NSArchiver

Inherits From: NSCoder : NSObject

Conforms To: NSObject (NSObject)

Declared In: Foundation/NSArchiver.h

Class at a Glance

Purpose

An NSArchiver encodes objects into a format that can be written to a file. The archiving process traverses a set of interconnected objects, making sure to encode each one only once.

Principal Attributes

- An NSMutableData object containing the encoded data

Creation

– initForWritingWithMutableData:

Commonly Used Methods

+ archiveRootObject:ToFile:	Archives a graph of objects to a file.
+ archivedDataWithRootObject:	Archives a graph of objects into an NSMutableData object.

Class Description

NSArchiver, a concrete subclass of NSCoder, provides a way to encode Objective-C objects into an architecture-independent format that can be stored in a file. When you archive a set of objects, their class

information and the values of their instance variables are written to the archive. NSArchiver's companion class, NSUnarchiver, decodes an archive into a set of objects equivalent to the original set.

NSArchiver implements encoding by placing the archived data in an NSMutableData object. After encoding the objects, you can have the NSArchiver write this NSMutableData immediately to a file, or you can retrieve the encoded data for some other use.

Archiving a Graph of Objects

The easiest way to archive an object is to invoke a single class method—either **archiveRootObject:toFile:** or **archivedDataWithRootObject:**, depending on whether you want the encoded data to be stored in a file immediately. These convenience methods create a temporary NSArchiver and send it an **encodeRootObject:** message—you need do no more. However, if you want to customize the archiving process (for example, by substituting certain classes for others), you must instead create an NSArchiver instance yourself, configure it as desired, and send it an **encodeRootObject:** message explicitly.

The "root object" that you specify as the argument to any of these three methods indicates the starting point for archiving. The NSArchiver commences archiving by invoking the root object's **encodeWithCoder:** method. That method typically encodes the root object's instance variables, which isn't necessarily a straightforward process—the instance variables can themselves be other objects that respond to **encodeWithCoder:**, and so on, yielding a possibly complex graph of objects that need to be archived.

The fact that many objects contain references to other objects poses two problems for archiving. The first is redundancy. An object graph isn't necessarily a simple tree structure. Two objects can contain references to each other, for example, creating a cycle. To address this problem, NSArchiver overrides NSCoder's **encodeRootObject:** method to keep track of all the objects encountered while traversing the graph. If any object is encountered more than once, the multiple references to it are stored, but the object itself is encoded only once.

The second problem is that it's not always appropriate to archive the entire graph. To use an example from the Application Kit, when you archive an NSView as the root object, its subviews should be archived, but not its superview. In this case, the superview is considered an extraneous part of the graph. On the other hand, if you archive the superview as the root object, the NSView should now include a reference to the superview. To solve this dilemma, NSArchiver implements conditional archiving, overriding the minimal **encodeConditionalObject:** method that's inherited from NSCoder. A class's **encodeWithCoder:** method can invoke **encodeConditionalObject:** to archive inessential object instance variables. The NSArchiver doesn't actually archive a conditionally encoded object unless some other object in the graph encodes it unconditionally (using one of the other **encode...Object:** methods declared by NSCoder). When everything is unarchived, all original references to the conditionally encoded object are properly restored as references to the single unarchived object. For example, an NSView encodes its superview with **encodeConditionalObject:**, because it doesn't own the superview but does need to preserve its connection to it if some other object archives the superview.

In contrast, **encodeObject:** unconditionally instructs an object to encode itself. This method is most often used in a class's **encodeWithCoder:** method for instance variables that are intrinsic to the receiver and

essential for proper functioning. An NSView encodes its subviews with **encodeObject:**, because it owns them.

All the objects to be placed in a single archive must be interconnected members of a single graph. In other words, there can only be one root object per archive. The only recommended way to archive objects is to send an NSArchiver a single **encodeRootObject:** message, whether directly, or indirectly by invoking **archiveRootObject:toFile:** or **archivedDataWithRootObject:**. Don't try to add data to the archive by invoking any of NSCoder's other **encode...** methods, except from within the **encodeWithCoder:** method of each object that's part of the graph. (These **encodeWithCoder:** methods are invoked automatically when you encode the root object.)

To extract an object graph from an archive, use the NSUnarchiver class method **unarchiveObjectWithFile:** or **unarchiveObjectWithData:**, assigning the return value to the desired root object.

Archiving other Data Types

It's possible to create an archive that doesn't contain any objects. To archive other data types, invoke **encodeValueOfObjCType:** directly for each data item to be archived, instead of using **setRootObject:**. When you create an archive in this way, the corresponding unarchiving code must follow exactly the same sequence of data types.

