Objective-C Runtime Reference

Developer

Contents

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Tables and Listings

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Objective-C Runtime Reference

Companion guide	Objective-C Runtime Programming Guide
Declared in	IONDRVLibraries.h NSObjCRuntime.h objc/message.h objc/objc-api.h objc/objc.h objc/runtime.h

Overview

This document describes the OS X Objective-C 2.0 runtime library support functions and data structures. The functions are implemented in the shared library found at `/usr/lib/libobjc.A.dylib`. This shared library provides support for the dynamic properties of the Objective-C language, and as such is linked to by all Objective-C applications.

This reference is useful primarily for developing bridge layers between Objective-C and other languages, or for low-level debugging. You typically do not need to use the Objective-C runtime library directly when programming in Objective-C.

The OS X implementation of the Objective-C runtime library is unique to the Mac. For other platforms, the GNU Compiler Collection provides a different implementation with a similar API. This document covers only the OS X implementation.

The low-level Objective-C runtime API is significantly updated in OS X version 10.5. Many functions and all existing data structures are replaced with new functions. The old functions and structures are deprecated in 32-bit and absent in 64-bit mode. The API constrains several values to 32-bit ints even in 64-bit mode—class count, protocol count, methods per class, ivars per class, arguments per method, sizeof(all arguments) per method, and class version number. In addition, the new Objective-C ABI (not described here) further constrains `sizeof(anInstance)` to 32 bits, and three other values to 24 bits—methods per class, ivars per class, and sizeof(a single ivar). Finally, the obsolete `NXHashTable` and `NXMapTable` are limited to 4 billion items.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

String encoding: All `char *` in the runtime API should be considered to have UTF-8 encoding.

"Deprecated" below means "deprecated in OS X version 10.5 for 32-bit code, and disallowed for 64-bit code."

Who Should Read This Document

The document is intended for readers who might be interested in learning about the Objective-C runtime.

Because this isn't a document about C, it assumes some prior acquaintance with that language. However, it doesn't have to be an extensive acquaintance.

Functions by Task

Working with Classes

`class_getName` (page 30)
Returns the name of a class.

`class_getSuperclass` (page 31)
Returns the superclass of a class.

`class_isMetaClass` (page 32)
Returns a Boolean value that indicates whether a class object is a metaclass.

`class_getInstanceSize` (page 27)
Returns the size of instances of a class.

`class_getInstanceVariable` (page 27)
Returns the `Ivar` for a specified instance variable of a given class.

`class_getClassVariable` (page 25)
Returns the `Ivar` for a specified class variable of a given class.

`class_addIvar` (page 17)
Adds a new instance variable to a class.

`class_copyIvarList` (page 21)
Describes the instance variables declared by a class.

`class_getIvarLayout` (page 28)
Returns a description of the `Ivar` layout for a given class.

`class_setIvarLayout` (page 35)
Sets the `Ivar` layout for a given class.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`class_getWeakIvarLayout` (page 32)
Returns a description of the layout of weak `Ivar`s for a given class.

`class_setWeakIvarLayout` (page 36)
Sets the layout for weak `Ivar`s for a given class.

`class_getProperty` (page 30)
Returns a property with a given name of a given class.

`class_copyPropertyList` (page 23)
Describes the properties declared by a class.

`class_addMethod` (page 18)
Adds a new method to a class with a given name and implementation.

`class_getInstanceMethod` (page 26)
Returns a specified instance method for a given class.

`class_getClassMethod` (page 25)
Returns a pointer to the data structure describing a given class method for a given class.

`class_copyMethodList` (page 22)
Describes the instance methods implemented by a class.

`class_replaceMethod` (page 33)
Replaces the implementation of a method for a given class.

`class_getMethodImplementation` (page 28)
Returns the function pointer that would be called if a particular message were sent to an instance of a class.

`class_getMethodImplementation_stret` (page 29)
Returns the function pointer that would be called if a particular message were sent to an instance of a class.

`class_respondsToSelector` (page 34)
Returns a Boolean value that indicates whether instances of a class respond to a particular selector.

`class_addProtocol` (page 20)
Adds a protocol to a class.

`class_addProperty` (page 19)
Adds a property to a class.

`class_replaceProperty` (page 34)
Replace a property of a class.

`class_conformsToProtocol` (page 21)
Returns a Boolean value that indicates whether a class conforms to a given protocol.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`class_copyProtocolList` (page 23)
Describes the protocols adopted by a class.

`class_getVersion` (page 31)
Returns the version number of a class definition.

`class_setVersion` (page 35)
Sets the version number of a class definition.

`objc_getFutureClass` (page 55)
Used by CoreFoundation's toll-free bridging.

`objc_setFutureClass` (page 65)
Used by CoreFoundation's toll-free bridging.

`class_setSuperclass` (page 97) Deprecated in OS X v10.5
Sets the superclass of a given class.

Adding Classes

`objc_allocateClassPair` (page 46)
Creates a new class and metaclass.

`objc_disposeClassPair` (page 51)
Destroys a class and its associated metaclass.

`objc_registerClassPair` (page 62)
Registers a class that was allocated using `objc_allocateClassPair`.

`objc_duplicateClass` (page 51)
Used by Foundation's Key-Value Observing.

Instantiating Classes

`class_createInstance` (page 24)
Creates an instance of a class, allocating memory for the class in the default malloc memory zone.

`objc_constructInstance` (page 47)
Creates an instance of a class at the specified location.

`objc_destructInstance` (page 50)
Destroys an instance of a class without freeing memory and removes any of its associated references.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Working with Instances

`object_copy` (page 66)
Returns a copy of a given object.

`object_dispose` (page 66)
Frees the memory occupied by a given object.

`object_setInstanceVariable` (page 70)
Changes the value of an instance variable of a class instance.

`object_getInstanceVariable` (page 68)
Obtains the value of an instance variable of a class instance.

`object_getIndexedIvars` (page 68)
Returns a pointer to any extra bytes allocated with a instance given object.

`object_getIvar` (page 69)
Reads the value of an instance variable in an object.

`object_setIvar` (page 71)
Sets the value of an instance variable in an object.

`object_getClassName` (page 67)
Returns the class name of a given object.

`object_getClass` (page 67)
Returns the class of an object.

`object_setClass` (page 70)
Sets the class of an object.

Obtaining Class Definitions

`objc_getClassList` (page 54)
Obtains the list of registered class definitions.

`objc_copyClassList` (page 48)
Creates and returns a list of pointers to all registered class definitions.

`objc_lookUpClass` (page 58)
Returns the class definition of a specified class.

`objc_getClass` (page 53)
Returns the class definition of a specified class.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`objc_getRequiredClass` (page 56)
Returns the class definition of a specified class.

`objc_getMetaClass` (page 55)
Returns the metaclass definition of a specified class.

Working with Instance Variables

`ivar_getName` (page 38)
Returns the name of an instance variable.

`ivar_getTypeEncoding` (page 39)
Returns the type string of an instance variable.

`ivar_getOffset` (page 39)
Returns the offset of an instance variable.

Associative References

`objc_setAssociatedObject` (page 64)
Sets an associated value for a given object using a given key and association policy.

`objc_getAssociatedObject` (page 52)
Returns the value associated with a given object for a given key.

`objc_removeAssociatedObjects` (page 63)
Removes all associations for a given object.

Sending Messages

When it encounters a method invocation, the compiler might generate a call to any of several functions to perform the actual message dispatch, depending on the receiver, the return value, and the arguments. You can use these functions to dynamically invoke methods from your own plain C code, or to use argument forms not permitted by NSObject's `perform...` methods. These functions are declared in `/usr/include/objc/objc-runtime.h`.

- `objc_msgSend` (page 58) sends a message with a simple return value to an instance of a class.
- `objc_msgSend_stret` (page 61) sends a message with a data-structure return value to an instance of a class.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

- `objc_msgSendSuper` (page 59) sends a message with a simple return value to the superclass of an instance of a class.
- `objc_msgSendSuper_stret` (page 60) sends a message with a data-structure return value to the superclass of an instance of a class.

`objc_msgSend` (page 58)
: Sends a message with a simple return value to an instance of a class.

`objc_msgSend_fpret` (page 61)
: Sends a message with a floating-point return value to an instance of a class.

`objc_msgSend_stret` (page 61)
: Sends a message with a data-structure return value to an instance of a class.

`objc_msgSendSuper` (page 59)
: Sends a message with a simple return value to the superclass of an instance of a class.

`objc_msgSendSuper_stret` (page 60)
: Sends a message with a data-structure return value to the superclass of an instance of a class.

Working with Methods

`method_invoke` (page 44)
: Calls the implementation of a specified method.

`method_invoke_stret` (page 45)
: Calls the implementation of a specified method that returns a data-structure.

`method_getName` (page 42)
: Returns the name of a method.

`method_getImplementation` (page 42)
: Returns the implementation of a method.

`method_getTypeEncoding` (page 44)
: Returns a string describing a method's parameter and return types.

`method_copyReturnType` (page 40)
: Returns a string describing a method's return type.

`method_copyArgumentType` (page 39)
: Returns a string describing a single parameter type of a method.

`method_getReturnType` (page 43)
: Returns by reference a string describing a method's return type.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`method_getNumberOfArguments` (page 43)
Returns the number of arguments accepted by a method.

`method_getArgumentType` (page 41)
Returns by reference a string describing a single parameter type of a method.

`method_getDescription` (page 41)
Returns a method description structure for a specified method.

`method_setImplementation` (page 45)
Sets the implementation of a method.

`method_exchangeImplementations` (page 40)
Exchanges the implementations of two methods.

Working with Libraries

`objc_copyImageNames` (page 49)
Returns the names of all the loaded Objective-C frameworks and dynamic libraries.

`class_getImageName` (page 26)
Returns the name of the dynamic library a class originated from.

`objc_copyClassNamesForImage` (page 49)
Returns the names of all the classes within a specified library or framework.

Working with Selectors

`sel_getName` (page 81)
Returns the name of the method specified by a given selector.

`sel_registerName` (page 83)
Registers a method with the Objective-C runtime system, maps the method name to a selector, and returns the selector value.

`sel_getUid` (page 81)
Registers a method name with the Objective-C runtime system.

`sel_isEqual` (page 82)
Returns a Boolean value that indicates whether two selectors are equal.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Working with Protocols

`objc_getProtocol` (page 56)
Returns a specified protocol.

`objc_copyProtocolList` (page 50)
Returns an array of all the protocols known to the runtime.

`objc_allocateProtocol` (page 47)
Creates a new protocol instance.

`objc_registerProtocol` (page 63)
Registers a newly created protocol with the Objective-C runtime.

`protocol_addMethodDescription` (page 73)
Adds a method to a protocol.

`protocol_addProtocol` (page 75)
Adds a registered protocol to another protocol that is under construction.

`protocol_addProperty` (page 74)
Adds a property to a protocol that is under construction.

`protocol_getName` (page 79)
Returns a the name of a protocol.

