

---

# Objective-C Runtime Reference

Languages



2009-10-19



Apple Inc.  
© 2002, 2009 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.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

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.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

.Mac is a registered service mark of Apple Inc.

Apple, the Apple logo, eMac, Mac, Mac OS, and Objective-C are trademarks of Apple Inc., registered in the United States and other countries.

iPhone is a trademark of Apple Inc.

Simultaneously published in the United States and Canada.

**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.**

# Contents

## Objective-C Runtime Reference 9

---

Overview	9
Who Should Read This Document	9
Functions by Task	10
Working with Classes	10
Adding Classes	11
Instantiating Classes	11
Working with Instances	11
Obtaining Class Definitions	12
Working with Instance Variables	12
Associative References	13
Sending Messages	13
Forwarding Messages	13
Working with Methods	14
Working with Selectors	14
Working with Protocols	15
Working with Properties	15
Functions	15
class_addIvar	15
class_addMethod	16
class_addProtocol	17
class_conformsToProtocol	17
class_copyIvarList	17
class_copyMethodList	18
class_copyPropertyList	18
class_copyProtocolList	19
class_createInstance	19
class_getClassMethod	20
class_getClassVariable	20
class_getInstanceMethod	20
class_getInstanceSize	21
class_getInstanceVariable	21
class_getIvarLayout	21
class_getMethodImplementation	22
class_getMethodImplementation_stret	22
class_getName	23
class_getProperty	23
class_getSuperclass	23
class_getVersion	24
class_getWeakIvarLayout	24
class_isMetaClass	24

class_replaceMethod	25
class_respondToSelector	25
class_setIvarLayout	26
class_setSuperclass	26
class_setVersion	26
class_setWeakIvarLayout	27
ivar_getName	27
ivar_getOffset	27
ivar_getTypeEncoding	28
marg_free	28
marg_getRef	28
marg_getValue	29
marg_malloc	29
marg_setValue	29
method_copyArgumentType	30
method_copyReturnType	30
method_exchangeImplementations	30
method_getArgumentType	31
method_getImplementation	31
method_getName	31
method_getNumberOfArguments	32
method_getReturnType	32
method_getTypeEncoding	32
method_setImplementation	33
objc_allocateClassPair	33
objc_copyProtocolList	33
objc_duplicateClass	34
objc_getAssociatedObject	34
objc_getClass	34
objc_getClassList	35
objc_getFutureClass	36
objc_getMetaClass	36
objc_getProtocol	36
objc_getRequiredClass	36
objc_lookUpClass	37
objc_msgSend	37
objc_msgSendSuper	38
objc_msgSendSuper_stret	38
objc_msgSend_fpret	39
objc_msgSend_stret	39
objc_registerClassPair	40
objc_removeAssociatedObjects	40
objc_setAssociatedObject	40
objc_setFutureClass	41
object_copy	41
object_dispose	41

object_getClass	42
object_getClassName	42
object_getIndexedIvars	42
object_getInstanceVariable	43
object_getIvar	43
object_setClass	44
object_setInstanceVariable	44
object_setIvar	44
property_getAttributes	45
property_getName	45
protocol_conformsToProtocol	45
protocol_copyMethodDescriptionList	46
protocol_copyPropertyList	46
protocol_copyProtocolList	47
protocol_getMethodDescription	47
protocol_getName	48
protocol_getProperty	48
protocol_isEqual	49
sel_getName	49
sel_getUid	49
sel_isEqual	50
sel_registerName	50
Data Types	51
Class-Definition Data Structures	51
Instance Data Types	54
Associative References	55
Constants	56
Associative Object Behaviors	56

## Appendix A      **Mac OS X Version 10.5 Delta 57**

---

Runtime Functions	57
Basic types	57
Instances	57
Class Inspection	58
Class Manipulation	59
Methods	59
Instance Variables	60
Selectors	60
Runtime	60
Messaging	61
Protocols	61
Exceptions	61
Synchronization	62
NXHashTable and NXMapTable	62
Structures	62

**Document Revision History 65**

---

**Index 67**

---

# Tables and Listings

## Objective-C Runtime Reference 9

---

Listing 1 Using `objc_getClassList` 35

## Appendix A Mac OS X Version 10.5 Delta 57

---

Table A-1 Substitutions for `objc_class` 62

Table A-2 Substitutions for `objc_method` 63

Table A-3 Substitutions for `objc_ivar` 63



# Objective-C Runtime Reference

---

<b>Declared in</b>	<code>/usr/include/objc/runtime.h</code>
<b>Companion guides</b>	Objective-C Runtime Programming Guide The Objective-C Programming Language

## Overview

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

The low-level Objective-C runtime API is significantly updated in Mac 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.

