
Objective-C 2.0 Runtime Reference

Cocoa > Objective-C Language



2007-12-11



Apple Inc.
© 2002, 2007 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, Cocoa, eMac, Mac, Mac OS, and Objective-C are trademarks of Apple Inc., registered in the United States and other countries.

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 2.0 Runtime Reference 9

Overview	9
Who Should Read This Document	10
Functions by Task	10
Working with Classes	10
Adding Classes	11
Instantiating Classes	12
Working with Instances	12
Obtaining Class Definitions	12
Working with Instance Variables	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	22
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_respondsToSelector	25
class_setIvarLayout	25
class_setSuperclass	25
class_setVersion	26
class_setWeakIvarLayout	26
ivar_getName	27
ivar_getOffset	27
ivar_getTypeEncoding	27
marg_free	27
marg_getRef	28
marg_getValue	28
marg_malloc	28
marg_setValue	29
method_copyArgumentType	29
method_copyReturnType	30
method_exchangeImplementations	30
method_getArgumentType	30
method_getImplementation	30
method_getName	31
method_getNumberOfArguments	31
method_getReturnType	31
method_getTypeEncoding	32
method_setImplementation	32
objc_allocateClassPair	32
objc_copyProtocolList	33
objc_duplicateClass	33
objc_getClass	33
objc_getClassList	34
objc_getFutureClass	35
objc_getMetaClass	35
objc_getProtocol	35
objc_getRequiredClass	36
objc_lookUpClass	36
objc_msgSend	36
objc_msgSendSuper	37
objc_msgSendSuper_stret	37
objc_msgSend_fpret	38
objc_msgSend_stret	38
objc_registerClassPair	39
objc_setFutureClass	39
object_copy	39
object_dispose	40
object_getClass	40

object_getClassName	40
object_getIndexedIvars	40
object_getInstanceVariable	41
object_getIvar	41
object_setClass	42
object_setInstanceVariable	42
object_setIvar	43
property_getAttributes	43
property_getName	43
protocol_conformsToProtocol	44
protocol_copyMethodDescriptionList	44
protocol_copyPropertyList	45
protocol_copyProtocolList	45
protocol_getMethodDescription	45
protocol_getName	46
protocol_getProperty	46
protocol_isEqual	47
sel_getName	47
sel_getUid	47
sel_isEqual	48
sel_registerName	48
Data Types	49
Class-Definition Data Structures	49
Instance Data Types	52
Boolean Value	53
Constants	54
Boolean Values	54
Null Values	54

Appendix A Mac OS X Version 10.5 Delta 55

Runtime Functions	55
Basic types	55
Instances	55
Class Inspection	56
Class Manipulation	57
Methods	57
Instance Variables	58
Selectors	58
Runtime	58
Messaging	59
Protocols	59
Exceptions	59
Synchronization	60
NXHashTable and NXMapTable	60
Structures	60

[Document Revision History](#) 63

[Index](#) 65

Tables and Listings

Objective-C 2.0 Runtime Reference 9

Listing 1 Using objc_getClassList 34

Appendix A Mac OS X Version 10.5 Delta 55

Table A-1 Substitutions for objc_class 60

Table A-2 Substitutions for objc_method 61

Table A-3 Substitutions for objc_ivar 61

Objective-C 2.0 Runtime Reference

Companion guide:	The Objective-C 2.0 Programming Language
Declared in:	MacTypes.h NSObjCRuntime.h wintypes.h

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 25)

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 22)

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

[class_setIvarLayout](#) (page 25)

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 26)

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_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 35)

Used by CoreFoundation's toll-free bridging.

[objc_setFutureClass](#) (page 39)

Used by CoreFoundation's toll-free bridging.

Adding Classes

[objc_allocateClassPair](#) (page 32)

Creates a new class and metaclass.

[objc_registerClassPair](#) (page 39)

Registers a class that was allocated using `objc_allocateClassPair`.

[objc_duplicateClass](#) (page 33)

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 39)

Returns a copy of a given object.

[object_dispose](#) (page 40)

Frees the memory occupied by a given object.

[object_setInstanceVariable](#) (page 42)

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

[object_getInstanceVariable](#) (page 41)

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

[object_getIndexedIvars](#) (page 40)

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

[object_getIvar](#) (page 41)

Reads the value of an instance variable in an object.

[object_setIvar](#) (page 43)

Sets the value of an instance variable in an object.

