OS X ABI Mach-O File Format Reference



Contents

[**OS X ABI Mach-O File Format Reference**](#page4) 4

[Overview](#page4) 4

[Basic Structure](#page4) 4

[Header Structure and Load Commands](#page5) 5

[Segments](#page6) 6

[Sections](#page7) 7

[Data Types](#page9) 9

[Header Data Structure](#page9) 9

[Load Command Data Structures](#page16) 16

[Symbol Table and Related Data Structures](#page44) 44

[Relocation Data Structures](#page57) 57

[Universal Binaries and 32-bit/64-bit PowerPC Binaries](#page63) 63

[**Document Revision History**](#page67) 67



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

2

Figures and Tables

[**OS X ABI Mach-O File Format Reference**](#page4) 4

[Figure 1](#page5) [Mach-O file format basic structure](#page5) 5

[Table 1](#page8) [The sections of a \_\_TEXT segment](#page8) 8

[Table 2](#page8) [The sections of a \_\_DATA segment](#page8) 8

[Table 3](#page9) [The sections of a \_\_IMPORT segment](#page9) 9

[Table 4](#page16) [Mach-O load commands](#page16) 16



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

3

OS X ABI Mach-O File Format Reference



**Declared in** fat.h

loader.h

nlist.h

reloc.h



Overview

This document describes the structure of the Mach-O (Mach object) file format, which is the standard used to store programs and libraries on disk in the Mac app binary interface (ABI). To understand how the Xcode tools work with Mach-O files, and to perform low-level debugging tasks, you need to understand this information.

The Mach-O file format provides both intermediate (during the build process) and final (after linking the final product) storage of machine code and data. It was designed as a flexible replacement for the BSD a.out format, to be used by the compiler and the static linker and to contain statically linked executable code at runtime. Features for dynamic linking were added as the goals of OS X evolved, resulting in a single file format for both statically linked and dynamically linked code.

Basic Structure

A Mach-O file contains three major regions (as shown in Figure 1):

* At the beginning of every Mach-O file is a **header structure** that identifies the file as a Mach-O file. The header also contains other basic file type information, indicates the target architecture, and contains flags specifying options that affect the interpretation of the rest of the file.
* Directly following the header are a series of variable-size **load commands** that specify the layout and linkage characteristics of the file. Among other information, the load commands can specify:
  + The initial layout of the file in virtual memory
  + The location of the symbol table (used for dynamic linking)
  + The initial execution state of the main thread of the program
  + The names of shared libraries that contain definitions for the main executable’s imported symbols



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

4

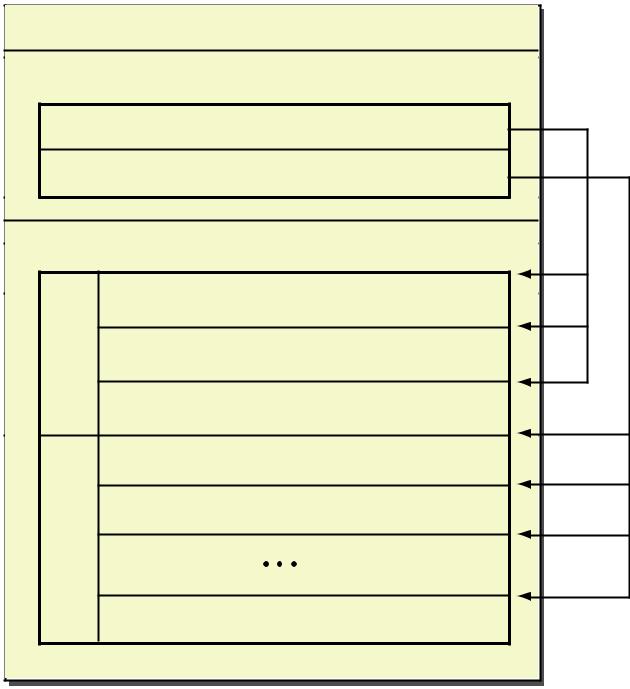
**OS X ABI Mach-O File Format Reference**

Overview



* Following the load commands, all Mach-O files contain the data of one or more segments. Each **segment** contains zero or more sections. Each **section** of a segment contains code or data of some particular type. Each segment defines a region of virtual memory that the dynamic linker maps into the address space of the process. The exact number and layout of segments and sections is specified by the load commands and the file type.
* In user-level fully linked Mach-O files, the last segment is the **link edit** segment. This segment contains the tables of link edit information, such as the symbol table, string table, and so forth, used by the dynamic loader to link an executable file or Mach-O bundle to its dependent libraries.

**Figure 1** Mach-O file format basic structure



|  |
| --- |
| Segment 2 Segment 1 |

Header

Load commands

Segment command 1

Segment command 2

Data

Section 1 data

Section 2 data

Section 3 data

Section 4 data

Section 5 data

Section n data

Various tables within a Mach-O file refer to sections by number. Section numbering begins at 1 (not 0) and continues across segment boundaries. Thus, the first segment in a file may contain sections 1 and 2 and the second segment may contain sections 3 and 4.

When using the Stabs debugging format, the symbol table also holds debugging information. When using DWARF, debugging information is stored in the image’s corresponding dSYM file, specified by the [uuid\_command](#page19) (page 19) structure

Header Structure and Load Commands

A Mach-O file contains code and data for one architecture. The header structure of a Mach-O file specifies the target architecture, which allows the kernel to ensure that, for example, code intended for PowerPC-based Macintosh computers is not executed on Intel-based Macintosh computers.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

5

**OS X ABI Mach-O File Format Reference**

Overview



You can group multiple Mach-O files (one for each architecture you want to support) in one binary using the format described in [“Universal Binaries and 32-bit/64-bit PowerPC Binaries”](#page63) (page 63).



**Note:** Binaries that contain object files for more than one architecture are not Mach-O files. Theyarchive one or more Mach-O files.



Segments and sections are normally accessed by name. Segments, by convention, are named using all uppercase letters preceded by two underscores (for example, \_\_TEXT); sections should be named using all lowercase letters preceded by two underscores (for example, \_\_text). This naming convention is standard, although not required for the tools to operate correctly.

Segments

A segment defines a range of bytes in a Mach-O file and the addresses and memory protection attributes at which those bytes are mapped into virtual memory when the dynamic linker loads the application. As such, segments are always virtual memory page aligned. A segment contains zero or more sections.

Segments that require more memory at runtime than they do at build time can specify a larger in-memory size than they actually have on disk. For example, the \_\_PAGEZERO segment generated by the linker for PowerPC executable files has a virtual memory size of one page but an on-disk size of 0. Because \_\_PAGEZERO contains no data, there is no need for it to occupy any space in the executable file.



**Note:** Sections that are to be filled with zeros must always be placed at the end of the segment.

Otherwise, the standard tools will not be able to successfully manipulate the Mach-O file.



For compactness, an intermediate object file contains only one segment. This segment has no name; it contains all the sections destined ultimately for different segments in the final object file. The data structure that defines a [section](#page23) (page 23) contains the name of the segment the section is intended for, and the static linker places each section in the final object file accordingly.

For best performance, segments should be aligned on virtual memory page boundaries—4096 bytes for PowerPC and x86 processors. To calculate the size of a segment, add up the size of each section, then round up the sum to the next virtual memory page boundary (4096 bytes, or 4 kilobytes). Using this algorithm, the minimum size of a segment is 4 kilobytes, and thereafter it is sized at 4 kilobyte increments.

The header and load commands are considered part of the first segment of the file for paging purposes. In an executable file, this generally means that the headers and load commands live at the start of the \_\_TEXT segment because that is the first segment that contains data. The \_\_PAGEZERO segment contains no data on disk, so it’s ignored for this purpose.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

6

**OS X ABI Mach-O File Format Reference**

Overview



These are the segments the standard OS X development tools (contained in the Xcode Tools CD) may include in an OS X executable:

* The static linker creates a \_\_PAGEZERO segment as the first segment of an executable file. This segment is located at virtual memory location 0 and has no protection rights assigned, the combination of which causes accesses to NULL, a common C programming error, to immediately crash. The \_\_PAGEZERO segment is the size of one full VM page for the current architecture (for Intel-based and PowerPC-based Macintosh computers, this is 4096 bytes or 0x1000 in hexadecimal). Because there is no data in the \_\_PAGEZERO segment, it occupies no space in the file (the file size in the segment command is 0).
* The \_\_TEXT segment contains executable code and other read-only data. To allow the kernel to map it directly from the executable into sharable memory, the static linker sets this segment’s virtual memory permissions to disallow writing. When the segment is mapped into memory, it can be shared among all processes interested in its contents. (This is primarily used with frameworks, bundles, and shared libraries, but it is possible to run multiple copies of the same executable in OS X, and this applies in that case as well.) The read-only attribute also means that the pages that make up the \_\_TEXT segment never need to be written back to disk. When the kernel needs to free up physical memory, it can simply discard one or more \_\_TEXT pages and re-read them from disk when they are next needed.
* The \_\_DATA segment contains writable data. The static linker sets the virtual memory permissions of this segment to allow both reading and writing. Because it is writable, the \_\_DATA segment of a framework or other shared library is logically copied for each process linking with the library. When memory pages such as those making up the \_\_DATA segment are readable and writable, the kernel marks them copy-on-write; therefore when a process writes to one of these pages, that process receives its own private copy of the page.
* The \_\_OBJC segment contains data used by the Objective-C language runtime support library.
* The \_\_IMPORT segment contains symbol stubs and non-lazy pointers to symbols not defined in the executable. This segment is generated only for executables targeted for the IA-32 architecture.
* The \_\_LINKEDIT segment contains raw data used by the dynamic linker, such as symbol, string, and relocation table entries.

Sections

The \_\_TEXT and \_\_DATA segments may contain a number of standard sections, listed in Table 1, [Table 2](#page8) (page 8), and [Table 3](#page9) (page 9). The \_\_OBJC segment contains a number of sections that are private to the Objective-C compiler. Note that the static linker and file analysis tools use the section type and attributes (instead of the section name) to determine how they should treat the section. The section name, type and attributes are explained further in the description of the [section](#page23) (page 23) data type.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

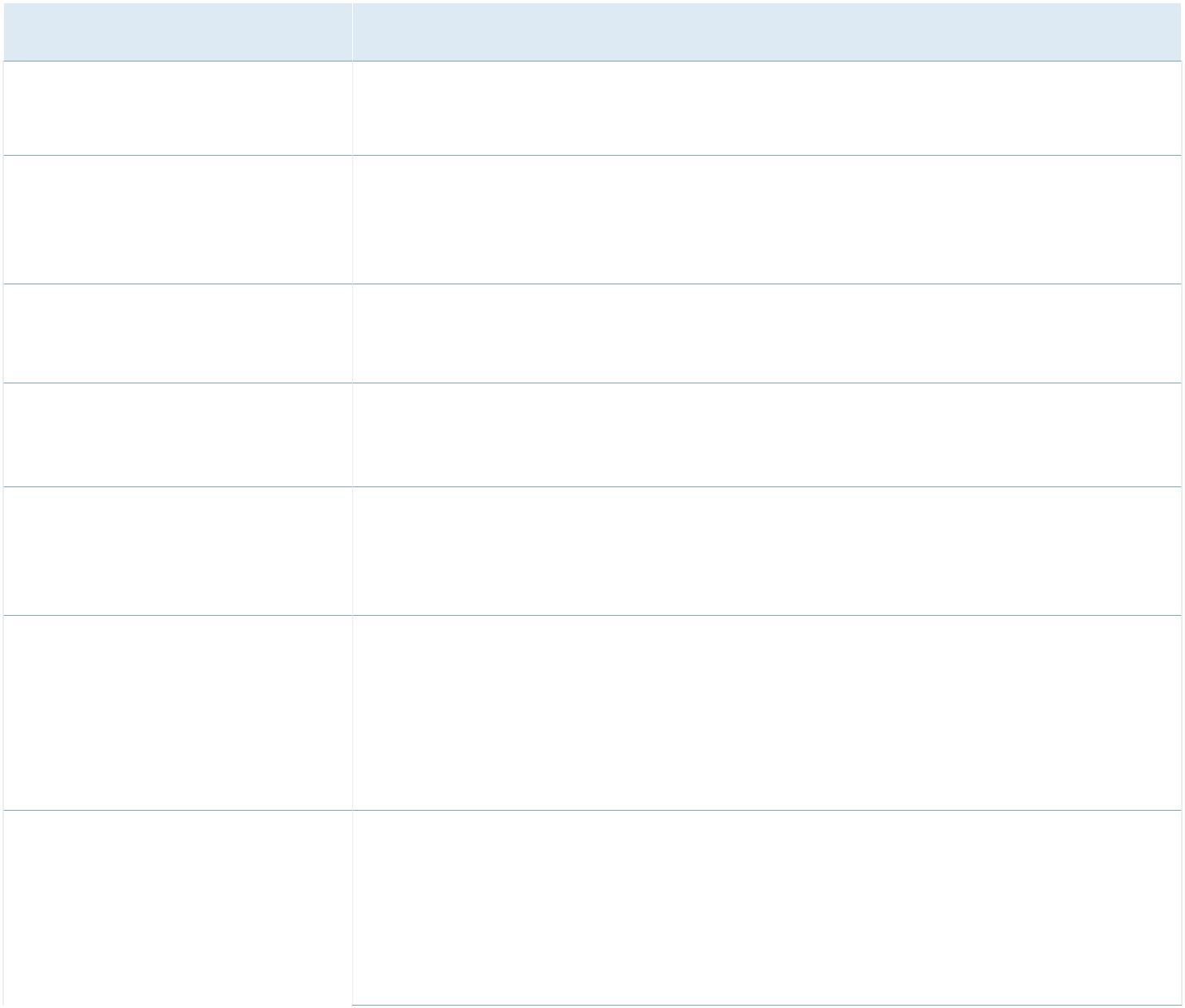
7

**OS X ABI Mach-O File Format Reference**

Overview



|  |  |  |
| --- | --- | --- |
| **Table 1** | The sections of a \_\_TEXT segment | |
| **Segment and section name** | | **Contents** |
| \_\_TEXT,\_\_text | | Executable machine code. The compiler generally places only executable |
|  |  | code in this section, no tables or data of any sort. |
| \_\_TEXT,\_\_cstring | | Constant C strings. A C string is a sequence of non-null bytes that ends |
|  |  | with a null byte ('\0'). The static linker coalesces constant C string |
|  |  | values, removing duplicates, when building the final product. |



\_\_TEXT,\_\_picsymbol\_-stub

\_\_TEXT,\_\_symbol\_stub

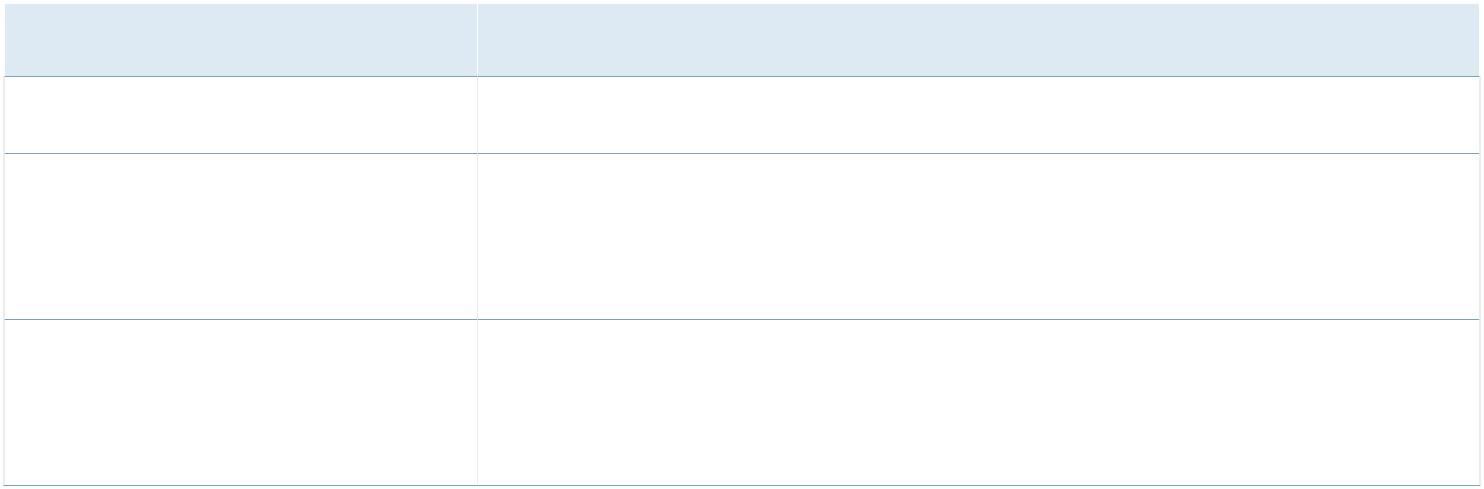
\_\_TEXT,\_\_const

Position-independent indirect symbol stubs. See “Position-Independent Code” in *Mach-O Programming Topics* for more information.