“Deprecated” below means “deprecated in Mac 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 23)  
Returns the name of a class.
- [class\\_getSuperclass](#) (page 23)  
Returns the superclass of a class.
- [class\\_setSuperclass](#) (page 26)  
Sets the superclass of a given class.
- [class\\_isMetaClass](#) (page 24)  
Returns a Boolean value that indicates whether a class object is a metaclass.
- [class\\_getInstanceSize](#) (page 21)  
Returns the size of instances of a class.
- [class\\_getInstanceVariable](#) (page 21)  
Returns the `Ivar` for a specified instance variable of a given class.
- [class\\_getClassVariable](#) (page 20)  
Returns the `Ivar` for a specified class variable of a given class.
- [class\\_addIvar](#) (page 15)  
Adds a new instance variable to a class.
- [class\\_copyIvarList](#) (page 17)  
Describes the instance variables declared by a class.
- [class\\_getIvarLayout](#) (page 21)  
Returns a description of the `Ivar` layout for a given class.
- [class\\_setIvarLayout](#) (page 26)  
Sets the `Ivar` layout for a given class.
- [class\\_getWeakIvarLayout](#) (page 24)  
Returns a description of the layout of weak `Ivars` for a given class.
- [class\\_setWeakIvarLayout](#) (page 27)  
Sets the layout for weak `Ivars` for a given class.
- [class\\_getProperty](#) (page 23)  
Returns a property with a given name of a given class.
- [class\\_copyPropertyList](#) (page 18)  
Describes the properties declared by a class.
- [class\\_addMethod](#) (page 16)  
Adds a new method to a class with a given name and implementation.
- [class\\_getInstanceMethod](#) (page 20)  
Returns a specified instance method for a given class.
- [class\\_getClassMethod](#) (page 20)  
Returns a pointer to the data structure describing a given class method for a given class.
- [class\\_copyMethodList](#) (page 18)  
Describes the instance methods implemented by a class.

[class\\_replaceMethod](#) (page 25)

Replaces the implementation of a method for a given class.

[class\\_getMethodImplementation](#) (page 22)

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

[class\\_getMethodImplementation\\_stret](#) (page 22)

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

[class\\_respondsToSelector](#) (page 25)

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

[class\\_addProtocol](#) (page 17)

Adds a protocol to a class.

[class\\_conformsToProtocol](#) (page 17)

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

[class\\_copyProtocolList](#) (page 19)

Describes the protocols adopted by a class.

[class\\_getVersion](#) (page 24)

Returns the version number of a class definition.

[class\\_setVersion](#) (page 26)

Sets the version number of a class definition.

[objc\\_getFutureClass](#) (page 36)

Used by CoreFoundation's toll-free bridging.

[objc\\_setFutureClass](#) (page 41)

Used by CoreFoundation's toll-free bridging.

## Adding Classes

[objc\\_allocateClassPair](#) (page 33)

Creates a new class and metaclass.

[objc\\_registerClassPair](#) (page 40)

Registers a class that was allocated using `objc_allocateClassPair`.

[objc\\_duplicateClass](#) (page 34)

Used by Foundation's Key-Value Observing.

## Instantiating Classes

[class\\_createInstance](#) (page 19)

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

## Working with Instances

[object\\_copy](#) (page 41)

Returns a copy of a given object.

[object\\_dispose](#) (page 41)

Frees the memory occupied by a given object.

[object\\_setInstanceVariable](#) (page 44)

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

[object\\_getInstanceVariable](#) (page 43)

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

[object\\_getIndexedIvars](#) (page 42)

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

[object\\_getIvar](#) (page 43)

Reads the value of an instance variable in an object.

[object\\_setIvar](#) (page 44)

Sets the value of an instance variable in an object.

[object\\_getClassName](#) (page 42)

Returns the class name of a given object.

[object\\_getClass](#) (page 42)

Returns the class of an object.

[object\\_setClass](#) (page 44)

Sets the class of an object.

## Obtaining Class Definitions

[objc\\_getClassList](#) (page 35)

Obtains the list of registered class definitions.

[objc\\_lookUpClass](#) (page 37)

Returns the class definition of a specified class.

[objc\\_getClass](#) (page 34)

Returns the class definition of a specified class.

[objc\\_getRequiredClass](#) (page 36)

Returns the class definition of a specified class.

