NSCoding

Adopted By: NSObject

Declared In: Foundation/NSObject.h

Protocol Description

The NSCoding protocol declares the two 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 and NSArchiver class specifications 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 **initWithCoder:**. **encodeWithCoder:** instructs the object to encode its instance variables to the coder provided; an object can receive this method any number of times. **initWithCoder:** instructs the object to initialize itself from data in the coder provided; as such, it replaces any other initialization method and is only sent once per object. Any object class that should be codable must adopt the NSCoding protocol and implement its methods.

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 protocol. 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, with **encodeConditionalObject:**, as described in the NSArchiver class specification).

For example, a fictitious MapView class that displays a legend and a map at various magnifications might implement **encodeWithCoder:** like this:

```
- (void)encodeWithCoder:(NSCoder *)coder
{
    [super encodeWithCoder:coder];
    [coder encodeValueOfObjCType:@encode(NSString) at:&mapName];
    [coder encodeValueOfObjCType:@encode(unsigned int) at:&magnification];
    [coder encodeObject:legendView];
    [coder encodeConditionalObject:auxiliaryView];
    return;
}
```

The **@encode()** compiler directive generates an Objective-C type code from a type expression. See *Object-Oriented Programming and the Objective-C Language* for more information.

Similarly, in **initWithCoder:** the object should first send a message to **super** (if appropriate) to initialize inherited instance variables, and then it should decode and initialize its own. MapView's implementation of **initWithCoder:** might look like this:

```
- (id)initWithCoder:(NSCoder *)coder
{
    self = [super initWithCoder:coder];
    [coder decodeValueOfObjCType:@encode(NSString) at:&mapName];
    [coder decodeValueOfObjCType:@encode(unsigned int) at:&magnification];
    legendView = [[coder decodeObject] retain];
    auxiliaryView = [[coder decodeObject] retain];
    return self;
}
```

Note the assignment of the return value of **initWithCoder:** to **self** in the example above. This is done in the subclass because the superclass, in its implementation of **initWithCoder:**, may decide to return a object other than itself.

Making Substitutions During Coding

During encoding or decoding a coder object invokes methods that allow the object being coded to substitute a replacement class or instance for itself. This allows archives to be shared among implementations with different class hierarchies or simply different versions of a class (for example, class clusters take advantage of this feature). It also allows classes that should maintain unique instances to enforce this policy on decoding (for example, there need only be a single NSFont instance for a given typeface and size).

Substitution methods are declared by NSObject, and come in two flavors: generic and specialized. The generic methods are these:

Method	Typical Use
classForCoder	Allows an object, before being encoded, to substitute a class other than its own. For example, the private subclasses of a class cluster substitute the name of their public superclass when being archived.
replacementObjectForCoder:	Allows an object, before being encoded, to substitute another instance in its place.
awakeAfterUsingCoder:	Allows an object, after being decoded, to substitute another object for itself. For example, an object that represents a font might, upon being decoded, release itself and return an existing object having the same font description as itself. In this way, redundant objects can be eliminated.

The specialized substitution methods are analogous to **classForCoder** and **replacementObjectForCoder:**, but they're designed for (and invoked by) a specific, concrete coder subclass. NSArchiver invokes **classForArchiver:** and **replacementObjectForArchiver:**, while NSPortCoder invokes **classForPortCoder**

and **replacementObjectForPortCoder:**. (There isn't a specialized version of **awakeAfterUsingCoder:**.) By implementing these specialized methods, your class can base its coding behavior on the specific coder class being used. For more information on these methods, see their method descriptions in the NSObject class specification, as well as the class description in the NSPortCoder class specification.

Method Types

Encoding and decoding objects	– encodeWithCoder: – initWithCoder:

Instance Methods

encodeWithCoder:

– (void)**encodeWithCoder:**(NSCoder *)*encoder*

Encodes the receiver using *encoder.*

initWithCoder:

– (id)**initWithCoder:**(NSCoder *)*decoder*

Initializes a newly allocated instance from data in *decoder.* Returns **self**.