data has been initialized using the contents of data. You should not call this method directly—use timeZoneWithName instead.

Instance Methods

abbreviation

```
public String abbreviation()
```

Returns the abbreviation for the time zone, such as “EDT” (Eastern Daylight Time). Invokes `abbreviationForTimestamp` with the current date as the argument.

abbreviationForTimestamp

```
public String abbreviationForTimestamp(NSTimestamp aTimestamp)
```

Returns the abbreviation for the time zone object at the date specified by `aTimestamp`. Note that the abbreviation may be different at different dates. For example, during Daylight Savings Time the US/Eastern time zone has an abbreviation of “EDT.” At other times, its abbreviation is “EST.”

classForCoder

```
public Class classForCoder()
```

Conformance to NSCoding.

See Also: `classForCoder` (NSCoding)

clone

```
public Object clone()
```

Simply returns the receiver. Since NSArrays are immutable, there’s no need to make an actual clone.

data

```
public NSData data()
```

Returns the data that stores the information used by the time zone. This data should be treated as an opaque object.

encodeWithCoder

```
public void encodeWithCoder(NSCoder coder)
```

Conformance to NSCoding. See the method description for `encodeWithCoder` in the NSCoding interface specification.

equals

```
public boolean equals(Object anObject)
```

Returns `true` if `anObject` is an NSTimeZone and its contents are equal to the receiver's or `false` otherwise. If you know that `anObject` is an NSTimestamp, use the more efficient method `isEqualToTimeZone` instead.

getDisplayName

```
public String getDisplayName(
    boolean daylight,
    int style,
    java.util.Locale locale)
```

Returns the name of the equivalent java.util.SimpleTimeZone if one exists. Otherwise returns the receiver's geopolitical region name.

getID

```
public String getID()
```

Returns the receiver's geopolitical region name.

getOffset

```
public int getOffset(
    int era,
    int year,
    int month,
    int day,
    int dayOfWeek,
    int milliseconds)
```

See the method description for `getOffset` in the java.util.TimeZone class specification.

getRawOffset

```
public int getRawOffset()
```

See the method description for `getRawOffset` in the java.util.TimeZone class specification. For NSTimeZones, this method always returns 0.

hashCode

```
public synchronized int hashCode()
```

See the method description for `hashCode` in the Object class specification.

hasSameRules

```
public boolean hasSameRules(java.util.TimeZone other)
```

Returns `true` if other is the same as the receiver (as determined by `equals`.).

inDaylightTime

```
public boolean inDaylightTime(java.util.Date date)
```

See the method description for `inDaylightTime` in the java.util.TimeZone class specification.

isDaylightSavingTime

```
public boolean isDaylightSavingTime()
```

Returns `true` if the time zone is currently using Daylight Savings Time. This method invokes `isDaylightSavingTimeForTimestamp` with the current date as the argument.

isDaylightSavingTimeForTimestamp

```
public boolean isDaylightSavingTimeForTimestamp(NSTimestamp aTimestamp)
```

Returns `true` if the time zone uses Daylight Savings Time at the date specified by `aTimestamp`.

isEqualToTimeZone

```
public boolean isEqualToTimeZone(NSTimeZone aTimeZone)
```

Returns `true` if `aTimeZone` and the receiving time zone have the same name and data.

name

```
public String name()
```

Returns the geopolitical region name that identifies the time zone.

readExternal

```
public void readExternal(java.io.ObjectInput input)
    throws java.io.IOException,ClassNotFoundException
```

Description forthcoming.

readResolve

```
public Object readResolve() throws java.io.ObjectStreamException
```

Conformance to java.io.Serializable.

secondsFromGMT

```
public int secondsFromGMT()
```

Returns the current difference in seconds between the time zone and Greenwich Mean Time.

secondsFromGMTForTimestamp

```
public int secondsFromGMTForTimestamp(NSTimestamp aTimestamp)
```

Returns the difference in seconds between the time zone and Greenwich Mean Time at the date specified by `aTimestamp`. This may be different from the current difference if the time zone changes its offset from GMT at different points in the year—for example, the U.S. time zones change with daylight savings time.

setID

```
public void setID(String ID)
```

Throws an IllegalStateException because NSTimeZones are immutable.

setRawOffset

```
public void setRawOffset(int offsetMillis)
```

Throws an IllegalStateException because NSTimeZones are immutable.

toString

```
public String toString()
```

Returns a string representation of the receiver that indicates the receiver's name, the receiver's current offset from GMT, and whether the receiver is currently using Daylight Savings Time.

useDaylightTime

```
public boolean useDaylightTime()
```

See the method description for `useDaylightTime` in the java.util.TimeZone class specification.

writeExternal

```
public void writeExternal(
    java.io.ObjectOutput output) throws java.io.IOException
```

Description forthcoming.

Notifications

SystemTimeZoneDidChangeNotification

```
public static final String SystemTimeZoneDidChangeNotification;
```

NSUndoManager

Inherits from:	Object
Implements:	NSDisposable Serializable
Package:	com.webobjects.foundation

Class Description

NSUndoManager is a general-purpose recorder of operations for undo and redo. You register an undo operation by specifying the object that's changing (or the owner of that object), along with a method to invoke to revert its state, and the arguments for that method. NSUndoManager groups all operations within a single cycle of the run loop, so that performing an undo reverts all changes that occurred during the loop. Also, when performing undo an NSUndoManager saves the operations reverted so that you can redo the undos.

NSUndoManager is implemented as a class of the Foundation framework because executables other than applications might want to revert changes to their states. For example, you might have an interactive command-line tool with undo and redo commands. However, users typically see undo and redo as application features. WebObjects applications can use NSUndoManagers to undo and redo user operations. Typically a session's editing context has an undo manager that provides undo and redo operations on enterprise objects. For more information, see the class specification for EOEditingContext (eocontrol package).

Operations and Groups

An undo operation is a method for reverting a change to an object, along with the arguments needed to revert the change (for example, its state before the change). Undo operations are typically collected in undo groups, which represent whole revertible actions, and are stored on a stack. Redo operations and groups are simply undo operations stored on a separate stack (described below). When an NSUndoManager performs undo or redo, it's actually undoing or redoing an entire group of operations. For example, a user could change the first name and the last name of an employee. An application might package both operations as a group, so when the user chooses Undo, both the first and last names are reverted. To undo a single operation, the operation must be packaged alone in a group.

NSUndoManager normally creates undo groups automatically during the run loop. The first time it's asked to record an undo operation in the run loop, it creates a new group. Then, at the end of the loop, it closes the group. You can create additional, nested undo groups within these default groups using the `beginUndoGrouping` and `enableUndoRegistration` methods. You can also turn off the default grouping behavior using `setGroupsByEvent`.

The Undo and Redo Stacks

Undo groups are stored on a stack, with the oldest groups at the bottom and the newest at the top. The undo stack is unlimited by default, but you can restrict it to a maximum number of groups using the `setLevelsOfUndo` method. When the stack exceeds the maximum, the oldest undo groups are dropped from the bottom.