[objc\\_getMetaClass](#) (page 36)

Returns the metaclass definition of a specified class.

## Working with Instance Variables

[ivar\\_getName](#) (page 27)

Returns the name of an instance variable.

[ivar\\_getTypeEncoding](#) (page 28)

Returns the type string of an instance variable.

[ivar\\_getOffset](#) (page 27)

Returns the offset of an instance variable.

## Associative References

[objc\\_setAssociatedObject](#) (page 40)

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

[objc\\_getAssociatedObject](#) (page 34)

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

[objc\\_removeAssociatedObjects](#) (page 40)

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 37) sends a message with a simple return value to an instance of a class.
- [objc\\_msgSend\\_stret](#) (page 39) sends a message with a data-structure return value to an instance of a class.
- [objc\\_msgSendSuper](#) (page 38) sends a message with a simple return value to the superclass of an instance of a class.
- [objc\\_msgSendSuper\\_stret](#) (page 38) sends a message with a data-structure return value to the superclass of an instance of a class.

[objc\\_msgSend](#) (page 37)

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

[objc\\_msgSend\\_fpret](#) (page 39)

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

[objc\\_msgSend\\_stret](#) (page 39)

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

[objc\\_msgSendSuper](#) (page 38)

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

[objc\\_msgSendSuper\\_stret](#) (page 38)

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

## Forwarding Messages

This section describes the functions used by `NSObject` and `NSInvocation` to forward method invocations. The arguments to the method are given as a list of arguments, and as such the nature of the calling convention varies for each CPU architecture.

[marg\\_malloc](#) (page 29)

Macro that allocates an argument list.

[marg\\_free](#) (page 28)

Macro that releases an argument list.

[marg\\_getRef](#) (page 28)

Macro that returns a pointer to an argument in an argument list.

[marg\\_getValue](#) (page 29)

Macro that returns the value of an argument in an argument list.

[marg\\_setValue](#) (page 29)

Macro that sets the value of an argument in an argument list.

## Working with Methods

[method\\_getName](#) (page 31)

Returns the name of a method.

[method\\_getImplementation](#) (page 31)

Returns the implementation of a method.

[method\\_getTypeEncoding](#) (page 32)

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

[method\\_copyReturnType](#) (page 30)

Returns a string describing a method's return type.

[method\\_copyArgumentType](#) (page 30)

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

[method\\_getReturnType](#) (page 32)

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

[method\\_getNumberOfArguments](#) (page 32)

Returns the number of arguments accepted by a method.

[method\\_getArgumentType](#) (page 31)

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

[method\\_setImplementation](#) (page 33)

Sets the implementation of a method.

[method\\_exchangeImplementations](#) (page 30)

Exchanges the implementations of two methods.

## Working with Selectors

[sel\\_getName](#) (page 49)

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

[sel\\_registerName](#) (page 50)

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

[sel\\_getUid](#) (page 49)

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

[sel\\_isEqual](#) (page 50)

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

## Working with Protocols

[objc\\_getProtocol](#) (page 36)

Returns a specified protocol.

[objc\\_copyProtocolList](#) (page 33)

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

[protocol\\_getName](#) (page 48)

Returns a the name of a protocol.

[protocol\\_isEqual](#) (page 49)

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

[protocol\\_copyMethodDescriptionList](#) (page 46)

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

[protocol\\_getMethodDescription](#) (page 47)

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

[protocol\\_copyPropertyList](#) (page 46)

Returns an array of the properties declared by a protocol.

[protocol\\_getProperty](#) (page 48)

Returns the specified property of a given protocol.

[protocol\\_copyProtocolList](#) (page 47)

Returns an array of the protocols adopted by a protocol.

[protocol\\_conformsToProtocol](#) (page 45)

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

## Working with Properties

[property\\_getName](#) (page 45)

Returns the name of a property.

[property\\_getAttributes](#) (page 45)

Returns the attribute string of an property.

## Functions

### **class\_addIvar**

Adds a new instance variable to a class.

```
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 33) and before `objc_registerClassPair` (page 40). 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))`.

**Declared In**

`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`.

*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 33).

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@:");
```

**Declared In**

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

**Declared In**

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.**Declared In**

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.

*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.

#### Declared In

`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 20) or [class\\_getClassMethod](#) (page 20).

#### Declared In

`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.

**Declared In**

`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.

*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.

**Declared In**

`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 wish 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*.

**Declared In**

`runtime.h`

**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 52). Pass the selector of the method you want to retrieve.

**Return Value**

A pointer to the [Method](#) (page 51) 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 an instance method with the specified selector.

