Foundation Functions Reference

Developer

Contents

Foundation Functions Reference 6
Overview 6
Functions by Task 6
- Assertions 6
- Bundles 7
- Byte Ordering 8
- Decimals 10
- Exception Handling 11
- Managing Object Allocation and Deallocation 11
- Interacting with the Objective-C Runtime 12
- Logging Output 12
- Managing File Paths 12
- Managing Ranges 13
- Uncaught Exception Handlers 13
- Core Foundation ARC Integration 14
- Managing Memory 14
- Managing Zones 14

Functions 16
- CFBridgingRelease 16
- CFBridgingRetain 16
- NSAllocateMemoryPages 17
- NSAllocateObject 18
- NSAssert 18
- NSAssert1 20
- NSAssert2 21
- NSAssert3 23
- NSAssert4 24
- NSAssert5 26
- NSCAssert 27
- NSCAssert1 29
- NSCAssert2 30
- NSCAssert3 31
- NSCAssert4 32
- NSCAssert5 33

NSClassFromString 34
NSConvertHostDoubleToSwapped 35
NSConvertHostFloatToSwapped 36
NSConvertSwappedDoubleToHost 36
NSConvertSwappedFloatToHost 37
NSCopyMemoryPages 37
NSCopyObject 38
NSCParameterAssert 39
NSCreateZone 40
NSDeallocateMemoryPages 40
NSDeallocateObject 41
NSDecimalAdd 41
NSDecimalCompact 42
NSDecimalCompare 43
NSDecimalCopy 43
NSDecimalDivide 44
NSDecimalIsNotANumber 45
NSDecimalMultiply 45
NSDecimalMultiplyByPowerOf10 46
NSDecimalNormalize 46
NSDecimalPower 47
NSDecimalRound 48
NSDecimalString 48
NSDecimalSubtract 49
NSDecrementExtraRefCountWasZero 49
NSDefaultMallocZone 50
NSEqualRanges 51
NSExtraRefCount 51
NSFullUserName 52
NSGetSizeAndAlignment 52
NSGetUncaughtExceptionHandler 53
NSHomeDirectory 53
NSHomeDirectoryForUser 54
NSHostByteOrder 54
NSIncrementExtraRefCount 55
NSIntersectionRange 55
NSLocalizedString 56
NSLocalizedStringFromTable 57
NSLocalizedStringFromTableInBundle 57

NSLocalizedStringWithDefaultValue 58
NSLocationInRange 59
NSLog 59
NSLogPageSize 60
NSLogv 60
NSMakeCollectable 61
NSMakeRange 62
NSMaxRange 63
NSOpenStepRootDirectory 63
NSPageSize 64
NSParameterAssert 64
NSProtocolFromString 65
NSRangeFromString 66
NSRealMemoryAvailable 67
NSRecycleZone 67
NSRoundDownToMultipleOfPageSize 68
NSRoundUpToMultipleOfPageSize 68
NSSearchPathForDirectoriesInDomains 69
NSSelectorFromString 70
NSSetUncaughtExceptionHandler 70
NSSetZoneName 71
NSShouldRetainWithZone 71
NSStringFromClass 72
NSStringFromProtocol 73
NSStringFromRange 73
NSStringFromSelector 74
NSSwapBigDoubleToHost 75
NSSwapBigFloatToHost 75
NSSwapBigIntToHost 76
NSSwapBigLongLongToHost 76
NSSwapBigLongToHost 77
NSSwapBigShortToHost 77
NSSwapDouble 78
NSSwapFloat 78
NSSwapHostDoubleToBig 79
NSSwapHostDoubleToLittle 79
NSSwapHostFloatToBig 80
NSSwapHostFloatToLittle 81
NSSwapHostIntToBig 81

NSSwapHostIntToLittle 82
NSSwapHostLongLongToBig 82
NSSwapHostLongLongToLittle 83
NSSwapHostLongToBig 83
NSSwapHostLongToLittle 84
NSSwapHostShortToBig 84
NSSwapHostShortToLittle 85
NSSwapInt 85
NSSwapLittleDoubleToHost 86
NSSwapLittleFloatToHost 86
NSSwapLittleIntToHost 87
NSSwapLittleLongLongToHost 87
NSSwapLittleLongToHost 88
NSSwapLittleShortToHost 89
NSSwapLong 89
NSSwapLongLong 90
NSSwapShort 90
NSTemporaryDirectory 91
NSUnionRange 91
NSUserName 92
NSZoneCalloc 92
NSZoneFree 93
NSZoneFromPointer 93
NSZoneMalloc 94
NSZoneName 94
NSZoneRealloc 95
NS_DURING 95
NS_ENDHANDLER 96
NS_HANDLER 96
NS_VALUERETURN 97
NS_VOIDRETURN 98

Document Revision History 99

Foundation Functions Reference

Framework	Foundation/Foundation.h
Declared in	NSBundle.h NSByteOrder.h NSDecimal.h NSException.h NSObjCRuntime.h NSObject.h NSPathUtilities.h NSRange.h NSZone.h

Overview

This chapter describes the functions and function-like macros defined in the Foundation Framework.

Functions by Task

Assertions

For additional information about Assertions, see *Assertions and Logging Programming Guide*.

`NSAssert` (page 18)
Generates an assertion if a given condition is false.

`NSAssert1` (page 20)
Generates an assertion if a given condition is false.

`NSAssert2` (page 21)
Generates an assertion if a given condition is false.

`NSAssert3` (page 23)
Generates an assertion if a given condition is false.

`NSAssert4` (page 24)

Generates an assertion if a given condition is false.

`NSAssert5` (page 26)

Generates an assertion if a given condition is false.

`NSCAssert` (page 27)

Generates an assertion if the given condition is false.

`NSCAssert1` (page 29)

`NSCAssert1` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert2` (page 30)

`NSCAssert2` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert3` (page 31)

`NSCAssert3` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert4` (page 32)

`NSCAssert4` is one of a series of macros that generate assertions if the given condition is false.

`NSCAssert5` (page 33)

`NSCAssert5` is one of a series of macros that generate assertions if the given condition is false.

`NSCParameterAssert` (page 39)

Evaluates the specified parameter.

`NSParameterAssert` (page 64)

Validates the specified parameter.

Bundles

For additional information on generating strings files see "Localizing String Resources" in *Internationalization Programming Topics*.

`NSLocalizedString` (page 56)

Returns a localized version of a string.

`NSLocalizedStringFromTable` (page 57)

Returns a localized version of a string.

`NSLocalizedStringFromTableInBundle` (page 57)

Returns a localized version of a string.

`NSLocalizedStringWithDefaultValue` (page 58)

Returns a localized version of a string.

Byte Ordering

`NSConvertHostDoubleToSwapped` (page 35)
Performs a type conversion.

`NSConvertHostFloatToSwapped` (page 36)
Performs a type conversion.

`NSConvertSwappedDoubleToHost` (page 36)
Performs a type conversion.

`NSConvertSwappedFloatToHost` (page 37)
Performs a type conversion.

`NSHostByteOrder` (page 54)
Returns the endian format.

`NSSwapBigDoubleToHost` (page 75)
A utility for swapping the bytes of a number.

`NSSwapBigFloatToHost` (page 75)
A utility for swapping the bytes of a number.

`NSSwapBigIntToHost` (page 76)
A utility for swapping the bytes of a number.

`NSSwapBigLongLongToHost` (page 76)
A utility for swapping the bytes of a number.

`NSSwapBigLongToHost` (page 77)
A utility for swapping the bytes of a number.

`NSSwapBigShortToHost` (page 77)
A utility for swapping the bytes of a number.

`NSSwapDouble` (page 78)
A utility for swapping the bytes of a number.

`NSSwapFloat` (page 78)
A utility for swapping the bytes of a number.

`NSSwapHostDoubleToBig` (page 79)
A utility for swapping the bytes of a number.

`NSSwapHostDoubleToLittle` (page 79)
A utility for swapping the bytes of a number.

`NSSwapHostFloatToBig` (page 80)
A utility for swapping the bytes of a number.

NSSwapHostFloatToLittle (page 81)
A utility for swapping the bytes of a number.

NSSwapHostIntToBig (page 81)
A utility for swapping the bytes of a number.

NSSwapHostIntToLittle (page 82)
A utility for swapping the bytes of a number.

NSSwapHostLongLongToBig (page 82)
A utility for swapping the bytes of a number.

NSSwapHostLongLongToLittle (page 83)
A utility for swapping the bytes of a number.

NSSwapHostLongToBig (page 83)
A utility for swapping the bytes of a number.

NSSwapHostLongToLittle (page 84)
A utility for swapping the bytes of a number.

NSSwapHostShortToBig (page 84)
A utility for swapping the bytes of a number.

NSSwapHostShortToLittle (page 85)
A utility for swapping the bytes of a number.

NSSwapInt (page 85)
A utility for swapping the bytes of a number.

NSSwapLittleDoubleToHost (page 86)
A utility for swapping the bytes of a number.

NSSwapLittleFloatToHost (page 86)
A utility for swapping the bytes of a number.

NSSwapLittleIntToHost (page 87)
A utility for swapping the bytes of a number.

NSSwapLittleLongLongToHost (page 87)
A utility for swapping the bytes of a number.

NSSwapLittleLongToHost (page 88)
A utility for swapping the bytes of a number.

NSSwapLittleShortToHost (page 89)
A utility for swapping the bytes of a number.

NSSwapLong (page 89)
A utility for swapping the bytes of a number.

`NSSwapLongLong` (page 90)

A utility for swapping the bytes of a number.

`NSSwapShort` (page 90)

A utility for swapping the bytes of a number.

Decimals

You can also use the class `NSDecimalNumber` for decimal arithmetic.

`NSDecimalAdd` (page 41)

Adds two decimal values.

`NSDecimalCompact` (page 42)

Compacts the decimal structure for efficiency.

`NSDecimalCompare` (page 43)

Compares two decimal values.

`NSDecimalCopy` (page 43)

Copies the value of a decimal number.

`NSDecimalDivide` (page 44)

Divides one decimal value by another.

`NSDecimalIsNotANumber` (page 45)

Returns a Boolean that indicates whether a given decimal contains a valid number.

`NSDecimalMultiply` (page 45)

Multiplies two decimal numbers together.

`NSDecimalMultiplyByPowerOf10` (page 46)

Multiplies a decimal by the specified power of 10.

`NSDecimalNormalize` (page 46)

Normalizes the internal format of two decimal numbers to simplify later operations.

`NSDecimalPower` (page 47)

Raises the decimal value to the specified power.

`NSDecimalRound` (page 48)

Rounds off the decimal value.