`protocol_isEqual` (page 80)
Returns a Boolean value that indicates whether two protocols are equal.

`protocol_copyMethodDescriptionList` (page 76)
Returns an array of method descriptions of methods meeting a given specification for a given protocol.

`protocol_getMethodDescription` (page 79)
Returns a method description structure for a specified method of a given protocol.

`protocol_copyPropertyList` (page 77)
Returns an array of the properties declared by a protocol.

`protocol_getProperty` (page 80)
Returns the specified property of a given protocol.

`protocol_copyProtocolList` (page 78)
Returns an array of the protocols adopted by a protocol.

`protocol_conformsToProtocol` (page 76)
Returns a Boolean value that indicates whether one protocol conforms to another protocol.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Working with Properties

`property_getName` (page 73)
Returns the name of a property.

`property_getAttributes` (page 73)
Returns the attribute string of a property.

`property_copyAttributeValue` (page 72)
Returns the value of a property attribute given the attribute name.

`property_copyAttributeList` (page 71)
Returns an array of property attributes for a given property.

Using Objective-C Language Features

`objc_enumerationMutation` (page 52)
Inserted by the compiler when a mutation is detected during a foreach iteration.

`objc_setEnumerationMutationHandler` (page 64)
Sets the current mutation handler.

`imp_implementationWithBlock` (page 37)
Creates a pointer to a function that calls the specified block when the method is called.

`imp_getBlock` (page 37)
Returns the block associated with an `IMP` that was created using `imp_implementationWithBlock` (page 37).

`imp_removeBlock` (page 38)
Disassociates a block from an `IMP` that was created using `imp_implementationWithBlock` (page 37), and releases the copy of the block that was created.

`objc_loadWeak` (page 57)
Loads the object referenced by a weak pointer and returns it.

`objc_storeWeak` (page 65)
Stores a new value in a `__weak` variable.

Functions

class_addIvar

Adds a new instance variable to a class.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

```
BOOL class_addIvar(Class cls, const char *name, size_t size, uint8_t alignment, const char *types)
```

Return Value

`YES` if the instance variable was added successfully, otherwise `NO` (for example, the class already contains an instance variable with that name).

Discussion

This function may only be called after `objc_allocateClassPair` (page 46) and before `objc_registerClassPair` (page 62). Adding an instance variable to an existing class is not supported.

The class must not be a metaclass. Adding an instance variable to a metaclass is not supported.

The instance variable's minimum alignment in bytes is `1<<align`. The minimum alignment of an instance variable depends on the ivar's type and the machine architecture. For variables of any pointer type, pass `log2(sizeof(pointer_type))`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

class_addMethod

Adds a new method to a class with a given name and implementation.

```
BOOL class_addMethod(Class cls, SEL name, IMP imp, const char *types)
```

Parameters

`cls`

The class to which to add a method.

`name`

A selector that specifies the name of the method being added.

`imp`

A function which is the implementation of the new method. The function must take at least two arguments—`self` and `_cmd`.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`types`

An array of characters that describe the types of the arguments to the method. For possible values, see *Objective-C Runtime Programming Guide* > "Type Encodings". Since the function must take at least two arguments—`self` and `_cmd`, the second and third characters must be "`@:`" (the first character is the return type).

Return Value

`YES` if the method was added successfully, otherwise `NO` (for example, the class already contains a method implementation with that name).

Discussion

`class_addMethod` will add an override of a superclass's implementation, but will not replace an existing implementation in this class. To change an existing implementation, use `method_setImplementation` (page 45).

An Objective-C method is simply a C function that take at least two arguments—`self` and `_cmd`. For example, given the following function:

```
void myMethodIMP(id self, SEL _cmd)
{
    // implementation ....
}
```

you can dynamically add it to a class as a method (called `resolveThisMethodDynamically`) like this:

```
class_addMethod([self class], @selector(resolveThisMethodDynamically), (IMP)
myMethodIMP, "v@:");
```

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

class_addProperty

Adds a property to a class.

```
BOOL class_addProperty(Class cls, const char *name, const objc_property_attribute_t
*attributes, unsigned int attributeCount)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Parameters

`cls`

The class to modify.

`name`

The name of the property.

`attributes`

An array of property attributes.

`attributeCount`

The number of attributes in `attributes`.

Return Value

`YES` if the property was added successfully; otherwise `NO` (for example, this function returns `NO` if the class already has that property).

Availability

Available in OS X v10.7 and later.

See Also

`class_replaceProperty` (page 34)

Declared in

`objc/runtime.h`

class_addProtocol

Adds a protocol to a class.

```
BOOL class_addProtocol(Class cls, Protocol *protocol)
```

Parameters

`cls`

The class to modify.

`outCount`

The protocol to add to `cls`.

Return Value

`YES` if the protocol was added successfully, otherwise `NO` (for example, the class already conforms to that protocol).

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_conformsToProtocol

Returns a Boolean value that indicates whether a class conforms to a given protocol.

```
BOOL class_conformsToProtocol(Class cls, Protocol *protocol)
```

Parameters

`cls`

The class you want to inspect.

`protocol`

A protocol.

Return Value
`YES` if `cls` conforms to `protocol`, otherwise `NO`.

Discussion
You should usually use `NSObject`'s `conformsToProtocol:` method instead of this function.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_copyIvarList

Describes the instance variables declared by a class.

```
Ivar * class_copyIvarList(Class cls, unsigned int *outCount)
```

Parameters

`cls`

The class to inspect.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`outCount`
: On return, contains the length of the returned array. If `outCount` is `NULL`, the length is not returned.

Return Value
An array of pointers of type Ivar describing the instance variables declared by the class. Any instance variables declared by superclasses are not included. The array contains `*outCount` pointers followed by a `NULL` terminator. You must free the array with `free()`.

If the class declares no instance variables, or `cls` is `Nil`, `NULL` is returned and `*outCount` is `0`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_copyMethodList

Describes the instance methods implemented by a class.

```
Method * class_copyMethodList(Class cls, unsigned int *outCount)
```

Parameters
`cls`
: The class you want to inspect.

`outCount`
: On return, contains the length of the returned array. If `outCount` is `NULL`, the length is not returned.

Return Value
An array of pointers of type `Method` describing the instance methods implemented by the class—any instance methods implemented by superclasses are not included. The array contains `*outCount` pointers followed by a `NULL` terminator. You must free the array with `free()`.

If `cls` implements no instance methods, or `cls` is `Nil`, returns `NULL` and `*outCount` is `0`.

Discussion
To get the class methods of a class, use `class_copyMethodList(object_getClass(cls), &count)`.

To get the implementations of methods that may be implemented by superclasses, use `class_getInstanceMethod` (page 26) or `class_getClassMethod` (page 25).

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_copyPropertyList

Describes the properties declared by a class.

```
objc_property_t * class_copyPropertyList(Class cls, unsigned int *outCount)
```

Parameters

`cls`

The class you want to inspect.

`outCount`

On return, contains the length of the returned array. If `outCount` is `NULL`, the length is not returned.

Return Value
An array of pointers of type `objc_property_t` describing the properties declared by the class. Any properties declared by superclasses are not included. The array contains `*outCount` pointers followed by a `NULL` terminator. You must free the array with `free()`.

If `cls` declares no properties, or `cls` is `Nil`, returns `NULL` and `*outCount` is `0`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_copyProtocolList

Describes the protocols adopted by a class.

```
Protocol ** class_copyProtocolList(Class cls, unsigned int *outCount)
```

Parameters

`cls`

The class you want to inspect.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`outCount`

On return, contains the length of the returned array. If `outCount` is `NULL`, the length is not returned.

Return Value

An array of pointers of type `Protocol*` describing the protocols adopted by the class. Any protocols adopted by superclasses or other protocols are not included. The array contains `*outCount` pointers followed by a `NULL` terminator. You must free the array with `free()`.

If `cls` adopts no protocols, or `cls` is `Nil`, returns `NULL` and `*outCount` is `0`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

class_createInstance

Creates an instance of a class, allocating memory for the class in the default malloc memory zone.

```
id class_createInstance(Class cls, size_t extraBytes)
```

Parameters

`cls`

The class that you want to allocate an instance of.

`extraBytes`

An integer indicating the number of extra bytes to allocate. The additional bytes can be used to store additional instance variables beyond those defined in the class definition.

Return Value

An instance of the class `cls`.

Availability

Available in OS X v10.0 and later.

See Also

`objc_constructInstance` (page 47)

Declared in

`objc/runtime.h`

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

class_getClassMethod

Returns a pointer to the data structure describing a given class method for a given class.

```
Method class_getClassMethod(Class aClass, SEL aSelector)
```

Parameters

`aClass`

A pointer to a class definition. Pass the class that contains the method you want to retrieve.

`aSelector`

A pointer of type `SEL` (page 85). Pass the selector of the method you want to retrieve.

Return Value

A pointer to the `Method` (page 84) data structure that corresponds to the implementation of the selector specified by `aSelector` for the class specified by `aClass`, or `NULL` if the specified class or its superclasses do not contain a class method with the specified selector.

Discussion

Note that this function searches superclasses for implementations, whereas `class_copyMethodList` (page 22) does not.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/runtime.h`

class_getClassVariable

Returns the `Ivar` for a specified class variable of a given class.

```
Ivar class_getClassVariable(Class cls, const char* name)
```

Parameters

`cls`

The class definition whose class variable you wish to obtain.

`name`

The name of the class variable definition to obtain.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Return Value
A pointer to an `Ivar` (page 84) data structure containing information about the class variable specified by `name`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_getImageName

Returns the name of the dynamic library a class originated from.

```
const char *class_getImageName(Class cls)
```

Parameters
`cls`
The class you are inquiring about.

Return Value
A C string representing the name of the library containing the `cls` class.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_getInstanceMethod

Returns a specified instance method for a given class.

```
Method class_getInstanceMethod(Class aClass, SEL aSelector)
```

Parameters
`aClass`
The class you want to inspect.

`aSelector`
The selector of the method you want to retrieve.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Return Value
The method that corresponds to the implementation of the selector specified by `aSelector` for the class specified by `aClass`, or `NULL` if the specified class or its superclasses do not contain an instance method with the specified selector.

Discussion
Note that this function searches superclasses for implementations, whereas `class_copyMethodList` (page 22) does not.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

class_getInstanceSize

Returns the size of instances of a class.

```
size_t class_getInstanceSize(Class cls)
```

Parameters
`cls`
A class object.

Return Value
The size in bytes of instances of the class `cls`, or `0` if `cls` is `Nil`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_getInstanceVariable

Returns the `Ivar` for a specified instance variable of a given class.

```
Ivar class_getInstanceVariable(Class cls, const char* name)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Parameters

`cls`

The class whose instance variable you wish to obtain.

`name`

The name of the instance variable definition to obtain.

Return Value

A pointer to an `Ivar` (page 84) data structure containing information about the instance variable specified by `name`.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/runtime.h`

class_getIvarLayout

Returns a description of the `Ivar` layout for a given class.

```
const char *class_getIvarLayout(Class cls)
```

Parameters

`cls`

The class to inspect.