**Discussion**

Note that this function searches superclasses for implementations, whereas [class\\_copyMethodList](#) (page 18) does not.

**Declared In**

`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.

**Return Value**

A pointer to an [Ivar](#) (page 51) data structure containing information about the class variable specified by *name*.

**Declared In**

`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.

#### 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 18) does not.

#### Declared In

`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.

#### Declared In

`runtime.h`

### **class\_getInstanceVariable**

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

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

#### 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 51) data structure containing information about the instance variable specified by *name*.

#### Declared In

`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*.

**Declared In**

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)
```

**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.

**Declared In**

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.

**Declared In**

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.**Declared In**

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.**Declared In**

runtime.h

**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.**Declared In**

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 51) 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.

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.

**Declared In**

`runtime.h`

**class\_getWeakIvarLayout**

Returns a description of the layout of weak Ivars 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 Ivars for *cls*.

**Declared In**

`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.

**Declared In**

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 16) 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 33) were called. The type encoding specified by *types* is ignored.

**Declared In**

runtime.h

**class\_respondToSelector**

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

```
BOOL class_respondToSelector(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 `respondToSelector:` or `instancesRespondToSelector:` methods instead of this function.

**Declared In**

`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 `Ivars` for *cls*.

**Declared In**

`runtime.h`

**class\_setSuperclass**

Sets the superclass of a given class.

```
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.

**Declared In**

`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 51) 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.

**Declared In**

`runtime.h`

**class\_setWeakIvarLayout**

Sets the layout for weak Ivars for a given class.

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

**Parameters**

*cls*

The class to modify.

*layout*

The layout of the weak Ivars for *cls*.

**Declared In**

`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.

**Declared In**

`runtime.h`

**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 43) and [object\\_setIvar](#) (page 44) instead of using this offset to access the instance variable data directly.

**Declared In**

`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.

**Declared In**

`runtime.h`

**marg\_free**

Macro that releases an argument list.

```
marg_free(margs)
```

**Parameters**

*margs*

A pointer of type [marg\\_list](#) (page 52). Pass the argument list to release.

**marg\_getRef**

Macro that returns a pointer to an argument in an argument list.

```
marg_getRef(margs, offset, type)
```

**Parameters**

*margs*

A pointer of type [marg\\_list](#) (page 52).

*offset*

A long integer value. Pass the byte offset to the argument in the list whose pointer you wish to obtain.

*type*

A type name. Pass the type of the argument located at *offset*.

**Return Value**

A pointer to the argument specified by the *offset* parameter.

**Discussion**

You can use this macro to manipulate any sort of `int` or pointer parameter. If you want to handle floats and structs, you should use `NSInvocation` instead.

**marg\_getValue**

Macro that returns the value of an argument in an argument list.

```
marg_getValue(margs, offset, type)
```

**Parameters**

*margs*

A pointer of type `marg_list` (page 52).

*offset*

A long integer value. Pass the byte offset to the argument in the list whose value you wish to obtain.

*type*

A type name. Pass the type of the argument located at `offset`.

**Return Value**

The value of the argument specified by the `offset` parameter.

**Discussion**

You can use this macro to manipulate any sort of `int` or pointer parameter. If you want to handle floats and structs, you should use `NSInvocation` instead.

**marg\_malloc**

Macro that allocates an argument list.

```
marg_malloc(margs, method)
```

**Parameters**

*margs*

A pointer of type `marg_list` (page 52). Pass the variable that contains the argument list pointer.

*method*

A pointer to an `Method` (page 51) data structure. Pass the method for which the argument list is allocated.

**Discussion**

You can use this macro to manipulate any sort of `int` or pointer parameter. If you want to handle floats and structs, you should use `NSInvocation` instead.

**marg\_setValue**

Macro that sets the value of an argument in an argument list.

```
marg_setValue(margs, offset, type, value)
```

**Parameters**

*margs*

A pointer of type `marg_list` (page 52).

*offset*

A long integer value. Pass the byte offset to the argument in the list whose pointer you wish to obtain.

*type*

A type name. Pass the type of the argument located at *offset*.

*value*

A value. Pass the new value for the argument.

#### **Discussion**

You can use this macro to manipulate any sort of `int` or pointer parameter. If you want to handle floats and structs, you should use `NSInvocation` instead.

### **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()`.

#### **Declared In**

`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()`.

#### **Declared In**

`runtime.h`

### **method\_exchangeImplementations**

Exchanges the implementations of two methods.

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

**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);
```

**Declared In**

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.