`NSDecimalString` (page 48)

Returns a string representation of the decimal value.

`NSDecimalSubtract` (page 49)
Subtracts one decimal value from another.

Exception Handling

You can find the following macros implemented in `NSException.h`. They are obsolete and should not be used. See *Exception Programming Topics* for information on how to handle exceptions.

`NS_DURING` (page 95)
Marks the start of the exception-handling domain.

`NS_ENDHANDLER` (page 96)
Marks the end of the local event handler.

`NS_HANDLER` (page 96)
Marks the end of the exception-handling domain and the start of the local exception handler.

`NS_VALUERETURN` (page 97)
Permits program control to exit from an exception-handling domain with a value of a specified type.

`NS_VOIDRETURN` (page 98)
Permits program control to exit from an exception-handling domain.

Managing Object Allocation and Deallocation

`NSAllocateObject` (page 18)
Creates and returns a new instance of a given class.

`NSCopyObject` (page 38)
Creates an exact copy of an object.

`NSDeallocateObject` (page 41)
Destroys an existing object.

`NSDecrementExtraRefCountWasZero` (page 49)
Decrements the specified object's reference count.

`NSExtraRefCount` (page 51)
Returns the specified object's reference count.

`NSIncrementExtraRefCount` (page 55)
Increments the specified object's reference count.

`NSShouldRetainWithZone` (page 71)
Indicates whether an object should be retained.

Interacting with the Objective-C Runtime

`NSGetSizeAndAlignment` (page 52)
Obtains the actual size and the aligned size of an encoded type.

`NSClassFromString` (page 34)
Obtains a class by name.

`NSStringFromClass` (page 72)
Returns the name of a class as a string.

`NSSelectorFromString` (page 70)
Returns the selector with a given name.

`NSStringFromSelector` (page 74)
Returns a string representation of a given selector.

`NSStringFromProtocol` (page 73)
Returns the name of a protocol as a string.

`NSProtocolFromString` (page 65)
Returns a the protocol with a given name.

Logging Output

`NSLog` (page 59)
Logs an error message to the Apple System Log facility.

`NSLogv` (page 60)
Logs an error message to the Apple System Log facility.

Managing File Paths

`NSFullUserName` (page 52)
Returns a string containing the full name of the current user.

`NSHomeDirectory` (page 53)
Returns the path to either the user's or application's home directory, depending on the platform.

`NSHomeDirectoryForUser` (page 54)
Returns the path to a given user's home directory.

`NSOpenStepRootDirectory` (page 63)
Returns the root directory of the user's system.

`NSSearchPathForDirectoriesInDomains` (page 69)

Creates a list of directory search paths.

`NSTemporaryDirectory` (page 91)

Returns the path of the temporary directory for the current user.

`NSUserName` (page 92)

Returns the logon name of the current user.

Managing Ranges

`NSEqualRanges` (page 51)

Returns a Boolean value that indicates whether two given ranges are equal.

`NSIntersectionRange` (page 55)

Returns the intersection of the specified ranges.

`NSLocationInRange` (page 59)

Returns a Boolean value that indicates whether a specified position is in a given range.

`NSMakeRange` (page 62)

Creates a new NSRange from the specified values.

`NSMaxRange` (page 63)

Returns the sum of the location and length of the range.

`NSRangeFromString` (page 66)

Returns a range from a textual representation.

`NSStringFromRange` (page 73)

Returns a string representation of a range.

`NSUnionRange` (page 91)

Returns the union of the specified ranges.

Uncaught Exception Handlers

Whether there's an uncaught exception handler function, any uncaught exceptions cause the program to terminate, unless the exception is raised during the posting of a notification.

`NSGetUncaughtExceptionHandler` (page 53)

Returns the top-level error handler.

`NSSetUncaughtExceptionHandler` (page 70)

Changes the top-level error handler.

Core Foundation ARC Integration

`CFBridgingRetain` (page 16)

Casts an Objective-C pointer to a Core Foundation pointer and also transfers ownership to the caller.

`CFBridgingRelease` (page 16)

Moves a non-Objective-C pointer to Objective-C and also transfers ownership to ARC.

Managing Memory

`NSAllocateMemoryPages` (page 17)

Allocates a new block of memory.

`NSCopyMemoryPages` (page 37)

Copies a block of memory.

`NSDeallocateMemoryPages` (page 40)

Deallocates the specified block of memory.

`NSLogPageSize` (page 60)

Returns the binary log of the page size.

`NSPageSize` (page 64)

Returns the number of bytes in a page.

`NSRealMemoryAvailable` (page 67)

Returns information about the user's system.

`NSRoundDownToMultipleOfPageSize` (page 68)

Returns the specified number of bytes rounded down to a multiple of the page size.

`NSRoundUpToMultipleOfPageSize` (page 68)

Returns the specified number of bytes rounded up to a multiple of the page size.

`NSMakeCollectable` (page 61)

Makes a newly allocated Core Foundation object eligible for collection. (Deprecated. Garbage collection is deprecated in OS X v10.8; instead,you should use AutomaticReference Counting—see *Transitioning to ARC Release Notes* .)

Managing Zones

Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.

`NSCreateZone` (page 40)

Creates a new zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSRecycleZone` (page 67)

Frees memory in a zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSSetZoneName` (page 71)

Sets the name of the specified zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSZoneCalloc` (page 92)

Allocates memory in a zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSZoneFree` (page 93)

Deallocates a block of memory in the specified zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSZoneFromPointer` (page 93)

Gets the zone for a given block of memory. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSZoneMalloc` (page 94)

Allocates memory in a zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSZoneName` (page 94)

Returns the name of the specified zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSZoneRealloc` (page 95)

Allocates memory in a zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

`NSDefaultMallocZone` (page 50)

Returns the default zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

Functions

CFBridgingRelease

Moves a non-Objective-C pointer to Objective-C and also transfers ownership to ARC.

```
id CFBridgingRelease(CFTypeRef X)
```

Discussion

You use this function to cast a Core Foundation-style object as an Objective-C object and transfer ownership of the object to ARC such that you don't have to release the object, as illustrated in this example:

```
CFStringRef cfName = ABRecordCopyValue(person, kABPersonFirstNameProperty);
NSString *name = (NSString *)CFBridgingRelease(cfName);
```

Availability

Available in iOS 5.0 and later.

See Also

`CFBridgingRetain` (page 16)

Related Sample Code

ABUIGroups
SimpleFTPSample
SimpleNetworkStreams
SimpleURLConnections
WiTap

Declared in

`NSObject.h`

CFBridgingRetain

Casts an Objective-C pointer to a Core Foundation pointer and also transfers ownership to the caller.

```
CFTypeRef CFBridgingRetain(id X)
```

Discussion

You use this function to cast an Objective-C object as Core Foundation-style object and take ownership of the object so that you can manage its lifetime. You are responsible for subsequently releasing the object, as illustrated in this example:

```
NSString *string = <#Get a string#>;
CFStringRef cfString = (CFStringRef)CFBridgingRetain(string);
// Use the CF string.
CFRelease(cfString);
```

Availability

Available in iOS 5.0 and later.

See Also

`CFBridgingRelease` (page 16)

Related Sample Code

ABUIGroups

GeocoderDemo

GLAirplay

Declared in

`NSObject.h`

NSAllocateMemoryPages

Allocates a new block of memory.

```
void * NSAllocateMemoryPages (
   NSUInteger bytes
);
```

Discussion

Allocates the integral number of pages whose total size is closest to, but not less than, `byteCount`. The allocated pages are guaranteed to be filled with zeros. If the allocation fails, raises `NSInvalidArgumentException`.

Availability

Available in iOS 2.0 and later.

See Also

`NSCopyMemoryPages` (page 37)

`NSDeallocateMemoryPages` (page 40)

Declared in
NSZone.h

NSAllocateObject

Creates and returns a new instance of a given class.

```
id NSAllocateObject (
   Class aClass,
   NSUInteger extraBytes,
   NSZone *zone
);
```

Parameters
`aClass`

The class of which to create an instance.

`extraBytes`

The number of extra bytes required for indexed instance variables (this value is typically `0`).

`zone`

The zone in which to create the new instance (pass `NULL` to specify the default zone).

Return Value
A new instance of `aClass` or `nil` if an instance could not be created.

Availability
Available in iOS 2.0 and later.

See Also
`NSDeallocateObject` (page 41)

Declared in
NSObject.h

NSAssert

Generates an assertion if a given condition is false.

```
#define NSAssert(condition, desc, ...)
```

Parameters

`condition`

An expression that evaluates to `YES` or `NO`.

`desc`

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for the arguments.

`...`

The arguments displayed in the `desc` string.

Discussion

The `NSAssert` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If `condition` evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` on the assertion handler for the current thread, passing `desc` as the description string.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability

Available in iOS 2.0 and later.

See Also

`NSLog` (page 59)
`NSLogv` (page 60)
`NSAssert1` (page 20)
`NSCAssert` (page 27)
`NSCParameterAssert` (page 39)
`NSParameterAssert` (page 64)

Related Sample Code

GenericKeychain

iAdSuite
Reachability
Sampler Unit Presets (LoadPresetDemo)
XMLPerformance

Declared in
`NSException.h`

NSAssert1

Generates an assertion if a given condition is false.

```
#define NSAssert1(condition, desc, arg1)
```

Parameters

`condition`

An expression that evaluates to `YES` or `NO`.

`desc`

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and a placeholder for a single argument.

`arg1`

An argument to be inserted, in place, into `desc`.

Discussion

The `NSAssert1` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If `condition` evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` on the assertion handler for the current thread, passing `desc` as the description string and `arg1` as a substitution variable.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 59)
`NSLogv` (page 60)
`NSAssert` (page 18)
`NSAssert2` (page 21)
`NSAssert3` (page 23)
`NSAssert4` (page 24)
`NSAssert5` (page 26)
`NSCAssert` (page 27)
`NSCParameterAssert` (page 39)
`NSParameterAssert` (page 64)

Related Sample Code
TopSongs
XMLPerformance

Declared in
`NSException.h`

NSAssert2

Generates an assertion if a given condition is false.

```
#define NSAssert2(condition, desc, arg1, arg2)
```

Parameters
`condition`
An expression that evaluates to `YES` or `NO`.

`desc`

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for two arguments.

`arg1`

An argument to be inserted, in place, into `desc`.

`arg2`

An argument to be inserted, in place, into `desc`.

Discussion

The `NSAssert2` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If `condition` evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` on the assertion handler for the current thread, passing `desc` as the description string and `arg1` and `arg2` as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability

Available in iOS 2.0 and later.