Return Value

A description of the `Ivar` layout for `cls`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

class_getMethodImplementation

Returns the function pointer that would be called if a particular message were sent to an instance of a class.

```
IMP class_getMethodImplementation(Class cls, SEL name)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Parameters

`cls`

The class you want to inspect.

`name`

A selector.

Return Value

The function pointer that would be called if `[object name]` were called with an instance of the class, or `NULL` if `cls` is `Nil`.

Discussion

`class_getMethodImplementation` may be faster than `method_getImplementation(class_getInstanceMethod(cls, name))`.

The function pointer returned may be a function internal to the runtime instead of an actual method implementation. For example, if instances of the class do not respond to the selector, the function pointer returned will be part of the runtime's message forwarding machinery.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

class_getMethodImplementation_stret

Returns the function pointer that would be called if a particular message were sent to an instance of a class.

```
IMP class_getMethodImplementation_stret(Class cls, SEL name)
```

Parameters

`cls`

The class you want to inspect.

`name`

A selector.

Return Value

The function pointer that would be called if `[object name]` were called with an instance of the class, or `NULL` if `cls` is `Nil`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_getName

Returns the name of a class.

```
const char * class_getName(Class cls)
```

Parameters
`cls`
A class object.

Return Value
The name of the class, or the empty string if `cls` is `Nil`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_getProperty

Returns a property with a given name of a given class.

```
objc_property_t class_getProperty(Class cls, const char *name)
```

Return Value
A pointer of type `objc_property_t` describing the property, or `NULL` if the class does not declare a property with that name, or `NULL` if `cls` is `Nil`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

class_getSuperclass

Returns the superclass of a class.

```
Class class_getSuperclass(Class cls)
```

Parameters

`cls`

A class object.

Return Value

The superclass of the class, or `Nil` if `cls` is a root class, or `Nil` if `cls` is `Nil`.

Discussion

You should usually use `NSObject`'s `superclass` method instead of this function.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

class_getVersion

Returns the version number of a class definition.

```
int class_getVersion(Class theClass)
```

Parameters

`theClass`

A pointer to an `Class` (page 83) data structure. Pass the class definition for which you wish to obtain the version.

Return Value

An integer indicating the version number of the class definition.

Discussion

You can use the version number of the class definition to provide versioning of the interface that your class represents to other classes. This is especially useful for object serialization (that is, archiving of the object in a flattened form), where it is important to recognize changes to the layout of the instance variables in different class-definition versions.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Classes derived from the Foundation framework `NSObject` class can obtain the class-definition version number using the `getVersion` class method, which is implemented using the `class_getVersion` function.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

class_getWeakIvarLayout

Returns a description of the layout of weak `Ivar`s for a given class.

```
const char *class_getWeakIvarLayout(Class cls)
```

Parameters
`cls`
The class to inspect.

Return Value
A description of the layout of the weak `Ivar`s for `cls`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_isMetaClass

Returns a Boolean value that indicates whether a class object is a metaclass.

```
BOOL class_isMetaClass(Class cls)
```

Parameters
`cls`
A class object.

Return Value
`YES` if `cls` is a metaclass, `NO` if `cls` is a non-meta class, `NO` if `cls` is `Nil`.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_replaceMethod

Replaces the implementation of a method for a given class.

```
IMP class_replaceMethod(Class cls, SEL name, IMP imp, const char *types)
```

Parameters
`cls`

The class you want to modify.

`name`

A selector that identifies the method whose implementation you want to replace.

`imp`

The new implementation for the method identified by `name` for the class identified by `cls`.

`types`

An array of characters that describe the types of the arguments to the method. For possible values, see *Objective-C Runtime Programming Guide* > "Type Encodings". Since the function must take at least two arguments—`self` and `_cmd`, the second and third characters must be "`@:`" (the first character is the return type).

Return Value
The previous implementation of the method identified by `name` for the class identified by `cls`.

Discussion
This function behaves in two different ways:

- If the method identified by `name` does not yet exist, it is added as if `class_addMethod` (page 18) were called. The type encoding specified by `types` is used as given.
- If the method identified by `name` does exist, its IMP is replaced as if `method_setImplementation` (page 45) were called. The type encoding specified by `types` is ignored.

Availability
Available in OS X v10.5 and later.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Declared in
`objc/runtime.h`

class_replaceProperty

Replace a property of a class.

```
void class_replaceProperty(Class cls, const char *name, const objc_property_attribute_t
*attributes, unsigned int attributeCount)
```

Parameters

`cls`

The class to modify.

`name`

The name of the property.

`attributes`

An array of property attributes.

`attributeCount`

The number of attributes in `attributes`.

Availability
Available in OS X v10.7 and later.

See Also
`class_addProperty` (page 19)

Declared in
`objc/runtime.h`

class_respondsToSelector

Returns a Boolean value that indicates whether instances of a class respond to a particular selector.

```
BOOL class_respondsToSelector(Class cls, SEL sel)
```

Parameters

`cls`

The class you want to inspect.

`sel`
A selector.

Return Value
`YES` if instances of the class respond to the selector, otherwise `NO`.

Discussion
You should usually use `NSObject`'s `respondsToSelector:` or `instancesRespondToSelector:` methods instead of this function.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_setIvarLayout

Sets the `Ivar` layout for a given class.

```
void class_setIvarLayout(Class cls, const char *layout)
```

Parameters
`cls`
The class to modify.

`layout`
The layout of the `Ivar`s for `cls`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

class_setVersion

Sets the version number of a class definition.

```
void class_setVersion(Class theClass, int version)
```

Parameters

`theClass`

A pointer to an `Class` (page 83) data structure. Pass the class definition for which you wish to set the version.

`version`

An integer. Pass the new version number of the class definition.

Discussion

You can use the version number of the class definition to provide versioning of the interface that your class represents to other classes. This is especially useful for object serialization (that is, archiving of the object in a flattened form), where it is important to recognize changes to the layout of the instance variables in different class-definition versions.

Classes derived from the Foundation framework `NSObject` class can set the class-definition version number using the `setVersion:` class method, which is implemented using the `class_setVersion` function.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/runtime.h`

class_setWeakIvarLayout

Sets the layout for weak `Ivar`s for a given class.

```
void class_setWeakIvarLayout(Class cls, const char *layout)
```

Parameters

`cls`

The class to modify.

`layout`

The layout of the weak `Ivar`s for `cls`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

imp_getBlock

Returns the block associated with an `IMP` that was created using `imp_implementationWithBlock` (page 37).

```
id imp_getBlock(IMP anImp)
```

Parameters

`anImp`

The `IMP` that calls this block.

Return Value

The block called by `anImp`.

Availability

Available in OS X v10.7 and later.

See Also

`imp_implementationWithBlock` (page 37)

Declared in

`objc/runtime.h`

imp_implementationWithBlock

Creates a pointer to a function that calls the specified block when the method is called.

```
IMP imp_implementationWithBlock(id block)
```

Parameters

`block`

The block that implements this method. The signature of `block` should be `method_return_type ^(id self, self, method_args …)`. The selector of the method is not available to `block`. `block` is copied with `Block_copy()`.

Return Value

The `IMP` (page 85) that calls `block`. You must dispose of the returned `IMP` using the function.

Availability

Available in OS X v10.7 and later.

See Also

`imp_getBlock` (page 37)

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Declared in
objc/runtime.h

imp_removeBlock

Disassociates a block from an IMP that was created using imp_implementationWithBlock (page 37), and releases the copy of the block that was created.

```
BOOL imp_removeBlock(IMP anImp)
```

Parameters
anImp
An IMP that was created using the imp_implementationWithBlock (page 37) function.

Return Value
YES if the block was released successfully; otherwise, NO (for example, the function returns NO if the block was not used to create anImp previously).

Availability
Available in OS X v10.7 and later.

See Also
imp_implementationWithBlock (page 37)

Declared in
objc/runtime.h

ivar_getName

Returns the name of an instance variable.

```
const char * ivar_getName(Ivar ivar)
```

Return Value
A C string containing the instance variable's name.

Availability
Available in OS X v10.5 and later.

Declared in
objc/runtime.h

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

ivar_getOffset

Returns the offset of an instance variable.

```
ptrdiff_t ivar_getOffset(Ivar ivar)
```

Discussion
For instance variables of type `id` or other object types, call `object_getIvar` (page 69) and `object_setIvar` (page 71) instead of using this offset to access the instance variable data directly.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

ivar_getTypeEncoding

Returns the type string of an instance variable.

```
const char * ivar_getTypeEncoding(Ivar ivar)
```

Return Value
A C string containing the instance variable's type encoding.

Discussion
For possible values, see *Objective-C Runtime Programming Guide* > "Type Encodings".

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

method_copyArgumentType

Returns a string describing a single parameter type of a method.

```
char * method_copyArgumentType(Method method, unsigned int index)
```

Parameters

`method`

The method to inspect.

`index`

The index of the parameter to inspect.

Return Value

A C string describing the type of the parameter at index `index`, or `NULL` if `method` has no parameter index `index`. You must free the string with `free()`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

method_copyReturnType

Returns a string describing a method's return type.

```
char * method_copyReturnType(Method method)
```

Parameters

`method`

The method to inspect.

Return Value

A C string describing the return type. You must free the string with `free()`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

method_exchangeImplementations

Exchanges the implementations of two methods.

```
void method_exchangeImplementations(Method m1, Method m2)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Discussion

This is an atomic version of the following:

```
IMP imp1 = method_getImplementation(m1);
IMP imp2 = method_getImplementation(m2);
method_setImplementation(m1, imp2);
method_setImplementation(m2, imp1);
```

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

method_getArgumentType

Returns by reference a string describing a single parameter type of a method.

```
void method_getArgumentType(Method method, unsigned int index, char *dst, size_t dst_len)
```

Discussion

The parameter type string is copied to `dst`. `dst` is filled as if `strncpy(dst, parameter_type, dst_len)` were called. If the method contains no parameter with that index, `dst` is filled as if `strncpy(dst, "", dst_len)` were called.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

method_getDescription

Returns a method description structure for a specified method.

```
struct objc_method_description *method_getDescription(Method m)
```

Parameters

`m`

The method you want to inquire about.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Return Value
An `objc_method_description` structure that describes the method specified by `m`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

method_getImplementation

Returns the implementation of a method.

```
IMP method_getImplementation(Method method)
```

Parameters
`method`
The method to inspect.

Return Value
A function pointer of type `IMP`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

method_getName

Returns the name of a method.

```
SEL method_getName(Method method)
```

Parameters
`method`
The method to inspect.

Return Value
A pointer of type SEL.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Discussion
To get the method name as a C string, call `sel_getName(method_getName(method))`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

method_getNumberOfArguments

Returns the number of arguments accepted by a method.

```
unsigned method_getNumberOfArguments(Method method)
```

Parameters
`method`

A pointer to a `Method` (page 84) data structure. Pass the method in question.