Indirect symbol stubs. See “Position-Independent Code” in *Mach-O* *Programming Topics* for more information.

Initialized constant variables. The compiler places all nonrelocatable data declared const in this section. (The compiler typically places uninitialized constant variables in a zero-filled section.)

|  |  |  |
| --- | --- | --- |
| \_\_TEXT,\_\_literal4 | | 4-byte literal values. The compiler places single-precision floating point |
|  |  | constants in this section. The static linker coalesces these values, |
|  |  | removing duplicates, when building the final product. With some |
|  |  | architectures, it’s more efficient for the compiler to use immediate load |
|  |  | instructions rather than adding to this section. |
| \_\_TEXT,\_\_literal8 | | 8-byte literal values. The compiler places double-precision floating point |
|  |  | constants in this section. The static linker coalesces these values, |
|  |  | removing duplicates, when building the final product. With some |
|  |  | architectures, it’s more efficient for the compiler to use immediate load |
|  |  | instructions rather than adding to this section. |
|  |  |  |
| **Table 2** | The sections of a \_\_DATA segment | |
| **Segment and section name** | | **Contents** |
| \_\_DATA,\_\_data | | Initialized mutable variables, such as writable C strings and data arrays. |
| \_\_DATA,\_\_la\_symbol\_ptr | | Lazy symbol pointers, which are indirect references to functions |
|  |  | imported from a different file. See “Position-Independent Code” in |
|  |  | *Mach-O Programming Topics* for more information. |
| \_\_DATA,\_\_nl\_symbol\_ptr | | Non-lazy symbol pointers, which are indirect references to data items |
|  |  | imported from a different file. See “Position-Independent Code” in |



*Mach-O Programming Topics* for more information.



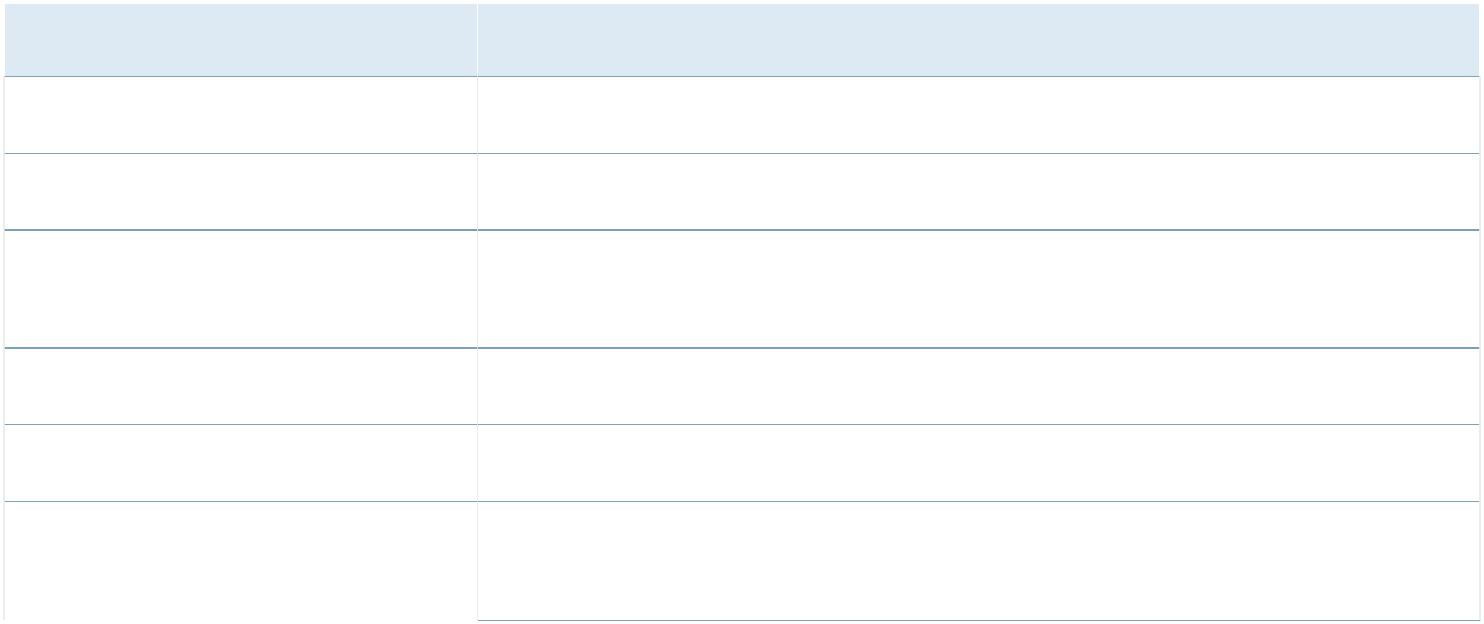
2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

8

[*mach\_header\_64*](#page13)

**OS X ABI Mach-O File Format Reference**

Data Types

**

**Segment and section name** **Contents**

\_\_DATA,\_\_dyld Placeholder section used by the dynamic linker.

\_\_DATA,\_\_const Initialized relocatable constant variables.

\_\_DATA,\_\_mod\_init\_func Module initialization functions. The C++ compiler places static

constructors here.

|  |  |  |  |
| --- | --- | --- | --- |
| \_\_DATA,\_\_mod\_term\_func | |  | Module termination functions. |
| \_\_DATA,\_\_bss | |  | Data for uninitialized static variables (for example, static int i;). |
| \_\_DATA,\_\_common | |  | Uninitialized imported symbol definitions (for example, int i;) |
|  |  |  | located in the global scope (outside of a function declaration). |
|  |  | |  |
| **Table 3** | The sections of a \_\_IMPORT segment | | |
| **Segment and section name** | | **Contents** | |

**

|  |  |
| --- | --- |
| \_\_IMPORT,\_\_jump\_table | Stubs for calls to functions in a dynamic library. |
| \_\_IMPORT,\_\_pointers | Non-lazy symbol pointers, which are direct references to functions |
|  | imported from a different file. |
|  |  |

**

**Note:** Compilers or any tools that create Mach-O files are free to define additional section names.

These additional names do not appear in Table 1.

**

Data Types

This reference describes the data types that compose a Mach-O file. Values for integer types in all Mach-O data structures are written using the host CPU’s byte ordering scheme, except for [fat\_header](#page64) (page 64) and

[fat\_arch](#page65) (page 65), which are written in big-endian byte order.

Header Data Structure

**mach\_header**

**

*Specifies the general attributes of a file. Appears at the beginning of object files targeted to 32-bit architectures.*

*Declared in /usr/include/mach-o/loader.h. See also* *(page 13).*

**

2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

9

**OS X ABI Mach-O File Format Reference**

Data Types



struct mach\_header

{

uint32\_t magic;

cpu\_type\_t cputype;

cpu\_subtype\_t cpusubtype;

uint32\_t filetype;

uint32\_t ncmds;

uint32\_t sizeofcmds;

uint32\_t flags;

};

**Fields**

magic

An integer containing a value identifying this file as a 32-bit Mach-O file. Use the constant MH\_MAGIC if the file is intended for use on a CPU with the same endianness as the computer on which the compiler is running. The constant MH\_CIGAM can be used when the byte ordering scheme of the target machine is the reverse of the host CPU.

cputype

An integer indicating the architecture you intend to use the file on. Appropriate values include:

* CPU\_TYPE\_POWERPC to target PowerPC-based Macintosh computers
* CPU\_TYPE\_I386 to target the Intel-based Macintosh computers

cpusubtype

An integer specifying the exact model of the CPU. To run on all PowerPC or x86 processors supported by the OS X kernel, this should be set to CPU\_SUBTYPE\_POWERPC\_ALL or CPU\_SUBTYPE\_I386\_ALL.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

10

**OS X ABI Mach-O File Format Reference**

Data Types



filetype

An integer indicating the usage and alignment of the file. Valid values for this field include:

* The MH\_OBJECT file type is the format used for intermediate object files. It is a very compact format containing all its sections in one segment. The compiler and assembler usually create one MH\_OBJECT file for each source code file. By convention, the file name extension for this format is .o.
* The MH\_EXECUTE file type is the format used by standard executable programs.
* The MH\_BUNDLE file type is the type typically used by code that you load at runtime (typically called bundles or plug-ins). By convention, the file name extension for this format is .bundle.
* The MH\_DYLIB file type is for dynamic shared libraries. It contains some additional tables to support multiple modules. By convention, the file name extension for this format is .dylib, except for the main shared library of a framework, which does not usually have a file name extension.
* The MH\_PRELOAD file type is an executable format used for special-purpose programs that are not loaded by the OS X kernel, such as programs burned into programmable ROM chips. Do not confuse this file type with the MH\_PREBOUND flag, which is a flag that the static linker sets in the header structure to mark a prebound image.
* The MH\_CORE file type is used to store core files, which are traditionally created when a program crashes. Core files store the entire address space of a process at the time it crashed. You can later run gdb on the core file to figure out why the crash occurred.
* The MH\_DYLINKER file type is the type of a dynamic linker shared library. This is the type of the dyld file.
* The MH\_DSYM file type designates files that store symbol information for a corresponding binary file.

ncmds

An integer indicating the number of load commands following the header structure.

sizeofcmds

An integer indicating the number of bytes occupied by the load commands following the header structure.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

11

**OS X ABI Mach-O File Format Reference**

Data Types



flags

An integer containing a set of bit flags that indicate the state of certain optional features of the Mach-O file format. These are the masks you can use to manipulate this field:

* MH\_NOUNDEFS—The object file contained no undefined references when it was built.
* MH\_INCRLINK—The object file is the output of an incremental link against a base file and cannot be linked again.
* MH\_DYLDLINK—The file is input for the dynamic linker and cannot be statically linked again.
* MH\_TWOLEVEL—The image is using two-level namespace bindings.
* MH\_BINDATLOAD—The dynamic linker should bind the undefined references when the file is loaded.
* MH\_PREBOUND—The file’s undefined references are prebound.
* MH\_PREBINDABLE—This file is not prebound but can have its prebinding redone. Used only when

MH\_PREBEOUND is not set.

* MH\_NOFIXPREBINDING—The dynamic linker doesn’t notify the prebinding agent about this executable.
* MH\_ALLMODSBOUND—Indicates that this binary binds to all two-level namespace modules of its dependent libraries. Used only when MH\_PREBINDABLE and MH\_TWOLEVEL are set.
* MH\_CANONICAL—This file has been canonicalized by unprebinding—clearing prebinding information from the file. See the redo\_prebinding man page for details.
* MH\_SPLIT\_SEGS—The file has its read-only and read-write segments split.
* MH\_FORCE\_FLAT—The executable is forcing all images to use flat namespace bindings.
* MH\_SUBSECTIONS\_VIA\_SYMBOLS—The sections of the object file can be divided into individual blocks. These blocks are dead-stripped if they are not used by other code. See Linking for details.
* MH\_NOMULTIDEFS—This umbrella guarantees there are no multiple definitions of symbols in its subimages. As a result, the two-level namespace hints can always be used.

**Special Considerations**

For all file types, except MH\_OBJECT, segments must be aligned on page boundaries for the given CPU architecture: 4096 bytes for PowerPC and x86 processors. This allows the kernel to page virtual memory directly from the segment into the address space of the process. The header and load commands must be aligned as part of the data of the first segment stored on disk (which would be the \_\_TEXT segment, in the file types described in filetype).

**Availability**

Available in OS X v10.6 and later.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

12

**OS X ABI Mach-O File Format Reference**

Data Types



**Declared in**

loader.h

**mach\_header\_64**



*Defines the general attributes of a file targeted for a 64-bit architecture. Declared in /usr/include/mach-o/loader.h.*

struct mach\_header\_64

{

uint32\_t magic;

cpu\_type\_t cputype;

cpu\_subtype\_t cpusubtype;

uint32\_t filetype;

uint32\_t ncmds;

uint32\_t sizeofcmds;

uint32\_t flags;

uint32\_t reserved;

};

**Fields**

magic

An integer containing a value identifying this file as a 64-bit Mach-O file. Use the constant MH\_MAGIC\_64 if the file is intended for use on a CPU with the same endianness as the computer on which the compiler is running. The constant MH\_CIGAM\_64 can be used when the byte ordering scheme of the target machine is the reverse of the host CPU.

cputype

An integer indicating the architecture you intend to use the file on. The only appropriate value for this structure is:

* CPU\_TYPE\_x86\_64 to target 64-bit Intel-based Macintosh computers.
* CPU\_TYPE\_POWERPC64 to target 64-bit PowerPC–based Macintosh computers.

cpusubtype

An integer specifying the exact model of the CPU. To run on all PowerPC processors supported by the OS X kernel, this should be set to CPU\_SUBTYPE\_POWERPC\_ALL.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

13

**OS X ABI Mach-O File Format Reference**

Data Types



filetype

An integer indicating the usage and alignment of the file. Valid values for this field include:

* The MH\_OBJECT file type is the format used for intermediate object files. It is a very compact format containing all its sections in one segment. The compiler and assembler usually create one MH\_OBJECT file for each source code file. By convention, the file name extension for this format is .o.
* The MH\_EXECUTE file type is the format used by standard executable programs.
* The MH\_BUNDLE file type is the type typically used by code that you load at runtime (typically called bundles or plug-ins). By convention, the file name extension for this format is .bundle.
* The MH\_DYLIB file type is for dynamic shared libraries. It contains some additional tables to support multiple modules. By convention, the file name extension for this format is .dylib, except for the main shared library of a framework, which does not usually have a file name extension.
* The MH\_PRELOAD file type is an executable format used for special-purpose programs that are not loaded by the OS X kernel, such as programs burned into programmable ROM chips. Do not confuse this file type with the MH\_PREBOUND flag, which is a flag that the static linker sets in the header structure to mark a prebound image.
* The MH\_CORE file type is used to store core files, which are traditionally created when a program crashes. Core files store the entire address space of a process at the time it crashed. You can later run gdb on the core file to figure out why the crash occurred.
* The MH\_DYLINKER file type is the type of a dynamic linker shared library. This is the type of the dyld file.
* The MH\_DSYM file type designates files that store symbol information for a corresponding binary file.

ncmds

An integer indicating the number of load commands following the header structure.

sizeofcmds

An integer indicating the number of bytes occupied by the load commands following the header structure.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

14

**OS X ABI Mach-O File Format Reference**

Data Types



flags

An integer containing a set of bit flags that indicate the state of certain optional features of the Mach-O file format. These are the masks you can use to manipulate this field:

* MH\_NOUNDEFS—The object file contained no undefined references when it was built.
* MH\_INCRLINK—The object file is the output of an incremental link against a base file and cannot be linked again.
* MH\_DYLDLINK—The file is input for the dynamic linker and cannot be statically linked again.
* MH\_TWOLEVEL—The image is using two-level namespace bindings.
* MH\_BINDATLOAD—The dynamic linker should bind the undefined references when the file is loaded.
* MH\_PREBOUND—The file’s undefined references are prebound.
* MH\_PREBINDABLE—This file is not prebound but can have its prebinding redone. Used only when

MH\_PREBEOUND is not set.