See Also

`NSLog` (page 59)
`NSLogv` (page 60)
`NSAssert` (page 18)
`NSAssert1` (page 20)
`NSAssert3` (page 23)
`NSAssert4` (page 24)
`NSAssert5` (page 26)
`NSCAssert` (page 27)
`NSCParameterAssert` (page 39)

NSParameterAssert (page 64)

Related Sample Code
TopSongs

XMLPerformance

Declared in
NSException.h

NSAssert3

Generates an assertion if a given condition is false.

```
#define NSAssert3(condition, desc, arg1, arg2, arg3)
```

Parameters

`condition`

An expression that evaluates to `YES` or `NO`.

`desc`

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for three arguments.

`arg1`

An argument to be inserted, in place, into `desc`.

`arg2`

An argument to be inserted, in place, into `desc`.

`arg3`

An argument to be inserted, in place, into `desc`.

Discussion

The `NSAssert3` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If `condition` evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` on the assertion handler for the current thread, passing `desc` as the description string and `arg1`, `arg2`, and `arg3` as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 59)
`NSLogv` (page 60)
`NSAssert` (page 18)
`NSAssert1` (page 20)
`NSAssert2` (page 21)
`NSAssert4` (page 24)
`NSAssert5` (page 26)
`NSCAssert` (page 27)
`NSCParameterAssert` (page 39)
`NSParameterAssert` (page 64)

Related Sample Code
TopSongs

Declared in
`NSException.h`

NSAssert4

Generates an assertion if a given condition is false.

```
#define NSAssert4(condition, desc, arg1, arg2, arg3, arg4)
```

Parameters
`condition`

An expression that evaluates to `YES` or `NO`.

`desc`

An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for four arguments.

`arg1`
An argument to be inserted, in place, into `desc`.

`arg2`
An argument to be inserted, in place, into `desc`.

`arg3`
An argument to be inserted, in place, into `desc`.

`arg4`
An argument to be inserted, in place, into `desc`.

Discussion

The `NSAssert4` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If `condition` evaluates to `NO`, the macro invokes `handleFailureInMethod:object:file:lineNumber:description:` on the assertion handler for the current thread, passing `desc` as the description string and `arg1`, `arg2`, `arg3`, and `arg4` as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability

Available in iOS 2.0 and later.

See Also

`NSLog` (page 59)
`NSLogv` (page 60)
`NSAssert` (page 18)
`NSAssert1` (page 20)
`NSAssert2` (page 21)
`NSAssert3` (page 23)
`NSAssert5` (page 26)

NSCAssert (page 27)
NSCParameterAssert (page 39)
NSParameterAssert (page 64)

Declared in
NSException.h

NSAssert5

Generates an assertion if a given condition is false.

```
#define NSAssert5(condition, desc, arg1, arg2, arg3, arg4, arg5)
```

Parameters

`condition`
An expression that evaluates to `YES` or `NO`.

`desc`
An `NSString` object that contains a `printf`-style string containing an error message describing the failure condition and placeholders for five arguments.

`arg1`
An argument to be inserted, in place, into `desc`.

`arg2`
An argument to be inserted, in place, into `desc`.

`arg3`
An argument to be inserted, in place, into `desc`.

`arg4`
An argument to be inserted, in place, into `desc`.

`arg5`
An argument to be inserted, in place, into `desc`.

Discussion
The `NSAssert5` macro evaluates the condition and serves as a front end to the assertion handler.

Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes the method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception. If `condition` evaluates to `NO`, the macro

invokes `handleFailureInMethod:object:file:lineNumber:description:` on the assertion handler for the current thread, passing `desc` as the description string and `arg1`, `arg2`, `arg3`, `arg4`, and `arg5` as substitution variables.

This macro should be used only within Objective-C methods.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 59)
`NSLogv` (page 60)
`NSAssert` (page 18)
`NSAssert1` (page 20)
`NSAssert2` (page 21)
`NSAssert3` (page 23)
`NSAssert4` (page 24)
`NSCAssert` (page 27)
`NSCParameterAssert` (page 39)
`NSParameterAssert` (page 64)

Declared in
`NSException.h`

NSCAssert

Generates an assertion if the given condition is false.

```
NSCAssert(condition, NSString *description)
```

Discussion

Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions. `NSCAssert` takes no arguments other than the condition and format string.

The `condition` must be an expression that evaluates to true or false. `description` is a printf-style format string that describes the failure condition.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability

Available in iOS 2.0 and later.

See Also

`NSLog` (page 59)
`NSLogv` (page 60)
`NSAssert` (page 18)
`NSCAssert1` (page 29)
`NSCParameterAssert` (page 39)
`NSParameterAssert` (page 64)

Related Sample Code

PrefsInCloud
Reachability
Sampler Unit Presets (LoadPresetDemo)
TopSongs
XMLPerformance

Declared in
NSException.h

NSCAssert1

NSCAssert1 is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert1(condition, NSString *description, arg1)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

The NSCAssert1 macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The condition expression must evaluate to true or false. description is a printf-style format string that describes the failure condition. arg1 is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

Important: Do not call functions with side effects in the condition parameter of this macro. The condition parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also
NSLog (page 59)
NSLogv (page 60)
NSCAssert (page 27)
NSCAssert2 (page 30)
NSCAssert3 (page 31)
NSCAssert4 (page 32)

NSCAssert5 (page 33)
NSCParameterAssert (page 39)
NSParameterAssert (page 64)

Related Sample Code
XMLPerformance

Declared in
NSException.h

NSCAssert2

NSCAssert2 is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert2(condition, NSString *description, arg1, arg2)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

The NSCAssert2 macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The condition expression must evaluate to true or false. description is a printf-style format string that describes the failure condition. Each argn is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

Important: Do not call functions with side effects in the condition parameter of this macro. The condition parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also

NSLog (page 59)
NSLogv (page 60)
NSCAssert (page 27)
NSCAssert1 (page 29)
NSCAssert3 (page 31)
NSCAssert4 (page 32)
NSCAssert5 (page 33)
NSCParameterAssert (page 39)
NSParameterAssert (page 64)

Related Sample Code

XMLPerformance

Declared in

NSException.h

NSCAssert3

NSCAssert3 is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert3(condition, NSString *description, arg1, arg2, arg3)
```

Discussion

Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

The NSCAssert3 macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The condition expression must evaluate to true or false. description is a printf-style format string that describes the failure condition. Each argn is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 59)
`NSLogv` (page 60)
`NSCAssert` (page 27)
`NSCAssert1` (page 29)
`NSCAssert2` (page 30)
`NSCAssert4` (page 32)
`NSCAssert5` (page 33)
`NSCParameterAssert` (page 39)
`NSParameterAssert` (page 64)

Declared in
`NSException.h`

NSCAssert4

`NSCAssert4` is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert4(condition, NSString *description, arg1, arg2, arg3, arg4)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert4` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The `condition` expression must evaluate to true or false. `description` is a printf-style format string that describes the failure condition. Each `argn` is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

Important: Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 59)
`NSLogv` (page 60)
`NSCAssert` (page 27)
`NSCAssert1` (page 29)
`NSCAssert2` (page 30)
`NSCAssert3` (page 31)
`NSCAssert5` (page 33)
`NSCParameterAssert` (page 39)
`NSParameterAssert` (page 64)

Declared in
`NSException.h`

NSCAssert5

`NSCAssert5` is one of a series of macros that generate assertions if the given condition is false.

```
NSCAssert5(condition, NSString *description, arg1, arg2, arg3, arg4, arg5)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

The `NSCAssert5` macro evaluates the condition and serves as a front end to the assertion handler. This macro should be used only within C functions.

The `condition` expression must evaluate to true or false. `description` is a printf-style format string that describes the failure condition. Each `argn` is an argument to be inserted, in place, into the description.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All macros return `void`.

> **Important:** Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

> **Note:** Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also
`NSLog` (page 59)
`NSLogv` (page 60)
`NSCAssert` (page 27)
`NSCAssert1` (page 29)
`NSCAssert2` (page 30)
`NSCAssert3` (page 31)
`NSCAssert4` (page 32)
`NSCParameterAssert` (page 39)
`NSParameterAssert` (page 64)

Declared in
`NSException.h`

NSClassFromString

Obtains a class by name.

```
Class NSClassFromString (
   NSString *aClassName
);
```

Parameters

`aClassName`

The name of a class.

Return Value

The class object named by `aClassName`, or `nil` if no class by that name is currently loaded. If `aClassName` is `nil`, returns `nil`.

Availability

Available in iOS 2.0 and later.

See Also

`NSStringFromClass` (page 72)

`NSProtocolFromString` (page 65)

`NSSelectorFromString` (page 70)

Related Sample Code

AdvancedURLConnections

GKAchievements

GKAuthentication

GKLeaderboards

Using External Displays

Declared in

`NSObjCRuntime.h`

NSConvertHostDoubleToSwapped

Performs a type conversion.

```
NSSwappedDouble NSConvertHostDoubleToSwapped (
   double x
);
```

Discussion

Converts the double value in `x` to a value whose bytes can be swapped. This function does not actually swap the bytes of `x`. You should not need to call this function directly.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapHostDoubleToBig` (page 79)

NSSwapHostDoubleToLittle (page 79)

Declared in
NSByteOrder.h

NSConvertHostFloatToSwapped

Performs a type conversion.

```
NSSwappedFloat NSConvertHostFloatToSwapped (
   float x
);
```

Discussion
Converts the float value in x to a value whose bytes can be swapped. This function does not actually swap the bytes of x. You should not need to call this function directly.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostFloatToBig (page 80)
NSSwapHostFloatToLittle (page 81)

Declared in
NSByteOrder.h

NSConvertSwappedDoubleToHost

Performs a type conversion.

```
double NSConvertSwappedDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the value in x to a double value. This function does not actually swap the bytes of x. You should not need to call this function directly.

Availability
Available in iOS 2.0 and later.

See Also

NSSwapBigDoubleToHost (page 75)
NSSwapLittleDoubleToHost (page 86)

Declared in

NSByteOrder.h

NSConvertSwappedFloatToHost

Performs a type conversion.

```
float NSConvertSwappedFloatToHost (
   NSSwappedFloat x
);
```

Discussion

Converts the value in `x` to a float value. This function does not actually swap the bytes of `x`. You should not need to call this function directly.

Availability

Available in iOS 2.0 and later.

See Also

NSSwapBigFloatToHost (page 75)
NSSwapLittleFloatToHost (page 86)

Declared in

NSByteOrder.h

NSCopyMemoryPages

Copies a block of memory.

