NSValue

Inherits From:	NSObject
Conforms To:	NSCoding NSCopying NSObject (NSObject)
Declared In:	Foundation/NSValue.h Foundation/NSGeometry.h

Class at a Glance

Purpose

An NSValue object serves as an object wrapper for a standard C or Objective-C data item, allowing it to be stored in a collection object such as an NSArray or NSDictionary.

Creation

+ value:withObjCType:	Returns an NSValue containing any C or Objective-C data item.
+ valueWithBytes:objCType:	Returns an NSValue containing any Objective-C data item, which is interpreted as being of the specified Objective-C type.
+ valueWithNonretainedObject:	Returns an NSValue containing an Objective-C object, without retaining the Objective-C object.
+ valueWithPointer:	Returns an NSValue that contains a pointer.

Commonly Used Methods

– objCType	Returns the Objective-C type for the data contained in an NSValue.
– getValue:	Copies an NSValue's contents into a buffer.
– nonretainedObjectValue	Returns an NSValue's contents as an **id**.
– pointerValue	Returns an NSValue's contents as a pointer to **void**.

Primitive Methods

– getValue:
– objCType

lass Description

An NSValue object is a simple container for a single C or Objective-C data item. It can hold any of the scalar types such as **int**, **float**, and **char**, as well as pointers, structures, and object **id**s. The purpose of this class is to allow items of such data types to be added to collection objects such as NSArrays and NSSets, which require their elements to be objects.

To create an NSValue object with a particular data item, you provide a pointer to the item along with a C string describing the item's type in Objective-C type encoding. You get this string using the **@encode()** compiler directive, which returns the platform-specific encoding for the given type. Fore example, this code excerpt creates **theValue** containing an NSRange:

```
NSRange myRange = {4, 10};
NSValue *theValue = [NSValue valueWithBytes:&myRange
    objCType:@encode(NSRange)];
```

Note that the type you specify must be of constant length. C strings, variable-length arrays and structures, and other data types of indeterminate length can't be stored in an NSValue. You should use NSString or NSData objects for these. If you must store a variable-length item in an NSValue, you have to store a pointer to the item, not the item itself. This code excerpt incorrectly attempts to place a C string directly into an NSValue object:

```
/* INCORRECT! */
char *myCString = "This is a string.";
NSValue *theValue = [NSValue value:myCString
    withObjCType:@encode(char *)];
```

In this code excerpt the *contents* of **myCString** are interpreted as a pointer to a **char**, so that the first four bytes contained in the string are treated as a pointer (the actual number of bytes used may vary with the hardware architecture). That is, the sequence "This" is interpreted as a pointer value, which is unlikely to be a legal address. The correct way to store such a data item, short of using an NSString object, is to pass the address of its pointer, not the pointer itself:

```
/* Correct. */
char *myCString = "This is a string.";
NSValue *theValue = [NSValue value:&myCString
    withObjCType:@encode(char *)];
```

Here the address of **myCString** is passed, so that the address of the first character of the string is stored in **theValue**. Note that the NSValue doesn't copy the contents of the string, but the pointer itself. If you create an NSValue with an allocated data item, don't deallocate its memory while the NSValue object exists.

dopted Protocols

NSCoding	– encodeWithCoder:
	– initWithCoder:

NSCopying	– copyWithZone:

Method Types

Creating an NSValue	– initWithBytes:objCType: + valueWithBytes:objCType: + value:withObjCType: + valueWithNonretainedObject: + valueWithPointer: + valueWithPoint: + valueWithRect: + valueWithSize:
Accessing data	– getValue: – nonretainedObjectValue – objCType – pointValue – pointerValue – rectValue – sizeValue
Comparing objects	– isEqual:

Class Methods

valueWithBytes:objCType:

+ (NSValue *)**valueWithBytes:**(const void *)*value* **objCType:**(const char *)*type*

Creates and returns an NSValue containing *value*, which is interpreted as being of the Objective-C type *type*. *type* should be created with the Objective-C **@encode()** compiler directive; it shouldn't be hard-coded as a C string. This method is equivalent to **value:withObjCType:**, which is part of OpenStep. See the class description for other considerations in creating an NSValue object and code examples.