[object_getClassName](#) (page 40)

Returns the class name of a given object.

[object_getClass](#) (page 40)

Returns the class of an object.

[object_setClass](#) (page 42)

Sets the class of an object.

Obtaining Class Definitions

[objc_getClassList](#) (page 34)

Obtains the list of registered class definitions.

[objc_lookupClass](#) (page 36)

Returns the class definition of a specified class.

[objc_getClass](#) (page 33)

Returns the class definition of a specified class.

[objc_getRequiredClass](#) (page 36)

Returns the class definition of a specified class.

[objc_getMetaClass](#) (page 35)

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 27)

Returns the type string of an instance variable.

[ivar_getOffset](#) (page 27)

Returns the offset of an instance variable.

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 36) sends a message with a simple return value to an instance of a class.
- [objc_msgSend_stret](#) (page 38) sends a message with a data-structure return value to an instance of a class.
- [objc_msgSendSuper](#) (page 37) sends a message with a simple return value to the superclass of an instance of a class.
- [objc_msgSendSuper_stret](#) (page 37) sends a message with a data-structure return value to the superclass of an instance of a class.

[objc_msgSend](#) (page 36)

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

[objc_msgSend_fpret](#) (page 38)

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

[objc_msgSend_stret](#) (page 38)

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

[objc_msgSendSuper](#) (page 37)

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

[objc_msgSendSuper_stret](#) (page 37)

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 28)

Macro that allocates an argument list.

[marg_free](#) (page 27)

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 28)

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 30)

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 29)

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

[method_getReturnType](#) (page 31)

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

[method_getNumberOfArguments](#) (page 31)

Returns the number of arguments accepted by a method.

[method_getArgumentType](#) (page 30)

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

[method_setImplementation](#) (page 32)

Sets the implementation of a method.

[method_exchangeImplementations](#) (page 30)

Exchanges the implementations of two methods.

Working with Selectors

[sel_getName](#) (page 47)

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

[sel_registerName](#) (page 48)

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

[sel_getUid](#) (page 47)

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

[sel_isEqual](#) (page 48)

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

Working with Protocols

[objc_getProtocol](#) (page 35)

Returns a specified protocol.

[objc_copyProtocolList](#) (page 33)

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

[protocol_getName](#) (page 46)

Returns a the name of a protocol.

[protocol_isEqual](#) (page 47)

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

[protocol_copyMethodDescriptionList](#) (page 44)

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

[protocol_getMethodDescription](#) (page 45)

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

[protocol_copyPropertyList](#) (page 45)

Returns an array of the properties declared by a protocol.

[protocol_getProperty](#) (page 46)

Returns the specified property of a given protocol.

[protocol_copyProtocolList](#) (page 45)

Returns an array of the protocols adopted by a protocol.

[protocol_conformsToProtocol](#) (page 44)

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

Working with Properties

[property_getName](#) (page 43)

Returns the name of a property.

[property_getAttributes](#) (page 43)

Returns the attribute string of an property.

Functions

class_addIvar

Adds a new instance variable to a class.

```
B00L 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 32) and before [objc_registerClassPair](#) (page 39). 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 *The Objective-C 2.0 Programming Language* > The Runtime System > “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 32).

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 ValueYES 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 50). Pass the selector of the method you want to retrieve.

Return Value

A pointer to the [Method](#) (page 49) 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 49) 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 49) 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 49) 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_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 49) 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 41) and [object_setIvar](#) (page 43) 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.

Declared In

runtime.h

marg_free

Macro that releases an argument list.

```
marg_free(margs)
```

Parameters

margs

A pointer of type [marg_list](#) (page 50). 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 50).

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 50).

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 50). Pass the variable that contains the argument list pointer.

method

A pointer to an `Method` (page 49) 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 50).

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 49) 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 39). 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_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 36) 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 36) 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 33), 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 33) is different from this function in that if the class is not registered, [objc_getClass](#) (page 33) 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 53) 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 50). 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 53) 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 50). 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 50). 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_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 49) 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` 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

A pointer to a value. Pass a pointer to the new value for the instance variable.

Return Value

A pointer to the [Ivar](#) (page 49) 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 42) 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.

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 required 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*, or NULL if *p* does not contain a method with that specification.

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.

```
B00L 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 50). 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 50) specifying the selector for the named method.