**Declared In**

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.

**Declared In**

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.

**Discussion**

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

**Declared In**

`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 51) data structure. Pass the method in question.

**Return Value**

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

**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.

**Declared In**

`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.

**Declared In**

`runtime.h`

**method\_setImplementation**

Sets the implementation of a method.

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

**Return Value**

The previous implementation of the method.

**Declared In**

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 16) and `class_addIvar` (page 15). When you are done building the class, call `objc_registerClassPair` (page 40). 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.

**Declared In**

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.

**Declared In**

`runtime.h`

**objc\_duplicateClass**

Used by Foundation's Key-Value Observing.

`objc_duplicateClass`

**Special Considerations**

Do not call this function yourself.

**Declared In**

`runtime.h`

**objc\_getAssociatedObject**

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

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

**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*.

**See Also**

[objc\\_setAssociatedObject](#) (page 40)

**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 37) 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 37) does not call the class handler callback.

**Special Considerations**

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

**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.

**objc\_getFutureClass**

Used by CoreFoundation's toll-free bridging.

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

**Special Considerations**

Do not call this function yourself.

**Declared In**

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.

**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.

**Declared In**

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 34), 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.

**Declared In**

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 34) is different from this function in that if the class is not registered, [objc\\_getClass](#) (page 34) 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.

**objc\_msgSend**

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

```
id objc_msgSend(id theReceiver, SEL theSelector, ...)
```

**Parameters**

*theReceiver*

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

*theSelector*

The selector of the method that handles the message.

...

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`.

**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 55) 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 52). 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`.

**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 55) 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 52). 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`.

**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.

**Declared In**

`objc-runtime.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, ...)
```

**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 52). 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`.

**objc\_registerClassPair**

Registers a class that was allocated using `objc_allocateClassPair`.

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

**Parameters**

*cls*

The class you want to register.

**Declared In**

`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 40) with a `nil` value to clear an association.

**See Also**

[objc\\_setAssociatedObject](#) (page 40)

[objc\\_getAssociatedObject](#) (page 34)

**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 56).

**See Also**

[objc\\_setAssociatedObject](#) (page 40)

[objc\\_removeAssociatedObjects](#) (page 40)

## **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.

**Declared In**

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*.

**Declared In**

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*.

**Declared In**

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.**Declared In**

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.**Declared In**

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 19) 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.

**Declared In**

runtime.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 51) data structure that defines the type and name of the instance variable specified by *name*.

**Declared In**

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 43) if the Ivar for the instance variable is already known.

**Declared In**

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.

**Declared In**

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.

*value*

The new value for the instance variable.

**Return Value**

A pointer to the [Ivar](#) (page 51) data structure that defines the type and name of the instance variable specified by *name*.

**Declared In**

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 44) if the Ivar for the instance variable is already known.

**Declared In**

`runtime.h`

**property\_getAttributes**

Returns the attribute string of an 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*.

**Declared In**

`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.

**Declared In**

`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.

**Declared In**

runtime.h

**protocol\_copyMethodDescriptionList**

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

```
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.

**Declared In**

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.

#### Declared In

`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.

#### Declared In

`runtime.h`

### 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}`.

**Discussion**

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

**Declared In**

`runtime.h`

**protocol\_getName**

Returns a the name of a protocol.

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

**Parameters**

*p*

A protocol.

**Return Value**

The name of the protocol *p* as a C string.

**Declared In**

`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 a required property.

**Return Value**

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

**Declared In**

`runtime.h`

**protocol\_isEqual**

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

```
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.

**Declared In**

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 52). Pass the selector whose name you wish to determine.

**Return Value**

A C string indicating the name of the selector.

**Declared In**

runtime.h

**sel\_getUid**

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

```
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 52) specifying the selector for the named method.

**Discussion**

The implementation of this method is identical to the implementation of [sel\\_registerName](#) (page 50).

**Version Notes**

Prior to Mac 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.

**Declared In**

runtime.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 *lhs* and *rhs* are equal, otherwise NO.

**Discussion**

`sel_isEqual` is equivalent to `==`.

**Declared In**

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 52) 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.

**Declared In**

runtime.h

## Data Types

### Class-Definition Data Structures

---

#### Class

An opaque type that represents an Objective-C class.

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

#### Declared In

objc.h

#### Method

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

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

#### Declared In

runtime.h

#### Ivar

An opaque type that represents an instance variable.

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

#### Declared In

runtime.h

#### Category

An opaque type that represents a category.