Initially, both stacks are empty. Recording undo operations adds to the undo stack, but the redo stack remains empty until an undo is performed. Performing an undo causes the reverting operations in the latest group to be applied to their objects. Since these operations cause changes to the objects' states, the objects presumably register new operations with the NSUndoManager, this time in the reverse direction from the original operations. Since the NSUndoManager is in the process of performing undo, it records these operations as redo operations on the redo stack. Consecutive undos add to the redo stack. Subsequent redo operations pull the operations off the redo stack, apply them to the objects, and push them back onto the undo stack.

The redo stack's contents last as long as undo and redo are performed successively. However, because applying a new change to an object invalidates the previous changes, as soon as a new undo operation is registered, the redo stack is cleared. This prevents redo from returning objects to an inappropriate prior state. You can check for the ability to undo and redo with the `canUndo` and `canRedo` methods.

Registering Undo Operations

To add an undo operation to the undo stack, you must register it with the object that will perform the undo operation. To register the undo operation you specify a selector with a single object argument. When an object changes, the object itself (or another object acting on its behalf) records its attributes prior to the change in the argument object. (This argument is frequently an NSDictionary object, but it can be any object.) Performing the undo then involves resetting the object with these attributes.

To record a simple undo operation, you need only invoke `registerUndoWithTarget`, giving the object to be sent the undo operation selector, the selector to invoke, and an argument to pass with that message. The target object is usually not the actual object whose state is changing; instead, it's the client object, a document or container that holds many undoable objects. The argument is an object that captures the state of the object before the change is made. If you have multiple arguments, use `registerUndoWithTargetAndArguments`.

In most applications a single instance of NSUndoManager belongs to an object that contains or manages other objects. This is particularly the case with document-based applications, where a single object is responsible for all undo and redo operations for a document. An object such as this is often called the NSUndoManager's client. Each client object has its own NSUndoManager. The client claims exclusive right to alter its undoable objects so that it can record undo operations for all changes. In the specific case of documents, this scheme keeps each pair of undo and redo stacks separate so that when an undo is performed, it applies to the focal document in the application (typically the one displayed in the key window). It also relieves the individual objects in a document from having to know the identity of their NSUndoManager or from having to track changes to themselves.

However, an object that is changed can have its own NSUndoManager and perform its own undo and redo operations. For example, you could have a custom view that displays images dragged into it; with each successful drag operation, it registers a new undo group. If the view is then selected (that is, made first responder) and the Undo command applied, the previously displayed image would be redisplayed.

Performing Undo and Redo

Performing undo and redo is usually as simple as sending `undo` and `redo` messages to the NSUndoManager. `undo` closes the last open undo group and then applies all the undo operations in that group (recording any undo operations as redo operations instead). `redo` likewise applies all the redo operations on the top redo group.

undo is intended for undoing top-level groups, and shouldn't be used for nested undo groups. If any unclosed, nested undo groups are on the stack when undo is invoked, it throws an exception. To undo nested groups, you must explicitly close the group with an enableUndoRegistration message, then use undoNestedGroup to undo it. Note also that if you turn off automatic grouping by event with setGroupsByEvent, you must explicitly close the current undo group with enableUndoRegistration before invoking either undo method.

Undo Notifications

An NSUndoManager regularly posts checkpoint notifications to synchronize the inclusion of undo operations in undo groups. Objects sometimes delay performing changes, for various reasons. This means they may also delay registering undo operations for those changes. Because NSUndoManager collects individual operations into groups, it must be sure to synchronize its client with the creation of these groups so that operations are entered into the proper undo groups. To this end, whenever an NSUndoManager opens or closes a new undo group (except when it opens a top-level group), it posts an CheckpointNotification so observers can apply their pending undo operations to the group in effect. The NSUndoManager's client should register itself as an observer for this notification and record undo operations for all pending changes upon receiving it.

NSUndoManager also posts a number of other notifications at specific intervals: when a group is created, when a group is closed, and just before and just after both undo and redo operations. For more on notifications, see "Notifications" (page 335).

Constants

NSUndoManager provides the following constant to specify the priority of closing an undo group compared to other operations that take place after the current event ends. Undo groups are automatically closed at the end of the event. The priority specified by this constant is lower than the priority for an EOEditingContext to flush its changes.

Constant	Type	Description
UndoCloseGroupingRunLoopOrdering	int	Specifies the priority for closing the current undo group compared to other operations in the delayed callback queue.

Interfaces Implemented

NSDisposable

dispose

Method Types

Registering undo operations

registerUndoWithTarget

registerUndoWithTargetAndArguments

Checking undo ability

canUndo

canRedo

Performing undo and redo

undo

undoNestedGroup

redo

Limiting the undo stack

setLevelsOfUndo

levelsOfUndo

Creating undo groups

beginUndoGrouping

endUndoGrouping

setGroupsByEvent

groupsByEvent

groupingLevel

Disabling undo

disableUndoRegistration

enableUndoRegistration

isUndoRegistrationEnabled

Checking whether undo or redo is being performed

isUndoing

isRedoing

Clearing undo operations

removeAllActions

removeAllActionsWithTarget

Constructors

NSUndoManager

```
public NSUndoManager()
```

Creates an NSUndoManager object.

Instance Methods

beginUndoGrouping

```
public void beginUndoGrouping()
```

Marks the beginning of an undo group. All individual undo operations before a subsequent `endUndoGrouping` message are grouped together and reversed by a later `undo` message. By default undo groups are begun automatically at the start of the event loop, but you can begin your own undo groups with this method, and nest them within other groups.

This method posts an `CheckpointNotification` unless a top-level undo is in progress. It posts a `DidOpenUndoGroupNotification` if a new group was successfully created.

canRedo

```
public boolean canRedo()
```

Returns `true` if the receiver has any actions to redo, `false` if it doesn't.

Because any undo operation registered clears the redo stack, this method posts an `CheckpointNotification` to allow clients to apply their pending operations before testing the redo stack.

canUndo

```
public boolean canUndo()
```

Returns `true` if the receiver has any actions to undo, `false` if it doesn't. This does not mean you can safely invoke `undo` or `undoNestedGroup`—you may have to close open undo groups first.

See Also: `enableUndoRegistration`, `registerUndoWithTarget`

disableUndoRegistration

```
public void disableUndoRegistration()
```

Disables the recording of undo operations, whether by `registerUndoWithTarget` or by invocation-based undo. This method can be invoked multiple times by multiple clients. `enableUndoRegistration` must be invoked an equal number of times to re-enable undo registration.

dispose

```
public void dispose()
```

Conformance to NSDisposable. See the method description of `dispose` in the interface specification for NSDisposable.

enableUndoRegistration

```
public void enableUndoRegistration()
```

Enables the recording of undo operations. Because undo registration is enabled by default, it is often used to balance a prior `disableUndoRegistration` message. Undo registration isn't actually re-enabled until an enable message balances the last disable message in effect. Throws an IllegalStateException if invoked while no `disableUndoRegistration` message is in effect.

endUndoGrouping

```
public void endUndoGrouping()
```

Marks the end of an undo group. All individual undo operations back to the matching `beginUndoGrouping` message are grouped together and reversed by a later `undo` or `undoNestedGroup` message. Undo groups can be nested, thus providing functionality similar to nested transactions. Throws an IllegalStateException if there's no `beginUndoGrouping` message in effect.

This method posts an `CheckpointNotification` and an `WillCloseUndoGroupNotification` just before the group is closed.