* MH\_NOFIXPREBINDING—The dynamic linker doesn’t notify the prebinding agent about this executable.
* MH\_ALLMODSBOUND—Indicates that this binary binds to all two-level namespace modules of its dependent libraries. Used only when MH\_PREBINDABLE and MH\_TWOLEVEL are set.
* MH\_CANONICAL—This file has been canonicalized by unprebinding—clearing prebinding information from the file. See the redo\_prebinding man page for details.
* MH\_SPLIT\_SEGS—The file has its read-only and read-write segments split.
* MH\_FORCE\_FLAT—The executable is forcing all images to use flat namespace bindings.
* MH\_SUBSECTIONS\_VIA\_SYMBOLS—The sections of the object file can be divided into individual blocks. These blocks are dead-stripped if they are not used by other code. See “Linking” for details.
* MH\_NOMULTIDEFS—This umbrella guarantees there are no multiple definitions of symbols in its subimages. As a result, the two-level namespace hints can always be used.

reserved

Reserved for future use.

**Special Considerations**

See comment in [mach\_header](#page9) (page 9)

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

15

**OS X ABI Mach-O File Format Reference**

Data Types



Load Command Data Structures

The load command structures are located directly after the header of the object file, and they specify both the logical structure of the file and the layout of the file in virtual memory. Each load command begins with fields that specify the command type and the size of the command data.

**load\_command**



*Contains fields that are common to all load commands.*

struct load\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

};

**Fields**

cmd

An integer indicating the type of load command. Table 4 lists the valid load command types.

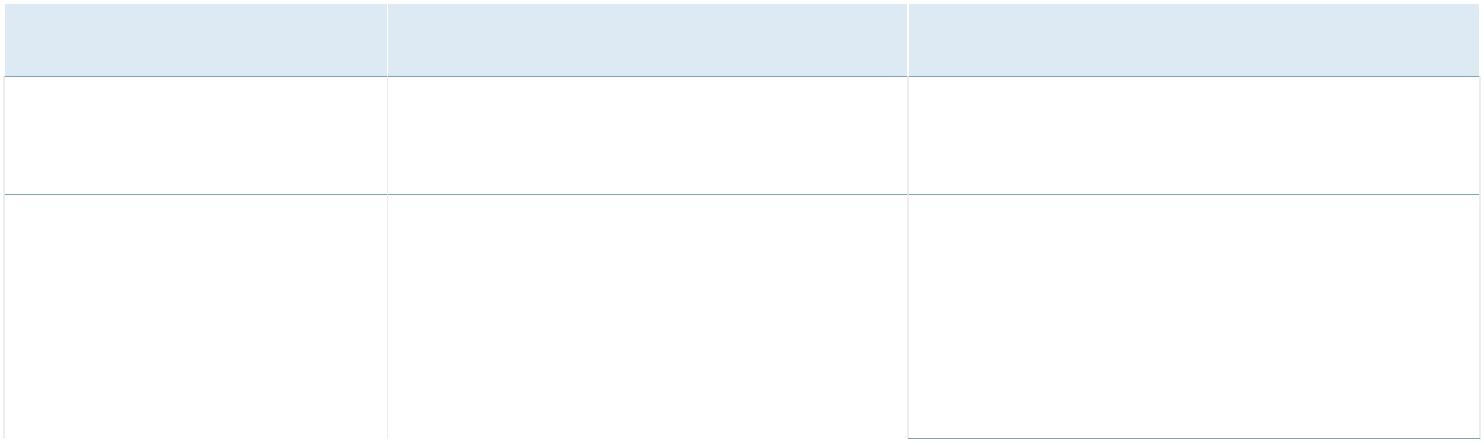
cmdsize

An integer specifying the total size in bytes of the load command data structure. Each load command structure contains a different set of data, depending on the load command type, so each might have a different size. In 32-bit architectures, the size must always be a multiple of 4; in 64-bit architectures, the size must always be a multiple of 8. If the load command data does not divide evenly by 4 or 8 (depending on whether the target architecture is 32-bit or 64-bit, respectively), add bytes containing zeros to the end until it does.

**Discussion**

Table 4 lists the valid load command types, with links to the full data structures for each type.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 4** | Mach-O load commands | |  |
| **Commands** |  | **Data structures** | **Purpose** |
| LC\_UUID |  | [uuid\_command](#page19) (page 19) | Specifies the 128-bit UUID for an |
|  |  |  | image or its corresponding dSYM file. |
| LC\_SEGMENT | | [segment\_command](#page19) (page 19) | Defines a segment of this file to be |
|  |  |  | mapped into the address space of the |
|  |  |  | process that loads this file. It also |
|  |  |  | includes all the sections contained by |
|  |  |  | the segment. |
|  |  |  |  |

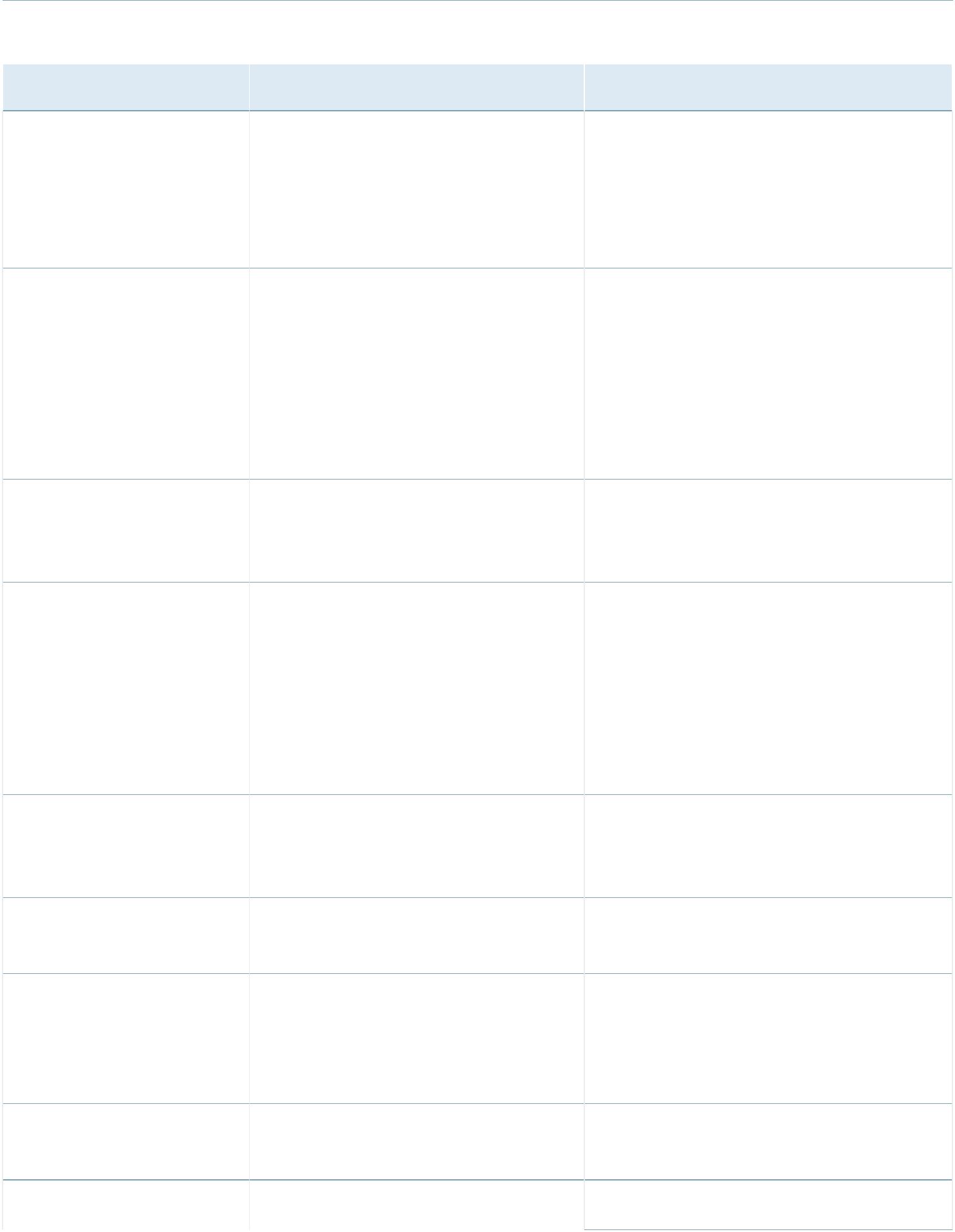


2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

16

**OS X ABI Mach-O File Format Reference**

Data Types



|  |  |  |
| --- | --- | --- |
| **Commands** | **Data structures** | **Purpose** |
| LC\_SEGMENT\_64 | [segment\_command\_64](#page21) (page 21) | Defines a 64-bit segment of this file |
|  |  | to be mapped into the address space |
|  |  | of the process that loads this file. It |
|  |  | also includes all the sections |
|  |  | contained by the segment. |
| LC\_SYMTAB | [symtab\_command](#page44) (page 44) | Specifies the symbol table for this file. |
|  |  | This information is used by both static |
|  |  | and dynamic linkers when linking the |
|  |  | file, and also by debuggers to map |
|  |  | symbols to the original source code |
|  |  | files from which the symbols were |
|  |  | generated. |
| LC\_DYSYMTAB | [dysymtab\_command](#page51) (page 51) | Specifies additional symbol table |
|  |  | information used by the dynamic |
|  |  | linker. |
| LC\_THREAD | [thread\_command](#page38) (page 38) | For an executable file, the |
| LC\_UNIXTHREAD |  | LC\_UNIXTHREAD command defines |
|  |  | the initial thread state of the main |
|  |  | thread of the process. LC\_THREAD is |
|  |  | similar to LC\_UNIXTHREAD but does |
|  |  | not cause the kernel to allocate a |
|  |  | stack. |
| LC\_LOAD\_DYLIB | [dylib\_command](#page35) (page 35) | Defines the name of a dynamic |
|  |  | shared library that this file links |
|  |  | against. |
| LC\_ID\_DYLIB | [dylib\_command](#page35) (page 35) | Specifies the install name of a |
|  |  | dynamic shared library. |
| LC\_PREBOUND\_DYLIB | [prebound\_dylib\_command](#page37) (page | For a shared library that this |
|  | 37) | executable is linked prebound |
|  |  | against, specifies the modules in the |
|  |  | shared library that are used. |
| LC\_LOAD\_DYLINKER | [dylinker\_command](#page36) (page 36) | Specifies the dynamic linker that the |
|  |  | kernel executes to load this file. |
| LC\_ID\_DYLINKER | [dylinker\_command](#page36) (page 36) | Identifies this file as a dynamic linker. |
|  |  |  |

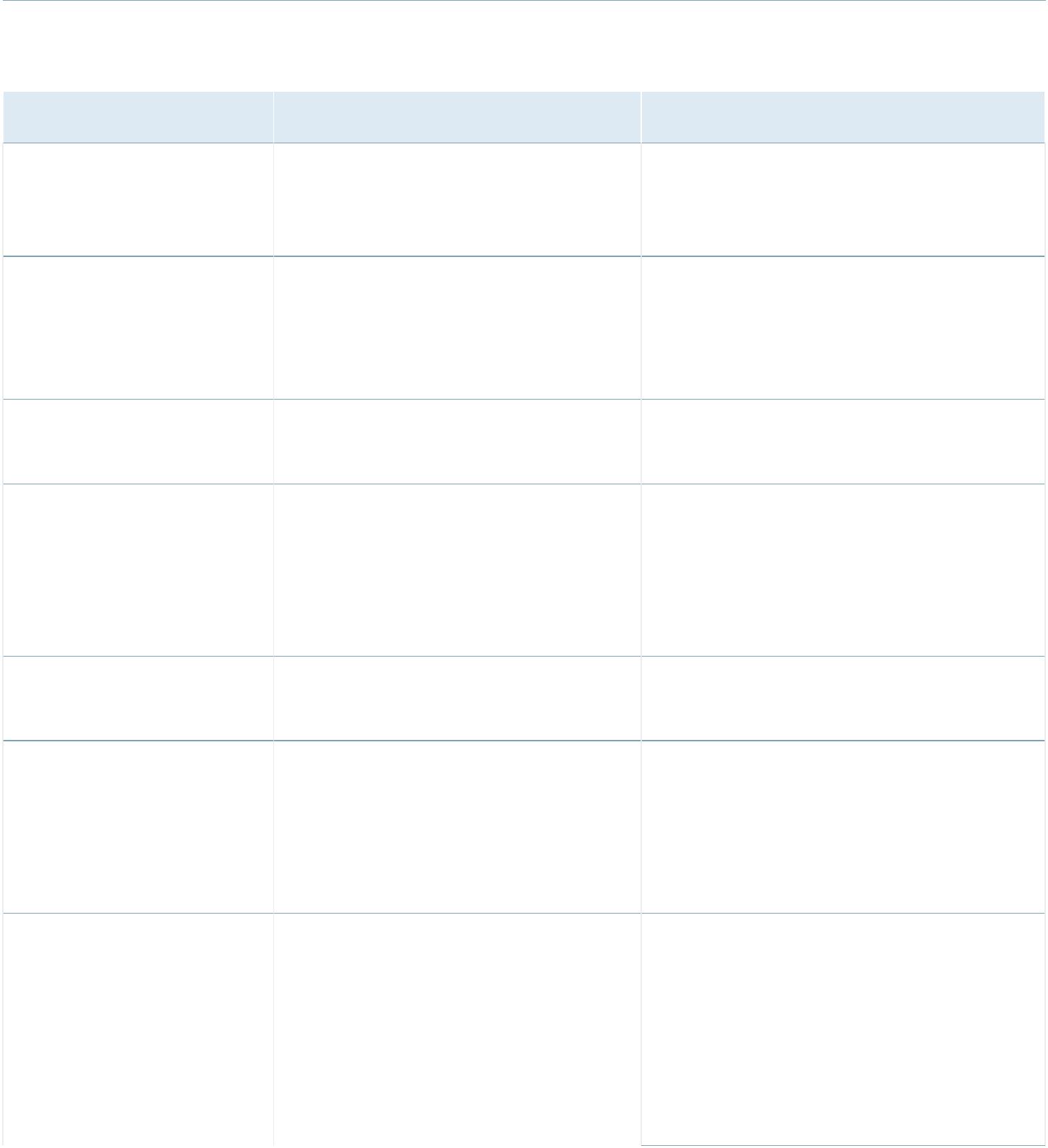


2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

17

**OS X ABI Mach-O File Format Reference**

Data Types



|  |  |  |
| --- | --- | --- |
| **Commands** | **Data structures** | **Purpose** |
| LC\_ROUTINES | [routines\_command](#page38) (page 38) | Contains the address of the shared |
|  |  | library initialization routine (specified |
|  |  | by the linker’s -init option). |
| LC\_ROUTINES\_64 | [routines\_command\_64](#page40) (page 40) | Contains the address of the shared |
|  |  | library 64-bit initialization routine |
|  |  | (specified by the linker’s -init |
|  |  | option). |
| LC\_TWOLEVEL\_HINTS | [twolevel\_hints\_command](#page31) (page | Contains the two-level namespace |
|  | 31) | lookup hint table. |
| LC\_SUB\_FRAMEWORK | [sub\_framework\_command](#page41) (page | Identifies this file as the |
|  | 41) | implementation of a subframework |
|  |  | of an umbrella framework. The name |
|  |  | of the umbrella framework is stored |
|  |  | in the string parameter. |
| LC\_SUB\_UMBRELLA | [sub\_umbrella\_command](#page42) (page | Specifies a file that is a subumbrella |
|  | 42) | of this umbrella framework. |
| LC\_SUB\_LIBRARY | [sub\_library\_command](#page42) (page 42) | Defines the attributes of the |
|  |  | LC\_SUB\_LIBRARY load command. |
|  |  | Identifies a sublibrary of this |
|  |  | framework and marks this framework |
|  |  | as an umbrella framework. |
| LC\_SUB\_CLIENT | [sub\_client\_command](#page43) (page 43) | A subframework can explicitly allow |
|  |  | another framework or bundle to link |
|  |  | against it by including an |
|  |  | LC\_SUB\_CLIENT load command |
|  |  | containing the name of the |
|  |  | framework or a client name for a |
|  |  | bundle. |
|  |  |  |

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

18

**OS X ABI Mach-O File Format Reference**

Data Types



**uuid\_command**



*Specifies the 128-bit universally unique identifier (UUID) for an image or for its corresponding dSYM file.*

struct uuid\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

uint8\_t uuid[16];

};

**Fields**

cmd

Set to LC\_UUID for this structure.

