

# NSSerializer

<b>Inherits From:</b>	NSObject
<b>Conforms To:</b>	NSObject (NSObject)
<b>Declared In:</b>	Foundation/NSSerialization.h

## Class Description

The `NSSerializer` class provides a mechanism for creating an abstract representation of a property list. (In OpenStep, property lists are defined to be—and to contain—objects of these classes: `NSDictionary`, `NSArray`, `NSString`, `NSData`). The `NSSerializer` class stores this representation in an `NSData` object in an architecture-independent format, so that property lists can be used with distributed applications. `NSSerializer`'s companion class `NSDeserializer` declares methods that take the abstract representation and recreate the property list in memory.

In contrast to archiving (see the `NSArchiver` class specification), the serialization process preserves only structural information, not class information. Thus, if a property list is serialized and then deserialized, the objects in the resulting property list might not be of the same class as the objects in the original property list. However, the structure and interrelationships of the data in the resulting property list are identical to that in the original, with one possible exception.

The exception is that when an object graph is serialized, the mutability of the containers objects (`NSDictionary` and `NSArray` objects) is preserved only down to the highest node in the graph that has an immutable container. Thus, if an `NSArray` contains an `NSMutableDictionary`, the serialized version of this object graph would not preserve the mutability of the dictionary or any of the mutable objects it contained. Since serialization doesn't preserve class information or—in some cases—mutability, coding (as implemented by `NSCoder` and `NSArchiver`) is the preferred way to make object graphs persistent.

The `NSSerializer` class object provides the interface to the serialization process; you don't create instances of `NSSerializer`. You might subclass `NSSerializer` to modify the representation it creates, for example, to encrypt the data or add authentication information.

Other types of data besides property lists can be serialized using methods declared by the `NSData` and `NSMutableData` classes (see **`serializeDataAt:ofObjCType:context:`** and **`deserializeDataAt:ofObjCType:atCursor:context:`**), allowing these types to be represented in an architecture-independent format. Furthermore, the `NSObjCTypeSerializationCallback` protocol allows you to serialize and deserialize objects that aren't property lists.

## Class Methods

### **serializePropertyList:**

+ (NSData \*)**serializePropertyList:(id)***aPropertyList*

Creates a data object, serializes *aPropertyList* into it, and returns the data object. *aPropertyList* must be a kind of NSData, NSString, NSArray, or NSDictionary.

### **serializePropertyList:intoData:**

+ (void)**serializePropertyList:(id)***aPropertyList intoData:(NSMutableData \*)**mdata*

Serializes the property list *aPropertyList* in the mutable data object *mdata* . *aPropertyList* must be a kind of NSData, NSString, NSArray, or NSDictionary.