Return Value
An integer containing the number of arguments accepted by the given method.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

method_getReturnType

Returns by reference a string describing a method's return type.

```
void method_getReturnType(Method method, char *dst, size_t dst_len)
```

Discussion
The method's return type string is copied to `dst`. `dst` is filled as if `strncpy(dst, parameter_type, dst_len)` were called.

Availability
Available in OS X v10.5 and later.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Declared in
`objc/runtime.h`

method_getTypeEncoding

Returns a string describing a method's parameter and return types.

```
const char * method_getTypeEncoding(Method method)
```

Parameters
`method`
The method to inspect.

Return Value
A C string. The string may be `NULL`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

method_invoke

Calls the implementation of a specified method.

```
id method_invoke(id receiver, Method m, ...)
```

Parameters
`receiver`
A pointer to the instance of the class that you want to invoke the method on. This value must not be `nil`.

`m`
The method whose implementation you want to call.

`...`
A variable argument list containing the arguments to the method.

Return Value
The return value of the method.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Discussion

Using this function to call the implementation of a method is faster than calling `method_getImplementation` (page 42) and `method_getName` (page 42).

Availability

Available in OS X v10.5 and later.

Declared in

`objc/message.h`

method_invoke_stret

Calls the implementation of a specified method that returns a data-structure.

```
void method_invoke_stret(id receiver, Method m, ...)
```

Parameters

`receiver`

A pointer to the instance of the class that you want to invoke the method on. This value must not be `nil`.

`m`

The method whose implementation you want to call.

`...`

A variable argument list containing the arguments to the method.

Discussion

Using this function to call the implementation of a method is faster than calling `method_getImplementation` (page 42) and `method_getName` (page 42).

Availability

Available in OS X v10.5 and later.

Declared in

`objc/message.h`

method_setImplementation

Sets the implementation of a method.

```
IMP method_setImplementation(Method method, IMP imp)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Return Value
The previous implementation of the method.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

objc_allocateClassPair

Creates a new class and metaclass.

```
objc_allocateClassPair(Class superclass, const char *name, size_t extraBytes)
```

Parameters
`superclass`
The class to use as the new class's superclass, or `Nil` to create a new root class.

`name`
The string to use as the new class's name. The string will be copied.

`extraBytes`
The number of bytes to allocate for indexed ivars at the end of the class and metaclass objects. This should usually be `0`.

Return Value
The new class, or `Nil` if the class could not be created (for example, the desired name is already in use).

Discussion
You can get a pointer to the new metaclass by calling `object_getClass(newClass)`.

To create a new class, start by calling `objc_allocateClassPair`. Then set the class's attributes with functions like `class_addMethod` (page 18) and `class_addIvar` (page 17). When you are done building the class, call `objc_registerClassPair` (page 62). The new class is now ready for use.

Instance methods and instance variables should be added to the class itself. Class methods should be added to the metaclass.

Availability
Available in OS X v10.5 and later.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

See Also
objc_disposeClassPair (page 51)

Declared in
objc/runtime.h

objc_allocateProtocol

Creates a new protocol instance.

```
Protocol *objc_allocateProtocol(const char *name)
```

Parameters
name

The name of the protocol you want to create.

Return Value
A new protocol instance or nil if a protocol with the same name as name already exists.

Discussion
You must register the returned protocol instance with the objc_registerProtocol (page 63) function before you can use it.

There is no dispose method associated with this function.

Availability
Available in OS X v10.7 and later.

See Also
objc_registerProtocol (page 63)

Declared in
objc/runtime.h

objc_constructInstance

Creates an instance of a class at the specified location.

```
id objc_constructInstance(Class cls, void *bytes)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Parameters

`cls`

The class that you want to allocate an instance of.

`bytes`

The location at which to allocate an instance of the `cls` class. `bytes` myst point to at least `class_getInstanceSize(cls)` bytes of well-aligned, zero-filled memory.

Return Value

An instance of the class `cls` at `bytes`, if successful; otherwise `nil` (for example, if `cls` or `bytes` are themselves `nil`).

Availability

Available in OS X v10.6 and later.

See Also

`class_createInstance` (page 24)

Declared in

`objc/runtime.h`

objc_copyClassList

Creates and returns a list of pointers to all registered class definitions.

```
Class *objc_copyClassList(unsigned int *outCount)
```

Parameters

`outCount`

An integer pointer used to store the number of classes returned by this function in the list. This parameter may be `nil`.

Return Value

A `nil` terminated array of classes. You must free the array with `free()`.

Availability

Available in OS X v10.7 and later.

See Also

`objc_getClassList` (page 54)

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Declared in
`objc/runtime.h`

objc_copyClassNamesForImage

Returns the names of all the classes within a specified library or framework.

```
const char **objc_copyClassNamesForImage(const char *image, unsigned int *outCount)
```

Parameters
`image`
The library or framework you are inquiring about.

`outCount`
The number of class names in the returned array.

Return Value
An array of C strings representing all of the class names within the specified library or framework.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

objc_copyImageNames

Returns the names of all the loaded Objective-C frameworks and dynamic libraries.

```
const char **objc_copyImageNames(unsigned int *outCount)
```

Parameters
`outCount`
The number of names in the returned array.

Return Value
An array of C strings representing the names of all the loaded Objective-C frameworks and dynamic libraries.

Availability
Available in OS X v10.5 and later.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Declared in
`objc/runtime.h`

objc_copyProtocolList

Returns an array of all the protocols known to the runtime.

```
Protocol **objc_copyProtocolList(unsigned int *outCount)
```

Parameters
`outCount`
Upon return, contains the number of protocols in the returned array.

Return Value
A C array of all the protocols known to the runtime. The array contains `*outCount` pointers followed by a `NULL` terminator. You must free the list with `free()`.

Discussion
This function acquires the runtime lock.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

objc_destructInstance

Destroys an instance of a class without freeing memory and removes any of its associated references.

```
void objc_destructInstance(id obj)
```

Discussion
This method does nothing if `obj` is `nil`.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Important: The garbage collector does not call this function. As a result, if you edit this function, you should also edit finalize. That said, Core Foundation and other clients do call this function under garbage collection.

Availability
Available in OS X v10.6 and later.

See Also
`objc_constructInstance` (page 47)

Declared in
`objc/runtime.h`

objc_disposeClassPair

Destroys a class and its associated metaclass.

```
void objc_disposeClassPair(Class cls)
```

Parameters
`cls`

The class to be destroyed. This class must have been allocated using `objc_allocateClassPair` (page 46).

Discussion
Do not call this function if instances of the `cls` class or any subclass exist.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

objc_duplicateClass

Used by Foundation's Key-Value Observing.

```
objc_duplicateClass
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Special Considerations
Do not call this function yourself.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

objc_enumerationMutation

Inserted by the compiler when a mutation is detected during a foreach iteration.

```
void objc_enumerationMutation(id obj)
```

Parameters
`obj`

The object being mutated.

Discussion
The compiler inserts this function when it detects that an object is mutated during a foreach iteration. The function is called when a mutation occurs, and the enumeration mutation handler is enacted if it is set up (via the `objc_setEnumerationMutationHandler` (page 64) function). If the handler is not set up, a fatal error occurs.

Availability
Available in OS X v10.5 and later.

See Also
`objc_setEnumerationMutationHandler` (page 64)

Declared in
`objc/runtime.h`

objc_getAssociatedObject

Returns the value associated with a given object for a given key.

```
id objc_getAssociatedObject(id object, void *key)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Parameters

`object`

The source object for the association.

`key`

The key for the association.

Return Value

The value associated with the key `key` for `object`.

Availability

Available in OS X v10.6 and later.

See Also

`objc_setAssociatedObject` (page 64)

Declared in

`objc/runtime.h`

objc_getClass

Returns the class definition of a specified class.

```
id objc_getClass(const char *name)
```

Parameters

`name`

The name of the class to look up.

Return Value

The Class object for the named class, or `nil` if the class is not registered with the Objective-C runtime.

Discussion

`objc_getClass` is different from `objc_lookUpClass` (page 58) in that if the class is not registered, `objc_getClass` calls the class handler callback and then checks a second time to see whether the class is registered. `objc_lookUpClass` (page 58) does not call the class handler callback.

Special Considerations

Earlier implementations of this function (prior to OS X v10.0) terminate the program if the class does not exist.

Availability

Available in OS X v10.0 and later.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Declared in
`objc/runtime.h`

objc_getClassList

Obtains the list of registered class definitions.

```
int objc_getClassList(Class *buffer, int bufferLen)
```

Parameters

`buffer`

An array of `Class` values. On output, each `Class` value points to one class definition, up to either `bufferLen` or the total number of registered classes, whichever is less. You can pass `NULL` to obtain the total number of registered class definitions without actually retrieving any class definitions.

`bufferLen`

An integer value. Pass the number of pointers for which you have allocated space in `buffer`. On return, this function fills in only this number of elements. If this number is less than the number of registered classes, this function returns an arbitrary subset of the registered classes.

Return Value
An integer value indicating the total number of registered classes.

Discussion
The Objective-C runtime library automatically registers all the classes defined in your source code. You can create class definitions at runtime and register them with the `objc_addClass` function.

Listing 1 demonstrates how to use this function to retrieve all the class definitions that have been registered with the Objective-C runtime in the current process.

Listing 1 Using objc_getClassList

```
int numClasses;
Class * classes = NULL;

classes = NULL;
numClasses = objc_getClassList(NULL, 0);

if (numClasses > 0 )
{
```

```
    classes = malloc(sizeof(Class) * numClasses);
    numClasses = objc_getClassList(classes, numClasses);
    free(classes);
}
```

Special Considerations
You cannot assume that class objects you get from this function are classes that inherit from `NSObject`, so you cannot safely call any methods on such classes without detecting that the method is implemented first.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

objc_getFutureClass

Used by CoreFoundation's toll-free bridging.

```
Class objc_getFutureClass(const char *name)
```

Special Considerations
Do not call this function yourself.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

objc_getMetaClass

Returns the metaclass definition of a specified class.

```
id objc_getMetaClass(const char *name)
```

Parameters
`name`

The name of the class to look up.

Return Value
The `Class` object for the metaclass of the named class, or `nil` if the class is not registered with the Objective-C runtime.

Discussion
If the definition for the named class is not registered, this function calls the class handler callback and then checks a second time to see if the class is registered. However, every class definition must have a valid metaclass definition, and so the metaclass definition is always returned, whether it's valid or not.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

objc_getProtocol

Returns a specified protocol.

```
Protocol *objc_getProtocol(const char *name)
```

Parameters
`name`
The name of a protocol.

Return Value
The protocol named `name`, or `NULL` if no protocol named `name` could be found.

Discussion
This function acquires the runtime lock.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

objc_getRequiredClass

Returns the class definition of a specified class.

```
id objc_getRequiredClass(const char *name)
```

Parameters

`name`

The name of the class to look up.