See Also: `levelsOfUndo`

groupingLevel

```
public int groupingLevel()
```

Returns the number of nested undo groups (or redo groups, if Redo was last invoked) in the current event loop. If zero is returned, there is no open undo or redo group.

See Also: `levelsOfUndo`, `setLevelsOfUndo`

groupsByEvent

```
public boolean groupsByEvent()
```

Returns `true` if the receiver automatically creates undo groups around each pass of the run loop, `false` if it doesn't. The default is `true`.

See Also: `beginUndoGrouping`

isRedoing

```
public boolean isRedoing()
```

Returns `true` if the receiver is in the process of performing its `redo` method, `false` otherwise.

isUndoRegistrationEnabled

```
public boolean isUndoRegistrationEnabled()
```

Returns whether the recording of undo operations is enabled. Undo registration is enabled by default.

See Also: `disableUndoRegistration`, `enableUndoRegistration`

isUndoing

```
public boolean isUndoing()
```

Returns `true` if the receiver is in the process of performing an `undo` or `undoNestedGroup`, `false` otherwise.

levelsOfUndo

```
public int levelsOfUndo()
```

Returns the maximum number of top-level undo groups the receiver will hold. If ending the current undo group will result in the number of groups exceeding this limit, the oldest groups are dropped from the stack. A limit of zero indicates no limit, so old undo groups are never dropped. The default is zero.

See Also: enableUndoRegistration, setLevelsOfUndo

redo

```
public void redo()
```

Performs the operations in the last group on the redo stack, if there are any, recording them on the undo stack as a single group. Throws an IllegalStateException if the method is invoked during an undo operation.

This method posts an CheckpointNotification and WillRedoChangeNotification before it performs the redo operation, and it posts the DidRedoChangeNotification after it performs the redo operation.

See Also: registerUndoWithTarget

registerUndoWithTarget

```
public void registerUndoWithTarget(
    Object target,
    NSSelector aSelector,
    Object anObject)
```

Records a single undo operation for target, so that when undo is performed it's sent aSelector with anObject as the sole argument. Also clears the redo stack. See "Registering Undo Operations" (page 325) in the class description for more information.

Throws an IllegalStateException if invoked when no undo group has been established using beginUndoGrouping. Undo groups are normally set by default, so you should rarely need to begin a top-level undo group explicitly.

See Also: undoNestedGroup, groupingLevel

registerUndoWithTargetAndArguments

```
public void registerUndoWithTargetAndArguments(
    Object target,
    NSSelector selector,
    Object[] parameters[])
```

removeAllActions

```
public void removeAllActions()
```

Clears the undo and redo stacks and reenables the receiver.

See Also: enableUndoRegistration, removeAllActionsWithTarget

removeAllActionsWithTarget

```
public void removeAllActionsWithTarget(Object target)
```

Clears the undo and redo stacks of all operations involving target as the recipient of the undo message. Doesn't re-enable the receiver if it's disabled. An object that shares an NSUndoManager with other clients should invoke this message in its implementation of finalize. <<True?>>

See Also: enableUndoRegistration, removeAllActions

setGroupsByEvent

```
public void setGroupsByEvent(boolean flag)
```

Sets whether the receiver automatically groups undo operations during the run loop. If flag is true, the receiver creates undo groups around each pass through the run loop; if flag is false it doesn't. The default is true.

If you turn automatic grouping off, you must close groups explicitly before invoking either undo or undoNestedGroup.

See Also: groupingLevel, groupsByEvent

setLevelsOfUndo

```
public void setLevelsOfUndo(int levels)
```

Sets the maximum number of top-level undo groups the receiver will hold to `levels`. When ending an undo group results in the number of groups exceeding this limit, the oldest groups are dropped from the stack. A limit of zero indicates no limit, so that old undo groups are never dropped. The default is zero.

If invoked with a limit below the prior limit, old undo groups are immediately dropped.

See Also: `enableUndoRegistration`, `levelsOfUndo`

undo

```
public void undo()
```

Closes the top-level undo group if necessary and invokes `undoNestedGroup`. It also invokes `endUndoGrouping` if the nesting level is 1. Throws an InternalInconsistencyException if more than one undo group is open (that is, if the last group isn't at the top level).

This method posts an "CheckpointNotification" (page 335).

See Also: `enableUndoRegistration`, `groupingLevel`

undoNestedGroup

```
public void undoNestedGroup()
```

Performs the undo operations in the last undo group (whether top-level or nested), recording the operations on the redo stack as a single group. Throws an InternalInconsistencyException if any undo operations have been registered since the last `enableUndoRegistration` message.

This method posts an "CheckpointNotification" (page 335) and "WillUndoChangeNotification" (page 337) before it performs the undo operation, and it posts the "DidUndoChangeNotification" (page 336) after it performs the undo operation.

See Also: `undo`

Notifications

CheckpointNotification

```
public static final String CheckpointNotification
```

Posted whenever an NSUndoManager opens or closes an undo group (except when it opens a top-level group), and when an NSUndoManager checks the redo stack in `canRedo`. The notification contains:

notification object
: The NSUndoManager

`userInfo`
: `null`

DidOpenUndoGroupNotification

```
public static final String DidOpenUndoGroupNotification
```

Posted whenever an NSUndoManager opens an undo group, which occurs in an invocation of `beginUndoGrouping`. The notification contains:

notification object
: The NSUndoManager

`userInfo`
: `null`

DidRedoChangeNotification

```
public static final String DidRedoChangeNotification
```

Posted just after an NSUndoManager performs a redo operation (`redo`). The notification contains:

notification object
: The NSUndoManager

userInfo
: null

DidUndoChangeNotification

```
public static final String DidUndoChangeNotification
```

Posted just after an NSUndoManager performs an undo operation. If you invoke `undo` or `undoNestedGroup`, this notification will be posted. The notification contains:

notification object
: The NSUndoManager

userInfo
: null

WillCloseUndoGroupNotification

```
public static final String WillCloseUndoGroupNotification
```

Posted whenever an NSUndoManager closes an undo group, which occurs in an invocation of `endUndoGrouping`. The notification contains:

notification object
: The NSUndoManager

userInfo
: null

WillRedoChangeNotification

```
public static final String WillRedoChangeNotification
```

Posted just before an NSUndoManager performs a redo operation (`redo`). The notification contains:

notification object
: The NSUndoManager

userInfo

null

WillUndoChangeNotification

```
public static final String WillUndoChangeNotification
```

Posted just before an NSUndoManager performs an undo operation. If you invoke `undo` or `undoNestedGroup`, this notification will be posted. The notification contains:

notification object

The NSUndoManager

userInfo

null

NSValidation.DefaultImplementation

Inherits from: Object

Package: com.webobjects.foundation

Class Description

The NSValidation.DefaultImplementation class provides default implementations of the NSValidation interface. For more information, see the NSValidation interface specification.

Static Methods

validateTakeValueForKeyPath

```
public static Object validateTakeValueForKeyPath(
    Object anObject,
    Object value,
    String key) throws NSValidation.ValidationException
```

Confirms that `value` is legal for the receiver's property named by `keyPath`, and assigns the value to the property if it's legal (and if `value` is different from the current value), or throws an NSValidation.ValidationException if `value` isn't legal.

