Copyright ©1996 by NeXT Software, Inc. All Rights Reserved.

Defined Types

Cache

DECLARED IN objc/objc-class.h

SYNOPSIS typedef struct objc_cache ***Cache**;

DESCRIPTION This is the defined type for a class's run-time cache of frequently used methods. Each class has its own cache.

Category

DECLARED IN objc/objc-class.h

SYNOPSIS typedef struct objc_category ***Category**;

DESCRIPTION This is the type name for the structure that contains information about a category definition.

Ivar

DECLARED IN objc/objc-class.h

SYNOPSIS typedef struct objc_ivar ***Ivar**;

DESCRIPTION The Ivar type identifies a structure containing information about a single instance variableÐincluding the name of the variable, its type, and its location in the object data€structure.

marg_list

DECLARED IN objc/objc-class.h

SYNOPSIS typedef void ***marg_list**;

DESCRIPTION This type is a pointer to the arguments that were passed in a message. It's used by the Object class's **forward::** method.

Method

DECLARED IN objc/objc-class.h

SYNOPSIS typedef struct objc_method ***Method**;

DESCRIPTION The Method type designates a structure containing information about a single methodÐincluding its return and argument types, the method selector, and the location of the method implementation.

Module

DECLARED IN objc/objc-runtime.h

SYNOPSIS typedef struct objc_module ***Module**;

DESCRIPTION This data type refers to a file that contributes to an Objective€C program. The compiler produces a Module data structure for each file that it encounters.

Symbolic Constants

Type Constants

DECLARED IN objc/objc-class.h

SYNOPSIS

Constant	Meaning	Defined As
_C_ID	**id**	\`@'
_C_CLASS	Class	\`#'
_C_SEL	SEL	\`:'
_C_VOID	**void**	\`v'
_C_CHR	**char**	\`c'
_C_UCHR	**unsigned char**	\`C'
_C_SHT	**short**	\`s'
_C_USHT	**unsigned short**	\`S'
_C_INT	**int**	\`i'
_C_UINT	**unsigned int**	\`I'
_C_LNG	**long**	\`l'
_C_ULNG	**unsigned long**	\`L'
_C_FLT	**float**	\`f'
_C_DBL	**double**	\`d'
_C_UNDEF	an undefined type	\`?'
_C_PTR	a pointer	\`^'
_C_CHARPTR	**char ***	\`*'
_C_BFLD	a bitfield	\`b'
_C_ARY_B	begin an array	\`['
_C_ARY_E	end an array	\`]'
_C_UNION_B	begin a union	\`('
_C_UNION_E	end a union	\`)'
_C_STRUCT_B	begin a structure	\`{'
_C_STRUCT_E	end a structure	\`}'

DESCRIPTION These constants identify the character codes used to store method return and argument types. They're the same codes returned by the **@encode()** directive.

Structures

objc_cache

DECLARED IN objc/objc-class.h

SYNOPSIS

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

DESCRIPTION This structure stores a class-specific cache of the methods most recently used by instances of the class or by the class object. The Cache data type is defined as a pointer to an **objc_cache** structure.

objc_category

DECLARED IN objc/objc-class.h

SYNOPSIS

```
struct objc_category {
        char *category_name;
        char *class_name;
        struct objc_method_list *instance_methods;
        struct objc_method_list *class_methods;
        struct objc_protocol_list *protocols;
};
```

DESCRIPTION This structure stores the information contained in a category definition. Its fields are:

category_name	The name assigned to the category in source code
class_name	The name of the class that the category belongs to
instance_methods	A list of instance methods defined in the category
class_methods	A list of class methods defined in the category
protocols	A list of the protocols adopted in the category

The Category data type is defined as a pointer to an **obj_category** structure.

objc_class

DECLARED IN objc/objc-class.h

SYNOPSIS

```
struct objc_class {
        struct objc_class *isa;
        struct objc_class *super_class;
        const char *name;
        long version;
        long info;
        long instance_size;
        struct objc_ivar_list *ivars;
        struct objc_method_list *methods;
        struct objc_cache *cache;
        struct objc_protocol_list *protocols;
};
```

DESCRIPTION This structure holds information about a class definition. Its fields are:

isa	The metaclass of this class
super_class	The superclass of this class
name	The name of this class
version	The current version of the class (as set by **setVersion:**)
info	The current status of the class
instance_size	The number of bytes to allocate for an instance of the class
ivars	The instance variables declared in the class interface
methods	The instance methods defined in the class implementation
cache	The cache of recently used methods
protocols	The protocols adopted by the class

This structure is also used to store metaclass information, in which case the **methods** field lists class methods rather than instance methods.