cmdsize

Set to sizeof(uuid\_command).

uuid

128-bit unique identifier.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**segment\_command**



*Specifies the range of bytes in a 32-bit Mach-O file into the address space of a program. Declared in*

[*segment\_command\_64*](#page21) *(page 21).*

*that make up a segment. Those bytes are mapped by the loader /usr/include/mach-o/loader.h. See also*

struct segment\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

char segname[16];

uint32\_t vmaddr;

uint32\_t vmsize;

uint32\_t fileoff;

uint32\_t filesize;

vm\_prot\_t maxprot;

vm\_prot\_t initprot;

uint32\_t nsects;



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

19

**OS X ABI Mach-O File Format Reference**

Data Types



uint32\_t flags;

};

**Fields**

cmd

Common to all load command structures. Set to LC\_SEGMENT for this structure.

cmdsize

Common to all load command structures. For this structure, set this field to sizeof(segment\_command) plus the size of all the section data structures that follow (sizeof(segment\_command + (sizeof(section) \* segment->nsect))).

segname

A C string specifying the name of the segment. The value of this field can be any sequence of ASCII characters, although segment names defined by Apple begin with two underscores and consist of capital letters (as in \_\_TEXT and \_\_DATA). This field is fixed at 16 bytes in length.

vmaddr

Indicates the starting virtual memory address of this segment.

vmsize

Indicates the number of bytes of virtual memory occupied by this segment. See also the description of filesize, below.

fileoff

Indicates the offset in this file of the data to be mapped at vmaddr.

filesize

Indicates the number of bytes occupied by this segment on disk. For segments that require more memory at runtime than they do at build time, vmsize can be larger than filesize. For example, the \_\_PAGEZERO segment generated by the linker for MH\_EXECUTABLE files has a vmsize of 0x1000 but a filesize of

1. Because \_\_PAGEZERO contains no data, there is no need for it to occupy any space until runtime. Also, the static linker often allocates uninitialized data at the end of the \_\_DATA segment; in this case, the

vmsize is larger than the filesize. The loader guarantees that any memory of this sort is initialized with zeros.

maxprot

Specifies the maximum permitted virtual memory protections of this segment.

initprot

Specifies the initial virtual memory protections of this segment.

nsects

Indicates the number of section data structures following this load command.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

20

[*section\_64*](#page27)

**OS X ABI Mach-O File Format Reference**

Data Types

**

flags

Defines a set of flags that affect the loading of this segment:

* SG\_HIGHVM—The file contents for this segment are for the high part of the virtual memory space; the low part is zero filled (for stacks in core files).
* SG\_NORELOC—This segment has nothing that was relocated in it and nothing relocated to it. It may be safely replaced without relocation.

**Special Considerations**

Segments with sections of type S\_GB\_ZEROFILL are placed after all other segments. See [section](#page23) (page 23) for additional information.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**segment\_command\_64**

**

*Specifies the range of bytes in a 64-bit Mach-O file that make up a segment. Those bytes are mapped by the loader*

*into the address space of a program. If the 64-bit segment has sections, they are defined by* *(page*

1. *structures. Declared in /usr/include/mach-o/loader.h.*

struct segment\_command\_64

{

uint32\_t cmd; uint32\_t cmdsize; char segname[16]; uint64\_t vmaddr; uint64\_t vmsize; uint64\_t fileoff; uint64\_t filesize; vm\_prot\_t maxprot; vm\_prot\_t initprot; uint32\_t nsects; uint32\_t flags; };

**Fields**

cmd

See description in [segment\_command](#page19) (page 19). Set to LC\_SEGMENT\_64 for this structure.

**

2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

21

**OS X ABI Mach-O File Format Reference**

Data Types



cmdsize

Common to all load command structures. For this structure, set this field to sizeof(segment\_command\_64) plus the size of all the section data structures that follow (sizeof(segment\_command\_64 + (sizeof(section\_64) \* segment->nsect))).

segname

A C string specifying the name of the segment. The value of this field can be any sequence of ASCII characters, although segment names defined by Apple begin with two underscores and consist of capital letters (as in \_\_TEXT and \_\_DATA). This field is fixed at 16 bytes in length.

vmaddr

Indicates the starting virtual memory address of this segment.

vmsize

Indicates the number of bytes of virtual memory occupied by this segment. See also the description of filesize, below.

fileoff

Indicates the offset in this file of the data to be mapped at vmaddr.

filesize

Indicates the number of bytes occupied by this segment on disk. For segments that require more memory at runtime than they do at build time, vmsize can be larger than filesize. For example, the \_\_PAGEZERO segment generated by the linker for MH\_EXECUTABLE files has a vmsize of 0x1000 but a filesize of

1. Because \_\_PAGEZERO contains no data, there is no need for it to occupy any space until runtime. Also, the static linker often allocates uninitialized data at the end of the \_\_DATA segment; in this case, the

vmsize is larger than the filesize. The loader guarantees that any memory of this sort is initialized with zeros.

maxprot

Specifies the maximum permitted virtual memory protections of this segment.

initprot

Specifies the initial virtual memory protections of this segment.

nsects

Indicates the number of section data structures following this load command.

flags

Defines a set of flags that affect the loading of this segment:

* SG\_HIGHVM—The file contents for this segment are for the high part of the virtual memory space; the low part is zero filled (for stacks in core files).
* SG\_NORELOC—This segment has nothing that was relocated in it and nothing relocated to it. It may be safely replaced without relocation.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

22

[*section\_64*](#page27)

**OS X ABI Mach-O File Format Reference**

Data Types

**

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**section**

**

*Defines the elements used by a 32-bit section. Directly following a segment\_command of section data structures, with the exact count determined by the nsects field of the 19) structure. Declared in /usr/include/mach-o/loader.h. See also*

*data structure is an array* [*segment\_command*](#page19) *(page*

*(page 27).*

struct section

{

char sectname[16]; char segname[16]; uint32\_t addr; uint32\_t size; uint32\_t offset; uint32\_t align; uint32\_t reloff; uint32\_t nreloc; uint32\_t flags; uint32\_t reserved1; uint32\_t reserved2; };

**Fields**

sectname

A string specifying the name of this section. The value of this field can be any sequence of ASCII characters, although section names defined by Apple begin with two underscores and consist of lowercase letters (as in \_\_text and \_\_data). This field is fixed at 16 bytes in length.

segname

A string specifying the name of the segment that should eventually contain this section. For compactness, intermediate object files—files of type MH\_OBJECT—contain only one segment, in which all sections are placed. The static linker places each section in the named segment when building the final product (any file that is not of type MH\_OBJECT).

addr

An integer specifying the virtual memory address of this section.

size

An integer specifying the size in bytes of the virtual memory occupied by this section.

**

2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

23

**OS X ABI Mach-O File Format Reference**

Data Types



offset

An integer specifying the offset to this section in the file.

align

An integer specifying the section’s byte alignment. Specify this as a power of two; for example, a section with 8-byte alignment would have an align value of 3 (2 to the 3rd power equals 8).

reloff

An integer specifying the file offset of the first relocation entry for this section.

nreloc

An integer specifying the number of relocation entries located at reloff for this section.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

24

**OS X ABI Mach-O File Format Reference**

Data Types



flags

An integer divided into two parts. The least significant 8 bits contain the section type, while the most significant 24 bits contain a set of flags that specify other attributes of the section. These types and flags are primarily used by the static linker and file analysis tools, such as otool, to determine how to modify or display the section. These are the possible types:

●

●

●

●

●

●

●

●

●

S\_REGULAR—This section has no particular type. The standard tools create a \_\_TEXT,\_\_text section of this type.

S\_ZEROFILL—Zero-fill-on-demand section—when this section is first read from or written to, each page within is automatically filled with bytes containing zero.

S\_CSTRING\_LITERALS—This section contains only constant C strings. The standard tools create a \_\_TEXT,\_\_cstring section of this type.

S\_4BYTE\_LITERALS—This section contains only constant values that are 4 bytes long. The standard tools create a \_\_TEXT,\_\_literal4 section of this type.

S\_8BYTE\_LITERALS—This section contains only constant values that are 8 bytes long. The standard tools create a \_\_TEXT,\_\_literal8 section of this type.

S\_LITERAL\_POINTERS—This section contains only pointers to constant values.

S\_NON\_LAZY\_SYMBOL\_POINTERS—This section contains only non-lazy pointers to symbols. The standard tools create a section of the \_\_DATA,\_\_nl\_symbol\_ptrs section of this type.

S\_LAZY\_SYMBOL\_POINTERS—This section contains only lazy pointers to symbols. The standard tools create a \_\_DATA,\_\_la\_symbol\_ptrs section of this type.

S\_SYMBOL\_STUBS——This section contains symbol stubs. The standard tools create \_\_TEXT,\_\_symbol\_stub and \_\_TEXT,\_\_picsymbol\_stub sections of this type. See “Position-Independent Code” in *Mach-O Programming Topics* for more information.

●

●

●

S\_MOD\_INIT\_FUNC\_POINTERS—This section contains pointers to module initialization functions.

The standard tools create \_\_DATA,\_\_mod\_init\_func sections of this type.

S\_MOD\_TERM\_FUNC\_POINTERS—This section contains pointers to module termination functions.

The standard tools create \_\_DATA,\_\_mod\_term\_func sections of this type.

S\_COALESCED—This section contains symbols that are coalesced by the static linker and possibly the dynamic linker. More than one file may contain coalesced definitions of the same symbol without causing multiple-defined-symbol errors.

* S\_GB\_ZEROFILL—This is a zero-filled on-demand section. It can be larger than 4 GB. This section must be placed in a segment containing only zero-filled sections. If you place a zero-filled section in a segment with non–zero-filled sections, you may cause those sections to be unreachable with a 31-bit offset. That outcome stems from the fact that the size of a zero-filled section can be larger than 4 GB (in a 32-bit address space). As a result of this, the static linker would be unable to build the output file. See [segment\_command](#page19) (page 19) for more information.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

25

**OS X ABI Mach-O File Format Reference**

Data Types



The following are the possible attributes of a section:

* S\_ATTR\_PURE\_INSTRUCTIONS—This section contains only executable machine instructions. The standard tools set this flag for the sections \_\_TEXT,\_\_text, \_\_TEXT,\_\_symbol\_stub, and

\_\_TEXT,\_\_picsymbol\_stub.

* S\_ATTR\_SOME\_INSTRUCTIONS—This section contains executable machine instructions.
* S\_ATTR\_NO\_TOC—This section contains coalesced symbols that must not be placed in the table of contents (SYMDEF member) of a static archive library.
* S\_ATTR\_EXT\_RELOC—This section contains references that must be relocated. These references refer to data that exists in other files (undefined symbols). To support external relocation, the maximum virtual memory protections of the segment that contains this section must allow both reading and writing.
* S\_ATTR\_LOC\_RELOC—This section contains references that must be relocated. These references refer to data within this file.
* S\_ATTR\_STRIP\_STATIC\_SYMS—The static symbols in this section can be stripped if the MH\_DYLDLINK flag of the image’s [mach\_header](#page9) (page 9) header structure is set.
* S\_ATTR\_NO\_DEAD\_STRIP—This section must not be dead-stripped. See “Linking” for details.
* S\_ATTR\_LIVE\_SUPPORT—This section must not be dead-stripped if they reference code that is live, but the reference is undetectable.

reserved1

An integer reserved for use with certain section types. For symbol pointer sections and symbol stubs sections that refer to indirect symbol table entries, this is the index into the indirect table for this section’s entries. The number of entries is based on the section size divided by the size of the symbol pointer or stub. Otherwise, this field is set to 0.

reserved2

For sections of type S\_SYMBOL\_STUBS, an integer specifying the size (in bytes) of the symbol stub entries contained in the section. Otherwise, this field is reserved for future use and should be set to 0.

**Discussion**

Each section in a Mach-O file has both a type and a set of attribute flags. In intermediate object files, the type and attributes determine how the static linker copies the sections into the final product. Object file analysis tools (such as otool) use the type and attributes to determine how to read and display the sections. Some section types and attributes are used by the dynamic linker.

These are important static-linking variants of the symbol type and attributes:

* **Regular sections**. In a regular section, only one definition of an external symbol may exist in intermediateobject files. The static linker returns an error if it finds any duplicate external symbol definitions.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

26

**OS X ABI Mach-O File Format Reference**

Data Types



* **Coalesced sections**. In the final product, the static linker retains only one instance of each symbol definedin coalesced sections. To support complex language features (such as C++ vtables and RTTI) the compiler may create a definition of a particular symbol in every intermediate object file. The static linker and the dynamic linker would then reduce the duplicate definitions to the single definition used by the program.
* **Coalesced sections with weak definitions** Weak symbol definitions may appear only in coalesced sections.When the static linker finds duplicate definitions for a symbol, it discards any coalesced symbol definition that has the weak definition attribute set (see [nlist](#page45) (page 45)). If there are no non-weak definitions, the first weak definition is used instead. This feature is designed to support C++ templates; it allows explicit template instantiations to override implicit ones. The C++ compiler places explicit definitions in a regular section, and it places implicit definitions in a coalesced section, marked as weak definitions. Intermediate object files (and thus static archive libraries) built with weak definitions can be used only with the static linker in OS X v10.2 and later. Final products (applications and shared libraries) should not contain weak definitions if they are expected to be used on earlier versions of OS X.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**section\_64**



*Defines the elements used by a 64-bit section. Directly following a* [*segment\_command\_64*](#page21) *(page 21) data structure is an array of section\_64 data structures, with the exact count determined by the nsects field of the segment\_command\_64 structure. Declared in /usr/include/mach-o/loader.h.*

struct section\_64

{

char sectname[16];

char segname[16];

uint64\_t addr;

uint64\_t size;

uint32\_t offset;

uint32\_t align;

uint32\_t reloff;

uint32\_t nreloc;

uint32\_t flags;

uint32\_t reserved1;

uint32\_t reserved2;

};



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

27

**OS X ABI Mach-O File Format Reference**

Data Types



**Fields**

sectname

A string specifying the name of this section. The value of this field can be any sequence of ASCII characters, although section names defined by Apple begin with two underscores and consist of lowercase letters (as in \_\_text and \_\_data). This field is fixed at 16 bytes in length.

segname

A string specifying the name of the segment that should eventually contain this section. For compactness, intermediate object files—files of type MH\_OBJECT—contain only one segment, in which all sections are placed. The static linker places each section in the named segment when building the final product (any file that is not of type MH\_OBJECT).

addr

An integer specifying the virtual memory address of this section.

size

An integer specifying the size in bytes of the virtual memory occupied by this section.

offset

An integer specifying the offset to this section in the file.

align

An integer specifying the section’s byte alignment. Specify this as a power of two; for example, a section with 8-byte alignment would have an align value of 3 (2 to the 3rd power equals 8).

reloff

An integer specifying the file offset of the first relocation entry for this section.

nreloc

An integer specifying the number of relocation entries located at reloff for this section.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

28

**OS X ABI Mach-O File Format Reference**

Data Types



flags

An integer divided into two parts. The least significant 8 bits contain the section type, while the most significant 24 bits contain a set of flags that specify other attributes of the section. These types and flags are primarily used by the static linker and file analysis tools, such as otool, to determine how to modify or display the section. These are the possible types:

●

●

●

●

●

●

●

●

●

S\_REGULAR—This section has no particular type. The standard tools create a \_\_TEXT,\_\_text section of this type.

S\_ZEROFILL—Zero-fill-on-demand section—when this section is first read from or written to, each page within is automatically filled with bytes containing zero.

S\_CSTRING\_LITERALS—This section contains only constant C strings. The standard tools create a \_\_TEXT,\_\_cstring section of this type.

S\_4BYTE\_LITERALS—This section contains only constant values that are 4 bytes long. The standard tools create a \_\_TEXT,\_\_literal4 section of this type.

S\_8BYTE\_LITERALS—This section contains only constant values that are 8 bytes long. The standard tools create a \_\_TEXT,\_\_literal8 section of this type.

S\_LITERAL\_POINTERS—This section contains only pointers to constant values.

S\_NON\_LAZY\_SYMBOL\_POINTERS—This section contains only non-lazy pointers to symbols. The standard tools create a section of the \_\_DATA,\_\_nl\_symbol\_ptrs section of this type.