```
void NSCopyMemoryPages (
   const void *source,
   void *dest,
   NSUInteger bytes
);
```

Discussion

Copies (or copies on write) `byteCount` bytes from `source` to `destination`.

Availability
Available in iOS 2.0 and later.

See Also
NSAllocateMemoryPages (page 17)
NSDeallocateMemoryPages (page 40)

Declared in
NSZone.h

NSCopyObject

Creates an exact copy of an object.

```
id NSCopyObject (
   id object,
   NSUInteger extraBytes,
   NSZone *zone
);
```

Parameters
`object`
The object to copy.

`extraBytes`
The number of extra bytes required for indexed instance variables (this value is typically `0`).

`zone`
The zone in which to create the new instance (pass `NULL` to specify the default zone).

Return Value
A new object that's an exact copy of `anObject`, or `nil` if `object` is `nil` or if `object` could not be copied.

Special Considerations
This function is dangerous and very difficult to use correctly. It's use as part of `copyWithZone:` by any class that can be subclassed, is highly error prone. Under GC or when using Objective-C 2.0, the zone is completely ignored.

This function is likely to be deprecated after OS X v10.6.

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 6.0.

See Also
NSAllocateObject (page 18)
NSDeallocateObject (page 41)

Related Sample Code
avTouch

Declared in
NSObject.h

NSCParameterAssert

Evaluates the specified parameter.

```
NSCParameterAssert(condition)
```

Discussion
Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class NSAssertionHandler. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an NSInternalInconsistencyException exception.

This macro validates a parameter for a C function. Simply provide the parameter as the condition argument. The macro evaluates the parameter and, if the parameter evaluates to false, logs an error message that includes the parameter and then raises an exception.

Assertions are disabled if the preprocessor macro NS_BLOCK_ASSERTIONS is defined. All macros return void.

Important: Do not call functions with side effects in the condition parameter of this macro. The condition parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

Note: Not all release configurations disable assertions by default.

Availability
Available in iOS 2.0 and later.

See Also

NSLog (page 59)
NSLogv (page 60)
NSAssert (page 18)
NSCAssert (page 27)
NSParameterAssert (page 64)

Declared in

NSException.h

NSCreateZone

Creates a new zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
NSZone * NSCreateZone (
   NSUInteger startSize,
   NSUInteger granularity,
   BOOL canFree
);
```

Return Value

A pointer to a new zone of `startSize` bytes, which will grow and shrink by `granularity` bytes. If `canFree` is 0, the allocator will never free memory, and `malloc` will be fast. Returns `NULL` if a new zone could not be created.

Availability

Available in iOS 2.0 and later.

Declared in

NSZone.h

NSDeallocateMemoryPages

Deallocates the specified block of memory.

```
void NSDeallocateMemoryPages (
   void *ptr,
   NSUInteger bytes
);
```

Discussion

This function deallocates memory that was allocated with `NSAllocateMemoryPages`.

Availability

Available in iOS 2.0 and later.

See Also

`NSCopyMemoryPages` (page 37)
`NSAllocateMemoryPages` (page 17)

Declared in

`NSZone.h`

NSDeallocateObject

Destroys an existing object.

```
void NSDeallocateObject (
   id object
);
```

Parameters

`object`

An object.

Discussion

This function deallocates `object`, which must have been allocated using `NSAllocateObject`.

Availability

Available in iOS 2.0 and later.

See Also

`NSAllocateObject` (page 18)

Declared in

`NSObject.h`

NSDecimalAdd

Adds two decimal values.

```
NSCalculationError NSDecimalAdd (
```

```
    NSDecimal *result,
    const NSDecimal *leftOperand,
    const NSDecimal *rightOperand,
    NSRoundingMode roundingMode
);
```

Discussion
Adds `leftOperand` to `rightOperand` and stores the sum in `result`.

An NSDecimal can represent a number with up to 38 significant digits. If a number is more precise than that, it must be rounded off. `roundingMode` determines how to round it off. There are four possible rounding modes:

`NSRoundDown`	Round return values down.
`NSRoundUp`	Round return values up.
`NSRoundPlain`	Round to the closest possible return value; when caught halfway between two positive numbers, round up; when caught between two negative numbers, round down.
`NSRoundBankers`	Round to the closest possible return value; when halfway between two possibilities, return the possibility whose last digit is even.

The return value indicates whether any machine limitations were encountered in the addition. If none were encountered, the function returns `NSCalculationNoError`. Otherwise it may return one of the following values: `NSCalculationLossOfPrecision`, `NSCalculationOverflow` or `NSCalculationUnderflow`. For descriptions of all these error conditions, see `exceptionDuringOperation:error:leftOperand:rightOperand:` in NSDecimalNumberBehaviors.

For more information, see *Number and Value Programming Topics* .

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalCompact

Compacts the decimal structure for efficiency.

```
void NSDecimalCompact (
   NSDecimal *number
);
```

Discussion
Formats number so that calculations using it will take up as little memory as possible. All the `NSDecimal...` arithmetic functions expect compact NSDecimal arguments.

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalCompare

Compares two decimal values.

```
NSComparisonResult NSDecimalCompare (
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand
);
```

Return Value
`NSOrderedDescending` if `leftOperand` is bigger than `rightOperand`; `NSOrderedAscending` if `rightOperand` is bigger than `leftOperand`; or `NSOrderedSame` if the two operands are equal.

Discussion
For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalCopy

Copies the value of a decimal number.

```
void NSDecimalCopy (
   NSDecimal *destination,
   const NSDecimal *source
);
```

Discussion
Copies the value in `source` to `destination`.

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalDivide

Divides one decimal value by another.

```
NSCalculationError NSDecimalDivide (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion
Divides `leftOperand` by `rightOperand` and stores the quotient, possibly rounded off according to `roundingMode`, in `result`. If `rightOperand` is 0, returns `NSDivideByZero`.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 41).

Note that repeating decimals or numbers with a mantissa larger than 38 digits cannot be represented precisely.

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalIsNotANumber

Returns a Boolean that indicates whether a given decimal contains a valid number.

```
BOOL NSDecimalIsNotANumber (
   const NSDecimal *dcm
);
```

Return Value

`YES` if the value in `decimal` represents a valid number, otherwise `NO`.

For more information, see *Number and Value Programming Topics*.

Availability

Available in iOS 2.0 and later.

Declared in

`NSDecimal.h`

NSDecimalMultiply

Multiplies two decimal numbers together.

```
NSCalculationError NSDecimalMultiply (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion

Multiplies `rightOperand` by `leftOperand` and stores the product, possibly rounded off according to `roundingMode`, in `result`.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 41).

For more information, see *Number and Value Programming Topics*.

Availability

Available in iOS 2.0 and later.

Declared in

`NSDecimal.h`

NSDecimalMultiplyByPowerOf10

Multiplies a decimal by the specified power of 10.

```
NSCalculationError NSDecimalMultiplyByPowerOf10 (
   NSDecimal *result,
   const NSDecimal *number,
   short power,
   NSRoundingMode roundingMode
);
```

Discussion
Multiplies `number` by `power` of 10 and stores the product, possibly rounded off according to `roundingMode`, in `result`.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 41).

For more information, see *Number and Value Programming Topics* .

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalNormalize

Normalizes the internal format of two decimal numbers to simplify later operations.

```
NSCalculationError NSDecimalNormalize (
   NSDecimal *number1,
   NSDecimal *number2,
   NSRoundingMode roundingMode
);
```

Discussion
An NSDecimal is represented in memory as a mantissa and an exponent, expressing the value mantissa x 10exponent. A number can have many NSDecimal representations; for example, the following table lists several valid NSDecimal representations for the number 100:

Mantissa	Exponent
100	0

Mantissa	Exponent
10	1
1	2

Format `number1` and `number2` so that they have equal exponents. This format makes addition and subtraction very convenient. Both `NSDecimalAdd` (page 41) and `NSDecimalSubtract` (page 49) call `NSDecimalNormalize`. You may want to use it if you write more complicated addition or subtraction routines.

For explanations of the possible return values, see `NSDecimalAdd` (page 41).

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalPower

Raises the decimal value to the specified power.

```
NSCalculationError NSDecimalPower (
   NSDecimal *result,
   const NSDecimal *number,
   NSUInteger power,
   NSRoundingMode roundingMode
);
```

Discussion
Raises `number` to `power`, possibly rounding off according to `roundingMode`, and stores the resulting value in `result`.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 41).

For more information, see *Number and Value Programming Topics*.

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalRound

Rounds off the decimal value.

```
void NSDecimalRound (
   NSDecimal *result,
   const NSDecimal *number,
   NSInteger scale,
   NSRoundingMode roundingMode
);
```

Discussion
Rounds `number` off according to the parameters `scale` and `roundingMode` and stores the result in `result`.

The `scale` value specifies the number of digits `result` can have after its decimal point. `roundingMode` specifies the way that number is rounded off. There are four possible values for `roundingMode`: `NSRoundDown`, `NSRoundUp`, `NSRoundPlain`, and `NSRoundBankers`. For thorough discussions of `scale` and `roundingMode`, see `NSDecimalNumberBehaviors`.

For more information, see *Number and Value Programming Topics* .

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalString

Returns a string representation of the decimal value.

```
NSString * NSDecimalString (
   const NSDecimal *dcm,
   id locale
);
```

Discussion
Returns a string representation of `decimal`. `locale` determines the format of the decimal separator.

For more information, see *Number and Value Programming Topics* .

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecimalSubtract

Subtracts one decimal value from another.

```
NSCalculationError NSDecimalSubtract (
   NSDecimal *result,
   const NSDecimal *leftOperand,
   const NSDecimal *rightOperand,
   NSRoundingMode roundingMode
);
```

Discussion
Subtracts `rightOperand` from `leftOperand` and stores the difference, possibly rounded off according to `roundingMode`, in `result`.

For explanations of the possible return values and rounding modes, see `NSDecimalAdd` (page 41).

For more information, see *Number and Value Programming Topics* .

Availability
Available in iOS 2.0 and later.

Declared in
`NSDecimal.h`

NSDecrementExtraRefCountWasZero

Decrements the specified object's reference count.

```
BOOL NSDecrementExtraRefCountWasZero (
   id object
);
```

Parameters
`object`
An object.

Return Value
`NO` if `anObject` had an extra reference count, or `YES` if `anObject` didn't have an extra reference count—indicating that the object should be deallocated (with `dealloc`).

Discussion
Decrements the "extra reference" count of `anObject`. Newly created objects have only one actual reference, so that a single release message results in the object being deallocated. Extra references are those beyond the single original reference and are usually created by sending the object a retain message. Your code should generally not use these functions unless it is overriding the `retain` or `release` methods.