Return Value

The Class object for the named class.

Discussion

This function is the same as `objc_getClass` (page 53), but kills the process if the class is not found.

This function is used by ZeroLink, where failing to find a class would be a compile-time link error without ZeroLink.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/runtime.h`

objc_loadWeak

Loads the object referenced by a weak pointer and returns it.

```
id objc_loadWeak(id *location)
```

Parameters

`location`

The address of the weak pointer.

Return Value

The object pointed to by `location`, or `nil` if `location` is `nil`.

Discussion

This function loads the object referenced by a weak pointer and returns it after retaining and autoreleasing the object. As a result, the object stays alive long enough for the caller to use it. This function is typically used anywhere a `__weak` variable is used in an expression.

Availability

Available in OS X v10.7 and later.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Declared in
`objc/runtime.h`

objc_lookUpClass

Returns the class definition of a specified class.

```
id objc_lookUpClass(const char *name)
```

Parameters
`name`
The name of the class to look up.

Return Value
The Class object for the named class, or `nil` if the class is not registered with the Objective-C runtime.

Discussion
`objc_getClass` (page 53) is different from this function in that if the class is not registered, `objc_getClass` (page 53) calls the class handler callback and then checks a second time to see whether the class is registered. This function does not call the class handler callback.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

objc_msgSend

Sends a message with a simple return value to an instance of a class.

```
id objc_msgSend(id self, SEL op, ...)
```

Parameters
`self`
A pointer that points to the instance of the class that is to receive the message.
`op`
The selector of the method that handles the message.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`...`
A variable argument list containing the arguments to the method.

Return Value
The return value of the method.

Discussion
When it encounters a method call, the compiler generates a call to one of the functions `objc_msgSend`, `objc_msgSend_stret`, `objc_msgSendSuper`, or `objc_msgSendSuper_stret`. Messages sent to an object's superclass (using the `super` keyword) are sent using `objc_msgSendSuper`; other messages are sent using `objc_msgSend`. Methods that have data structures as return values are sent using `objc_msgSendSuper_stret` and `objc_msgSend_stret`.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/message.h`

objc_msgSendSuper

Sends a message with a simple return value to the superclass of an instance of a class.

```
id objc_msgSendSuper(struct objc_super *super, SEL op,  ...)
```

Parameters
`super`
A pointer to an `objc_super` (page 90) data structure. Pass values identifying the context the message was sent to, including the instance of the class that is to receive the message and the superclass at which to start searching for the method implementation.

`op`
A pointer of type `SEL` (page 85). Pass the selector of the method that will handle the message.

`...`
A variable argument list containing the arguments to the method.

Return Value
The return value of the method identified by `op`.

Discussion

When it encounters a method call, the compiler generates a call to one of the functions `objc_msgSend`, `objc_msgSend_stret`, `objc_msgSendSuper`, or `objc_msgSendSuper_stret`. Messages sent to an object's superclass (using the `super` keyword) are sent using `objc_msgSendSuper`; other messages are sent using `objc_msgSend`. Methods that have data structures as return values are sent using `objc_msgSendSuper_stret` and `objc_msgSend_stret`.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/message.h`

objc_msgSendSuper_stret

Sends a message with a data-structure return value to the superclass of an instance of a class.

```
void objc_msgSendSuper_stret(struct objc_super *super, SEL op, ...)
```

Parameters

`super`

A pointer to an `objc_super` (page 90) data structure. Pass values identifying the context the message was sent to, including the instance of the class that is to receive the message and the superclass at which to start searching for the method implementation.

`op`

A pointer of type `SEL` (page 85). Pass the selector of the method.

`...`

A variable argument list containing the arguments to the method.

Discussion

When it encounters a method call, the compiler generates a call to one of the functions `objc_msgSend`, `objc_msgSend_stret`, `objc_msgSendSuper`, or `objc_msgSendSuper_stret`. Messages sent to an object's superclass (using the `super` keyword) are sent using `objc_msgSendSuper`; other messages are sent using `objc_msgSend`. Methods that have data structures as return values are sent using `objc_msgSendSuper_stret` and `objc_msgSend_stret`.

Availability

Available in OS X v10.0 and later.

Declared in
`objc/message.h`

objc_msgSend_fpret

Sends a message with a floating-point return value to an instance of a class.

```
double objc_msgSend_fpret(id self, SEL op, ...)
```

Parameters

`self`

A pointer that points to the instance of the class that is to receive the message.

`op`

The selector of the method that handles the message.

`...`

A variable argument list containing the arguments to the method.

Discussion
On the i386 platform, the ABI for functions returning a floating-point value is incompatible with that for functions returning an integral type. On the i386 platform, therefore, you *must* use `objc_msgSend_fpret` for functions that for functions returning non-integral type. For `float` or `long double` return types, cast the function to an appropriate function pointer type first.

This function is not used on the PPC or PPC64 platforms.

Availability
Available in OS X v10.4 and later.

Declared in
`objc/message.h`

objc_msgSend_stret

Sends a message with a data-structure return value to an instance of a class.

```
void objc_msgSend_stret(void * stretAddr, id theReceiver, SEL theSelector,  ...)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Parameters

`stretAddr`

On input, a pointer that points to a block of memory large enough to contain the return value of the method. On output, contains the return value of the method.

`theReceiver`

A pointer to the instance of the class that is to receive the message.

`theSelector`

A pointer of type `SEL` (page 85). Pass the selector of the method that handles the message.

`...`

A variable argument list containing the arguments to the method.

Discussion

When it encounters a method call, the compiler generates a call to one of the functions `objc_msgSend`, `objc_msgSend_stret`, `objc_msgSendSuper`, or `objc_msgSendSuper_stret`. Messages sent to an object's superclass (using the `super` keyword) are sent using `objc_msgSendSuper`; other messages are sent using `objc_msgSend`. Methods that have data structures as return values are sent using `objc_msgSendSuper_stret` and `objc_msgSend_stret`.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/message.h`

objc_registerClassPair

Registers a class that was allocated using `objc_allocateClassPair`.

```
void objc_registerClassPair(Class cls)
```

Parameters

`cls`

The class you want to register.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

objc_registerProtocol

Registers a newly created protocol with the Objective-C runtime.

```
void objc_registerProtocol(Protocol *proto)
```

Parameters

`proto`

The protocol you want to register with the Objective-C runtime.

Discussion

When you create a new protocol using the `objc_allocateProtocol` (page 47), you then register it with the Objective-C runtime by calling this function. After a protocol is successfully registered, it is immutable and ready to use.

Availability

Available in OS X v10.7 and later.

See Also

`objc_allocateProtocol` (page 47)

Declared in

`objc/runtime.h`

objc_removeAssociatedObjects

Removes all associations for a given object.

```
void objc_removeAssociatedObjects(id object)
```

Parameters

`object`

An object that maintains associated objects.

Discussion

The main purpose of this function is to make it easy to return an object to a "pristine state". You should not use this function for general removal of associations from objects, since it also removes associations that other clients may have added to the object. Typically you should use `objc_setAssociatedObject` (page 64) with a `nil` value to clear an association.

Availability

Available in OS X v10.6 and later.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

See Also

objc_setAssociatedObject (page 64)
objc_getAssociatedObject (page 52)

Declared in

objc/runtime.h

objc_setAssociatedObject

Sets an associated value for a given object using a given key and association policy.

```
void objc_setAssociatedObject(id object, void *key, id value, objc_AssociationPolicy policy)
```

Parameters

`object`

The source object for the association.

`key`

The key for the association.

`value`

The value to associate with the key `key` for `object`. Pass `nil` to clear an existing association.

`policy`

The policy for the association. For possible values, see "Associative Object Behaviors" (page 95).

Availability

Available in OS X v10.6 and later.

See Also

objc_setAssociatedObject (page 64)
objc_removeAssociatedObjects (page 63)

Declared in

objc/runtime.h

objc_setEnumerationMutationHandler

Sets the current mutation handler.

```
void objc_setEnumerationMutationHandler(void (*handler)(id))
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Parameters

`handler`

A function pointer to the new mutation handler.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

objc_setFutureClass

Used by CoreFoundation's toll-free bridging.

```
void objc_setFutureClass(Class cls, const char *name)
```

Special Considerations

Do not call this function yourself.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

objc_storeWeak

Stores a new value in a `__weak` variable.

```
id objc_storeWeak(id *location, id obj)
```

Parameters

`location`

The address of the weak pointer.

`obj`

The new object you want the weak pointer to now point to.

Return Value

The value stored in `location` (that is, `obj`).

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Discussion
This function is typically used anywhere a `__weak` variable is the target of an assignment.

Availability
Available in OS X v10.7 and later.

Declared in
`objc/runtime.h`

object_copy

Returns a copy of a given object.

```
id object_copy(id obj, size_t size)
```

Parameters
`obj`
An Objective-C object.

`size`
The size of the object `obj`.

Return Value
A copy of `obj`.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

object_dispose

Frees the memory occupied by a given object.

```
id object_dispose(id obj)
```

Parameters
`obj`
An Objective-C object.

Return Value
`nil`.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

object_getClass

Returns the class of an object.

```
Class object_getClass(id object)
```

Parameters
`object`
The object you want to inspect.

Return Value
The class object of which `object` is an instance, or `Nil` if `object` is `nil`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

object_getClassName

Returns the class name of a given object.

```
const char *object_getClassName(id obj)
```

Parameters
`obj`
An Objective-C object.

Return Value
The name of the class of which `obj` is an instance.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

object_getIndexedIvars

Returns a pointer to any extra bytes allocated with a instance given object.

```
OBJC_EXPORT void *object_getIndexedIvars(id obj)
```

Parameters
`obj`
An Objective-C object.

Return Value
A pointer to any extra bytes allocated with `obj`. If `obj` was not allocated with any extra bytes, then dereferencing the returned pointer is undefined.

Discussion
This function returns a pointer to any extra bytes allocated with the instance (as specified by `class_createInstance` (page 24) with extraBytes>0). This memory follows the object's ordinary ivars, but may not be adjacent to the last ivar.

The returned pointer is guaranteed to be pointer-size aligned, even if the area following the object's last ivar is less aligned than that. Alignment greater than pointer-size is never guaranteed, even if the area following the object's last ivar is more aligned than that.

In a garbage-collected environment, the memory is scanned conservatively.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/objc.h`

object_getInstanceVariable

Obtains the value of an instance variable of a class instance.

```
Ivar object_getInstanceVariable(id obj, const char *name, void **outValue)
```

Parameters

`obj`

A pointer to an instance of a class. Pass the object containing the instance variable whose value you wish to obtain.

`name`

A C string. Pass the name of the instance variable whose value you wish to obtain.

`outValue`

On return, contains a pointer to the value of the instance variable.

Return Value

A pointer to the `Ivar` (page 84) data structure that defines the type and name of the instance variable specified by `name`.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/runtime.h`

object_getIvar

Reads the value of an instance variable in an object.

```
id object_getIvar(id object, Ivar ivar)
```

Parameters

`object`

The object containing the instance variable whose value you want to read.

