# EOCustomClassArchiving

**(informal protocol)**

**Category Of:** NSObject

**Declared In:** EOAccess/EOAttribute.h

## Category Description

EOCustomClassArchiving defines methods that can be used to write any object that conforms to NSCoding to the database as binary data, as generated by NSArchiver. Since data in this fomat is neither human-readable nor readable by non-OpenStep applications, it's usually preferable to supply other custom archiving methods for your custom value classes. For a comprehensive discussion of working with custom data types, see the EOAttribute class specification and the chapter "Advanced Enterprise Object Modeling" in the *Enterprise Objects Framework Developer's Guide*.

When you create an attribute with a custom type in EOModeler, it by default specifies the factory (or class creation) method as **objectWithArchiveData:**, and the conversion (or data extraction) method as **archiveData**. These methods operate in terms of NSData objects, using the Foundation Framework's archiving classes to translate between data and objects. If your custom value class adopts the NSCoding protocol, the default implementations of these methods will work as they are. Otherwise, you need to implement these methods directly or define your own factory and conversion methods. For information on archiving, see the Foundation Framework specifications for the NSCoding protocol and the NSCoder, NSArchiver, and NSUnarchiver classes.

### Custom Value Methods and Argument Types

An EOAttribute records its internal and external types, and for an internal type that's a custom value class, it also records the names of the factory and conversion methods to use for that class, along with the type to pass to the factory method. These are by default **objectWithArchiveData:** and **archiveData**, but you'd be more likely to implement your own custom methods. You normally specify all this in EOModeler, but you can also do so programmatically with the EOAttribute methods **setValueFactoryMethodName:**, **setFactoryMethodArgumentType:**, and **setAdaptorValueConversionMethodName:**. If an EOAttribute isn't mapped to a custom class, it uses NSData objects for binary columns and NSString objects for string or character columns.

If an EOAttribute represents a binary column in the database, the factory method argument type can be either EOFactoryMethodArgumentIsNSData or EOFactoryMethodArgumentIsBytes, indicating that the method takes an NSData object or raw bytes as an argument. If the EOAttribute represents a string or character column, the factory method argument type can be either EOFactoryMethodArgumentIsNSString or EOFactoryMethodArgumentIsBytes, indicating that the method takes an NSString object or raw bytes as an argument. These types apply when fetching custom values. For more discussion of this topic, see the EOAttribute class specification.

## Class Methods

### objectWithArchiveData:

+ (id)**objectWithArchiveData:**(NSData *)*data*

Returns an object created from *data*. NSObject's implementation of this method invokes NSUnarchiver's **unarchiveObjectWithData:** method and returns the result. Your custom value class can therefore take advantage of this method merely by implementing the NSCoding protocol method **initWithCoder:**.

**See also:** – **archiveData**

## Instance Methods

### archiveData

– (NSData *)**archiveData**

Return the receiver's value as an NSData object whose bytes can be stored in an external repository. NSObject's implementation of this method invokes NSArchiver's **archivedDataWithRootObject:** method and returns the result. Your custom value class can therefore take advantage of this method merely by implementing the NSCoding protocol method **encodeWithCoder:**.

**See also:** + **objectWithArchiveData:**