The Class data type is defined (in **objc.h**) as a pointer to an **objc_class** structure.

objc_ivar

DECLARED IN objc/objc-class.h

SYNOPSIS

```
struct objc_ivar {
        char *ivar_name;
        char *ivar_type;
        int ivar_offset;
};
```

DESCRIPTION This structure describes a single instance variable. It's fields are:

ivar_name	The name of the instance variable
ivar_type	The data type declared for the instance variable
ivar_offset	The position of the variable in the object (as an offset in bytes)

The Ivar data type is defined as a pointer to an **objc_ivar** structure.

objc_ivar_list

DECLARED IN objc/objc-class.h

SYNOPSIS

```
struct objc_ivar_list {
        int ivar_count;
        struct objc_ivar ivar_list[1];
};
```

DESCRIPTION This structure holds information about the instance variables declared in a class definition. The first field, **ivar_count**, gives the number of variables declared and the second field, **ivar_list**, is a variable-length array of all the variables.

objc_method

DECLARED IN objc/objc-class.h

SYNOPSIS

```
struct objc_method {
        SEL method_name;
        char *method_types;
        IMP method_imp;
};
```

DESCRIPTION This structure describes a single method implemented by the class. The fields are:

method_name	The method selector (not the full name)
method_types	A string encoding the method return and argument types
method_imp	A pointer to the method implementation

The Method data type is defined as a pointer to an **objc_method** structure.

objc_method_description

DECLARED IN objc/Protocol.h

SYNOPSIS

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

DESCRIPTION This structure holds the method information returned by two methods defined in the Protocol class, **descriptionForClassMethod:** and **descriptionForInstanceMethod:**, and by two Object methods, **descriptionForMethod:** and **descriptionForInstanceMethod:**.

objc_method_description_list

DECLARED IN objc/Protocol.h

SYNOPSIS

```
struct objc_method_description_list {
        int count;
        struct objc_method_description list[1];
};
```

DESCRIPTION This structure points to a list of **objc_method_description** structures. Typically the list describes all the methods declared in a particular protocol.

objc_method_list

DECLARED IN objc/objc-class.h

SYNOPSIS

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

DESCRIPTION This structure lists all the class or all the instance methods defined within a class or category (within one group bracketed by **@implementation** and **@end**). Its fields are:

method_next	A pointer to another group of methods for the same class
method_count	The number of methods listed in this group
method_list	A variable-length array of method descriptions

Class methods and instance methods are listed in separate structures.

objc_module

DECLARED IN objc/objc-runtime.h

SYNOPSIS

```
struct objc_module {
        unsigned long version;
        unsigned long size;
        const char *name;
        Symtab symtab;
};
```

DESCRIPTION This structure holds information about an object file compiled from Objective€C source code. Its fields are:

version	The version of run-time data structures
size	The size of the module in bytes
name	The name of the file
symtab	An obsolete field

The Module data type is defined as a pointer to this structure.

objc_protocol_list

DECLARED IN objc/objc-class.h

SYNOPSIS

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

DESCRIPTION This structure lists all the protocols adopted by a class in one place. Separate lists are kept for the class interface and for each category that adopts protocols on the class's behalf. The fields of the structure are:

next	A pointer to another list of protocols adopted by the class
count	The number of protocols listed here
list	A variable-length array of Protocol objects

objc_super

DECLARED IN objc/objc-runtime.h

SYNOPSIS struct **objc_super** {
id **receiver**;
Class **class**;
};

DESCRIPTION This structure helps the messaging function find which method implementation to invoke in response to a message sent to **super**. Its fields are:

receiver	The receiver of the message (the object designated by **super**)
class	The class where the message is sent

Global Variables

Function Pointers

DECLARED IN objc/objc-runtime.h

SYNOPSIS id **(*_alloc)(**Class *aClass*, unsigned int *indexedIvarBytes***)**
id **(*_dealloc)(**Object **anObject***)**
id **(*_realloc)(**Object **anObject*, unsigned int *numBytes***)**
id **(*_copy)(**Object **anObject*, unsigned int *indexedIvarBytes***)**
id **(*_zoneAlloc)(**Class *aClass*, unsigned int *indexedIvarBytes*, NXZone **zone***)**
id **(*_zoneRealloc)(**Object **anObject*, unsigned int *numBytes*, NXZone **zone***)**
id **(*_zoneCopy)(**Object **anObject*, unsigned int *indexedIvarBytes*, NXZone **zone***)**
void **(*_error)(**Object **anObject*, const char **format*, va_list *ap***)**

DESCRIPTION These variables point to the functions that the run-time system uses to manage memory and handle errors. By reassigning a variable, a function can be replaced with another of the same type. The example below shows a temporary reassignment of the **_zoneAlloc** function:

```
id (*theFunction)();
theFunction = _zoneAlloc;
_zoneAlloc = someOtherFunction;
/*
 * code that calls the class_createInstanceFromZone() function,
 * or sends alloc and allocFromZone: messages, goes here
 */
_zoneAlloc = theFunction;
```

- **_alloc** points to the function, called through **class_createInstance()**, used to allocate memory for new instances, and **_zoneAlloc** points to the function, called through **class_createInstanceFromZone()**, used to allocate the memory for a new instance from a specified *zone*.
- **_dealloc** points to the function, called through **object_dispose()**, used to free instances.
- **_realloc** points to the function, called through **object_realloc()**, used to reallocate memory for an object, and **_zoneRealloc** points to the function, called through **object_reallocFromZone()**, used to reallocate memory from a specified *zone*.
- **_copy** points to the function, called through **object_copy()**, used to create an exact copy of an object, and **_zoneCopy** points to the function, called through **object_copyFromZone()**, used to create the copy from memory in the specified *zone*.

- **_error** points to the function that the run-time system calls in response to an error. By default, it prints formatted error messages to the standard error stream (or logs them to the console if there is no standard error stream) and calls **abort()** to produce a core file.