S\_LAZY\_SYMBOL\_POINTERS—This section contains only lazy pointers to symbols. The standard tools create a \_\_DATA,\_\_la\_symbol\_ptrs section of this type.

S\_SYMBOL\_STUBS——This section contains symbol stubs. The standard tools create \_\_TEXT,\_\_symbol\_stub and \_\_TEXT,\_\_picsymbol\_stub sections of this type. See “Position-Independent Code” in *Mach-O Programming Topics* for more information.

●

●

●

S\_MOD\_INIT\_FUNC\_POINTERS—This section contains pointers to module initialization functions.

The standard tools create \_\_DATA,\_\_mod\_init\_func sections of this type.

S\_MOD\_TERM\_FUNC\_POINTERS—This section contains pointers to module termination functions.

The standard tools create \_\_DATA,\_\_mod\_term\_func sections of this type.

S\_COALESCED—This section contains symbols that are coalesced by the static linker and possibly the dynamic linker. More than one file may contain coalesced definitions of the same symbol without causing multiple-defined-symbol errors.

* S\_GB\_ZEROFILL—This is a zero-filled on-demand section. It can be larger than 4 GB. This section must be placed in a segment containing only zero-filled sections. If you place a zero-filled section in a segment with non–zero-filled sections, you may cause those sections to be unreachable with a 31-bit offset. That outcome stems from the fact that the size of a zero-filled section can be larger than 4 GB (in a 32-bit address space). As a result of this, the static linker would be unable to build the output file. See [segment\_command\_64](#page21) (page 21) for more information.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

29

**OS X ABI Mach-O File Format Reference**

Data Types



The following are the possible attributes of a section:

* S\_ATTR\_PURE\_INSTRUCTIONS—This section contains only executable machine instructions. The standard tools set this flag for the sections \_\_TEXT,\_\_text, \_\_TEXT,\_\_symbol\_stub, and

\_\_TEXT,\_\_picsymbol\_stub.

* S\_ATTR\_SOME\_INSTRUCTIONS—This section contains executable machine instructions.
* S\_ATTR\_NO\_TOC—This section contains coalesced symbols that must not be placed in the table of contents (SYMDEF member) of a static archive library.
* S\_ATTR\_EXT\_RELOC—This section contains references that must be relocated. These references refer to data that exists in other files (undefined symbols). To support external relocation, the maximum virtual memory protections of the segment that contains this section must allow both reading and writing.
* S\_ATTR\_LOC\_RELOC—This section contains references that must be relocated. These references refer to data within this file.
* S\_ATTR\_STRIP\_STATIC\_SYMS—The static symbols in this section can be stripped if the MH\_DYLDLINK flag of the image’s [mach\_header](#page9) (page 9) header structure is set.
* S\_ATTR\_NO\_DEAD\_STRIP—This section must not be dead-stripped. See “Linking” for details.
* S\_ATTR\_LIVE\_SUPPORT—This section must not be dead-stripped if they reference code that is live, but the reference is undetectable.

reserved1

An integer reserved for use with certain section types. For symbol pointer sections and symbol stubs sections that refer to indirect symbol table entries, this is the index into the indirect table for this section’s entries. The number of entries is based on the section size divided by the size of the symbol pointer or stub. Otherwise, this field is set to 0.

reserved2

For sections of type S\_SYMBOL\_STUBS, an integer specifying the size (in bytes) of the symbol stub entries contained in the section. Otherwise, this field is reserved for future use and should be set to 0.

**Discussion**

Each section in a Mach-O file has both a type and a set of attribute flags. In intermediate object files, the type and attributes determine how the static linker copies the sections into the final product. Object file analysis tools (such as otool) use the type and attributes to determine how to read and display the sections. Some section types and attributes are used by the dynamic linker.

These are important static-linking variants of the symbol type and attributes:

* **Regular sections**. In a regular section, only one definition of an external symbol may exist in intermediateobject files. The static linker returns an error if it finds any duplicate external symbol definitions.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

30

**OS X ABI Mach-O File Format Reference**

Data Types



* **Coalesced sections**. In the final product, the static linker retains only one instance of each symbol definedin coalesced sections. To support complex language features (such as C++ vtables and RTTI) the compiler may create a definition of a particular symbol in every intermediate object file. The static linker and the dynamic linker would then reduce the duplicate definitions to the single definition used by the program.
* **Coalesced sections with weak definitions** Weak symbol definitions may appear only in coalesced sections.When the static linker finds duplicate definitions for a symbol, it discards any coalesced symbol definition that has the weak definition attribute set (see [nlist](#page45) (page 45)). If there are no non-weak definitions, the first weak definition is used instead. This feature is designed to support C++ templates; it allows explicit template instantiations to override implicit ones. The C++ compiler places explicit definitions in a regular section, and it places implicit definitions in a coalesced section, marked as weak definitions. Intermediate object files (and thus static archive libraries) built with weak definitions can be used only with the static linker in OS X v10.2 and later. Final products (applications and shared libraries) should not contain weak definitions if they are expected to be used on earlier versions of OS X.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**twolevel\_hints\_command**



*Defines the attributes of a LC\_TWOLEVEL\_HINTS load command. Declared in /usr/include/mach-o/loader.h.*

struct twolevel\_hints\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

uint32\_t offset;

uint32\_t nhints;

};

**Fields**

cmd

Common to all load command structures. Set to LC\_TWOLEVEL\_HINTS for this structure.

cmdsize

Common to all load command structures. For this structure, set to sizeof(twolevel\_hints\_command).



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

31

**OS X ABI Mach-O File Format Reference**

Data Types



offset

An integer specifying the byte offset from the start of this file to an array of [twolevel\_hint](#page32) (page 32) data structures, known as the two-level namespace hint table.

nhints

The number of twolevel\_hint data structures located at offset.

**Discussion**

The static linker adds the LC\_TWOLEVEL\_HINTS load command and the two-level namespace hint table to the output file when building a two-level namespace image.

**Special Considerations**

By default, ld does not include the LC\_TWOLEVEL\_HINTS command or the two-level namespace hint table in an MH\_BUNDLE file because the presence of this load command causes the version of the dynamic linker shipped with OS X v10.0 to crash. If you know the code will run only on OS X v10.1 and later, you should explicitly enable the two-level namespace hint table. See -twolevel\_namespace\_hints in the ld man page for more information.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**twolevel\_hint**



*Specifies an entry in the two-level namespace hint table. Declared in /usr/include/mach-o/loader.h.*

struct twolevel\_hint

{

uint32\_t isub\_image:8,

itoc:24;

};

**Fields**

isub\_image

The subimage in which the symbol is defined. It is an index into the list of images that make up the umbrella image. If this field is 0, the symbol is in the umbrella image itself. If the image is not an umbrella framework or library, this field is 0.

itoc

The symbol index into the table of contents of the image specified by the isub\_image field.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

32

**OS X ABI Mach-O File Format Reference**

Data Types



**Discussion**

The two-level namespace hint table provides the dynamic linker with suggested positions to start searching for symbols in the libraries the current image is linked against.

Every undefined symbol (that is, every symbol of type N\_UNDF or N\_PBUD) in a two-level namespace image has a corresponding entry in the two-level hint table, at the same index.

The static linker adds the LC\_TWOLEVEL\_HINTS load command and the two-level namespace hint table to the output file when building a two-level namespace image.

By default, the linker does not include the LC\_TWOLEVEL\_HINTS command or the two-level namespace hint table in an MH\_BUNDLE file, because the presence of this load command causes the version of the dynamic linker shipped with OS X v10.0 to crash. If you know the code will run only on OS X v10.1 and later, you should explicitly enable the two-level namespace hints. See the linker documentation for more information.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**lc\_str**



*Defines a variable-length string. Declared in /usr/include/mach-o/loader.h.*

union lc\_str

{

uint32\_t offset;

#ifndef \_\_LP64\_\_

char \*ptr;

#endif

};

**Fields**

offset

A long integer. A byte offset from the start of the load command that contains this string to the start of the string data.

ptr

A pointer to an array of bytes. At runtime, this pointer contains the virtual memory address of the string data. The ptr field is not used in Mach-O files.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

33

**OS X ABI Mach-O File Format Reference**

Data Types



**Discussion**

Load commands store variable-length data such as library names using the lc\_str data structure. Unless otherwise specified, the data consists of a C string.

The data pointed to is stored just after the load command, and the size is added to the size of the load command. The string should be null terminated; any extra bytes to round up the size should be null. You can also determine the size of the string by subtracting the size of the load command data structure from the cmdsize field of the load command data structure.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**dylib**



*Defines the data used by the dynamic linker to match a shared library against the files that have linked to it. Used exclusively in the* [*dylib\_command*](#page35) *(page 35) data structure. Declared in /usr/include/mach-o/loader.h.*

struct dylib

{

union lc\_str name;

uint\_32 timestamp;

uint\_32 current\_version;

uint\_32 compatibility\_version;

};

**Fields**

name

A data structure of type [lc\_str](#page33) (page 33). Specifies the name of the shared library.

timestamp

The date and time when the shared library was built.

current\_version

The current version of the shared library.

compatibility\_version

The compatibility version of the shared library.

**Availability**

Available in OS X v10.6 and later.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

34

**OS X ABI Mach-O File Format Reference**

Data Types



**Declared in**

loader.h

**dylib\_command**



*Defines the attributes of the LC\_LOAD\_DYLIB and LC\_ID\_DYLIB load commands. Declared in*

*/usr/include/mach-o/loader.h.*

struct dylib\_command

{

uint\_32 cmd;

uint\_32 cmdsize;

struct dylib dylib;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to either LC\_LOAD\_DYLIB, LC\_LOAD\_WEAK\_DYLIB, or LC\_ID\_DYLIB.

cmdsize

Common to all load command structures. For this structure, set to sizeof(dylib\_command) plus the size of the data pointed to by the name field of the dylib field.

dylib

A data structure of type [dylib](#page34) (page 34). Specifies the attributes of the shared library.

**Discussion**

For each shared library that a file links against, the static linker creates an LC\_LOAD\_DYLIB command and sets

its dylib field to the value of the dylib field of the LC\_ID\_DYLD load command of the target library. All the

LC\_LOAD\_DYLIB commands together form a list that is ordered according to location in the file, earliest

LC\_LOAD\_DYLIB command first. For two-level namespace files, undefined symbol entries in the symbol table

refer to their parent shared libraries by index into this list. The index is called a *library ordinal* , and it is stored

in the n\_desc field of the [nlist](#page45) (page 45) data structure.

At runtime, the dynamic linker uses the name in the dyld field of the LC\_LOAD\_DYLIB command to locate

the shared library. If it finds the library, the dynamic linker compares the version information of the

LC\_LOAD\_DYLIB load command against the library’s version. For the dynamic linker to successfully link the

shared library, the compatibility version of the shared library must be less than or equal to the compatibility

version in the LC\_LOAD\_DYLIB command.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

35

**OS X ABI Mach-O File Format Reference**

Data Types



The dynamic linker uses the timestamp to determine whether it can use the prebinding information. The current version is returned by the function NSVersionOfRunTimeLibrary to allow you to determine the version of the library your program is using.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**dylinker\_command**



*Defines the attributes of the LC\_LOAD\_DYLINKER and LC\_ID\_DYLINKER load commands. Declared in /usr/include/mach-o/loader.h.*

struct dylinker\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

union lc\_str name;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to either LC\_ID\_DYLINKER or

LC\_LOAD\_DYLINKER.

cmdsize

Common to all load command structures. For this structure, set to sizeof(dylinker\_command), plus the size of the data pointed to by the name field.

name

A data structure of type [lc\_str](#page33) (page 33). Specifies the name of the dynamic linker.

**Discussion**

Every executable file that is dynamically linked contains a LC\_LOAD\_DYLINKER command that specifies the name of the dynamic linker that the kernel must load in order to execute the file. The dynamic linker itself specifies its name using the LC\_ID\_DYLINKER load command.

**Availability**

Available in OS X v10.6 and later.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

36

**OS X ABI Mach-O File Format Reference**

Data Types



**Declared in**

loader.h

**prebound\_dylib\_command**



*Defines the attributes of the LC\_PREBOUND\_DYLIB load command. For every library that a prebound executable file links to, the static linker adds one LC\_PREBOUND\_DYLIB command. Declared in*

*/usr/include/mach-o/loader.h.*

struct prebound\_dylib\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

union lc\_str name;

uint32\_t nmodules;

union lc\_str linked\_modules;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_PREBOUND\_DYLIB.

cmdsize

Common to all load command structures. For this structure, set to sizeof(prebound\_dylib\_command) plus the size of the data pointed to by the name and linked\_modules fields.

name

A data structure of type [lc\_str](#page33) (page 33). Specifies the name of the prebound shared library.

nmodules

An integer. Specifies the number of modules the prebound shared library contains. The size of the linked\_modules string is (nmodules / 8) + (nmodules % 8).

linked\_modules

A data structure of type [lc\_str](#page33) (page 33). Usually, this data structure defines the offset of a C string; in this usage, it is a variable-length bitset, containing one bit for each module. Each bit represents whether the corresponding module is linked to a module in the current file, 1 for yes, 0 for no. The bit for the first module is the low bit of the first byte.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

37

**OS X ABI Mach-O File Format Reference**

Data Types



**thread\_command**



*Defines the attributes of the LC\_THREAD and LC\_UNIXTHREAD load commands. The data of this command is specific to each architecture and appears in thread\_status.h, located in the architecture’s directory in /usr/include/mach. Declared in /usr/include/mach-o/loader.h.*

struct thread\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

/\* uint32\_t flavor;\*/

/\* uint32\_t count; \*/

/\* struct cpu\_thread\_state state;\*/

};

**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_THREAD or LC\_UNIXTHREAD.

cmdsize

Common to all load command structures. For this structure, set to sizeof(thread\_command) plus the size of the flavor and count fields plus the size of the CPU-specific thread state data structure.

flavor

Integer specifying the particular flavor of the thread state data structure. See the thread\_status.h file for your target architecture.

count

Size of the thread state data, in number of 32-bit integers. The thread state data structure must be fully padded to 32-bit alignment.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**routines\_command**



*Defines the attributes of the LC\_ROUTINES load command, used in 32-bit architectures. Describes the location of the shared library initialization function, which is a function that the dynamic linker calls before allowing any of the routines in the library to be called. Declared in /usr/include/mach-o/loader.h. See also*

[*routines\_command\_64*](#page40) *(page 40).*



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

38

**OS X ABI Mach-O File Format Reference**

Data Types



struct routines\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

uint32\_t init\_address;

uint32\_t init\_module;

uint32\_t reserved1;

uint32\_t reserved2;

uint32\_t reserved3;

uint32\_t reserved4;

uint32\_t reserved5;

uint32\_t reserved6;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_ROUTINES.

cmdsize

Common to all load command structures. For this structure, set to sizeof(routines\_command).

init\_address

An integer specifying the virtual memory address of the initialization function.

init\_module

An integer specifying the index into the module table of the module containing the initialization function.

reserved1

Reserved for future use. Set this field to 0.

reserved2

Reserved for future use. Set this field to 0.

reserved3

Reserved for future use. Set this field to 0.

reserved4

Reserved for future use. Set this field to 0.

reserved5

Reserved for future use. Set this field to 0.

reserved6

Reserved for future use. Set this field to 0.

**Discussion**

The static linker adds an LC\_ROUTINES command when you specify a shared library initialization function using the -init option (see the ld man page for more information).



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

39

**OS X ABI Mach-O File Format Reference**

Data Types



**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**routines\_command\_64**



*Defines the attributes of the LC\_ROUTINES\_64 load command, used in 64-bit architectures. Describes the location of the shared library initialization function, which is a function that the dynamic linker calls before allowing any of the routines in the library to be called. Declared in /usr/include/mach-o/loader.h.*

struct routines\_command\_64

{

uint32\_t cmd;

uint32\_t cmdsize;

uint64\_t init\_address;

uint64\_t init\_module;

uint64\_t reserved1;

uint64\_t reserved2;

uint64\_t reserved3;

uint64\_t reserved4;

uint64\_t reserved5;

uint64\_t reserved6;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_ROUTINES\_64.

cmdsize

Common to all load command structures. For this structure, set to sizeof(routines\_command\_64).

init\_address

An integer specifying the virtual memory address of the initialization function.

init\_module

An integer specifying the index into the module table of the module containing the initialization function.

reserved1

Reserved for future use. Set this field to 0.

reserved2

Reserved for future use. Set this field to 0.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

40

**OS X ABI Mach-O File Format Reference**

Data Types



reserved3

Reserved for future use. Set this field to 0.

reserved4

Reserved for future use. Set this field to 0.

reserved5

Reserved for future use. Set this field to 0.

reserved6

Reserved for future use. Set this field to 0.