See Also: `validateTakeValueForKeyPath` (NSValidation)

validateValueForKey

```
public static Object validateValueForKey(
    Object anObject,
    Object value,
    String key) throws NSValidation.ValidationException
```

Confirms that `value` is legal for the receiver's property named by `key`, and returns the validated value if it's legal, or throws an NSValidation.ValidationException if it isn't.

See Also: `validateValueForKey` (NSValidation)

NSValidation.ValidationException

Inherits from: RuntimeException : Exception : Throwable : Object

Package: com.webobjects.foundation

Class Description

Instances of the NSValidation.ValidationException class are created and thrown when an error condition is encountered during the validation of an object that implements NSValidation. For more information, see the interface specification for NSValidation.

Constants

NSValidation.ValidationException defines the following constants:

Constant	Type	Description
AdditionalExceptionsKey	String	The key for an entry in the exception's user info dictionary that contains subexceptions. This constant is deprecated. You should access this user info dictionary entry using the additionalExceptions method.
ValidatedKeyUserInfoKey	String	The key for an entry in the exception's user info dictionary. The entry contains the key for the property that failed to validate. This constant is deprecated. You should access this user info dictionary entry using the key method.
ValidatedObjectUserInfoKey	String	A key for an entry in the exception's user info dictionary. The entry contains the object that failed to validate. This constant is deprecated. You should access this user info dictionary entry using the object method.

Constructors

NSValidation.ValidationException

```
public NSValidation.ValidationException(String message)
```

Creates and returns a new exception with message as the message.

```
public NSValidation.ValidationException(
    String message,
    NSDictionary userInfo)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `NSValidation.ValidationException(String,Object,String)` instead.

```
public NSValidation.ValidationException(
    String message,
    Object anObject,
    String key)
```

Creates and returns a new exception with `message` as the message and a userInfo dictionary specifying `anObject` for the `ValidatedObjectUserInfoKey` and `key` for the `ValidatedKeyUserInfoKey`.

Static Methods

aggregateExceptionWithExceptions

```
public static NSValidation.ValidationException aggregateExceptionWithExceptions(
    NSArray exceptions)
```

Returns an exception that is the aggregate of the exceptions in the `exceptions` array. The returned aggregate exception has the message and userInfo dictionary of the first exception in the `exceptions` array, but the userInfo dictionary is augmented with the list of subexceptions under the key `AdditionalExceptionsKey`.

Instance Methods

additionalExceptions

```
public NSArray additionalExceptions()
```

Returns the array in the receiver's userInfo dictionary for the `AdditionalExceptionsKey`.

exceptionAddingEntriesToUserInfo

```
public NSValidation.ValidationException exceptionAddingEntriesToUserInfo(
    Object anObject,
    String key)
```

Deprecated in the Java Foundation framework. Don't use this method. Use `exceptionWithObjectAndKey` instead. Returns a new exception that is a copy of the receiver message and userInfo, but whose userInfo dictionary has been augmented with anObject and key.

exceptionWithObjectAndKey

```
public NSValidation.ValidationException exceptionWithObjectAndKey(Object anObject,
    String key)
```

Returns a new exception with the same message as the receiver, but whose userInfo dictionary contains contains `anObject` and `key`. When validation exceptions are raised by certain validation methods such as `validateValueForKey`, this method is invoked on the exception to create a duplicate exception with object and property information stored to the new exception's userInfo dictionary. The information is stored under the keys `ValidatedObjectUserInfoKey` and `ValidatedKeyUserInfoKey`, respectively. The exception this method returns has the same message as the original, receiving exception; the only difference is the userInfo dictionary.

key

```
public String key()
```

Returns the key in the receiver's userInfo dictionary for the `ValidatedKeyUserInfoKey`.

object

```
public Object object()
```

Returns the value in the receiver's userInfo dictionary for the `ValidatedObjectUserInfoKey`.

userInfo

```
public NSDictionary userInfo()
```

Deprecated in the Java Foundation framework. Don't use this method. Access the individual entries using the `key` and `object` methods.

Returns the receiver's userInfo dictionary.

NSValidation.Utility

Inherits from: Object

Package: com.webobjects.foundation

Class Description

The NSValidation.Utility class is a convenience that allows you to access the properties of NSValidation objects and non-NSValidation objects using the same code. For more information, see the NSValidation interface specification.

Static Methods

validateTakeValueForKeyPath

```
public static Object validateTakeValueForKeyPath(
    Object anObject,
    Object value,
    String keyPath) throws NSValidation.ValidationException
```

If anObject is an NSValidation, invokes validateTakeValueForKeyPath on anObject; otherwise invokes NSValidation.DefaultImplementation's validateTakeValueForKeyPath method with anObject as the object on which to operate.

validateValueForKey

```
public static Object validateValueForKey(
    Object anObject,
    Object value,
    String key) throws NSValidation.ValidationException
```

If `anObject` is an NSValidation, invokes `validateValueForKey` on `anObject`; otherwise invokes NSValidation.DefaultImplementation's `validateValueForKey` method with `anObject` as the object on which to operate.

NSArray.Operator

Package: com.webobjects.foundation

Interface Description

The NSArray.Operator interface defines an API for performing operations on the elements in an array. The method, `compute`, takes as its arguments an array and a key path. The array provides the objects on which to operate, and the optional key path further specifies a particular property on which to operate.

As an example, consider an operator that computes averages. To get the average salary for a set of Employee objects, send the operator a `compute` message with "salary" as the key path. The operator gets the `salary` value from each of the objects in the array and then returns the computed average.

Instead of invoking `compute` directly on an operator, you can use NSArray's key-value coding methods with a specially formatted key. The character "@" introduces the name of the operator you want to perform. For example, to compute the average salary of an array's elements, you could invoke `valueForKeyPath` on the array with "@avg.salary" as the key path. For more information, see the NSArray class specification.

Additionally, NSArray provides a set of default operators. To access the operators, use the method `operatorForKey`, specifying the name of the operator as an argument. For information on the default operators, see the NSArray class specification.

You can augment the set of default operators with your own custom operator. Simply create a class that implements NSArray.Operator. To make it available to NSArray for use with key-value coding, use the method `setOperatorForKey`.

Instance Methods

compute

```
public Object compute(
    NSArray values,
    String keyPath)
```

Performs an operation on the elements in values and returns the result. The keyPath argument optionally specifies a particular property of the elements in values to perform the operation on.

NSCoding

Package: com.webobjects.foundation

Interface Description

The NSCoding interface declares the methods that a class must implement so that instances of that class can be encoded and decoded. This capability provides the basis for archiving (where objects and other structures are stored on disk) and distribution (where objects are copied to different address spaces). See the NSCoder class specification for an introduction to coding.

In keeping with object-oriented design principles, an object being encoded or decoded is responsible for encoding and decoding its instance variables. A coder instructs the object to do so by invoking `encodeWithCoder` or `decodeObject`, respectively. `encodeWithCoder` instructs the object to encode its instance variables to the coder provided. Conversely, `decodeObject` is the method that creates an object from the data in the coder provided.

The method `decodeObject` isn't strictly part of the NSCoding interface, but it is required of any class that implements the interface. `decodeObject` is a static method, and therefore can't be formally declared in the NSCoding interface. Any class that should be codable must adopt the NSCoding interface, implement its methods, and implement the static method `decodeObject`.

Encoding

When an object receives an encodeWithCoder message, it should encode all of its vital instance variables, after sending a message to super if its superclass also conforms to the NSCoding interface. An object doesn't have to encode all of its instance variables. Some values may not be important to reestablish and others may be derivable from related state upon decoding. Other instance variables should be encoded only under certain conditions.

For example, suppose you were creating a fictitious MapView class that displays a legend and a map at various magnifications. The MapView class defines several instance variables, including the name of the map and the current magnification. The encodeWithCoder method of MapView might look like the following:

```
public void encodeWithCoder(NSCoder coder) {
     super.encodeWithCoder(coder);
     coder.encodeObject(mapName);
     coder.encodeInt(magnification);
}
```

This example assumes that the superclass of MapView also implements the NSCoding interface. If the superclass of your class does not implement NSCoding, you should omit the line that invokes super's encodeWithCoder method.

encodeObject and encodeInt are coder methods that you can use to encode instance variables of your class. There are other coder methods for other types. The coder also defines corresponding methods for decoding values. See the NSCoder class specification for a list of methods.

Decoding

In decodeObject the class should first send a message to super's implementation of decodeObject (if appropriate) to initialize inherited instance variables. It should then decode and initialize its own. MapView's implementation of decodeObject might look like this:

```
public static Object decodeObject(NSCoder coder) {
     MapView result = (MapView)(MapView.class.decodeObject(coder));
     result.mapName = (String)coder.decodeObject();
     result.magnification = coder.decodeInt();
     return result;
}
```

If the superclass of your class does not implement NSCoding, you should simply create a new instance of your class instead of invoking the superclass's `decodeObject` method.

Making Substitutions During Coding

During encoding a coder allows an object being coded to substitute a different class for itself than the object's actual class. For example, this allows a private class to be represented in a coder by a public class. To allow the substitution, a coder invokes the method `classForCoder` on the object before its encoded. The coder uses the class returned by this method instead of the object's actual class.

Static Methods (in Java) or Class Methods (in ObjC)

decodeObject

```
public static Object decodeObject(NSCoder coder)
```

Creates an object from the data in `coder`. Classes that implement the NSCoding interface must also implement this method.

This method isn't strictly part of this interface because static methods can't be formally declared in an interface. However, this method is so closely related to the interface as to be considered part of it.

Instance Methods

classForCoder

```
public Class classForCoder()
```

Allows the receiver, before being encoded, to substitute a class other than its own in a coder. For example, private subclasses can substitute the name of a public superclass when being encoded.

encodeWithCoder

```
public void encodeWithCoder(NSCoder coder)
```

Encodes the receiver using coder.

NSDisposable

Implemented by: NSDisposableRegistry
NSUndoManager

Package: com.webobjects.foundation

Interface Description

The NSDisposable interface declares one method, `dispose`, in which an object prepares for destruction. In `dispose`, an object should clear all references that other objects have to it. For example, if an NSDisposable object has assigned itself as another object's delegate, the NSDisposable object should set the other object's delegate to `null` in `dispose`, thus clearing the other object's reference to the NSDisposable object. You should implement this interface if your object is a delegate for another object.

NSDisposable is needed to clean up references to objects that ought to be destroyed. As an example, consider NSNotificationCenter. When an object registers for notifications, the notification center creates a reference to that object so that it can perform the notification at the appropriate time. Unless the object removes itself as an observer of the notification, the NSNotificationCenter's reference to the object prevents the object from being garbage collected.

By implementing NSDisposable, objects are given a chance to remove references that other objects have to them. This allows other objects to send `dispose` messages to NSDisposable objects when the NSDisposable objects are no longer needed. As an example, Direct to Java Client disposes of controllers when they're no longer needed, and subsequently, the NSDisposable controllers are garbage collected.

Guidelines

You should implement NSDisposable if your object is a delegate for another object. If you do implement NSDisposable, you should be sure that your `dispose` method will be invoked. If it won't be invoked automatically, you can add yourself to an appropriate NSDisposableRegistry. Known registries are provided by the com.webobjects.eoapplication classes EOController and EOArchive.

Instance Methods

dispose

```
public void dispose()
```

Invoked when the receiver should prepare itself for destruction. Implementations of this method should break connections that other objects have to the receiver, including unregistering for notifications, resigning as other objects' delegates, and so on.

NSKeyValueCoding

Package: com.webobjects.foundation

Interface Description

The NSKeyValueCoding interface defines a data transport mechanism in which the properties of an object are accessed indirectly by name (or key), rather than directly through invocation of an accessor method or as instance variables. Thus, all of an object's properties can be accessed in a consistent manner. The `takeValueForKey` method sets the value for an object's property, and `valueForKey` returns the value for an object's property.

The NSKeyValueCoding interface contains an inner interface, NSKeyValueCoding.ErrorHandling, that defines an extension to the basic NSKeyValueCoding interface for handling errors that occur during key-value coding (see the NSKeyValueCoding.ErrorHandling interface specification).

Additionally, NSKeyValueCoding contains two inner classes, NSKeyValueCoding. DefaultImplementation and NSKeyValueCoding.Utility. The former provides a default implementation of the interface, making it easy to implement NSKeyValueCoding on your own custom classes. The latter is a convenience that allows you to access the properties of NSKeyValueCoding objects and non-NSKeyValueCoding objects using the same code. Both the DefaultImplementation class and the Utility class provide NSKeyValueCoding.ErrorHandling API in addition to basic NSKeyValueCoding API.

Default Implementation

The methods in the NSKeyValueCoding. DefaultImplementation class are just like the methods defined by the NSKeyValueCoding interface (and the NSKeyValueCoding.ErrorHandling interface), except they are static methods and they take an extra argument—the object on which the default implementation should operate.

For example, suppose you want to implement an Employee class that implements NSKeyValueCoding using NSKeyValueCoding. DefaultImplementation. Employee's `valueForKey` method would then look like this:

```
public Object valueForKey(String key) {
    return NSKeyValueCoding.DefaultImplementation.valueForKey(this, key);
}
```

The NSKeyValueCoding. DefaultImplementation methods use accessor methods normally implemented by objects (such as `set`*Key* and *key*), or they access instance variables directly if no accessors for a key exist (see "Directly Accessing Instance Variables"). For detailed information, see the `takeValueForKey` and `valueForKey` method descriptions, which describe the default behavior provided by NSKeyValueCoding. DefaultImplementation. Additionally, see the method descriptions in the NSKeyValueCoding.ErrorHandling interface specification to see how the DefaultImplementation class handles errors.

Note: Always use the default implementation of NSKeyValueCoding provided by the foundation package. The default implementations have significant performance optimizations. To benefit from them, implement NSKeyValueCoding on a custom class as shown above by using the methods in NSKeyValueCoding. DefaultImplementation; or if your class inherits from an WebObjects class that implements NSKeyValueCoding, don't override the inherited implementation. Using a custom implementation incurs significant performance penalties.

Utility

Recall that the NSKeyValueCoding.Utility class is a convenience that allows you to access the properties of NSKeyValueCoding objects and non-NSKeyValueCoding objects using the same code.

Utility's methods are similar to DefaultImplementation's methods in that they are static methods and they take an extra argument—the object on which the method should operate. However, Utility's methods simply check to see if the object on which they operate is an

NSKeyValueCoding object and invoke the corresponding NSKeyValueCoding method on the object if it is. Otherwise, they invoke the corresponding DefaultImplementation method, passing the object on which to operate.

For example, suppose that you want to access an object with the NSKeyValueCoding API but you don't know if the object is an NSKeyValueCoding object. To do so, you simply use the corresponding Utility API, as in the following line of code:

```
theValue = NSKeyValueCoding.Utility.valueForKey(object, value, key);
```

The above line of code is simply a short-cut for the following:

```
if (object instanceof NSKeyValueCoding) {
     theValue = ((NSKeyValueCoding)object).valueForKey(key);
} else {
     theValue = NSKeyValueCoding.DefaultImplementation.valueForKey(object, key);
}
```

Directly Accessing Instance Variables

By default, key-value coding methods directly access instance variables if there are no accessor methods for setting and retrieving values. However, you can modify this behavior without overriding the default implementation. To instruct the default implementation not to directly access instance variables, implement the static method `canAccessFieldsDirectly` on your NSKeyValueCoding class to return `false`.

Constants

NSKeyValueCoding defines the following constant:

Constant	Type	Description
`NullValue`	NSKeyValueCoding.Null	A shared instance of NSKeyValueCoding.Null.

Static Methods (in Java) or Class Methods (in ObjC)

canAccessFieldsDirectly

```
public static boolean canAccessFieldsDirectly()
```

Returns `true` if the key-value coding methods can access the corresponding field value directly on finding no accessor method for a property. Returns `false` if they shouldn't.

An NSKeyValueCoding class doesn't have to implement this method. It's an optional method that allows a class to tailor key-value coding behavior. By default, the key-value implementation provided by NSKeyValueCoding. DefaultImplementation accesses fields directly if it can't find corresponding accessor methods. An NSKeyValueCoding class can override this behavior by implementing this method to return `false`, in which case the key-value coding methods don't access fields directly.

This method isn't strictly part of this interface because static methods can't be formally declared in an interface. However, this method is so closely related to the interface as to be considered part of it.

Instance Methods

takeValueForKey

```
public void takeValueForKey(
    Object value,
    String key)
```

Sets the receiver's value for the property identified by `key` to `value`.

The default implementation provided by NSKeyValueCoding. DefaultImplementation works as follows:

1. Searches for a public accessor method of the form `set`*Key*, and invokes it if there is one.

2. If a public accessor method isn't found, searches for a private accessor method of the form _setKey, and invokes it if there is one.
3. If an accessor method isn't found and the static method canAccessFieldsDirectly returns true, searches for an instance variable based on key and sets its value directly. For the key "lastName", this would be _lastName or lastName. (See "Directly Accessing Instance Variables".)
4. If neither an accessor method nor an instance variable is found, it's an error condition. It invokes handleTakeValueForUnboundKey if the object implements NSKeyValueCoding.ErrorHandling or throws NSKeyValueCoding. UnknownKeyException if the object doesn't.

Note: Always use the default implementation of NSKeyValueCoding provided by the foundation package. The default implementations have significant performance optimizations. To benefit from them, implement NSKeyValueCoding on a custom class as shown above by using the methods in NSKeyValueCoding. DefaultImplementation; or if your class inherits from an WebObjects class that implements NSKeyValueCoding, don't override the inherited implementation. Using a custom implementation incurs significant performance penalties.

valueForKey

```
public Object valueForKey(String key)
```

Returns the receiver's value for the property identified by key.

The default implementation provided by NSKeyValueCoding. DefaultImplementation works as follows:

1. Searches for a public accessor method based on key. For example, with a key of "lastName", the method looks for a method named getLastName or lastName.
2. If a public accessor method isn't found, searches for a private accessor method based on key (a method preceded by an underbar). For example, with a key of "lastName", the method looks for a method named _getLastName or _lastName.
3. If an accessor method isn't found and the static method canAccessFieldsDirectly returns true, the method searches for an instance variable based on key and returns its value directly. For the key "lastName", this would be _lastName or lastName. (See "Directly Accessing Instance Variables".)

4. If neither an accessor method nor an instance variable is found, the method invokes `handleQueryWithUnboundKey` (defined in NSKeyValueCoding.ErrorHandling).

Note: Always use the default implementation of NSKeyValueCoding provided by the foundation package. The default implementations have significant performance optimizations. To benefit from them, implement NSKeyValueCoding on a custom class as shown above by using the methods in NSKeyValueCoding. DefaultImplementation; or if your class inherits from an WebObjects class that implements NSKeyValueCoding, don't override the inherited implementation. Using a custom implementation incurs significant performance penalties.

NSKeyValueCoding.ErrorHandling

Package: com.webobjects.foundation

Interface Description

The NSKeyValueCoding.ErrorHandling interface declares an API for handling errors that occur during key-value coding. For more information, see the NSKeyValueCoding interface specification.

Instance Methods

handleQueryWithUnboundKey

```
public Object handleQueryWithUnboundKey(String key)
```

Invoked from `valueForKey` when it finds no property binding for `key`. The default implementation (see the NSKeyValueCoding. DefaultImplementation class specification) throws an NSKeyValueCoding.UnknownKeyException, with the target object (`TargetObjectUserInfoKey`) and key (`UnknownUserInfoKey`) in the user info. An NSKeyValueCoding.ErrorHandling class can override this method to handle the query in some other way. The method can return a value, in which case that value is returned by the corresponding `valueForKey` invocation.

handleTakeValueForUnboundKey

```
public void handleTakeValueForUnboundKey(
    Object value,
    String key)
```

Invoked from `takeValueForKey` when it finds no property binding for `key`. The default implementation (see the NSKeyValueCoding. DefaultImplementation class specification) throws an NSKeyValueCoding.UnknownKeyException, with the target object (`TargetObjectUserInfoKey`) and key (`UnknownUserInfoKey`) in the user info.

unableToSetNullForKey

```
public void unableToSetNullForKey(String key)
```

Invoked from `takeValueForKey` when it's given a `null` value for a scalar property (such as an `int` or a `float`). The default implementation (see the NSKeyValueCoding. DefaultImplementation class specification) throws an IllegalArgumentException. You might want to implement the method (or override the inherited implementation) to handle the request in some other way, such as by substituting zero or a sentinel value and invoking `takeValueForKey` again.

NSKeyValueCodingAdditions

Implements: NSKeyValueCoding

Package: com.webobjects.foundation

Interface Description

The NSKeyValueCodingAdditions interface defines an extension to the basic NSKeyValueCoding interface. The pair of methods in NSKeyValueCodingAdditions—`takeValueForKeyPath` and `valueForKeyPath`—give access to properties across relationships with **key paths** of the form `relationship.property`; for example, "department.name". For more information on the basic key-value coding, see the NSKeyValueCoding interface specification.

The NSKeyValueCodingAdditions interface contains two inner classes, NSKeyValueCodingAdditions. DefaultImplementation and NSKeyValueCodingAdditions.Utility. The former provides a default implementation of the interface, making it easy to implement on your own custom classes. The latter is a convenience that allows you to access the properties of NSKeyValueCodingAdditions objects and non-`NSKeyValueCodingAdditions` objects using the same code.

Default Implementation

The methods in the NSKeyValueCodingAdditions. DefaultImplementation class are just like the methods defined by the NSKeyValueCodingAdditions interface, except they are static methods and they take an extra argument—the object on which the default implementation should operate.

For example, suppose you want to implement an Employee class that implements NSKeyValueCodingAdditions using NSKeyValueCodingAdditions. DefaultImplementation. Employee's `valueForKeyPath` method would then look like this:

```
public Object valueForKeyPath(String keyPath) {
     return NSKeyValueCodingAdditions.DefaultImplementation.valueForKeyPath(
          this,
          keyPath);
}
```

Utility

Recall that the NSKeyValueCodingAdditions.Utility class is a convenience that allows you to access the properties of `NSKeyValueCodingAdditions` objects and non-NSKeyValueCodingAdditions objects using the same code.

Utility's methods are similar to DefaultImplementation's methods in that they are static methods and they take an extra argument—the object on which the method should operate. However, Utility's methods simply check to see if the object on which they operate is an NSKeyValueCodingAdditions object and invoke the corresponding NSKeyValueCodingAdditions method on the object if it is. Otherwise, they invoke the corresponding DefaultImplementation method, passing the object on which to operate.

For example, suppose that you want to access an object with the NSKeyValueCodingAdditions API but you don't know if the object is an NSKeyValueCodingAdditions object. To do so, you simply use the corresponding Utility API, as in the following line of code:

```
theValue = NSKeyValueCodingAdditions.Utility.valueForKeyPath(object, keyPath);
```

The above line of code is essentially a short-cut for the following:

```
if (object instanceof NSKeyValueCodingAdditions) {
     theValue = ((NSKeyValueCodingAdditions)object).valueForKeyPath(keyPath);
} else {
     theValue = NSKeyValueCodingAdditions.DefaultImplementation.valueForKeyPath(
               object, keyPath);
}
```

Constants

NSKeyValueCodingAdditions defines the following constant:

Constant	Type	Description
KeyPathSeparator	String	The string used to separate components of a key path—a period (.).

Instance Methods

takeValueForKeyPath

```
public void takeValueForKeyPath(
    Object value,
    String keyPath)
```

Sets the value for the property identified by keyPath to value. A key path has the form relationship.property (with one or more relationships); for example “movieRole.roleName” or “movieRole.talent.lastName”. The default implementation of this method (provided by NSKeyValueCodingAdditions. DefaultImplementation) gets the destination object for each relationship using valueForKey, and sends the final object a takeValueForKey message with value and property.

valueForKeyPath

```
public Object valueForKeyPath(String keyPath)
```

Returns the value for the derived property identified by keyPath. A key path has the form relationship.property (with one or more relationships); for example “movieRole.roleName” or “movieRole.talent.lastName”. The default implementation of this method (provided by NSKeyValueCodingAdditions. DefaultImplementation) gets the destination object for each relationship using valueForKey, and returns the result of a valueForKey message to the final object.

NSLocking

Package: com.webobjects.foundation

Interface Description

The NSLocking protocol declares the elementary methods adopted by classes that define lock objects. A lock object is used to coordinate the actions of multiple threads of execution within a single application. By using a lock object, an application can protect critical sections of code from being executed simultaneously by separate threads, thus protecting shared data and other shared resources from corruption.

For example, consider a multithreaded application in which each thread updates a shared counter. If two threads simultaneously fetch the current value into local storage, increment it, and then write the value back, the counter will be incremented only once, losing one thread's contribution. However, if the code that manipulates the shared data (the critical section of code) must be locked before being executed, only one thread at a time can perform the updating operation, and collisions are prevented.

A lock object is either locked or unlocked. You acquire a lock by sending the object a lock message, thus putting the object in the locked state. You relinquish a lock by sending an unlock message, and thus putting the object in the unlocked state. (The Foundation classes that adopt this protocol define additional ways to acquire and relinquish locks.)

The lock method as declared in this protocol is blocking. That is, the thread that sends a lock message is blocked from further execution until the lock is acquired (presumably because some other thread relinquishes its lock). Classes that adopt this protocol typically add methods that offer nonblocking alternatives.

These Foundation classes conform to the NSLocking protocol:

Class	Adds these features to the basic protocol
NSLock	A nonblocking lock method; the ability to limit the duration of a locking attempt.
NSMultiReaderLock	The ability for multiple threads to acquire the lock for reading while allowing only a single thread to acquire the lock for writing.
NSRecursiveLock	The ability for a single thread to acquire a lock more than once without deadlocking.

These classes use a locking mechanism that causes a thread to sleep while waiting to acquire a lock rather than to poll the system constantly. Thus, lock objects can be used to lock time-consuming operations without causing system performance to degrade. See the class specifications for these classes for further information on their behavior and usage.

Java also has a locking mechanism, which is based on the `synchronized` keyword. In certain cases, you may want to use the Foundation locking classes instead:

- The Foundation classes have extra features that are not supported by the `synchronized` locking mechanism like nonblocking lock methods.
- Foundation locking objects can coordinate the locking of many objects at the same time, unlike the `synchronized` locking mechanism, which can lock only a single method, class, or instance at a time.
- These locking classes were available in previous versions of WebObjects. You may encounter these classes if you are porting older applications to this version of WebObjects.

Constants

NSLocking defines the following constants to simplify locking method invocations that take time intervals as parameters.

Constant	Type	Description
`OneSecond`	`long`	Number of milliseconds in one second
`OneMinute`	`long`	Number of milliseconds in one minute
`OneHour`	`long`	Number of milliseconds in one hour

Constant	Type	Description
OneDay	long	Number of milliseconds in one day
OneWeek	long	Number of milliseconds in one week
OneYear	long	Number of milliseconds in one year (defined as 365.2425 days)
OneCentury	long	Number of milliseconds in one century

Method Types

All methods

lock

unlock

Instance Methods

lock

```
public void lock()
```

Attempts to acquire a lock. This method blocks a thread's execution until the lock can be acquired.

An application protects a critical section of code by requiring a thread to acquire a lock before executing the code. Once the critical section is past, the thread relinquishes the lock by invoking unlock.

unlock

```
public void unlock()
```

Relinquishes a previously acquired lock.

NSValidation

Package: com.webobjects.foundation

Interface Description

The NSValidation interface defines a validation mechanism in which the properties of an object are validated indirectly by name (or key), rather than directly through invocation of an specific validation method. Thus, all of an object's properties can be validated in a consistent manner. The `validateValueForKey` method confirms that a value is legal for a particular property, and `validateTakeValueForKeyPath` performs the validation and assigns the value if it's legal and different from the current value.

The NSValidation interface contains two inner classes, NSValidation.DefaultImplementation and NSValidation.Utility. The former provides a default implementation of the interface, making it easy to implement on your own custom classes. The latter is a convenience that allows you to access the properties of NSValidation objects and non-NSValidation objects using the same code.

Default Implementation

The methods in the NSValidation.DefaultImplementation class are just like the methods defined by the NSValidation interface, except they are static methods and they take an extra argument—the object on which the default implementation should operate.

For example, suppose you want to implement an Employee class that implements NSValidation using NSValidation.DefaultImplementation. Employee's `validateValueForKey` method would then look like this:

```
public Object validateValueForKey(Object value, String key) {
     return NSValidation.DefaultImplementation.validateValueForKey(this, value, key);
}
```

The NSValidation.DefaultImplementation methods search for property specific methods of the form `validateKey` and invoke them if they exist. Thus an NSValidation class should implement a validate method for each property that has associated validation logic. For example, a `validateAge` method could check that the value a user enters as an age is within acceptable limits and throw an NSValidation.ValidationException if it finds an unacceptable value. For a more information on the `validateKey` methods, see "Writing validateKey Methods" (page 375).

Because you implement custom validation logic in the `validateKey` methods, you rarely need to implement the NSValidation methods from scratch. Rather, the default implementation provided by NSValidation.DefaultImplementation is generally sufficient.

Note: Always use the default implementation of NSValidation provided by the foundation package. The default implementations have significant performance optimizations. To benefit from them, implement NSValidation on a custom class as shown above by using the methods in NSValidation.DefaultImplementation; or if your class inherits from an WebObjects class that implements NSValidation, don't override the inherited implementation. Using a custom implementation incurs significant performance penalties.

Utility

Recall that the NSValidation.Utility class is a convenience that allows you to access the properties of NSValidation objects and non-NSValidation objects using the same code.

Utility's methods are similar to DefaultImplementation's methods in that they are static methods and they take an extra argument—the object on which the method should operate. However, Utility's methods simply check to see if the object on which they operate is an NSValidation object and invoke the corresponding NSValidation method on the object if it is. Otherwise, they invoke the corresponding DefaultImplementation method, passing the object on which to operate.

For example, suppose that you want to access an object with the NSValidation API but you don't know if the object is an NSValidation object. To do so, you simply use the corresponding Utility API, as in the following line of code:

```
theValue = NSValidation.Utility.validateValueForKey(object, value, key);
```

The above line of code is simply a short-cut for the following:

```
if (object instanceof NSValidation) {
     theValue = ((NSValidation)object).validateValueForKey(key);
} else {
     theValue = validateValueForKey.DefaultImplementation.validateValueForKey(
               object, value, key);
}
```

Writing validateKey Methods

To implement validation logic in an NSValidation class, you create a `validateKey` method for each property needing validation. The default implementations of NSValidation's methods look for these `validateKey` methods and use them to perform the actual validation.

A class's `validateKey` methods should have the following form:

```
public Object validateKey(Object aValue) throws NSValidation.ValidationException
```

The implementation should confirm that the value passed in is legal and throw an NSValidation.ValidationException if it's not. It should also **coerce** the argument to the proper type, if necessary. Note that a `validateKey` method's argument type is Object, and not specifically the class of the corresponding property (String, Integer, or NSTimestamp, for example). Thus, a `validateKey` method needs to check the type of the argument and convert it to a different type if necessary. The return type of a `validateKey` method doesn't have to be Object. In fact, it's a good idea to specify the class of the value the method returns, which is generally the class of the corresponding property. The argument type, on the other hand, should generally be Object. For more information, see "The validateKey Argument Type" (page 376).

The following `validateAge` method is an example that validates values for a property named `age` that's stored as an Integer. The method handles arguments of type String and Number. If the argument is an instance of any other class, the method throws a NSValidation.ValidationException.

```
public Number validateAge(Object aValue) throws NSValidation.ValidationException {
     Integer numberValue;
     int age;

     if (aValue instanceof String) {
          // Convert the String to an Integer.
          try {
```

```
                numberValue = new Integer((String)aValue);
            } catch (NumberFormatException numberFormatException) {
            throw new NSValidation.ValidationException(
                "Validation exception: Unable to convert the String " + aValue
                + " to an Integer");
            }
        } else if (aValue instanceof Number) {
            numberValue = new Integer(((Number)aValue).intValue());
        } else {
            throw new NSValidation.ValidationException
                ("Validation exception: Unable to convert the Object " + aValue
                + "to an Integer");
        }

        age = numberValue.intValue();
        if (age < 16) {
            throw new NSValidation.ValidationException
                ("Age of " + age + " is below minimum.", this, "age");
        }
        return numberValue;
}
```

The `validateAge` method checks the argument's class. If the argument is a String or Number, it creates an Integer from the argument and validates it. If the Integer fails the validation test (if the value is less than 16), the method throws a NSValidation.ValidationException inserting the NSValidation object and the key into the exception's userInfo dictionary by providing them to the constructor. On the other hand, if the Integer passes the validation test, the method returns the Integer.

The code that invokes the validation process is expected to use the value returned from the `validateKey` method instead of the original value it provided. Thus, a `validateKey` method has an opportunity to coerce a value into a type it prefers. As another example of coercion, a `validateKey` method can return `null`. A method might do this, for example, if it is invoked with the empty string as the argument.

The validateKey Argument Type

The argument type of a `validateKey` method doesn't have to be specifically Object; it can be Object or any subclass. However, generally Object is the most appropriate type for a `validateKey` method's argument (the one exception is described later in this section). As explained in

"Writing validateKey Methods" (page 375), a validateKey method should be able to handle any reasonable argument type. You *can* type the argument to the common superclass of all reasonable arguments, but this is frequently Object.

You should not overload a validateKey method for a particular property, creating different validateKey methods for each argument type. Instead you should create one version of the validateKey method for the property that takes Object as its argument (or at least a superclass of all the possible argument types). The method's implementation should test the argument's type and coerce it appropriately.

There's one situation in which the argument type should be more specific than Object. If the property corresponding to the validateKey method is a to-one relationship to an enterprise object, the argument's type should be EOEnterpriseObject.

Instance Methods

validateTakeValueForKeyPath

```
public Object validateTakeValueForKeyPath(
    Object value,
    String keyPath) throws NSValidation.ValidationException
```

Confirms that value is legal for the receiver's property named by keyPath, and assigns the value to the property if it's legal (and if value is different from the current value), or throws an NSValidation.ValidationException if value isn't legal.

A key path has the form relationship.property (with one or more relationships); for example "movieRole.roleName" or "movieRole.talent.lastName".

The default implementation of this method (provided by NSValidation.DefaultImplementation) gets the destination object for each relationship using valueForKey, and sends the final object a validateValueForKey message with value and property.

validateValueForKey

```
public Object validateValueForKey(
    Object value,
    String key) throws NSValidation.ValidationException
```

Confirms that `value` is legal for the receiver's property named by `key`, and returns the validated value if it's legal, or throws an NSValidation.ValidationException if it isn't. Note that the value returned from this method can be different than the one passed as an argument.

The default implementation of this method (provided by NSValidation.DefaultImplementation) checks for a method of the form `validateKey` (for example, `validateBudget` for a key of "budget"). If such a method exists, it invokes it and returns the result. Thus, NSValidation objects can implement individual `validateKey` methods to check limits, test for nonsense values, and coerce values (convert strings to dates or numbers, for example). For more information on `validateKey` methods, see "Writing validateKey Methods" (page 375).