Availability
Available in iOS 2.0 and later.

See Also
`NSExtraRefCount` (page 51)
`NSIncrementExtraRefCount` (page 55)

Declared in
`NSObject.h`

NSDefaultMallocZone

Returns the default zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
NSZone * NSDefaultMallocZone (void);
```

Return Value
The default zone, which is created automatically at startup.

Discussion
This zone is used by the standard C function `malloc`.

Availability
Available in iOS 2.0 and later.

Declared in
`NSZone.h`

NSEqualRanges

Returns a Boolean value that indicates whether two given ranges are equal.

```
BOOL NSEqualRanges (
   NSRange range1,
   NSRange range2
);
```

Return Value

`YES` if `range1` and `range2` have the same locations and lengths.

Availability

Available in iOS 2.0 and later.

Declared in

`NSRange.h`

NSExtraRefCount

Returns the specified object's reference count.

```
NSUInteger NSExtraRefCount (
   id object
);
```

Parameters

`object`

An object.

Return Value

The current reference count of `object`.

Discussion

This function is used in conjunction with `NSIncrementExtraRefCount` (page 55) and `NSDecrementExtraRefCountWasZero` (page 49) in situations where you need to override an object's `retain` and `release` methods.

Availability

Available in iOS 2.0 and later.

Declared in

`NSObject.h`

NSFullUserName

Returns a string containing the full name of the current user.

```
NSString * NSFullUserName (void);
```

Return Value
A string containing the full name of the current user.

Availability
Available in iOS 2.0 and later.

See Also
`NSUserName` (page 92)

Declared in
`NSPathUtilities.h`

NSGetSizeAndAlignment

Obtains the actual size and the aligned size of an encoded type.

```
const char * NSGetSizeAndAlignment (
   const char *typePtr,
   NSUInteger *sizep,
   NSUInteger *alignp
);
```

Discussion
Obtains the actual size and the aligned size of the first data type represented by `typePtr` and returns a pointer to the position of the next data type in `typePtr`. You can specify `NULL` for either `sizep` or `alignp` to ignore the corresponding information.

The value returned in `alignp` is the aligned size of the data type; for example, on some platforms, the aligned size of a `char` might be 2 bytes while the actual physical size is 1 byte.

Availability
Available in iOS 2.0 and later.

Declared in
`NSObjCRuntime.h`

NSGetUncaughtExceptionHandler

Returns the top-level error handler.

```
NSUncaughtExceptionHandler * NSGetUncaughtExceptionHandler (void);
```

Return Value
A pointer to the top-level error-handling function where you can perform last-minute logging before the program terminates.

Availability
Available in iOS 2.0 and later.

See Also
NSSetUncaughtExceptionHandler (page 70)

Declared in
NSException.h

NSHomeDirectory

Returns the path to either the user's or application's home directory, depending on the platform.

```
NSString * NSHomeDirectory (void);
```

Return Value
The path to the current home directory..

Discussion
In iOS, the home directory is the application's sandbox directory. In OS X, it is the application's sandbox directory or the current user's home directory (if the application is not in a sandbox)

For more information on file-system utilities, see *Low-Level File Management Programming Topics*.

Availability
Available in iOS 2.0 and later.

See Also
NSFullUserName (page 52)
NSUserName (page 92)
NSHomeDirectoryForUser (page 54)

Declared in
NSPathUtilities.h

NSHomeDirectoryForUser

Returns the path to a given user's home directory.

```
NSString * NSHomeDirectoryForUser (
   NSString *userName
);
```

Parameters
userName
The name of a user.

Return Value
The path to the home directory for the user specified by userName.

Discussion
For more information on file system utilities, see *Low-Level File Management Programming Topics*.

Availability
Available in iOS 2.0 and later.

See Also
NSFullUserName (page 52)
NSUserName (page 92)
NSHomeDirectory (page 53)

Declared in
NSPathUtilities.h

NSHostByteOrder

Returns the endian format.

```
long NSHostByteOrder (void);
```

Return Value
The endian format, either NS_LittleEndian or NS_BigEndian.

Availability
Available in iOS 2.0 and later.

Declared in
`NSByteOrder.h`

NSIncrementExtraRefCount

Increments the specified object's reference count.

```
void NSIncrementExtraRefCount (
   id object
);
```

Parameters
`object`
An object.

Discussion
This function increments the "extra reference" count of `object`. Newly created objects have only one actual reference, so that a single release message results in the object being deallocated. Extra references are those beyond the single original reference and are usually created by sending the object a retain message. Your code should generally not use these functions unless it is overriding the retain or release methods.

Availability
Available in iOS 2.0 and later.

See Also
`NSExtraRefCount` (page 51)
`NSDecrementExtraRefCountWasZero` (page 49)

Declared in
`NSObject.h`

NSIntersectionRange

Returns the intersection of the specified ranges.

```
NSRange NSIntersectionRange (
   NSRange range1,
   NSRange range2
);
```

Return Value
A range describing the intersection of `range1` and `range2`—that is, a range containing the indices that exist in both ranges.

Discussion
If the returned range's length field is 0, then the two ranges don't intersect, and the value of the location field is undefined.

Availability
Available in iOS 2.0 and later.

See Also
`NSUnionRange` (page 91)

Declared in
`NSRange.h`

NSLocalizedString

Returns a localized version of a string.

```
NSString *NSLocalizedString(NSString *key, NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` on the main bundle and a `nil` table.

Discussion
You can specify Unicode characters in `key` using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see `NSBundle`.

Special Considerations
In OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the `key` parameter must not contain any high-ASCII characters.

Availability
Available in iOS 2.0 and later.

Related Sample Code
Handling Touches Using Responder Methods and Gesture Recognizers
LocateMe
NavBar
UICatalog

XMLPerformance

Declared in
NSBundle.h

NSLocalizedStringFromTable

Returns a localized version of a string.

```
NSString *NSLocalizedStringFromTable(NSString *key,
   NSString *tableName,
   NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` on the main bundle, passing it the specified `key` and `tableName`.

Discussion
You can specify Unicode characters in `key` using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see `NSBundle`.

Special Considerations
In OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the `key` parameter must not contain any high-ASCII characters.

Availability
Available in iOS 2.0 and later.

Declared in
NSBundle.h

NSLocalizedStringFromTableInBundle

Returns a localized version of a string.

```
NSString *NSLocalizedStringFromTableInBundle(NSString *key,
   NSString *tableName,
   NSBundle *bundle,
   NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` on `bundle`, passing it the specified `key` and `tableName`.

Discussion
You can specify Unicode characters in `key` using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

For more information, see `NSBundle`.

Special Considerations
In OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the `key` parameter must not contain any high-ASCII characters.

Availability
Available in iOS 2.0 and later.

Declared in
`NSBundle.h`

NSLocalizedStringWithDefaultValue

Returns a localized version of a string.

```
NSString *NSLocalizedStringWithDefaultValue(NSString *key,
   NSString *tableName,
   NSBundle *bundle,
   NSString *value,
   NSString *comment)
```

Return Value
The result of invoking `localizedStringForKey:value:table:` on `bundle`, passing it the specified `key`, `value`, and `tableName`.

Discussion
You can specify Unicode characters in `key` using `\\Uxxxx`—see the `-u` option for for the `genstrings` utility.

If you use `genstrings` to parse your code for localizable strings, you can use this method to specify an initial value that is different from `key`.

For more information, see `NSBundle`.

Special Considerations

In OS X v10.4 and earlier, to ensure correct parsing by the `genstrings` utility, the `key` parameter must not contain any high-ASCII characters.

Availability

Available in iOS 2.0 and later.

Declared in

`NSBundle.h`

NSLocationInRange

Returns a Boolean value that indicates whether a specified position is in a given range.

```
BOOL NSLocationInRange (
   NSUInteger loc,
   NSRange range
);
```

Return Value

`YES` if `loc` lies within `range`—that is, if it's greater than or equal to `range.location` and less than `range.location` plus `range.length`.

Availability

Available in iOS 2.0 and later.

Declared in

`NSRange.h`

NSLog

Logs an error message to the Apple System Log facility.

```
void NSLog (
   NSString *format,
   ...
);
```

Discussion

Simply calls `NSLogv` (page 60), passing it a variable number of arguments.

Availability
Available in iOS 2.0 and later.

See Also
NSLogv (page 60)

Related Sample Code
AdvancedURLConnections
Audio Mixer (MixerHost)
AVCaptureAudioDataOutput To AudioUnit iOS
BTLE Central Peripheral Transfer
CoreBluetooth Temperature Sensor

Declared in
NSObjCRuntime.h

NSLogPageSize

Returns the binary log of the page size.

```
NSUInteger NSLogPageSize (void);
```

Return Value
The binary log of the page size.

Availability
Available in iOS 2.0 and later.

See Also
NSRoundDownToMultipleOfPageSize (page 68)
NSRoundUpToMultipleOfPageSize (page 68)
NSPageSize (page 64)

Declared in
NSZone.h

NSLogv

Logs an error message to the Apple System Log facility.

```
void NSLogv (
```

```
   NSString *format,
   va_list args
);
```

Discussion

Logs an error message to the Apple System Log facility (see `man 3 asl`). If the `STDERR_FILENO` file descriptor has been redirected away from the default or is going to a tty, it will also be written there. If you want to direct output elsewhere, you need to use a custom logging facility.

The message consists of a timestamp and the process ID prefixed to the string you pass in. You compose this string with a format string, `format`, and one or more arguments to be inserted into the string. The format specification allowed by these functions is that which is understood by `NSString`'s formatting capabilities (which is not necessarily the set of format escapes and flags understood by `printf`). The supported format specifiers are described in "String Format Specifiers". A final hard return is added to the error message if one is not present in the format.

In general, you should use the `NSLog` (page 59) function instead of calling this function directly. If you do use this function directly, you must have prepared the variable argument list in the `args` argument by calling the standard C macro `va_start`. Upon completion, you must similarly call the standard C macro `va_end` for this list.

Output from `NSLogv` is serialized, in that only one thread in a process can be doing the writing/logging described above at a time. All attempts at writing/logging a message complete before the next thread can begin its attempts.

The effects of `NSLogv` are not serialized with subsystems other than those discussed above (such as the standard I/O package) and do not produce side effects on those subsystems (such as causing buffered output to be flushed, which may be undesirable).

Availability

Available in iOS 2.0 and later.