`ivar`

The Ivar describing the instance variable whose value you want to read.

Return Value

The value of the instance variable specified by `ivar`, or `nil` if `object` is `nil`.

Discussion

`object_getIvar` is faster than `object_getInstanceVariable` (page 68) if the Ivar for the instance variable is already known.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

object_setClass

Sets the class of an object.

```
Class object_setClass(id object, Class cls)
```

Parameters
`object`
The object to modify.

`sel`
A class object.

Return Value
The previous value of `object`'s class, or `Nil` if `object` is `nil`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

object_setInstanceVariable

Changes the value of an instance variable of a class instance.

```
Ivar object_setInstanceVariable(id obj, const char *name, void *value)
```

Parameters
`obj`
A pointer to an instance of a class. Pass the object containing the instance variable whose value you wish to modify.

`name`
A C string. Pass the name of the instance variable whose value you wish to modify.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`value`
The new value for the instance variable.

Return Value
A pointer to the `Ivar` (page 84) data structure that defines the type and name of the instance variable specified by `name`.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

object_setIvar

Sets the value of an instance variable in an object.

```
void object_setIvar(id object, Ivar ivar, id value)
```

Parameters
`object`
The object containing the instance variable whose value you want to set.

`ivar`
The Ivar describing the instance variable whose value you want to set.

`value`
The new value for the instance variable.

Discussion
`object_setIvar` is faster than `object_setInstanceVariable` (page 70) if the Ivar for the instance variable is already known.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

property_copyAttributeList

Returns an array of property attributes for a given property.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

```
objc_property_attribute_t *property_copyAttributeList(objc_property_t property, unsigned int
 *outCount)
```

Parameters

`property`

The property whose attributes you want to copy.

`outCount`

The number of attributes returned in the array.

Return Value

An array of property attributes. You must free the array with `free()`.

Availability

Available in OS X v10.7 and later.

Declared in

`objc/runtime.h`

property_copyAttributeValue

Returns the value of a property attribute given the attribute name.

```
char *property_copyAttributeValue(objc_property_t property, const char *attributeName)
```

Parameters

`property`

The property whose value you are interested in.

`attributeName`

A C string representing the name of the attribute.

Return Value

The value string of the `attributeName` attribute, if one exists in `property`; otherwise, `nil`. You must free the returned value string with `free()`.

Availability

Available in OS X v10.7 and later.

Declared in

`objc/runtime.h`

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

property_getAttributes

Returns the attribute string of a property.

```
const char *property_getAttributes(objc_property_t property)
```

Return Value
A C string containing the property's attributes.

Discussion
The format of the attribute string is described in "Declared Properties" in *Objective-C Runtime Programming Guide*.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

property_getName

Returns the name of a property.

```
const char *property_getName(objc_property_t property)
```

Return Value
A C string containing the property's name.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

protocol_addMethodDescription

Adds a method to a protocol.

```
void protocol_addMethodDescription(Protocol *proto, SEL name, const char *types, BOOL
isRequiredMethod, BOOL isInstanceMethod)
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Parameters

`proto`

The protocol you want to add a method to.

`name`

The name of the method you want to add.

`types`

A C string representing the signature of the method you want to add.

`isRequiredMethod`

A Boolean indicating whether the method is a required method of the `proto` protocol. If `YES`, the method is a required method; if `NO`, the method is an optional method.

`isInstanceMethod`

A Boolean indicating whether the method is an instance method. If `YES`, the method is an instance method; if `NO`, the method is a class method.

Discussion

To add a method to a protocol using this function, the protocol must be under construction. That is, you must add any methods to `proto` before you register it with the Objective-C runtime (via the `objc_registerProtocol` (page 63) function).

Availability

Available in OS X v10.7 and later.

Declared in

`objc/runtime.h`

protocol_addProperty

Adds a property to a protocol that is under construction.

```
void protocol_addProperty(Protocol *proto, const char *name, const objc_property_attribute_t
 *attributes, unsigned int attributeCount, BOOL isRequiredProperty, BOOL isInstanceProperty)
```

Parameters

`proto`

The protocol you want to add a property to.

`name`

The name of the property you want to add.

`attributes`
: An array of property attributes.

`attributeCount`
: The number of properties in `attributes`.

`isRequiredProperty`
: A Boolean indicating whether the property's accessor methods are required methods of the `proto` protocol. If `YES`, the property's accessor methods are required methods; if `NO`, the property's accessor methods are optional methods.

`isInstanceProperty`
: A Boolean indicating whether the property's accessor methods are instance methods. If `YES`, the property's accessor methods are instance methods. `YES` is the only value allowed for a property. As a result, if you set this value to `NO`, the property will not be added to the protocol.

Discussion
The protocol you want to add the property to must be under construction—allocated but not yet registered with the Objective-C runtime (via the `objc_registerProtocol` (page 63) function).

Availability
Available in OS X v10.7 and later.

Declared in
`objc/runtime.h`

protocol_addProtocol

Adds a registered protocol to another protocol that is under construction.

```
void protocol_addProtocol(Protocol *proto, Protocol *addition)
```

Parameters
`proto`
: The protocol you want to add the registered protocol to.

`addition`
: The registered protocol you want to add to `proto`.

Discussion
The protocol you want to add to (`proto`) must be under construction—allocated but not yet registered with the Objective-C runtime. The protocol you want to add (`addition`) must be registered already.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Availability
Available in OS X v10.7 and later.

Declared in
`objc/runtime.h`

protocol_conformsToProtocol

Returns a Boolean value that indicates whether one protocol conforms to another protocol.

```
BOOL protocol_conformsToProtocol(Protocol *proto, Protocol *other)
```

Parameters
`proto`
A protocol.

`other`
A protocol.

Return Value
`YES` if `proto` conforms to `other`, otherwise `NO`.

Discussion
One protocol can incorporate other protocols using the same syntax that classes use to adopt a protocol:

```
@protocol ProtocolName < protocol list >
```

All the protocols listed between angle brackets are considered part of the ProtocolName protocol.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

protocol_copyMethodDescriptionList

Returns an array of method descriptions of methods meeting a given specification for a given protocol.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

```
struct objc_method_description *protocol_copyMethodDescriptionList(Protocol *p, BOOL
isRequiredMethod, BOOL isInstanceMethod, unsigned int *outCount)
```

Parameters

`p`

A protocol.

`isRequiredMethod`

A Boolean value that indicates whether returned methods should be required methods (pass `YES` to specify required methods).

`isInstanceMethod`

A Boolean value that indicates whether returned methods should be instance methods (pass `YES` to specify instance methods).

`outCount`

Upon return, contains the number of method description structures in the returned array.

Return Value

A C array of `objc_method_description` structures containing the names and types of `p`'s methods specified by `isRequiredMethod` and `isInstanceMethod`. The array contains `*outCount` pointers followed by a `NULL` terminator. You must free the list with `free()`.

If the protocol declares no methods that meet the specification, `NULL` is returned and `*outCount` is `0`.

Discussion

Methods in other protocols adopted by this protocol are not included.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

protocol_copyPropertyList

Returns an array of the properties declared by a protocol.

```
objc_property_t * protocol_copyPropertyList(Protocol *protocol, unsigned int *outCount)
```

Parameters

`proto`

A protocol.

`outCount`

Upon return, contains the number of elements in the returned array.

Return Value

A C array of pointers of type `objc_property_t` describing the properties declared by `proto`. Any properties declared by other protocols adopted by this protocol are not included. The array contains `*outCount` pointers followed by a `NULL` terminator. You must free the array with `free()`.

If the protocol declares no properties, `NULL` is returned and `*outCount` is `0`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

protocol_copyProtocolList

Returns an array of the protocols adopted by a protocol.

```
Protocol **protocol_copyProtocolList(Protocol *proto, unsigned int *outCount)
```

Parameters

`proto`

A protocol.

`outCount`

Upon return, contains the number of elements in the returned array.

Return Value

A C array of protocols adopted by `proto`. The array contains `*outCount` pointers followed by a `NULL` terminator. You must free the array with `free()`.

If the protocol declares no properties, `NULL` is returned and `*outCount` is `0`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

protocol_getMethodDescription

Returns a method description structure for a specified method of a given protocol.

```
struct objc_method_description protocol_getMethodDescription(Protocol *p, SEL aSel, BOOL
isRequiredMethod, BOOL isInstanceMethod)
```

Parameters

`p`

A protocol.

`aSel`

A selector

`isRequiredMethod`

A Boolean value that indicates whether `aSel` is a required method.

`isInstanceMethod`

A Boolean value that indicates whether `aSel` is an instance method.

Return Value

An `objc_method_description` structure that describes the method specified by `aSel`, `isRequiredMethod`, and `isInstanceMethod` for the protocol `p`.

If the protocol does not contain the specified method, returns an `objc_method_description` structure with the value `{NULL, NULL}`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

protocol_getName

Returns a the name of a protocol.

```
const char *protocol_getName(Protocol *p)
```

Parameters

`p`

A protocol.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Return Value
The name of the protocol `p` as a C string.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

protocol_getProperty

Returns the specified property of a given protocol.

```
objc_property_t protocol_getProperty(Protocol *proto, const char *name, BOOL
isRequiredProperty, BOOL isInstanceProperty)
```

Parameters

`proto`
A protocol.

`name`
The name of a property.

`isRequiredProperty`
A Boolean value that indicates whether `name` is a required property.

`isInstanceProperty`
A Boolean value that indicates whether `name` is an instance property.

Return Value
The property specified by `name`, `isRequiredProperty`, and `isInstanceProperty` for `proto`, or `NULL` if none of `proto`'s properties meets the specification.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

protocol_isEqual

Returns a Boolean value that indicates whether two protocols are equal.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

```
BOOL protocol_isEqual(Protocol *proto, Protocol *other)
```

Parameters

`proto`

A protocol.

`other`

A protocol.

Return Value

`YES` if `proto` is the same as `other`, otherwise `NO`.

Availability

Available in OS X v10.5 and later.

Declared in

`objc/runtime.h`

sel_getName

Returns the name of the method specified by a given selector.

```
const char* sel_getName(SEL aSelector)
```

Parameters

`aSelector`

A pointer of type `SEL` (page 85). Pass the selector whose name you wish to determine.

Return Value

A C string indicating the name of the selector.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/runtime.h`

sel_getUid

Registers a method name with the Objective-C runtime system.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

```
SEL sel_getUid(const char *str)
```

Parameters

`str`

A pointer to a C string. Pass the name of the method you wish to register.

Return Value

A pointer of type `SEL` (page 85) specifying the selector for the named method.

Discussion

The implementation of this method is identical to the implementation of `sel_registerName` (page 83).

Version Notes

Prior to OS X version 10.0, this method tried to find the selector mapped to the given name and returned `NULL` if the selector was not found. This was changed for safety, because it was observed that many of the callers of this function did not check the return value for `NULL`.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/objc.h`

sel_isEqual

Returns a Boolean value that indicates whether two selectors are equal.