**Discussion**

The static linker adds an LC\_ROUTINES\_64 command when you specify a shared library initialization function using the -init option (see the ld man page for more information).

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**sub\_framework\_command**



*Defines the attributes of the LC\_SUB\_FRAMEWORK load command. Identifies the umbrella framework of which this file is a subframework. Declared in /usr/include/mach-o/loader.h.*

struct sub\_framework\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

union lc\_str umbrella;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_SUB\_FRAMEWORK.

cmdsize

Common to all load command structures. For this structure, set to sizeof(sub\_framework\_command) plus the size of the data pointed to by the umbrella field.

umbrella

A data structure of type [lc\_str](#page33) (page 33). Specifies the name of the umbrella framework of which this file is a member.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

41

**OS X ABI Mach-O File Format Reference**

Data Types



**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**sub\_umbrella\_command**



*Definestheattributesofthe LC\_SUB\_UMBRELLAloadcommand. Identifiesthenamedframeworkasasubumbrella of this framework. Unlike a subframework, any client may link to a subumbrella. Declared in /usr/include/mach-o/loader.h.*

struct sub\_umbrella\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

union lc\_str sub\_umbrella;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_SUB\_UMBRELLA.

cmdsize

Common to all load command structures. For this structure, set to sizeof(sub\_umbrella\_command) plus the size of the data pointed to by the sub\_umbrella field.

sub\_umbrella

A data structure of type [lc\_str](#page33) (page 33). Specifies the name of the umbrella framework of which this file is a member.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**sub\_library\_command**



*Defines the attributes of the LC\_SUB\_LIBRARY load command. Identifies a sublibrary of this framework and marks this framework as an umbrella framework. Unlike a subframework, any client may link to a sublibrary. Declared in /usr/include/mach-o/loader.h.*



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

42

**OS X ABI Mach-O File Format Reference**

Data Types



struct sub\_library\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

union lc\_str sub\_library;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_SUB\_LIBRARY.

cmdsize

Common to all load command structures. For this structure, set to sizeof(sub\_library\_command) plus the size of the data pointed to by the sub\_library field.

sub\_library

A data structure of type [lc\_str](#page33) (page 33). Specifies the name of the sublibrary of which this file is a member.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**sub\_client\_command**



*Defines the attributes of the LC\_SUB\_CLIENT load command. Specifies the name of a file that is allowed to link to this subframework. This file would otherwise be required to link to the umbrella framework of which this file is a component. Declared in /usr/include/mach-o/loader.h.*

struct sub\_client\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

union lc\_str client;

};

**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_SUB\_CLIENT.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

43

**OS X ABI Mach-O File Format Reference**

Data Types



cmdsize

Common to all load command structures. For this structure, set to sizeof(sub\_client\_command) plus the size of the data pointed to by the client field.

client

A data structure of type [lc\_str](#page33) (page 33). Specifies the name of a client authorized to link to this library.

**Special Considerations**

The ld tool generates a sub\_client\_command load command in the built product if you pass the option -allowable\_client <name>, where <name> is the install name of a framework or the client name of a bundle. See the ld man page, specifically about the options -allowable\_client and -client\_name, for more information.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

Symbol Table and Related Data Structures

Two load commands, LC\_SYMTAB and LC\_DYSYMTAB, describe the size and location of the symbol tables, along with additional metadata. The other data structures listed in this section represent the symbol tables themselves.

**symtab\_command**



*Defines the attributes of the LC\_SYMTAB load command. Describes the size and location of the symbol table data structures. Declared in /usr/include/mach-o/loader.h.*

struct symtab\_command

{

uint\_32 cmd;

uint\_32 cmdsize;

uint\_32 symoff;

uint\_32 nsyms;

uint\_32 stroff;

uint\_32 strsize;

};



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

44

**OS X ABI Mach-O File Format Reference**

Data Types



**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_SYMTAB.

cmdsize

Common to all load command structures. For this structure, set to sizeof(symtab\_command).

symoff

An integer containing the byte offset from the start of the file to the location of the symbol table entries.

The symbol table is an array of [nlist](#page45) (page 45) data structures.

nsyms

An integer indicating the number of entries in the symbol table.

stroff

An integer containing the byte offset from the start of the image to the location of the string table.

strsize

An integer indicating the size (in bytes) of the string table.

**Discussion**

LC\_SYMTAB should exist in both statically linked and dynamically linked file types.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**nlist**



*Describes an entry in the symbol table for 32-bit architectures. Declared in /usr/include/mach-o/nlist.h.*

*See also* [*nlist\_64*](#page48) *(page 48).*

struct nlist

{

union {

#ifndef \_\_LP64\_\_

char \*n\_name;

#endif

int32\_t n\_strx;

} n\_un;

uint8\_t n\_type;

uint8\_t n\_sect;

int16\_t n\_desc;



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

45

**OS X ABI Mach-O File Format Reference**

Data Types



uint32\_t n\_value;

};

**Fields**

n\_un

A union that holds an index into the string table, n\_strx. To specify an empty string (""), set this value to 0. The n\_name field is not used in Mach-O files.

n\_type

A byte value consisting of data accessed using four bit masks:

* N\_STAB (0xe0)—If any of these 3 bits are set, the symbol is a symbolic debugging table (stab) entry. In that case, the entire n\_type field is interpreted as a stab value. See

/usr/include/mach-o/stab.h for valid stab values.

* N\_PEXT (0x10)—If this bit is on, this symbol is marked as having limited global scope. When the file is fed to the static linker, it clears the N\_EXT bit for each symbol with the N\_PEXT bit set. (The ld option -keep\_private\_externs turns off this behavior.) With OS X GCC, you can use the \_\_private\_extern\_\_ function attribute to set this bit.
* N\_TYPE (0x0e)—These bits define the type of the symbol.
* N\_EXT (0x01)—If this bit is on, this symbol is an external symbol, a symbol that is either defined outside this file or that is defined in this file but can be referenced by other files.

Values for the N\_TYPE field include:

* N\_UNDF (0x0)—The symbol is undefined. Undefined symbols are symbols referenced in this module but defined in a different module. The n\_sect field is set to NO\_SECT.
* N\_ABS (0x2)—The symbol is absolute. The linker does not change the value of an absolute symbol. The n\_sect field is set to NO\_SECT.
* N\_SECT (0xe)—The symbol is defined in the section number given in n\_sect.
* N\_PBUD (0xc)—The symbol is undefined and the image is using a prebound value for the symbol. The n\_sect field is set to NO\_SECT.
* N\_INDR ( 0xa)—The symbol is defined to be the same as another symbol. The n\_value field is an index into the string table specifying the name of the other symbol. When that symbol is linked, both this and the other symbol have the same defined type and value.

n\_sect

An integer specifying the number of the section that this symbol can be found in, or NO\_SECT if the symbol is not to be found in any section of this image. The sections are contiguously numbered across segments, starting from 1, according to the order they appear in the LC\_SEGMENT load commands.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

46

**OS X ABI Mach-O File Format Reference**

Data Types



n\_desc

A 16-bit value providing additional information about the nature of this symbol for non-stab symbols.

The reference flags can be accessed using the REFERENCE\_TYPE mask (0xF) and are defined as follows:

* REFERENCE\_FLAG\_UNDEFINED\_NON\_LAZY (0x0)—This symbol is a reference to an external non-lazy (data) symbol.
* REFERENCE\_FLAG\_UNDEFINED\_LAZY (0x1)—This symbol is a reference to an external lazy symbol—that is, to a function call.
* REFERENCE\_FLAG\_DEFINED (0x2)—This symbol is defined in this module.
* REFERENCE\_FLAG\_PRIVATE\_DEFINED (0x3)—This symbol is defined in this module and is visible only to modules within this shared library.
* REFERENCE\_FLAG\_PRIVATE\_UNDEFINED\_NON\_LAZY (0x4)—This symbol is defined in another module in this file, is a non-lazy (data) symbol, and is visible only to modules within this shared library.
* REFERENCE\_FLAG\_PRIVATE\_UNDEFINED\_LAZY (0x5)—This symbol is defined in another module in this file, is a lazy (function) symbol, and is visible only to modules within this shared library.

Additionally, the following bits might also be set:

* REFERENCED\_DYNAMICALLY (0x10)—Must be set for any defined symbol that is referenced by dynamic-loader APIs (such as dlsym and NSLookupSymbolInImage) and not ordinary undefined symbol references. The strip tool uses this bit to avoid removing symbols that must exist: If the symbol has this bit set, strip does not strip it.
* N\_DESC\_DISCARDED (0x20)—Sometimes used by the dynamic linker at runtime in a fully linked image. Do not set this bit in a fully linked image.
* N\_NO\_DEAD\_STRIP(0x20)—When set in a relocatable object file (file type MH\_OBJECT) on a defined symbol, indicates to the static linker to never dead-strip the symbol. (Note that the same bit (0x20) is used for two nonoverlapping purposes.)
* N\_WEAK\_REF (0x40)—Indicates that this undefined symbol is a weak reference. If the dynamic linker cannot find a definition for this symbol, it sets the address of this symbol to 0. The static linker sets this symbol given the appropriate weak-linking flags.
* N\_WEAK\_DEF (0x80)—Indicates that this symbol is a weak definition. If the static linker or the dynamic linker finds another (non-weak) definition for this symbol, the weak definition is ignored. Only symbols in a coalesced [section](#page23) (page 23) can be marked as a weak definition.

If this file is a two-level namespace image (that is, if the MH\_TWOLEVEL flag of the mach\_header structure

is set), the high 8 bits of n\_desc specify the number of the library in which this undefined symbol is

defined. Use the macro GET\_LIBRARY\_ORDINAL to obtain this value and the macro

SET\_LIBRARY\_ORDINAL to set it. Zero specifies the current image. 1 through 253 specify the library

number according to the order of LC\_LOAD\_DYLIB commands in the file. The value 254 is used for



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

47

**OS X ABI Mach-O File Format Reference**

Data Types



undefined symbols that are to be dynamically looked up (supported only in OS X v10.3 and later). For plug–ins that load symbols from the executable program they are linked against, 255 specifies the executable image. For flat namespace images, the high 8 bits must be 0.

n\_value

An integer that contains the value of the symbol. The format of this value is different for each type of symbol table entry (as specified by the n\_type field). For the N\_SECT symbol type, n\_value is the address of the symbol. See the description of the n\_type field for information on other possible values.

**Discussion**

Common symbols must be of type N\_UNDF and must have the N\_EXT bit set. The n\_value for a common symbol is the size (in bytes) of the data of the symbol. In C, a common symbol is a variable that is declared but not initialized in this file. Common symbols can appear only in MH\_OBJECT Mach-O files.

**Availability**

Available in OS X v10.8 and later.

**Declared in**

nlist.h

**nlist\_64**



*Describes an entry in the symbol table for 64-bit architectures. Declared in /usr/include/mach-o/nlist.h.*

struct nlist\_64

{

union {

uint32\_t n\_strx;

} n\_un;

uint8\_t n\_type;

uint8\_t n\_sect;

uint16\_t n\_desc;

uint64\_t n\_value;

};

**Fields**

n\_un

A union that holds an index into the string table, n\_strx. To specify an empty string (""), set this value to 0.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

48

**OS X ABI Mach-O File Format Reference**

Data Types



n\_type

A byte value consisting of data accessed using four bit masks:

* N\_STAB (0xe0)—If any of these 3 bits are set, the symbol is a symbolic debugging table (stab) entry. In that case, the entire n\_type field is interpreted as a stab value. See

/usr/include/mach-o/stab.h for valid stab values.

* N\_PEXT (0x10)—If this bit is on, this symbol is marked as having limited global scope. When the file is fed to the static linker, it clears the N\_EXT bit for each symbol with the N\_PEXT bit set. (The ld option -keep\_private\_externs turns off this behavior.) With OS X GCC, you can use the \_\_private\_extern\_\_ function attribute to set this bit.
* N\_TYPE (0x0e)—These bits define the type of the symbol.
* N\_EXT (0x01)—If this bit is on, this symbol is an external symbol, a symbol that is either defined outside this file or that is defined in this file but can be referenced by other files.

Values for the N\_TYPE field include:

* N\_UNDF (0x0)—The symbol is undefined. Undefined symbols are symbols referenced in this module but defined in a different module. Set the n\_sect field to NO\_SECT.
* N\_ABS (0x2)—The symbol is absolute. The linker does not update the value of an absolute symbol. Set the n\_sect field to NO\_SECT.
* N\_SECT (0xe)—The symbol is defined in the section number given in n\_sect.
* N\_PBUD (0xc)—The symbol is undefined and the image is using a prebound value for the symbol. Set the n\_sect field to NO\_SECT.
* N\_INDR ( 0xa)—The symbol is defined to be the same as another symbol. The n\_value field is an index into the string table specifying the name of the other symbol. When that symbol is linked, both this and the other symbol point to the same defined type and value.

n\_sect

An integer specifying the number of the section that this symbol can be found in, or NO\_SECT if the symbol is not to be found in any section of this image. The sections are contiguously numbered across segments, starting from 1, according to the order they appear in the LC\_SEGMENT load commands.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

49

**OS X ABI Mach-O File Format Reference**

Data Types



n\_desc

A 16-bit value providing additional information about the nature of this symbol. The reference flags can be accessed using the REFERENCE\_TYPE mask (0xF) and are defined as follows:

* REFERENCE\_FLAG\_UNDEFINED\_NON\_LAZY (0x0)—This symbol is a reference to an external non-lazy (data) symbol.
* REFERENCE\_FLAG\_UNDEFINED\_LAZY (0x1)—This symbol is a reference to an external lazy symbol—that is, to a function call.
* REFERENCE\_FLAG\_DEFINED (0x2)—This symbol is defined in this module.
* REFERENCE\_FLAG\_PRIVATE\_DEFINED (0x3)—This symbol is defined in this module and is visible only to modules within this shared library.
* REFERENCE\_FLAG\_PRIVATE\_UNDEFINED\_NON\_LAZY (0x4)—This symbol is defined in another module in this file, is a non-lazy (data) symbol, and is visible only to modules within this shared library.
* REFERENCE\_FLAG\_PRIVATE\_UNDEFINED\_LAZY (0x5)—This symbol is defined in another module in this file, is a lazy (function) symbol, and is visible only to modules within this shared library.

Additionally, the following bits might also be set:

* REFERENCED\_DYNAMICALLY (0x10)—Must be set for any defined symbol that is referenced by dynamic-loader APIs (such as dlsym and NSLookupSymbolInImage) and not ordinary undefined symbol references. The strip tool uses this bit to avoid removing symbols that must exist: If the symbol has this bit set, strip does not strip it.
* N\_DESC\_DISCARDED (0x20)—Used by the dynamic linker at runtime. Do not set this bit.
* N\_WEAK\_REF (0x40)—Indicates that this symbol is a weak reference. If the dynamic linker cannot find a definition for this symbol, it sets the address of this symbol to 0. The static linker sets this symbol given the appropriate weak-linking flags.
* N\_WEAK\_DEF (0x80)—Indicates that this symbol is a weak definition. If the static linker or the dynamic linker finds another (non-weak) definition for this symbol, the weak definition is ignored. Only symbols in a coalesced [section](#page23) (page 23) can be marked as a weak definition.

If this file is a two-level namespace image (that is, if the MH\_TWOLEVEL flag of the mach\_header structure is set), the high 8 bits of n\_desc specify the number of the library in which this symbol is defined. Use the macro GET\_LIBRARY\_ORDINAL to obtain this value and the macro SET\_LIBRARY\_ORDINAL to set it. Zero specifies the current image. 1 through 254 specify the library number according to the order of LC\_LOAD\_DYLIB commands in the file. For plug–ins that load symbols from the executable program they are linked against, 255 specifies the executable image. For flat namespace images, the high 8 bits must be 0.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

50

**OS X ABI Mach-O File Format Reference**

Data Types



n\_value

An integer that contains the value of the symbol. The format of this value is different for each type of symbol table entry (as specified by the n\_type field). For the N\_SECT symbol type, n\_value is the address of the symbol. See the description of the n\_type field for information on other possible values.