Discussion

The implementation of this method is identical to the implementation of [sel_registerName](#) (page 48).

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.

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 50) 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 12).

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 48).

When using selectors, you must use the value returned from [sel_registerName](#) (page 48) 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 49) 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 49) 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 49) 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 52) pointer to an instance of a class.
- [objc_object](#) (page 52) represents an instance of a class.
- [objc_super](#) (page 53) 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 52). Specifies an instance of a class.

`class`

A pointer to an `Class` (page 49) 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.

Boolean Value

BOOL

Type to represent a Boolean value.

```
typedef signed char BOOL;
```

Discussion

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

For values, see “[Boolean Values](#)” (page 54).

Availability

Available in Mac OS X v10.1 and later.

Declared In

wintypes.h

Constants

Boolean Values

These macros define convenient constants to represent Boolean values.

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

Constants

YES

Defines YES as 1.

Available in Mac OS X v10.0 and later.

Declared in NSObjCRuntime.h

NO

Defines NO as 0.

Available in Mac OS X v10.0 and later.

Declared in NSObjCRuntime.h

Declared In

objc.h

Null Values

These macros define null values for classes and instances.

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

Constants

nil

Defines the id of a null instance.

Available in Mac OS X v10.0 and later.

Declared in MacTypes.h

Nil

Defines the id of a null class.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in NSObjCRuntime.h

Declared In

objc.h

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 40)
- [object_getClassName](#) (page 40)
- [object_getIndexedIvars](#) (page 40)
- [object_setInstanceVariable](#) (page 42)
- [object_getInstanceVariable](#) (page 41)

The following function is modified:

- [object_copy](#) (page 39) (The `nBytes` parameter is changed from `unsigned` to `size_t`.)

The following functions are added:

- [object_getClass](#) (page 40)
- [object_setClass](#) (page 42)
- [object_getIvar](#) (page 41)
- [object_setIvar](#) (page 43)

The following functions are deprecated:

- `object_copyFromZone`: deprecated in favor of [object_copy](#) (page 39)
- `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 34)
- [objc_lookUpClass](#) (page 36)
- [objc_getClass](#) (page 33)
- [objc_getMetaClass](#) (page 35)
- [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 34)
- `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_respondsToSelector`: 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 32)
`objc_registerClassPair` (page 39)
`objc_duplicateClass` (page 33)
`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 32) and
`objc_registerClassPair` (page 39)
`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 32)

Methods

The following function is unchanged:

`method_getNumberOfArguments` (page 31)

The following functions are added:

`method_getName` (page 31)
`method_getImplementation` (page 30)
`method_getTypeEncoding` (page 32)
`method_copyReturnType` (page 30)
`method_copyArgumentType` (page 29)
`method_setImplementation` (page 32)

The following functions are deprecated:

`method_getArgumentInfo`
`method_getSizeOfArguments`

Instance Variables

The following functions are added:

`ivar_getName` (page 27)
`ivar_getTypeEncoding` (page 27)
`ivar_getOffset` (page 27)

Selectors

The following functions are unchanged:

`sel_getName` (page 47)
`sel_registerName` (page 48)
`sel_getUid` (page 47)

The following function is added:

`sel_isEqual` (page 48)

The following function is deprecated:

`sel_isMapped`: deprecated with no substitute

Runtime

The following functions are deprecated favor of `dyld`:

`objc_loadModules`
`objc_loadModule`
`objc_unloadModules`

The following functions are deprecated:

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

The following previously undocumented functions are deprecated with no substitute:

`objc_getOrigClass`
`_objc_create_zone`
`_objc_error`
`_objc_flush_caches`
`_objc_resolve_categories_for_class`

`_objc_setClassLoader`
`_objc_setNilReceiver`
`_objc_getNilReceiver`
`_objcInit`

The following undocumented functions are unchanged:

`_objc_getFreedObjectClass`
`instrumentObjcMessageSends`
`_objc_debug_class_hash`
`_class_printDuplicateCacheEntries`
`_class_printMethodCaches`
`_class_printMethodCacheStatistics`

Messaging

The following functions are unchanged:

[objc_msgSend](#) (page 36)
[objc_msgSend_stret](#) (page 38)
[objc_msgSendSuper](#) (page 37)
[objc_msgSendSuper_stret](#) (page 37)
[objc_msgSendSuper_stret](#) (page 37)

The following functions are removed:

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

Protocols

The following functions are added:

[objc_getProtocol](#) (page 35)
[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
<code>struct objc_class *isa;</code>	<code>object_getClass()</code> , <code>object_setClass()</code>
<code>struct objc_class *super_class;</code>	<code>class_getSuperclass()</code>
<code>const char *name;</code>	<code>class_getName()</code>
<code>long version;</code>	<code>class_getVersion()</code> , <code>class_setVersion()</code>

Variable	Substitution
long info;	class_isMetaClass()
long instance_size;	no substitute
struct objc_ivar_list *ivars;	class_copyIvarList(), class_addIvar()
struct objc_method_list **methodLists;	class_copyMethodList(), class_addMethod()
struct objc_cache *cache;	no substitute
struct objc_protocol_list *protocols;	class_copyProtocolList(), class_addProtocol()

Table 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 2.0 Runtime Reference*.

Date	Notes
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 2.0 Programming Language</i> .

REVISION HISTORY

Document Revision History

Index

B

BOOL data type [53](#)
Boolean Values [54](#)

C

Category data type [49](#)
Class structure [49](#)
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 [22](#)
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_respondToSelector function [25](#)
class_setIvarLayout function [25](#)
class_setSuperclass function [25](#)
class_setVersion function [26](#)
class_setWeakIvarLayout function [26](#)

D

development environment [10](#), [55](#)

I

id data type [52](#)
IMP data type [50](#)
Ivar data type [49](#)
ivar_getName function [27](#)
ivar_getOffset function [27](#)
ivar_getTypeEncoding function [27](#)

M

marg_free function [27](#)
marg_getRef function [28](#)
marg_getValue function [28](#)
marg_list data type [50](#)
marg_malloc macro [28](#)
marg_setValue function [29](#)
Method data type [49](#)
method_copyArgumentType function [29](#)
method_copyReturnType function [30](#)
method_exchangeImplementations function [30](#)
method_getArgumentType function [30](#)
method_getImplementation function [30](#)
method_getName function [31](#)
method_getNumberOfArguments function [31](#)
method_getReturnType function [31](#)
method_getTypeEncoding function [32](#)
method_setImplementation function [32](#)

N

Nil constant [54](#)
nil constant [54](#)

NO constant [54](#)
 Null Values [54](#)

O

objc_allocateClassPair [function 32](#)
 objc_cache [structure 51](#)
 objc_copyProtocolList [function 33](#)
 objc_duplicateClass [function 33](#)
 objc_getClass [function 33](#)
 objc_getClassList [function 34](#)
 objc_getFutureClass [function 35](#)
 objc_getMetaClass [function 35](#)
 objc_getProtocol [function 35](#)
 objc_getRequiredClass [function 36](#)
 objc_lookupClass [function 36](#)
 objc_method_list [structure 51](#)
 objc_msgSend [function 36](#)
 objc_msgSendSuper [function 37](#)
 objc_msgSendSuper_stret [function 37](#)
 objc_msgSend_fpret [function 38](#)
 objc_msgSend_stret [function 38](#)
 objc_object [structure 52](#)
 objc_property_t [data type 50](#)
 objc_protocol_list [structure 52](#)
 objc_registerClassPair [function 39](#)
 objc_setFutureClass [function 39](#)
 objc_super [structure 53](#)
 object_copy [function 39](#)
 object_dispose [function 40](#)
 object_getClass [function 40](#)
 object_getClassName [function 40](#)
 object_getIndexedIvars [function 40](#)
 object_getInstanceVariable [function 41](#)
 object_getIvar [function 41](#)
 object_setClass [function 42](#)
 object_setInstanceVariable [function 42](#)
 object_setIvar [function 43](#)

P

property_getAttributes [function 43](#)
 property_getName [function 43](#)
 protocol_conformsToProtocol [function 44](#)
 protocol_copyMethodDescriptionList [function 44](#)
 protocol_copyPropertyList [function 45](#)
 protocol_copyProtocolList [function 45](#)
 protocol_getMethodDescription [function 45](#)
 protocol_getName [function 46](#)
 protocol_getProperty [function 46](#)

protocol_isEqual [function 47](#)

S

SEL data type [50](#)
 @selector() [directive 50](#)
 sel_getName [function 47](#)
 sel_getUid [function 47](#)
 sel_isEqual [function 48](#)
 sel_registerName [function 48](#)

Y

YES constant [54](#)