See Also

`NSLog` (page 59)

Declared in

`NSObjCRuntime.h`

NSMakeCollectable

Makes a newly allocated Core Foundation object eligible for collection. (Deprecated. Garbage collection is deprecated in OS X v10.8; instead,you should use AutomaticReference Counting–see Transitioning to ARC Release Notes.)

```
NS_INLINE id NSMakeCollectable(CFTypeRef cf) {
   return cf ? (id)CFMakeCollectable(cf) : nil;
}
```

Discussion

This function is a wrapper for `CFMakeCollectable`, but its return type is `id`—avoiding the need for casting when using Cocoa objects.

This function may be useful when returning Core Foundation objects in code that must support both garbage-collected and non-garbage-collected environments, as illustrated in the following example.

```
- (CFDateRef)foo {
    CFDateRef aCFDate;
    // ...
    return [NSMakeCollectable(aCFDate) autorelease];
}
```

CFTypeRef style objects are garbage collected, yet only sometime after the last `CFRelease` is performed. Particularly for fully-bridged CFTypeRef objects such as CFStrings and collections (such as CFDictionary), you must call either `CFMakeCollectable` or the more type safe `NSMakeCollectable`, preferably right upon allocation.

Availability

Available in iOS 2.0 and later.

Declared in

`NSZone.h`

NSMakeRange

Creates a new NSRange from the specified values.

```
NSRange NSMakeRange (
   NSUInteger loc,
   NSUInteger len
);
```

Return Value

An `NSRange` with location `location` and length `length`.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CoreTextPageViewer

EADemo

GKTapper

LaunchMe

Simple UISearchBar with State Restoration

Declared in
`NSRange.h`

NSMaxRange

Returns the sum of the location and length of the range.

```
NSUInteger NSMaxRange (
   NSRange range
);
```

Return Value
The sum of the location and length of the range—that is, `range.location` + `range.length`.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CoreTextPageViewer

TableView Fundamentals for iOS

Declared in
`NSRange.h`

NSOpenStepRootDirectory

Returns the root directory of the user's system.

```
NSString * NSOpenStepRootDirectory (void);
```

Return Value
A string identifying the root directory of the user's system.

Discussion

For more information on file system utilities, see *Low-Level File Management Programming Topics*.

Availability

Available in iOS 2.0 and later.

See Also

`NSHomeDirectory` (page 53)
`NSHomeDirectoryForUser` (page 54)

Declared in

`NSPathUtilities.h`

NSPageSize

Returns the number of bytes in a page.

```
NSUInteger NSPageSize (void);
```

Return Value

The number of bytes in a page.

Availability

Available in iOS 2.0 and later.

See Also

`NSRoundDownToMultipleOfPageSize` (page 68)
`NSRoundUpToMultipleOfPageSize` (page 68)
`NSLogPageSize` (page 60)

Declared in

`NSZone.h`

NSParameterAssert

Validates the specified parameter.

```
NSParameterAssert(condition)
```

Discussion

Assertions evaluate a condition and, if the condition evaluates to false, call the assertion handler for the current thread, passing it a format string and a variable number of arguments. Each thread has its own assertion handler, which is an object of class `NSAssertionHandler`. When invoked, an assertion handler prints an error message that includes method and class names (or the function name). It then raises an `NSInternalInconsistencyException` exception.

This macro validates a parameter for an Objective-C method. Simply provide the parameter as the `condition` argument. The macro evaluates the parameter and, if it is false, it logs an error message that includes the parameter and then raises an exception.

Assertions are disabled if the preprocessor macro `NS_BLOCK_ASSERTIONS` is defined. All assertion macros return void.

> **Important:** Do not call functions with side effects in the `condition` parameter of this macro. The `condition` parameter is not evaluated when assertions are disabled, so if you call functions with side effects, those functions may never get called when you build the project in a non-debug configuration.

> **Note:** Not all release configurations disable assertions by default.

Availability

Available in iOS 2.0 and later.

See Also

`NSLog` (page 59)
`NSLogv` (page 60)
`NSAssert` (page 18)
`NSCAssert` (page 27)
`NSCParameterAssert` (page 39)

Related Sample Code

MTAudioProcessingTap Audio Processor
Reachability

Declared in

`NSException.h`

NSProtocolFromString

Returns a the protocol with a given name.

```
Protocol *NSProtocolFromString (
   NSString *namestr
);
```

Parameters

`namestr`

The name of a protocol.

Return Value

The protocol object named by `namestr`, or `nil` if no protocol by that name is currently loaded. If `namestr` is `nil`, returns `nil`.

Availability

Available in iOS 2.0 and later.

See Also

`NSStringFromProtocol` (page 73)
`NSClassFromString` (page 34)
`NSSelectorFromString` (page 70)

Declared in

`NSObjCRuntime.h`

NSRangeFromString

Returns a range from a textual representation.

```
NSRange NSRangeFromString (
   NSString *aString
);
```

Discussion

Scans `aString` for two integers which are used as the location and length values, in that order, to create an `NSRange` struct. If `aString` only contains a single integer, it is used as the location value. If `aString` does not contain any integers, this function returns an `NSRange` struct whose location and length values are both 0.

Availability

Available in iOS 2.0 and later.

See Also
NSStringFromRange (page 73)

Declared in
NSRange.h

NSRealMemoryAvailable

Returns information about the user's system.

```
NSUInteger NSRealMemoryAvailable (void);
```

Return Value
The number of bytes available in RAM.

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 6.0.

Declared in
NSZone.h

NSRecycleZone

Frees memory in a zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
void NSRecycleZone (
   NSZone *zone
);
```

Discussion
Frees zone after adding any of its pointers still in use to the default zone. (This strategy prevents retained objects from being inadvertently destroyed.)

Availability
Available in iOS 2.0 and later.

Declared in
NSZone.h

NSRoundDownToMultipleOfPageSize

Returns the specified number of bytes rounded down to a multiple of the page size.

```
NSUInteger NSRoundDownToMultipleOfPageSize (
   NSUInteger bytes
);
```

Return Value

In bytes, the multiple of the page size that is closest to, but not greater than, `byteCount` (that is, the number of bytes rounded down to a multiple of the page size).

Availability

Available in iOS 2.0 and later.

See Also

`NSPageSize` (page 64)
`NSLogPageSize` (page 60)
`NSRoundUpToMultipleOfPageSize` (page 68)

Declared in

`NSZone.h`

NSRoundUpToMultipleOfPageSize

Returns the specified number of bytes rounded up to a multiple of the page size.

```
NSUInteger NSRoundUpToMultipleOfPageSize (
   NSUInteger bytes
);
```

Return Value

In bytes, the multiple of the page size that is closest to, but not less than, `byteCount` (that is, the number of bytes rounded up to a multiple of the page size).

Availability

Available in iOS 2.0 and later.

See Also

`NSPageSize` (page 64)
`NSLogPageSize` (page 60)
`NSRoundDownToMultipleOfPageSize` (page 68)

Declared in
NSZone.h

NSSearchPathForDirectoriesInDomains

Creates a list of directory search paths.

```
NSArray * NSSearchPathForDirectoriesInDomains (
   NSSearchPathDirectory directory,
   NSSearchPathDomainMask domainMask,
   BOOL expandTilde
);
```

Discussion
Creates a list of path strings for the specified directories in the specified domains. The list is in the order in which you should search the directories. If `expandTilde` is `YES`, tildes are expanded as described in `stringByExpandingTildeInPath`.

You should consider using the `NSFileManager` methods `URLsForDirectory:inDomains:` and `URLForDirectory:inDomain:appropriateForURL:create:error:`. which return URLs, which are the preferred format.

For more information on file system utilities, see *File System Programming Guide*.

> **Note:** The directory returned by this method may not exist. This method simply gives you the appropriate location for the requested directory. Depending on the application's needs, it may be up to the developer to create the appropriate directory and any in between.

Availability
Available in iOS 2.0 and later.

Related Sample Code
AirDrop Examples
AQOfflineRenderTest
MultipeerGroupChat
PhotosByLocation
XMLPerformance

Declared in
NSPathUtilities.h

NSSelectorFromString

Returns the selector with a given name.

```
SEL NSSelectorFromString (
   NSString *aSelectorName
);
```

Parameters

`aSelectorName`

A string of any length, with any characters, that represents the name of a selector.

Return Value

The selector named by `aSelectorName`. If `aSelectorName` is `nil`, or cannot be converted to UTF-8 (this should be only due to insufficient memory), returns `(SEL)0`.

Discussion

To make a selector, `NSSelectorFromString` passes a UTF-8 encoded character representation of `aSelectorName` to `sel_registerName` and returns the value returned by that function. Note, therefore, that if the selector does not exist it is registered and the newly-registered selector is returned.

Recall that a colon (":") is part of a method name; `setHeight` is not the same as `setHeight:`.

Availability

Available in iOS 2.0 and later.

See Also

`NSStringFromSelector` (page 74)
`NSProtocolFromString` (page 65)
`NSClassFromString` (page 34)

Declared in

`NSObjCRuntime.h`

NSSetUncaughtExceptionHandler

Changes the top-level error handler.

```
void NSSetUncaughtExceptionHandler (
   NSUncaughtExceptionHandler *
);
```

Discussion

Sets the top-level error-handling function where you can perform last-minute logging before the program terminates.

Availability

Available in iOS 2.0 and later.

See Also

`NSGetUncaughtExceptionHandler` (page 53)

Declared in

`NSException.h`

NSSetZoneName

Sets the name of the specified zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
void NSSetZoneName (
   NSZone *zone,
   NSString *name
);
```

Discussion

Sets the name of `zone` to `name`, which can aid in debugging.

Availability

Available in iOS 2.0 and later.

Declared in

`NSZone.h`

NSShouldRetainWithZone

Indicates whether an object should be retained.

```
BOOL NSShouldRetainWithZone (
   id anObject,
   NSZone *requestedZone
);
```

Parameters

`anObject`

An object.

`requestedZone`

A memory zone.

Return Value

Returns `YES` if `requestedZone` is `NULL`, the default zone, or the zone in which `anObject` was allocated; otherwise `NO`.

Discussion

This function is typically called from inside an `NSObject`'s `copyWithZone:`, when deciding whether to retain `anObject` as opposed to making a copy of it.

Availability

Available in iOS 2.0 and later.

Declared in

`NSObject.h`

NSStringFromClass

Returns the name of a class as a string.

```
NSString * NSStringFromClass (
   Class aClass
);
```

Parameters

`aClass`

A class.

Return Value

A string containing the name of `aClass`. If `aClass` is `nil`, returns `nil`.

Availability

Available in iOS 2.0 and later.