**Discussion**

See discussion in [nlist](#page45) (page 45).

**Availability**

Available in OS X v10.8 and later.

**Declared in**

nlist.h

**dysymtab\_command**



*The data structure for the LC\_DYSYMTAB load command. It describes the sizes and locations of the parts of the symbol table used for dynamic linking. Declared in /usr/include/mach-o/loader.h.*

struct dysymtab\_command

{

uint32\_t cmd;

uint32\_t cmdsize;

uint32\_t ilocalsym;

uint32\_t nlocalsym;

uint32\_t iextdefsym;

uint32\_t nextdefsym;

uint32\_t iundefsym;

uint32\_t nundefsym;

uint32\_t tocoff;

uint32\_t ntoc;

uint32\_t modtaboff;

uint32\_t nmodtab;

uint32\_t extrefsymoff;

uint32\_t nextrefsyms;

uint32\_t indirectsymoff;

uint32\_t nindirectsyms;

uint32\_t extreloff;

uint32\_t nextrel;

uint32\_t locreloff;

uint32\_t nlocrel;

};



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

51

**OS X ABI Mach-O File Format Reference**

Data Types



**Fields**

cmd

Common to all load command structures. For this structure, set to LC\_DYSYMTAB.

cmdsize

Common to all load command structures. For this structure, set to sizeof(dysymtab\_command).

ilocalsym

An integer indicating the index of the first symbol in the group of local symbols.

nlocalsym

An integer indicating the total number of symbols in the group of local symbols.

iextdefsym

An integer indicating the index of the first symbol in the group of defined external symbols.

nextdefsym

An integer indicating the total number of symbols in the group of defined external symbols.

iundefsym

An integer indicating the index of the first symbol in the group of undefined external symbols.

nundefsym

An integer indicating the total number of symbols in the group of undefined external symbols.

tocoff

An integer indicating the byte offset from the start of the file to the table of contents data.

ntoc

An integer indicating the number of entries in the table of contents.

modtaboff

An integer indicating the byte offset from the start of the file to the module table data.

nmodtab

An integer indicating the number of entries in the module table.

extrefsymoff

An integer indicating the byte offset from the start of the file to the external reference table data.

nextrefsyms

An integer indicating the number of entries in the external reference table.

indirectsymoff

An integer indicating the byte offset from the start of the file to the indirect symbol table data.

nindirectsyms

An integer indicating the number of entries in the indirect symbol table.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

52

**OS X ABI Mach-O File Format Reference**

Data Types



extreloff

An integer indicating the byte offset from the start of the file to the external relocation table data.

nextrel

An integer indicating the number of entries in the external relocation table.

locreloff

An integer indicating the byte offset from the start of the file to the local relocation table data.

nlocrel

An integer indicating the number of entries in the local relocation table.

**Discussion**

The LC\_DYSYMTAB load command contains a set of indexes into the symbol table and a set of file offsets that define the location of several other tables. Fields for tables not used in the file should be set to 0. These tables are described in “Position-Independent Code” in *Mach-O Programming Topics* .

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**dylib\_table\_of\_contents**



*Describes an entry in the table of contents of a dynamic shared library. Declared in /usr/include/mach-o/loader.h.*

struct dylib\_table\_of\_contents

{

uint32\_t symbol\_index;

uint32\_t module\_index;

};

**Fields**

symbol\_index

An index into the symbol table indicating the defined external symbol to which this entry refers.

module\_index

An index into the module table indicating the module in which this defined external symbol is defined.

**Availability**

Available in OS X v10.6 and later.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

53

[*dylib\_module\_64*](#page55)

**OS X ABI Mach-O File Format Reference**

Data Types

**

**Declared in**

loader.h

**dylib\_module**

**

*Describes a module table entry for a dynamic shared library for 32-bit architectures. Declared in*

*/usr/include/mach-o/loader.h. See also* *(page 55).*

struct dylib\_module

{

uint32\_t module\_name; uint32\_t iextdefsym; uint32\_t nextdefsym; uint32\_t irefsym; uint32\_t nrefsym; uint32\_t ilocalsym; uint32\_t nlocalsym; uint32\_t iextrel; uint32\_t nextrel; uint32\_t iinit\_iterm; uint32\_t ninit\_nterm; uint32\_t objc\_module\_info\_addr; uint32\_t objc\_module\_info\_size; };

**Fields**

module\_name

An index to an entry in the string table indicating the name of the module.

iextdefsym

The index into the symbol table of the first defined external symbol provided by this module.

nextdefsym

The number of defined external symbols provided by this module.

irefsym

The index into the external reference table of the first entry provided by this module.

nrefsym

The number of external reference entries provided by this module.

ilocalsym

The index into the symbol table of the first local symbol provided by this module.

nlocalsym

The number of local symbols provided by this module.

**

2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

54

**OS X ABI Mach-O File Format Reference**

Data Types



iextrel

The index into the external relocation table of the first entry provided by this module.

nextrel

The number of entries in the external relocation table that are provided by this module.

iinit\_iterm

Contains both the index into the module initialization section (the low 16 bits) and the index into the module termination section (the high 16 bits) to the pointers for this module.

ninit\_nterm

Contains both the number of pointers in the module initialization (the low 16 bits) and the number of pointers in the module termination section (the high 16 bits) for this module.

objc\_module\_info\_addr

The statically linked address of the start of the data for this module in the \_\_module\_info section in the \_\_OBJC segment.

objc\_module\_info\_size

The number of bytes of data for this module that are used in the \_\_module\_info section in the \_\_OBJC segment.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**dylib\_module\_64**



*Describes a module table entry for a dynamic shared library for 64-bit architectures. Declared in /usr/include/mach-o/loader.h.*

struct dylib\_module\_64

{

uint32\_t module\_name;

uint32\_t iextdefsym;

uint32\_t nextdefsym;

uint32\_t irefsym;

uint32\_t nrefsym;

uint32\_t ilocalsym;

uint32\_t nlocalsym;

uint32\_t iextrel;

uint32\_t nextrel;

uint32\_t iinit\_iterm;

uint32\_t ninit\_nterm;



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

55

**OS X ABI Mach-O File Format Reference**

Data Types



uint32\_t objc\_module\_info\_size;

uint64\_t objc\_module\_info\_addr;

};

**Fields**

module\_name

An index to an entry in the string table indicating the name of the module.

iextdefsym

The index into the symbol table of the first defined external symbol provided by this module.

nextdefsym

The number of defined external symbols provided by this module.

irefsym

The index into the external reference table of the first entry provided by this module.

nrefsym

The number of external reference entries provided by this module.

ilocalsym

The index into the symbol table of the first local symbol provided by this module.

nlocalsym

The number of local symbols provided by this module.

iextrel

The index into the external relocation table of the first entry provided by this module.

nextrel

The number of entries in the external relocation table that are provided by this module.

iinit\_iterm

Contains both the index into the module initialization section (the low 16 bits) and the index into the module termination section (the high 16 bits) to the pointers for this module.

ninit\_nterm

Contains both the number of pointers in the module initialization (the low 16 bits) and the number of pointers in the module termination section (the high 16 bits) for this module.

objc\_module\_info\_addr

The statically linked address of the start of the data for this module in the \_\_module\_info section in the \_\_OBJC segment.

objc\_module\_info\_size

The number of bytes of data for this module that are used in the \_\_module\_info section in the \_\_OBJC segment.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

56

**OS X ABI Mach-O File Format Reference**

Data Types



**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

**dylib\_reference**



*Defines the attributes of an external reference table entry for the external reference entries provided by a module in a shared library. Declared in /usr/include/mach-o/loader.h.*

struct dylib\_reference

{

uint32\_t isym:24,

flags:8;

};

**Fields**

isym

An index into the symbol table for the symbol being referenced.

flags

A constant for the type of reference being made. Use the same REFERENCE\_FLAG constants as described in the [nlist](#page45) (page 45) structure description.

**Availability**

Available in OS X v10.6 and later.

**Declared in**

loader.h

Relocation Data Structures

**Relocation** is the process of moving symbols to a different address. When the static linker moves a symbol (afunction or an item of data) to a different address, it needs to change all the references to that symbol to use the new address. The **relocation entries** in a Mach-O file contain offsets in the file to addresses that need to be relocated when the contents of the file are relocated. The addresses stored in CPU instructions can be absolute or relative. Each relocation entry specifies the exact format of the address. When creating the intermediate object file, the compiler generates one or more relocation entries for every instruction that contains an address. Because relocation to symbols at fixed addresses, and to relative addresses for position independent references, does not occur at runtime, the static linker typically removes some or all the relocation entries when building the final product.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

57

**OS X ABI Mach-O File Format Reference**

Data Types



**Note:** In the OS X x86-64 environment scattered relocations are not used. Compiler-generated codeuses mostly external relocations, in which the r\_extern bit is set to 1 and the r\_symbolnum field contains the symbol-table index of the target label.



**relocation\_info**



*Describes an item in the file that uses an address that needs to be updated when the address is changed. Declared in /usr/include/mach-o/reloc.h.*

struct relocation\_info

{

int32\_t r\_address;

uint32\_t r\_symbolnum:24,

r\_pcrel:1,

r\_length:2,

r\_extern:1,

r\_type:4;

};

**Fields**

r\_address

In MH\_OBJECT files, this is an offset from the start of the section to the item containing the address requiring relocation. If the high bit of this field is set (which you can check using the R\_SCATTERED bit mask), the relocation\_info structure is actually a [scattered\_relocation\_info](#page62) (page 62) structure.

In images used by the dynamic linker, this is an offset from the virtual memory address of the data of the first [segment\_command](#page19) (page 19) that appears in the file (not necessarily the one with the lowest address). For images with the MH\_SPLIT\_SEGS flag set, this is an offset from the virtual memory address of data of the first read/write [segment\_command](#page19) (page 19).

r\_symbolnum

Indicates either an index into the symbol table (when the r\_extern field is set to 1) or a section number (when the r\_extern field is set to 0). As previously mentioned, sections are ordered from 1 to 255 in the order in which they appear in the LC\_SEGMENT load commands. This field is set to R\_ABS for relocation entries for absolute symbols, which need no relocation.

r\_pcrel

Indicates whether the item containing the address to be relocated is part of a CPU instruction that uses PC-relative addressing.

For addresses contained in PC-relative instructions, the CPU adds the address of the instruction to the address contained in the instruction.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

58

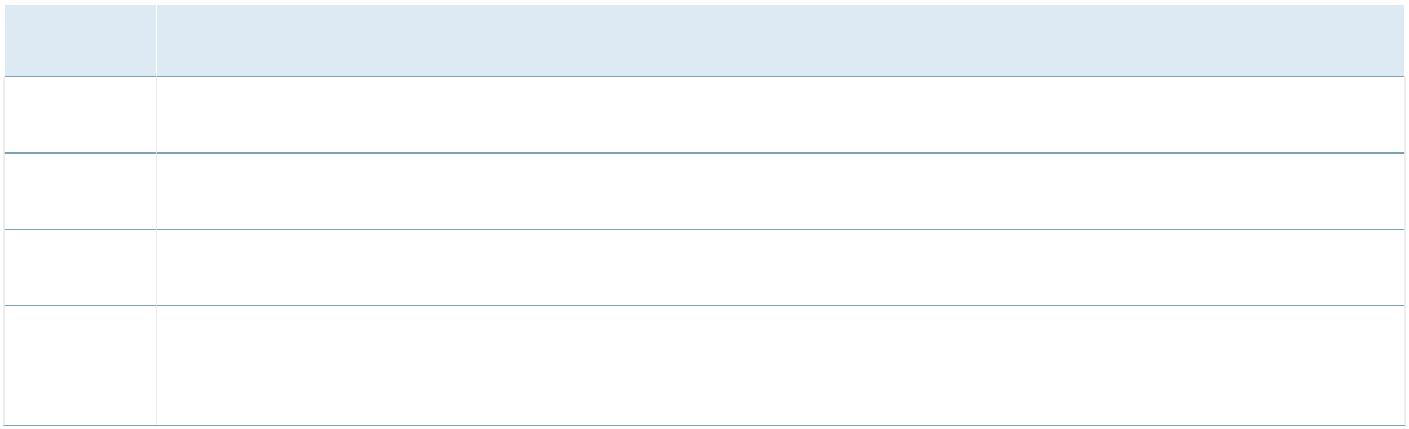
**OS X ABI Mach-O File Format Reference**

Data Types



r\_length

Indicates the length of the item containing the address to be relocated. The following table lists r\_length values and the corresponding address length.



**Value** **Address length**

* 1 byte
* 2 bytes
* 4 bytes
* 4 bytes. See description for the PPC\_RELOC\_BR14 r\_type in

[scattered\_relocation\_info](#page62) (page 62).

r\_extern

Indicates whether the r\_symbolnum field is an index into the symbol table (1) or a section number (0).



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

59

**OS X ABI Mach-O File Format Reference**

Data Types



r\_type

For the x86 environment, the r\_type field may contain any of these values:

* GENERIC\_RELOC\_VANILLA—A generic relocation entry for both addresses contained in data and addresses contained in CPU instructions.
* GENERIC\_RELOC\_PAIR—The second relocation entry of a pair.
* GENERIC\_RELOC\_SECTDIFF—A relocation entry for an item that contains the difference of two section addresses. This is generally used for position-independent code generation. GENERIC\_RELOC\_SECTDIFF contains the address from which to subtract; it must be followed by a GENERIC\_RELOC\_PAIR containing the address to subtract.
* GENERIC\_RELOC\_LOCAL\_SECTDIFF—Similar to GENERIC\_RELOC\_SECTDIFF except that this entry refers specifically to the address in this item. If the address is that of a globally visible coalesced symbol, this relocation entry does not change if the symbol is overridden. This is used to associate stack unwinding information with the object code this relocation entry describes.
* GENERIC\_RELOC\_PB\_LA\_PTR—A relocation entry for a prebound lazy pointer. This is always a scattered relocation entry. The r\_value field contains the non-prebound value of the lazy pointer.

For the x86-64 environment, the r\_type field may contain any of these values:

* X86\_64\_RELOC\_BRANCH—A CALL/JMP instruction with 32-bit displacement.
* X86\_64\_RELOC\_GOT\_LOAD—A MOVQ load of a GOT entry.
* X86\_64\_RELOC\_GOT—Other GOT references.
* X86\_64\_RELOC\_SIGNED—Signed 32-bit displacement.
* X86\_64\_RELOC\_UNSIGNED—Absolute address.
* X86\_64\_RELOC\_SUBTRACTOR—Must be followed by a X86\_64\_RELOC\_UNSIGNED relocation.

For PowerPC environments, the r\_type field is usually PPC\_RELOC\_VANILLA for addresses contained in data. Relocation entries for addresses contained in CPU instructions are described by other r\_type values:

* PPC\_RELOC\_PAIR—The second relocation entry of a pair. A PPC\_RELOC\_PAIR entry must follow each of the other relocation entry types, except for PPC\_RELOC\_VANILLA, PPC\_RELOC\_BR14,

PPC\_RELOC\_BR24, and PPC\_RELOC\_PB\_LA\_PTR.