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

#### Declared In

runtime.h

#### objc\_property\_t

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

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

#### Declared In

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.

**marg\_list**

A reference to an argument list.

```
typedef void* marg_list;
```

**Discussion**

This data type is a reference to a list of method arguments. Use it with the functions described in [“Working with Instances”](#) (page 11).

**SEL**

Defines an opaque type that represents a method selector.

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

**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 50).

When using selectors, you must use the value returned from [`sel\_registerName`](#) (page 50) or the Objective-C compiler directive `@selector()`. You cannot simply cast a C string to `SEL`.

**Declared In**

`objc.h`

**objc\_method\_list**

Contains an array of method definitions.

```

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 51) data structures.**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.

occupied

An integer specifying the total number of occupied cache buckets.

buckets

An array of pointers to [Method](#) (page 51) 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 51) 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.

## Instance Data Types

---

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

- `id` (page 54) pointer to an instance of a class.
- `objc_object` (page 54) represents an instance of a class.
- `objc_super` (page 55) specifies the superclass of an instance.

**id**

A pointer to an instance of a class.

```
typedef struct objc_object {
    Class isa;
} *id;
```

**Declared In**

`objc.h`

**objc\_object**

Represents an instance of a class.

```
struct objc_object
{
    struct objc_class *isa;
    /* ...variable length data containing instance variable values... */
};
```

**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 19) 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 54). Specifies an instance of a class.

class

A pointer to an `Class` (page 51) 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.

## Associative References

---

**objc\_AssociationPolicy**

Type to specify behavior the behavior of an association.

```
typedef uintptr_t objc_AssociationPolicy;
```

**Discussion**

For values, see “[Associative Object Behaviors](#)” (page 56).

## Constants

### 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.

`OBJC_ASSOCIATION_RETAIN_NONATOMIC`

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

`OBJC_ASSOCIATION_COPY_NONATOMIC`

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

`OBJC_ASSOCIATION_RETAIN`

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

`OBJC_ASSOCIATION_COPY`

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

# Mac OS X Version 10.5 Delta

---

The low-level Objective-C runtime API is significantly updated in Mac 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 41)
- [object\\_getClassName](#) (page 42)
- [object\\_getIndexedIvars](#) (page 42)
- [object\\_setInstanceVariable](#) (page 44)
- [object\\_getInstanceVariable](#) (page 43)

The following function is modified:

- [object\\_copy](#) (page 41) (The `nBytes` parameter is changed from `unsigned` to `size_t`.)

The following functions are added:

- [object\\_getClass](#) (page 42)
- [object\\_setClass](#) (page 44)
- [object\\_getIvar](#) (page 43)
- [object\\_setIvar](#) (page 44)

The following functions are deprecated:

- `object_copyFromZone`: deprecated in favor of [object\\_copy](#) (page 41)
- `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 35)  
[objc\\_lookupClass](#) (page 37)  
[objc\\_getClass](#) (page 34)  
[objc\\_getMetaClass](#) (page 36)  
[class\\_getVersion](#) (page 24)  
[class\\_getInstanceVariable](#) (page 21)  
[class\\_getInstanceMethod](#) (page 20)  
[class\\_getClassMethod](#) (page 20)

The following function is modified:

`class_createInstance`: *idxIvars* parameter Changed from unsigned to *size\_t*

The following functions are added:

[class\\_getName](#) (page 23)  
[class\\_getSuperclass](#) (page 23)  
[class\\_isMetaClass](#) (page 24)  
[class\\_copyMethodList](#) (page 18)  
[class\\_getMethodImplementation](#) (page 22)  
[class\\_getMethodImplementation\\_stret](#) (page 22)  
[class\\_respondsToSelector](#) (page 25)  
[class\\_conformsToProtocol](#) (page 17)  
[class\\_copyProtocolList](#) (page 19)  
[class\\_copyIvarList](#) (page 17)

The following functions are deprecated:

`objc_getClasses`: deprecated in favor of [objc\\_getClassList](#) (page 35)  
`class_createInstanceFromZone`: deprecated in favor of [class\\_createInstance](#) (page 19)  
`class_nextMethodList`: deprecated in favor of new [class\\_copyMethodList](#) (page 18)  
`class_lookupMethod`: deprecated in favor of [class\\_getMethodImplementation](#) (page 22)  
`class_respondsToMethod`: deprecated in favor of [class\\_respondsToSelector](#) (page 25)

The following function is used only by ZeroLink:

`objc_getRequiredClass`

## Class Manipulation

---

The following function is unchanged:

`class_setVersion` (page 26)

The following functions are added:

`objc_allocateClassPair` (page 33)

`objc_registerClassPair` (page 40)

`objc_duplicateClass` (page 34)

`class_addMethod` (page 16)

`class_addIvar` (page 15)

`class_addProtocol` (page 17)

The following functions are deprecated:

`objc_addClass`: deprecated in favor of `objc_allocateClassPair` (page 33) and `objc_registerClassPair` (page 40)

`class_addMethods`: deprecated in favor of new `class_addMethod` (page 16)

`class_removeMethods`: deprecated with no substitute

`class_poseAs`: deprecated in favor of categories and `method_setImplementation` (page 33)

## Methods

---

The following function is unchanged:

`method_getNumberOfArguments` (page 32)

The following functions are added:

`method_getName` (page 31)

`method_getImplementation` (page 31)

`method_getTypeEncoding` (page 32)

`method_copyReturnType` (page 30)

`method_copyArgumentType` (page 30)

`method_setImplementation` (page 33)

The following functions are deprecated:

`method_getArgumentInfo`

`method_getSizeOfArguments`

## Instance Variables

---

The following functions are added:

[ivar\\_getName](#) (page 27)  
[ivar\\_getTypeEncoding](#) (page 28)  
[ivar\\_getOffset](#) (page 27)

## Selectors

---

The following functions are unchanged:

[sel\\_getName](#) (page 49)  
[sel\\_registerName](#) (page 50)  
[sel\\_getUid](#) (page 49)

The following function is added:

[sel\\_isEqual](#) (page 50)

The following function is deprecated:

`sel_isMapped`: deprecated with no substitute

## Runtime

---

The following functions are deprecated favor of `dylid`:

`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`  
`_objclnit`

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 37)
objc_msgSend_stret (page 39)
objc_msgSendSuper (page 38)
objc_msgSendSuper_stret (page 38)
objc_msgSendSuper_stret (page 38)
```