See Also

`NSClassFromString` (page 34)

`NSStringFromProtocol` (page 73)

NSStringFromSelector (page 74)

Related Sample Code
Handling Touches Using Responder Methods and Gesture Recognizers
iAdSuite
Inter-App Audio Examples
TableView Fundamentals for iOS
UnwindSegue

Declared in
NSObjCRuntime.h

NSStringFromProtocol

Returns the name of a protocol as a string.

```
NSString * NSStringFromProtocol (
   Protocol *proto
);
```

Parameters
proto
A protocol.

Return Value
A string containing the name of proto.

Availability
Available in iOS 2.0 and later.

See Also
NSProtocolFromString (page 65)
NSStringFromClass (page 72)
NSStringFromSelector (page 74)

Declared in
NSObjCRuntime.h

NSStringFromRange

Returns a string representation of a range.

```
NSString * NSStringFromRange (
   NSRange range
);
```

Return Value

A string of the form "{*a*, *b*}", where *a* and *b* are non-negative integers representing `aRange`.

Availability

Available in iOS 2.0 and later.

Declared in

`NSRange.h`

NSStringFromSelector

Returns a string representation of a given selector.

```
NSString *NSStringFromSelector (
   SEL aSelector
);
```

Parameters

`aSelector`

A selector.

Return Value

A string representation of `aSelector`.

Availability

Available in iOS 2.0 and later.

See Also

`NSSelectorFromString` (page 70)

`NSStringFromProtocol` (page 73)

`NSStringFromClass` (page 72)

Related Sample Code

AirDrop Examples

Birthdays

Inter-App Audio Examples

Declared in
NSObjCRuntime.h

NSSwapBigDoubleToHost

A utility for swapping the bytes of a number.

```
double NSSwapBigDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the big-endian value in x to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls NSSwapDouble (page 78) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostDoubleToBig (page 79)
NSSwapLittleDoubleToHost (page 86)

Declared in
NSByteOrder.h

NSSwapBigFloatToHost

A utility for swapping the bytes of a number.

```
float NSSwapBigFloatToHost (
   NSSwappedFloat x
);
```

Discussion
Converts the big-endian value in x to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls NSSwapFloat (page 78) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostFloatToBig (page 80)

NSSwapLittleFloatToHost (page 86)

Declared in
NSByteOrder.h

NSSwapBigIntToHost

A utility for swapping the bytes of a number.

```
unsigned int NSSwapBigIntToHost (
   unsigned int x
);
```

Discussion
Converts the big-endian value in x to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls NSSwapInt (page 85) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostIntToBig (page 81)
NSSwapLittleIntToHost (page 87)

Declared in
NSByteOrder.h

NSSwapBigLongLongToHost

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapBigLongLongToHost (
   unsigned long long x
);
```

Discussion
Converts the big-endian value in x to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls NSSwapLongLong (page 90) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also

NSSwapHostLongLongToBig (page 82)
NSSwapLittleLongLongToHost (page 87)

Declared in

NSByteOrder.h

NSSwapBigLongToHost

A utility for swapping the bytes of a number.

```
unsigned long NSSwapBigLongToHost (
   unsigned long x
);
```

Discussion

Converts the big-endian value in x to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls NSSwapLong (page 89) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

NSSwapHostLongToBig (page 83)
NSSwapLittleLongToHost (page 88)

Declared in

NSByteOrder.h

NSSwapBigShortToHost

A utility for swapping the bytes of a number.

```
unsigned short NSSwapBigShortToHost (
   unsigned short x
);
```

Discussion

Converts the big-endian value in x to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls NSSwapShort (page 90) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostShortToBig (page 84)
NSSwapLittleShortToHost (page 89)

Declared in
NSByteOrder.h

NSSwapDouble

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapDouble (
   NSSwappedDouble x
);
```

Discussion
Swaps the bytes of x and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of x are numbered from 1 to 8, this function swaps bytes 1 and 8, bytes 2 and 7, bytes 3 and 6, and bytes 4 and 5.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapLongLong (page 90)
NSSwapFloat (page 78)

Declared in
NSByteOrder.h

NSSwapFloat

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapFloat (
   NSSwappedFloat x
);
```

Discussion

Swaps the bytes of `x` and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of `x` are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapLong` (page 89)
`NSSwapDouble` (page 78)

Declared in

`NSByteOrder.h`

NSSwapHostDoubleToBig

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapHostDoubleToBig (
   double x
);
```

Discussion

Converts the value in `x`, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapDouble` (page 78) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapBigDoubleToHost` (page 75)
`NSSwapHostDoubleToLittle` (page 79)

Declared in

`NSByteOrder.h`

NSSwapHostDoubleToLittle

A utility for swapping the bytes of a number.

```
NSSwappedDouble NSSwapHostDoubleToLittle (
   double x
);
```

Discussion

Converts the value in `x`, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapDouble` (page 78) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapLittleDoubleToHost` (page 86)
`NSSwapHostDoubleToBig` (page 79)

Declared in

`NSByteOrder.h`

NSSwapHostFloatToBig

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapHostFloatToBig (
   float x
);
```

Discussion

Converts the value in `x`, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapFloat` (page 78) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapBigFloatToHost` (page 75)
`NSSwapHostFloatToLittle` (page 81)

Declared in

`NSByteOrder.h`

NSSwapHostFloatToLittle

A utility for swapping the bytes of a number.

```
NSSwappedFloat NSSwapHostFloatToLittle (
   float x
);
```

Discussion

Converts the value in `x`, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapFloat` (page 78) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapLittleFloatToHost` (page 86)
`NSSwapHostFloatToBig` (page 80)

Declared in

`NSByteOrder.h`

NSSwapHostIntToBig

A utility for swapping the bytes of a number.

```
unsigned int NSSwapHostIntToBig (
   unsigned int x
);
```

Discussion

Converts the value in `x`, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapInt` (page 85) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapBigIntToHost` (page 76)
`NSSwapHostIntToLittle` (page 82)

Declared in

`NSByteOrder.h`

NSSwapHostIntToLittle

A utility for swapping the bytes of a number.

```
unsigned int NSSwapHostIntToLittle (
   unsigned int x
);
```

Discussion

Converts the value in `x`, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapInt` (page 85) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapLittleIntToHost` (page 87)
`NSSwapHostIntToBig` (page 81)

Declared in

`NSByteOrder.h`

NSSwapHostLongLongToBig

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapHostLongLongToBig (
   unsigned long long x
);
```

Discussion

Converts the value in `x`, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLongLong` (page 90) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapBigLongLongToHost` (page 76)
`NSSwapHostLongLongToLittle` (page 83)

Declared in
NSByteOrder.h

NSSwapHostLongLongToLittle

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapHostLongLongToLittle (
   unsigned long long x
);
```

Discussion
Converts the value in x, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapLongLong (page 90) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapLittleLongLongToHost (page 87)
NSSwapHostLongLongToBig (page 82)

Declared in
NSByteOrder.h

NSSwapHostLongToBig

A utility for swapping the bytes of a number.

```
unsigned long NSSwapHostLongToBig (
   unsigned long x
);
```

Discussion
Converts the value in x, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapLong (page 89) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapBigLongToHost (page 77)

NSSwapHostLongToLittle (page 84)

Declared in
NSByteOrder.h

NSSwapHostLongToLittle

A utility for swapping the bytes of a number.

```
unsigned long NSSwapHostLongToLittle (
   unsigned long x
);
```

Discussion
Converts the value in x, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapLong (page 89) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapLittleLongToHost (page 88)
NSSwapHostLongToBig (page 83)

Declared in
NSByteOrder.h

NSSwapHostShortToBig

A utility for swapping the bytes of a number.

```
unsigned short NSSwapHostShortToBig (
   unsigned short x
);
```

Discussion
Converts the value in x, specified in the current endian format, to big-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapShort (page 90) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also

NSSwapBigShortToHost (page 77)
NSSwapHostShortToLittle (page 85)

Declared in

NSByteOrder.h

NSSwapHostShortToLittle

A utility for swapping the bytes of a number.

```
unsigned short NSSwapHostShortToLittle (
   unsigned short x
);
```

Discussion

Converts the value in x, specified in the current endian format, to little-endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls NSSwapShort (page 90) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

NSSwapLittleShortToHost (page 89)
NSSwapHostShortToBig (page 84)

Declared in

NSByteOrder.h

NSSwapInt

A utility for swapping the bytes of a number.

```
unsigned int NSSwapInt (
   unsigned int inv
);
```

Discussion

Swaps the bytes of iv and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of inv are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapShort (page 90)
NSSwapLong (page 89)
NSSwapLongLong (page 90)

Declared in
NSByteOrder.h

NSSwapLittleDoubleToHost

A utility for swapping the bytes of a number.

```
double NSSwapLittleDoubleToHost (
   NSSwappedDouble x
);
```

Discussion
Converts the little-endian formatted value in x to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls NSSwapDouble (page 78) to perform the swap.

Availability
Available in iOS 2.0 and later.

See Also
NSSwapHostDoubleToLittle (page 79)
NSSwapBigDoubleToHost (page 75)
NSConvertSwappedDoubleToHost (page 36)

Declared in
NSByteOrder.h

NSSwapLittleFloatToHost

A utility for swapping the bytes of a number.

```
float NSSwapLittleFloatToHost (
   NSSwappedFloat x
);
```

Discussion

Converts the little-endian formatted value in `x` to the current endian format and returns the resulting value. If it is necessary to swap the bytes of `x`, this function calls `NSSwapFloat` (page 78) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapHostFloatToLittle` (page 81)
`NSSwapBigFloatToHost` (page 75)
`NSConvertSwappedFloatToHost` (page 37)

Declared in

`NSByteOrder.h`

NSSwapLittleIntToHost

A utility for swapping the bytes of a number.

```
unsigned int NSSwapLittleIntToHost (
   unsigned int x
);
```

Discussion

Converts the little-endian formatted value in `x` to the current endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapInt` (page 85) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapHostIntToLittle` (page 82)
`NSSwapBigIntToHost` (page 76)

Declared in

`NSByteOrder.h`

NSSwapLittleLongLongToHost

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapLittleLongLongToHost (
   unsigned long long x
);
```

Discussion

Converts the little-endian formatted value in `x` to the current endian format and returns the resulting value. If it is necessary to swap the bytes, this function calls `NSSwapLongLong` (page 90) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapHostLongLongToLittle` (page 83)
`NSSwapBigLongLongToHost` (page 76)

Declared in

`NSByteOrder.h`

NSSwapLittleLongToHost

A utility for swapping the bytes of a number.