```
BOOL sel_isEqual(SEL lhs, SEL rhs)
```

Parameters

`lhs`

The selector to compare with `rhs`.

`rhs`

The selector to compare with `lhs`.

Return Value

`YES` if `rhs` and `rhs` are equal, otherwise `NO`.

Discussion

`sel_isEqual` is equivalent to `==`.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/runtime.h`

sel_registerName

Registers a method with the Objective-C runtime system, maps the method name to a selector, and returns the selector value.

```
SEL sel_registerName(const char *str)
```

Parameters
`str`
A pointer to a C string. Pass the name of the method you wish to register.

Return Value
A pointer of type `SEL` (page 85) specifying the selector for the named method.

Discussion
You must register a method name with the Objective-C runtime system to obtain the method's selector before you can add the method to a class definition. If the method name has already been registered, this function simply returns the selector.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/runtime.h`

Data Types

Class-Definition Data Structures

Class

An opaque type that represents an Objective-C class.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

```
typedef struct objc_class *Class;
```

Availability
Available in OS X v10.6 and later.

Declared in
`objc/objc.h`

Method

An opaque type that represents a method in a class definition.

```
typedef struct objc_method *Method;
```

Availability
Available in OS X v10.6 and later.

Declared in
`objc/runtime.h`

Ivar

An opaque type that represents an instance variable.

```
typedef struct objc_ivar *Ivar;
```

Availability
Available in OS X v10.6 and later.

Declared in
`objc/runtime.h`

Category

An opaque type that represents a category.

```
typedef struct objc_category *Category;
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Availability
Available in OS X v10.6 and later.

Declared in
`objc/runtime.h`

objc_property_t

An opaque type that represents an Objective-C declared property.

```
typedef struct objc_property *objc_property_t;
```

Availability
Available in OS X v10.6 and later.

Declared in
`objc/runtime.h`

IMP

A pointer to the start of a method implementation.

```
id (*IMP)(id, SEL, ...)
```

Discussion
This data type is a pointer to the start of the function that implements the method. This function uses standard C calling conventions as implemented for the current CPU architecture. The first argument is a pointer to `self` (that is, the memory for the particular instance of this class, or, for a class method, a pointer to the metaclass). The second argument is the method selector. The method arguments follow.

SEL

Defines an opaque type that represents a method selector.

```
typedef struct objc_selector *SEL;
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Discussion

Method selectors are used to represent the name of a method at runtime. A method selector is a C string that has been registered (or "mapped") with the Objective-C runtime. Selectors generated by the compiler are automatically mapped by the runtime when the class is loaded.

You can add new selectors at runtime and retrieve existing selectors using the function `sel_registerName` (page 83).

When using selectors, you must use the value returned from `sel_registerName` (page 83) or the Objective-C compiler directive `@selector()`. You cannot simply cast a C string to `SEL`.

Availability

Available in OS X v10.6 and later.

Declared in

`objc/objc.h`

objc_method_description

Defines an Objective-C method.

```
struct objc_method_description {
   SEL name;
   char *types;
};
```

Fields

`name`

The name of the method at runtime.

`types`

The types of the method arguments.

Availability

Available in OS X v10.6 and later.

Declared in

`objc/runtime.h`

objc_method_list

Contains an array of method definitions.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

```
struct objc_method_list
{
struct objc_method_list *obsolete;
int method_count;
struct objc_method method_list[1];
}
```

Fields

`obsolete`

Reserved for future use.

`method_count`

An integer specifying the number of methods in the method list array.

`method_list`

An array of `Method` (page 84) data structures.

Availability

Available in OS X v10.5 and later.

Deprecated in OS X v10.5.

Not available to 64-bit applications.

Declared in

`objc/runtime.h`

objc_cache

Performance optimization for method calls. Contains pointers to recently used methods.

```
struct objc_cache
{
unsigned int mask;
unsigned int occupied;
Method buckets[1];
};
```

Fields

`mask`

An integer specifying the total number of allocated cache buckets (minus one). During method lookup, the Objective-C runtime uses this field to determine the index at which to begin a linear search of the `buckets` array. A pointer to a method's selector is masked against this field using a logical AND operation (`index = (mask & selector)`). This serves as a simple hashing algorithm.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`occupied`

An integer specifying the total number of occupied cache buckets.

`buckets`

An array of pointers to `Method` (page 84) data structures. This array may contain no more than `mask + 1` items. Note that pointers may be `NULL`, indicating that the cache bucket is unoccupied, and occupied buckets may not be contiguous. This array may grow over time.

Discussion

To limit the need to perform linear searches of method lists for the definitions of frequently accessed methods—an operation that can considerably slow down method lookup—the Objective-C runtime functions store pointers to the definitions of the most recently called method of the class in an `objc_cache` data structure.

objc_protocol_list

Represents a list of formal protocols.

```
struct objc_protocol_list
{
struct objc_protocol_list *next;
int count;
Protocol *list[1];
};
```

Fields

`next`

A pointer to another `objc_protocol_list` data structure.

`count`

The number of protocols in this list.

`list`

An array of pointers to `Class` (page 83) data structures that represent protocols.

Discussion

A formal protocol is a class definition that declares a set of methods, which a class must implement. Such a class definition contains no instance variables. A class definition may promise to implement any number of formal protocols.

Availability

Available in OS X v10.6 and later.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Not available to 64-bit applications.

Declared in
`objc/runtime.h`

objc_property_attribute_t

Defines a property attribute.

```
typedef struct {
   const char *name;
   const char *value;
} objc_property_attribute_t;
```

Fields

`name`
The name of the attribute.

`value`
The value of the attribute (usually empty).

Availability
Available in OS X v10.7 and later.

Declared in
`objc/runtime.h`

Instance Data Types

These are the data types that represent objects, classes, and superclasses.

- `id` (page 89) pointer to an instance of a class.
- `objc_object` (page 90) represents an instance of a class.
- `objc_super` (page 90) specifies the superclass of an instance.

id

A pointer to an instance of a class.

```
typedef struct objc_object *id;
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Availability
Available in OS X v10.6 and later.

Declared in
`objc/objc.h`

objc_object

Represents an instance of a class.

```
struct objc_object {
   Class isa;
};
```

Fields
`isa`
A pointer to the class definition of which this object is an instance.

Discussion
When you create an instance of a particular class, the allocated memory contains an `objc_object` data structure, which is directly followed by the data for the instance variables of the class.

The `alloc` and `allocWithZone:` methods of the Foundation framework class NSObject use the function `class_createInstance` (page 24) to create `objc_object` data structures.

objc_super

Specifies the superclass of an instance.

```
struct objc_super
{
id receiver;
Class class;
};
```

Fields
`receiver`
A pointer of type `id` (page 89). Specifies an instance of a class.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`class`

A pointer to an `Class` (page 83) data structure. Specifies the particular superclass of the instance to message.

Discussion
The compiler generates an `objc_super` data structure when it encounters the `super` keyword as the receiver of a message. It specifies the class definition of the particular superclass that should be messaged.

Availability
Available in OS X v10.6 and later.

Declared in
`objc/message.h`

Boolean Value

BOOL

Type to represent a Boolean value.

```
typedef signed char BOOL;
```

Discussion
`BOOL` is explicitly signed so `@encode(BOOL)` is `c` rather than `C` even if `-funsigned-char` is used.

For values, see "`Boolean Values`" (page 92).

Special Considerations
Since the type of `BOOL` is actually `char`, it does not behave in the same way as a C `_Bool` value or a C++ *bool* value. For example, the conditional in the following code will be false on i386 (and true on PPC):

```
- (BOOL)value {
    return 256;
}
// then
if ([self value]) doStuff();
```

By contrast, the conditional in the following code will be true on all platforms (even where `sizeof(bool) == 1`):

```
- (bool)value {
    return 256;
}
// then
if ([self value]) doStuff();
```

Availability
Available in OS X v10.1 and later.

Declared in
`objc/objc.h`

Associative References

objc_AssociationPolicy

Type to specify the behavior of an association.

```
typedef uintptr_t objc_AssociationPolicy;
```

Discussion
For values, see "Associative Object Behaviors" (page 95).

Availability
Available in OS X v10.6 and later.

Declared in
`objc/runtime.h`

Constants

Boolean Values

These macros define convenient constants to represent Boolean values.

```
#define YES (BOOL)1
#define NO  (BOOL)0
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Constants

`YES`

Defines `YES` as `1`.

Available in OS X v10.0 and later.

Declared in `objc/objc.h`.

`NO`

Defines `NO` as `0`.

Available in OS X v10.0 and later.

Declared in `objc/objc.h`.

Declared in

`objc.h`

Null Values

These macros define null values for classes and instances.

```
#define nil __DARWIN_NULL
#define Nil __DARWIN_NULL
```

Constants

`nil`

Defines the `id` of a null instance.

Available in OS X v10.0 and later.

Declared in `IONDRVLibraries.h`.

`Nil`

Defines the `id` of a null class.

Available in OS X v10.0 and later.

Declared in `objc/objc.h`.

Declared in

`objc.h`

Dispatch Function Prototypes

This macro indicates whether dispatch functions must be cast to an appropriate function pointer type.

```
#define OBJC_OLD_DISPATCH_PROTOTYPES 1
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Constants

`OBJC_OLD_DISPATCH_PROTOTYPES`

`OBJC_OLD_DISPATCH_PROTOTYPES == 0` enforces the rule that the dispatch functions must be cast to an appropriate function pointer type.

Available in OS X v10.8 and later.

Declared in `objc/objc-api.h`.

Declared in

`objc-api.h`

Objective-C Root Class

This macro annotates a class as being an Objective-C root class.

```
#define OBJC_ROOT_CLASS
```

Constants

`OBJC_ROOT_CLASS`

If you define an Objective-C root class, you receive a compiler error indicating that the class is defined without specifying a base class. You can avoid this compiler error by preceding the definition of the root class (that is, before the `@interface` directive) with `OBJC_ROOT_CLASS`.

Available in OS X v10.9 and later.

Declared in `objc/objc-api.h`.

Declared in

`objc-api.h`

Local Variable Storage Duration

This macro indicates that the values stored in certain local variables should not be aggressively released by the compiler during optimization.

```
#define NS_VALID_UNTIL_END_OF_SCOPE
```

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Constants

`NS_VALID_UNTIL_END_OF_SCOPE`

Marks local variables of type `id` or pointer-to-ObjC-object-type so that values stored into those local variable are not aggressively released by the compiler during optimization. Instead, the values are held until either the variable is assigned to again, or the end of the scope of the local variable (such as in a compound statement or a method definition).

Available in OS X v10.8 and later.

Declared in `NSObjCRuntime.h`.