The following functions are removed:

objc_msgSendv	Given an argument list, send a message with a simple return value.
objc_msgSendv_stret	Given an argument list, send a message with a data-structure return value.
objc_msgSendv_fpret	Given an argument list, send a message with a floating point return value.

## Protocols

---

The following functions are added:

```
objc_getProtocol (page 36)
objc_copyProtocolList (page 33)
```

## Exceptions

---

The following functions are unchanged:

```
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.

**Table A-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()

Variable	Substitution
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 A-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 A-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 {...};
objc_category {...};
objc_method_list {...};
objc_ivar_list {...};
objc_protocol_list {...};
objc_cache {...};
objc_module {...};
objc_symtab {...};

```



# Document Revision History

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

Date	Notes
2009-10-19	Added functions related to associative references.
2009-06-02	Updated for Mac 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 <code>object_getIndexedIvars</code> .
2007-10-31	Updated for Mac OS X v10.5. Corrected the code example for the <code>objc_getClassList</code> function.
2007-05-25	Included new features in Objective-C 2.0.
2005-10-04	Minor correction to <code>CreateClassDefinition</code> function and definitions of <code>marg_</code> macros.
2005-08-11	Corrected errors and documented macros.
	Corrected declaration of <code>class_getClassMethod</code> (page 20).
	Renamed the “Class Handler Callback” section to <code>ClassHandlerCallback</code> and added example function declaration to the description.
	Corrected result description of <code>method_getArgumentInfo</code> .
	Documented <code>YES</code> and <code>NO</code> macros in “Macros”.
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 <i>The Objective-C Programming Language</i> .

## REVISION HISTORY

### Document Revision History

# Index

---

## A

---

Associative Object Behaviors [56](#)

## C

---

Category **data type** [51](#)  
Class **structure** [51](#)  
class\_addIvar **function** [15](#)  
class\_addMethod **function** [16](#)  
class\_addProtocol **function** [17](#)  
class\_conformsToProtocol **function** [17](#)  
class\_copyIvarList **function** [17](#)  
class\_copyMethodList **function** [18](#)  
class\_copyPropertyList **function** [18](#)  
class\_copyProtocolList **function** [19](#)  
class\_createInstance **function** [19](#)  
class\_getClassMethod **function** [20](#)  
class\_getClassVariable **function** [20](#)  
class\_getInstanceMethod **function** [20](#)  
class\_getInstanceSize **function** [21](#)  
class\_getInstanceVariable **function** [21](#)  
class\_getIvarLayout **function** [21](#)  
class\_getMethodImplementation **function** [22](#)  
class\_getMethodImplementation\_stret **function** [22](#)  
class\_getName **function** [23](#)  
class\_getProperty **function** [23](#)  
class\_getSuperclass **function** [23](#)  
class\_getVersion **function** [24](#)  
class\_getWeakIvarLayout **function** [24](#)  
class\_isMetaClass **function** [24](#)  
class\_replaceMethod **function** [25](#)  
class\_respondToSelector **function** [25](#)  
class\_setIvarLayout **function** [26](#)  
class\_setSuperclass **function** [26](#)  
class\_setVersion **function** [26](#)  
class\_setWeakIvarLayout **function** [27](#)