This approach shouldn't be used to archive objects. Use **setRootObject:** instead, to avoid the problems mentioned in the previous section and to simplify unarchiving.

An NSSerializer provides another means to store data in an architecture-independent format. See the NSSerializer class specification for more information.

Superclass Methods to Avoid

NSArchiver's superclass, NSCoder, supplies methods for both encoding and decoding. However, only the encoding methods are applicable to NSArchiver—don't send an NSArchiver any **decode...** messages. (Similarly, don't send **encode...** messages to an NSUnarchiver.)

Method Types

Initializing an NSArchiver	– initForWritingWithMutableData:
Archiving data	+ archivedDataWithRootObject: + archiveRootObject:toFile: – encodeRootObject: – encodeConditionalObject:
Getting the archived data	– archiverData

Substituting classes or objects	– classNameEncodedForTrueClassName: – encodeClassName:intoClassName: – replaceObject:withObject:

Class Methods

archiveRootObject:toFile:

+ (BOOL)**archiveRootObject:**(id)*rootObject* **toFile:**(NSString *)*path*

Archives *rootObject* by encoding it into a data object in a temporary NSArchiver and writing that data object to the file *path*. This convenience method invokes **archivedDataWithRootObject:** to get the encoded data, and then sends that data object the message **writeToFile:atomically:**, using *path* for the first argument and YES for the second. Returns YES upon success.

archivedDataWithRootObject:

+ (NSData *)**archivedDataWithRootObject:**(id)*rootObject*

Returns a data object containing the encoded form of the object graph whose root object is *rootObject*. This method invokes **initForWritingWithMutableData:** and **encodeRootObject:** to create a temporary archiver that encodes the object graph.

Instance Methods

archiverData

– (NSMutableData *)**archiverData**

Returns the archived data. The returned data object is the same one that was specified as the argument to **initForWritingWithMutableData:**. It contains whatever data has been encoded thus far by invocations of the various encoding methods. It's safest not to invoke this method until after **encodeRootObject:** has returned. In other words, although it's possible for a class to invoke this method from within its **encodeWithCoder:** method, that method must not alter the data.

classNameEncodedForTrueClassName:

– (NSString *)**classNameEncodedForTrueClassName:**(NSString *)*trueName*

Returns the class name used to archive instances of the class *trueName*.

See also: – **encodeClassName:intoClassName:**

encodeClassName:intoClassName:

– (void)**encodeClassName:**(NSString *)*trueName* **intoClassName:**(NSString *)*inArchiveName*

Encodes in the archive a substitute name for the class name *trueName*. Any subsequently encountered objects of class *trueName* will be archived as instances of class *inArchiveName*. It's safest not to invoke this method during the archiving process (that is, within an **encodeWithCoder:** method). Instead, invoke it before **encodeRootObject:**.

See also: – **classNameEncodedForTrueClassName:**

encodeConditionalObject:

– (void)**encodeConditionalObject:**(id)*object*

Archives *object* conditionally. This method overrides the superclass implementation to allow *object* to be encoded only if it's also encoded unconditionally by another object in the object graph. Conditional encoding lets you encode one part of a graph detached from the rest. (See the class description for more information.)

This method should be invoked only from within an **encodeWithCoder:** method. If *object* is **nil**, the NSArchiver encodes it unconditionally as **nil**. Raises an NSInvalidArgumentException if no root object has been encoded.

encodeRootObject:

– (void)**encodeRootObject:**(id)*rootObject*

Archives *rootObject* along with all the objects it's connected to. If any object is encountered more than once while traversing the graph, it's encoded only once, but the multiple references to it are stored. (See the discussion of object graphs in the class description.)

This message mustn't be sent more than once to a given NSArchiver; an NSInvalidArgumentException is raised if a root object has already been encoded. Therefore, don't attempt to reuse an NSArchiver; instead, create a new one. To encode multiple object graphs, use distinct NSArchivers.

initForWritingWithMutableData:

– (id)**initForWritingWithMutableData:**(NSMutableData *)*data*

Initializes an archiver, encoding stream and version information into *data*. Raises an NSInvalidArgumentException if *data* is **nil**.

See also: – **archiverData**

NS+ replaceObject:withObject:

– (void)**replaceObject:**(id)*object* **withObject:**(id)*newObject*

Causes the NSArchiver to treat subsequent requests to encode *object* as though they were requests to encode *newObject*.