* PPC\_RELOC\_BR14—The instruction contains a 14-bit branch displacement. If the r\_length is 3, the branch was statically predicted by setting or clearing the Y bit depending on the sign of the displacement or the opcode.
* PPC\_RELOC\_BR24—The instruction contains a 24-bit branch displacement.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

60

**OS X ABI Mach-O File Format Reference**

Data Types



* PPC\_RELOC\_HI16—The instruction contains the high 16 bits of a relocatable expression. The next relocation entry must be a PPC\_RELOC\_PAIR specifying the low 16 bits of the expression in the low 16 bits of the r\_value field.
* PPC\_RELOC\_LO16—The instruction contains the low 16 bits of an address. The next relocation entry must be a PPC\_RELOC\_PAIR specifying the high 16 bits of the expression in the low (*not* the high) 16 bits of the r\_value field.
* PPC\_RELOC\_HA16—Same as the PPC\_RELOC\_HI16 except the low 16 bits and the high 16 bits are added together with the low 16 bits sign-extended first. This means if bit 15 of the low 16 bits is set, the high 16 bits stored in the instruction are adjusted.
* PPC\_RELOC\_LO14—Same as PPC\_RELOC\_LO16 except that the low 2 bits are not stored in the CPU instruction and are always 0. PPC\_RELOC\_LO14 is used in 64-bit load/store instructions.
* PPC\_RELOC\_SECTDIFF—A relocation entry for an item that contains the difference of two section addresses. This is generally used for position-independent code generation. PPC\_RELOC\_SECTDIFF contains the address from which to subtract; it must be followed by a PPC\_RELOC\_PAIR containing the section address to subtract.
* PPC\_RELOC\_LOCAL\_SECTDIFF—Similar to PPC\_RELOC\_SECTDIFF except that this entry refers specifically to the address in this item. If the address is that of a globally visible coalesced symbol, this relocation entry does not change if the symbol is overridden. This is used to associate stack unwinding information with the object code this relocation entry describes
* PPC\_RELOC\_PB\_LA\_PTR—A relocation entry for a prebound lazy pointer. This is always a scattered relocation entry. The r\_value field contains the non-prebound value of the lazy pointer.

●

●

●

●

PPC\_RELOC\_HI16\_SECTDIFF—Section difference form of PPC\_RELOC\_HI16.

PPC\_RELOC\_LO16\_SECTDIFF—Section difference form of PPC\_RELOC\_LO16.

PPC\_RELOC\_HA16\_SECTDIFF—Section difference form of PPC\_RELOC\_HA16.

PPC\_RELOC\_JBSR—A relocation entry for the assembler synthetic opcode jbsr, which is a 24-bit branch-and-link instruction using a branch island. The branch displacement is assembled to the branch island address and the relocation entry indicates the actual target symbol. If the linker is able to make the branch reach the actual target symbol, it does. Otherwise, the branch is relocated to the branch island.

* PPC\_RELOC\_LO14\_SECTDIFF—Section difference form of PPC\_RELOC\_LO14.

**Availability**

Available in OS X v10.8 and later.

**Declared in**

reloc.h



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

61

**OS X ABI Mach-O File Format Reference**

Data Types



**scattered\_relocation\_info**



*Describes an item in the file—using a nonzero constant in its relocatable expression or two addresses in its relocatable expression—that needs to be updated if the addresses that it uses are changed. This information is needed to reconstruct the addresses that make up the relocatable expression’s value in order to change the addresses independently of each other. Declared in /usr/include/mach-o/reloc.h.*

struct scattered\_relocation\_info

{

#ifdef \_\_BIG\_ENDIAN\_\_

uint32\_t r\_scattered:1,

r\_pcrel:1,

r\_length:2,

r\_type:4,

r\_address:24;

int32\_t r\_value;

#endif /\* \_\_BIG\_ENDIAN\_\_ \*/

#ifdef \_\_LITTLE\_ENDIAN\_\_

uint32\_t r\_address:24,

r\_type:4,

r\_length:2,

r\_pcrel:1,

r\_scattered:1;

int32\_t r\_value;

#endif /\* \_\_LITTLE\_ENDIAN\_\_ \*/

};

**Fields**

r\_scattered

If this bit is 0, this structure is actually a [relocation\_info](#page58) (page 58) structure.

r\_address

In MH\_OBJECT files, this is an offset from the start of the section to the item containing the address requiring relocation. If the high bit of this field is clear (which you can check using the R\_SCATTERED bit mask), this structure is actually a [relocation\_info](#page58) (page 58) structure.

In images used by the dynamic linker, this is an offset from the virtual memory address of the data of the first [segment\_command](#page19) (page 19) that appears in the file (not necessarily the one with the lowest address). For images with the MH\_SPLIT\_SEGS flag set, this is an offset from the virtual memory address of data of the first read/write [segment\_command](#page19) (page 19).

Since this field is only 24 bits long, the offset in this field can never be larger than 0x00FFFFFF, thus limiting the size of the relocatable contents of this image to 16 megabytes.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

62

**OS X ABI Mach-O File Format Reference**

Data Types



r\_pcrel

Indicates whether the item containing the address to be relocated is part of a CPU instruction that uses PC-relative addressing.

For addresses contained in PC-relative instructions, the CPU adds the address of the instruction to the address contained in the instruction.

r\_length

Indicates the length of the item containing the address to be relocated. A value of 0 indicates a single byte; a value of 1 indicates a 2-byte address, and a value of 2 indicates a 4-byte address.

r\_type

Indicates the type of relocation to be performed. Possible values for this field are shared between this structure and the relocation\_info data structure; see the description of the r\_type field in the [relocation\_info](#page58) (page 58) data structure for more details.

r\_value

The address of the relocatable expression for the item in the file that needs to be updated if the address is changed. For relocatable expressions with the difference of two section addresses, the address from which to subtract (in mathematical terms, the minuend) is contained in the first relocation entry and the address to subtract (the subtrahend) is contained in the second relocation entry.

**Discussion**

Mach-O relocation data structures support two types of relocatable expressions in machine code and data:

* **Symbol address + constant**. The most typical form of relocation is referencing a symbol’s address withno constant added. In this case, the value of the constant expression is 0.
* **Address of section y – address of section x + constant**. The section difference form of relocation. Thisform of relocation supports position-independent code.

**Availability**

Available in OS X v10.8 and later.

**Declared in**

reloc.h

Universal Binaries and 32-bit/64-bit PowerPC Binaries

The standard development tools accept as parameters two kinds of binaries:

* Object files targeted at one architecture. These include Mach-O files, static libraries, and dynamic libraries.
* Binaries targeted at more than one architecture. These binaries contain compiled code and data for one of these system types:



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

63

**OS X ABI Mach-O File Format Reference**

Data Types



* PowerPC-based (32-bit and 64-bit) Macintosh computers. Binaries that contain code for both 32-bit and 64-bit PowerPC-based Macintosh computers are are known as **PPC/PPC64 binaries**.
* Intel-based and PowerPC-based (32-bit, 64-bit, or both) Macintosh computers. Binaries that contain code for both Intel-based and PowerPC-based Macintosh computers are known as **universal binaries**.

Each object file is stored as a continuous set of bytes at an offset from the beginning of the binary. They use a simple archive format to store the two object files with a special header at the beginning of the file to allow the various runtime tools to quickly find the code appropriate for the current architecture.

A binary that contains code for more than one architecture always begins with a [fat\_header](#page64) (page 64) data structure, followed by two [fat\_arch](#page65) (page 65) data structures and the actual data for the architectures contained in the file. All data in these data structures is stored in big-endian byte order.

**fat\_header**



*Defines the layout of a binary that contains code for more than one architecture. Declared in the header /usr/include/mach-o/fat.h.*

struct fat\_header

{

uint32\_t magic;

uint32\_t nfat\_arch;

};

**Fields**

magic

An integer containing the value 0xCAFEBABE in big-endian byte order format. On a big-endian host CPU, this can be validated using the constant FAT\_MAGIC; on a little-endian host CPU, it can be validated using the constant FAT\_CIGAM.

nfat\_arch

An integer specifying the number of [fat\_arch](#page65) (page 65) data structures that follow. This is the number of architectures contained in this binary.

**Discussion**

The fat\_header data structure is placed at the start of a binary that contains code for multiple architectures. Directly following the fat\_header data structure is a set of [fat\_arch](#page65) (page 65) data structures, one for each architecture included in the binary.

Regardless of the content this data structure describes, all its fields are stored in big-endian byte order.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

64

**OS X ABI Mach-O File Format Reference**

Data Types



**Availability**

Available in OS X v10.8 and later.

**Declared in**

fat.h

**fat\_arch**



*Describes the location within the binary of an object file targeted at a single architecture. Declared in /usr/include/mach-o/fat.h.*

struct fat\_arch

{

cpu\_type\_t cputype;

cpu\_subtype\_t cpusubtype;

uint32\_t offset;

uint32\_t size;

uint32\_t align;

};

**Fields**

cputype

An enumeration value of type cpu\_type\_t. Specifies the CPU family.

cpusubtype

An enumeration value of type cpu\_subtype\_t. Specifies the specific member of the CPU family on which this entry may be used or a constant specifying all members.

offset

Offset to the beginning of the data for this CPU.

size

Size of the data for this CPU.

align

The power of 2 alignment for the offset of the object file for the architecture specified in cputype within the binary. This is required to ensure that, if this binary is changed, the contents it retains are correctly aligned for virtual memory paging and other uses.

**Discussion**

An array of fat\_arch data structures appears directly after the [fat\_header](#page64) (page 64) data structure of a binary that contains object files for multiple architectures.

Regardless of the content this data structure describes, all its fields are stored in big-endian byte order.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

65

**OS X ABI Mach-O File Format Reference**

Data Types



**Availability**

Available in OS X v10.8 and later.

**Declared in**

fat.h



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

66

Document Revision History

This table describes the changes to *OS X ABI Mach-O File Format Reference* .



|  |  |
| --- | --- |
| **Date** | **Notes** |
| 2009-02-04 | Made minor changes. |
|  |  |
| 2007-04-26 | Added details about Mach-O files targeted for the OS X x86-64 |
|  | environment. |
|  | Added relocation details to [“Relocation Data Structures”](#page57) (page 57). |
|  | Updated the cputype field of the [mach\_header\_64](#page13) (page 13) structure. |
|  | Updated the r\_type bit of the [relocation\_info](#page58) (page 58) structure. |
|  | Added DWARF debugging-format information to the introduction. |
|  |  |
| 2006-10-03 | Added information about the uuid\_command load command. |
|  | Added [uuid\_command](#page19) (page 19) and updated [load\_command](#page16) (page 16). |
|  |  |
| 2006-09-05 | Added information about IA-32-specific structures and the file type for |
|  | dSYM files. |
|  |  |
| 2006-03-08 | Corrected the mach\_header\_64 description. |
|  | Replaced cross-references to “Indirect Addressing” throughout to |
|  | cross-references to “Position-Independent Code” in *Mach-O Programming* |
|  | *Topics* . |
|  | Removed CPU\_SUBTYPE\_I386\_ALL from the description for the |
|  | cpusubtype field of [mach\_header\_64](#page13) (page 13). |
|  |  |
| 2005-11-09 | Changed title from "Mach-O File Format Reference." |
|  | Added the phrase “Mac app binary interface (ABI) to the introduction to |
|  | raise this document’s visibility in searches. |
|  |  |



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

67

**Document Revision History**



|  |  |
| --- | --- |
| **Date** | **Notes** |
| 2005-08-11 | Clarified terminology for binaries that contain object files for more than |
|  | one architecture. |
|  |  |
| 2005-04-29 | Added information on 64-bit support in the Mach-O file format. Removed |
|  | the "Overview of the Runtime Architecture" and "Runtime Conventions |
|  | for PowerPC" chapters. That content was placed in "OS X Runtime |
|  | Overview" and "PowerPC Runtime Programming Guide," respectively. |
|  | Changed title to *Mach-O File Format Reference* . |
|  | Updated symbol declarations to match headers. |
|  |  |
| 2004-08-31 | Added information on parameter passing, section names, dynamic linking |
|  | of libraries, dead-code stripping flags, and GPR11. Removed dynamic |
|  | linking functions reference. Minor technical and editorial corrections |
|  | throughout. |

Added information on MH\_SUBSECTIONS\_VIA\_SYMBOLS flag to [mach\_header](#page9) (page 9) struct.

Added information on the S\_ATTR\_STRIP\_STATIC\_SYMS,

S\_ATTR\_LIVE\_SUPPORT, and S\_ATTR\_NO\_DEAD\_STRIP flags to [section](#page23) (page 23) struct.

Added explanation of PPC\_RELOC\_LO14\_SECTDIFF to [scattered\_relocation\_info](#page62) (page 62).

Added clarification on when callers put parameters in the stack, in addition to placing them in registers. See .

Added details on parameter passing for single-member structures. See 32-bit PowerPC Function Calling Conventions.

Refined description of GPR11. See 32-bit PowerPC Function Calling Conventions.

Specified correct sizes for composite parameters that are preceded by padding to make them 4 bytes in size. See 32-bit PowerPC Function Calling Conventions.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

68

**Document Revision History**



|  |  |
| --- | --- |
| **Date** | **Notes** |
|  | Added note to [“Introduction”](#page4) (page 4) indicating that compilers can |
|  | define additional section names that are not shown in [Table 1](#page8) (page 8). |
|  | Corrected example of a private external symbol. See *Mach-O Programming* |
|  | *Topics* . |
|  | Corrected ranges for unsigned int, unsigned long, and unsigned |
|  | long long, and vector unsigned int. See 32-bit PowerPC Function |
|  | Calling Conventions. |
|  | Corrected framework-building example. See *Mach-O Programming Topics* . |
|  | Removed “Mach-O Dynamic Linking Functions Reference” chapter and |
|  | placed its content in *Mach-O Runtime Reference* . |
|  |  |
| 2003-08-07 | Added description of new API for OS X version 10.3. |
|  |  |
| 2003-01-01 | Incorporated developer feedback. Updated code-generation examples. |
|  | Fixed bugs 2462895, 2749339, 2909989, 2910422, 2921574. |
|  |  |
| 2002-07-01 | More developer feedback. Document weak definitions and weak references |
|  | (new for 10.2). Substantially update the glossary. Other tweaks and |
|  | additional material. Clarify common vs. coalesced symbol definitions. |
|  | ABI: Rewrote position-independent and indirect code section, |
|  | incorporating correct examples and separating PIC and indirect code |
|  | generation. Add C99 \_Bool data type. See “32-bit PowerPC Function |
|  | Calling Conventions”. |
|  | Fixed bugs 2909989, 2910422, and 2921574. |
|  |  |
| 2002-05-01 | This was a preliminary draft distributed with the WWDC 2002 developer |
|  | tools. |

Incorporated many corrections from developer review. More to come.



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

69

**Document Revision History**



|  |  |
| --- | --- |
| **Date** | **Notes** |
|  | By popular demand, added some common usage scenarios to map runtime |
|  | features to the options in the standard OS X tools that implement those |
|  | features. To satisfy a related popular demand, this information is collected |
|  | in a separate chapter, which allows users of third-party tool sets to ignore |
|  | it. This chapter is currently unfinished, and the overview chapter is yet to |
|  | be modified to cross-reference it. |
|  | Updated umbrella framework description to better match reality. |
|  | Added long double and long long return value information. Removed |
|  | last vestiges of CFM. Rewrote data alignment section, incorporating the |
|  | correct rules (inherited from IBM’s xlc compiler) for power alignment |
|  | mode, and adding new natural alignment mode. |
|  |  |
| 2002-04-01 | This was a preliminary draft distributed with the April 2002 Developer |
|  | Tools CD. |
|  |  |



2009-02-04 | Copyright © 2003, 2009 Apple Inc. All Rights Reserved.

70



Apple Inc.

Copyright © 2003, 2009 Apple Inc.

All rights reserved.

No part of this publication may be reproduced,

stored in a retrieval system, or transmitted, in any

form or by any means, mechanical, electronic,

photocopying, recording, or otherwise, without

prior written permission of Apple Inc., with the

following exceptions: Any person is hereby

authorized to store documentation on a single

computer for personal use only and to print

copies of documentation for personal use

provided that the documentation contains

Apple’s copyright notice.

No licenses, express or implied, are granted with

respect to any of the technology described in this

document. Apple retains all intellectual property

rights associated with the technology described

in this document. This document is intended to

assist application developers to develop

applications only for Apple-labeled computers.

Apple Inc.

1 Infinite Loop

Cupertino, CA 95014

408-996-1010

Apple, the Apple logo, Mac, Macintosh,

Objective-C, OS X, and Xcode are trademarks of

Apple Inc., registered in the U.S. and other

countries.

Intel and Intel Core are registered trademarks of

Intel Corporation or its subsidiaries in the United

States and other countries.

PowerPC and the PowerPC logo are trademarks

of International Business Machines Corporation,

used under license therefrom.

UNIX is a registered trademark of The Open

Group.

**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.**