## D

---

development environment [9, 57](#)

## I

---

id **data type** [54](#)  
IMP **data type** [52](#)  
Ivar **data type** [51](#)  
ivar\_getName **function** [27](#)  
ivar\_getOffset **function** [27](#)  
ivar\_getTypeEncoding **function** [28](#)

## M

---

marg\_free **function** [28](#)  
marg\_getRef **function** [28](#)  
marg\_getValue **function** [29](#)  
marg\_list **data type** [52](#)  
marg\_malloc **macro** [29](#)  
marg\_setValue **function** [29](#)  
Method **data type** [51](#)  
method\_copyArgumentType **function** [30](#)  
method\_copyReturnType **function** [30](#)  
method\_exchangeImplementations **function** [30](#)  
method\_getArgumentType **function** [31](#)  
method\_getImplementation **function** [31](#)  
method\_getName **function** [31](#)  
method\_getNumberOfArguments **function** [32](#)  
method\_getReturnType **function** [32](#)  
method\_getTypeEncoding **function** [32](#)  
method\_setImplementation **function** [33](#)

## O

---

objc\_allocateClassPair **function** [33](#)  
objc\_AssociationPolicy **data type** [55](#)

OBJC\_ASSOCIATION\_ASSIGN **constant** 56  
 OBJC\_ASSOCIATION\_COPY **constant** 56  
 OBJC\_ASSOCIATION\_COPY\_NONATOMIC **constant** 56  
 OBJC\_ASSOCIATION\_RETAIN **constant** 56  
 OBJC\_ASSOCIATION\_RETAIN\_NONATOMIC **constant** 56  
 objc\_cache **structure** 53  
 objc\_copyProtocolList **function** 33  
 objc\_duplicateClass **function** 34  
 objc\_getAssociatedObject **function** 34  
 objc\_getClass **function** 34  
 objc\_getClassList **function** 35  
 objc\_getFutureClass **function** 36  
 objc\_getMetaClass **function** 36  
 objc\_getProtocol **function** 36  
 objc\_getRequiredClass **function** 36  
 objc\_lookupClass **function** 37  
 objc\_method\_list **structure** 52  
 objc\_msgSend **function** 37  
 objc\_msgSendSuper **function** 38  
 objc\_msgSendSuper\_stret **function** 38  
 objc\_msgSend\_fpret **function** 39  
 objc\_msgSend\_stret **function** 39  
 objc\_object **structure** 54  
 objc\_property\_t **data type** 51  
 objc\_protocol\_list **structure** 53  
 objc\_registerClassPair **function** 40  
 objc\_removeAssociatedObjects **function** 40  
 objc\_setAssociatedObject **function** 40  
 objc\_setFutureClass **function** 41  
 objc\_super **structure** 55  
 object\_copy **function** 41  
 object\_dispose **function** 41  
 object\_getClass **function** 42  
 object\_getClassName **function** 42  
 object\_getIndexedIvars **function** 42  
 object\_getInstanceVariable **function** 43  
 object\_getIvar **function** 43  
 object\_setClass **function** 44  
 object\_setInstanceVariable **function** 44  
 object\_setIvar **function** 44

## P

---

property\_getAttributes **function** 45  
 property\_getName **function** 45  
 protocol\_conformsToProtocol **function** 45  
 protocol\_copyMethodDescriptionList **function** 46  
 protocol\_copyPropertyList **function** 46  
 protocol\_copyProtocolList **function** 47  
 protocol\_getMethodDescription **function** 47  
 protocol\_getName **function** 48  
 protocol\_getProperty **function** 48

protocol\_isEqual **function** 49

## S

---

SEL **data type** 52  
 @selector() **directive** 52  
 sel\_getName **function** 49  
 sel\_getUid **function** 49  
 sel\_isEqual **function** 50  
 sel\_registerName **function** 50