Declared in

`NSObjCRuntime.h`

Associative Object Behaviors

Policies related to associative references.

```
enum {
   OBJC_ASSOCIATION_ASSIGN = 0,
   OBJC_ASSOCIATION_RETAIN_NONATOMIC = 1,
   OBJC_ASSOCIATION_COPY_NONATOMIC = 3,
   OBJC_ASSOCIATION_RETAIN = 01401,
   OBJC_ASSOCIATION_COPY = 01403
};
```

Constants

`OBJC_ASSOCIATION_ASSIGN`

Specifies a weak reference to the associated object.

Available in OS X v10.6 and later.

Declared in `objc/runtime.h`.

`OBJC_ASSOCIATION_RETAIN_NONATOMIC`

Specifies a strong reference to the associated object, and that the association is not made atomically.

Available in OS X v10.6 and later.

Declared in `objc/runtime.h`.

`OBJC_ASSOCIATION_COPY_NONATOMIC`

Specifies that the associated object is copied, and that the association is not made atomically.

Available in OS X v10.6 and later.

Declared in `objc/runtime.h`.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`OBJC_ASSOCIATION_RETAIN`

Specifies a strong reference to the associated object, and that the association is made atomically.

Available in OS X v10.6 and later.

Declared in `objc/runtime.h`.

`OBJC_ASSOCIATION_COPY`

Specifies that the associated object is copied, and that the association is made atomically.

Available in OS X v10.6 and later.

Declared in `objc/runtime.h`.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Deprecated Objective-C Runtime Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in OS X v10.5

class_setSuperclass

Sets the superclass of a given class. (Deprecated in OS X v10.5.)

```
Class class_setSuperclass(Class cls, Class newSuper)
```

Parameters

`cls`

The class whose superclass you want to set.

`newSuper`

The new superclass for `cls`.

Return Value

The old superclass for `cls`.

Special Considerations

You should not use this function.

Availability

Available in OS X v10.5 and later.

Deprecated in OS X v10.5.

Declared in

`objc/runtime.h`

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

OS X Version 10.5 Delta

The low-level Objective-C runtime API is significantly updated in OS X version 10.5. Many functions and all existing data structures are replaced with new functions. This document describes the differences between the 10.5 version and previous versions.

Runtime Functions

Basic types

`arith_t`: Changed from `int` to `intptr_t`.

`uarith_t`: Changed from `unsigned` to `uintptr_t`.

Instances

The following functions are unchanged:

- `object_dispose` (page 66)
- `object_getClassName` (page 67)
- `object_getIndexedIvars` (page 68)
- `object_setInstanceVariable` (page 70)
- `object_getInstanceVariable` (page 68)

The following function is modified:

- `object_copy` (page 66) (The `nBytes` parameter is changed from `unsigned` to `size_t`.)

The following functions are added:

- `object_getClass` (page 67)
- `object_setClass` (page 70)
- `object_getIvar` (page 69)

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`object_setIvar` (page 71)

The following functions are deprecated:

object_copyFromZone: deprecated in favor of `object_copy` (page 66)

object_realloc

object_reallocFromZone: no substitute

_alloc: no substitute

_copy: no substitute

_realloc: no substitute

_dealloc: no substitute

_zoneAlloc: no substitute

_zoneRealloc: no substitute

_zoneCopy: no substitute

_error: no substitute

Class Inspection

The following functions are unchanged:

`objc_getClassList` (page 54)

`objc_lookUpClass` (page 58)

`objc_getClass` (page 53)

`objc_getMetaClass` (page 55)

`class_getVersion` (page 31)

`class_getInstanceVariable` (page 27)

`class_getInstanceMethod` (page 26)

`class_getClassMethod` (page 25)

The following function is modified:

class_createInstance: `idxIvars` parameter Changed from `unsigned` to `size_t`

The following functions are added:

`class_getName` (page 30)

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`class_getSuperclass` (page 31)

`class_isMetaClass` (page 32)

`class_copyMethodList` (page 22)

`class_getMethodImplementation` (page 28)

`class_getMethodImplementation_stret` (page 29)

`class_respondsToSelector` (page 34)

`class_conformsToProtocol` (page 21)

`class_copyProtocolList` (page 23)

`class_copyIvarList` (page 21)

The following functions are deprecated:

`objc_getClasses`: deprecated in favor of `objc_getClassList` (page 54)

`class_createInstanceFromZone`: deprecated in favor of `class_createInstance` (page 24)

`class_nextMethodList`: deprecated in favor of new `class_copyMethodList` (page 22)

`class_lookupMethod`: deprecated in favor of `class_getMethodImplementation` (page 28)

`class_respondsToMethod`: deprecated in favor of `class_respondsToSelector` (page 34)

The following function is used only by ZeroLink:

objc_getRequiredClass

Class Manipulation

The following function is unchanged:

`class_setVersion` (page 35)

The following functions are added:

`objc_allocateClassPair` (page 46)

`objc_registerClassPair` (page 62)

`objc_duplicateClass` (page 51)

`class_addMethod` (page 18)

`class_addIvar` (page 17)

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

`class_addProtocol` (page 20)

The following functions are deprecated:

`objc_addClass`: deprecated in favor of `objc_allocateClassPair` (page 46) and `objc_registerClassPair` (page 62)

`class_addMethods`: deprecated in favor of new `class_addMethod` (page 18)

`class_removeMethods`: deprecated with no substitute

`class_poseAs`: deprecated in favor of categories and `method_setImplementation` (page 45)

Methods

The following function is unchanged:

`method_getNumberOfArguments` (page 43)

The following functions are added:

`method_getName` (page 42)

`method_getImplementation` (page 42)

`method_getTypeEncoding` (page 44)

`method_copyReturnType` (page 40)

`method_copyArgumentType` (page 39)

`method_setImplementation` (page 45)

The following functions are deprecated:

`method_getArgumentInfo`

`method_getSizeOfArguments`

Instance Variables

The following functions are added:

`ivar_getName` (page 38)

`ivar_getTypeEncoding` (page 39)

`ivar_getOffset` (page 39)

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Selectors

The following functions are unchanged:

- `sel_getName` (page 81)
- `sel_registerName` (page 83)
- `sel_getUid` (page 81)

The following function is added:

- `sel_isEqual` (page 82)

The following function is deprecated:

- `sel_isMapped`: deprecated with no substitute

Runtime

The following functions are deprecated favor of dyld:

- `objc_loadModules`
- `objc_loadModule`
- `objc_unloadModules`

The following functions are deprecated:

- `objc_setClassHandler`: deprecated with no substitute
- `objc_setMultithreaded`: deprecated with no substitute

The following previously undocumented functions are deprecated with no substitute:

- objc_getOrigClass
- _objc_create_zone
- _objc_error
- _objc_flush_caches
- _objc_resolve_categories_for_class
- _objc_setClassLoader
- _objc_setNilReceiver
- _objc_getNilReceiver

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

_objcInit

The following undocumented functions are unchanged:

_objc_getFreedObjectClass

instrumentObjcMessageSends

_objc_debug_class_hash

_class_printDuplicateCacheEntries

_class_printMethodCaches

_class_printMethodCacheStatistics

Messaging

The following functions are unchanged:

`objc_msgSend` (page 58)

`objc_msgSend_stret` (page 61)

`objc_msgSendSuper` (page 59)

`objc_msgSendSuper_stret` (page 60)

`objc_msgSendSuper_stret` (page 60)

The following functions are removed:

`objc_msgSendv`
`objc_msgSendv_stret`
`objc_msgSendv_fpret`

Protocols

The following functions are added:

`objc_getProtocol` (page 56)

`objc_copyProtocolList` (page 50)

Exceptions

The following functions are unchanged:

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

objc_exception_throw

objc_exception_try_enter

objc_exception_try_exit

objc_exception_extract

objc_exception_match

objc_exception_get_functions

objc_exception_set_functions

Synchronization

The following functions are unchanged:

objc_sync_enter

objc_sync_exit

objc_sync_wait

objc_sync_notify

objc_sync_notifyAll

These functions are only used by the compiler.

NXHashTable and NXMapTable

`NXHashTable` and `NXMapTable` are unchanged. They are limited to 4 billion entries.

Structures

The `objc_super` struct is unchanged:

```
struct objc_super {
    id receiver;
    Class super_class;
};
```

All other structures deprecated in favor of opaque types and functional API. Substitutes are shown in the following tables.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Table B-1 Substitutions for `objc_class`

Variable	Substitution
`struct objc_class *isa;`	object_getClass(), object_setClass()
struct objc_class *super_class;	class_getSuperclass()
const char *name;	class_getName()
long version;	class_getVersion(), class_setVersion()
long info;	class_isMetaClass()
long instance_size;	no substitute
struct objc_ivar_list *ivars;	class_copyIvarList(), class_addIvar()
struct objc_method_list **methodLists;	class_copyMethodList(), class_addMethod()
struct objc_cache *cache;	no substitute
struct objc_protocol_list *protocols;	class_copyProtocolList(), class_addProtocol()

Table B-2 Substitutions for `objc_method`

Variable	Substitution
SEL method_name;	method_getName()
char *method_types;	method_getTypeEncoding()
IMP method_imp;	method_getImplementation(), method_setImplementation()

Table B-3 Substitutions for `objc_ivar`

Variable	Substitution
char *ivar_name;	ivar_getName()
char *ivar_type;	ivar_getTypeEncoding()
int ivar_offset;	ivar_getOffset()

There are no substitutes for the following structs:

objc_object {...};

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

objc_category {...};

objc_method_list {...};

objc_ivar_list {...};

objc_protocol_list {...};

objc_cache {...};

objc_module {...};

objc_symtab {...};

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Objective-C Runtime Reference*.

Date	Notes
2013-10-22	Added missing API.
2010-06-17	Removed obsolete functions marg-related API (such as marg_setValue).
2009-10-19	Added functions related to associative references.
2009-06-02	Updated for OS X v10.6.
2008-11-19	Added links to the new Objective-C 2.0 Runtime Programming Guide.
2008-10-15	TBD
2007-12-11	Enhanced description of object_getIndexedIvars.
2007-10-31	Updated for OS X v10.5. Corrected the code example for the objc_getClassList function.
2007-05-25	Included new features in Objective-C 2.0.
2005-10-04	Minor correction to CreateClassDefinition function and definitions of marg_ macros.
2005-08-11	Corrected errors and documented macros. Corrected declaration of `class_getClassMethod` (page 25). Renamed the "Class Handler Callback" section to `ClassHandlerCallback` and added example function declaration to the description. Corrected result description of `method_getArgumentInfo`. Documented `YES` and `NO` macros in "Macros".

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Date	Notes
2004-08-31	New document that describes the data structures and programming interface used in the Objective-C runtime system. This document replaces information about the printing system that was published previously in *The Objective-C Programming Language*.

2013-10-22 | Copyright © 2002, 2013 Apple Inc. All Rights Reserved.

Apple Inc.
Copyright © 2002, 2013 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

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

Apple, the Apple logo, Mac, Objective-C, and OS X are trademarks of Apple Inc., registered in the U.S. and other countries.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.