See also: – **initWithBytes:objCType:**

value:withObjCType:

+ (NSValue *)**value:**(const void *)*value* **withObjCType:**(const char *)*type*

Creates and returns an NSValue containing *value*, which is interpreted as being of the Objective-C type *type*. *type* should be created with the Objective-C **@encode()** compiler directive; it shouldn't be hard-coded as a

C string. See the class description for other considerations in creating an NSValue object and code examples.

See also: **+ valueWithBytes:objCType:**

valueWithNonretainedObject:

+ (NSValue *)**valueWithNonretainedObject:**(id)*anObject*

Creates and returns an NSValue containing *anObject*, but doesn't retain it. This method is equivalent to invoking **value:withObjCType:** in this manner:

```
NSValue *theValue = [NSValue value:&anObject
    withObjCType:@encode(void *)];
```

This method is useful for preventing an object from being retained when it's added to a collection object (such as an NSArray or NSDictionary).

See also: – **nonretainedObjectValue**

valueWithPoint:

+ (NSValue *)**valueWithPoint:**(NSPoint)*Point*

Creates and returns a value object that contains the specified NSPoint structure (which represents a geometrical point in two dimensions).

See also: – **pointValue**

valueWithPointer:

+ (NSValue *)**valueWithPointer:**(const void *)*aPointer*

Creates and returns an NSValue object that contains *aPointer*. This method is equivalent to invoking **value:withObjCType:** in this manner:

```
NSValue *theValue = [NSValue value:&aPointer
    withObjCType:@encode(void *)];
```

This method doesn't copy the contents of *aPointer*, so you should be sure not to deallocate that memory while the NSValue object exists. NSData objects may be more suited for arbitrary pointers than NSValue objects.

See also: – **pointerValue**

valueWithRect:

+ (NSValue *)**valueWithRect:**(NSRect)*rect*

Creates and returns a value object that contains the specified NSRect structure (which represents the coordinates of the rectangle's origin).

See also: – **rectValue**

valueWithSize:

+ (NSValue *)**valueWithPointer:**(NSSize)*size*

Creates and returns an NSValue that contains the specified NSSize structure (which represents the width and height of a rectangle).

See also: – **sizeValue**

nstance Methods

getValue:

– (void)**getValue:**(void *)*buffer*

Copies the NSValue's contents into *buffer*. *buffer* should be large enough to hold the value.

initWithBytes:objCType:

– (id)**initWithBytes:**(const void *)*value* **objCType:**(const char *)*type*

Initializes a newly created NSValue to contain *value*, which is interpreted as being of the Objective-C type *type*. *type* should be created with the Objective-C **@encode()** compiler directive; it shouldn't be hard-coded as a C string. See the class description for other considerations in creating an NSValue object.

This is the designated initializer for the NSValue class. Returns **self**.

isEqual:

@protocol NSObject
– (BOOL)**isEqual:**(id)*anObject*

Returns YES if the receiver is equal to *anObject*, otherwise returns NO. For an NSValue, the class, type, and contents are compared to determine equality.

nonretainedObjectValue

– (id)**nonretainedObjectValue**

For an NSValue object created to hold a pointer-sized data item, returns that item as an **id.** For any other NSValue the result is undefined.

See also: – **getValue:**

objCType

– (const char *)**objCType**

Returns a C string containing the Objective-C type of the data contained in the NSValue object, as encoded by the **@encode()** compiler directive**.**

pointValue:

– (NSPoint)**pointValue**

Returns an NSPoint structure (which represents a geometrical point in two dimensions).

See also: – **rectValue,** – **sizeValue**

pointerValue

– (void *)**pointerValue**

For an NSValue object created to hold a pointer-sized data item, returns that item as a pointer to **void**. For any other NSValue the result is undefined.

See also: – **getValue:**

rectValue

– (NSRect)**rectValue**

Returns an NSRect structure (which represents the coordinates of the rectangle’s origin).

See also: – **pointValue,** – **sizeValue**

valueWithSize:

+ (NSValue *)**valueWithPointer:**(NSSize)*size*

Returns an NSSize structure (which represents the width and height of a rectangle).

See also: **– pointValue, – rectValue**