```
unsigned long NSSwapLittleLongToHost (
   unsigned long x
);
```

Discussion

Converts the little-endian formatted value in `x` to the current endian format and returns the resulting value. If it is necessary to swap the bytes of x, this function calls `NSSwapLong` (page 89) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapHostLongToLittle` (page 84)
`NSSwapBigLongToHost` (page 77)
`NSSwapLong` (page 89)

Declared in

`NSByteOrder.h`

NSSwapLittleShortToHost

A utility for swapping the bytes of a number.

```
unsigned short NSSwapLittleShortToHost (
   unsigned short x
);
```

Discussion

Converts the little-endian formatted value in `x` to the current endian format and returns the resulting value. If it is necessary to swap the bytes of `x`, this function calls `NSSwapShort` (page 90) to perform the swap.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapHostShortToLittle` (page 85)
`NSSwapBigShortToHost` (page 77)

Declared in

`NSByteOrder.h`

NSSwapLong

A utility for swapping the bytes of a number.

```
unsigned long NSSwapLong (
   unsigned long inv
);
```

Discussion

Swaps the bytes of `inv` and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of `inv` are numbered from 1 to 4, this function swaps bytes 1 and 4, and bytes 2 and 3.

Availability

Available in iOS 2.0 and later.

See Also

`NSSwapLongLong` (page 90)
`NSSwapInt` (page 85)
`NSSwapFloat` (page 78)

Declared in
NSByteOrder.h

NSSwapLongLong

A utility for swapping the bytes of a number.

```
unsigned long long NSSwapLongLong (
   unsigned long long inv
);
```

Discussion
Swaps the bytes of `inv` and returns the resulting value. Bytes are swapped from each low-order position to the corresponding high-order position and vice versa. For example, if the bytes of `inv` are numbered from 1 to 8, this function swaps bytes 1 and 8, bytes 2 and 7, bytes 3 and 6, and bytes 4 and 5.

Availability
Available in iOS 2.0 and later.

See Also
`NSSwapLong` (page 89)
`NSSwapDouble` (page 78)

Declared in
NSByteOrder.h

NSSwapShort

A utility for swapping the bytes of a number.

```
unsigned short NSSwapShort (
   unsigned short inv
);
```

Discussion
Swaps the low-order and high-order bytes of `inv` and returns the resulting value.

Availability
Available in iOS 2.0 and later.

See Also

NSSwapInt (page 85)
NSSwapLong (page 89)

Declared in

NSByteOrder.h

NSTemporaryDirectory

Returns the path of the temporary directory for the current user.

```
NSString * NSTemporaryDirectory (void);
```

Return Value

A string containing the path of the temporary directory for the current user. If no such directory is currently available, returns `nil`.

Discussion

See the `NSFileManager` method `URLForDirectory:inDomain:appropriateForURL:create:error:` for the preferred means of finding the correct temporary directory.

Availability

Available in iOS 2.0 and later.

See Also

NSSearchPathForDirectoriesInDomains (page 69)
NSHomeDirectory (page 53)

Related Sample Code

AVMovieExporter
Inter-App Audio Examples
SpeakHere

Declared in

NSPathUtilities.h

NSUnionRange

Returns the union of the specified ranges.

```
NSRange NSUnionRange (
   NSRange range1,
```

```
   NSRange range2
);
```

Return Value
A range covering all indices in and between `range1` and `range2`. If one range is completely contained in the other, the returned range is equal to the larger range.

Availability
Available in iOS 2.0 and later.

See Also
`NSIntersectionRange` (page 55)

Declared in
`NSRange.h`

NSUserName

Returns the logon name of the current user.

```
NSString * NSUserName (void);
```

Return Value
The logon name of the current user.

Availability
Available in iOS 2.0 and later.

See Also
`NSFullUserName` (page 52)
`NSHomeDirectory` (page 53)
`NSHomeDirectoryForUser` (page 54)

Declared in
`NSPathUtilities.h`

NSZoneCalloc

Allocates memory in a zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
void * NSZoneCalloc (
   NSZone *zone,
   NSUInteger numElems,
   NSUInteger byteSize
);
```

Discussion
Allocates enough memory from `zone` for `numElems` elements, each with a size `numBytes` bytes, and returns a pointer to the allocated memory. The memory is initialized with zeros. This function returns `NULL` if it was unable to allocate the requested memory.

Availability
Available in iOS 2.0 and later.

Declared in
`NSZone.h`

NSZoneFree

Deallocates a block of memory in the specified zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
void NSZoneFree (
   NSZone *zone,
   void *ptr
);
```

Discussion
Returns memory to the `zone` from which it was allocated. The standard C function `free` does the same, but spends time finding which zone the memory belongs to.

Availability
Available in iOS 2.0 and later.

Declared in
`NSZone.h`

NSZoneFromPointer

Gets the zone for a given block of memory. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
NSZone * NSZoneFromPointer (
   void *ptr
);
```

Return Value
The zone for the block of memory indicated by `pointer`, or `NULL` if the block was not allocated from a zone.

Discussion
`pointer` must be one that was returned by a prior call to an allocation function.

Availability
Available in iOS 2.0 and later.

Declared in
`NSZone.h`

NSZoneMalloc

Allocates memory in a zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
void * NSZoneMalloc (
   NSZone *zone,
   NSUInteger size
);
```

Discussion
Allocates `size` bytes in `zone` and returns a pointer to the allocated memory. This function returns `NULL` if it was unable to allocate the requested memory.

Availability
Available in iOS 2.0 and later.

Declared in
`NSZone.h`

NSZoneName

Returns the name of the specified zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
NSString * NSZoneName (
   NSZone *zone
);
```

Return Value
A string containing the name associated with `zone`. If `zone` is `nil`, the default zone is used. If no name is associated with `zone`, the returned string is empty.

Availability
Available in iOS 2.0 and later.

Declared in
`NSZone.h`

NSZoneRealloc

Allocates memory in a zone. (Deprecated. Zones are ignored on iOS and 64-bit runtime on OS X. You should not use zones in current development.)

```
void * NSZoneRealloc (
   NSZone *zone,
   void *ptr,
   NSUInteger size
);
```

Discussion
Changes the size of the block of memory pointed to by `ptr` to `size` bytes. It may allocate new memory to replace the old, in which case it moves the contents of the old memory block to the new block, up to a maximum of `size` bytes. `ptr` may be `NULL`. This function returns `NULL` if it was unable to allocate the requested memory.

Availability
Available in iOS 2.0 and later.

Declared in
`NSZone.h`

NS_DURING

Marks the start of the exception-handling domain.

```
NS_DURING
```

Discussion

The `NS_DURING` macro marks the start of the exception-handling domain for a section of code. (The `NS_HANDLER` (page 96)macro marks the end of the domain.) Within the exception-handling domain you can raise an exception, giving the local exception handler (or lower exception handlers) a chance to handle it.

Availability

Available in iOS 2.0 and later.

Related Sample Code

Quartz Composer SQLiteQuery

Declared in

`NSException.h`

NS_ENDHANDLER

Marks the end of the local event handler.

```
NS_ENDHANDLER
```

Discussion

The `NS_ENDHANDLER` marks the end of a section of code that is a local exception handler. (The `NS_HANDLER` (page 96)macros marks the beginning of this section.) If an exception is raised in the exception handling domain marked off by the `NS_DURING` (page 95) and `NS_HANDLER` (page 96), the local exception handler (if specified) is given a chance to handle the exception.

Availability

Available in iOS 2.0 and later.

Related Sample Code

Quartz Composer SQLiteQuery

Declared in

`NSException.h`

NS_HANDLER

Marks the end of the exception-handling domain and the start of the local exception handler.

```
NS_HANDLER
```

Discussion

The NS_HANDLER macro marks end of a section of code that is an exception-handling domain while at the same time marking the beginning of a section of code that is a local exception handler for that domain. (The NS_DURING (page 95) macro marks the beginning of the exception-handling domain; the NS_ENDHANDLER (page 96) marks the end of the local exception handler.) If an exception is raised in the exception-handling domain, the local exception handler is first given the chance to handle the exception before lower-level handlers are given a chance.

Availability

Available in iOS 2.0 and later.

Related Sample Code

Quartz Composer SQLiteQuery

Declared in

NSException.h

NS_VALUERETURN

Permits program control to exit from an exception-handling domain with a value of a specified type.

```
NS_VALUERETURN(val, type)
```

Parameters

`val`

A value to preserve beyond the exception-handling domain.

`type`

The type of the value specified in `val`.

Discussion

The NS_VALUERETURN macro returns program control to the caller out of the exception-handling domain—that is, a section of code between the NS_DURING (page 95) and NS_HANDLER (page 96) macros that might raise an exception. The specified value (of the specified type) is returned to the caller. The standard `return` statement does not work as expected in the exception-handling domain.

Availability

Available in iOS 2.0 and later.

Declared in

NSException.h

NS_VOIDRETURN

Permits program control to exit from an exception-handling domain.

```
NS_VOIDRETURN
```

Discussion

The `NS_VOIDRETURN` macro returns program control to the caller out of the exception-handling domain—that is, a section of code between the `NS_DURING` (page 95) and `NS_HANDLER` (page 96) macros that might raise an exception. The standard `return` statement does not work as expected in the exception-handling domain.

Availability

Available in iOS 2.0 and later.

Declared in

`NSException.h`

Document Revision History

This table describes the changes to *Foundation Functions Reference*.

Date	Notes
2013-08-08	Updated description of NSMapInsertIfAbsent to describe the actual return value.
2013-04-23	Added note about NSAssert condition evaluation in non-debug builds.
2012-09-19	Added the Core Foundation bridging functions for ARC.
2012-07-17	Updated for OS X v10.8.
2012-02-16	Updated file search function links and recommended using NSFileManager methods instead.
2011-07-07	Updated for OS X v10.7.
2010-03-24	Corrected the description of NSRectToCGRect to show the actual implementation.
2009-10-19	Clarified that a directory returned by NSSearchPathForDirectoriesInDomains() may not exist. Added usage warning to NSCopyObject().
2009-05-26	Corrected typographical errors.
2008-10-15	Corrected discussions of NSMapInsert and NSTemporaryDirectory; clarified behavior of NSPointInRect.
2008-09-09	Updated descriptions of the NSLocalizedString, NSLocalizedStringFromTable, NSLocalizedStringFromTableInBundle, and NSLocalizedStringWithDefaultValue for OS X v10.5.

Date	Notes
2008-07-11	Corrected descriptions of the functions NSRoundUpToMultipleOfPageSize and NSRoundDownToMultipleOfPageSize.
2008-02-08	Corrected minor typographical errors.
2007-07-23	Updated to include API introduced in OS X v10.5.
2006-09-05	Corrected broken link.
2006-06-28	Removed references to retired document.
2006-05-23	First publication of this content as a separate document.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010